Notice of Copyright

Copyright © 2002-2008 the Open Watcom Contributors. Portions Copyright © 1984-2002 Sybase, Inc. and its subsidiaries. All rights reserved.

Any part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of anyone.

For more information please visit http://www.openwatcom.org/
Preface

This manual describes the Watcom C Library. It includes the Standard C Library (as defined in the ANSI C Standard) plus many additional library routines which make application development for personal computers much easier.

Acknowledgements

This book was produced with the Watcom GML electronic publishing system, a software tool developed by WATCOM. In this system, writers use an ASCII text editor to create source files containing text annotated with tags. These tags label the structural elements of the document, such as chapters, sections, paragraphs, and lists. The Watcom GML software, which runs on a variety of operating systems, interprets the tags to format the text into a form such as you see here. Writers can produce output for a variety of printers, including laser printers, using separately specified layout directives for such things as font selection, column width and height, number of columns, etc. The result is type-set quality copy containing integrated text and graphics.


Trademarks Used in this Manual

IBM is a registered trademark of International Business Machines Corp.

Intel is a registered trademark of Intel Corp.

Microsoft, MS, MS-DOS, Windows, Win32, Win32s, Windows NT and Windows 2000 are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

NetWare, NetWare 386, and Novell are registered trademarks of Novell, Inc.

UNIX is a registered trademark of The Open Group.

WATCOM is a trademark of Sybase, Inc. and its subsidiaries.
Table of Contents

1 C Library Overview .......................................................................................................................... 3
  1.1 Classes of Functions ................................................................................................................ 3
    1.1.1 Character Manipulation Functions ................................................................. 5
    1.1.2 Wide Character Manipulation Functions ................................................... 6
    1.1.3 Multibyte Character Manipulation Functions ........................................... 6
    1.1.4 Memory Manipulation Functions ................................................................. 8
    1.1.5 String Manipulation Functions ................................................................. 8
    1.1.6 Wide String Manipulation Functions ........................................................ 10
    1.1.7 Multibyte String Manipulation Functions ................................................ 11
    1.1.8 Conversion Functions .................................................................................. 13
    1.1.9 Memory Allocation Functions .................................................................... 14
    1.1.10 Heap Functions ......................................................................................... 15
    1.1.11 Math Functions .......................................................................................... 15
    1.1.12 Searching Functions .................................................................................. 17
    1.1.13 Time Functions .......................................................................................... 17
    1.1.14 Variable-length Argument Lists .............................................................. 17
    1.1.15 Stream I/O Functions ............................................................................... 18
    1.1.16 Wide Character Stream I/O Functions .................................................... 19
    1.1.17 Process Primitive Functions ................................................................... 20
    1.1.18 Process Environment .................................................................................... 22
    1.1.19 Directory Functions .................................................................................... 22
    1.1.20 Operating System I/O Functions ............................................................... 23
    1.1.21 File Manipulation Functions ...................................................................... 23
    1.1.22 Console I/O Functions ............................................................................... 24
    1.1.23 Default Windowing Functions ................................................................... 24
    1.1.24 BIOS Functions ............................................................................................. 24
    1.1.25 DOS-Specific Functions ............................................................................. 25
    1.1.26 Intel 80x86 Architecture-Specific Functions .......................................... 25
    1.1.27 Intel Pentium Multimedia Extension Functions ...................................... 26
    1.1.28 Miscellaneous Functions ............................................................................ 27
  1.2 Header Files ............................................................................................................................ 28
    1.2.1 Header Files in /watcom/h ........................................................................... 28
    1.2.2 Header Files in /watcom/h/sys ..................................................................... 31
  1.3 Global Data .............................................................................................................................. 32
  1.4 The TZ Environment Variable ............................................................................................... 36

2 Graphics Library .......................................................................................................................... 39
  2.1 Graphics Functions ................................................................................................................ 39
  2.2 Graphics Adapters .................................................................................................................. 39
  2.3 Classes of Graphics Functions .............................................................................................. 40
    2.3.1 Environment Functions .................................................................................. 40
    2.3.2 Coordinate System Functions ...................................................................... 41
    2.3.3 Attribute Functions ....................................................................................... 42
    2.3.4 Drawing Functions ......................................................................................... 42
    2.3.5 Text Functions ............................................................................................... 43
    2.3.6 Graphics Text Functions ............................................................................... 43
    2.3.7 Image Manipulation Functions ..................................................................... 44
    2.3.8 Font Manipulation Functions ....................................................................... 44
    2.3.9 Presentation Graphics Functions .................................................................. 45
      2.3.9.1 Display Functions .................................................................................. 45
# Table of Contents

2.3.9.2 Analyze Functions ................................................................. 46  
2.3.9.3 Utility Functions ................................................................. 46  
2.4 Graphics Header Files .............................................................. 46  

3 DOS Considerations .................................................................................. 47  
3.1 DOS Devices .................................................................................. 47  
3.2 DOS Directories ........................................................................... 47  
3.3 DOS File Names ........................................................................... 48  
3.4 DOS Files .................................................................................... 49  
3.5 DOS Commands ........................................................................... 50  
3.6 DOS Interrupts ............................................................................ 50  
3.7 DOS Processes ............................................................................ 50  

4 Library Functions and Macros ............................................................ 51  
abort .............................................................................................. 54  
abort_handler_s ............................................................................. 55  
abs ............................................................................................... 56  
access, _access, _waccess ............................................................ 57  
acos ............................................................................................... 59  
acosh .............................................................................................. 60  
alloca .............................................................................................. 61  
__arc, __arc_w, __arc_wxy ............................................................. 62  
asctime Functions ........................................................................... 64  
asctime_s, _wasctime_s ................................................................. 66  
atan ............................................................................................... 71  
atan2 .............................................................................................. 72  
atanh .............................................................................................. 74  
atexit .............................................................................................. 75  
atof, _wtof .................................................................................... 76  
atoi, _wtoi .................................................................................... 77  
atol, _wtol .................................................................................... 78  
atoll, _wtoll .................................................................................. 79  
atouni .............................................................................................. 80  
basename ........................................................................................ 81  
bname .............................................................................................. 82  
_bbeginthread, _bbeginthreadex .................................................... 83  
bessel Functions ............................................................................ 87  
bcmp .............................................................................................. 88  
bcopy .............................................................................................. 89  
__bfreebuf ....................................................................................... 90  
__bgetcmd ...................................................................................... 92  
__bheapseg .................................................................................... 93  
__bios_disk ................................................................................... 95  
__bios_equiplist ........................................................................... 97  
__ bios_keybrd ............................................................................. 98  
__bios_nemsize ............................................................................ 100  
__bios_printer .............................................................................. 101  
__bios_serialcom ........................................................................ 102  
__bios_timeofday ........................................................................ 104
# Table of Contents

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>_bprintf, _bwprintf</td>
<td>105</td>
</tr>
<tr>
<td>break... Functions</td>
<td>106</td>
</tr>
<tr>
<td>bsearch</td>
<td>107</td>
</tr>
<tr>
<td>bsearch_s</td>
<td>109</td>
</tr>
<tr>
<td>btowc</td>
<td>111</td>
</tr>
<tr>
<td>bzero</td>
<td>112</td>
</tr>
<tr>
<td>cabs</td>
<td>113</td>
</tr>
<tr>
<td>calloc Functions</td>
<td>114</td>
</tr>
<tr>
<td>ceil</td>
<td>116</td>
</tr>
<tr>
<td>egets</td>
<td>117</td>
</tr>
<tr>
<td>_chain_intr</td>
<td>118</td>
</tr>
<tr>
<td>chdir, _chdir, _wchdir</td>
<td>119</td>
</tr>
<tr>
<td>_chdrive</td>
<td>121</td>
</tr>
<tr>
<td>chmod, _chmod, _wchmod</td>
<td>122</td>
</tr>
<tr>
<td>chsize, _chsize</td>
<td>124</td>
</tr>
<tr>
<td>_clear87</td>
<td>125</td>
</tr>
<tr>
<td>clearenv</td>
<td>126</td>
</tr>
<tr>
<td>clearerr</td>
<td>127</td>
</tr>
<tr>
<td>_clearscreen</td>
<td>128</td>
</tr>
<tr>
<td>clock</td>
<td>129</td>
</tr>
<tr>
<td>close, _close</td>
<td>130</td>
</tr>
<tr>
<td>closedir, _wclosedir</td>
<td>131</td>
</tr>
<tr>
<td>_cmdname</td>
<td>133</td>
</tr>
<tr>
<td>_control87</td>
<td>134</td>
</tr>
<tr>
<td>_controlfp</td>
<td>136</td>
</tr>
<tr>
<td>cos</td>
<td>138</td>
</tr>
<tr>
<td>cosh</td>
<td>139</td>
</tr>
<tr>
<td>cprintf</td>
<td>140</td>
</tr>
<tr>
<td>cputs</td>
<td>141</td>
</tr>
<tr>
<td>creat, _creat, _wcreat</td>
<td>142</td>
</tr>
<tr>
<td>cscanf</td>
<td>144</td>
</tr>
<tr>
<td>ctime Functions</td>
<td>145</td>
</tr>
<tr>
<td>ctime_s, _wctime_s</td>
<td>147</td>
</tr>
<tr>
<td>cwait</td>
<td>149</td>
</tr>
<tr>
<td>delay</td>
<td>152</td>
</tr>
<tr>
<td>_dieetomsbin</td>
<td>153</td>
</tr>
<tr>
<td>diffmtime</td>
<td>154</td>
</tr>
<tr>
<td>dirname</td>
<td>155</td>
</tr>
<tr>
<td>_disable</td>
<td>156</td>
</tr>
<tr>
<td>_displaycursor</td>
<td>157</td>
</tr>
<tr>
<td>div</td>
<td>158</td>
</tr>
<tr>
<td>_dmsbintoeeee</td>
<td>159</td>
</tr>
<tr>
<td>_dos_allocmem</td>
<td>160</td>
</tr>
<tr>
<td>_dos_close</td>
<td>161</td>
</tr>
<tr>
<td>_dos_commit</td>
<td>162</td>
</tr>
<tr>
<td>_dos_creat</td>
<td>163</td>
</tr>
<tr>
<td>_dos_creatnew</td>
<td>164</td>
</tr>
<tr>
<td>dosexterr</td>
<td>166</td>
</tr>
<tr>
<td>_dos_find... Functions</td>
<td>168</td>
</tr>
<tr>
<td>_dos_freemem</td>
<td>171</td>
</tr>
<tr>
<td>_dos_getdate</td>
<td>172</td>
</tr>
<tr>
<td>_dos_getdiskfree</td>
<td>173</td>
</tr>
</tbody>
</table>
Table of Contents

_dos_getdrive .............................................................. 174
_dos_getfileattr ........................................................... 175
_dos_gettime .............................................................. 177
_dos_gettime .............................................................. 179
_dos_gettime .............................................................. 180
_dos_keep ................................................................. 181
_dos_open ................................................................. 182
_dos_read ................................................................. 184
_dos_setblock ............................................................ 185
_dos_setdrive ............................................................ 187
_dos_setdrive ............................................................ 189
_dos_setfileattr .......................................................... 190
_dos_settime ............................................................ 192
_dos_settime ............................................................ 194
_dos_setvect ............................................................. 196
_dos_setvect ............................................................. 197
_dos_write ............................................................... 197
dup, _dup ................................................................. 198
dup2, _dup2 ............................................................... 200
dwDeleteOnClose ....................................................... 202
dwSetAboutDlg .......................................................... 203
dwSetAppTitle ........................................................... 204
dwSetConTitle ........................................................... 205
dwShutDown .............................................................. 206
dwYield ................................................................. 207
ecvt, _ecvt, _wecvt .................................................... 208
_ellipse, _ellipse_w, _ellipse_wxy ................................ 210
_enable .................................................................. 212
_endthread, _endthreadex ........................................... 213
eof, _eof ................................................................. 215
exec... Functions ....................................................... 216
_exit, _Exit .............................................................. 220
exit ........................................................................ 221
exp ................................................................. 222
_expand Functions ..................................................... 223
fabs ...................................................................... 225
fcloseall ............................................................... 226
fcloseall ............................................................... 227
fcvt, _fcvt, _wfcvt .................................................... 228
fdopen, _fdopen, _wfopen .......................................... 230
fclearexcept ............................................................ 231
__fedisableexcept ...................................................... 232
__fenableexcept ......................................................... 233
fegetenv ............................................................... 234
fegetexceptflag ........................................................ 235
fegotround .............................................................. 236
feholdexcept ........................................................... 237
feof ................................................................. 238
feraiseexcept .......................................................... 239
ferror ................................................................. 240
fesetenv ............................................................... 241
fesetexceptflag ........................................................ 242
fesetround ............................................................. 243
# Table of Contents

<table>
<thead>
<tr>
<th>Function/Function Group</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>fetestexcept</td>
<td>244</td>
</tr>
<tr>
<td>feupdateenv</td>
<td>245</td>
</tr>
<tr>
<td>fflsl</td>
<td>246</td>
</tr>
<tr>
<td>ffs</td>
<td>247</td>
</tr>
<tr>
<td>fgetc, fgetwc</td>
<td>248</td>
</tr>
<tr>
<td>fgetchar, _fgetchar, _fgetwchar</td>
<td>249</td>
</tr>
<tr>
<td>fgetpos</td>
<td>250</td>
</tr>
<tr>
<td>fgets, fgetws</td>
<td>251</td>
</tr>
<tr>
<td>_fieetomsbin</td>
<td>252</td>
</tr>
<tr>
<td>filename, _filename, _filenamei64</td>
<td>253</td>
</tr>
<tr>
<td>FILENAME_MAX</td>
<td>254</td>
</tr>
<tr>
<td>fileno</td>
<td>255</td>
</tr>
<tr>
<td>_findclose</td>
<td>256</td>
</tr>
<tr>
<td>_findfirst, _findfirsti64, _wfindfirst, _wfindfirsti64</td>
<td>257</td>
</tr>
<tr>
<td>_findnext, _findnexti64, _wfindnext, _wfindnexti64</td>
<td>259</td>
</tr>
<tr>
<td>_finite</td>
<td>261</td>
</tr>
<tr>
<td>_floodfill, _floodfill_w</td>
<td>262</td>
</tr>
<tr>
<td>floor</td>
<td>263</td>
</tr>
<tr>
<td>flushall</td>
<td>264</td>
</tr>
<tr>
<td>fmod</td>
<td>265</td>
</tr>
<tr>
<td>_fmsbintoeieee</td>
<td>266</td>
</tr>
<tr>
<td>fnmatch</td>
<td>267</td>
</tr>
<tr>
<td>fopen, _wfopen</td>
<td>269</td>
</tr>
<tr>
<td>fopen_s, _wfopen_s</td>
<td>271</td>
</tr>
<tr>
<td>FP_OFF</td>
<td>274</td>
</tr>
<tr>
<td>FP_SEG</td>
<td>275</td>
</tr>
<tr>
<td>fpclassify</td>
<td>276</td>
</tr>
<tr>
<td>_fpreset</td>
<td>277</td>
</tr>
<tr>
<td>fprintf, fwprintf</td>
<td>278</td>
</tr>
<tr>
<td>fprintf_s, fwprintf_s</td>
<td>279</td>
</tr>
<tr>
<td>fputc, fputwc</td>
<td>281</td>
</tr>
<tr>
<td>fputchar, _fputchar, _fputwchar</td>
<td>282</td>
</tr>
<tr>
<td>fputs, fputws</td>
<td>283</td>
</tr>
<tr>
<td>fread</td>
<td>284</td>
</tr>
<tr>
<td>free Functions</td>
<td>285</td>
</tr>
<tr>
<td>_freect</td>
<td>287</td>
</tr>
<tr>
<td>freopen, _wfreopen</td>
<td>288</td>
</tr>
<tr>
<td>freopen_s, _wfreopen_s</td>
<td>289</td>
</tr>
<tr>
<td>frexp</td>
<td>291</td>
</tr>
<tr>
<td>fscanf, fwscanf</td>
<td>292</td>
</tr>
<tr>
<td>fscanf_s, _fwscanf_s</td>
<td>293</td>
</tr>
<tr>
<td>fseek</td>
<td>295</td>
</tr>
<tr>
<td>fsetpos</td>
<td>297</td>
</tr>
<tr>
<td>_fsopen, _wfsopen</td>
<td>298</td>
</tr>
<tr>
<td>fstat, _fstat, _fstati64, _wfstat, _wfstati64</td>
<td>301</td>
</tr>
<tr>
<td>fsync</td>
<td>305</td>
</tr>
<tr>
<td>ftell</td>
<td>307</td>
</tr>
<tr>
<td>ftimemicroseconds</td>
<td>308</td>
</tr>
<tr>
<td>_fullpath, _wfullpath</td>
<td>309</td>
</tr>
<tr>
<td>fwrite</td>
<td>311</td>
</tr>
<tr>
<td>fwrite</td>
<td>312</td>
</tr>
<tr>
<td>gcvt, _gcvt, _wgcvt</td>
<td>313</td>
</tr>
</tbody>
</table>
# Table of Contents

- `_getactivepage` ........................................................................................................ 314
- `_getarctinfo` ........................................................................................................ 315
- `_getbkcolor` .......................................................................................................... 317
- `getc, getw` ........................................................................................................... 318
- `getch` .................................................................................................................... 319
- `getchar, getwchar` .............................................................................................. 320
- `getche` .................................................................................................................. 321
- `_getcliprgn` ......................................................................................................... 322
- `getcmd` .................................................................................................................. 323
- `_getcolor` .............................................................................................................. 324
- `_getcurrentposition, _getcurrentposition_w` ...................................................... 325
- `getcwd, _wgetcwd` .............................................................................................. 326
- `getdcwd, _wgetdcwd` .......................................................................................... 328
- `getdiskfree` .......................................................................................................... 330
- `getdrive` ............................................................................................................... 331
- `getenv, _wgetenv` ............................................................................................... 332
- `getenv_s` ............................................................................................................... 333
- `_getfillmask` ....................................................................................................... 335
- `_getfontinfo` .................................................................................................... 336
- `getgtextextent` .................................................................................................... 337
- `getgtextvector` .................................................................................................... 338
- `getimage, _getimage_w, _getimage_wxy` ............................................................ 339
- `getlinestyle` ........................................................................................................ 341
- `_getmbcp` ............................................................................................................ 342
- `getopt` .................................................................................................................. 343
- `_get_osfhandle` .................................................................................................. 346
- `_getphyscoord` ................................................................................................... 348
- `getpid` .................................................................................................................. 349
- `getpixel, _getpixel_w` ....................................................................................... 350
- `getplotaction` ...................................................................................................... 351
- `gets, _gets` ......................................................................................................... 352
- `gets_s` .................................................................................................................. 353
- `getgtextcolor` ..................................................................................................... 354
- `getgtextcursor` .................................................................................................... 355
- `getgtextextent` .................................................................................................... 356
- `getgtextposition` ............................................................................................... 358
- `getgtextsettings` ............................................................................................... 359
- `getgtextwindow` ................................................................................................. 360
- `getvideoconfig` .................................................................................................. 361
- `getviewcoord, _getviewcoord_w, _getviewcoord_wxy` .................................... 364
- `getviewportpage` ............................................................................................... 365
- `getw` .................................................................................................................... 366
- `getwindowcoord` ............................................................................................... 367
- `gmtime Functions` ............................................................................................. 368
- `gmtime_s` ............................................................................................................ 370
- `_grow_handles` ................................................................................................... 372
- `grstatus` ............................................................................................................. 374
- `grtext, _grtext_w` ............................................................................................. 375
- `malloc` ................................................................................................................ 377
- `_harderr, _hardresume, _hardret` ...................................................................... 378
- `hdopen` .............................................................................................................. 381
- `heapchk Functions` .......................................................................................... 382
Table of Contents
# Table of Contents

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ismbcgraph</td>
<td>462</td>
</tr>
<tr>
<td>_ismbchira</td>
<td>464</td>
</tr>
<tr>
<td>_ismbckata</td>
<td>466</td>
</tr>
<tr>
<td>_ismbcl0</td>
<td>468</td>
</tr>
<tr>
<td>_ismbcl1</td>
<td>470</td>
</tr>
<tr>
<td>_ismbcl2</td>
<td>472</td>
</tr>
<tr>
<td>_ismbclelegal</td>
<td>474</td>
</tr>
<tr>
<td>_ismbcllower</td>
<td>476</td>
</tr>
<tr>
<td>_ismbcprint</td>
<td>478</td>
</tr>
<tr>
<td>_ismbcpunct</td>
<td>480</td>
</tr>
<tr>
<td>_ismbcspace</td>
<td>482</td>
</tr>
<tr>
<td>_ismbcsymbol</td>
<td>484</td>
</tr>
<tr>
<td>_ismbcupper</td>
<td>486</td>
</tr>
<tr>
<td>_ismbcxdigit</td>
<td>488</td>
</tr>
<tr>
<td>isnan</td>
<td>490</td>
</tr>
<tr>
<td>isnormal</td>
<td>491</td>
</tr>
<tr>
<td>isprint, iswprint</td>
<td>492</td>
</tr>
<tr>
<td>ispunct, iswpunct</td>
<td>493</td>
</tr>
<tr>
<td>isspace, iswspace</td>
<td>495</td>
</tr>
<tr>
<td>isupper, iswupper</td>
<td>497</td>
</tr>
<tr>
<td>iswctype</td>
<td>498</td>
</tr>
<tr>
<td>isxdigit, iswxdigit</td>
<td>500</td>
</tr>
<tr>
<td>itoa, _itoa, _itow</td>
<td>501</td>
</tr>
<tr>
<td>kbhit, _kbhit</td>
<td>503</td>
</tr>
<tr>
<td>labs</td>
<td>504</td>
</tr>
<tr>
<td>ldexp</td>
<td>505</td>
</tr>
<tr>
<td>ldiv</td>
<td>506</td>
</tr>
<tr>
<td>lfind</td>
<td>507</td>
</tr>
<tr>
<td>_lineto, _lineto_w</td>
<td>509</td>
</tr>
<tr>
<td>llabs</td>
<td>511</td>
</tr>
<tr>
<td>lldiv</td>
<td>512</td>
</tr>
<tr>
<td>localeconv</td>
<td>513</td>
</tr>
<tr>
<td>localtime Functions</td>
<td>516</td>
</tr>
<tr>
<td>localtime_s</td>
<td>518</td>
</tr>
<tr>
<td>lock</td>
<td>520</td>
</tr>
<tr>
<td>locking, _locking</td>
<td>521</td>
</tr>
<tr>
<td>log</td>
<td>523</td>
</tr>
<tr>
<td>log10</td>
<td>524</td>
</tr>
<tr>
<td>log2</td>
<td>525</td>
</tr>
<tr>
<td>longjmp</td>
<td>526</td>
</tr>
<tr>
<td>_lrotl</td>
<td>527</td>
</tr>
<tr>
<td>_lrotr</td>
<td>528</td>
</tr>
<tr>
<td>lsearch</td>
<td>529</td>
</tr>
<tr>
<td>lseek, _lseek, _lseekf64</td>
<td>531</td>
</tr>
<tr>
<td>ltoa, _ltoa, _ltow</td>
<td>534</td>
</tr>
<tr>
<td>ltoa, _ltow, _ltow</td>
<td>536</td>
</tr>
<tr>
<td>main, wmain, WinMain, wWinMain</td>
<td>538</td>
</tr>
<tr>
<td>_makepath, _wmakepath</td>
<td>542</td>
</tr>
<tr>
<td>malloc Functions</td>
<td>544</td>
</tr>
<tr>
<td>matherr</td>
<td>546</td>
</tr>
<tr>
<td>max</td>
<td>548</td>
</tr>
<tr>
<td>_mbstombc</td>
<td>549</td>
</tr>
</tbody>
</table>
# Table of Contents

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mbbtype</td>
<td>550</td>
</tr>
<tr>
<td>_mbccmp, _fmbccmp</td>
<td>553</td>
</tr>
<tr>
<td>_mbccpy, _fmbccpy</td>
<td>555</td>
</tr>
<tr>
<td>_mbicmp, _fmbicmp</td>
<td>556</td>
</tr>
<tr>
<td>_mbcistojis</td>
<td>558</td>
</tr>
<tr>
<td>_mbcjmstojis</td>
<td>559</td>
</tr>
<tr>
<td>_mbcrlen, _fmbcrlen</td>
<td>560</td>
</tr>
<tr>
<td>_mbctolower</td>
<td>562</td>
</tr>
<tr>
<td>_mbctoupper</td>
<td>564</td>
</tr>
<tr>
<td>_mbctohira</td>
<td>566</td>
</tr>
<tr>
<td>_mbctokata</td>
<td>568</td>
</tr>
<tr>
<td>_mbctombb</td>
<td>570</td>
</tr>
<tr>
<td>_mbgcolor, _fmbbgcolor</td>
<td>571</td>
</tr>
<tr>
<td>_mblen, _fmblen</td>
<td>572</td>
</tr>
<tr>
<td>_mb_putchar, _fmb_putchar</td>
<td>575</td>
</tr>
<tr>
<td>_mbrlen, _fmbbrlen</td>
<td>576</td>
</tr>
<tr>
<td>_mbrtocwc, _fmbrtocwc</td>
<td>579</td>
</tr>
<tr>
<td>_mbbsbtype, _fmbbsbtype</td>
<td>582</td>
</tr>
<tr>
<td>_mbbsnbcat, _fmbbsnbcat</td>
<td>585</td>
</tr>
<tr>
<td>_mbbsncmp, _fmbbsncmp</td>
<td>587</td>
</tr>
<tr>
<td>_mbbsnbcnt, _fmbbsbcnt, _strcnt, _wscnt</td>
<td>588</td>
</tr>
<tr>
<td>_mbbsncpy, _fnmbbsncpy</td>
<td>590</td>
</tr>
<tr>
<td>_mbbsnbcmp, _fmbbsnbcmp</td>
<td>592</td>
</tr>
<tr>
<td>_mbbsnbset, _fmbbsnbset</td>
<td>593</td>
</tr>
<tr>
<td>_mbbsncnt, _fmbbsncnt, _strcnt, _wscnt</td>
<td>594</td>
</tr>
<tr>
<td>_mbbsnexxc, _fmbbsnxxc, _strnxxc, _wcsnxxc</td>
<td>596</td>
</tr>
<tr>
<td>_mbsrtowcs, _fmbssrtowcs</td>
<td>598</td>
</tr>
<tr>
<td>_mbsrtowcs_s, _fmbssrtowcs_s</td>
<td>601</td>
</tr>
<tr>
<td>_mbsrtowcs, _fmbssrtowcs</td>
<td>604</td>
</tr>
<tr>
<td>_mbsrtowcs_s, _fmbssrtowcs_s</td>
<td>605</td>
</tr>
<tr>
<td>_mbrterm, _fmbterm</td>
<td>607</td>
</tr>
<tr>
<td>_mbrtowc, _fmbrtowc</td>
<td>609</td>
</tr>
<tr>
<td>_mbrtop, _fmbrtop</td>
<td>611</td>
</tr>
<tr>
<td>_memavl</td>
<td>613</td>
</tr>
<tr>
<td>_memccpy, _fmemccpy</td>
<td>614</td>
</tr>
<tr>
<td>_memchr, _fmemchr, wmemechr</td>
<td>615</td>
</tr>
<tr>
<td>_memcmp, _fmemcmp, _wmemcmp</td>
<td>616</td>
</tr>
<tr>
<td>_memccpy, _fmemccpy, _wmemccpy</td>
<td>617</td>
</tr>
<tr>
<td>_memcmp_s, _wmemccpy_s</td>
<td>618</td>
</tr>
<tr>
<td>_memcmp, _fmemcmp, _wmemcmp</td>
<td>619</td>
</tr>
<tr>
<td>_memmax</td>
<td>620</td>
</tr>
<tr>
<td>_memmove, _fmemmove, _wmemmove</td>
<td>621</td>
</tr>
<tr>
<td>_memmove_s, _wmemmove_s</td>
<td>622</td>
</tr>
<tr>
<td>_m_empty</td>
<td>624</td>
</tr>
<tr>
<td>_memset, _fmemset, _wmemset</td>
<td>626</td>
</tr>
<tr>
<td>_m_from_int</td>
<td>627</td>
</tr>
<tr>
<td>_min</td>
<td>628</td>
</tr>
<tr>
<td>_mkdirc, _nmkdir, _wnmkdir</td>
<td>629</td>
</tr>
<tr>
<td>MK_FP</td>
<td>630</td>
</tr>
<tr>
<td>_mkstemp</td>
<td>631</td>
</tr>
<tr>
<td>_mktemp, _wmktemp</td>
<td>633</td>
</tr>
<tr>
<td>_mktime</td>
<td>635</td>
</tr>
</tbody>
</table>
Table of Contents

modf ........................................................................................................... 637
movedata ..................................................................................................... 638
__moveto, __moveto_w .............................................................. 639
__m_packssdw .............................................................. 640
__m_packsswb .............................................................. 642
__m_packuswb .............................................................. 644
__m_padd .............................................................. 646
__m_padd .............................................................. 647
__m_padd .............................................................. 648
__m_padd .............................................................. 649
__m_padd .............................................................. 650
__m_padd .............................................................. 651
__m_padd .............................................................. 652
__m_padd .............................................................. 653
__m_padd .............................................................. 654
__mncmpceqb .............................................................. 655
__mncmpceqd .............................................................. 656
__mncmpceqw .............................................................. 657
__mncmpgtb .............................................................. 658
__mncmpgtd .............................................................. 659
__mncmpgtw .............................................................. 660
__mpmaddwd .............................................................. 661
__mpmulhw .............................................................. 662
__mpmullw .............................................................. 663
__m_por .............................................................. 664
__m_pslld .............................................................. 665
__m_pslldi .............................................................. 666
__m_pslsq .............................................................. 667
__m_pslsqi .............................................................. 668
__m_pslsw .............................................................. 669
__m_pslswi .............................................................. 670
__m_psrad .............................................................. 671
__m_psradi .............................................................. 672
__m_psrw .............................................................. 673
__m_psrwi .............................................................. 674
__m_psrld .............................................................. 675
__m_psrldi .............................................................. 676
__m_psrli .............................................................. 677
__m_psrliq .............................................................. 678
__m_psrliq .............................................................. 679
__m_psrli .............................................................. 680
__m_psrw .............................................................. 681
__m_psrwi .............................................................. 682
__m_psub .............................................................. 683
__m_psub .............................................................. 684
__m_psub .............................................................. 685
__m_psub .............................................................. 686
__m_psub .............................................................. 687
__m_punpckhbw .............................................................. 688
__m_punpckhdq .............................................................. 690
__m_punpckhwd .............................................................. 691
__m_punpcklbw .............................................................. 692
## Table of Contents

- _m_punpckldq ................................................................. 694
- _m_punpcklwd ................................................................. 695
- _m_pxor ............................................................................. 696
- _msize Functions ............................................................ 697
- _m_to_int .......................................................................... 698
- nosound ........................................................................... 699
- offsetof ........................................................................... 700
- onexit ............................................................................... 701
- open, _open, _wopen ....................................................... 702
- opendir, _wopendir .......................................................... 705
- _open_osfhandle ............................................................... 708
- _os_handle ......................................................................... 711
- _outgtext .......................................................................... 712
- _outmem ........................................................................... 714
- outp .................................................................................. 715
- outpd ................................................................................ 716
- outpw ............................................................................... 717
- _outtext ........................................................................... 718
- pclose ............................................................................... 719
- perror, _wperror ............................................................ 720
- _pg_analyzechart, _pg_analyzecartms ................................. 721
- _pg_analyzepie ................................................................. 723
- _pg_analyzescatter, _pg_analyzescatterms ......................... 725
- _pg_chart, _pg_chartms .................................................... 727
- _pg_chartpie ...................................................................... 730
- _pg_chartscatter, _pg_chartscatterms ................................. 733
- _pg_defaultchart ............................................................... 736
- _pg_getchardef ................................................................. 738
- _pg_getpalette .................................................................. 739
- _pg_getstyleset ............................................................... 741
- _pg_hlabelchart ............................................................... 743
- _pg_inichart ...................................................................... 744
- _pg_resetpalette ............................................................. 746
- _pg_resetstyleset ............................................................. 748
- _pg_setchardef ................................................................. 750
- _pg_setpalette ................................................................. 751
- _pg_setstyleset ............................................................... 753
- _pg_vlabelchart ............................................................... 755
- _pie, _pie_w, _pie_wxy .................................................... 756
- _pipe ............................................................................... 759
- _polygon, _polygon_w, _polygon_wxy .............................. 762
- _popen, _wpopen ............................................................ 764
- pow .................................................................................. 766
- printf, wprintf ............................................................... 767
- printf_s, wprintf_s ........................................................... 773
- pute, putw .......................................................................... 775
- putch ............................................................................... 776
- putwchar, putwchar ........................................................ 777
- putenv, _putenv, _wputenv ............................................... 778
- _putimage, _putimage_w .................................................. 780
- puts, _putws ...................................................................... 782
- _putw ............................................................................... 783
# Table of Contents

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>qsort</td>
<td>784</td>
</tr>
<tr>
<td>qsort_s</td>
<td>785</td>
</tr>
<tr>
<td>raise</td>
<td>787</td>
</tr>
<tr>
<td>rand</td>
<td>789</td>
</tr>
<tr>
<td>read, _read</td>
<td>790</td>
</tr>
<tr>
<td>readdir, _wreaddir</td>
<td>792</td>
</tr>
<tr>
<td>realloc Functions</td>
<td>795</td>
</tr>
<tr>
<td>_rectangle, _rectangle_w, _rectangle_wxy</td>
<td>797</td>
</tr>
<tr>
<td>_registerfonts</td>
<td>799</td>
</tr>
<tr>
<td>_remapallpalette</td>
<td>800</td>
</tr>
<tr>
<td>_remappalette</td>
<td>801</td>
</tr>
<tr>
<td>remove, _wremove</td>
<td>802</td>
</tr>
<tr>
<td>rename, _rename</td>
<td>803</td>
</tr>
<tr>
<td>rewind</td>
<td>804</td>
</tr>
<tr>
<td>rewinddir, _wrewinddir</td>
<td>805</td>
</tr>
<tr>
<td>rmdir, _rmdir, _wrmdir</td>
<td>807</td>
</tr>
<tr>
<td>_rotl</td>
<td>808</td>
</tr>
<tr>
<td>_rotr</td>
<td>809</td>
</tr>
<tr>
<td>sbrk</td>
<td>810</td>
</tr>
<tr>
<td>scanf, wscanf</td>
<td>812</td>
</tr>
<tr>
<td>scanf_s, wscanf_s</td>
<td>818</td>
</tr>
<tr>
<td>_scrolltextwindow</td>
<td>819</td>
</tr>
<tr>
<td>_searchenv, _wsearchenv</td>
<td>820</td>
</tr>
<tr>
<td>segread</td>
<td>821</td>
</tr>
<tr>
<td>_selectpalette</td>
<td>822</td>
</tr>
<tr>
<td>set_constraint_handler_s</td>
<td>823</td>
</tr>
<tr>
<td>_setactivepage</td>
<td>825</td>
</tr>
<tr>
<td>_setbkcolor</td>
<td>826</td>
</tr>
<tr>
<td>setbuf</td>
<td>827</td>
</tr>
<tr>
<td>_setcharsize, _setcharsize_w</td>
<td>828</td>
</tr>
<tr>
<td>_setcharspacing, _setcharspacing_w</td>
<td>830</td>
</tr>
<tr>
<td>_setcliprgn</td>
<td>832</td>
</tr>
<tr>
<td>_setcolor</td>
<td>833</td>
</tr>
<tr>
<td>setenv, _setenv, _wsetenv</td>
<td>834</td>
</tr>
<tr>
<td>_setfillmask</td>
<td>836</td>
</tr>
<tr>
<td>_setfont</td>
<td>838</td>
</tr>
<tr>
<td>_setgettextvector</td>
<td>840</td>
</tr>
<tr>
<td>setjmp</td>
<td>841</td>
</tr>
<tr>
<td>_setlinestyle</td>
<td>842</td>
</tr>
<tr>
<td>setlocale, _wsetlocale</td>
<td>844</td>
</tr>
<tr>
<td>_set_matherr</td>
<td>846</td>
</tr>
<tr>
<td>_setmbeq</td>
<td>848</td>
</tr>
<tr>
<td>setmode, _setmode</td>
<td>849</td>
</tr>
<tr>
<td>set_new_handler, _set_new_handler</td>
<td>850</td>
</tr>
<tr>
<td>_setpixel, _setpixel_w</td>
<td>852</td>
</tr>
<tr>
<td>_setplotaction</td>
<td>853</td>
</tr>
<tr>
<td>_settextalign</td>
<td>854</td>
</tr>
<tr>
<td>_settextcolor</td>
<td>856</td>
</tr>
<tr>
<td>_settextcursor</td>
<td>857</td>
</tr>
<tr>
<td>_settextorient</td>
<td>858</td>
</tr>
<tr>
<td>_settextpath</td>
<td>859</td>
</tr>
<tr>
<td>_settextposition</td>
<td>861</td>
</tr>
</tbody>
</table>

xvi
## Table of Contents

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>_settextrows</td>
<td>862</td>
</tr>
<tr>
<td>_settextwindow</td>
<td>863</td>
</tr>
<tr>
<td>setvbuf</td>
<td>864</td>
</tr>
<tr>
<td>_setvideomode</td>
<td>865</td>
</tr>
<tr>
<td>_setvideomoderows</td>
<td>868</td>
</tr>
<tr>
<td>_setvieworg</td>
<td>869</td>
</tr>
<tr>
<td>_setviewport</td>
<td>870</td>
</tr>
<tr>
<td>_setvisualpage</td>
<td>871</td>
</tr>
<tr>
<td>_setwindow</td>
<td>872</td>
</tr>
<tr>
<td>signal</td>
<td>874</td>
</tr>
<tr>
<td>signbit</td>
<td>877</td>
</tr>
<tr>
<td>sin</td>
<td>878</td>
</tr>
<tr>
<td>sinh</td>
<td>879</td>
</tr>
<tr>
<td>mbsinit, sisinit</td>
<td>880</td>
</tr>
<tr>
<td>sleep</td>
<td>883</td>
</tr>
<tr>
<td>_snprintf, _snwprintf</td>
<td>884</td>
</tr>
<tr>
<td>snprintf, swprintf</td>
<td>886</td>
</tr>
<tr>
<td>snprintf_s, snwprintf_s</td>
<td>888</td>
</tr>
<tr>
<td>sopen, _sopen, _wsopen</td>
<td>890</td>
</tr>
<tr>
<td>sound</td>
<td>894</td>
</tr>
<tr>
<td>spawn.. Functions</td>
<td>896</td>
</tr>
<tr>
<td>_splitpath, _wsplitpath</td>
<td>902</td>
</tr>
<tr>
<td>_splitpath2, _wsplitpath2</td>
<td>904</td>
</tr>
<tr>
<td>sprintf, swprintf</td>
<td>906</td>
</tr>
<tr>
<td>sprintf_s, swprintf_s</td>
<td>908</td>
</tr>
<tr>
<td>sqrt</td>
<td>910</td>
</tr>
<tr>
<td>srand</td>
<td>911</td>
</tr>
<tr>
<td>scanf, swscanf</td>
<td>912</td>
</tr>
<tr>
<td>scanf_s, swscanf_s</td>
<td>913</td>
</tr>
<tr>
<td>stackavail, _stackavail</td>
<td>915</td>
</tr>
<tr>
<td>stat, _stat, _stat64, _wstat, _wstat64, lstat</td>
<td>916</td>
</tr>
<tr>
<td>_status87</td>
<td>919</td>
</tr>
<tr>
<td>strcasecmp</td>
<td>920</td>
</tr>
<tr>
<td>strcascmp</td>
<td>921</td>
</tr>
<tr>
<td>strcat, _fstrcat, wcscat, _mbscat, _fmbscat</td>
<td>922</td>
</tr>
<tr>
<td>strcat_s, wcscat_s</td>
<td>923</td>
</tr>
<tr>
<td>strchr, _fstrchr, wcschr, _mbstrchr, _fmbstrchr</td>
<td>925</td>
</tr>
<tr>
<td>strcmp, _fstrcmp, wcscmp, _mbscmp, _fmbscmp</td>
<td>926</td>
</tr>
<tr>
<td>strcmpl, wcscmpeq</td>
<td>928</td>
</tr>
<tr>
<td>strcoll, wcscoll, _mbscoll</td>
<td>929</td>
</tr>
<tr>
<td>strcpyp, _fstrcpyp, wcscpy, _mbscpyp, _fmbscpyp</td>
<td>930</td>
</tr>
<tr>
<td>strcpy_s, wscpy_s</td>
<td>932</td>
</tr>
<tr>
<td>strcsn, _fstrcsn, wcscsn, _mbcsn, _fmbcsn</td>
<td>934</td>
</tr>
<tr>
<td>_strdate, _wstrdate</td>
<td>936</td>
</tr>
<tr>
<td>_strdec, _wcsdec, _mbdec, _fmbdec</td>
<td>937</td>
</tr>
<tr>
<td>strdup, _strdup, _fstrdup, wcscdup, _mbsdup, _fmbdup</td>
<td>939</td>
</tr>
<tr>
<td>strerror, wcerror</td>
<td>940</td>
</tr>
<tr>
<td>strerror_s, wcerror_s</td>
<td>941</td>
</tr>
<tr>
<td>strerroren_s, wcerror_s_</td>
<td>943</td>
</tr>
<tr>
<td>strftime, wcsftime, _wstrftime_ms</td>
<td>944</td>
</tr>
<tr>
<td>stricmp, _stricmp, wcscmp, _mbscmp, _fmbcmp</td>
<td>948</td>
</tr>
<tr>
<td>_stricoll, _wcsicoll, _mbicoll</td>
<td>950</td>
</tr>
<tr>
<td>_stricn, _wcsinc, _mbinc, _fmbinc</td>
<td>951</td>
</tr>
</tbody>
</table>
# Table of Contents

strlcat, wcsncat ................................................................. 954
strlcpy, wcsncpy ............................................................. 955
strlen, _fstrlen, wcslen, _mbslen, _fmbsslen ...................... 956
strlwr, _strlwr, _strlwr, _wslwr, _mbslwr, _fmbsslwr ........... 958
strncasecmp ................................................................. 960
strncat, _fstrncat, wcsncat, _mbsncat, _fmbssncat ............... 962
strncat_s, wcsncat_s ....................................................... 964
strncpy_s, wcsncpy_s ..................................................... 966
strncpy, _fstrncpy, wcsncpy, _mbsncpy, _fmbncpy ................. 968
strncpy_s, wcsncpy_s ..................................................... 971
strnicmp, _strnicmp, _fstrnicmp, _wcsnicmp, _mbsnicmp, _fmbssnicmp ............... 973
strnicoll, _strnicoll, wcsnicoll, _mbsncoll ................................ 975
strnicoll, _wcsnicoll, _mbsncoll ...................................... 977
strnlen_s, wcsnlen_s ...................................................... 978
strncpy_s, wcsncpy_s ..................................................... 981
strncpy, _fstrncpy, wcsncpy, _mbsncpy, _fmbncpy ................. 984
strncoll, _strncoll, _wcsncoll, _fmbncoll ......................... 986
strncoll, _wcsncoll, _mbsncoll ....................................... 988
strncoll, _strncoll, _fstrncoll, _mbssncoll, _fmbssncoll ........... 990
strncoll, _strncoll, _wcsncoll, _fmbssncoll ....................... 992
strncoll, _wcsncoll, _mbssncoll ..................................... 994
strncoll, _strncoll, _fstrncoll, _mbsncoll, _fmbssncoll .......... 996
strncoll, _strncoll, _mbsncoll ....................................... 998
strncoll, _strncoll, _fstrncoll, _mbsncoll, _fmbssncoll .......... 1000
strncoll, _strncoll, _fstrncoll, _mbssncoll, _fmbssncoll ......... 1002
strncoll, _strncoll, _fstrncoll, _mbsncoll, _fmbssncoll .......... 1004
strncoll, _strncoll, _mbsncoll ....................................... 1006
strncoll, _strncoll, _mbssncoll ..................................... 1008
strncoll, _strncoll, _mbsncoll ....................................... 1010
strncoll, _strncoll, _fstrncoll, _mbssncoll, _fmbssncoll ......... 1012
strncoll, _strncoll, _fstrncoll, _mbsncoll, _fmbssncoll .......... 1014
strncoll, _strncoll, _mbsncoll ....................................... 1016
strncoll, _strncoll, _mbssncoll ..................................... 1018
strncoll, _strncoll, _mbsncoll ....................................... 1020
strncoll, _strncoll, _fstrncoll, _mbssncoll, _fmbssncoll ......... 1022
strncoll, _strncoll, _fstrncoll, _mbsncoll, _fmbssncoll .......... 1024
strncoll, _strncoll, _mbsncoll ....................................... 1026
strncoll, _strncoll, _mbssncoll ..................................... 1028
strncoll, _strncoll, _mbsncoll ....................................... 1030
strncoll, _strncoll, _fstrncoll, _mbssncoll, _fmbssncoll ......... 1032
strncoll, _strncoll, _fstrncoll, _mbsncoll, _fmbssncoll .......... 1034

swab .................................................................................. 1035
Table of Contents

umask, _umask ......................................................................................................................... 1036
ungetc, ungetwc ....................................................................................................................... 1038
ungetch ..................................................................................................................................... 1039
unlink, _unlink, _wunlink .................................................................................................... 1040
unlock ....................................................................................................................................... 1041
_unregisterfonts ...................................................................................................................... 1042
utime, _utime, _wutime ........................................................................................................... 1043
utoa, _utoa, _utow .................................................................................................................. 1045
va_arg ....................................................................................................................................... 1046
va_end ....................................................................................................................................... 1047
tva_start ................................................................................................................................... 1049
_vbprintf, _vbwprintf ............................................................................................................ 1050
vprintf, __vprintf ..................................................................................................................... 1051
_vcscanf .................................................................................................................................... 1052
_vcprintf ................................................................................................................................... 1053
_vfprintf, _vfprintf .................................................................................................................. 1054
_vfprintf_s, _vfprintf_s ............................................................................................................ 1055
_vfscanf, _vfscanf .................................................................................................................... 1056
_vfscanf_s, _vfscanf_s ............................................................................................................... 1057
_vfprintf, _vfprintf .................................................................................................................... 1058
_vfprintf_s, _vfprintf_s ............................................................................................................ 1059
_vscanf, _vscanf ....................................................................................................................... 1060
_vscanf_s, _vscanf_s ................................................................................................................... 1061
_vsnprintf, _vsnprintf .............................................................................................................. 1062
_vsnprintf_s, _vsnprintf_s ....................................................................................................... 1063
_vsnprintf_s, _vsnprintf_s ....................................................................................................... 1064
_vscanf, _vscanf ....................................................................................................................... 1065
_vsnprintf, _vsnprintf .............................................................................................................. 1066
_vsnprintf, _vsnprintf .............................................................................................................. 1067
_vsnprintf, _vsnprintf .............................................................................................................. 1068
_vsnprintf, _vsnprintf .............................................................................................................. 1069
_vsnprintf, _vsnprintf .............................................................................................................. 1070
_vsnprintf, _vsnprintf .............................................................................................................. 1071
_vsnprintf, _vsnprintf .............................................................................................................. 1072
_vsnprintf, _vsnprintf .............................................................................................................. 1073
_vsnprintf, _vsnprintf .............................................................................................................. 1074
_vsnprintf, _vsnprintf .............................................................................................................. 1075
_vsnprintf, _vsnprintf .............................................................................................................. 1076
_vsnprintf, _vsnprintf .............................................................................................................. 1077
_vsnprintf, _vsnprintf .............................................................................................................. 1078
_vsnprintf, _vsnprintf .............................................................................................................. 1079
_vsnprintf, _vsnprintf .............................................................................................................. 1080
_vsnprintf, _vsnprintf .............................................................................................................. 1081
_vsnprintf, _vsnprintf .............................................................................................................. 1082
_wait ......................................................................................................................................... 1083
_wctomb, _fwctomb .................................................................................................................. 1084
_wctomb_s, _fwctomb_s ............................................................................................................. 1085
_wcstombs, _fwctombs ............................................................................................................... 1086
_wcstombs_s, _fwctombs_s ....................................................................................................... 1087
_wcstombs_s, _fwctombs_s ....................................................................................................... 1088
_wcsrtomb, _fcwsrtomb ............................................................................................................ 1089
_wcsrtomb_s, _fcwsrtomb_s ...................................................................................................... 1090
_wcsrtomb_s, _fcwsrtomb_s ...................................................................................................... 1091
_wcsrtomb_s, _fcwsrtomb_s ...................................................................................................... 1092
_wcsrtomb_s, _fcwsrtomb_s ...................................................................................................... 1093
_write ........................................................................................................................................ 1094
_write ........................................................................................................................................ 1095
_write ........................................................................................................................................ 1096
_write ........................................................................................................................................ 1097
_write ........................................................................................................................................ 1098
_write ........................................................................................................................................ 1099
_write ........................................................................................................................................ 1100
_write ........................................................................................................................................ 1101
_write ........................................................................................................................................ 1102
_write ........................................................................................................................................ 1103
_write ........................................................................................................................................ 1104
_write ........................................................................................................................................ 1105
_write ........................................................................................................................................ 1106
_write ........................................................................................................................................ 1107
_write ........................................................................................................................................ 1108
_write ........................................................................................................................................ 1109
_write ........................................................................................................................................ 1110
_write ........................................................................................................................................ 1111
_write ........................................................................................................................................ 1112
_write ........................................................................................................................................ 1113
_write ........................................................................................................................................ 1114
_write ........................................................................................................................................ 1115

5 Re-entrant Functions .................................................................................................................. 1117

Appendices ...................................................................................................................................... 1119

A. Implementation-Defined Behavior of the C Library ................................................................. 1121
   A.1 NULL Macro .......................................................................................................................... 1121
   A.2 Diagnostic Printed by the assert Function .......................................................................... 1121

xix
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.3 Character Testing</td>
<td>1121</td>
</tr>
<tr>
<td>A.4 Domain Errors</td>
<td>1122</td>
</tr>
<tr>
<td>A.5 Underflow of Floating-Point Values</td>
<td>1122</td>
</tr>
<tr>
<td>A.6 The fmod Function</td>
<td>1122</td>
</tr>
<tr>
<td>A.7 The signal Function</td>
<td>1122</td>
</tr>
<tr>
<td>A.8 Default Signals</td>
<td>1123</td>
</tr>
<tr>
<td>A.9 The SIGILL Signal</td>
<td>1123</td>
</tr>
<tr>
<td>A.10 Terminating Newline Characters</td>
<td>1123</td>
</tr>
<tr>
<td>A.11 Space Characters</td>
<td>1123</td>
</tr>
<tr>
<td>A.12 Null Characters</td>
<td>1124</td>
</tr>
<tr>
<td>A.13 File Position in Append Mode</td>
<td>1124</td>
</tr>
<tr>
<td>A.14 Truncation of Text Files</td>
<td>1124</td>
</tr>
<tr>
<td>A.15 File Buffering</td>
<td>1124</td>
</tr>
<tr>
<td>A.16 Zero-Length Files</td>
<td>1124</td>
</tr>
<tr>
<td>A.17 File Names</td>
<td>1124</td>
</tr>
<tr>
<td>A.18 File Access Limits</td>
<td>1125</td>
</tr>
<tr>
<td>A.19 Deleting Open Files</td>
<td>1125</td>
</tr>
<tr>
<td>A.20 Renaming with a Name that Exists</td>
<td>1125</td>
</tr>
<tr>
<td>A.21 Printing Pointer Values</td>
<td>1125</td>
</tr>
<tr>
<td>A.22 Reading Pointer Values</td>
<td>1125</td>
</tr>
<tr>
<td>A.23 Reading Ranges</td>
<td>1126</td>
</tr>
<tr>
<td>A.24 File Position Errors</td>
<td>1126</td>
</tr>
<tr>
<td>A.25 Messages Generated by the perror Function</td>
<td>1126</td>
</tr>
<tr>
<td>A.26 Allocating Zero Memory</td>
<td>1126</td>
</tr>
<tr>
<td>A.27 The abort Function</td>
<td>1127</td>
</tr>
<tr>
<td>A.28 The atexit Function</td>
<td>1127</td>
</tr>
<tr>
<td>A.29 Environment Names</td>
<td>1127</td>
</tr>
<tr>
<td>A.30 The system Function</td>
<td>1127</td>
</tr>
<tr>
<td>A.31 The strerror Function</td>
<td>1127</td>
</tr>
<tr>
<td>A.32 The Time Zone</td>
<td>1128</td>
</tr>
<tr>
<td>A.33 The clock Function</td>
<td>1128</td>
</tr>
</tbody>
</table>
Watcom C Library Reference
1 C Library Overview

The C library provides much of the power usually associated with the C language. This chapter introduces the individual functions (and macros) that comprise the Watcom C library. The chapter Library Functions and Macros describes each function and macro in complete detail.

Library functions are called as if they had been defined within the program. When the program is linked, the code for these routines is incorporated into the program by the linker.

Strictly speaking, it is not necessary to declare most library functions since they return `int` values for the most part. It is preferred, however, to declare all functions by including the header files found in the synopsis section with each function. Not only does this declare the return value, but also the type expected for each of the arguments as well as the number of arguments. This enables the Watcom C and C++ compilers to check the arguments coded with each function call.

1.1 Classes of Functions

The functions in the Watcom C library can be organized into a number of classes:

**Character Manipulation Functions**
These functions deal with single characters.

**Wide Character Manipulation Functions**
These functions deal with wide characters.

**Multibyte Character Manipulation Functions**
These functions deal with multibyte characters.

**Memory Manipulation Functions**
These functions manipulate blocks of memory.

**String Manipulation Functions**
These functions manipulate strings of characters. A character string is an array of zero or more adjacent characters followed by a null character (`\0`) which marks the end of the string.

**Wide String Manipulation Functions**
These functions manipulate strings of wide characters. A wide character string is an array of zero or more adjacent wide characters followed by a null wide character (L`\0`) which marks the end of the wide string.

**Multibyte String Manipulation Functions**
These functions manipulate strings of multibyte characters. A multibyte character is either a single-byte or double-byte character. The Chinese, Japanese and Korean character sets are examples of character sets containing both single-byte and double-byte characters.
What determines whether a character is a single-byte or double-byte character is the value of the lead byte in the sequence. For example, in the Japanese DBCS (double-byte character set), double-byte characters are those in which the first byte falls in the range 0x81 - 0x9F or 0xE0 - 0xFC and the second byte falls in the range 0x40 - 0x7E or 0x80 - 0xFC. A string of multibyte characters must be scanned from the first byte (index 0) to the last byte (index n) in sequence in order to determine if a particular byte is part of a double-byte character. For example, suppose that a multibyte character string contains the following byte values:

0x31 0x40 0x41 0x81 0x41 // "1@A.." where .. is a DB char

Among other characters, it contains the letter "A" (the first 0x41) and a double-byte character (0x81 0x41). The second 0x41 is not the letter "A" and that could only be determined by scanning from left to right starting with the first byte (0x31).

Conversion Functions
These functions convert values from one representation to another. Numeric values, for example, can be converted to strings.

Memory Allocation Functions
These functions are concerned with allocating and deallocating memory.

Heap Functions
These functions provide the ability to shrink and grow the heap, as well as, find heap related problems.

Math Functions
The mathematical functions perform mathematical computations such as the common trigonometric calculations. These functions operate on double values, also known as floating-point values.

Searching Functions
These functions provide searching and sorting capabilities.

Time Functions
These functions provide facilities to obtain and manipulate times and dates.

Variable-length Argument Lists
These functions provide the capability to process a variable number of arguments to a function.

Stream I/O Functions
These functions provide the "standard" functions to read and write files. Data can be transmitted as characters, strings, blocks of memory or under format control.

Wide Character Stream I/O Functions
These functions provide the "standard" functions to read and write files of wide characters. Data can be transmitted as wide characters, wide character strings, blocks of memory or under format control.

Process Primitive Functions
These functions deal with process creation, execution and termination, signal handling, and timer operations.
Process Environment
These functions deal with process identification, user identification, process groups, system identification, system time and process time, environment variables, terminal identification, and configurable system variables.

Directory Functions
These functions provide directory services.

Operating System I/O Functions
These "non-standard" file operations are more primitive than the "standard" functions in that they are directly interfaced to the operating system. They are included to provide compatibility with other C implementations and to provide the capability to directly use operating-system file operations.

File Manipulation Functions
These functions operate directly on files, providing facilities such as deletion of files.

Console I/O Functions
These functions provide the capability to directly read and write characters from the console.

Default Windowing Functions
These functions provide the capability to manipulate various dialog boxes in Watcom’s default windowing system.

BIOS Functions
This set of functions allows access to services provided by the BIOS.

DOS-Specific Functions
This set of functions allows access to DOS-specific functions.

Intel 80x86 Architecture-Specific Functions
This set of functions allows access to Intel 80x86 processor-related functions.

Intel Pentium Multimedia Extension Functions
This set of functions allows access to Intel Architecture Multimedia Extensions (MMX).

Miscellaneous Functions
This collection consists of the remaining functions.

The following subsections describe these function classes in more detail. Each function in the class is noted with a brief description of its purpose. The chapter Library Functions and Macros provides a complete description of each function and macro.

1.1.1 Character Manipulation Functions
These functions operate upon single characters of type char. The functions test characters in various ways and convert them between upper and lowercase. The following functions are defined:

- isalnum: test for letter or digit
- isalpha: test for letter
- isascii: test for ASCII character
- isascii: test for ASCII character
- isblank: test for blank character
1.1.2 Wide Character Manipulation Functions

These functions operate upon wide characters of type wchar_t. The functions test wide characters in various ways and convert them between upper and lowercase. The following functions are defined:

- iswalnum: test for letter or digit
- iswalpha: test for letter
- iswascii: test for ASCII character
- iswblank: test for blank character
- iswctrl: test for control character
- __iswcsym: test for letter, underscore or digit
- __iswcsymf: test for letter or underscore
- iswdigit: test for digit
- iswgraph: test for printable character, except space
- iswlower: test for letter in lowercase
- iswprint: test for printable character, including space
- ispunct: test for punctuation characters
- iswspace: test for "white space" characters
- iswupper: test for letter in uppercase
- iswxdigit: test for hexadecimal digit
- wctype: construct a property value for a given "property"
- iswctype: test a character for a specific property
- towlower: convert character to lowercase
- toupper: convert character to uppercase
- wctrans: construct mapping value for a given "property"
- towctrans: convert a character based on a specific property

1.1.3 Multibyte Character Manipulation Functions

These functions operate upon multibyte characters. The functions test wide characters in various ways and convert them between upper and lowercase. The following functions are defined:

- _fmbccmp: compare one multibyte character with another
- _fmbccpy: copy one multibyte character from one string to another
- _fmbccicmp: compare one multibyte character with another (case insensitive)
- _fmbcclen: return number of bytes comprising multibyte character
- _fmbclen: determine length of next multibyte character
- _fmbgetcode: get next single-byte or double-byte character from far string
_fmbputchar  store single-byte or double-byte character into far string
_fmbrlen  determine length of next multibyte character
_fmbrtowc  convert far multibyte character to wide character
_fmbstrype  return type of byte in multibyte character string
_fmbtowc  convert far multibyte character to wide character
_ismbbkalnum  test for isalnum or _ismbbkalnum
_ismbbkalpha  test for isalpha or _ismbbkalpha
_ismbbgraph  test for isgraph or _ismbbkprint
_ismbbkalnum  test for non-ASCII text symbol other than punctuation
_ismbbkana  test for single-byte Katakana character
_ismbbkapla  test for non-ASCII text symbol other than digits or punctuation
_ismbbkprint  test for non-ASCII text or non-ASCII punctuation symbol
_ismbbkpunct  test for non-ASCII punctuation character
_ismbblead  test for valid first byte of multibyte character
_ismbbpunct  test for isprint or _ismbbkprint
_ismbbpunct  test for isprint or _ismbbkprint
_ismbbrail  test for valid second byte of multibyte character
_ismbcalnum  test for _ismbcalpha or _ismbcdigit
_ismbcalpha  test for a multibyte alphabetic character
_ismbccentrl  test for a multibyte control character
_ismbcdigit  test for a multibyte decimal-digit character '0' through '9'
_ismbcgraph  test for a printable multibyte character except space
_ismbchira  test for a double-byte Hiragana character
_ismbckata  test for a double-byte Katakana character
_ismbcl0  test for a double-byte non-Kanji character
_ismbcl1  test for a JIS level 1 double-byte character
_ismbcl2  test for a JIS level 2 double-byte character
_ismbclegal  test for a valid multibyte character
_ismbclower  test for a valid lowercase multibyte character
_ismbcprint  test for any multibyte punctuation character
_ismbcspace  test for any multibyte space character
_ismbcsymbol  test for valid multibyte symbol (punctuation and other special graphics)
_ismbcupper  test for valid uppercase multibyte character
_ismbcdigit  test for any multibyte hexadecimal-digit character
_mbbtombc  return double-byte equivalent to single-byte character
_mbbtype  determine type of byte in multibyte character
_mbccmp  compare one multibyte character with another
_mbccpy  copy one multibyte character from one string to another
_mbccmp  compare one multibyte character with another (case insensitive)
_mbcjstojms  convert JIS code to shift-JIS code
_mbcjstojis  convert shift-JIS code to JIS code
_mbclen  return number of bytes comprising multibyte character
_mbctolower  convert double-byte uppercase character to double-byte lowercase character
_mbctoupper  convert double-byte lowercase character to double-byte uppercase character
_mbcchira  convert double-byte Katakana character to Hiragana character
_mbcckata  convert double-byte Hiragana character to Katakana character
_mbcctomb  return single-byte equivalent to double-byte character
_mbbgetcode  get next single-byte or double-byte character from string
_mblen  determine length of next multibyte character
_mbpuchar  store single-byte or double-byte character into string
_mbrlen  determine length of next multibyte character
_mbrtowc  convert multibyte character to wide character
_mbsbtype  return type of byte in multibyte character string

Classes of Functions 7
### 1.1.4 Memory Manipulation Functions

These functions manipulate blocks of memory. In each case, the address of the memory block and its size is passed to the function. The functions that begin with "_f" accept far pointers as their arguments allowing manipulation of any memory location regardless of which memory model your program has been compiled for. The following functions are defined:

- `_fmemccpy` copy far memory block up to a certain character
- `_fmemchr` search far memory block for a character value
- `_fmemcmp` compare any two memory blocks (near or far)
- `_fmemcpy` copy far memory block, overlap not allowed
- `_fmemicmp` compare far memory, case insensitive
- `_fmemmove` copy far memory block, overlap allowed
- `_fmemset` set any memory block (near or far) to a character

See the section "String Manipulation Functions" for descriptions of functions that manipulate strings of data. See the section "Wide String Manipulation Functions" for descriptions of functions that manipulate wide strings of data.

### 1.1.5 String Manipulation Functions

A **string** is an array of characters (with type `char`) that is terminated with an extra null character (`'\0'`). Functions are passed only the address of the string since the size can be determined by searching for the terminating character. The functions that begin with "_f" accept far pointers as their arguments allowing manipulation of any memory location regardless of which memory model your program has been compiled for. The following functions are defined:

- `bcmp` compare two byte strings
- `bcopy` copy a byte string
- `_bprintf` formatted transmission to fixed-length string
- `bzero` zero a byte string
- `_fstrcat` concatenate two far strings
- `_fstrchr` locate character in far string
- `_fstrcmp` compare two far strings
- `_fstrcpy` copy far string

---

---
Classes of Functions

- _fstrcspn: get number of string characters not from a set of characters
- _fstricmp: compare two far strings with case insensitivity
- _fstrlen: length of a far string
- _fstrlwr: convert far string to lowercase
- _fstrncat: concatenate two far strings, up to a maximum length
- _fstrncmp: compare two far strings up to maximum length
- _fstrncpy: copy a far string, up to a maximum length
- _fstrnicmp: compare two far strings with case insensitivity up to a maximum length
- _fstrncpy: fill far string with character to a maximum length
- _fstrpbrk: locate occurrence of a string within a second string
- _fstrchr: locate last occurrence of character from a character set
- _fstrrev: reverse a far string in place
- _fstrset: fill far string with a character
- _fstrspn: find number of characters at start of string which are also in a second string
- _fstrstr: find first occurrence of string in second string
- _fstrtok: get next token from a far string
- _fstrupr: convert far string to uppercase
- sprintf: formatted transmission to string
- sscanf: scan from string under format control
- strcat: concatenate string
- strchr: locate character in string
- strcmp: compare two strings
- strcmpi: compare two strings with case insensitivity
- strcoll: compare two strings using "locale" collating sequence
- strcpy: copy a string
- strcsnp: get number of string characters not from a set of characters
- _strdec: returns pointer to the previous character in string
- _strdup: allocate and duplicate a string
- strerror: get error message as string
- _stricmp: compare two strings with case insensitivity
- _strinc: return pointer to next character in string
- strlcat: concatenate string into a bounded buffer
- strlcpy: copy string into a bounded buffer
- strlen: string length
- _strlwr: convert string to lowercase
- strncat: concatenate two strings, up to a maximum length
- strncmp: compare two strings up to maximum length
- _strncpy: count the number of characters in the first "n" bytes
- strncpy: copy a string, up to a maximum length
- _strnextc: return integer value of the next character in string
- _strnicmp: compare two strings with case insensitivity up to a maximum length
- _strnicnc: increment character pointer by "n" characters
- _strnicnt: fill string with character to a maximum length
- strpbrk: locate occurrence of a string within a second string
- strrchr: locate last occurrence of character from a character set
- _strrev: reverse a string in place
- _strset: fill string with a character
- strstr: find first occurrence of string in second string
- strtok: get next token from string
- _strupr: convert string to uppercase
- strxfrm: transform string to locale’s collating sequence
- _vbprintf: same as "_bprintf" but with variable arguments
Watcom C Library Reference

vsscanf   same as "sscanf" but with variable arguments

For related functions see the sections Conversion Functions (conversions to and from strings), Time Functions (formatting of dates and times), and Memory Manipulation Functions (operate on arrays without terminating null character).

1.1.6 Wide String Manipulation Functions

A wide string is an array of wide characters (with type wchar_t) that is terminated with an extra null wide character (L'\0'). Functions are passed only the address of the string since the size can be determined by searching for the terminating character. The functions that begin with "_f" accept far pointers as their arguments allowing manipulation of any memory location regardless of which memory model your program has been compiled for. The following functions are defined:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_bwprintf</td>
<td>formatted wide character transmission to fixed-length wcsing</td>
</tr>
<tr>
<td>swprintf</td>
<td>formatted wide character transmission to string</td>
</tr>
<tr>
<td>swscanf</td>
<td>scan from wide character string under format control</td>
</tr>
<tr>
<td>_vbwprintf</td>
<td>same as &quot;_bwprintf&quot; but with variable arguments</td>
</tr>
<tr>
<td>vswscanf</td>
<td>same as &quot;swscanf&quot; but with variable arguments</td>
</tr>
<tr>
<td>wcschr</td>
<td>locate character in string</td>
</tr>
<tr>
<td>wcsncmp</td>
<td>compare two strings</td>
</tr>
<tr>
<td>wcsnmp</td>
<td>compare two strings with case insensitivity</td>
</tr>
<tr>
<td>wcsnoll</td>
<td>compare two strings using &quot;locale&quot; collating sequence</td>
</tr>
<tr>
<td>wcsncpy</td>
<td>copy a string</td>
</tr>
<tr>
<td>wcsnspm</td>
<td>get number of string characters not from a set of characters</td>
</tr>
<tr>
<td>_wcsdec</td>
<td>returns pointer to the previous character in string</td>
</tr>
<tr>
<td>_wcsdup</td>
<td>allocate and duplicate a string</td>
</tr>
<tr>
<td>wcserror</td>
<td>get error message as string</td>
</tr>
<tr>
<td>_wcsicmp</td>
<td>compare two strings with case insensitivity</td>
</tr>
<tr>
<td>_wcsicmp</td>
<td>return pointer to next character in string</td>
</tr>
<tr>
<td>wcslen</td>
<td>string length</td>
</tr>
<tr>
<td>_wcsstrw</td>
<td>convert string to lowercase</td>
</tr>
<tr>
<td>wcsncat</td>
<td>concatenate two strings, up to a maximum length</td>
</tr>
<tr>
<td>wcsncmp</td>
<td>compare two strings up to maximum length</td>
</tr>
<tr>
<td>_wcsncnt</td>
<td>count the number of characters in the first &quot;n&quot; bytes</td>
</tr>
<tr>
<td>wcsncpy</td>
<td>copy a string, up to a maximum length</td>
</tr>
<tr>
<td>_wcsnextc</td>
<td>return integer value of the next multibyte-character in string</td>
</tr>
<tr>
<td>_wcsnicmp</td>
<td>compare two strings with case insensitivity up to a maximum length</td>
</tr>
<tr>
<td>_wesninc</td>
<td>increment wide character pointer by &quot;n&quot; characters</td>
</tr>
<tr>
<td>_wcsnset</td>
<td>fill string with character to a maximum length</td>
</tr>
<tr>
<td>wespbrk</td>
<td>locate occurrence of a string within a second string</td>
</tr>
<tr>
<td>wcspbrk</td>
<td>locate last occurrence of character from a character set</td>
</tr>
<tr>
<td>_wcsprev</td>
<td>reverse a string in place</td>
</tr>
<tr>
<td>_wcsset</td>
<td>fill string with a character</td>
</tr>
<tr>
<td>wcscat</td>
<td>concatenate string into a bounded buffer</td>
</tr>
<tr>
<td>wcsclcpy</td>
<td>copy string into a bounded buffer</td>
</tr>
<tr>
<td>wcsclen</td>
<td>string length</td>
</tr>
<tr>
<td>_wcslwr</td>
<td>convert string to lowercase</td>
</tr>
<tr>
<td>wcscncat</td>
<td>concatenate two strings, up to a maximum length</td>
</tr>
<tr>
<td>wcsncmp</td>
<td>compare two strings up to maximum length</td>
</tr>
<tr>
<td>_wcspcnt</td>
<td>count the number of characters in the first &quot;n&quot; bytes</td>
</tr>
<tr>
<td>wcsncpy</td>
<td>copy a string, up to a maximum length</td>
</tr>
<tr>
<td>_wcsnextc</td>
<td>return integer value of the next multibyte-character in string</td>
</tr>
<tr>
<td>_wcsnicmp</td>
<td>compare two strings with case insensitivity up to a maximum length</td>
</tr>
<tr>
<td>_wesninc</td>
<td>increment wide character pointer by &quot;n&quot; characters</td>
</tr>
<tr>
<td>_wcsnset</td>
<td>fill string with character to a maximum length</td>
</tr>
<tr>
<td>wespbrk</td>
<td>locate occurrence of a string within a second string</td>
</tr>
<tr>
<td>wcspbrk</td>
<td>locate last occurrence of character from a character set</td>
</tr>
<tr>
<td>_wcsprev</td>
<td>reverse a string in place</td>
</tr>
<tr>
<td>_wcsset</td>
<td>fill string with a character</td>
</tr>
<tr>
<td>wcscat</td>
<td>concatenate string into a bounded buffer</td>
</tr>
<tr>
<td>wcsclcpy</td>
<td>copy string into a bounded buffer</td>
</tr>
<tr>
<td>wcsclen</td>
<td>string length</td>
</tr>
<tr>
<td>_wcslwr</td>
<td>convert string to lowercase</td>
</tr>
<tr>
<td>wcscncat</td>
<td>concatenate two strings, up to a maximum length</td>
</tr>
<tr>
<td>wcsncmp</td>
<td>compare two strings up to maximum length</td>
</tr>
<tr>
<td>_wcspcnt</td>
<td>count the number of characters in the first &quot;n&quot; bytes</td>
</tr>
<tr>
<td>wcsncpy</td>
<td>copy a string, up to a maximum length</td>
</tr>
<tr>
<td>_wcsnextc</td>
<td>return integer value of the next multibyte-character in string</td>
</tr>
<tr>
<td>_wcsnicmp</td>
<td>compare two strings with case insensitivity up to a maximum length</td>
</tr>
<tr>
<td>_wesninc</td>
<td>increment wide character pointer by &quot;n&quot; characters</td>
</tr>
<tr>
<td>_wcsnset</td>
<td>fill string with character to a maximum length</td>
</tr>
<tr>
<td>wespbrk</td>
<td>locate occurrence of a string within a second string</td>
</tr>
<tr>
<td>wcspbrk</td>
<td>locate last occurrence of character from a character set</td>
</tr>
<tr>
<td>_wcsprev</td>
<td>reverse a string in place</td>
</tr>
<tr>
<td>_wcsset</td>
<td>fill string with a character</td>
</tr>
<tr>
<td>wcscat</td>
<td>concatenate string into a bounded buffer</td>
</tr>
<tr>
<td>wcsclcpy</td>
<td>copy string into a bounded buffer</td>
</tr>
<tr>
<td>wcsclen</td>
<td>string length</td>
</tr>
<tr>
<td>_wcslwr</td>
<td>convert string to lowercase</td>
</tr>
<tr>
<td>wcscncat</td>
<td>concatenate two strings, up to a maximum length</td>
</tr>
<tr>
<td>wcsncmp</td>
<td>compare two strings up to maximum length</td>
</tr>
<tr>
<td>_wcspcnt</td>
<td>count the number of characters in the first &quot;n&quot; bytes</td>
</tr>
<tr>
<td>wcsncpy</td>
<td>copy a string, up to a maximum length</td>
</tr>
<tr>
<td>_wcsnextc</td>
<td>return integer value of the next multibyte-character in string</td>
</tr>
<tr>
<td>_wcsnicmp</td>
<td>compare two strings with case insensitivity up to a maximum length</td>
</tr>
<tr>
<td>_wesninc</td>
<td>increment wide character pointer by &quot;n&quot; characters</td>
</tr>
<tr>
<td>_wcsnset</td>
<td>fill string with character to a maximum length</td>
</tr>
<tr>
<td>wespbrk</td>
<td>locate occurrence of a string within a second string</td>
</tr>
<tr>
<td>wcspbrk</td>
<td>locate last occurrence of character from a character set</td>
</tr>
<tr>
<td>_wcsprev</td>
<td>reverse a string in place</td>
</tr>
<tr>
<td>_wcsset</td>
<td>fill string with a character</td>
</tr>
<tr>
<td>wcscat</td>
<td>concatenate string into a bounded buffer</td>
</tr>
<tr>
<td>wcsclcpy</td>
<td>copy string into a bounded buffer</td>
</tr>
<tr>
<td>wcsclen</td>
<td>string length</td>
</tr>
<tr>
<td>_wcslwr</td>
<td>convert string to lowercase</td>
</tr>
<tr>
<td>wcscncat</td>
<td>concatenate two strings, up to a maximum length</td>
</tr>
<tr>
<td>wcsncmp</td>
<td>compare two strings up to maximum length</td>
</tr>
<tr>
<td>_wcspcnt</td>
<td>count the number of characters in the first &quot;n&quot; bytes</td>
</tr>
<tr>
<td>wcsncpy</td>
<td>copy a string, up to a maximum length</td>
</tr>
<tr>
<td>_wcsnextc</td>
<td>return integer value of the next multibyte-character in string</td>
</tr>
<tr>
<td>_wcsnicmp</td>
<td>compare two strings with case insensitivity up to a maximum length</td>
</tr>
<tr>
<td>_wesninc</td>
<td>increment wide character pointer by &quot;n&quot; characters</td>
</tr>
<tr>
<td>_wcsnset</td>
<td>fill string with character to a maximum length</td>
</tr>
<tr>
<td>wespbrk</td>
<td>locate occurrence of a string within a second string</td>
</tr>
<tr>
<td>wcspbrk</td>
<td>locate last occurrence of character from a character set</td>
</tr>
<tr>
<td>_wcsprev</td>
<td>reverse a string in place</td>
</tr>
<tr>
<td>_wcsset</td>
<td>fill string with a character</td>
</tr>
<tr>
<td>wcscat</td>
<td>concatenate string into a bounded buffer</td>
</tr>
<tr>
<td>wcsclcpy</td>
<td>copy string into a bounded buffer</td>
</tr>
<tr>
<td>wcsclen</td>
<td>string length</td>
</tr>
<tr>
<td>_wcslwr</td>
<td>convert string to lowercase</td>
</tr>
<tr>
<td>wcscncat</td>
<td>concatenate two strings, up to a maximum length</td>
</tr>
<tr>
<td>wcsncmp</td>
<td>compare two strings up to maximum length</td>
</tr>
<tr>
<td>_wcspcnt</td>
<td>count the number of characters in the first &quot;n&quot; bytes</td>
</tr>
<tr>
<td>wcsncpy</td>
<td>copy a string, up to a maximum length</td>
</tr>
<tr>
<td>_wcsnextc</td>
<td>return integer value of the next multibyte-character in string</td>
</tr>
<tr>
<td>_wcsnicmp</td>
<td>compare two strings with case insensitivity up to a maximum length</td>
</tr>
<tr>
<td>_wesninc</td>
<td>increment wide character pointer by &quot;n&quot; characters</td>
</tr>
<tr>
<td>_wcsnset</td>
<td>fill string with character to a maximum length</td>
</tr>
<tr>
<td>wespbrk</td>
<td>locate occurrence of a string within a second string</td>
</tr>
<tr>
<td>wcspbrk</td>
<td>locate last occurrence of character from a character set</td>
</tr>
<tr>
<td>_wcsprev</td>
<td>reverse a string in place</td>
</tr>
<tr>
<td>_wcsset</td>
<td>fill string with a character</td>
</tr>
<tr>
<td>wcscat</td>
<td>concatenate string into a bounded buffer</td>
</tr>
<tr>
<td>wcsclcpy</td>
<td>copy string into a bounded buffer</td>
</tr>
<tr>
<td>wcsclen</td>
<td>string length</td>
</tr>
<tr>
<td>_wcslwr</td>
<td>convert string to lowercase</td>
</tr>
<tr>
<td>wcscncat</td>
<td>concatenate two strings, up to a maximum length</td>
</tr>
<tr>
<td>wcsncmp</td>
<td>compare two strings up to maximum length</td>
</tr>
<tr>
<td>_wcspcnt</td>
<td>count the number of characters in the first &quot;n&quot; bytes</td>
</tr>
<tr>
<td>wcsncpy</td>
<td>copy a string, up to a maximum length</td>
</tr>
<tr>
<td>_wcsnextc</td>
<td>return integer value of the next multibyte-character in string</td>
</tr>
<tr>
<td>_wcsnicmp</td>
<td>compare two strings with case insensitivity up to a maximum length</td>
</tr>
<tr>
<td>_wesninc</td>
<td>increment wide character pointer by &quot;n&quot; characters</td>
</tr>
<tr>
<td>_wcsnset</td>
<td>fill string with character to a maximum length</td>
</tr>
<tr>
<td>wespbrk</td>
<td>locate occurrence of a string within a second string</td>
</tr>
<tr>
<td>wcspbrk</td>
<td>locate last occurrence of character from a character set</td>
</tr>
<tr>
<td>_wcsprev</td>
<td>reverse a string in place</td>
</tr>
<tr>
<td>_wcsset</td>
<td>fill string with a character</td>
</tr>
<tr>
<td>wcscat</td>
<td>concatenate string into a bounded buffer</td>
</tr>
<tr>
<td>wcsclcpy</td>
<td>copy string into a bounded buffer</td>
</tr>
<tr>
<td>wcsclen</td>
<td>string length</td>
</tr>
<tr>
<td>_wcslwr</td>
<td>convert string to lowercase</td>
</tr>
<tr>
<td>wcscncat</td>
<td>concatenate two strings, up to a maximum length</td>
</tr>
<tr>
<td>wcsncmp</td>
<td>compare two strings up to maximum length</td>
</tr>
<tr>
<td>_wcspcnt</td>
<td>count the number of characters in the first &quot;n&quot; bytes</td>
</tr>
<tr>
<td>wcsncpy</td>
<td>copy a string, up to a maximum length</td>
</tr>
<tr>
<td>_wcsnextc</td>
<td>return integer value of the next multibyte-character in string</td>
</tr>
<tr>
<td>_wcsnicmp</td>
<td>compare two strings with case insensitivity up to a maximum length</td>
</tr>
<tr>
<td>_wesninc</td>
<td>increment wide character pointer by &quot;n&quot; characters</td>
</tr>
<tr>
<td>_wcsnset</td>
<td>fill string with character to a maximum length</td>
</tr>
<tr>
<td>wespbrk</td>
<td>locate occurrence of a string within a second string</td>
</tr>
<tr>
<td>wcspbrk</td>
<td>locate last occurrence of character from a character set</td>
</tr>
<tr>
<td>_wcsprev</td>
<td>reverse a string in place</td>
</tr>
<tr>
<td>_wcsset</td>
<td>fill string with a character</td>
</tr>
<tr>
<td>wcscat</td>
<td>concatenate string into a bounded buffer</td>
</tr>
<tr>
<td>wcsclcpy</td>
<td>copy string into a bounded buffer</td>
</tr>
<tr>
<td>wcsclen</td>
<td>string length</td>
</tr>
<tr>
<td>_wcslwr</td>
<td>convert string to lowercase</td>
</tr>
<tr>
<td>wcscncat</td>
<td>concatenate two strings, up to a maximum length</td>
</tr>
<tr>
<td>wcsncmp</td>
<td>compare two strings up to maximum length</td>
</tr>
<tr>
<td>_wcspcnt</td>
<td>count the number of characters in the first &quot;n&quot; bytes</td>
</tr>
<tr>
<td>wcsncpy</td>
<td>copy a string, up to a maximum length</td>
</tr>
<tr>
<td>_wcsnextc</td>
<td>return integer value of the next multibyte-character in string</td>
</tr>
<tr>
<td>_wcsnicmp</td>
<td>compare two strings with case insensitivity up to a maximum length</td>
</tr>
<tr>
<td>_wesninc</td>
<td>increment wide character pointer by &quot;n&quot; characters</td>
</tr>
</tbody>
</table>
For related functions see the sections Conversion Functions (conversions to and from strings), Time Functions (formatting of dates and times), and Memory Manipulation Functions (operate on arrays without terminating null character).

### 1.1.7 Multibyte String Manipulation Functions

A *wide string* is an array of wide characters (with type `wchar_t`) that is terminated with an extra null wide character (L'\0'). Functions are passed only the address of the wide string since the size can be determined by searching for the terminating character. The functions that begin with "_f" accept _far_ pointers as their arguments allowing manipulation of any memory location regardless of which memory model your program has been compiled for. The following functions are defined:

- `btowc` return wide-character version of single-byte character
- `_fnbscat` concatenate two far strings
- `_fnbschr` locate character in far string
- `_fnbscmp` compare two far strings
- `_fnbscpyp` copy far string
- `_fnbscsnpn` get number of string characters not from a set of characters
- `_fnbsdpc` returns far pointer to the previous character in far string
- `_fnbsdup` allocate and duplicate a far string
- `_fnbsicmp` compare two far strings with case insensitivity
- `_fnbsinc` return far pointer to next character in far string
- `_fnbslen` length of a far string
- `_fnbslwr` convert far string to lowercase
- `_fnbsnbcat` append up to "n" bytes of string to another string
- `_fnbsnbcmp` compare up to "n" bytes in two strings
- `_fnbsnbcnt` count the number of characters in the first "n" bytes
- `_fnbsncpy` copy up to "n" bytes of a string
- `_fnbsnicmp` compare up to "n" bytes in two strings with case insensitivity
- `_fnbsnset` fill string with up to "n" bytes
- `_fnbsncat` concatenate two far strings, up to a maximum length
- `_fnbsncnt` count the number of characters in the first "n" bytes
- `_fnbsncmp` compare two far strings up to maximum length
- `_fnbsncpy` copy a far string, up to a maximum length
- `_fnbsnextc` return integer value of the next multibyte-character in far string
- `_fnbsnicmp` compare two far strings with case insensitivity up to a maximum length
- `_fnbsninc` increment wide character far pointer by "n" characters
- `_fnbsnset` fill far string with character to a maximum length
- `_fnbspbrk` locate occurrence of a string within a second string
- `_fnbsrehr` locate last occurrence of character from a character set
- `_fnbsrev` reverse a far string in place
- `_fnbsrtowcs` convert multibyte character string to wide character string
- `_fnbsset` fill far string with a character
- `_fnbssnpn` find number of characters at start of string which are also in a second string
- `_fnbssnpn` return far pointer to first character of far string not in set
- `_fnbsstr` find first occurrence of string in second string
- `_fnbstok` get next token from a far string
- `_fnbstowecs` convert multibyte character string to wide character string
- `_fnbsupr` convert far string to uppercase
- `_fnbterm` determine if next multibyte character in string is null
- `_fnbvtap` store multibyte character into far string
- `_fwctomb` convert wide character to multibyte character and store
- `_fwcsrtombs` convert far wide character string to far multibyte character string
Watcom C Library Reference

_fwctombs  convert far wide character string to far multibyte character string
_fwctomb  convert wide character to multibyte character
_mbscat  concatenate string
_mbschr  locate character in string
_mbscmp  compare two strings
_mbscoll  compare two strings using "locale" collating sequence
_mbscpy  copy a string
_mbscsnp  get number of string characters not from a set of characters
_mbsdec  returns pointer to the previous character in string
_mbsdup  allocate and duplicate a string
_mbsicmp  compare two strings with case insensitivity
_mbsinc  return pointer to next character in string
_mbsinit  determine if mbstate_t object describes an initial conversion state
_mbslen  string length
_mbslwr  convert string to lowercase
_mbsnbcat  append up to "n" bytes of string to another string
_mbsnbcmp  compare up to "n" bytes in two strings
_mbsnbcnt  count the number of characters in the first "n" bytes
_mbsnbcpy  copy up to "n" bytes of a string
_mbsnbicmp  compare up to "n" bytes in two strings with case insensitivity
_mbsnbset  fill string with up to "n" bytes
_mbsncat  concatenate two strings, up to a maximum length
_mbsncnt  count the number of characters in the first "n" bytes
_mbsncmp  compare two strings up to maximum length
_mbsncpy  copy up to "n" bytes of a string
_mbsnicmp  compare up to "n" bytes in two strings with case insensitivity up to a maximum length
_mbsnicicmp  compare two strings with case insensitivity up to a maximum length
_mbsnicmp  count characters in the first "n" bytes
_mbsset  fill string with a character
_mbssspn  find number of characters at start of string which are also in a second string
_mbssspnp  return pointer to first character of string not in set
_mbssstr  find first occurrence of string in second string
_mbstok  get next token from string
_mbstowcs  convert multibyte character string to wide character string
_mbstrv  reverse a string
_mbstring  convert multibyte character string to wide character string
_mbstring  convert multibyte character to uppercase
_mbterm  determine if next multibyte character in string is null
_mbvtop  store multibyte character into string
_wctomb  convert wide character to multibyte character and store
_wctomb  convert wide character string to multibyte character string
_wctomb  convert wide character string to multibyte character string
_wctob  return single-byte character version of wide character
_wctomb  convert wide character to multibyte character

For related functions see the sections Conversion Functions (conversions to and from strings), Time Functions (formatting of dates and times), and Memory Manipulation Functions (operate on arrays without terminating null character).
1.1.8 Conversion Functions

These functions perform conversions between objects of various types and strings. The following functions are defined:

- `atof`  
  string to "double"
- `atoi`  
  string to "int"
- `atol`  
  string to "long int"
- `atoll`  
  string to "long long int"
- `ecvt`  
  "double" to E-format string
- `fcvt`  
  "double" to F-format string
- `gcvt`  
  "double" to string
- `itoa`  
  "int" to string
- `l1toa`  
  "long long int" to string
- `ltoa`  
  "long int" to string
- `strtod`  
  string to "double"
- `strtol`  
  string to "long int"
- `strtoll`  
  string to "long long int"
- `strtoul`  
  string to "unsigned long int"
- `strtoull`  
  string to "unsigned long long int"
- `ulltoa`  
  "unsigned long long int" to string
- `ultoa`  
  "unsigned long int" to string
- `utoa`  
  "unsigned int" to string

These functions perform conversions between objects of various types and wide character strings. The following functions are defined:

- `_itow`  
  "int" to wide character string
- `_lltow`  
  "long long int" to wide character string
- `_ltow`  
  "long int" to wide character string
- `_ultow`  
  "unsigned long long int" to wide character string
- `_ultow`  
  "unsigned long int" to wide character string
- `wcstod`  
  wide character string to "double"
- `wcstol`  
  wide character string to "long int"
- `wcstoll`  
  wide character string to "long long int"
- `wcstoul`  
  wide character string to "unsigned long int"
- `wcstoull`  
  wide character string to "unsigned long long int"
- `_wtof`  
  wide character string to "double"
- `_wtoi`  
  wide character string to "int"
- `_wtol`  
  wide character string to "long int"
- `_wtoll`  
  wide character string to "long long int"

See also `tolower`, `towlower`, `_mbctolower`, `toupper`, `towupper`, `_mbctoupper`, `strlwr`, `_wcslwr`, `_mbslwr`, `strupr`, `_wcsupr` and `_mbsupr` which convert the cases of characters and strings.
1.1.9 Memory Allocation Functions

These functions allocate and de-allocate blocks of memory.

Unless you are running your program in 32-bit protect mode, where segments have a limit of 4 gigabytes, the default data segment has a maximum size of 64K bytes. It may be less in a machine with insufficient memory or when other programs in the computer already occupy some of the memory. The _nmalloc function allocates space within this area while the _fmalloc function allocates space outside the area (if it is available).

In a small data model, the malloc, calloc and realloc functions use the _nmalloc function to acquire memory; in a large data model, the _fmalloc function is used.

It is also possible to allocate memory from a based heap using _bmalloc. Based heaps are similar to far heaps in that they are located outside the normal data segment. Based pointers only store the offset portion of the full address, so they behave much like near pointers. The selector portion of the full address specifies which based heap a based pointer belongs to, and must be passed to the various based heap functions.

It is important to use the appropriate memory-deallocation function to free memory blocks. The _nfree function should be used to free space acquired by the _ncalloc, _nmalloc, or _nrealloc functions. The _ffree function should be used to free space acquired by the _fcalloc, _fmalloc, or _frealloc functions. The _bfree function should be used to free space acquired by the _balloc, _bmalloc, or _brealloc functions.

The free function will use the _nfree function when the small data memory model is used; it will use the _ffree function when the large data memory model is being used.

It should be noted that the _fmalloc and _nmalloc functions can both be used in either data memory model. The following functions are defined:

- alloca: allocate auto storage from stack
- _balloc: allocate and zero memory from a based heap
- _bexpand: expand a block of memory in a based heap
- _bfree: free a block of memory in a based heap
- _bfreeseg: free a based heap
- _bheapseg: allocate a based heap
- _bmalloc: allocate a memory block from a based heap
- _bmsize: return the size of a memory block
- _brealloc: re-allocate a memory block in a based heap
- _callloc: allocate and zero memory
- _expand: expand a block of memory
- _fcallloc: allocate and zero a memory block (outside default data segment)
- _fexpand: expand a block of memory (outside default data segment)
- _ffree: free a block allocated using "_fmalloc"
- _fmalloc: allocate a memory block (outside default data segment)
- _fmsize: return the size of a memory block
- _frealloc: re-allocate a memory block (outside default data segment)
- _free: free a block allocated using "malloc", "calloc" or "realloc"
- _frect: return number of objects that can be allocated
- _halloc: allocate huge array
- _hfree: free huge array
- malloc: allocate a memory block (using current memory model)
1.1.10 Heap Functions

These functions provide the ability to shrink and grow the heap, as well as, find heap related problems. The following functions are defined:

- `_heapchk` perform consistency check on the heap
- `_bheapchk` perform consistency check on a based heap
- `_fheapchk` perform consistency check on the far heap
- `_nheapchk` perform consistency check on the near heap
- `_heapgrow` grow the heap
- `_fheapgrow` grow the far heap
- `_nheapgrow` grow the near heap up to its limit of 64K
- `_heapsmin` shrink the heap as small as possible
- `_bheapsmin` shrink a based heap as small as possible
- `_fheapsmin` shrink the far heap as small as possible
- `_nheapsmin` shrink the near heap as small as possible
- `_heapset` fill unallocated sections of heap with pattern
- `_bheapset` fill unallocated sections of based heap with pattern
- `_fheapset` fill unallocated sections of far heap with pattern
- `_nheapset` fill unallocated sections of near heap with pattern
- `_heapsrink` shrink the heap as small as possible
- `_fheapsrink` shrink the far heap as small as possible
- `_bheapsrink` shrink a based heap as small as possible
- `_nheapsrink` shrink the near heap as small as possible
- `_heapwalk` walk through each entry in the heap
- `_bheapwalk` walk through each entry in a based heap
- `_fheapwalk` walk through each entry in the far heap
- `_nheapwalk` walk through each entry in the near heap

1.1.11 Math Functions

These functions operate with objects of type `double`, also known as floating-point numbers. The Intel 8087 processor (and its successor chips) is commonly used to implement floating-point operations on personal computers. Functions ending in "87" pertain to this specific hardware and should be isolated in programs when portability is a consideration. The following functions are defined:

- `abs` absolute value of an object of type "int"
- `acos` arc cosine
- `acosh` inverse hyperbolic cosine
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asin</td>
<td>arcsine</td>
</tr>
<tr>
<td>asinh</td>
<td>inverse hyperbolic sine</td>
</tr>
<tr>
<td>atan</td>
<td>arctangent of one argument</td>
</tr>
<tr>
<td>atan2</td>
<td>arctangent of two arguments</td>
</tr>
<tr>
<td>atanh</td>
<td>inverse hyperbolic tangent</td>
</tr>
<tr>
<td>bessel</td>
<td>bessel functions j0, j1, jn, y0, y1, and yn</td>
</tr>
<tr>
<td>cabs</td>
<td>absolute value of complex number</td>
</tr>
<tr>
<td>ceil</td>
<td>ceiling function</td>
</tr>
<tr>
<td>_clear87</td>
<td>clears floating-point status</td>
</tr>
<tr>
<td>_control87</td>
<td>sets new floating-point control word</td>
</tr>
<tr>
<td>cos</td>
<td>cosine</td>
</tr>
<tr>
<td>cosh</td>
<td>hyperbolic cosine</td>
</tr>
<tr>
<td>div</td>
<td>compute quotient, remainder from division of an &quot;int&quot; object</td>
</tr>
<tr>
<td>exp</td>
<td>exponential function</td>
</tr>
<tr>
<td>fabs</td>
<td>absolute value of &quot;double&quot;</td>
</tr>
<tr>
<td>_finite</td>
<td>determines whether floating-point value is valid</td>
</tr>
<tr>
<td>floor</td>
<td>floor function</td>
</tr>
<tr>
<td>fmod</td>
<td>modulus function</td>
</tr>
<tr>
<td>_fpreset</td>
<td>initializes for floating-point operations</td>
</tr>
<tr>
<td>frexp</td>
<td>fractional exponent</td>
</tr>
<tr>
<td>hypot</td>
<td>compute hypotenuse</td>
</tr>
<tr>
<td>imaxabs</td>
<td>get quotient, remainder from division of object of maximum-size integer type</td>
</tr>
<tr>
<td>imaxdiv</td>
<td>absolute value of an object of maximum-size integer type</td>
</tr>
<tr>
<td>j0</td>
<td>return Bessel functions of the first kind (described under &quot;bessel Functions&quot;)</td>
</tr>
<tr>
<td>j1</td>
<td>return Bessel functions of the first kind (described under &quot;bessel Functions&quot;)</td>
</tr>
<tr>
<td>jn</td>
<td>return Bessel functions of the first kind (described under &quot;bessel Functions&quot;)</td>
</tr>
<tr>
<td>labs</td>
<td>absolute value of an object of type &quot;long int&quot;</td>
</tr>
<tr>
<td>ldexp</td>
<td>multiply by a power of two</td>
</tr>
<tr>
<td>ldiv</td>
<td>get quotient, remainder from division of object of type &quot;long int&quot;</td>
</tr>
<tr>
<td>log</td>
<td>natural logarithm</td>
</tr>
<tr>
<td>log10</td>
<td>logarithm, base 10</td>
</tr>
<tr>
<td>log2</td>
<td>logarithm, base 2</td>
</tr>
<tr>
<td>matherr</td>
<td>handles error from math functions</td>
</tr>
<tr>
<td>max</td>
<td>return maximum of two arguments</td>
</tr>
<tr>
<td>min</td>
<td>return minimum of two arguments</td>
</tr>
<tr>
<td>modf</td>
<td>get integral, fractional parts of &quot;double&quot;</td>
</tr>
<tr>
<td>pow</td>
<td>raise to power</td>
</tr>
<tr>
<td>rand</td>
<td>random integer</td>
</tr>
<tr>
<td>_set_matherr</td>
<td>specify a math error handler</td>
</tr>
<tr>
<td>sin</td>
<td>sine</td>
</tr>
<tr>
<td>sinh</td>
<td>hyperbolic sine</td>
</tr>
<tr>
<td>sqrt</td>
<td>square root</td>
</tr>
<tr>
<td>srand</td>
<td>set starting point for generation of random numbers using &quot;rand&quot; function</td>
</tr>
<tr>
<td>_status87</td>
<td>gets floating-point status</td>
</tr>
<tr>
<td>tan</td>
<td>tangent</td>
</tr>
<tr>
<td>tanh</td>
<td>hyperbolic tangent</td>
</tr>
<tr>
<td>y0</td>
<td>return Bessel functions of the second kind (described under &quot;bessel&quot;)</td>
</tr>
<tr>
<td>y1</td>
<td>return Bessel functions of the second kind (described under &quot;bessel&quot;)</td>
</tr>
<tr>
<td>yn</td>
<td>return Bessel functions of the second kind (described under &quot;bessel&quot;)</td>
</tr>
</tbody>
</table>
### 1.1.12 Searching Functions

These functions provide searching and sorting capabilities. The following functions are defined:

- `bsearch`: find a data item in an array using binary search
- `lfind`: find a data item in an array using linear search
- `lsearch`: linear search array, add item if not found
- `qsort`: sort an array

### 1.1.13 Time Functions

These functions are concerned with dates and times. The following functions are defined:

- `asctime`: makes time string from time structure
- `_asctime`: makes time string from time structure
- `_wasctime`: makes time string from time structure
- `__wasctime`: makes time string from time structure
- `clock`: gets time since program start
- `ctime`: gets calendar time string
- `_ctime`: gets calendar time string
- `_wctime`: gets calendar time string
- `__wctime`: gets calendar time string
- `difftime`: calculate difference between two times
- `ftime`: returns the current time in a "timeb" structure
- `gmtime`: convert calendar time to Coordinated Universal Time (UTC)
- `_gmtime`: convert calendar time to Coordinated Universal Time (UTC)
- `localtime`: convert calendar time to local time
- `_localtime`: convert calendar time to local time
- `mktime`: make calendar time from local time
- `_strdate`: return date in buffer
- `strftime`: format date and time
- `wcsftime`: format date and time
- `_wstrftime`: format date and time
- `strtime`: return time in buffer
- `wstrtime`: return time in buffer
- `time`: get current calendar time
- `tzset`: set global variables to reflect the local time zone
- `_wstrdate`: return date in buffer

### 1.1.14 Variable-length Argument Lists

Variable-length argument lists are used when a function does not have a fixed number of arguments. These macros provide the capability to access these arguments. The following functions are defined:

- `va_arg`: get next variable argument
- `va_end`: complete access of variable arguments
- `va_start`: start access of variable arguments
1.1.15 Stream I/O Functions

A stream is the name given to a file or device which has been opened for data transmission. When a stream is opened, a pointer to a FILE structure is returned. This pointer is used to reference the stream when other functions are subsequently invoked.

There are two modes by which data can be transmitted:

- **binary**: Data is transmitted unchanged.
- **text**: On input, carriage-return characters are removed before following linefeed characters. On output, carriage-return characters are inserted before linefeed characters.

These modes are required since text files are stored with the two characters delimiting a line of text, while the C convention is for only the linefeed character to delimit a text line.

When a program begins execution, there are a number of streams already open for use:

- **stdin**: Standard Input: input from the console
- **stdout**: Standard Output: output to the console
- **stderr**: Standard Error: output to the console (used for error messages)
- **stdaux**: Standard Auxiliary: auxiliary port, available for use by a program (not available in some Windows platforms)
- **stdprn**: Standard Printer: available for use by a program (not available in some Windows platforms)

These standard streams may be re-directed by use of the freopen function.

See also the section File Manipulation Functions for other functions which operate upon files.

The functions referenced in the section Operating System I/O Functions may also be invoked (use the fileno function to obtain the file handle). Since the stream functions may buffer input and output, these functions should be used with caution to avoid unexpected results.

The following functions are defined:

- `clearerr` clear end-of-file and error indicators for stream
- `fclose` close stream
- `fcloseall` close all open streams
- `fdopen` open stream, given handle
- `feof` test for end of file
- `ferror` test for file error
- `fflush` flush output buffer
- `fgetc` get next character from file
- `_fgetchar` equivalent to "fgetc" with the argument "stdin"
- `fgetpos` get current file position
- `fgets` get a string
- `flushall` flush output buffers for all streams
- `fopen` open a stream
The previous section describes some general aspects of stream input/output. The following describes functions dealing with streams containing multibyte character sequences.

After a stream is associated with an external file, but before any operations are performed on it, the stream is without orientation. Once a wide character input/output function has been applied to a stream without orientation, the stream becomes wide-oriented. Similarly, once a byte input/output function has been applied to a stream without orientation, the stream becomes byte-oriented. Only a successful call to `freopen` can otherwise alter the orientation of a stream (it removes any orientation). You cannot mix byte input/output functions and wide character input/output functions on the same stream.

A file positioning function can cause the next wide character output function to overwrite a partial multibyte character. This can lead to the subsequent reading of a stream of multibyte characters containing an invalid character.

When multibyte characters are read from a stream, they are converted to wide characters. Similarly, when wide characters are written to a stream, they are converted to multibyte characters.
The following functions are defined:

- `fgetwc`: get next wide character from file
- `_fgetwchar`: equivalent to "fgetwc" with the argument "stdin"
- `fgetws`: get a wide character string
- `fprintf`: "C" and "S" extensions to the format specifier
- `fputwc`: write a wide character
- `_fputwchar`: write a character to the "stdout" stream
- `fputws`: write a wide character string
- `fscanf`: "C" and "S" extensions to the format specifier
- `fwprintf`: formatted wide character output
- `fscanf`: scan wide character input according to format
- `getwc`: read wide character
- `getwchar`: get next wide character from "stdin"
- `getws`: get wide character string from "stdin"
- `putwc`: write wide character to file
- `putwchar`: write wide character to "stdout"
- `_putws`: write wide character string to "stdout"
- `ungetwc`: push wide character back on input stream
- `vfwprintf`: same as "fwprintf" but with variable arguments
- `vfscanf`: same as "fscanf" but with variable arguments
- `vwprintf`: same as "wprintf" but with variable arguments
- `vwscanf`: same as "wscanf" but with variable arguments
- `_wfdopen`: open stream, given handle using a wide character "mode"
- `_wopen`: open a stream using wide character arguments
- `_wfreopen`: re-opens a stream using wide character arguments
- `_wfopen`: open a shared stream using wide character arguments
- `_wperror`: write error message to "stderr" stream
- `wprintf`: format wide character output to "stdout"
- `wscanf`: scan wide character input from "stdin" under format control

See the section Directory Functions for functions which are related to directories.

### 1.1.17 Process Primitive Functions

These functions deal with process creation, execution and termination, signal handling, and timer operations.

When a new process is started, it may replace the existing process

- **P_OVERLAY** is specified with the `spawn...` functions
- **the exec...** routines are invoked

or the existing process may be suspended while the new process executes (control continues at the point following the place where the new process was started)

- **P_WAIT** is specified with the `spawn...` functions
- **system** is used

The following functions are defined:

20 Classes of Functions
There are eight spawn... and exec... functions each. The "..." is one to three letters:
• "l" or "v" (one is required) to indicate the way the process parameters are passed

• "p" (optional) to indicate whether the PATH environment variable is searched to locate the program for the process

• "e" (optional) to indicate that the environment variables are being passed

1.1.18 Process Environment

These functions deal with process identification, process groups, system identification, system time, environment variables, and terminal identification. The following functions are defined:

_bgetcmd get command line
clearenv delete environment variables
cgetm get command line
getpid return process ID of calling process
getenv get environment variable value
isatty determine if file descriptor associated with a terminal
putenv add, change or delete environment variable
_searchenv search for a file in list of directories
setenv add, change or delete environment variable
_wgetenv get environment variable value
_wputenv add, change or delete environment variable
_wsearchenv search for a file in list of directories
_wsetenv add, change or delete environment variable

1.1.19 Directory Functions

These functions pertain to directory manipulation. The following functions are defined:

chdir change current working directory
closedir close opened directory file
cwd get current working directory
mkdir make a new directory
opendir open directory file
readdir read file name from directory
rewinddir reset position of directory stream
rmdir remove a directory
_wchdir change current working directory
_wclosedir close opened directory file
_wgetcwd get current working directory
_wgetcwd get current directory on drive
_wmkdir make a new directory
_wopendir open directory file
_wreadir read file name from directory
_wrewinddir reset position of directory stream
_wrmdir remove a directory
1.1.20 Operating System I/O Functions

These functions operate at the operating-system level and are included for compatibility with other C implementations. It is recommended that the functions used in the section File Manipulation Functions be used for new programs, as these functions are defined portably and are part of the ANSI standard for the C language.

The functions in this section reference opened files and devices using a file handle which is returned when the file is opened. The file handle is passed to the other functions.

The following functions are defined:

- `chsize`  
  change the size of a file
- `close`  
  close file
- `creat`  
  create a file
- `dup`  
  duplicate file handle, get unused handle number
- `dup2`  
  duplicate file handle, supply new handle number
- `eof`  
  test for end of file
- `filelength`  
  get file size
- `fileno`  
  get file handle for stream file
- `fstat`  
  get file status
- `fsync`  
  write queued file and filesystem data to disk
- `_hdopen`  
  get POSIX handle from OS handle
- `lock`  
  lock a section of a file
- `locking`  
  lock/unlock a section of a file
- `lseek`  
  set current file position
- `open`  
  open a file
- `_os_handle`  
  get OS handle from POSIX handle
- `read`  
  read a record
- `setmode`  
  set file mode
- `sopen`  
  open a file for shared access
- `tell`  
  get current file position
- `umask`  
  set file permission mask
- `unlink`  
  delete a file
- `unlock`  
  unlock a section of a file
- `write`  
  write a record
- `_wcreat`  
  create a file
- `_wopen`  
  open a file
- `_wpopen`  
  open a pipe
- `_wsopen`  
  open a file for shared access
- `_wunlink`  
  delete a file

1.1.21 File Manipulation Functions

These functions operate directly with files. The following functions are defined:

- `access`  
  test file or directory for mode of access
- `chmod`  
  change permissions for a file
- `lstat`  
  get file status
- `remove`  
  delete a file
- `rename`  
  rename a file
- `stat`  
  get file status
tmpnam  create name for temporary file
utime  set modification time for a file
_waccess  test file or directory for mode of access
_wchmod  change permissions for a file
_wremove  delete a file
_wrename  rename a file
_wstat  get file status
_wtmpnam  create name for temporary file
_wutime  set modification time for a file

1.1.22 Console I/O Functions

These functions provide the capability to read and write data from the console. Data is read or written without any special initialization (devices are not opened or closed), since the functions operate at the hardware level.

The following functions are defined:

cgets  get a string from the console
cprintf  print formatted string to the console
cputs  write a string to the console
cscanf  scan formatted data from the console
getch  get character from console, no echo
getche  get character from console, echo it
kbhit  test if keystroke available
putch  write a character to the console
ungetch  push back next character from console

1.1.23 Default Windowing Functions

These functions provide the capability to manipulate attributes of various windows created by Watcom’s default windowing system for Microsoft Windows and IBM OS/2.

The following functions are defined:

_dwDeleteOnClose  delete console window upon close
_dwSetAboutDlg  set about dialogue box title and contents
_dwSetAppNameTitle  set main window’s application title
_dwSetConTitle  set console window’s title
_dwShutDown  shut down default windowing system
_dwYield  yield control to other processes

1.1.24 BIOS Functions

This set of functions allows access to services provided by the BIOS. The following functions are defined:

_bios_disk  provide disk access functions
_bios_equiplist  determine equipment list
_bios_keybrd  provide low-level keyboard access
_bios_memsiz  determine amount of system board memory
_bios_print  provide access to printer services

24 Classes of Functions
_bios_serialcom
provide access to serial services

_bios_timeofday
get and set system clock

### 1.1.25 DOS-Specific Functions

These functions provide the capability to invoke DOS functions directly from a program. The following functions are defined:

- `bdos`: DOS call (short form)
- `dosexterr`: extract DOS error information
- `_dos_allocmem`: allocate a block of memory
- `_dos_close`: close a file
- `_dos_commit`: flush buffers to disk
- `_dos_creat`: create a file
- `_dos_creatnew`: create a new file
- `_dos_findclose`: close find file matching
- `_dos_findfirst`: find first file matching a specified pattern
- `_dos_findnext`: find the next file matching a specified pattern
- `_dos_freemem`: free a block of memory
- `_dos_getdate`: get current system date
- `_dos_getdiskfree`: get information about disk
- `_dos_getdrive`: get the current drive
- `_dos_getfileattr`: get file attributes
- `_dos_gettime`: get the current system time
- `_dos_getvect`: get contents of interrupt vector
- `_dos_keep`: install a terminate-and-stay-resident program
- `_dos_open`: open a file
- `_dos_read`: read data from a file
- `_dos_setblock`: change the size of allocated block
- `_dos_setdate`: change current system date
- `_dos_setdrive`: change the current default drive
- `_dos_setfileattr`: set the attributes of a file
- `_dos_setftime`: set a file’s last modification time
- `_dos_settime`: set a file’s last modification time
- `_dos_setvect`: set an interrupt vector
- `_dos_write`: write data to a file
- `intdos`: cause DOS interrupt
- `intdosx`: cause DOS interrupt, with segment registers
- `_wdos_findclose`: close find file matching
- `_wdos_findfirst`: find first file matching a specified pattern
- `_wdos_findnext`: find the next file matching a specified pattern

### 1.1.26 Intel 80x86 Architecture-Specific Functions

These functions provide the capability to invoke Intel 80x86 processor-related functions directly from a program. Functions that apply to the Intel 8086 CPU apply to that family including the 80286, 80386, 80486 and Pentium processors. The following functions are defined:

- `_chain_intr`: chain to the previous interrupt handler
- `_disable`: disable interrupts
- `_enable`: enable interrupts
1.1.27 Intel Pentium Multimedia Extension Functions

This set of functions allows access to Intel Architecture Multimedia Extensions (MMX). These functions are implemented as in-line intrinsic functions. The general format for most functions is:

\[
\text{mm_result} = \text{mm_function}(\text{mm_operand1}, \text{mm_operand2});
\]

These functions provide a simple model for use of Intel Multimedia Extension (MMX). More advanced use of MMX can be implemented in much the same way that these functions are implemented. See the `<mmintrin.h>` header file for examples. The following functions are defined:

- `_m_empty` empty multimedia state
- `_m_from_int` form 64-bit MM value from unsigned 32-bit integer value
- `_m_packssdw` pack and saturate 32-bit double-words from two MM elements into signed 16-bit words
- `_m_packsswb` pack and saturate 16-bit words from two MM elements into signed bytes
- `_m_paddb` add packed bytes
- `_m_padddd` add packed 32-bit double-words
- `_m_paddsb` add packed signed bytes with saturation
- `_m_paddsw` add packed signed 16-bit words with saturation
- `_m_paddusb` add packed unsigned bytes with saturation
- `_m_paddusw` add packed unsigned 16-bit words with saturation
- `_m_paddw` add packed 16-bit words
- `_m_pand` AND 64 bits of two MM elements
- `_m_pandn` invert the 64 bits in MM element, then AND 64 bits from second MM element
- `_m_pcmpeqb` compare packed bytes for equality
- `_m_pcmpeqd` compare packed double-words for equality
- `_m_pcmpeqw` compare packed 16-bit words for equality
- `_m_pcmpgtb` compare packed bytes for greater than relationship
- `_m_pcmpgtd` compare packed double-words for greater than relationship
- `_m_pcmpgtw` compare packed 16-bit words for greater than relationship
- `_m_pmaddwd` multiply packed 16-bit words, then add 32-bit results pair-wise
- `_m_pmulhw` multiply the packed 16-bit words of two MM elements, then store high-order 16 bits of results
- `_m_pmulhw` multiply the packed 16-bit words of two MM elements, then store low-order 16 bits of results
OR 64 bits of two MM elements

shift left each 32-bit double-word by amount specified in second MM element

shift left each 32-bit double-word by amount specified in constant value

shift left each 64-bit quad-word by amount specified in second MM element

shift left each 64-bit quad-word by amount specified in constant value

shift left each 16-bit word by amount specified in second MM element

shift left each 16-bit word by amount specified in constant value

shift right (with sign propagation) each 32-bit double-word by amount specified in second MM element

shift right (with sign propagation) each 32-bit double-word by amount specified in constant value

shift right (with sign propagation) each 16-bit word by amount specified in second MM element

shift right (with sign propagation) each 16-bit word by amount specified in constant value

shift right (with zero fill) each 32-bit double-word by an amount specified in second MM element

shift right (with zero fill) each 32-bit double-word by an amount specified in constant value

shift right (with zero fill) each 64-bit quad-word by an amount specified in second MM element

shift right (with zero fill) each 64-bit quad-word by an amount specified in constant value

shift right (with zero fill) each 16-bit word by an amount specified in second MM element

shift right (with zero fill) each 16-bit word by an amount specified in constant value

subtract packed bytes in MM element from second MM element

subtract packed 32-bit dwords in MM element from second MM element

subtract packed signed bytes in MM element from second MM element with saturation

subtract packed signed 16-bit words in MM element from second MM element with saturation

subtract packed unsigned bytes in MM element from second MM element with saturation

subtract packed unsigned 16-bit words in MM element from second MM element with saturation

subtract packed 16-bit words in MM element from second MM element with saturation

interleave bytes from the high halves of two MM elements

interleave 32-bit double-words from the high halves of two MM elements

interleave 16-bit words from the high halves of two MM elements

interleave bytes from the low halves of two MM elements

interleave 32-bit double-words from the low halves of two MM elements

interleave 16-bit words from the low halves of two MM elements

XOR 64 bits from two MM elements

retrieve low-order 32 bits from MM value

1.1.28 Miscellaneous Functions

The following functions are defined:

assert

test an assertion and output a string upon failure

_fullpath

return full path specification for file
_getmbcp get current multibyte code page
getopt a command-line parser that can be used by applications that follow guidelines outlined in the Single UNIX Specification
_harderr critical error handler
_hardresum critical error handler resume
localeconv obtain locale specific conversion information
longjmp return and restore environment saved by "setjmp"
_rotl rotate an "unsigned long" left
_rotl rotate an "unsigned long" right
main the main program (user written)
offsetof get offset of field in structure
_rotl rotate an "unsigned int" left
_rotl rotate an "unsigned int" right
setjmp save environment for use with "longjmp" function
_makepath make a full filename from specified components
localeconv get locale category
_setmbcp set current multibyte code page
_splitpath split a filename into its components
_splitpath2 split a filename into its components
_wfullpath return full path specification for file
_wmakepath make a full filename from specified components
_wsetlocale set locale category
_wsplitpath split a filename into its components
_wsplitpath2 split a filename into its components

1.2 Header Files

The following header files are supplied with the C library. As has been previously noted, when a library function is referenced in a source file, the related header files (shown in the synopsis for that function) should be included into that source file. The header files provide the proper declarations for the functions and for the number and types of arguments used with them. Constant values used in conjunction with the functions are also declared. The files can be included multiple times and in any order.

1.2.1 Header Files in /watcom/h

The following header files are provided with the software. The header files that are located in the \WATCOM\H directory are described first.

assert.h This ISO C90 header file is required when an assert macro is used. These assertions will be ignored when the identifier NDEBUG is defined.
bios.h This header file declares all BIOS related functions.
conio.h This header file declares console and Intel 80x86 port input/output functions.
ctype.h This ISO C90 header file declares functions that perform character classification and case conversion operations. Similar functions for wide characters are declared in <wctype.h>.
direct.h This header file declares functions related to directories and the type DIR which describes an entry in a directory.
**dos.h**  This header file declares functions that interact with DOS. It includes the definitions of the `FP_OFF`, `FP_SEG` and `MK_FP` macros, and for the following structures and unions:

- **DOSERROR** describes the DOS error information.
- **REGS** describes the CPU registers for Intel 8086 family.
- **SREGS** describes the segment registers for the Intel 8086 family.
- **REGPACK** describes the CPU registers and segment registers for Intel 8086 family.
- **INTPACK** describes the input parameter to an "interrupt" function.

**env.h**  This POSIX header file declares environment string functions.

**errno.h**  This ISO C90 header file provides the `extern` declaration for error variable `errno` and provides the symbolic names for error codes that can be placed in the error variable.

**fcntl.h**  This POSIX header file defines the flags used by the `open` and `sopen` functions. The function declarations for these functions are found in the `<io.h>` header file.

**fenv.h**  This ISO C99 header file defines several types and declares several functions that give access to the floating point environment. These functions can be used to control status flags and control modes in the floating point processor.

**float.h**  This ISO C90 header file declares constants related to floating-point numbers, declarations for low-level floating-point functions, and the declaration of the floating-point exception codes.

**fnmatch.h**  This header file declares the pattern matching function `fnmatch`

**graph.h**  This header file contains structure definitions and function declarations for the Watcom C Graphics library functions.

**inttypes.h**  This ISO C99 header file includes `<stdint.h>` and expands on it by definition macros for printing and scanning specific sized integer types. This header also declares several functions for manipulating maximum sized integers.

Note that the format macros are not visible in C++ programs unless the macro `__STDC_FORMAT_MACROS` is defined.

**io.h**  This header file declares functions that perform input/output operations at the operating system level. These functions use file handles to reference files or devices. The function `fstat` is declared in the `<sys/stat.h>` header file.

**limits.h**  This ISO C90 header file contains constant declarations for limits or boundary values for ranges of integers and characters.

**locale.h**  This ISO C90 header file contains declarations for the categories (LC...) of locales which can be selected using the `setlocale` function which is also declared.

**malloc.h**  This header file declares the memory allocation and deallocation functions.
math.h
This ANSI header file declares the mathematical functions (which operate with floating-point numbers) and the structures:

exception describes the exception structure passed to the matherr function; symbolic constants for the types of exceptions are included

complex declares a complex number

mmintrin.h
This header file declares functions that interact with the Intel Architecture Multimedia Extensions. It defines the datatype used to store multimedia values:

__m64 describes the 64-bit multimedia data element. Note: the underlying implementation details of this datatype are subject to change. Other compilers may implement a similar datatype in a different manner.

It also contains prototypes for multimedia functions and pragmas for the in-line generation of code that operates on multimedia registers.

process.h
This header file declares the spawn... functions, the exec... functions, and the system function. The file also contains declarations for the constants P_WAIT, P_NOWAIT, P_NOWAITO, and P_OVERLAY.

search.h
This header file declares the functions lfind and lsearch

setjmp.h
This ISO C90 header file declares the setjmp and longjmp functions.

share.h
This header file defines constants for shared access to files using the sopen function.

signal.h
This ISO C90 header file declares the signal and raise functions.

stdarg.h
This ISO C90 header file defines the macros which handle variable argument lists.

stdbool.h
This ISO C99 header file defines the macro bool and the macros true and false for use in C programs. If this header is included in a C++ program there is no effect. The C++ reserved words will not be redefined. However the definition of bool, true, and false used in a C program will be compatible with their C++ counterparts. In particular, a C function declared as taking a bool parameter and a structure containing a bool member can both be shared between C and C++ without error.

stddef.h
This ISO C90 header file defines a few popular constants and types including NULL (null pointer), size_t (unsigned size of an object), and ptrdiff_t (difference between two pointers). It also contains a declaration for the offsetof macro.

stdint.h
This ISO C99 header file defines numerous type names for integers of various sizes. Such type names provide a reasonably portable way to refer to integers with a specific number of bits. This header file also defines macros that describe the minimum and maximum values for these types (similar to the macros in limits.h), and macros for writing integer constants with specific sized types.

Note that in C++ programs the limit macros are not visible unless the macro __STDC_LIMIT_MACROS is defined. Similarly the constant writing macros are not visible unless the macro __STDC_CONSTANT_MACROS is defined.
C Library Overview

**stdio.h**
This ISO C90 header file declares the standard input/output functions. Files, devices and directories are referenced using pointers to objects of the type `FILE`.

**stdlib.h**
This ISO C90 header file declares many standard functions excluding those declared in other header files discussed in this section.

**string.h**
This ISO C90 header file declares functions that manipulate strings or blocks of memory.

**time.h**
This ANSI header file declares functions related to times and dates and defines the structure `struct tm`.

**varargs.h**
This UNIX System V header file provides an alternate way of handling variable argument lists. The equivalent ANSI header file is `<stdarg.h>`.

**wchar.h**
This ISO C99 header file defines several data types including `wchar_t`, `size_t`, `mbstate_t` (an object that can hold conversion state information necessary to convert between multibyte characters and wide characters), `wctype_t` (a scalar type that can hold values which represent locale-specific character classification), and `wint_t` which is an integral type that can hold any `wchar_t` value as well as `WEOF` (a character that is not in the set of "wchar_t" characters and that is used to indicate end-of-file on an input stream). The functions that are declared in this header file are grouped as follows:

- Wide character classification and case conversion.
- Input and output of wide characters, or multibyte characters, or both.
- Wide string numeric conversion.
- Wide string manipulation.
- Wide string data and time conversion.
- Conversion between multibyte and wide character sequences.

**wctype.h**
This ISO C99 header file declares functions that perform character classification and case conversion operations on wide characters. Similar functions for ordinary characters are declared in `<ctype.h>`.

### 1.2.2 Header Files in /watcom/h/sys

The following header files are present in the `sys` subdirectory. Their presence in this directory indicates that they are system-dependent header files.

**sys\locking.h**
This header file contains the manifest constants used by the `locking` function.

**sys\stat.h**
This POSIX header file contains the declarations pertaining to file status, including definitions for the `fstat` and `stat` functions and for the structure:

```c
struct stat
```

- `stat` describes the information obtained for a directory, file or device

**sys\timeb.h**
This header file describes the `timeb` structure used in conjunction with the `ftime` function.
1.3 Global Data

Certain data items are used by the Watcom C/C++ run-time library and may be inspected (or changed in some cases) by a program. The defined items are:

- **amblksize**: Prototype in `<stdlib.h>`. This unsigned int data item contains the increment by which the "break" pointer for memory allocation will be advanced when there is no freed block large enough to satisfy a request to allocate a block of memory. This value may be changed by a program at any time.

- **argc**: Prototype in `<stdlib.h>`. This int item contains the number of arguments passed to `main`.

- **argv**: Prototype in `<stdlib.h>`. This char ** item contains a pointer to a vector containing the actual arguments passed to `main`.

- **daylight**: Prototype in `<time.h>`. This unsigned int has a value of one when daylight saving time is supported in this locale and zero otherwise. Whenever a time function is called, the `tzset` function is called to set the value of the variable. The value will be determined from the value of the TZ environment variable.

- **doserrno**: Prototype in `<stdlib.h>`. This int item contains the actual error code returned when a DOS, Windows or OS/2 function fails.

- **environ**: Prototype in `<stdlib.h>`. This char ** __near data item is a pointer to an array of character pointers to the environment strings.

- **errno**: Prototype in `<errno.h>`. This int item contains the number of the last error that was detected. The run-time library never resets `errno` to 0. Symbolic names for these errors are found in the `<errno.h>` header file. See the descriptions for the `perror` and `strerror` functions for information about the text which describes these errors.

- **fltused_**: The C compiler places a reference to the `fltused_` symbol into any module that uses a floating-point library routine or library routine that requires floating-point support (e.g., the use of a `float` or `double` as an argument to the `printf` function).

- **fmode**: Prototype in `<stdlib.h>`. This data item contains the default type of file (text or binary) translation for a file. It will contain a value of either
O_BINARY indicates that data is transmitted to and from streams unchanged.

O_TEXT indicates that carriage return characters are added before linefeed characters on output operations and are removed on input operations when they precede linefeed characters.

These values are defined in the `<fcntl.h>` header file. The value of _fmode may be changed by a program to change the default behavior of the open, fopen, creat and sopen functions. The default setting of _fmode is O_TEXT, for text-mode translation. O_BINARY is the setting for binary mode. You can change the value of _fmode in either of two ways:

- You can include the object file BINMODE.OBJ when linking your application. This object file contains code to change the initial setting of _fmode to O_BINARY, causing all files except stdin, stdout, and stderr to be opened in binary mode.

- You can change the value of _fmode directly by setting it in your program.

__MaxThreads

There is a limit to the number of threads an application can create under 16-bit OS/2 and 32-bit NetWare. The default limit is 32. This limit can be adjusted by statically initializing the unsigned global variable __MaxThreads.

Under 32-bit OS/2, there is no limit to the number of threads an application can create. However, due to the way in which multiple threads are supported in the Watcom libraries, there is a small performance penalty once the number of threads exceeds the default limit of 32 (this number includes the initial thread). If you are creating more than 32 threads and wish to avoid this performance penalty, you can redefine the threshold value of 32. You can statically initialize the global variable __MaxThreads.

By adding the following line to your multi-threaded application, the new threshold value will be set to 48.

```
unsigned __MaxThreads = { 48 };
```

__minreal

Prototype in `<stdlib.h>`.
This data item contains the minimum amount of real memory (below 640K) to reserve when running a 32-bit DOS extended application.

optarg

Prototype in `<unistd.h>`.
This char * variable contains a pointer to an option-argument parsed by the getopt function.

opterr

Prototype in `<unistd.h>`.
The int variable controls whether the getopt function will print error messages. The default value is non-zero and will cause the getopt function to print error messages on the console.

optind

Prototype in `<unistd.h>`.
The int variable holds the index of the argument array element currently processed by the getopt function.

optopt

Prototype in `<unistd.h>`.
This int variable contains the unrecognized option character in case the getopt function returns an error.

_omajor
Prototype in <stdlib.h>.
This unsigned char variable contains the major number for the version of DOS executing on the computer. If the current version is 3.20, then the value will be 3.

_osminor
Prototype in <stdlib.h>.
This unsigned char variable contains the minor number for the version of DOS executing on the computer. If the current version is 3.20, then the value will be 20.

_osbuild
(Win32 only) Prototype in <stdlib.h>.
This unsigned short variable contains the operating system build number for the version of Windows executing on the computer.

_osver
(Win32 only) Prototype in <stdlib.h>.
This unsigned int variable contains the operating system build number for the version of Windows executing on the computer.

On Win32s or Windows 95/98 platforms, the high bit of the low-order 16-bit word is turned on. Windows 95/98 do not have build numbers.

unsigned short dwBuild;
// Get build numbers for Win32 or Win32s
if( _osver < 0x8000 )        // Windows NT/2000
dwBuild = _osver;
else if (_winmajor < 4)   // Win32s
    dwBuild = _osver & 0x8000;
else                           // Windows 95 or 98
    dwBuild = 0;             // No build numbers provided

Note that the Win32 GetVersionEx function is the preferred method for obtaining operating system version number information.

_osmode
(16-bit only) Prototype in <stdlib.h>.
This unsigned char variable contains either the value DOS_MODE which indicates the program is running in real address mode, or it contains the value OS2_MODE which indicates the program is running in protected address mode.

_psp
Prototype in <stdlib.h>.
This data item contains the segment value for the DOS Program Segment Prefix. Consult the technical documentation for your DOS system for the process information contained in the Program Segment Prefix.

_stacksize
On 16-bit 80x86 systems, this unsigned int value contains the size of the stack for a TINY memory model program. Changing the value of this item during the execution of a program will have no effect upon the program, since the value is used when the program starts execution. To change the size of the stack to be 8K bytes, a statement such as follows can be included with the program.

    unsigned int _stacksize = { 8 * 1024 };

_stdaux
Prototype in <stdio.h>.
This variable (with type FILE *) indicates the standard auxiliary port (not available in some Windows platforms).
**C Library Overview**

- **stderr**: Prototype in `<stdio.h>`. This variable (with type FILE *) indicates the standard error stream (set to the console by default).

- **stdin**: Prototype in `<stdio.h>`. This variable (with type FILE *) indicates the standard input stream (set to the console by default).

- **stdout**: Prototype in `<stdio.h>`. This variable (with type FILE *) indicates the standard output stream (set to the console by default).

- **stdprn**: Prototype in `<stdio.h>`. This variable (with type FILE *) indicates the standard printer. (not available in some Windows platforms).

- **sys_errlist**: Prototype in `<stdlib.h>`. This variable is an array of pointers to character strings for each error code defined in the `<errno.h>` header file.

- **sys_nerr**: Prototype in `<stdlib.h>`. This int variable contains the number of messages declared in `sys_errlist`.

- **_threadid**: Prototype in `<stddef.h>`. This variable/function may be used to obtain the id of the current thread which is an int. In the 32-bit libraries, `_threadid` is a function that returns a pointer to an int. In the 16-bit libraries, `_threadid` is a far pointer to an int. Note that the value stored where `_threadid` points does not necessarily change when a thread context switch occurs (so do not make a copy of the pointer ... it may change). To obtain the current thread identifier, simply code:

  ```c
  int tid = *_threadid;
  ```

- **timezone**: Prototype in `<time.h>`. This long int contains the number of seconds of time that the local time zone is earlier than Coordinated Universal Time (UTC) (formerly known as Greenwich Mean Time (GMT)). Whenever a time function is called, the `tzset` function is called to set the value of the variable. The value will be determined from the value of the TZ environment variable.

- **tzname**: Prototype in `<time.h>`. This array of two pointers to character strings indicates the name of the standard abbreviation for the time zone and the name of the abbreviation for the time zone when daylight saving time is in effect. Whenever a time function is called, the `tzset` function is called to set the values in the array. These values will be determined from the value of the TZ environment variable.

- **__wargc**: Prototype in `<stdlib.h>`. This int item contains the number of arguments passed to `wmain`.

- **__wargv**: Prototype in `<stdlib.h>`. This wchar_t ** item contains a pointer to a vector containing the actual arguments passed to `wmain`. 
_wenviron

Prototype in `<stdlib.h>`.
This `wchar_t ** __near` data item is a pointer to an array of wide-character pointers
to the wide-character equivalents of the environment strings.

__win_alloc_flags

Prototype in `<stdlib.h>`.
This `unsigned long int` variable contains the flags to be used when allocating
memory in Windows.

__win_realloc_flags

Prototype in `<stdlib.h>`.
This `unsigned long int` variable contains the flags to be used when reallocating
memory in Windows.

_winmajor

(Win32 only) Prototype in `<stdlib.h>`.
This `unsigned int` variable contains the operating system major version number for the
version of Windows executing on the computer. For example, the major version number of
the Daytona release of Windows NT is 3.

Note that the Win32 GetVersionEx function is the preferred method for obtaining
operating system version number information.

_winminor

(Win32 only) Prototype in `<stdlib.h>`.
This `unsigned int` variable contains the operating system minor version number for
the version of Windows executing on the computer. For example, the minor version
number of the Daytona release of Windows NT is 5.

Note that the Win32 GetVersionEx function is the preferred method for obtaining
operating system version number information.

_winver

(Win32 only) Prototype in `<stdlib.h>`.
This `unsigned int` variable contains the operating system version number for the
version of Windows executing on the computer. The low-order byte contains the minor
version number (see also _winminor). The next byte contains the major version number
(see also _winmajor). The high-order word contains no useful information.

Note that the Win32 GetVersionEx function is the preferred method for obtaining
operating system version number information.

### 1.4 The TZ Environment Variable

The TZ environment variable is used to establish the local time zone. The value of the variable is used by
various time functions to compute times relative to Coordinated Universal Time (UTC) (formerly known as
Greenwich Mean Time (GMT)).

The time on the computer should be set to the local time. Use the DOS `time` command and the DOS
`date` command if the time is not automatically maintained by the computer hardware.

The TZ environment variable can be set (before the program is executed) by using the DOS `set` command
as follows:

```
SET TZ=PST8PDT
```
or (during the program execution) by using the `setenv` or `putenv` library functions:

```c
setenv( "TZ", "PST8PDT", 1 );
putenv( "TZ=PST8PDT" );
```

The value of the variable can be obtained by using the `getenv` function:

```c
char *tzvalue;
...

tzvalue = getenv( "TZ" );
```

The `tzset` function processes the `TZ` environment variable and sets the global variables `daylight` (indicates if daylight saving time is supported in the locale), `timezone` (contains the number of seconds of time difference between the local time zone and Coordinated Universal Time (UTC)), and `tzname` (a vector of two pointers to character strings containing the standard and daylight time-zone names).

The value of the `TZ` environment variable should be set as follows (spaces are for clarity only):

```
std offset dst offset , rule
```

The expanded format is as follows:

```
stdoffset[dst[offset][,start[/time],end[/time]]]
```

### `std, dst`

Three or more letters that are the designation for the standard (`std`) or summer (`dst`) time zone. Only `std` is required. If `dst` is omitted, then summer time does not apply in this locale. Upper- and lowercase letters are allowed. Any characters except for a leading colon (`:`), digits, comma (`,`), minus (`-`), plus (`+`), and ASCII NUL (`\0`) are allowed.

### `offset`

Indicates the value one must add to the local time to arrive at Coordinated Universal Time (UTC). The `offset` has the form:

```
hh[:mm[:ss]]
```

The minutes (`mm`) and seconds (`ss`) are optional. The hour (`hh`) is required and may be a single digit. The `offset` following `std` is required. If no `offset` follows `dst`, summer time is assumed to be one hour ahead of standard time. One or more digits may be used; the value is always interpreted as a decimal number. The hour may be between 0 and 24, and the minutes (and seconds) - if present - between 0 and 59. If preceded by a `-`, the time zone will be east of the Prime Meridian; otherwise it will be west (which may be indicated by an optional preceding `+`).

### `rule`

Indicates when to change to and back from summer time. The `rule` has the form:

```
date/time,date/time
```

Where the first `date` describes when the change from standard to summer time occurs and the second `date` describes when the change back happens. Each `time` field describes when, in current local time, the change to the other time is made.

The format of `date` may be one of the following:
The Julian day \( n \) (1 \( \leq \) \( n \) \( \leq \) 365). Leap days are not counted. That is, in all years - including leap years - February 28 is day 59 and March 1 is day 60. It is impossible to explicitly refer to the occasional February 29.

\( n \) The zero-based Julian day (0 \( \leq \) \( n \) \( \leq \) 365). Leap years are counted, and it is possible to refer to February 29.

\( Mm.n.d \) The \( d \)'th day (0 \( \leq \) \( d \) \( \leq \) 6) of week \( n \) of month \( m \) of the year (1 \( \leq \) \( n \) \( \leq \) 5, 1 \( \leq \) \( m \) \( \leq \) 12, where week 5 means "the last \( d \) day in month \( m \)" which may occur in the fourth or fifth week). Week 1 is the first week in which the \( d \)'th day occurs. Day zero is Sunday.

The \textit{time} has the same format as \textit{offset} except that no leading sign ("+" or ")" is allowed. The default, if \textit{time} is omitted, is \texttt{02:00:00}.

Whenever \texttt{ctime}, \texttt{ctime}, \texttt{localtime}, \texttt{localtime} or \texttt{mktime} is called, the time zone names contained in the external variable \texttt{tzname} will be set as if the \texttt{tzset} function had been called. The same is true if the \%Z directive of \texttt{strftime} is used.

Some examples are:

\texttt{TZ=EST5EDT} Eastern Standard Time is 5 hours earlier than Coordinated Universal Time (UTC). Standard time and daylight saving time both apply to this locale. By default, Eastern Daylight Time (EDT) is one hour ahead of standard time (i.e., EDT4). Since it is not specified, daylight saving time starts on the first Sunday of April at 2:00 A.M. and ends on the last Sunday of October at 2:00 A.M. This is the default when the \texttt{TZ} variable is not set.

\texttt{TZ=EST5EDT4,M4.1.0/02:00:00,M10.5.0/02:00:00} This is the full specification for the default when the \texttt{TZ} variable is not set. Eastern Standard Time is 5 hours earlier than Coordinated Universal Time (UTC). Standard time and daylight saving time both apply to this locale. Eastern Daylight Time (EDT) is one hour ahead of standard time. Daylight saving time starts on the first (1) Sunday (0) of April (4) at 2:00 A.M. and ends on the last (5) Sunday (0) of October (10) at 2:00 A.M.

\texttt{TZ=PST8PDT} Pacific Standard Time is 8 hours earlier than Coordinated Universal Time (UTC). Standard time and daylight saving time both apply to this locale. By default, Pacific Daylight Time is one hour ahead of standard time (i.e., PDT7). Since it is not specified, daylight saving time starts on the first Sunday of April at 2:00 A.M. and ends on the last Sunday of October at 2:00 A.M.

\texttt{TZ=NST3:30NDT1:30} Newfoundland Standard Time is 3 and 1/2 hours earlier than Coordinated Universal Time (UTC). Standard time and daylight saving time both apply to this locale. Newfoundland Daylight Time is 1 and 1/2 hours earlier than Coordinated Universal Time (UTC).

\texttt{TZ=Central Europe Time-2:00} Central European Time is 2 hours later than Coordinated Universal Time (UTC). Daylight saving time does not apply in this locale.
2 Graphics Library

The Watcom C Graphics Library consists of a large number of functions that provide graphical image support under DOS and QNX. This chapter provides an overview of this support. The following topics are discussed.

- Graphics Functions
- Graphics Adapters
- Classes of Graphics Functions
  1. Environment Functions
  2. Coordinate System Functions
  3. Attribute Functions
  4. Drawing Functions
  5. Text Functions
  6. Graphics Text Functions
  7. Image Manipulation Functions
  8. Font Manipulation Functions
  9. Presentation Graphics Functions
    - Display Functions
    - Analyze Functions
    - Utility Functions
- Graphics Header Files

2.1 Graphics Functions

Graphics functions are used to display graphical images such as lines and circles upon the computer screen. Functions are also provided for displaying text along with the graphics output.

2.2 Graphics Adapters

Support is provided for both color and monochrome screens which are connected to the computer using any of the following graphics adapters:

- IBM Monochrome Display/Printer Adapter (MDPA)
- IBM Color Graphics Adapter (CGA)
- IBM Enhanced Graphics Adapter (EGA)
- IBM Multi-Color Graphics Array (MCGA)
Watcom C Library Reference

- IBM Video Graphics Array (VGA)
- Hercules Monochrome Adapter
- SuperVGA adapters (SVGA) supplied by various manufacturers

2.3 Classes of Graphics Functions

The functions in the Watcom C Graphics Library can be organized into a number of classes:

*Environment Functions*
These functions deal with the hardware environment.

*Coordinate System Functions*
These functions deal with coordinate systems and mapping coordinates from one system to another.

*Attribute Functions*
These functions control the display of graphical images.

*Drawing Functions*
These functions display graphical images such as lines and ellipses.

*Text Functions*
These functions deal with displaying text in both graphics and text modes.

*Graphics Text Functions*
These functions deal with displaying graphics text.

*Image Manipulation Functions*
These functions store and retrieve screen images.

*Font Manipulation Functions*
These functions deal with displaying font based text.

*Presentation Graphics Functions*
These functions deal with displaying presentation graphics elements such as bar charts and pie charts.

The following subsections describe these function classes in more detail. Each function in the class is noted with a brief description of its purpose.

2.3.1 Environment Functions

These functions deal with the hardware environment. The `_getvideoconfig` function returns information about the current video mode and the hardware configuration. The `_setvideomode` function selects a new video mode.

Some video modes support multiple pages of screen memory. The visual page (the one displayed on the screen) may be different than the active page (the one to which objects are being written).
The following functions are defined:

- `_getactivepage` get the number of the current active graphics page
- `_getvideoconfig` get information about the graphics configuration
- `_getvisualpage` get the number of the current visual graphics page
- `_grstatus` get the status of the most recently called graphics library function
- `_setactivepage` set the active graphics page (the page to which graphics objects are drawn)
- `_settextrows` set the number of rows of text displayed on the screen
- `_setvideomode` select the video mode to be used
- `_setvideomoderows` select the video mode and the number of text rows to be used
- `_setvisualpage` set the visual graphics page (the page displayed on the screen)

### 2.3.2 Coordinate System Functions

These functions deal with coordinate systems and mapping coordinates from one system to another. The Watcom C Graphics Library supports three coordinate systems:

1. Physical coordinates
2. View coordinates
3. Window coordinates

Physical coordinates match the physical dimensions of the screen. The physical origin, denoted \((0,0)\), is located at the top left corner of the screen. A pixel to the right of the origin has a positive x-coordinate and a pixel below the origin will have a positive y-coordinate. The x- and y-coordinates will never be negative values.

The view coordinate system can be defined upon the physical coordinate system by moving the origin from the top left corner of the screen to any physical coordinate (see the `_setvieworg` function). In the view coordinate system, negative x- and y-coordinates are allowed. The scale of the view and physical coordinate systems is identical (both are in terms of pixels).

The window coordinate system is defined in terms of a range of user-specified values (see the `_setwindow` function). These values are scaled to map onto the physical coordinates of the screen. This allows for consistent pictures regardless of the resolution (number of pixels) of the screen.

The following functions are defined:

- `_getcliprgn` get the boundary of the current clipping region
- `_getphyscoord` get the physical coordinates of a point in view coordinates
- `_getviewcoord` get the view coordinates of a point in physical coordinates
- `_getviewcoord_w` get the view coordinates of a point in window coordinates
- `_getviewcoord_wxy` get the view coordinates of a point in window coordinates
- `_getwindowcoord` get the window coordinates of a point in view coordinates
- `_setcliprgn` set the boundary of the clipping region
- `_setvieworg` set the position to be used as the origin of the view coordinate system
- `_setviewport` set the boundary of the clipping region and the origin of the view coordinate system
- `_setwindow` define the boundary of the window coordinate system
2.3.3 Attribute Functions

These functions control the display of graphical images such as lines and circles. Lines and figures are drawn using the current color (see the \_setcolor function), the current line style (see the \_setlinestyle function), the current fill mask (see the \_setfillmask function), and the current plotting action (see the \_setplotaction function).

The following functions are defined:

- \_getarcinfo: get the endpoints of the most recently drawn arc
- \_getbkcolor: get the background color
- \_getcolor: get the current color
- \_getfillmask: get the current fill mask
- \_getlinestyle: get the current line style
- \_getplotaction: get the current plotting action
- \_remapallpalette: assign colors for all pixel values
- \_remappalette: assign color for one pixel value
- \_selectpalette: select a palette
- \_setbkcolor: set the background color
- \_setcolor: set the current color
- \_setfillmask: set the current fill mask
- \_setlinestyle: set the current line style
- \_setplotaction: set the current plotting action

2.3.4 Drawing Functions

These functions display graphical images such as lines and ellipses. Functions exist to draw straight lines (see the \_lineto functions), rectangles (see the \_rectangle functions), polygons (see the \_polygon functions), ellipses (see the \_ellipse functions), elliptical arcs (see the \_arc functions) and pie-shaped wedges from ellipses (see the \_pie functions).

These figures are drawn using the attributes described in the previous section. The functions ending with \_w or \_wxy use the window coordinate system; the others use the view coordinate system.

The following functions are defined:

- \_arc: draw an arc
- \_arc_w: draw an arc using window coordinates
- \_arc_wxy: draw an arc using window coordinates
- \_clearscreen: clear the screen and fill with the background color
- \_ellipse: draw an ellipse
- \_ellipse_w: draw an ellipse using window coordinates
- \_ellipse_wxy: draw an ellipse using window coordinates
- \_floodfill: fill an area of the screen with the current color
- \_floodfill_w: fill an area of the screen in window coordinates with the current color
- \_getcurrentposition: get the coordinates of the current output position
- \_getcurrentposition_w: get the window coordinates of the current output position
- \_getpixel: get the color of the pixel at the specified position
- \_getpixel_w: get the color of the pixel at the specified position in window coordinates
- \_lineto: draw a line from the current position to a specified position
Graphics Library

### _lineto_w
- Draw a line from the current position to a specified position in window coordinates.

### _moveto
- Set the current output position.

### _moveto_w
- Set the current output position using window coordinates.

### _pie
- Draw a wedge of a "pie".

### _pie_w
- Draw a wedge of a "pie" using window coordinates.

### _pie_wxy
- Draw a wedge of a "pie" using window coordinates.

### _polygon
- Draw a polygon.

### _polygon_w
- Draw a polygon using window coordinates.

### _polygon_wxy
- Draw a polygon using window coordinates.

### _rectangle
- Draw a rectangle.

### _rectangle_w
- Draw a rectangle using window coordinates.

### _rectangle_wxy
- Draw a rectangle using window coordinates.

### _setpixel
- Set the color of the pixel at the specified position.

### _setpixel_w
- Set the color of the pixel at the specified position in window coordinates.

### 2.3.5 Text Functions

These functions deal with displaying text in both graphics and text modes. This type of text output can be displayed in only one size.

This text is displayed using the _outtext and _outmem functions. The output position for text follows the last text that was displayed or can be reset (see the _settextposition function). Text windows can be created (see the _settextwindow function) in which the text will scroll. Text is displayed with the current text color (see the _settextcolor function).

The following functions are defined:

- _clearscreen: clear the screen and fill with the background color.
- _displaycursor: determine whether the cursor is to be displayed after a graphics function completes execution.
- _getbkcolor: get the background color.
- _gettextcolor: get the color used to display text.
- _gettextcursor: get the shape of the text cursor.
- _gettextposition: get the current output position for text.
- _gettextwindow: get the boundary of the current text window.
- _outmem: display a text string of a specified length.
- _outtext: display a text string.
- _scrolltextwindow: scroll the contents of the text window.
- _setbkcolor: set the background color.
- _settextcolor: set the color used to display text.
- _settextcursor: set the shape of the text cursor.
- _settextposition: set the output position for text.
- _settextwindow: set the boundary of the region used to display text.
- _wrapon: permit or disallow wrap-around of text in a text window.

### 2.3.6 Graphics Text Functions

These functions deal with displaying graphics text. Graphics text is displayed as a sequence of line segments, and can be drawn in different sizes (see the _setcharsize function), with different orientations (see the _settextorient function) and alignments (see the _settextalign function).
The functions ending with _w use the window coordinate system; the others use the view coordinate system.

The following functions are defined:

- `gettextextent`: get the bounding rectangle for a graphics text string
- `gettextsettings`: get information about the current settings used to display graphics text
- `grtext`: display graphics text
- `grtext_w`: display graphics text using window coordinates
- `setcharsize`: set the character size used to display graphics text
- `setcharsize_w`: set the character size in window coordinates used to display graphics text
- `setcharspacing`: set the character spacing used to display graphics text
- `setcharspacing_w`: set the character spacing in window coordinates used to display graphics text
- `settextalign`: set the alignment used to display graphics text
- `settextorient`: set the orientation used to display graphics text
- `settextpath`: set the path used to display graphics text

### 2.3.7 Image Manipulation Functions

These functions are used to transfer screen images. The `getimage` function transfers a rectangular image from the screen into memory. The `putimage` function transfers an image from memory back onto the screen. The functions ending with _w or _wxy use the window coordinate system; the others use the view coordinate system.

The following functions are defined:

- `getimage`: store an image of an area of the screen into memory
- `getimage_w`: store an image of an area of the screen in window coordinates into memory
- `getimage_wxy`: store an image of an area of the screen in window coordinates into memory
- `imagesize`: get the size of a screen area
- `imagesize_w`: get the size of a screen area in window coordinates
- `imagesize_wxy`: get the size of a screen area in window coordinates
- `putimage`: display an image from memory on the screen
- `putimage_w`: display an image from memory on the screen using window coordinates

### 2.3.8 Font Manipulation Functions

These functions are for the display of fonts compatible with Microsoft Windows. Fonts are contained in files with an extension of `.FON`. Before font based text can be displayed, the fonts must be registered with the `registerfonts` function, and a font must be selected with the `setfont` function.
The following functions are defined:

- `_getfontinfo` get information about the currently selected font
- `_getgtextextent` get the length in pixels of a text string
- `_getgtextvector` get the current value of the font text orientation vector
- `_outgtext` display a string of text in the current font
- `_registerfonts` initialize the font graphics system
- `_setfont` select a font from among the registered fonts
- `_setgtextvector` set the font text orientation vector
- `_unregisterfonts` frees memory allocated by the font graphics system

### 2.3.9 Presentation Graphics Functions

These functions provide a system for displaying and manipulating presentation graphics elements such as bar charts and pie charts. The presentation graphics functions can be further divided into three classes:

**Display Functions**
These functions are for the initialization of the presentation graphics system and the displaying of charts.

**Analyze Functions**
These functions calculate default values for chart elements without actually displaying the chart.

**Utility Functions**
These functions provide additional support to control the appearance of presentation graphics elements.

The following subsections describe these function classes in more detail. Each function in the class is noted with a brief description of its purpose.

#### 2.3.9.1 Display Functions

These functions are for the initialization of the presentation graphics system and the displaying of charts. The `_pg_initchart` function initializes the system and should be the first presentation graphics function called. The single-series functions display a single set of data on a chart; the multi-series functions (those ending with `ms`) display several sets of data on the same chart.

The following functions are defined:

- `_pg_chart` display a bar, column or line chart
- `_pg_chartms` display a multi-series bar, column or line chart
- `_pg_chartpie` display a pie chart
- `_pg_chartscatter` display a scatter chart
- `_pg_chartscatterms` display a multi-series scatter chart
- `_pg_defaultchart` initialize the chart environment for a specific chart type
- `_pg_initchart` initialize the presentation graphics system
2.3.9.2 Analyze Functions

These functions calculate default values for chart elements without actually displaying the chart. The functions ending with ms analyze multi-series charts; the others analyze single-series charts.

The following functions are defined:

- `_pg_analyzechart` analyze a bar, column or line chart
- `_pg_analyzechartms` analyze a multi-series bar, column or line chart
- `_pg_analyzepie` analyze a pie chart
- `_pg_analyzescatter` analyze a scatter chart
- `_pg_analyzescatterms` analyze a multi-series scatter chart

2.3.9.3 Utility Functions

These functions provide additional support to control the appearance of presentation graphics elements.

The following functions are defined:

- `_pg_getchardef` get bit-map definition for a specific character
- `_pg_getpalette` get presentation graphics palette (colors, line styles, fill patterns and plot characters)
- `_pg_getstyleset` get presentation graphics style-set (line styles for window borders and grid lines)
- `_pg_hlabelchart` display text horizontally on a chart
- `_pg_resetpalette` reset presentation graphics palette to default values
- `_pg_resetstyleset` reset presentation graphics style-set to default values
- `_pg_setchardef` set bit-map definition for a specific character
- `_pg_setpalette` set presentation graphics palette (colors, line styles, fill patterns and plot characters)
- `_pg_setstyleset` set presentation graphics style-set (line styles for window borders and grid lines)
- `_pg_vlabelchart` display text vertically on a chart

2.4 Graphics Header Files

All program modules which use the Graphics Library should include the header file `graph.h`. This file contains prototypes for all the functions in the library as well as the structures and constants used by them.

Modules using the presentation graphics functions should also include the header file `pgchart.h`. 
3 DOS Considerations

For the most part, DOS (Disk Operating System) for your personal computer can be ignored, unless an application is highly dependent upon the hardware or uses specialized functions from the operating system. In this section, some of these aspects will be addressed. For a more detailed explanation, the technical documentation for the DOS that you are using should be consulted.

3.1 DOS Devices

Most of the hardware devices attached to your computer have names which are recognized by DOS. These names cannot be used as the names of files. Some examples are:

- **CON**: the console (screen)
- **AUX**: the serial (auxiliary) port
- **COM1**: serial port 1
- **COM2**: serial port 2
- **PRN**: the printer on the parallel port
- **LPT1**: the printer on the first parallel port
- **LPT2**: the printer on the second parallel port
- **LPT3**: the printer on the third parallel port
- **NUL**: a non-existent device, which accepts (and discards) output

Disks (such as diskette drives and hard disks) are specified as single letters, starting with the letter A. A colon character (:) follows the letter for the drive. Either uppercase or lowercase letters can be used. Some examples are:

- **A:** the first disk drive
- **a:** the first disk drive
- **e:** the fifth disk drive

3.2 DOS Directories

Each disk drive is conceptually divided into directories. Each directory is capable of containing files and/or other directories. The initial directory, called the **root directory**, is not named; all other directories are named and can be accessed with a **path** specification. A path is either absolute or relative to the current working directory. Some examples are:

- **b:** the root directory of the second disk drive
- **\**: the root directory of the current disk drive

\outer\middle\inner
directory **inner** which is contained within directory **middle** which is contained within directory **outer** which is contained within the root directory of the current disk drive.
Directory names are separated by backslash characters (\). The initial backslash character informs DOS that the path starts with the root directory. When the first character is not a backslash, the path starts with the current working directory on the indicated device.

The DOS CHDIR (CD) command can be used to change the current working directory for a device. Suppose that the following DOS commands were issued:

```bash
chdir a:\apps\payroll
chdir c:\mydir
```

Then, the following path specifications are:

<table>
<thead>
<tr>
<th>Relative Path</th>
<th>Absolute Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>a:xxx\y</td>
<td>a:\apps\payroll\xxx\y</td>
</tr>
<tr>
<td>c:zzzz</td>
<td>c:\mydir\zzzz</td>
</tr>
</tbody>
</table>

When no drive is specified, DOS uses the current disk drive.

### 3.3 DOS File Names

The name of a file within a directory has the format `filename.ext` where the required `filename` portion is up to eight characters in length and the optional `ext` portion is up to three characters in length. A period character (.) separates the two names when the `ext` portion is present.

More than eight characters can be given in the `filename`. DOS truncates the name to eight characters when a longer `filename` is given. This may lead to erroneous results in some cases, since the files `MYBIGDATAFILE` and `MYBIGDATES` both refer to the file `MYBIGDAT`.

The characters used in file names may be letters, digits as well as some other characters documented in your DOS technical documentation. Most people restrict their file names to contain only letters and digits. Uppercase and lowercase letters are treated as being equivalent (file names are case insensitive). Thus, the files

```plaintext
MYDATA.NEW
mydata.new
MyData.New
```

all refer to the same file.

You cannot use a DOS device name (such as CON or PRN, for example) for a file name. See the section `DOS Devices` for a list of these reserved names.

A complete file designation has the following format:

```plaintext
drive:\path\filename.ext
```

where:
**DOS Considerations**

`drive:` is an optional disk drive specification. If omitted, the default drive is used. Some examples are:

- **A**: (first disk drive)
- **C**: (third disk drive)

`\path\` is the path specification for the directory containing the desired file. Some examples are:

```
\mylib\apps\payroll\n```

`filename.ext` is the name of the file.

Suppose that the current working directories are as follows:

<table>
<thead>
<tr>
<th>Drive</th>
<th>Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>\payroll</td>
</tr>
<tr>
<td>B</td>
<td>\</td>
</tr>
<tr>
<td>C</td>
<td>\source\c</td>
</tr>
</tbody>
</table>

and that the default disk drive is `C:`. Then, the following file designations will result in the indicated file references:

<table>
<thead>
<tr>
<th>Designation</th>
<th>Actual File</th>
</tr>
</thead>
<tbody>
<tr>
<td>pgm.c</td>
<td>C:\SOURCE\C\PGM.C</td>
</tr>
<tr>
<td>basic.dat</td>
<td>C:\BASIC.DAT</td>
</tr>
<tr>
<td>paypgm\outsep.c</td>
<td>C:\SOURCE\C\PAYPGM\OUTSEP.C</td>
</tr>
<tr>
<td>b: DATA</td>
<td>B:\DATA</td>
</tr>
<tr>
<td>a: employee</td>
<td>A:\PAYROLL\EMPLOYEE</td>
</tr>
<tr>
<td>a:deduct\yr1988</td>
<td>A:\DEDUCT\YR1988</td>
</tr>
</tbody>
</table>

### 3.4 DOS Files

DOS files are stored within directories on disk drives. Most software, including Watcom C/C++, treats files in two representations:

**BINARY**

These files can contain arbitrary data. It is the responsibility of the software to recognize records within the file if they exist.

**TEXT**

These files contain lines of "printable" characters. Each line is delimited by a carriage return character followed by a linefeed character.

Since the conceptual view of text files in the C and C++ languages is that lines are terminated by only linefeed characters, the Watcom C library will remove carriage returns on input and add them on output, provided the mode is set to be `text`. This mode is set upon opening the file or with the `setmode` function.
3.5 DOS Commands

DOS commands are documented in the technical documentation for your DOS system. These may be invoked from a C or C++ program with the `system` function.

3.6 DOS Interrupts

DOS interrupts and 8086 interrupts are documented in the technical documentation for your DOS system. These may be generated from a C or C++ program by calling the `bdos`, `intdos`, `intdosx`, `intr`, `int386`, `int386x`, `int86` and `int86x` functions.

3.7 DOS Processes

Currently, DOS has the capability to execute only one process at a time. Thus, when a process is initiated with the `spawn...` parameter `P_WAIT`, the new process will execute to completion before control returns to the initiating program. Otherwise, the new task replaces the initial task. Tasks can be started by using the `system`, `exec...` and `spawn...` functions.
Each of the functions or macros in the C Library is described in this chapter. Each description consists of a number of subsections:

**Synopsis:** This subsection gives the header files that should be included within a source file that references the function or macro. It also shows an appropriate declaration for the function or for a function that could be substituted for a macro. This declaration is not included in your program; only the header file(s) should be included.

When a pointer argument is passed to a function and that function does not modify the item indicated by that pointer, the argument is shown with `const` before the argument. For example,

```c
const char *string
```
indicates that the array pointed at by `string` is not changed.

**Constraints:** This subsection describes Runtime-constraints for Safer C Library functions.

**Safer C:** This subsection points to the Safer C version of the described "unsafe" function.

**Description:** This subsection is a description of the function or macro.

**Returns:** This subsection describes the return value (if any) for the function or macro.

**Errors:** This subsection describes the possible `errno` values.

**See Also:** This optional subsection provides a list of related functions or macros.

**Example:** This optional subsection consists of one or more examples of the use of the function. The examples are often just fragments of code (not complete programs) for illustration purposes.

**Classification:** This subsection provides an indication of where the function or macro is commonly found. The following notation is used:

- **ANSI** These functions or macros are defined by the ANSI/ISO C standard.
- **POSIX 1003.1** These functions or macros are not defined by the ANSI/ISO C standard. These function are specified in the document *IEEE Standard Portable Operating System Interface for Computer Environments* (IEEE Draft Standard 1003.1-1990).
- **BIOS** These functions access a service of the BIOS found in IBM Personal Computers and compatibles. These functions should not be used if portability is a consideration.
- **DOS** These functions or macros are neither ANSI/ISO nor POSIX. They perform a function related to DOS. They may be found in other implementations of C for personal computers with DOS. Use these functions with caution, if portability is a consideration.
Intel

These functions or macros are neither ANSI/ISO nor POSIX. They perform a function related to the Intel x86 architecture. They may be found in other implementations of C for personal computers using Intel chips. Use these functions with caution, if portability is a consideration.

OS/2

These functions are specific to OS/2.

PC Graphics

These functions are part of the PC graphics library.

Windows

These functions are specific to Microsoft Windows.

WATCOM

These functions or macros are neither ANSI/ISO nor POSIX. They may be found in other implementations of the C language, but caution should be used if portability is a consideration.

TR 24731

These functions are "safer" versions of normal C library functions. They perform more checks on parameters and should be used in preference over their "unsafe" version.

Systems:

This subsection provides an indication of where the function or macro is supported. The following notation is used:

All

This function is available on all systems (we do not include Netware or DOS/PM in this category).

DOS

This function is available on both 16-bit DOS and 32-bit extended DOS.

DOS/16

This function is available on 16-bit, real-mode DOS.

DOS/32

This function is available on 32-bit, protected-mode extended DOS.

DOS/PM

This 16-bit DOS protected-mode function is supported under Phar Lap’s 286|DOS-Extender "RUN286". The function is found in one of Watcom’s 16-bit protected-mode DOS libraries (DOSPM*.LIB under the 16-bit OS2 subdirectory).

MACRO

This function is implemented as a macro (#define) on all systems.

Math

This function is a math function. Math functions are available on all systems.

Netware

This function is available on the 32-bit Novell Netware operating system.

OS/2 1.x

This function is available on IBM OS/2 1.x, a 16-bit protected-mode system for Intel 80286 and upwards compatible systems.

When "(MT)" appears after OS/2, it refers to the CLIBMTL library which supports multi-threaded applications.

When "(DL)" appears after OS/2, it refers to the CLIBDLL library which supports creation of Dynamic Link Libraries.

When "(all)" appears after "OS/2 1", it means all versions of the OS/2 1.x libraries.
If a function is missing from the OS/2 library, it may be found in Watcom’s 16-bit protected-mode DOS libraries (DOSPM*.LIB) for Phar Lap’s 286|DOS-Extender (RUN286).

- **OS/2-32**
  This function is available on 32-bit IBM OS/2, a protected-mode system for Intel 80386 and upwards compatible systems.

- **QNX**
  This function is available on QNX Software Systems’ 16 or 32-bit operating systems.

- **QNX/16**
  This function is available on QNX Software Systems’ 16-bit operating system.

- **QNX/32**
  This function is available on QNX Software Systems’ 32-bit operating system.

- **Windows**
  This function is available on 16-bit, protected-mode Windows 3.x.

- **Win386**
  This function is available on Microsoft Windows 3.x, using Watcom’s Windows Extender for 32-bit protected-mode applications running on Intel 386 or upward compatible systems.

- **Win32**
  This function is available on 32-bit Microsoft Windows platforms (Windows 95, Windows 98, Windows NT, Windows 2000, etc.). It may also be available for Windows 3.x using Win32s support.
Synopsis:  
```
#include <stdlib.h>
void abort( void );
```

Description:  The `abort` function raises the signal SIGABRT. The default action for SIGABRT is to terminate program execution, returning control to the process that started the calling program (usually the operating system). The status *unsuccessful termination* is returned to the invoking process by means of the function call `raise(SIGABRT)`. The exit code returned to the invoking process is `EXIT_FAILURE` which is defined in the `<stdlib.h>` header file.

Returns:  The `abort` function does not return to its caller.

See Also:  `atexit`, `_bgetcmd`, `exec...`, `exit`, `_Exit`, `_exit`, `getcmd`, `getenv`, `main`, `onexit`, `putenv`, `spawn...`, `system`

Example:  
```
#include <stdlib.h>

void main()
{
    int major_error = 1;
    if( major_error )
        abort();
}
```

Classification:  ANSI

Systems:  All, Netware
Synopsis: #define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
void abort_handler_s(
    const char * restrict msg,
    void * restrict ptr,
    errno_t error );

Description: The abort_handler_s function may be passed as an argument to the
set_constraint_handler_s function. It writes a message on the standard error stream in the
following format:

Runtime-constraint violation: <msg>

The abort_handler_s function then calls the abort function.

Returns: The abort_handler_s function does not return to its caller.

See Also: ignore_handler_s, set_constraint_handler_s

Example: #define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
#include <stdio.h>

void main( void )
{
    constraint_handler_t old_handler;

    old_handler = set_constraint_handler_s( abort_handler_s );
    if( getenv_s( NULL, NULL, 0, NULL ) ) {
        printf( "getenv_s failed\n" );
    }
    set_constraint_handler_s( old_handler );
}

produces the following:

Runtime-constraint violation: getenv_s, name == NULL.
ABNORMAL TERMINATION

Classification: TR 24731

Systems: All, Netware
abs

Synopsis:  
#include <stdlib.h>
int abs( int j );

Description:  The abs function returns the absolute value of its integer argument j.

Returns:  The abs function returns the absolute value of its argument.

See Also:  labs, llabs, imaxabs, fabs

Example:  
#include <stdio.h>
#include <stdlib.h>

void main( void )
{
    printf( "%d %d %d\n", abs( -5 ), abs( 0 ), abs( 5 ) );
}

produces the following:
5 0 5

Classification:  ISO C90

Systems:  All, Netware
Synopsis: 

```c
#include <io.h>
int access( const char *path, int mode );
int _access( const char *path, int mode );
int _waccess( const wchar_t *path, int mode );
```

Description: The `access` function determines if the file or directory specified by `path` exists and if it can be accessed with the file permission given by `mode`. 

The `_access` function is identical to `access`. Use `_access` for ANSI naming conventions.

When the value of `mode` is zero, only the existence of the file is verified. The read and/or write permission for the file can be determined when `mode` is a combination of the bits:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_OK</td>
<td>test for read permission</td>
</tr>
<tr>
<td>W_OK</td>
<td>test for write permission</td>
</tr>
<tr>
<td>X_OK</td>
<td>test for execute permission</td>
</tr>
<tr>
<td>F_OK</td>
<td>test for existence of file</td>
</tr>
</tbody>
</table>

With DOS, all files have read permission; it is a good idea to test for read permission anyway, since a later version of DOS may support write-only files.

The `_waccess` function is identical to `access` except that it accepts a wide-character string argument for `path`.

Returns: The `access` function returns zero if the file or directory exists and can be accessed with the specified mode. Otherwise, -1 is returned and `errno` is set to indicate the error.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EACCES</td>
<td>Access denied because the file’s permission does not allow the specified access.</td>
</tr>
<tr>
<td>ENOENT</td>
<td>Path or file not found.</td>
</tr>
</tbody>
</table>

See Also: `chmod`, `fstat`, `open`, `sopen`, `stat`

Example:

```c
#include <stdio.h>
#include <stdlib.h>
#include <io.h>

void main( int argc, char *argv[] )
{
    if( argc != 2 ) {
        fprintf( stderr, "Use: check <filename>\n" );
        exit( 1 );
    }
}
```
if( access( argv[1], F_OK ) == 0 ) {
    printf( "%s exists\n", argv[1] );
} else {
    printf( "%s does not exist\n", argv[1] );
    exit( EXIT_FAILURE );
}
if( access( argv[1], R_OK ) == 0 ) {
    printf( "%s is readable\n", argv[1] );
}
if( access( argv[1], W_OK ) == 0 ) {
    printf( "%s is writeable\n", argv[1] );
}
if( access( argv[1], X_OK ) == 0 ) {
    printf( "%s is executable\n", argv[1] );
}
exit( EXIT_SUCCESS );

Classification: access is POSIX 1003.1
_access is not POSIX
_waccess is not POSIX

Systems: access - All, Netware
_access - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_waccess - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <math.h>  
double acos( double x );

Description:  The acos function computes the principal value of the arccosine of x. A domain error occurs for arguments not in the range [-1,1].

Returns:  The acos function returns the arccosine in the range [0,π]. When the argument is outside the permissible range, the matherr function is called. Unless the default matherr function is replaced, it will set the global variable errno to EDOM, and print a "DOMAIN error" diagnostic message using the stderr stream.

See Also:  asin, atan, atan2, matherr

Example:  
#include <stdio.h>  
#include <math.h>

gvoid main()
{
   printf( "%f\n", acos(.5) );
}

produces the following:

1.047197

Classification:  ANSI

Systems:  Math
Synopsis:  
#include <math.h>  
double acosh( double x );

Description:  The acosh function computes the inverse hyperbolic cosine of x. A domain error occurs if the value of x is less than 1.0.

Returns:  The acosh function returns the inverse hyperbolic cosine value. When the argument is outside the permissible range, the matherr function is called. Unless the default matherr function is replaced, it will set the global variable errno to EDOM, and print a "DOMAIN error" diagnostic message using the stderr stream.

See Also:  asinh, atanh, cosh, matherr

Example:  
#include <stdio.h>  
#include <math.h>  

void main()  
{  
    printf( "%f\n", acosh( 1.5 ) );
}

produces the following:

0.962424

Classification:  WATCOM

Systems:  Math
Synopsis: #include <malloc.h>
void *alloca( size_t size );

Description: The alloca function allocates space for an object of size bytes from the stack. The allocated space is automatically discarded when the current function exits. The alloca function should not be used in an expression that is an argument to a function.

Returns: The alloca function returns a pointer to the start of the allocated memory. The return value is NULL if there is insufficient stack space available.

See Also: calloc, malloc, stackavail

Example: #include <stdio.h>
#include <string.h>
#include <malloc.h>
FILE *open_err_file( char * );

void main()
{
    FILE *fp;

    fp = open_err_file( "alloca" );
    if( fp == NULL ) {
        printf( "Unable to open error file\n" );
    } else {
        fclose( fp );
    }
}

FILE *open_err_file( char *name )
{
    char *buffer;
    /* allocate temp buffer for file name */
    buffer = (char *) alloca( strlen(name) + 5 );
    if( buffer ) {
        sprintf( buffer, "%s.err", name );
        return( fopen( buffer, "w" ) );
    }
    return( (FILE *) NULL );
}

Classification: WATCOM

Systems: MACRO
_arc Functions

Synopsis:

```c
#include <graph.h>
short _FAR _arc( short x1, short y1,
                 short x2, short y2,
                 short x3, short y3,
                 short x4, short y4 );

short _FAR _arc_w( double x1, double y1,
                   double x2, double y2,
                   double x3, double y3,
                   double x4, double y4 );

short _FAR _arc_wxy( struct _wxycoord _FAR *p1,
                    struct _wxycoord _FAR *p2,
                    struct _wxycoord _FAR *p3,
                    struct _wxycoord _FAR *p4 );
```

Description: The _arc functions draw elliptical arcs. The _arc function uses the view coordinate system. The _arc_w and _arc_wxy functions use the window coordinate system.

The center of the arc is the center of the rectangle established by the points \((x1, y1)\) and \((x2, y2)\). The arc is a segment of the ellipse drawn within this bounding rectangle. The arc starts at the point on this ellipse that intersects the vector from the centre of the ellipse to the point \((x3, y3)\). The arc ends at the point on this ellipse that intersects the vector from the centre of the ellipse to the point \((x4, y4)\). The arc is drawn in a counter-clockwise direction with the current plot action using the current color and the current line style.

The following picture illustrates the way in which the bounding rectangle and the vectors specifying the start and end points are defined.

![Diagram](码头)

When the coordinates \((x1, y1)\) and \((x2, y2)\) establish a line or a point (this happens when one or more of the x-coordinates or y-coordinates are equal), nothing is drawn.
The current output position for graphics output is set to be the point at the end of the arc that was drawn.

**Returns:** The _arc functions return a non-zero value when the arc was successfully drawn; otherwise, zero is returned.

**See Also:** _ellipse, _pie, _rectangle, _getarcinfo, _setcolor, _setlinestyle, _setplotaction

**Example:**

```
#include <conio.h>
#include <graph.h>

main()
{
  _setvideomode(_VRES16COLOR);
  _arc( 120, 90, 520, 390, 500, 20, 450, 460);
  getch();
  _setvideomode(_DEFAULTMODE);
}
```

produces the following:

![Arc Example](image)

**Classification:** PC Graphics

**Systems:**
- _arc - DOS, QNX
- _arc_w - DOS, QNX
- _arc_wxy - DOS, QNX
asctime Functions

Synopsis:

```c
#include <time.h>

char * asctime( const struct tm *timeptr );  
char * asctime( const struct tm *timeptr, char *buf );
wchar_t * _wasctime( const struct tm *timeptr );
wchar_t * __wasctime( const struct tm *timeptr, wchar_t *buf );
```

```c
struct tm {
    int tm_sec;  /* seconds after the minute -- [0,61] */
    int tm_min;  /* minutes after the hour   -- [0,59] */
    int tm_hour; /* hours after midnight     -- [0,23] */
    int tm_mday; /* day of the month         -- [1,31] */
    int tm_mon;  /* months since January     -- [0,11] */
    int tm_year; /* years since 1900                   */
    int tm_wday; /* days since Sunday        -- [0,6]  */
    int tm_yday; /* days since January 1     -- [0,365]*/
    int tm_isdst; /* Daylight Savings Time flag */
};
```

Safer C: The Safer C Library extension provides the asctime_s function which is a safer alternative to asctime. This newer asctime_s function is recommended to be used instead of the traditional "unsafe" asctime function.

Description: The asctime functions convert the time information in the structure pointed to by timeptr into a string containing exactly 26 characters. This string has the form shown in the following example:

```text
Sat Mar 21 15:58:27 1987
```

All fields have a constant width. The new-line character ‘\n’ and the null character ‘\0’ occupy the last two positions of the string.

The ANSI function asctime places the result string in a static buffer that is re-used each time asctime or ctime is called. The non-ANSI function _asctime places the result string in the buffer pointed to by buf:

The _wasctime and __wasctime functions are identical to their asctime and _asctime counterparts except that they deal with wide-character strings.

Returns: The asctime functions return a pointer to the character string result.

See Also: asctime_s, clock, ctime Functions, ctime_s,difftime, gmtime, gmtime_s, localtime, localtime_s, mktime, strftime, time, tzset

Example: 

```c
#include <stdio.h>
#include <time.h>

void main()
{
    struct tm  time_of_day;
    time_t     ltime;
    auto char  buf[26];
```
time( &ltime );
__localtime( &ltime, &time_of_day );
printf( "Date and time is: %s\n",
    __asctime( &time_of_day, buf ) );
)

produces the following:

Date and time is: Sat Mar 21 15:58:27 1987

**Classification:** asctime is ANSI
- _asctime is not ANSI
- __wasctime is not ANSI
- __wasctime is not ANSI

**Systems:** asctime - All, Netware
- _asctime - All, Netware
- __wasctime - All
- __wasctime - All
Synopsis:  
```c
#define __STDC_WANT_LIB_EXT1__ 1
#include <time.h>
errno_t asctime_s( char * s,
  rsize_t maxsize,
const struct tm * timeptr);
errno_t _wasctime_s( wchar_t * s,
  rsize_t maxsize,
const struct tm * timeptr);
```

```c
struct tm {
  int tm_sec; /* seconds after the minute -- [0,61] */
  int tm_min; /* minutes after the hour -- [0,59] */
  int tm_hour; /* hours after midnight -- [0,23] */
  int tm_mday; /* day of the month -- [1,31] */
  int tm_mon; /* months since January -- [0,11] */
  int tm_year; /* years since 1900 */
  int tm_wday; /* days since Sunday -- [0,6] */
  int tm_yday; /* days since January 1 -- [0,365] */
  int tm_isdst; /* Daylight Savings Time flag */
};
```

Constraints:  
If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and asctime_s will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither s nor timeptr shall be a null pointer. maxsize shall not be less than 26 and shall not be greater than RSIZE_MAX. The broken-down time pointed to by timeptr shall be normalized. The calendar year represented by the broken-down time pointed to by timeptr shall not be less than calendar year 0 and shall not be greater than calendar year 9999. If there is a runtime-constraint violation, there is no attempt to convert the time, and s[0] is set to a null character if s is not a null pointer and maxsize is not zero and is not greater than RSIZE_MAX.

Description:  
The asctime_s function converts the normalized broken-down time in the structure pointed to by timeptr into a 26 character (including the null character) string in the form

```
Sun Sep 16 01:03:52 1973
```

The fields making up this string are (in order):

1. The name of the day of the week represented by timeptr->tm_wday using the following three character weekday names:
   Sun, Mon, Tue, Wed, Thu, Fri, and Sat.
2. The character space.
3. The name of the month represented by timeptr->tm_mon using the following three character month names:
4. The character space.
5. The value of timeptr->tm_mday as if printed using the fprintf format "%2d".
6. The character space.

7. The value of timeptr->tm_hour as if printed using the fprintf format "%.2d".

8. The character colon.

9. The value of timeptr->tm_min as if printed using the fprintf format "%.2d".

10. The character colon.

11. The value of timeptr->tm_sec as if printed using the fprintf format "%.2d".

12. The character space.

13. The value of timeptr->tm_year + 1900 as if printed using the fprintf format "%4d".

14. The character new line.

15. The null character.

The _wasctime_s function is a wide-character version of asctime_s that operates with wide-character strings.

Returns: The asctime_s function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

See Also: asctime Functions, clock, ctime Functions, ctime_s, difftime, gmtime, gmtime_s, localtime, localtime_s, mktime, strftime, time, tzset

Example:
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <time.h>

void main()
{
    struct tm time_of_day;
    time_t ltime;
    auto char buf[26];

    time( &ltime );
    _localtime( &ltime, &time_of_day );
    asctime_s( buf, sizeof( buf ), &time_of_day );
    printf( "Date and time is: %s\n", buf );
}

produces the following:

Date and time is: Mon Jan 30 11:32:45 2006

Classification: asctime_s is TR 24731
_wasctime_s is not TR 24731

Systems: asctime_s - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, Netware
asctime_s, _wasctime_s

_wasctime_s - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <math.h>  
double asin( double x );

Description:  
The asin function computes the principal value of the arcsine of x. A domain error occurs for arguments not in the range [-1,1].

Returns:  
The asin function returns the arcsine in the range [-π/2,π/2]. When the argument is outside the permissible range, the matherr function is called. Unless the default matherr function is replaced, it will set the global variable errno to EDOM, and print a "DOMAIN error" diagnostic message using the stderr stream.

See Also:  
acos, atan, atan2, matherr

Example:  
#include <stdio.h>  
#include <math.h>

void main()
{
    printf( "%f\n", asin(.5) );
}

produces the following:

0.523599

Classification:  ANSI

Systems:  Math


**Synopsis:**

```c
#include <math.h>
double asinh( double x );
```

**Description:**
The `asinh` function computes the inverse hyperbolic sine of `x`.

**Returns:**
The `asinh` function returns the inverse hyperbolic sine value.

**See Also:**
`acosh`, `atanh`, `sinh`, `matherr`

**Example:**
```c
#include <stdio.h>
#include <math.h>

void main()
{
    printf( "%f\n", asinh( 0.5 ) );
}
```

produces the following:

```
0.481212
```

**Classification:** WATCOM

**Systems:** Math
Synopsis: 
#include <assert.h>
void assert( int expression );

Description: The assert macro prints a diagnostic message upon the stderr stream and terminates the program if expression is false (0). The diagnostic message has the form

Assertion failed: expression, file filename, line linenumber

where filename is the name of the source file and linenumber is the line number of the assertion that failed in the source file. Filename and linenumber are the values of the preprocessing macros __FILE__ and __LINE__ respectively. No action is taken if expression is true (non-zero).

The assert macro is typically used during program development to identify program logic errors. The given expression should be chosen so that it is true when the program is functioning as intended. After the program has been debugged, the special "no debug" identifier NDEBUG can be used to remove assert calls from the program when it is re-compiled. If NDEBUG is defined (with any value) with a -d command line option or with a #define directive, the C preprocessor ignores all assert calls in the program source.

Returns: The assert macro does not return a value.

Example: 
#include <stdio.h>
#include <assert.h>

void process_string( char *string )
{
    /* use assert to check argument */
    assert( string != NULL );
    assert( *string != '0' );
    /* rest of code follows here */
}

void main()
{
    process_string( "hello" );
    process_string( "" );
}

Classification: ANSI

Systems: MACRO
Synopsis:  
#include <math.h>  
double atan( double x );

Description:  The atan function computes the principal value of the arctangent of x.

Returns:  The atan function returns the arctangent in the range (-\pi/2, \pi/2).

See Also:  acos, asin, atan2

Example:  
#include <stdio.h>  
#include <math.h>

void main()  
{  
    printf( "%f\n", atan(.5) );
}

produces the following:
0.463648

Classification:  ANSI

Systems:  Math
Synopsis:  
#include <math.h>  
double atan2( double y, double x );

Description:  The atan2 function computes the principal value of the arctangent of \( \frac{y}{x} \), using the signs of both arguments to determine the quadrant of the return value. A domain error occurs if both arguments are zero.

Returns:  The atan2 function returns the arctangent of \( \frac{y}{x} \), in the range \((-\pi, \pi)\). When the argument is outside the permissible range, the matherr function is called. Unless the default matherr function is replaced, it will set the global variable errno to EDOM, and print a "DOMAIN error" diagnostic message using the stderr stream.

See Also:  acos, asin, atan, matherr

Example:  
#include <stdio.h>  
#include <math.h>  

void main()  
{  
    printf("%.16f\n", atan2(.5, 1.));  
}  

produces the following:  
0.463648

Classification:  ANSI

Systems:  Math
Synopsis:  
#include <math.h>  
double atanh( double x );

Description:  The atanh function computes the inverse hyperbolic tangent of x. A domain error occurs if the value of x is outside the range (-1,1).

Returns:  The atanh function returns the inverse hyperbolic tangent value. When the argument is outside the permissible range, the matherr function is called. Unless the default matherr function is replaced, it will set the global variable errno to EDOM, and print a "DOMAIN error" diagnostic message using the stderr stream.

See Also:  acosh, asinh, matherr, tanh

Example:  
#include <stdio.h>  
#include <math.h>  

void main()  
{  
    printf( "%f\n", atanh( 0.5 ) );  
}

produces the following:

0.549306

Classification:  WATCOM

Systems:  Math
Synopsis: #include <stdlib.h>
int atexit( void (*func)(void) );

Description: The atexit function is passed the address of function *func* to be called when the program terminates normally. Successive calls to atexit create a list of functions that will be executed on a "last-in, first-out" basis. No more than 32 functions can be registered with the atexit function.

The functions have no parameters and do not return values.

Returns: The atexit function returns zero if the registration succeeds, non-zero if it fails.

See Also: abort, _exit, exit

Example: #include <stdio.h>
#include <stdlib.h>

void main()
{
    extern void func1(void), func2(void), func3(void);

    atexit( func1 );
    atexit( func2 );
    atexit( func3 );
    printf( "Do this first.\n" );
}

void func1(void) { printf( "last.\n" ); }

void func2(void) { printf( "this " ); }

void func3(void) { printf( "Do " ); }

produces the following:
Do this first.
Do this last.

Classification: ANSI

Systems: All, Netware
Synopsis:  
#include <stdlib.h>
double atof( const char *ptr );
double _wtof( const wchar_t *ptr );

Description:  
The _atof function converts the string pointed to by _ptr to _double representation. It is equivalent to

    atof( ptr, (char **)NULL )

The _wtof function is identical to _atof except that it accepts a wide-character string argument. It is equivalent to

    wcstod( ptr, (wchar_t **)NULL )

Returns:  
The _atof function returns the converted value. Zero is returned when the input string cannot be converted. In this case, _errno is not set. When an error has occurred, _errno contains a value indicating the type of error that has been detected.

See Also:  
sscanf, _strtof

Example:  
#include <stdlib.h>

    void main()
    {
        double x;

        x = atof( "3.1415926" );
    }

Classification:  
atof is ANSI
_wtof is not ANSI

Systems:  
atof - Math
_wtof - Math
Synopsis:  
#include <stdlib.h>  
int atoi( const char *ptr );  
int _wtoi( const wchar_t *ptr );

Description:  The `atoi` function converts the string pointed to by `ptr` to `int` representation.  

The `_wtoi` function is identical to `atoi` except that it accepts a wide-character string argument.

Returns:  The `atoi` function returns the converted value.

See Also:  `atol`, `atoll`, `itoa`, `ltoa`, `lltoa`, `sscanf`, `strtol`, `strtoll`, `strtoul`, `strtoull`, `strtoimax`, `strtoimax`, `ultoa`, `ulltoa`, `utoa`

Example:  
#include <stdlib.h>  

void main()  
{  
  int x;  

  x = atoi( "-289" );  
}

Classification:  `atoi` is ANSI  
`_wtoi` is not ANSI

Systems:  `atoi` - All, Netware  
`_wtoi` - All
**Synopsis:**

```c
#include <stdlib.h>
long int atol( const char *ptr );
long int _wtol( const wchar_t *ptr );
```

**Description:**

The *atol* function converts the string pointed to by `ptr` to `long int` representation.

The *_wtol* function is identical to *atol* except that it accepts a wide-character string argument.

**Returns:**

The *atol* function returns the converted value.

**See Also:**

`atoi`, `atoll`, `itoa`, `ltoa`, `l1toa`, `sscanf`, `strtol`, `strtoll`, `strtoul`, `strtoull`, `strtoimax`, `strtoumax`, `ultoa`, `ulltoa`, `utoa`

**Example:**

```c
#include <stdlib.h>

void main()
{
    long int x;
    x = atol( "-289" );
}
```

**Classification:**

- atol is ANSI
- _wtol is not ANSI

**Systems:**

- atol - All, Netware
- _wtol - All
Synopsis:
#include <stdlib.h>
long long int atoll( const char *ptr );
long long int _wtoll( const wchar_t *ptr );

Description: The atoll function converts the string pointed to by ptr to long long int representation.
The _wtoll function is identical to atoll except that it accepts a wide-character string argument.

Returns: The atoll function returns the converted value.

See Also: atoi, atol, itoa, ltoa, lltoa, sscanf, strtol, strtoll, strtohl, strtoimax, strtooumax, ultoa, ulltoa, utoa

Example: #include <stdlib.h>

void main()
{
    long int x;

    x = atoll( "-289356768201" );
}

Classification: atoll is ANSI
                _wtoll is not ANSI

Systems: atoll - All, Netware
            _wtoll - All
_atouni

Synopsis:  
#include <stdlib.h>  
wchar_t *_atouni( wchar_t *wcs, const char *sbcsp );

Description:  
The _atouni function converts the string pointed to by sbcsp to a wide-character string and places it in the buffer pointed to by wcs.

The conversion ends at the first null character.

Returns:  
The _atouni function returns the first argument as a result.

See Also:  
atoi, atol, itoa, ltoa, strtof, strtol, strtoul, ultoa, utoa

Example:  
#include <stdlib.h>

void main()
{
    wchar_t wcs[12];
    _atouni( wcs, "Hello world" );
}

Classification:  
WATCOM

Systems:  
All, Netware
Synopsis:  
#include <libgen.h>  
char *basename( char *path );

Description:  The basename function returns a pointer to the final component of a pathname pointed to by the path argument, deleting trailing path separators.

If the string pointed to by path consists entirely of path separators, a string consisting of single path separator is returned.

If path is a null pointer or points to an empty string, a pointer to the string "." is returned.

The basename function may modify the string pointed to by path and may return a pointer to static storage that may be overwritten by a subsequent call to basename.

The basename function is not re-entrant or thread-safe.

Returns:  The basename function returns a pointer to the final component of path.

See Also:  dirname

Example:  
#include <stdio.h>  
#include <libgen.h>

int main( void )
{
    puts( basename( "/usr/lib" ) );
    puts( basename( "//usr//lib//" ) );
    puts( basename( "/" ) );
    puts( basename( "foo" ) );
    puts( basename( NULL ) );
    return( 0 );
}

produces the following:

lib
lib
/
foo
.

Classification:  POSIX

Systems:  All, Netware
Synopsis:  
```c
#include <dos.h>
int bdos( int dos_func, unsigned dx, unsigned char al );
```

Description:  The `bdos` function causes the computer’s central processor (CPU) to be interrupted with an interrupt number hexadecimal 21 (0x21), which is a request to invoke a specific DOS function. Before the interrupt, the DX register is loaded from `dx`, the AH register is loaded with the DOS function number from `dos_func` and the AL register is loaded from `al`. The remaining registers are passed unchanged to DOS.

You should consult the technical documentation for the DOS operating system you are using to determine the expected register contents before and after the interrupt in question.

Returns:  The `bdos` function returns the value of the AX register after the interrupt has completed.

See Also:  `int386`, `int386x`, `int86`, `int86x`, `intdos`, `intdosx`, `intr`, `segread`

Example:  
```c
#include <dos.h>
#define DISPLAY_OUTPUT  2

void main()
{
    int rc;

    rc = bdos( DISPLAY_OUTPUT, 'B', 0 );
    rc = bdos( DISPLAY_OUTPUT, 'D', 0 );
    rc = bdos( DISPLAY_OUTPUT, 'O', 0 );
    rc = bdos( DISPLAY_OUTPUT, 'S', 0 );
}
```

Classification:  DOS

Systems:  DOS, Windows, Win386, DOS/PM
Synopsis:
#include <process.h>
#if defined(__386__)
# define FAR
#else
# define FAR __far
#endif

#if defined(__NT__)
unsigned long _beginthread(
    void (*start_address)(void *),
    unsigned stack_size,
    void *arglist);
unsigned long _beginthreadex(
    void *security,
    unsigned stack_size,
    unsigned (__stdcall *start_address)(void *),
    void *arglist,
    unsigned initflag,
    unsigned *thrdid);
#else
int FAR _beginthread(
    void (FAR *start_address)(void FAR *),
    void FAR *stack_bottom,
    unsigned stack_size,
    void FAR *arglist);
#endif

Description: The _beginthread function is used to start a new thread of execution at the function identified by start_address with a single parameter identified by arglist.

For each operating environment under which _beginthread is supported, the _beginthread function uses the appropriate system call to begin a new thread of execution.

The new thread will use the memory identified by stack_bottom and stack_size for its stack.

Note for 16-bit applications: If the stack is not in DGROUP (i.e., the stack pointer does not point to an area in DGROUP) then you must compile your application with the "zu" option. For example, the pointer returned by malloc in a large data model may not be in DGROUP. The "zu" option relaxes the restriction that the SS register contains the base address of the default data segment, "DGROUP". Normally, all data items are placed into the group DGROUP and the SS register contains the base address of this group. In a thread, the SS register will likely not contain the base address of this group. When the "zu" option is selected, the SS register is volatile (assumed to point to another segment) and any global data references require loading a segment register such as DS with the base address of DGROUP.

Note for OS/2 32-bit applications: Memory for a stack need not be provided by the application. The stack_bottom may be NULL in which case the run-time system will provide a stack. You must specify a non-zero stack_size for this stack.

Note for Win32 applications: Memory for a stack is provided by the run-time system. The size of the stack is determined by stack_size and must not be zero.

The _beginthreadex function can be used to create a new thread, in a running or suspended state specified by initflag, with security attributes specified by security.
The initial state of the new thread (running or suspended) is specified by the *initflag* argument. If the `CREATE_SUSPENDED` flag (WINBASE.H) is specified, the thread is created in a suspended state, and will not run until the Win32 `ResumeThread` function is called with the thread handle as an argument. If this value is zero, the thread runs immediately after creation.

The security descriptor for the new thread is specified by the *security* argument. This is a pointer to a Win32 `SECURITY_ATTRIBUTES` structure (see Microsoft’s Win32 Programmer’s Reference for more information). For default behaviour, the security structure pointer can be NULL.

The thread identifier is returned in the location identified by the *thrdid* argument.

The thread ends when it exits from its main function or calls `exit`, `_exit`, `_endthread` or `_endthreadex`.

The variable/function `_threadid` which is defined in `<stddef.h>` may be used by the executing thread to obtain its thread ID. In the 16-bit libraries, `_threadid` is a far pointer to an int. In the 32-bit libraries, it is a function that returns an int.

There is no limit to the number of threads an application can create under Win32 platforms.

There is a limit to the number of threads an application can create under 16-bit OS/2 and 32-bit NetWare. The default limit is 32. This limit can be adjusted by statically initializing the unsigned global variable `__MaxThreads`.

Under 32-bit OS/2, there is no limit to the number of threads an application can create. However, due to the way in which multiple threads are supported in the Watcom libraries, there is a small performance penalty once the number of threads exceeds the default limit of 32 (this number includes the initial thread). If you are creating more than 32 threads and wish to avoid this performance penalty, you can redefine the threshold value of 32. You can statically initialize the global variable `__MaxThreads`.

By adding the following line to your multi-threaded application, the new threshold value will be set to 48.

```
unsigned __MaxThreads = { 48 };
```

**Returns:**

Under Win32, the `_beginthread` function returns the thread handle for the new thread if successful; otherwise it returns -1 to indicate that the thread could not be started.

Under all other systems that support the `_beginthread` function (OS/2, Netware and QNX), it returns the thread ID for the new thread if successful; otherwise it returns -1 to indicate that the thread could not be started.

The `_beginthreadex` function returns the thread handle for the new thread if successful; otherwise it returns 0 to indicate that the thread could not be started.

When the thread could not be started, the value of `errno` could be set to `EAGAIN` if there are too many threads, or to `EINVAL` if the argument is invalid or the stack size is incorrect, or to `ENOMEM` if there is not enough available memory.

**See Also:**

`_endthread`
Example:

```c
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <malloc.h>
#include <process.h>
#include <dos.h>

#if defined(__386__)
    #define FAR
    #define STACK_SIZE    8192
#else
    #define FAR           __far
    #define STACK_SIZE    4096
#endif

static volatile int     WaitForThread;

void FAR child( void FAR *parm )
{
    char * FAR *argv = (char * FAR *) parm;
    int   i;

    printf( "Child thread ID = %x\n", _threadid );
    for( i = 0; argv[i]; i++ ) {
        printf( "argv[%d] = %s\n", i, argv[i] );
    }
    WaitForThread = 0;
    _endthread();
}
```
void main()
{
    char       *args[3];
#if defined(__NT__)
    unsigned long   tid;
#else
    char           *stack;
    int             tid;
#endif
    args[0] = "child";
    args[1] = "parm";
    args[2] = NULL;
    WaitForThread = 1;
#if defined(__NT__)  
    tid = _beginthread( child, STACK_SIZE, args );
    printf( "Thread handle = %lx\n", tid );
#else
    #if defined(__386__)  
        stack = (char *) malloc( STACK_SIZE );
    #else
        stack = (char *) _nmalloc( STACK_SIZE );
    #endif
    tid = _beginthread( child, stack, STACK_SIZE, args );
    printf( "Thread ID = %x\n", tid );
#endif
    while( WaitForThread ) {
        sleep( 0 );
    }
}

Classification: WATCOM

Systems:   _beginthread - Win32, QNX/32, OS/2 1.x(MT), OS/2 1.x(DL), OS/2-32, Netware
            _beginthreadex - Win32

86  Library Functions and Macros
Synopsis:
#include <math.h>
double j0( double x);
double j1( double x);
double jn( int n, double x);
double y0( double x);
double y1( double x);
double yn( int n, double x);

Description: Functions j0, j1, and jn return Bessel functions of the first kind.

Functions y0, y1, and yn return Bessel functions of the second kind. The argument x must be positive.
If x is negative, _matherr will be called to print a DOMAIN error message to stderr, set errno to EDOM, and return the value -HUGE_VAL. This error handling can be modified by using the matherr routine.

Returns: These functions return the result of the desired Bessel function of x.

See Also: matherr

Example: #include <stdio.h>
#include <math.h>

void main()
{
    double x, y, z;
    x = j0( 2.4 );
y = y1( 1.58 );
z = jn( 3, 2.4 );
    printf( "j0(2.4) = %f, y1(1.58) = %f\n", x, y );
    printf( "jn(3,2.4) = %f\n", z );
}

Classification: WATCOM

Systems: j0 - Math
def - Math
dfn - Math
ey0 - Math
ey1 - Math
eyn - Math
Synopsis:  
#include <string.h>  
int bcmp(const void *s1, const void *s2, size_t n);

Description:  
The `bcmp` function compares the byte string pointed to by `s1` to the string pointed to by `s2`. The number of bytes to compare is specified by `n`. Null characters may be included in the comparison.

Note that this function is similar to the ANSI `memcmp` function but just tests for equality (new code should use the ANSI function).

Returns:  
The `bcmp` function returns zero if the byte strings are identical otherwise it returns 1.

See Also:  
`bcopy`, `bzero`, `memcmp`, `strcmp`

Example:  
```c
#include <stdio.h>
#include <string.h>

void main()
{
    if( bcmp( "Hello there", "Hello world", 6 ) ) {
        printf( "Not equal\n" );
    } else {
        printf( "Equal\n" );
    }
}
```

produces the following:

Equal

Classification:  WATCOM

Systems:  All, Netware
Synopsis:  

```c
#include <string.h>
void bcopy( const void *src, void *dst, size_t n );
```

Description:  The `bcopy` function copies the byte string pointed to by `src` (including any null characters) into the array pointed to by `dst`. The number of bytes to copy is specified by `n`. Copying of overlapping objects is guaranteed to work properly.

Note that this function is similar to the ANSI `memmove` function but the order of arguments is different (new code should use the ANSI function).

Returns:  The `bcopy` function has no return value.

See Also:  `bcmp`, `bzero`, `memmove`, `strcpy`

Example:  

```c
#include <stdio.h>
#include <string.h>

void main()
{
    auto char buffer[80];
    
    bcopy( "Hello ", buffer, 6 );
    bcopy( "world", &buffer[6], 6 );
    printf( "%s\n", buffer );
}
```

produces the following:

```
Hello world
```

Classification:  WATCOM

Systems:  All, Netware
Synopsis: 
#include <malloc.h>
int _bfreeseg( __segment seg );

Description: The _bfreeseg function frees a based-heap segment.

The argument seg indicates the segment returned by an earlier call to _bheapseg.

Returns: The _bfreeseg function returns 0 if successful and -1 if an error occurred.

See Also: _bcalloc, _bexpand, _bfree, _bheapseg, _bmalloc, _brealloc

Example: #include <stdio.h>
#include <stdlib.h>
#include <malloc.h>

struct list {
    struct list __based(__self) *next;
    int value;
};

void main()
{
    int i;
    __segment seg;
    struct list __based(seg) *head;
    struct list __based(seg) *p;

    /* allocate based heap */
    seg = _bheapseg( 1024 );
    if( seg == _NULLSEG ) {
        printf( "Unable to allocate based heap\n" );
        exit( 1 );
    }

    /* create a linked list in the based heap */
    head = 0;
    for( i = 1; i < 10; i++ ) {
        p = _bmalloc( seg, sizeof( struct list ) );
        if( p == _NULLOFF ) {
            printf( "_bmalloc failed\n" );
            break;
        }
        p->next = head;
        p->value = i;
        head = p;
    }

    /* traverse the linked list, printing out values */
    for( p = head; p != 0; p = p->next ) {
        printf( "Value = %d\n", p->value );
    }

90 Library Functions and Macros
/* free all the elements of the linked list */
for( ; p = head; ) {
    head = p->next;
    _bfree( seg, p );
}
/* free the based heap */
_bfreeseg( seg );
Synopsis: 
#include <process.h>
int _bgetcmd( char *cmd_line, int len );

Description: The _bgetcmd function causes the command line information, with the program name removed, to be copied to cmd_line. The argument len specifies the size of cmd_line. The information is terminated with a ‘\0’ character. This provides a method of obtaining the original parameters to a program unchanged (with the white space intact).

This information can also be obtained by examining the vector of program parameters passed to the main function in the program.

Returns: The number of bytes required to store the entire command line, excluding the terminating null character, is returned.

See Also: abort, atexit, exec..., exit, _Exit, _exit, getcmd, getenv, main, onexit, putenv, spawn..., system

Example: Suppose a program were invoked with the command line

    myprog arg-1 ( my stuff ) here

where that program contains

    #include <stdio.h>
    #include <stdlib.h>
    #include <process.h>

    void main( void )
    {
        char  *cmdline;
        int   cmdlen;

        cmdlen = _bgetcmd( NULL, 0 ) + 1;
        cmdline = malloc( cmdlen );
        if( cmdline != NULL ) {
            cmdlen = _bgetcmd( cmdline, cmdlen );
            printf( "%s\n", cmdline );
        }
    }

produces the following:

    arg-1 ( my stuff ) here

Classification: WATCOM

Systems: All, Netware
Synopsis:  
#include <malloc.h>
__segment _bheapseg( size_t size );

Description:  The _bheapseg function allocates a based-heap segment of at least size bytes.

The argument size indicates the initial size for the heap. The heap will automatically be enlarged as needed if there is not enough space available within the heap to satisfy an allocation request by _bcalloc, _bexpand, _bmalloc, or _brealloc.

The value returned by _bheapseg is the segment value or selector for the based heap. This value must be saved and used as an argument to other based heap functions to indicate which based heap to operate upon.

Each call to _bheapseg allocates a new based heap.

Returns:  The value returned by _bheapseg is the segment value or selector for the based heap. This value must be saved and used as an argument to other based heap functions to indicate which based heap to operate upon. A special value of _NULLSEG is returned if the segment could not be allocated.

See Also:  _bfreeseg, _bcalloc, _bexpand, _bmalloc, _brealloc

Example:  
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>

struct list {
  struct list __based(__self) *next;
  int         value;
};

void main()
{
  int         i;
  __segment   seg;
  struct list __based(seg) *head;
  struct list __based(seg) *p;

  /* allocate based heap */
  seg = _bheapseg( 1024 );
  if( seg == _NULLSEG ) {
    printf( "Unable to allocate based heap\n" );
    exit( 1 );
  }
}
/* create a linked list in the based heap */
head = 0;
for( i = 1; i < 10; i++ ) {
    p = _bmalloc( seg, sizeof( struct list ) );
    if( p == _NULLOFF ) {
        printf( "_bmalloc failed\n" );
        break;
    }
    p->next = head;
    p->value = i;
    head = p;
}

/* traverse the linked list, printing out values */
for( p = head; p != 0; p = p->next ) {
    printf( "Value = %d\n", p->value );
}

/* free all the elements of the linked list */
for( ; p = head; ) {
    head = p->next;
    _bfree( seg, p );
}/* free the based heap */
_bfreeseg( seg );

Classification: WATCOM

Systems: DOS/16, Windows, QNX/16, OS/2 1.x(all)
Synopsis:

```c
#include <bios.h>

unsigned short _bios_disk( unsigned service,
    struct diskinfo_t *diskinfo );

struct diskinfo_t {
    /* disk parameters */
    unsigned drive;     /* drive number */
    unsigned head;      /* head number */
    unsigned track;     /* track number */
    unsigned sector;    /* sector number */
    unsigned nsectors;  /* number of sectors */
    void __far *buffer; /* buffer address */
};
```

Description: The `_bios_disk` function uses INT 0x13 to provide access to the BIOS disk functions. Information for the desired `service` is passed the `diskinfo_t` structure pointed to by `diskinfo`. The value for `service` can be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>_DISK_RESET</td>
<td>Forces the disk controller to do a reset on the disk. This request does not use the <code>diskinfo</code> argument.</td>
</tr>
<tr>
<td>_DISK_STATUS</td>
<td>Obtains the status of the last disk operation.</td>
</tr>
<tr>
<td>_DISK_READ</td>
<td>Reads the specified number of sectors from the disk. This request uses all of the information passed in the <code>diskinfo</code> structure.</td>
</tr>
<tr>
<td>_DISK_WRITE</td>
<td>Writes the specified amount of data to the disk. This request uses all of the information passed in the <code>diskinfo</code> structure.</td>
</tr>
<tr>
<td>_DISK_VERIFY</td>
<td>Checks the disk to be sure the specified sectors exist and can be read. A CRC (cyclic redundancy check) test is performed. This request uses all of the information passed in the <code>diskinfo</code> structure except for the <code>buffer</code> field.</td>
</tr>
<tr>
<td>_DISK_FORMAT</td>
<td>Formats the specified track on the disk. The <code>head</code> and <code>track</code> fields indicate the track to be formatted. Only one track can be formatted per call. The <code>buffer</code> field points to a set of sector markers, whose format depends on the type of disk drive. This service has no return value.</td>
</tr>
</tbody>
</table>

This function is not supported by DOS/4GW (you must use the Simulate Real-Mode Interrupt DPMI call).

Returns: The `_bios_disk` function returns status information in the high-order byte when `service` is _DISK_STATUS, _DISK_READ, _DISK_WRITE, or _DISK_VERIFY. The possible values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>Operation successful</td>
</tr>
<tr>
<td>0x01</td>
<td>Bad command</td>
</tr>
<tr>
<td>0x02</td>
<td>Address mark not found</td>
</tr>
<tr>
<td>0x03</td>
<td>Attempt to write to write-protected disk</td>
</tr>
<tr>
<td>0x04</td>
<td>Sector not found</td>
</tr>
<tr>
<td>0x05</td>
<td>Reset failed</td>
</tr>
</tbody>
</table>
### _bios_disk_

<table>
<thead>
<tr>
<th>Hex Value</th>
<th>Error Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x06</td>
<td>Disk changed since last operation</td>
</tr>
<tr>
<td>0x07</td>
<td>Drive parameter activity failed</td>
</tr>
<tr>
<td>0x08</td>
<td>DMA overrun</td>
</tr>
<tr>
<td>0x09</td>
<td>Attempt to DMA across 64K boundary</td>
</tr>
<tr>
<td>0x0A</td>
<td>Bad sector detected</td>
</tr>
<tr>
<td>0x0B</td>
<td>Bad track detected</td>
</tr>
<tr>
<td>0x0C</td>
<td>Unsupported track</td>
</tr>
<tr>
<td>0x10</td>
<td>Data read (CRC/ECC) error</td>
</tr>
<tr>
<td>0x11</td>
<td>CRC/ECC corrected data error</td>
</tr>
<tr>
<td>0x20</td>
<td>Controller failure</td>
</tr>
<tr>
<td>0x40</td>
<td>Seek operation failed</td>
</tr>
<tr>
<td>0x80</td>
<td>Disk timed out or failed to respond</td>
</tr>
<tr>
<td>0xAA</td>
<td>Drive not ready</td>
</tr>
<tr>
<td>0xBB</td>
<td>Undefined error occurred</td>
</tr>
<tr>
<td>0xCC</td>
<td>Write fault occurred</td>
</tr>
<tr>
<td>0xE0</td>
<td>Status error</td>
</tr>
<tr>
<td>0xFF</td>
<td>Sense operation failed</td>
</tr>
</tbody>
</table>

**Example:**
```c
#include <stdio.h>
#include <bios.h>

void main()
{
    struct diskinfo_t di;
    unsigned short status;

    di.drive = di.head = di.track = di.sector = 0;
    di.nsectors = 1;
    di.buffer = NULL;
    status = _bios_disk( _DISK_VERIFY, &di );
    printf( "Status = 0x%4.4X\n", status );
}
```

**Classification:** BIOS

**Systems:** DOS, Windows, Win386
Synopsis: #include <bios.h>
unsigned short _bios_equiplist( void );

Description: The _bios_equiplist function uses INT 0x11 to determine what hardware and peripherals are installed on the machine.

Returns: The _bios_equiplist function returns a set of bits indicating what is currently installed on the machine. Those bits are defined as follows:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 0</td>
<td>Set to 1 if system boots from disk</td>
</tr>
<tr>
<td>bit 1</td>
<td>Set to 1 if a math coprocessor is installed</td>
</tr>
<tr>
<td>bits 2-3</td>
<td>Indicates motherboard RAM size</td>
</tr>
<tr>
<td>bits 4-5</td>
<td>Initial video mode</td>
</tr>
<tr>
<td>bits 6-7</td>
<td>Number of diskette drives</td>
</tr>
<tr>
<td>bit 8</td>
<td>Set to 1 if machine does not have DMA</td>
</tr>
<tr>
<td>bits 9-11</td>
<td>Number of serial ports</td>
</tr>
<tr>
<td>bit 12</td>
<td>Set to 1 if a game port is attached</td>
</tr>
<tr>
<td>bit 13</td>
<td>Set to 1 if a serial printer is attached</td>
</tr>
<tr>
<td>bits 14-15</td>
<td>Number of parallel printers installed</td>
</tr>
</tbody>
</table>

Example: #include <stdio.h>
#include <bios.h>

void main()
{
    unsigned short equipment;
    equipment = _bios_equiplist();
    printf( "Equipment flags = 0x%4.4X\n", equipment );
}

Classification: BIOS

Systems: DOS, Windows, Win386
Synopsis: #include <bios.h>
unsigned short _bios_keybrd( unsigned service );

Description: The _bios_keybrd function uses INT 0x16 to access the BIOS keyboard services. The possible values for service are the following constants:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>_KEYBRD_READ</td>
<td>Reads the next character from the keyboard. The function will wait until a character has been typed.</td>
</tr>
<tr>
<td>_KEYBRD_READY</td>
<td>Checks to see if a character has been typed. If there is one, then its value will be returned, but it is not removed from the input buffer.</td>
</tr>
<tr>
<td>_KEYBRD_SHIFTSTATUS</td>
<td>Returns the current state of special keys.</td>
</tr>
<tr>
<td>_NKEYBRD_READ</td>
<td>Reads the next character from an enhanced keyboard. The function will wait until a character has been typed.</td>
</tr>
<tr>
<td>_NKEYBRD_READY</td>
<td>Checks to see if a character has been typed on an enhanced keyboard. If there is one, then its value will be returned, but it is not removed from the input buffer.</td>
</tr>
<tr>
<td>_NKEYBRD_SHIFTSTATUS</td>
<td>Returns the current state of special keys on an enhanced keyboard.</td>
</tr>
</tbody>
</table>

Returns: The return value depends on the service requested.

The _KEYBRD_READ and _NKEYBRD_READ services return the character’s ASCII value in the low-order byte and the character’s keyboard scan code in the high-order byte.

The _KEYBRD_READY and _NKEYBRD_READY services return zero if there was no character available, otherwise it returns the same value returned by _KEYBRD_READ and _NKEYBRD_READ.

The shift status is returned in the low-order byte with one bit for each special key defined as follows:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 0 (0x01)</td>
<td>Right SHIFT key is pressed</td>
</tr>
<tr>
<td>bit 1 (0x02)</td>
<td>Left SHIFT key is pressed</td>
</tr>
<tr>
<td>bit 2 (0x04)</td>
<td>CTRL key is pressed</td>
</tr>
<tr>
<td>bit 3 (0x08)</td>
<td>ALT key is pressed</td>
</tr>
<tr>
<td>bit 4 (0x10)</td>
<td>SCROLL LOCK is on</td>
</tr>
<tr>
<td>bit 5 (0x20)</td>
<td>NUM LOCK is on</td>
</tr>
<tr>
<td>bit 6 (0x40)</td>
<td>CAPS LOCK is on</td>
</tr>
<tr>
<td>bit 7 (0x80)</td>
<td>Insert mode is set</td>
</tr>
</tbody>
</table>

Example:

```c
#include <stdio.h>
#include <bios.h>

void main()
{
    unsigned short key_state;
```
key_state = __bios_keybrd( __KEYBRD_SHIFTSTATUS );
if( key_state & 0x10 )
    printf( "SCROLL LOCK is on\n" );
if( key_state & 0x20 )
    printf( "NUM LOCK is on\n" );
if( key_state & 0x40 )
    printf( "CAPS LOCK is on\n" );
}

produces the following:

NUM LOCK is on

Classification: BIOS

Systems: DOS, Windows, Win386
### _bios_memsize

**Synopsis:**
```
#include <bios.h>
unsigned short _bios_memsize( void );
```

**Description:** The `_bios_memsize` function uses INT 0x12 to determine the total amount of memory available.

**Returns:** The `_bios_memsize` function returns the total amount of 1K blocks of memory installed (maximum 640).

**Example:**
```
#include <stdio.h>
#include <bios.h>

void main()
{
    unsigned short memsize;
    memsize = _bios_memsize();
    printf( "The total amount of memory is: %dK\n", memsize );
}
```

produces the following:

```
The total amount of memory is: 640K
```

**Classification:** BIOS

**Systems:** DOS, Windows, Win386
Synopsis:  
#include <bios.h>  
unsigned short _bios_printer( unsigned service,  
                               unsigned port,  
                               unsigned data );

Description:  
The _bios_printer function uses INT 0x17 to perform printer output services to the printer specified by port. The values for service are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>_PRINTER_WRITE</td>
<td>Sends the low-order byte of data to the printer specified by port.</td>
</tr>
<tr>
<td>_PRINTER_INIT</td>
<td>Initializes the printer specified by port.</td>
</tr>
<tr>
<td>_PRINTER_STATUS</td>
<td>Get the status of the printer specified by port.</td>
</tr>
</tbody>
</table>

Returns:  
The _bios_printer function returns a printer status byte defined as follows:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 0 (0x01)</td>
<td>Printer timed out</td>
</tr>
<tr>
<td>bits 1-2</td>
<td>Unused</td>
</tr>
<tr>
<td>bit 3 (0x08)</td>
<td>I/O error</td>
</tr>
<tr>
<td>bit 4 (0x10)</td>
<td>Printer selected</td>
</tr>
<tr>
<td>bit 5 (0x20)</td>
<td>Out of paper</td>
</tr>
<tr>
<td>bit 6 (0x40)</td>
<td>Printer acknowledge</td>
</tr>
<tr>
<td>bit 7 (0x80)</td>
<td>Printer not busy</td>
</tr>
</tbody>
</table>

Example:  
#include <stdio.h>  
#include <bios.h>  

void main()  
{  
    unsigned short status;  
    status = _bios_printer( _PRINTER_STATUS, 1, 0 );  
    printf( "Printer status: 0x%2.2X\n", status );  
}

Classification:  BIOS

Systems:  DOS, Windows, Win386
Synopsis:    #include <bios.h>
            unsigned short _bios_serialcom( unsigned service,
                                   unsigned serial_port,
                                   unsigned data );

Description: The _bios_serialcom function uses INT 0x14 to provide serial communications services to the serial port specified by serial_port. 0 represents COM1, 1 represents COM2, etc. The values for service are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>_COM_INIT</td>
<td>Initializes the serial port to the parameters specified in data.</td>
</tr>
<tr>
<td>_COM_SEND</td>
<td>Transmits the low-order byte of data to the serial port.</td>
</tr>
<tr>
<td>_COM_RECEIVE</td>
<td>Reads an input character from the serial port.</td>
</tr>
<tr>
<td>_COM_STATUS</td>
<td>Returns the current status of the serial port.</td>
</tr>
</tbody>
</table>

The value passed in data for the _COM_INIT service can be built using the appropriate combination of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>_COM_110</td>
<td>110 baud</td>
</tr>
<tr>
<td>_COM_150</td>
<td>150 baud</td>
</tr>
<tr>
<td>_COM_300</td>
<td>300 baud</td>
</tr>
<tr>
<td>_COM_600</td>
<td>600 baud</td>
</tr>
<tr>
<td>_COM_1200</td>
<td>1200 baud</td>
</tr>
<tr>
<td>_COM_2400</td>
<td>2400 baud</td>
</tr>
<tr>
<td>_COM_4800</td>
<td>4800 baud</td>
</tr>
<tr>
<td>_COM_9600</td>
<td>9600 baud</td>
</tr>
<tr>
<td>_COM_NOPARITY</td>
<td>No parity</td>
</tr>
<tr>
<td>_COM_EVENPARITY</td>
<td>Even parity</td>
</tr>
<tr>
<td>_COM_ODDPARITY</td>
<td>Odd parity</td>
</tr>
<tr>
<td>_COM_CHR7</td>
<td>7 data bits</td>
</tr>
<tr>
<td>_COM_CHR8</td>
<td>8 data bits</td>
</tr>
<tr>
<td>_COM_STOP1</td>
<td>1 stop bit</td>
</tr>
<tr>
<td>_COM_STOP2</td>
<td>2 stop bits</td>
</tr>
</tbody>
</table>

Returns: The _bios_serialcom function returns a 16-bit value with the high-order byte containing status information defined as follows:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 15 (0x8000)</td>
<td>Timed out</td>
</tr>
<tr>
<td>bit 14 (0x4000)</td>
<td>Transmit shift register empty</td>
</tr>
<tr>
<td>bit 13 (0x2000)</td>
<td>Transmit holding register empty</td>
</tr>
</tbody>
</table>
bit 12 (0x1000)   Break detected
bit 11 (0x0800)   Framing error
bit 10 (0x0400)   Parity error
bit  9 (0x0200)   Overrun error
bit  8 (0x0100)   Data ready

The low-order byte of the return value depends on the value of the service argument.

When service is _COM_SEND, bit 15 will be set if the data could not be sent. If bit 15 is clear, the return value equals the byte sent.

When service is _COM_RECEIVE, the byte read will be returned in the low-order byte if there was no error. If there was an error, at least one of the high-order status bits will be set.

When service is _COM_INIT or _COM_STATUS the low-order bits are defined as follows:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit  0 (0x01)</td>
<td>Clear to send (CTS) changed</td>
</tr>
<tr>
<td>bit  1 (0x02)</td>
<td>Data set ready changed</td>
</tr>
<tr>
<td>bit  2 (0x04)</td>
<td>Trailing-edge ring detector</td>
</tr>
<tr>
<td>bit  3 (0x08)</td>
<td>Receive line signal detector changed</td>
</tr>
<tr>
<td>bit  4 (0x10)</td>
<td>Clear to send</td>
</tr>
<tr>
<td>bit  5 (0x20)</td>
<td>Data-set ready</td>
</tr>
<tr>
<td>bit  6 (0x40)</td>
<td>Ring indicator</td>
</tr>
<tr>
<td>bit  7 (0x80)</td>
<td>Receive-line signal detected</td>
</tr>
</tbody>
</table>

Example:
```c
#include <stdio.h>
#include <bios.h>

void main()
{
    unsigned short status;

    status = _bios_serialcom(_COM_STATUS, 1, 0);
    printf( "Serial status: 0x%2.2X\n", status );
}
```

Classification: BIOS

Systems: DOS, Windows, Win386
Synopsis:  
#include <bios.h>
int _bios_timeofday( int service, long *timeval );

Description:  The _bios_timeofday function uses INT 0x1A to get or set the current system clock value. The values for service are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>_TIME_GETCLOCK</td>
<td>Places the current system clock value in the location pointed to by timeval. The function returns zero if midnight has not passed since the last time the system clock was read or set; otherwise, it returns 1.</td>
</tr>
<tr>
<td>_TIME_SETCLOCK</td>
<td>Sets the system clock to the value in the location pointed to by timeval.</td>
</tr>
</tbody>
</table>

Returns:  A value of -1 is returned if neither _TIME_GETCLOCK nor _TIME_SETCLOCK were specified; otherwise 0 is returned.

Example:  
#include <stdio.h>
#include <bios.h>

void main()
{
    long time_of_day;
    _bios_timeofday( _TIME_GETCLOCK, &time_of_day );
    printf( "Ticks since midnight: %lu\n", time_of_day );
}

produces the following:

Ticks since midnight: 762717

Classification:  BIOS

Systems:  DOS, Windows, Win386
Synopsis:  
#include <stdio.h>  
int _bprintf( char *buf, size_t bufsize,  
   const char *format, ... ); 
int _bwprintf( wchar_t *buf, size_t bufsize,  
   const wchar_t *format, ... );

Description:  
The _bprintf function is equivalent to the sprintf function, except that the argument bufsize specifies the size of the character array buf into which the generated output is placed. A null character is placed at the end of the generated character string. The format string is described under the description of the printf function.

The _bwprintf function is identical to _bprintf except that the argument buf specifies an array of wide characters into which the generated output is to be written, rather than converted to multibyte characters and written to a stream. The _bwprintf function accepts a wide-character string argument for format.

Returns:  
The _bprintf function returns the number of characters written into the array, not counting the terminating null character. An error can occur while converting a value for output. When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also:  
cprintf, fprintf, printf, sprintf, _vbprintf, vcprintf, vfprintf, vprintf, vsprintf

Example:  
#include <stdio.h>  
void main( int argc, char *argv[] )  
{  
   char file_name[9];  
   char file_ext[4];  
   _bprintf( file_name, 9, "%s", argv[1] );  
   _bprintf( file_ext, 4, "%s", argv[2] );  
   printf( "%s.%s\n", file_name, file_ext );  
}

Classification:  
WATCOM

Systems:  
_bprintf - All, Netware  
_bwprintf - All
Synopsis:  
#include <stdlib.h>
void break_off( void );
void break_on( void );

Description:  The break_off function can be used with DOS to restrict break checking (Ctrl/C, Ctrl/Break) to screen output and keyboard input. The break_on function can be used with DOS to add break checking (Ctrl/C, Ctrl/Break) to other activities such as disk file input/output.

Returns:  The break_off and break_on functions to not return anything.

See Also:  signal

Example:  
#include <stdio.h>
#include <stdlib.h>

void main()
{
    long i;
    FILE *tmpf;

    tmpf = tmpfile();
    if( tmpf != NULL ) {
        printf( "Start\n" );
        break_off();
        for( i = 1; i < 100000; i++ )
            fprintf( tmpf, "%ld\n", i );
        break_on();
        printf( "Finish\n" );
    }
}

Classification:  DOS

Systems:  break_off - DOS, Windows, Win386
          break_on - DOS, Windows, Win386
**Synopsis:**
```
#include <stdlib.h>
void *bsearch( const void *key,
               const void *base,
               size_t num,
               size_t width,
               int (*compar)( const void *pkey,
                              const void *pbase) );
```

**Safer C:** The Safer C Library extension provides the `bsearch_s` function which is a safer alternative to `bsearch`. This newer `bsearch_s` function is recommended to be used instead of the traditional "unsafe" `bsearch` function.

**Description:** The `bsearch` function performs a binary search of a sorted array of `num` elements, which is pointed to by `base`, for an item which matches the object pointed to by `key`. Each element in the array is `width` bytes in size. The comparison function pointed to by `compar` is called with two arguments that point to elements in the array. The first argument `pkey` points to the same object pointed to by `key`. The second argument `pbase` points to a element in the array. The comparison function shall return an integer less than, equal to, or greater than zero if the `key` object is less than, equal to, or greater than the element in the array.

**Returns:** The `bsearch` function returns a pointer to the matching member of the array, or `NULL` if a matching object could not be found. If there are multiple values in the array which are equal to the `key`, the return value is not necessarily the first occurrence of a matching value when the array is searched linearly.

**See Also:** `bsearch_s`, `lfind`, `lsearch`, `qsort`, `qsort_s`

**Example:**
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

static const char *keywords[] = {
    "auto",
    "break",
    "case",
    "char",
    /* . */
    /* . */
    /* . */
    "while"
};

#define NUM_KW sizeof(keywords) / sizeof(char *)

int kw_compare( const void *p1, const void *p2 )
{
    const char *plc = (const char *) p1;
    const char **p2c = (const char **) p2;
    return( strcmp( plc, *p2c ) );
}
```
```c
int keyword_lookup( const char *name )
{
    const char **key;
    key = (char const **) bsearch( name, keywords, NUM_KW,
                                 sizeof( char * ), kw_compare);
    if( key == NULL ) return( -1 );
    return key - keywords;
}

void main()
{
    printf( "%d\n", keyword_lookup( "case" ) );
    printf( "%d\n", keyword_lookup( "crigger" ) );
    printf( "%d\n", keyword_lookup( "auto" ) );
}

//************ Sample program output ************
//2
//-1
//0

produces the following:

2
-1
0

Classification: ANSI

Systems: All, Netware
Synopsis:

```c
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
void *bsearch_s( const void *key,
  const void *base,
  rsize_t nmemb,
  rsize_t size,
  int (*compar)( const void *k, const void *y, void *context ),
  void *context );
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and bsearch_s will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither `nmemb` nor `size` shall be greater than `RSIZE_MAX`. If `nmemb` is not equal to zero, then none of `key`, `base`, or `compar` shall be a null pointer. If there is a runtime-constraint violation, the `bsearch_s` function does not search the array.

Description: The `bsearch_s` function searches an array of `nmemb` objects, the initial element of which is pointed to by `base`, for an element that matches the object pointed to by `key`. The size of each element of the array is specified by `size`. The comparison function pointed to by `compar` is called with three arguments. The first two point to the key object and to an array element, in that order. The function shall return an integer less than, equal to, or greater than zero if the key object is considered, respectively, to be less than, to match, or to be greater than the array element. The array shall consist of: all the elements that compare less than, all the elements that compare equal to, and all the elements that compare greater than the key object, in that order. The third argument to the comparison function is the `context` argument passed to `bsearch_s`. The sole use of context by `&funcs` is to pass it to the comparison function.

Returns: The `bsearch_s` function returns a pointer to a matching element of the array, or a null pointer if no match is found or there is a runtime-constraint violation. If two elements compare as equal, which element is matched is unspecified.

See Also: `bsearch`, `lfind`, `lsearch`, `qsort`, `qsort_s`

Example:

```c
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

static const char *keywords[] = {
  "auto",
  "break",
  "case",
  "char",
  /* . */
  /* . */
  /* . */
  "while"
};
static void * context = NULL;

#define NUM_KW  sizeof(keywords) / sizeof(char *)
```
int kw_compare( const void *p1, const void *p2, void *context )
{
    const char *plc = (const char *) p1;
    const char **p2c = (const char **) p2;
    return( strcmp( plc, *p2c ) );
}

int keyword_lookup( const char *name )
{
    const char **key;
    key = (char const **) bsearch_s( name, keywords, NUM_KW,
    sizeof( char * ), kw_compare, context );
    if( key == NULL ) return( -1 );
    return key - keywords;
}

int main()
{
    printf( "%d\n", keyword_lookup( "case" ) );
    printf( "%d\n", keyword_lookup( "crigger" ) );
    printf( "%d\n", keyword_lookup( "auto" ) );
    return 0;
}

//************* Sample program output *************
//2
//-1
//0

produces the following:

2
-1
0

Classification: TR 24731

Systems: All, Netware
btowc

Synopsis:  
```c
#include <wchar.h>
wint_t btowc( int c );
```

Description:  The `btowc` function determines whether `c` is a valid single-byte character in the initial shift state.

Returns:  The `btowc` function returns `WEOF` if `c` has the value `EOF` or if `(unsigned char)c` does not constitute a valid single-byte character in the initial shift state. Otherwise, `btowc` returns the wide character representation of that character.

See Also:  `_mbccmp`, `_mbccpy`, `_mbccmp`, `_mbcjistojms`, `_mbclen`, `_mbctohira`, `_mbctokata`, `_mbctolower`, `_mbctombb`, `_mbctoupper`, `mblen`, `mbrlen`, `mbrlen`, `mbrtowc`, `mbsrtowcs`, `mbsrtowcs`, `mbstowcs`, `mbstowcs`, `mbtowc`, `wctomb`, `wcrtomb`, `mbtowc`, `wcrtowcs`, `wctomb`, `wctomb`, `wctomb`, `wctomb`, `wctomb`

Example:  
```c
#include <stdio.h>
#include <wchar.h>

void main( void )
{
    printf( "EOF is %sa valid single-byte character\n", btowc( EOF ) == WEOF ? "not " : "" );
}
```

produces the following:

EOF is not a valid single-byte character

Classification:  ANSI

Systems:  DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:
#include <string.h>
void bzero( void *dst, size_t n );

Description: The `bzero` function fills the first `n` bytes of the object pointed to by `dst` with zero (null) bytes.

Note that this function is similar to the ANSI `memset` function (new code should use the ANSI function).

Returns: The `bzero` function has no return value.

See Also: `bcmp`, `bcopy`, `memset`, `strset`

Example:
#include <string.h>

void main()
{
    char buffer[80];
    bzero( buffer, 80 );
}

Classification: WATCOM

Systems: All, Netware
Synopsis:
#include <math.h>
double cabs( struct complex value );

struct _complex {
    double x; /* real part */
    double y; /* imaginary part */
};

Description: The `cabs` function computes the absolute value of the complex number `value` by a calculation which is equivalent to

\[ \sqrt{ (value.x*value.x) + (value.y*value.y) } \]

In certain cases, overflow errors may occur which will cause the `matherr` routine to be invoked.

Returns: The absolute value is returned.

Example:
#include <stdio.h>
#include <math.h>

struct _complex c = { -3.0, 4.0 };

void main()
{
    printf( "%f\n", cabs( c ) );
}

produces the following:

5.000000

Classification: WATCOM

Systems: Math
**calloc Functions**

**Synopsis:**
#include <stdlib.h>  For ANSI compatibility (calloc only)
#include <malloc.h>  Required for other function prototypes
void *calloc( size_t n, size_t size );
void __based(void) *__balloc( __segment seg, size_t n, size_t size );
void __far  *__falloc( size_t n, size_t size );
void __near  *__nalloc( size_t n, size_t size );

**Description:** The calloc functions allocate space for an array of n objects, each of length size bytes. Each element is initialized to 0.

Each function allocates memory from a particular heap, as listed below:

<table>
<thead>
<tr>
<th>Function</th>
<th>Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>calloc</td>
<td>Depends on data model of the program</td>
</tr>
<tr>
<td>_balloc</td>
<td>Based heap specified by seg value</td>
</tr>
<tr>
<td>_falloc</td>
<td>Far heap (outside the default data segment)</td>
</tr>
<tr>
<td>_nalloc</td>
<td>Near heap (inside the default data segment)</td>
</tr>
</tbody>
</table>

In a small data memory model, the calloc function is equivalent to the _nalloc function; in a large data memory model, the calloc function is equivalent to the _falloc function.

A block of memory allocated should be freed using the appropriate free function.

**Returns:** The calloc functions return a pointer to the start of the allocated memory. The return value is NULL (_NULLOFF for _balloc) if there is insufficient memory available or if the value of the size argument is zero.

**See Also:** _expand Functions, free Functions, halloc, hfree, malloc Functions, _msize Functions, realloc Functions, sbrk

**Example:**
#include <stdlib.h>
void main()
{
  char *buffer;
  buffer = (char *)calloc( 80, sizeof(char) );
}

**Classification:** calloc is ANSI
_falloc is not ANSI
_balloc is not ANSI
_nalloc is not ANSI

**Systems:**
calloc - All, Netware
_balloc - DOS/16, Windows, QNX/16, OS/2 1.x(all)
_falloc - DOS/16, Windows, QNX/16, OS/2 1.x(all)
_ncalloc - DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2 1.x(MT), OS/2-32
Synopsis:  
#include <math.h>  
double ceil( double x );

Description:  The ceil function (ceiling function) computes the smallest integer not less than x.

Returns:  The ceil function returns the smallest integer not less than x, expressed as a double.

See Also:  floor

Example:  
#include <stdio.h>  
#include <math.h>  

void main()  
{  
    printf( "%f %f %f %f %f\n", ceil( -2.1 ), ceil( -2. ),  
            ceil( 0.0 ), ceil( 2. ), ceil( 2.1 ) );  
}

produces the following:

-2.000000 -2.000000 0.000000 2.000000 3.000000

Classification:  ANSI

Systems:  Math
Synopsis: 
#include <conio.h>
char *cgets( char *buf );

Description: The cgets function gets a string of characters directly from the console and stores the string and its length in the array pointed to by buf. The first element of the array buf[0] must contain the maximum length in characters of the string to be read. The array must be big enough to hold the string, a terminating null character, and two additional bytes.

The cgets function reads characters until a carriage-return line-feed combination is read, or until the specified number of characters is read. The string is stored in the array starting at buf[2]. The carriage-return line-feed combination, if read, is replaced by a null character. The actual length of the string read is placed in buf[1].

Returns: The cgets function returns a pointer to the start of the string which is at buf[2].

See Also: fgets, getch, getche, gets

Example: 
#include <conio.h>

void main()
{
    char buffer[82];

    buffer[0] = 80;
    cgets( buffer );
    cprintf( "%s\r\n", &buffer[2] );
}

Classification: WATCOM

Systems: All, Netware
#include <dos.h>

void _chain_intr( void (__interrupt __far *func)() );

The _chain_intr function is used at the end of an interrupt routine to start executing another interrupt handler (usually the previous handler for that interrupt). When the interrupt handler designated by func receives control, the stack and registers appear as though the interrupt just occurred.

The _chain_intr function does not return.

_volatile int clock_ticks;
void (__interrupt __far *prev_int_1c)();
#define BLIP_COUNT (5*18) /* 5 seconds */

void __interrupt __far timer_rtn()
{
   ++clock_ticks;
   _chain_intr( prev_int_1c );
}

int delays = 0;

int compile_a_line()
{
   if( delays > 15 ) return( 0 );
   delay( 1000 ); /* delay for 1 second */
   printf( "Delayed for 1 second\n" );
   delays++;
   return( 1 );
}

void main()
{
   prev_int_1c = _dos_getvect( 0x1c );
   _dos_setvect( 0x1c, timer_rtn );
   while( compile_a_line() ) {
      if( clock_ticks >= BLIP_COUNT ) {
         putchar( '.' );
         clock_ticks -= BLIP_COUNT;
      }
   }
   _dos_setvect( 0x1c, prev_int_1c );
}

Classification: WATCOM

Systems: DOS, Windows
Synopsis:  
#include <sys/types.h>
#include <direc.h>
int chdir( const char *path );
int _chdir( const char *path );
int _wchdir( const wchar_t *path );

Description:  The chdir function changes the current directory on the specified drive to the specified path. If no drive is specified in path then the current drive is assumed. The path can be either relative to the current directory on the specified drive or it can be an absolute path name.

Each drive under DOS, OS/2 or Windows has a current directory. The current working directory is the current directory of the current drive. If you wish to change the current drive, you must use the _dos_setdrive function.

The _chdir function is identical to chdir. Use _chdir for ANSI/ISO naming conventions.

The _wchdir function is identical to chdir except that it accepts a wide-character string argument.

Returns: The chdir function returns zero if successful. Otherwise, -1 is returned, errno is set to indicate the error, and the current working directory remains unchanged.

Errors: When an error has occurred, errno contains a value indicating the type of error that has been detected.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENOENT</td>
<td>The specified path does not exist or path is an empty string.</td>
</tr>
</tbody>
</table>

See Also: chmod, _dos_setdrive, getcwd, mkdir, rmdir, stat, umask

Example:  
#include <stdio.h>
#include <stdlib.h>
#include <direct.h>

void main( int argc, char *argv[] )
{
    if( argc != 2 ) {
        fprintf( stderr, "Use: cd <directory>\n" );
        exit( 1 );
    }

    if( chdir( argv[1] ) == 0 ) {
        printf( "Directory changed to %s\n", argv[1] );
        exit( 0 );
    } else {
        perror( argv[1] );
        exit( 1 );
    }
}

Classification: chdir is POSIX 1003.1
_chdir is not POSIX
_wchdir is not POSIX
_wchdir conforms to ANSI/ISO naming conventions
chdir, _chdir, _wchdir

Systems:  
chdir - All, Netware
_chdir - All, Netware
_wchdir - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <direct.h>
void _chdrive( int drive );

Description:  The _chdrive function changes the current working drive to the one specified by drive. A value of 1 is drive A, 2 is drive B, 3 is drive C, etc.

Returns:  The _chdrive function returns zero if drive is successfully changed. Otherwise, -1 is returned.

See Also:  _dos_getdrive, _dos_setdrive, _getdrive

Example:  
#include <stdio.h>
#include <direct.h>

void main( void )
{
    int drive = 3;

    if( _chdrive( drive ) == 0 )
        printf( "Changed the current drive to %c\n", 'A' + drive - 1 );
}

produces the following:

Changed the current drive to C

Classification:  DOS

Systems:  DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM
**Synopsis:**

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <io.h>
int chmod( const char *path, int permission );
int _chmod( const char *path, int permission );
int _wchmod( const wchar_t *path, int permission );
```

**Description:**
The `chmod` function changes the permissions for a file specified by `path` to be the settings in the mode given by `permission`. The access permissions for the file or directory are specified as a combination of bits (defined in the `<sys/stat.h>` header file).

The following bits define permissions for the owner.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_IRWXU</td>
<td>Read, write, execute/search</td>
</tr>
<tr>
<td>S_IRUSR</td>
<td>Read permission</td>
</tr>
<tr>
<td>S_IWUSR</td>
<td>Write permission</td>
</tr>
<tr>
<td>S_IXUSR</td>
<td>Execute/search permission</td>
</tr>
</tbody>
</table>

The following bits define permissions for the group.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_IRWXG</td>
<td>Read, write, execute/search</td>
</tr>
<tr>
<td>S_IRGRP</td>
<td>Read permission</td>
</tr>
<tr>
<td>S_IWGRP</td>
<td>Write permission</td>
</tr>
<tr>
<td>S_IXGRP</td>
<td>Execute/search permission</td>
</tr>
</tbody>
</table>

The following bits define permissions for others.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_IRWXO</td>
<td>Read, write, execute/search</td>
</tr>
<tr>
<td>S_IROTH</td>
<td>Read permission</td>
</tr>
<tr>
<td>S_IWOTH</td>
<td>Write permission</td>
</tr>
<tr>
<td>S_IXOTH</td>
<td>Execute/search permission</td>
</tr>
</tbody>
</table>

The following bits define miscellaneous permissions used by other implementations.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_IREAD</td>
<td>is equivalent to S_IRUSR (read permission)</td>
</tr>
<tr>
<td>S_IWRITE</td>
<td>is equivalent to S_IWUSR (write permission)</td>
</tr>
<tr>
<td>S_IEXEC</td>
<td>is equivalent to S_IXUSR (execute/search permission)</td>
</tr>
</tbody>
</table>

Upon successful completion, the `chmod` function will mark for update the `st_ctime` field of the file.

The `_chmod` function is identical to `chmod`. Use `_chmod` for ANSI naming conventions.

The `_wchmod` function is identical to `chmod` except that it accepts a wide-character string argument.

---

122 Library Functions and Macros
Returns: The `chmod` returns zero if the new settings are successfully made; otherwise, -1 is returned and `errno` is set to indicate the error.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EACCES</td>
<td>Search permission is denied for a component of <code>path</code>.</td>
</tr>
<tr>
<td>ENOENT</td>
<td>The specified <code>path</code> does not exist or <code>path</code> is an empty string.</td>
</tr>
</tbody>
</table>

See Also: `fstat`, `open`, `sopen`, `stat`

Example:
```c
/*
 * change the permissions of a list of files
 * to be read/write by the owner only
 */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <io.h>

void main( int argc, char *argv[] )
{
    int i;
    int ecode = 0;

    for( i = 1; i < argc; i++ ) {
        if( chmod( argv[i], S_IRUSR | S_IWUSR ) == -1 ) {
            perror( argv[i] );
            ecode++;
        }
    }
    exit( ecode );
}
```

Classification: `chmod` is POSIX 1003.1
`_chmod` is not POSIX
`_wchmod` is not POSIX

Systems:
- `chmod` - All, Netware
- `_chmod` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
- `_wchmod` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <io.h>
int chsize( int handle, long size );
int _chsize( int handle, long size );

Description:  The chsize function changes the size of the file associated with handle by extending or truncating the file to the length specified by size. If the file needs to be extended, the file is padded with NULL ('\0') characters.

The _chsize function is identical to chsize. Use _chsize for ANSI naming conventions.

Returns:  The chsize function returns zero if successful. A return value of -1 indicates an error, and errno is set to indicate the error.

Errors:  When an error has occurred, errno contains a value indicating the type of error that has been detected.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EACCES</td>
<td>The specified file is locked against access.</td>
</tr>
<tr>
<td>EBADF</td>
<td>Invalid file handle.</td>
</tr>
<tr>
<td>ENOSPC</td>
<td>Not enough space left on the device to extend the file.</td>
</tr>
</tbody>
</table>

See Also:  close, creat, open

Example:  #include <stdio.h>
#include <io.h>
#include <fcntl.h>
#include <sys\stat.h>

void main()
{
  int handle;

  handle = open( "file", O_RDWR | O_CREAT,
    S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );
  if( handle != -1 ) {
    if( chsize( handle, 32 * 1024L ) != 0 ) {
      printf( "Error extending file\n" );
    }
    close( handle );
  }
}

Classification: WATCOM

Systems: chsize - All, Netware
          _chsize - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis: #include <float.h> unsigned int _clear87( void );

Description: The _clear87 function clears the floating-point status word which is used to record the status of 8087/80287/80387/80486 floating-point operations.

Returns: The _clear87 function returns the old floating-point status. The description of this status is found in the <float.h> header file.

See Also: _control87, _controlfp, _finite, _freset, _status87

Example: #include <stdio.h> #include <float.h>

void main()
{
    unsigned int fp_status;

    fp_status = _clear87();

    printf( "80x87 status =" );
    if( fp_status & SW_INVALID )
        printf( " invalid" );
    if( fp_status & SW_DENORMAL )
        printf( " denormal" );
    if( fp_status & SW_ZERODIVIDE )
        printf( " zero_divide" );
    if( fp_status & SW_OVERFLOW )
        printf( " overflow" );
    if( fp_status & SW_UNDERFLOW )
        printf( " underflow" );
    if( fp_status & SW_INEXACT )
        printf( " inexact_result" );
    printf( "\n" );
}

Classification: Intel

Systems: Math
# clearenv

## Synopsis:
```c
#include <env.h>
int clearenv( void );
```

## Description:
The `clearenv` function clears the process environment area. No environment variables are defined immediately after a call to the `clearenv` function. Note that this clears the PATH, COMSPEC, and TZ environment variables which may then affect the operation of other library functions.

The `clearenv` function may manipulate the value of the pointer `environ`.

## Returns:
The `clearenv` function returns zero upon successful completion. Otherwise, it will return a non-zero value and set `errno` to indicate the error.

## Errors:
When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENOMEM</td>
<td>Not enough memory to allocate a control structure.</td>
</tr>
</tbody>
</table>

## See Also:
`exec...`, `getenv`, `getenv_s`, `putenv`, `_searchenv`, `setenv`, `spawn...`, `system`

## Example:
The following example clears the entire environment area and sets up a new TZ environment variable.

```c
#include <env.h>

void main()
{
    clearenv();
    setenv( "TZ", "EST5EDT", 0 );
}
```

## Classification:
WATCOM

## Systems:
All, Netware
Synopsis:    

```
#include <stdio.h>
void clearerr( FILE *fp );
```

Description: The `clearerr` function clears the end-of-file and error indicators for the stream pointed to by `fp`. These indicators are cleared only when the file is opened or by an explicit call to the `clearerr` or `rewind` functions.

Returns: The `clearerr` function returns no value.

See Also: `feof`, `ferror`, `perror`, `strerror`

Example:    
```
#include <stdio.h>

void main()
{
    FILE *fp;
    int c;

    c = 'J';
    fp = fopen( "file", "w" );
    if( fp != NULL ) {
        fputc( c, fp );
        if( ferror( fp ) ) { /* if error */
            clearerr( fp ); /* clear the error */
            fputc( c, fp ); /* and retry it */
        }
    }
}
```

Classification: ANSI

Systems: All, Netware
Synopsis: #include <graph.h>
void _FAR _clearscreen( short area );

Description: The _clearscreen function clears the indicated area and fills it with the background color. The area argument must be one of the following values:

_GCLEARSCREEN area is entire screen
_GVIEWPORT area is current viewport or clip region
_GWINDOW area is current text window

Returns: The _clearscreen function does not return a value.

See Also: _setbkcolor, _setviewport, _setcliplgn, _settextwindow

Example: #include <conio.h>
#include <graph.h>

tvoid main()
{
_setvideomode( _VRES16COLOR );
_rectangle( _GFILLINTERIOR, 100, 100, 540, 380 );
getch();
_setviewport( 200, 200, 440, 280 );
_clearscreen( _GVIEWPORT );
getch();
_setvideomode( _DEFAULTMODE );
}

Classification: PC Graphics

Systems: DOS, QNX
Synopsis: #include <time.h>
clock_t clock(void);

Description: The clock function returns the number of clock ticks of processor time used by program since the program started executing. This can be converted to seconds by dividing by the value of the macro _CLOCKS_PER_SEC_.

Note that under DOS and OS/2, the clock tick counter will reset to 0 for each subsequent 24 hour interval that elapses.

Returns: The clock function returns the number of clock ticks that have occurred since the program started executing.

See Also: asctime Functions, asctime_s, ctime Functions, ctime_s, difftime, gmtime, gmtime_s, localtime, localtime_s, mktime, strftime, time, tzset

Example: #include <stdio.h>
#include <math.h>
#include <time.h>

void compute( void )
{
    int i, j;
    double x;

    x = 0.0;
    for( i = 1; i <= 100; i++ )
        for( j = 1; j <= 100; j++ )
            x += sqrt( (double) i * j );
    printf( "%16.7f\n", x );
}

void main()
{
    clock_t start_time, end_time;

    start_time = clock();
    compute();
    end_time = clock();
    printf( "Execution time was %lu seconds\n",
            (end_time - start_time) / _CLOCKS_PER_SEC_ );
}

Classification: ANSI

Systems: All, Netware
Synopsis:  
#include <io.h>
int close( int handle );
int _close( int handle );

Description:  The close function closes a file at the operating system level. The handle value is the file handle returned by a successful execution of one of the creat, dup, dup2, open or sopen functions.

The _close function is identical to close. Use _close for ANSI/ISO naming conventions.

Returns:  The close function returns zero if successful. Otherwise, it returns -1 and errno is set to indicate the error.

Errors:  When an error has occurred, errno contains a value indicating the type of error that has been detected.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBADF</td>
<td>The handle argument is not a valid file handle.</td>
</tr>
</tbody>
</table>

See Also:  creat, dup, dup2, open, sopen

Example:  
#include <fcntl.h>
#include <io.h>

void main()
{
    int handle;

    handle = open( "file", O_RDONLY );
    if( handle != -1 ) {
        /* process file */
        close( handle );
    }
}

Classification:  close is POSIX 1003.1
_close is not POSIX
_close conforms to ANSI/ISO naming conventions

Systems:  close - All, Netware
_close - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:       #include <direct.h>
int closedir( struct dirent *dirp );
int _wclosedir( struct _wdirent *dirp );

Description:   The closedir function closes the directory specified by dirp and frees the memory allocated by
opendir.

The _wclosedir function is identical to closedir except that it closes a directory of
wide-character filenames opened by _wopendir.

Returns:       The closedir function returns zero if successful, non-zero otherwise.

Errors:        When an error has occurred, errno contains a value indicating the type of error that has been detected.

Constant         Meaning
EBADF          The argument dirp does not refer to an open directory stream.

See Also:      _dos_find..., opendir, readdir, rewinddir

Example:       To get a list of files contained in the directory \watcom\h on your default disk:

#include <stdio.h>
#include <direct.h>

typedef struct {
    unsigned short twoseconds : 5; /* seconds / 2 */
    unsigned short minutes : 6;
    unsigned short hours : 5;
} ftime_t;

typedef struct {
    unsigned short day : 5;
    unsigned short month : 4;
    unsigned short year : 7;
} fdate_t;

void main()
{
    DIR *dirp;
    struct dirent *direntp;
    ftime_t *f_time;
    fdate_t *f_date;
dirp = opendir( "\\watcom\\h" );
if( dirp != NULL ) {
    for(;;) {
        direntp = readdir( dirp );
        if( direntp == NULL ) break;
        f_time = (ftime_t *)&direntp->d_time;
        f_date = (fdate_t *)&direntp->d_date;
        printf( "%-12s %d/%2.2d/%2.2d 
",
            direntp->d_name,
            f_date->year + 1980,
            f_date->month,
            f_date->day,
            f_time->hours,
            f_time->minutes,
            f_time->twosecs * 2 );
    }
    closedir( dirp );
}

Note the use of two adjacent backslash characters (\) within character-string constants to signify a single backslash.

**Classification:** closedir is POSIX 1003.1
_wclosedir is not POSIX

**Systems:**
- closedir - All, Netware
- _wclosedir - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <process.h>  
char *cmdname( char *buffer );

Description: The _cmdname function obtains a copy of the executing program’s pathname and places it in buffer.

Returns: If the pathname of the executing program cannot be determined then NULL is returned; otherwise the address of buffer is returned.

See Also: getcmd

Example:  
#include <stdio.h>  
#include <process.h>

void main()  
{
    char buffer[PATH_MAX];

    printf( "%s\n", _cmdname( buffer ) );
}

Classification: WATCOM

Systems: All, Netware
**Synopsis:**

```c
#include <float.h>
unsigned int _control87( unsigned int newcw,
                        unsigned int mask );
```

**Description:**

The `_control87` function updates the control word of the 8087/80287/80387/80486. If `mask` is zero, then the control word is not updated. If `mask` is non-zero, then the control word is updated with bits from `newcw` corresponding to every bit that is on in `mask`.

**Returns:**

The `_control87` function returns the new control word. The description of bits defined for the control word is found in the `<float.h>` header file.

**See Also:**

`_clear87`, `_controlfp`, `_finite`, `_fpreset`, `_status87`

**Example:**

```c
#include <stdio.h>
#include <float.h>

char *status[2] = { "disabled", "enabled" };

void main()
{
    unsigned int fp_cw = 0;
    unsigned int fp_mask = 0;
    unsigned int bits;

    fp_cw = _control87( fp_cw,
                        fp_mask );

    printf( "Interrupt Exception Masks\n" );
    bits = fp_cw & MCW_EM;
    printf( "  Invalid Operation exception %s\n",
            status[ (bits & EM_INVALID) == 0 ] );
    printf( "  Denormalized exception %s\n",
            status[ (bits & EM_DENORMAL) == 0 ] );
    printf( "  Divide-By-Zero exception %s\n",
            status[ (bits & EM_ZERODIVIDE) == 0 ] );
    printf( "  Overflow exception %s\n",
            status[ (bits & EM_OVERFLOW) == 0 ] );
    printf( "  Underflow exception %s\n",
            status[ (bits & EM_UNDERFLOW) == 0 ] );
    printf( "  Precision exception %s\n",
            status[ (bits & EM_PRECISION) == 0 ] );

    printf( "Infinity Control = " );
    bits = fp_cw & MCW_IC;
    if( bits == IC_AFFINE ) printf( "affine\n" );
    if( bits == IC_PROJECTIVE ) printf( "projective\n" );

    printf( "Rounding Control = " );
    bits = fp_cw & MCW_RC;
    if( bits == RC_NEAR )       printf( "near\n" );
    if( bits == RC_DOWN )       printf( "down\n" );
    if( bits == RC_UP )         printf( "up\n" );
    if( bits == RC_CHOP )       printf( "chop\n" );
```
printf( "Precision Control = " );
bits = fp_cw & MCW_PC;
if( bits == PC_24 )            printf( "24 bits\n" );
if( bits == PC_53 )            printf( "53 bits\n" );
if( bits == PC_64 )            printf( "64 bits\n" );
)

**Classification:** Intel

**Systems:** All, Netware
Synopsis: 

```c
#include <float.h>
unsigned int _controlfp( unsigned int newcw, 
                         unsigned int mask );
```

Description: The `_controlfp` function updates the control word of the 8087/80287/80387/80486. If `mask` is zero, then the control word is not updated. If `mask` is non-zero, then the control word is updated with bits from `newcw` corresponding to every bit that is on in `mask`.

Returns: The `_controlfp` function returns the new control word. The description of bits defined for the control word is found in the `<float.h>` header file.

See Also: `_clear87`, `_control87`, `_finite`, `_fpreset`, `_status87`

Example:

```c
#include <stdio.h>
#include <float.h>

char *status[2] = { "disabled", "enabled" };

void main()
{
    unsigned int fp_cw = 0;
    unsigned int fp_mask = 0;
    unsigned int bits;

    fp_cw = _controlfp( fp_cw, 
                         fp_mask );

    printf( "Interrupt Exception Masks\n" );
    bits = fp_cw & MCW_EM;
    printf( "  Invalid Operation exception %s\n", 
            status[ (bits & EM_INVALID) == 0 ] );
    printf( "  Denormalized exception %s\n", 
            status[ (bits & EM_DENORMAL) == 0 ] );
    printf( "  Divide-By-Zero exception %s\n", 
            status[ (bits & EM_ZERODIVIDE) == 0 ] );
    printf( "  Overflow exception %s\n", 
            status[ (bits & EM_OVERFLOW) == 0 ] );
    printf( "  Underflow exception %s\n", 
            status[ (bits & EM_UNDERFLOW) == 0 ] );
    printf( "  Precision exception %s\n", 
            status[ (bits & EM_PRECISION) == 0 ] );

    printf( "Infinity Control = " );
    bits = fp_cw & MCW_IC;
    if( bits == IC_AFFINE )     printf( "affine\n" );
    if( bits == IC_PROJECTIVE ) printf( "projective\n" );

    printf( "Rounding Control = " );
    bits = fp_cw & MCW_RC;
    if( bits == RC_NEAR )       printf( "near\n" );
    if( bits == RC_DOWN )       printf( "down\n" );
    if( bits == RC_UP )         printf( "up\n" );
    if( bits == RC_CHOP )       printf( "chop\n" );
```
printf( "Precision Control = " );
bits = fp_cw & MCW_PC;
if( bits == PC_24 ) printf( "24 bits\n" );
if( bits == PC_53 ) printf( "53 bits\n" );
if( bits == PC_64 ) printf( "64 bits\n" );
}

Classification: Intel

Systems: All, Netware
Synopsis:  
#include <math.h>  
double cos( double x );

Description:  The \texttt{cos} function computes the cosine of \( x \) (measured in radians). A large magnitude argument may yield a result with little or no significance.

Returns:  The \texttt{cos} function returns the cosine value.

See Also:  \texttt{acos}, \texttt{sin}, \texttt{tan}

Example:  
#include <math.h>

void main()
{
    double value;
    value = cos( 3.1415278 );
}

Classification:  ANSI

Systems:  Math
Synopsis:  
#include <math.h>  
double cosh( double x );

Description:  The `cosh` function computes the hyperbolic cosine of \( x \). A range error occurs if the magnitude of \( x \) is too large.

Returns:  The `cosh` function returns the hyperbolic cosine value. When the argument is outside the permissible range, the `matherr` function is called. Unless the default `matherr` function is replaced, it will set the global variable `errno` to `ERANGE`, and print a "RANGE error" diagnostic message using the `stderr` stream.

See Also:  `sinh`, `tanh`, `matherr`

Example:  
```c
#include <stdio.h>
#include <math.h>

void main()
{
    printf( "%f\n", cosh(.5) );
}
```
produces the following:

1.127626

Classification:  ANSI

Systems:  Math
Synopsis:    #include <conio.h>
      int cprintf( const char *format, ... );

Description: The cprintf function writes output directly to the console under control of the argument format. The putchar function is used to output characters to the console. The format string is described under the description of the printf function.

Returns:    The cprintf function returns the number of characters written.

See Also:   _bprintf, fprintf, printf, sprintf, _vbprintf, vcprintf, vfprintf, vprintf, vsprintf

Example:    #include <conio.h>

      void main()
      {
        char *weekday, *month;
        int day, year;

        weekday = "Saturday";
        month = "April";
        day = 18;
        year = 1987;
        cprintf( "%s, %s %d, %d\n",
                   weekday, month, day, year );
      }

      produces the following:

            Saturday, April 18, 1987

Classification: WATCOM

Systems:    All, Netware

140  Library Functions and Macros
Synopsis:  
#include <conio.h>  
int cputs( const char *buf );

Description:  The `cputs` function writes the character string pointed to by `buf` directly to the console using the `putch` function. Unlike the `puts` function, the carriage-return and line-feed characters are not appended to the string. The terminating null character is not written.

Returns:  The `cputs` function returns a non-zero value if an error occurs; otherwise, it returns zero. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also:  `fputs`, `putch`, `puts`

Example:  
#include <conio.h>

void main()
{
    char buffer[82];
    buffer[0] = 80;
    cgets( buffer );
    cputs( &buffer[2] );
    putch( '\r' );
    putch( '\n' );
}

Classification:  WATCOM

Systems:  All, Netware
**Synopsis:**
```
#include <sys/types.h>
#include <sys/stat.h>
#include <io.h>

int creat( const char *path, int mode );
int _creat( const char *path, int mode );
int _wcreat( const wchar_t *path, int mode );
```

**Description:**
The `creat` function creates (and opens) a file at the operating system level. It is equivalent to:
```
open( path, O_WRONLY | O_CREAT | O_TRUNC, mode );
```

The `_creat` function is identical to `creat`. Use `_creat` for ANSI naming conventions.

The `_wcreat` function is identical to `creat` except that it accepts a wide character string argument.

The name of the file to be created is given by `path`. When the file exists (it must be writeable), it is truncated to contain no data and the preceding `mode` setting is unchanged.

When the file does not exist, it is created with access permissions given by the `mode` argument. The access permissions for the file or directory are specified as a combination of bits (defined in the `<sys/stat.h>` header file).

The following bits define permissions for the owner.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_IRWXU</td>
<td>Read, write, execute/search</td>
</tr>
<tr>
<td>S_IRUSR</td>
<td>Read permission</td>
</tr>
<tr>
<td>S_IWUSR</td>
<td>Write permission</td>
</tr>
<tr>
<td>S_IXUSR</td>
<td>Execute/search permission</td>
</tr>
</tbody>
</table>

The following bits define permissions for the group.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_IRWXG</td>
<td>Read, write, execute/search</td>
</tr>
<tr>
<td>S_IRGRP</td>
<td>Read permission</td>
</tr>
<tr>
<td>S_IWGRP</td>
<td>Write permission</td>
</tr>
<tr>
<td>S_IXGRP</td>
<td>Execute/search permission</td>
</tr>
</tbody>
</table>

The following bits define permissions for others.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_IRWXO</td>
<td>Read, write, execute/search</td>
</tr>
<tr>
<td>S_IROTH</td>
<td>Read permission</td>
</tr>
<tr>
<td>S_IWOTH</td>
<td>Write permission</td>
</tr>
<tr>
<td>S_IXOTH</td>
<td>Execute/search permission</td>
</tr>
</tbody>
</table>

The following bits define miscellaneous permissions used by other implementations.
Permission | Meaning
---|---
S_IREAD | is equivalent to S_IRUSR (read permission)
S_IWRITE | is equivalent to S_IWUSR (write permission)
S_IEXEC | is equivalent to S_IXUSR (execute/search permission)

All files are readable with DOS; however, it is a good idea to set S_IREAD when read permission is intended for the file.

**Returns:**
If successful, `creat` returns a handle for the file. When an error occurs while opening the file, -1 is returned, and `errno` is set to indicate the error.

**Errors:**
When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EACCES</td>
<td>Access denied because <code>path</code> specifies a directory or a volume ID, or a read-only file.</td>
</tr>
<tr>
<td>EMFILE</td>
<td>No more handles available (too many open files).</td>
</tr>
<tr>
<td>ENOENT</td>
<td>The specified <code>path</code> does not exist or <code>path</code> is an empty string.</td>
</tr>
</tbody>
</table>

**See Also:**
chsize, close, dup, dup2, eof, exec,..., fdopen, filesize, fileno, fstat, _grow_handles, isatty, lseek, open, read, setmode, sopen, stat, tell, write, umask

**Example:**
```c
#include <sys/types.h>
#include <sys/stat.h>
#include <io.h>

void main()
{
    int handle;

    handle = creat( "file", S_IWRITE | S_IREAD );
    if( handle != -1 ) {
        /* process file */
        close( handle );
    }
}
```

**Classification:**
`creat` is POSIX 1003.1
`_creat` is not POSIX
`_wcreat` is not POSIX

**Systems:**
`creat` - All, Netware
`_creat` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
`_wcreat` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:
#include <conio.h>
int cscanf( const char *format, ... );

Description: The cscanf function scans input from the console under control of the argument format. Following
the format string is a list of addresses to receive values. The cscanf function uses the function
getche to read characters from the console. The format string is described under the description of
the scanf function.

Returns: The cscanf function returns EOF when the scanning is terminated by reaching the end of the input
stream. Otherwise, the number of input arguments for which values were successfully scanned and
stored is returned. When a file input error occurs, the errno global variable may be set.

See Also: fscanf, scanf, sscanf, vcscanf, vfscanf, vscanf, vsscanf

Example: To scan a date in the form "Saturday April 18 1987":

#include <conio.h>

void main()
{
    int day, year;
    char weekday[10], month[10];

    cscanf( "%s %s %d %d",
            weekday, month, &day, &year );
    cprintf( "\n%s, %s %d, %d
",
            weekday, month, day, year );
}

Classification: WATCOM

Systems: All, Netware
ctime Functions

Synopsis:

```c
#include <time.h>
char * ctime( const time_t *timer );
char * _ctime( const time_t *timer, char *buf );
wchar_t * _wctime( const time_t *timer );
wchar_t * __wctime( const time_t *timer, wchar_t *buf );
```

Safer C:
The Safer C Library extension provides the ctime_s function which is a safer alternative to ctime. This newer ctime_s function is recommended to be used instead of the traditional "unsafe" ctime function.

Description:
The ctime functions convert the calendar time pointed to by timer to local time in the form of a string. The ctime function is equivalent to

```c
asctime( localtime( timer ) )
```

The ctime functions convert the time into a string containing exactly 26 characters. This string has the form shown in the following example:

```
Sat Mar 21 15:58:27 1987
```

All fields have a constant width. The new-line character ‘\n’ and the null character ‘\0’ occupy the last two positions of the string.

The ANSI function ctime places the result string in a static buffer that is re-used each time ctime or asctime is called. The non-ANSI function _ctime places the result string in the buffer pointed to by buf.

The wide-character function _wctime is identical to ctime except that it produces a wide-character string (which is twice as long). The wide-character function __wctime is identical to _ctime except that it produces a wide-character string (which is twice as long).

Whenever the ctime functions are called, the tzset function is also called.

The calendar time is usually obtained by using the time function. That time is Coordinated Universal Time (UTC) (formerly known as Greenwich Mean Time (GMT)).

The time set on the computer with the DOS time command and the DOS date command reflects the local time. The environment variable TZ is used to establish the time zone to which this local time applies. See the section The TZ Environment Variable for a discussion of how to set the time zone.

Returns:
The ctime functions return the pointer to the string containing the local time.

See Also:
asctime Functions, asctime_s, clock, ctime_s, difftime, gmtime, gmtime_s, localtime, localtime_s, mktime, strftime, time, tzset

Example:
```c
#include <stdio.h>
#include <time.h>

void main()
{
    time_t time_of_day;
    auto char buf[26];
```
ctime Functions

```c
  time_of_day = time( NULL );
  printf( "It is now: %s", _ctime( &time_of_day, buf ) );
```

produces the following:

```
  It is now: Fri Dec 25 15:58:42 1987
```

**Classification:**
- ctime is ANSI
- _ctime is not ANSI
- _wctime is not ANSI
- __wctime is not ANSI

**Systems:**
- ctime - All, Netware
- _ctime - All
- _wctime - All
- __wctime - All
Synopsis:
#define __STDC_WANT_LIB_EXT1__ 1
#include <time.h>
errno_t ctime_s( char * s,
rsize_t maxsize,
const time_t * timer);
#include <wchar.h>
errno_t _wctime_s( wchar_t * s,
rsize_t maxsize,
const time_t * timer);

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and ctime_s will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither s nor timer shall be a null pointer. maxsize shall not be less than 26 and shall not be greater than RSIZE_MAX. If there is a runtime-constraint violation, s[0] is set to a null character if s is not a null pointer and maxsize is not equal zero and is not greater than RSIZE_MAX.

Description: The ctime_s function converts the calendar time pointed to by timer to local time in the form of a string. It is equivalent to

    asctime_s( s, maxsize, localtime_s( timer ) )

Recommended practice:

The strftime function allows more flexible formatting and supports locale-specific behavior. If you do not require the exact form of the result string produced by the ctime_s function, consider using the strftime function instead.

Returns: The ctime_s function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

See Also: asctime Functions, asctime_s, clock, ctime Functions, difftime, gmtime, gmtime_s, localtime, localtime_s, mktime, strftime, time, tzset

Example:
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <time.h>

void main()
{
    time_t time_of_day;
    auto char buf[26];

    time_of_day = time( NULL );
    ctime_s( buf, sizeof( buf ), &time_of_day );
    printf( "It is now: %s", buf );
}

produces the following:

It is now: Mon Jan 30 14:29:55 2006

Classification: ctime_s is TR 24731
ctime_s, _wctime_s

_wctime_s is not TR 24731

Systems:
ctime_s - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, Netware
_wctime_s - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis: #include <process.h>
int cwait( int *status, int process_id, int action );

Description: The cwait function suspends the calling process until the specified process terminates.

If status is not NULL, it points to a word that will be filled in with the termination status word and return code of the terminated child process.

If the child process terminated normally, then the low order byte of the status word will be set to 0, and the high order byte will contain the low order byte of the return code that the child process passed to the DOSEXIT function. The DOSEXIT function is called whenever main returns, or exit or _exit are explicitly called.

If the child process did not terminate normally, then the high order byte of the status word will be set to 0, and the low order byte will contain one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hard-error abort</td>
</tr>
<tr>
<td>2</td>
<td>Trap operation</td>
</tr>
<tr>
<td>3</td>
<td>SIGTERM signal not intercepted</td>
</tr>
</tbody>
</table>

Note: This implementation of the status value follows the OS/2 model and differs from the Microsoft implementation. Under Microsoft, the return code is returned in the low order byte and it is not possible to determine whether a return code of 1, 2, or 3 imply that the process terminated normally. For portability to Microsoft compilers, you should ensure that the application that is waited on does not return one of these values. The following shows how to handle the status value in a portable manner.
The `process_id` argument specifies which process to wait for. Under Win32, any process can wait for any other process for which the process id is known. Under OS/2, a process can wait for any of its child processes. For example, a process id is returned by certain forms of the `spawn` function that is used to start a child process.

The `action` argument specifies when the parent process resumes execution. This argument is ignored in Win32, but is accepted for compatibility with OS/2 (although Microsoft handles the `status` value differently from OS/2!). The possible values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>WAIT_CHILD</code></td>
<td>Wait until the specified child process has ended.</td>
</tr>
<tr>
<td><code>WAIT_GRANDCHILD</code></td>
<td>Wait until the specified child process and all of the child processes of that child process have ended.</td>
</tr>
</tbody>
</table>

Under Win32, there is no parent-child relationship.

### Returns:

The `cwait` function returns the (child’s) process id if the (child) process terminated normally. Otherwise, `cwait` returns `-1` and sets `errno` to one of the following values:
cwait

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EINVAL</td>
<td>Invalid action code</td>
</tr>
<tr>
<td>ECHILD</td>
<td>Invalid process id, or the child does not exist.</td>
</tr>
<tr>
<td>EINTR</td>
<td>The child process terminated abnormally.</td>
</tr>
</tbody>
</table>

See Also: exit, _exit, spawn..., wait

Example:

```c
#include <stdio.h>
#include <process.h>

void main()
{
    int   process_id;
    int   status;

    process_id = spawnl( P_NOWAIT, "child.exe",
                         "child", "parm",  NULL );
    cwait( &status, process_id, WAIT_CHILD );
}
```

Classification: WATCOM

Systems: Win32, OS/2 1.x(all), OS/2-32
delay

Synopsis:    #include <i86.h>
            void delay( unsigned milliseconds );

Description:  The delay function suspends execution by the specified number of milliseconds.

Returns:     The delay function has no return value.

See Also:    sleep

Example:     #include <i86.h>

            void main()
            {
                sound( 200 );
                delay( 500 ); /* delay for 1/2 second */
                nosound();
            }

Classification: WATCOM

Systems:      All, Netware
Synopsis: 
#include <math.h>
extern int _dieeetomsbin( double *src, double *dest );

Description: The _dieeetomsbin function loads the double pointed to by src in IEEE format and converts it to Microsoft binary format, storing the result into the double pointed to by dest.

For _dieeetomsbin, IEEE Nan’s and Infinities will cause overflow. IEEE denormals will be converted if within range. Otherwise, they will be converted to 0 in the Microsoft binary format.

The range of Microsoft binary format floats is 2.938736e-39 to 1.701412e+38. The range of Microsoft binary format doubles is 2.938735877056e-39 to 1.701411834605e+38.

Microsoft Binary Format was used by early versions of Microsoft QuickBASIC before coprocessors became standard.

Returns: The _dieeetomsbin function returns 0 if the conversion was successful. Otherwise, it returns 1 if conversion would cause an overflow.

See Also: _dmsbintoieee, _fieeetomsbin, _fmsbintoieee

Example: 
#include <stdio.h>
#include <math.h>

void main() { 
    float fieee, fmsb;
    double dieee, dmsb;

    fieee = 0.5;
    dieee = -2.0;

    /* Convert IEEE format to Microsoft binary format */
    _fieeetomsbin( &fieee, &fmsb );
    _dieeetomsbin( &dieee, &dmsb );

    /* Convert Microsoft binary format back to IEEE format */
    _fmsbintoieee( &fmsb, &fieee );
    _dmsbintoieee( &dmsb, &dieee );

    /* Display results */
    printf( "%f, %f\n", fieee, dieee );
}

produces the following:

fieee = 0.500000, dieee = -2.000000

Classification: WATCOM

Systems: All, Netware
# difftime

## Synopsis:
```
#include <time.h>
double difftime( time_t time1, time_t time0 );
```

## Description:
The `difftime` function calculates the difference between the two calendar times:
```
time1 - time0
```

## Returns:
The `difftime` function returns the difference between the two times in seconds as a `double`.

## See Also:
`asctime` Functions, `asctime_s`, `clock`, `ctime` Functions, `ctime_s`, `gmtime`, `gmtime_s`, `localtime`, `localtime_s`, `mktime`, `strftime`, `time`, `tzset`

## Example:
```
#include <stdio.h>
#include <time.h>

void compute( void );

int main()
{
    time_t start_time, end_time;

    start_time = time( NULL );
    compute();
    end_time = time( NULL );
    printf( "Elapsed time: %f seconds\n",
            difftime( end_time, start_time ) );
}

void compute( void )
{
    int i, j;

    for( i = 1; i <= 20; i++ ) {
        for( j = 1; j <= 20; j++ )
            printf( "%3d ", i * j );
        printf( "\n" );
    }
}
```

## Classification: ANSI

## Systems: Math
Synopsis:
#include <libgen.h>
char *dirname( char *path );

Description: The `dirname` function takes a pointer to a character string that contains a pathname, and returns a
tpointer to a string that is a pathname of the parent directory of that file. Trailing path separators are not
considered as part of the path.

The `dirname` function may modify the string pointed to by `path` and may return a pointer to static
storage that may be overwritten by a subsequent call to `dirname`.

The `dirname` function is not re-entrant or thread-safe.

Returns: The `dirname` function returns a pointer to a string that is the parent directory of `path`. If `path` is a null
pointer or points to an empty string, a pointer to the string "." is returned.

See Also: basename

Example: #include <stdio.h>
#include <libgen.h>

int main( void )
{
    puts( dirname( "/usr/lib" ) );
    puts( dirname( "/usr/" ) );
    puts( dirname( "usr" ) );
    puts( dirname( "/" ) );
    puts( dirname( ".." ) );
    return( 0 );
}

produces the following:

/usr
/
.
/
.

Classification: POSIX

Systems: All, Netware
**Synopsis:**

```c
#include <i86.h>
void _disable( void );
```

**Description:** The `_disable` function causes interrupts to become disabled.

The `_disable` function would be used in conjunction with the `_enable` function to make sure that a sequence of instructions are executed without any intervening interrupts occurring.

**Returns:** The `_disable` function returns no value.

**See Also:** `_enable`

**Example:**

```c
#include <stdio.h>
#include <stdlib.h>
#include <i86.h>

struct list_entry {
    struct list_entry *next;
    int    data;
};
volatile struct list_entry *ListHead = NULL;
volatile struct list_entry *ListTail = NULL;

void insert( struct list_entry *new_entry ) {
    /* insert new_entry at end of linked list */
    new_entry->next = NULL;
    _disable();       /* disable interrupts */
    if( ListTail == NULL ) {
        ListHead = new_entry;
    } else {
        ListTail->next = new_entry;
    }
    ListTail = new_entry;
    _enable();        /* enable interrupts now */
}

void main() {
    struct list_entry *p;
    int i;

    for( i = 1; i <= 10; i++ ) {
        p = (struct list_entry *)
            malloc( sizeof( struct list_entry ) );
        if( p == NULL ) break;
        p->data = i;
        insert( p );
    }
}
```

**Classification:** Intel

**Systems:** All, Netware
Synopsis:  
```
#include <graph.h>
short _FAR _displaycursor( short mode );
```

Description:  The _displaycursor function is used to establish whether the text cursor is to be displayed when graphics functions complete. On entry to a graphics function, the text cursor is turned off. When the function completes, the mode setting determines whether the cursor is turned back on. The mode argument can have one of the following values:

- **_GCURSORON** the cursor will be displayed
- **_GCURSOROFF** the cursor will not be displayed

Returns:  The _displaycursor function returns the previous setting for mode.

See Also:  __gettextcursor__, __settextcursor__

Example:  
```
#include <stdio.h>
#include <graph.h>

main()
{
    char buf[ 80 ];
    
    _setvideomode( _TEXTC80 );
    _settextposition( 2, 1 );
    _displaycursor( _GCURSORON );
    _outtext( "Cursor ON\n\nEnter your name >" );
    gets( buf );
    _displaycursor( _GCURSOROFF );
    _settextposition( 6, 1 );
    _outtext( "Cursor OFF\n\nEnter your name >" );
    gets( buf );
    _setvideomode( _DEFAULTMODE );
}
```

Classification:  _displaycursor is PC Graphics

Systems:  DOS, QNX
Synopsis:    #include <stdlib.h>
    div_t div( int numer, int denom );

typedef struct {
    int quot;     /* quotient */
    int rem;      /* remainder */
} div_t;

Description:  The div function calculates the quotient and remainder of the division of the numerator numer by the denominator denom.

Returns:      The div function returns a structure of type div_t which contains the fields quot and rem.

See Also:     ldiv, lldiv, imaxdiv

Example:      #include <stdio.h>
              #include <stdlib.h>

              void print_time( int seconds )
              {
                  div_t   min_sec;

                  min_sec = div( seconds, 60 );
                  printf( "It took %d minutes and %d seconds\n",
                          min_sec.quot, min_sec.rem );
              }

              void main( void )
              {
                  print_time( 130 );
              }

              produces the following:

              It took 2 minutes and 10 seconds

Classification:  ISO C90

Systems:        All, Netware
Synopsis:
#include <math.h>
extern int _dmsbintoieee( double *src, double *dest );

Description: The _dmsbintoieee function loads the double pointed to by src in Microsoft binary format and converts it to IEEE format, storing the result into the double pointed to by dest.

The range of Microsoft binary format floats is $2.938736 \times 10^{-39}$ to $1.701412 \times 10^{38}$. The range of Microsoft binary format doubles is $2.938735877056 \times 10^{-39}$ to $1.701411834605 \times 10^{38}$.

Microsoft Binary Format was used by early versions of Microsoft QuickBASIC before coprocessors became standard.

Returns: The _dmsbintoieee function returns 0 if the conversion was successful. Otherwise, it returns 1 if conversion would cause an overflow.

See Also: _dieeetomsbin, _fieeetomsbin, _fmsbintoieee

Example: #include <stdio.h>
#include <math.h>

void main()
{
    float fieee, fmsb;
    double dieee, dmsb;

    fieee = 0.5;
    dieee = -2.0;

    /* Convert IEEE format to Microsoft binary format */
    _fieeetomsbin( &fieee, &fmsb );
    _dieeetomsbin( &dieee, &dmsb );

    /* Convert Microsoft binary format back to IEEE format */
    _fmsbintoieee( &fmsb, &fieee );
    _dmsbintoieee( &dmsb, &dieee );

    /* Display results */
    printf( "fieee = %f, dieee = %f\n", fieee, dieee );
}

produces the following:
fieee = 0.500000, dieee = -2.000000

Classification: WATCOM

Systems: All, Netware
Synopsis:
#include <dos.h>
#if defined(__NT__) || \n ( defined(__OS2__) && \n (defined(__386__) || defined(__PPC__))) 
unsigned _dos_allocmem( unsigned size, 
    void * *segment);
#else
unsigned _dos_allocmem( unsigned size, 
    unsigned *segment);
#endif

Description: The _dos_allocmem function uses system call 0x48 to allocate size paragraphs directly from DOS. The size of a paragraph is 16 bytes. The allocated memory is always paragraph aligned. The segment descriptor for the allocated memory is returned in the word pointed to by segment. If the allocation request fails, the maximum number of paragraphs that can be allocated is returned in this word instead.

For 32-bit DOS applications, it is recommended that the corresponding DPMI services be used.

Returns: The _dos_allocmem function returns zero if successful. Otherwise, it returns an OS error code and sets errno accordingly.

See Also: alloca, calloc, _dos_freemem, _dos_setblock, halloc, malloc

Example:  
#include <stdio.h>
#include <dos.h>

void main( void )
{
#if defined(__NT__) || \n ( defined(__OS2__) && \n (defined(__386__) || defined(__PPC__))) 
    void *segment;
#else
    unsigned segment;
#endif
/* Try to allocate 100 paragraphs, then free them */
if( _dos_allocmem( 100, &segment ) != 0 ) {
    printf( "allocmem failed\n" );
    printf( "Only %u paragraphs available\n", segment );
} else {
    printf( "allocmem succeeded\n" );
    if( _dos_freemem( segment ) != 0 ) {
        printf( "freemem failed\n" );
    } else {
        printf( "freemem succeeded\n" );
    }
}
//}

Classification: DOS

Systems: DOS, Win32, OS/2 1.x(all), OS/2-32, DOS/PM
Synopsis:  
#include <dos.h>
unsigned _dos_close( int handle );

Description:  The _dos_close function uses system call 0x3E to close the file indicated by handle. The value for handle is the one returned by a function call that created or last opened the file.

Returns: The _dos_close function returns zero if successful. Otherwise, it returns an OS error code and sets errno accordingly.

See Also:  creat, _dos_creat, _dos_creatnew, _dos_open, dup, fclose, open

Example:  
#include <stdio.h>
#include <dos.h>
#include <fcntl.h>

void main()
{
    int handle;

    /* Try to open "stdio.h" and then close it */
    if( _dos_open( "stdio.h", O_RDONLY, &handle ) != 0 ){
        printf( "Unable to open file\n" );
    } else {
        printf( "Open succeeded\n" );
        if( _dos_close( handle ) != 0 ) {
            printf( "Close failed\n" );
        } else {
            printf( "Close succeeded\n" );
        }
    }
}

Classification: DOS

Systems:  DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM
Synopsis:  
#include <dos.h>  
unsigned _dos_commit( int handle );

Description:  
The _dos_commit function uses system call 0x68 to flush to disk the DOS buffers associated with the file indicated by handle. It also forces an update on the corresponding disk directory and the file allocation table.

Returns:  
The _dos_commit function returns zero if successful. Otherwise, it returns an OS error code and sets errno accordingly.

See Also:  
_dos_close, _dos_creat, _dos_open, _dos_write

Example:  
#include <stdio.h>  
#include <dos.h>  
#include <fcntl.h>

void main()
{
    int handle;

    if( _dos_open( "file", O_RDONLY, handle ) != 0 ) {  
        printf( "Unable to open file\n" );  
    } else {  
        if( _dos_commit( handle ) == 0 ) {  
            printf( "Commit succeeded.\n" );  
        }  
        _dos_close( handle );  
    }
}

produces the following:

Commit succeeded.

Classification:  
DOS

Systems:  
DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM
Synopsis:    

```
#include <dos.h>
unsigned _dos_creat( const char *path, 
                    unsigned attribute, 
                    int *handle );
```

Description: The _dos_creat function uses system call 0x3C to create a new file named path, with the access attributes specified by attribute. The handle for the new file is returned in the word pointed to by handle. If the file already exists, the contents will be erased, and the attributes of the file will remain unchanged. The possible values for attribute are:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>_A_NORMAL</td>
<td>Indicates a normal file. File can be read or written without any restrictions.</td>
</tr>
<tr>
<td>_A_RDONLY</td>
<td>Indicates a read-only file. File cannot be opened for &quot;write&quot;.</td>
</tr>
<tr>
<td>_A_HIDDEN</td>
<td>Indicates a hidden file. This file will not show up in a normal directory search.</td>
</tr>
<tr>
<td>_A_SYSTEM</td>
<td>Indicates a system file. This file will not show up in a normal directory search.</td>
</tr>
</tbody>
</table>

Returns: The _dos_creat function returns zero if successful. Otherwise, it returns an OS error code and sets errno accordingly.

See Also: creat, _dos_creatnew, _dos_open, _dos_open, open, fdopen, fopen, freopen, _fsopen, _grow_handles, _hdopen, open, _open_osfhandle, _popen, sopen

Example:    
```
#include <stdio.h>
#include <dos.h>

void main()
{
    int handle;

    if( _dos_creat( "file", _A_NORMAL, &handle ) != 0 ){
        printf( "Unable to create file\n" );
    } else {
        printf( "Create succeeded\n" );
        _dos_close( handle );
    }
}
```

Classification: DOS

Systems:    DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM
Synopsis:  
#include <dos.h>  
unsigned _dos_creatnew( const char *path,  
                      unsigned attribute,  
                      int *handle );

Description:  The _dos_creatnew function uses system call 0x5B to create a new file named path, with the access attributes specified by attribute. The handle for the new file is returned in the word pointed to by handle. If the file already exists, the create will fail. The possible values for attribute are:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>_A_NORMAL</td>
<td>Indicates a normal file. File can be read or written without any restrictions.</td>
</tr>
<tr>
<td>_A_RDONLY</td>
<td>Indicates a read-only file. File cannot be opened for &quot;write&quot;.</td>
</tr>
<tr>
<td>_A_HIDDEN</td>
<td>Indicates a hidden file. This file will not show up in a normal directory search.</td>
</tr>
<tr>
<td>_A_SYSTEM</td>
<td>Indicates a system file. This file will not show up in a normal directory search.</td>
</tr>
</tbody>
</table>

Returns:  The _dos_creatnew function returns zero if successful. Otherwise, it returns an OS error code and sets errno. Possible values and their interpretations:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EACCES</td>
<td>Access denied because the directory is full, or the file exists and cannot be overwritten.</td>
</tr>
<tr>
<td>EEXIST</td>
<td>File already exists</td>
</tr>
<tr>
<td>EMFILE</td>
<td>No more handles available (i.e., too many open files)</td>
</tr>
<tr>
<td>ENOENT</td>
<td>Path or file not found</td>
</tr>
</tbody>
</table>

See Also:  creat, _dos_creat, _dos_open, _dos_openopen, fdopen, fopen, freopen, _fsopen,  
_grow_handles, _hdopen, open, _open_osfhandle, _popen, sopen

Example:  #include <stdio.h>  
#include <dos.h>

void main()  
{  
  int handle1, handle2;
  if( _dos_creat( "file", _A_NORMAL, &handle1 ) )  
    printf( "Unable to create file\n" );
  else  
    printf( "Create succeeded\n" );
  if( _dos_creatnew( "file", _A_NORMAL, &handle2 ) )  
    printf( "Unable to create new file\n" );
 }  
_dos_close( handle1 );
}

Classification:  DOS

164  Library Functions and Macros
Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM
Synopsis:  
#include <dos.h>
int dosexterr( struct DOSERROR *err_info );

struct _DOSERROR {
    int exterror; /* contents of AX register */
    char errclass; /* contents of BH register */
    char action; /* contents of BL register */
    char locus; /* contents of CH register */
};

Description:  The dosexterr function extracts extended error information following a failed DOS function. This information is placed in the structure located by err_info. This function is only useful with DOS version 3.0 or later.

You should consult the technical documentation for the DOS system on your computer for an interpretation of the error information.

Returns:  The dosexterr function returns an unpredictable result when the preceding DOS call did not result in an error. Otherwise, dosexterr returns the number of the extended error.

See Also:  perror

Example:  
#include <stdio.h>
#include <dos.h>
#include <fcntl.h>

struct _DOSERROR dos_err;

void main()
{
    int handle;

    /* Try to open "stdio.h" and then close it */
    if( _dos_open( "stdio.h", O_RDONLY, &handle ) != 0 ){
        dosexterr( &dos_err );
        printf( "Unable to open file\n" );
        printf( "exterror (AX) = %d", dos_err.exterror );
        printf( "errclass (BH) = %d", dos_err.errclass );
        printf( "action (BL) = %d", dos_err.action );
        printf( "locus (CH) = %d", dos_err.locus );
    } else {
        printf( "Open succeeded\n" );
        if( _dos_close( handle ) != 0 ) {
            printf( "Close failed\n" );
        } else {
            printf( "Close succeeded\n" );
        }
    }
}

produces the following:

166  Library Functions and Macros
Unable to open file
error (AX) = 2
errclass (BH) = 8
action (BL) = 3
locus (CH) = 2

Classification: DOS

Systems: DOS, Windows, Win386, DOS/PM
Synopsis:
#include <dos.h>
unsigned _dos_findfirst( const char *path,
                        unsigned attributes,
                        struct find_t *buffer );
unsigned _dos_findnext( struct find_t *buffer );
unsigned _dos_findclose( struct find_t *buffer );

struct find_t {
    char reserved[21];      /* reserved for use by DOS */
    char attrib;            /* attribute byte for file */
    unsigned short wr_time; /* time of last write to file */
    unsigned short wr_date; /* date of last write to file */
    unsigned long  size;    /* length of file in bytes */
#if defined(__OS2__) || defined(__NT__)
    char name[256];         /* null-terminated filename */
#else
    char name[13];          /* null-terminated filename */
#endif
};

unsigned _wdos_findfirst( const wchar_t *path,
                       unsigned attributes,
                       struct _wfind_t *buffer );
unsigned _wdos_findnext( struct _wfind_t *buffer );
unsigned _wdos_findclose( struct _wfind_t *buffer );

struct _wfind_t {
    char reserved[21];      /* reserved for use by DOS */
    char attrib;            /* attribute byte for file */
    unsigned short wr_time; /* time of last write to file */
    unsigned short wr_date; /* date of last write to file */
    unsigned long  size;    /* length of file in bytes */
#if defined(__OS2__) || defined(__NT__)
    wchar_t name[256];      /* null-terminated filename */
#else
    wchar_t name[13];       /* null-terminated filename */
#endif
};

Description: The _dos_findfirst function uses system call 0x4E to return information on the first file whose name and attributes match the path and attributes arguments. The information is returned in a find_t structure pointed to by buffer. The path argument may contain wildcard characters ("?" and "*"). The attributes argument may be any combination of the following constants:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>_A_NORMAL</td>
<td>Indicates a normal file. File can be read or written without any restrictions.</td>
</tr>
<tr>
<td>_A_RDONLY</td>
<td>Indicates a read-only file. File cannot be opened for &quot;write&quot;.</td>
</tr>
<tr>
<td>_A_HIDDEN</td>
<td>Indicates a hidden file. This file will not show up in a normal directory search.</td>
</tr>
<tr>
<td>_A_SYSTEM</td>
<td>Indicates a system file. This file will not show up in a normal directory search.</td>
</tr>
<tr>
<td>_A_VOLID</td>
<td>Indicates a volume-ID.</td>
</tr>
</tbody>
</table>
_A_SUBDIR  Indicates a sub-directory.

_A_ARCH    This is the archive flag. It is set whenever the file is modified, and is cleared by the
            MS-DOS BACKUP command and other backup utility programs.

The attributes argument is interpreted by DOS as follows:

1. If _A_NORMAL is specified, then normal files are included in the search.
2. If any of _A_HIDDEN, _A_SYSTEM, _A_SUBDIR are specified, then normal files and the
   specified type of files are included in the search.
3. If _A_VOLID is specified, then volume-ID’s are also included in the search. Note: The
   _A_VOLID attribute is not supported on systems other than DOS (e.g. Win32, OS/2).
4. _A_RDONLY and _A_ARCH are ignored by this function.

The format of the wr_time field is described by the following structure (this structure is not defined in
any Watcom header file).

typedef struct {
    unsigned short twosecs : 5;    /* seconds / 2 */
    unsigned short minutes : 6;    /* minutes (0,59) */
    unsigned short hours   : 5;    /* hours (0,23) */
} ftime_t;

The format of the wr_date field is described by the following structure (this structure is not defined in
any Watcom header file).

typedef struct {
    unsigned short day     : 5;    /* day (1,31) */
    unsigned short month   : 4;    /* month (1,12) */
    unsigned short year    : 7;    /* 0 is 1980 */
} fdate_t;

The _dos_findnext function uses system call 0x4F to return information on the next file whose
name and attributes match the pattern supplied to the _dos_findfirst function.

On some systems (e.g. Win32, OS/2), you must call _dos_findclose to indicate that you are done
matching files. This function deallocates any resources that were allocated by the _dos_findfirst
function. The wide-character _wdos_findclose, _wdos_findfirst and _wdos_findnext
functions are similar to their counterparts but operate on wide-character strings.

Returns: The _dos_find... functions return zero if successful. Otherwise, the _dos_findfirst and
_dos_findnext functions return an OS error code and set errno accordingly.

See Also: opendir, readdir, closedir

Example: #include <stdio.h>
#include <dos.h>

void main()
{
    struct find_t   fileinfo;
    unsigned        rc;         /* return code */
/* Display name and size of "*.c" files */
rc = _dos_findfirst( "*.c", _A_NORMAL, &fileinfo );
while( rc == 0 ) {
    printf( "%14s %10ld\n", fileinfo.name, fileinfo.size );
    rc = _dos_findnext( &fileinfo );
}  
#endif
_dos_findclose( &fileinfo );
#endif

Classification: DOS

Systems:  _dos_findclose - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM
_dos_findfirst - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM
_dos_findnext - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM
_wdos_findclose - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_wdos_findfirst - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_wdos_findnext - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

```c
#include <dos.h>
#if defined(__NT__) || 
( defined(__OS2__) && 
(defined(__386__) || defined(__PPC__)))
unsigned _dos_freemem( void *   segment );
#else
unsigned _dos_freemem( unsigned segment );
#endif
```

Description:
The `_dos_freemem` function uses system call 0x49 to release memory that was previously allocated by `_dos_allocmem`. The value contained in `segment` is the one returned by a previous call to `_dos_allocmem`.

For 32-bit DOS applications, it is recommended that the corresponding DPMI services be used.

Returns:
The `_dos_freemem` function returns zero if successful. Otherwise, it returns an OS error code and sets `errno` accordingly.

See Also:
 `_dos_allocmem`, `_dos_setblock`, `free`, `hfree`

Example:

```c
#include <stdio.h>
#include <dos.h>

void main( void )
{
#if defined(__NT__) || 
( defined(__OS2__) && 
(defined(__386__) || defined(__PPC__)))
void *segment;
#else
unsigned segment;
#endif
/* Try to allocate 100 paragraphs, then free them */
if( _dos_allocmem( 100, &segment ) != 0 ) {
    printf( "_dos_allocmem failed\n" );
    printf( "Only %u paragraphs available\n", segment );
} else {
    printf( "_dos_allocmem succeeded\n" );
    if( _dos_freemem( segment ) != 0 ) {
        printf( "_dos_freemem failed\n" );
    } else {
        printf( "_dos_freemem succeeded\n" );
    }
}
}
```

Classification: DOS

Systems: DOS, Win32, OS/2 1.x(all), OS/2-32, DOS/PM
 Synopsis: 
#include <dos.h> 
void _dos_getdate( struct dosdate_t *date );

 struct dosdate_t {
     unsigned char day;      /* 1-31 */
     unsigned char month;    /* 1-12 */
     unsigned short year;    /* 1980-2099 */
     unsigned char dayofweek;/* 0-6 (0=Sunday) */
 };

 Description: The _dos_getdate function uses system call 0x2A to get the current system date. The date information is returned in a dosdate_t structure pointed to by date.

 Returns: The _dos_getdate function has no return value.

 See Also: _dos_gettime, _dos_setdate, _dos_settime, gmtime, localtime, mktime, time

 Example: #include <stdio.h>
#include <dos.h>

 void main()
{ 
    struct dosdate_t date;
    struct dostime_t time;

    /* Get and display the current date and time */
    _dos_getdate( &date );
    _dos_gettime( &time );
    printf( "The date (MM-DD-YYYY) is: %d-%d-%d\n",
            date.month, date.day, date.year );
    printf( "The time (HH:MM:SS) is: %.2d:%.2d:%.2d\n",
            time.hour, time.minute, time.second );
}

 produces the following:

 The date (MM-DD-YYYY) is: 12-25-1989
 The time (HH:MM:SS) is: 14:23:57

 Classification: DOS

 Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM
Synopsis:

```c
#include <dos.h>

unsigned _dos_getdiskfree( unsigned drive,
                        struct diskfree_t *diskspace );
```

```c
struct diskfree_t {
    unsigned short total_clusters;
    unsigned short avail_clusters;
    unsigned short sectors_per_cluster;
    unsigned short bytes_per_sector;
};
```

Description: The `_dos_getdiskfree` function uses system call 0x36 to obtain useful information on the disk drive specified by `drive`. Specify 0 for the default drive, 1 for drive A, 2 for drive B, etc. The information about the drive is returned in the structure `diskfree_t` pointed to by `diskspace`.

Returns: The `_dos_getdiskfree` function returns zero if successful. Otherwise, it returns a non-zero value and sets `errno` to `EINVAL` indicating an invalid drive was specified.

See Also: `_dos_getdrive`, `_dos_setdrive`, `_getdiskfree`, `_getdrive`

Example:

```c
#include <stdio.h>
#include <dos.h>

void main()
{
    struct diskfree_t disk_data;

    /* get information about drive 3 (the C drive) */
    if( _dos_getdiskfree( 3, &disk_data ) == 0 ) {
        printf( "total clusters: %u\n",
                disk_data.total_clusters );
        printf( "available clusters: %u\n",
                disk_data.avail_clusters );
        printf( "sectors/cluster: %u\n",
                disk_data.sectors_per_cluster );
        printf( "bytes per sector: %u\n",
                disk_data.bytes_per_sector );
    } else {
        printf( "Invalid drive specified\n" );
    }
}
```

produces the following:

```
total clusters: 16335
available clusters: 510
sectors/cluster: 4
bytes per sector: 512
```

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM
**dos_getdrive**

**Synopsis:**
```c
#include <dos.h>
void _dos_getdrive( unsigned *drive );
```

**Description:** The `_dos_getdrive` function uses system call 0x19 to get the current disk drive number. The current disk drive number is returned in the word pointed to by `drive`. A value of 1 is drive A, 2 is drive B, 3 is drive C, etc.

**Returns:** The `_dos_getdrive` function has no return value.

**See Also:** `_dos_getdiskfree`, `_dos_setdrive`, `_getdiskfree`, `_getdrive`

**Example:**
```c
#include <stdio.h>
#include <dos.h>

void main()
{
    unsigned drive;

    _dos_getdrive( &drive );
    printf( "The current drive is %c\n",
            'A' + drive - 1 );
}
```

produces the following:

```
The current drive is C
```

**Classification:** DOS

**Systems:** DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM
Synopsis:  
#include <dos.h>  
unsigned _dos_getfileattr( const char *path,  
                        unsigned *attributes );

Description:  The _dos_getfileattr function uses system call 0x43 to get the current attributes of the file or directory that path points to. The possible attributes are:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>_A_NORMAL</td>
<td>Indicates a normal file. File can be read or written without any restrictions.</td>
</tr>
<tr>
<td>_A_RDONLY</td>
<td>Indicates a read-only file. File cannot be opened for &quot;write&quot;.</td>
</tr>
<tr>
<td>_A_HIDDEN</td>
<td>Indicates a hidden file. This file will not show up in a normal directory search.</td>
</tr>
<tr>
<td>_A_SYSTEM</td>
<td>Indicates a system file. This file will not show up in a normal directory search.</td>
</tr>
<tr>
<td>_A_VOLID</td>
<td>Indicates a volume-ID.</td>
</tr>
<tr>
<td>_A_SUBDIR</td>
<td>Indicates a sub-directory.</td>
</tr>
<tr>
<td>_A_ARCH</td>
<td>This is the archive flag. It is set whenever the file is modified, and is cleared by the MS-DOS BACKUP command and other backup utility programs.</td>
</tr>
</tbody>
</table>

Returns: The _dos_getfileattr function returns zero if successful. Otherwise, it returns an OS error code and sets errno accordingly.

See Also:  _dos_setfileattr

Example: #include <stdio.h>  
#include <dos.h>

print_attribute()  
{  
    unsigned attribute;

    _dos_getfileattr( "file", &attribute );
    printf( "File attribute is %d\n", attribute );
    if( attribute & _A_RDONLY ) {  
        printf( "This is a read-only file.\n" );
    } else {  
        printf( "This is not a read-only file.\n" );
    }
}

t void main()  
{  
    int    handle;
if( _dos_creat( "file", _A_RDONLY, &handle ) != 0 ) {
    printf( "Error creating file\n" );
}
print_attribute();
_dos_setfileattr( "file", _A_NORMAL );
print_attribute();
_dos_close( handle );

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM
Synopsis:  
#include <dos.h>
unsigned _dos_getftime( int handle,  
unsigned *date,  
unsigned *time );

Description:  The _dos_getftime function uses system call 0x57 to get the date and time that the file associated with handle was last modified. The date consists of the year, month and day packed into 16 bits as follows:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bits 0-4</td>
<td>Day (1-31)</td>
</tr>
<tr>
<td>bits 5-8</td>
<td>Month (1-12)</td>
</tr>
<tr>
<td>bits 9-15</td>
<td>Year (0-119 representing 1980-2099)</td>
</tr>
</tbody>
</table>

The time consists of the hour, minute and seconds/2 packed into 16 bits as follows:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bits 0-4</td>
<td>Seconds/2 (0-29)</td>
</tr>
<tr>
<td>bits 5-10</td>
<td>Minutes (0-59)</td>
</tr>
<tr>
<td>bits 11-15</td>
<td>Hours (0-23)</td>
</tr>
</tbody>
</table>

Returns: The _dos_getftime function returns zero if successful. Otherwise, it returns an OS error code and sets errno accordingly.

See Also:  
_dos_setftime

Example:  
#include <stdio.h>
#include <dos.h>
#include <fcntl1.h>

#define YEAR(t)   (((t & 0xFE00) >> 9) + 1980)
#define MONTH(t)  ((t & 0x01E0) >> 5)
#define DAY(t)    (t & 0x001F)
#define HOUR(t)   ((t & 0xF800) >> 11)
#define MINUTE(t) ((t & 0x07E0) >> 5)
#define SECOND(t) ((t & 0x001F) << 1)

void main( void )
{
    int handle;
    unsigned date, time;
if( _dos_open( "file", O_RDONLY, &handle ) != 0 ) {
    printf( "Unable to open file\n" );
} else {
    printf( "Open succeeded\n" );
    _dos_gettime( handle, &date, &time );
    printf( "The file was last modified on %d/%d/%d",
            MONTH(date), DAY(date), YEAR(date) );
    printf( " at %.2d:%.2d:%.2d\n",
            HOUR(time), MINUTE(time), SECOND(time) );
    _dos_close( handle );
}

produces the following:

Open succeeded
The file was last modified on 12/29/1989 at 14:32:46

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM
Synopsis: 
#include <dos.h>

void _dos_gettime( struct dostime_t *time );

struct dostime_t {
    unsigned char hour; /* 0-23 */
    unsigned char minute; /* 0-59 */
    unsigned char second; /* 0-59 */
    unsigned char hsecond; /* 1/100 second; 0-99 */
};

Description: The _dos_gettime function uses system call 0x2C to get the current system time. The time information is returned in a dostime_t structure pointed to by time.

Returns: The _dos_gettime function has no return value.

See Also: _dos_getdate, _dos_setdate, _dos_settime, gmtime, localtime, mktime, time

Example: #include <stdio.h>
#include <dos.h>

void main()
{
    struct dosdate_t date;
    struct dostime_t time;

    /* Get and display the current date and time */
    _dos_getdate( &date );
    _dos_gettime( &time );
    printf( "The date (MM-DD-YYYY) is: %d-%d-%d\n",
            date.month, date.day, date.year );
    printf( "The time (HH:MM:SS) is: %.2d:%.2d:%.2d\n",
            time.hour, time.minute, time.second );
}

produces the following:

The date (MM-DD-YYYY) is: 12-25-1989
The time (HH:MM:SS) is: 14:23:57

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM
_dos_getvect

Synopsis:    
#include <dos.h>
void (__interrupt __far *__dos_getvect(unsigned intnum))();

Description:  The _dos_getvect function gets the current value of interrupt vector number intnum.

Returns:  The _dos_getvect function returns a far pointer to the current interrupt handler for interrupt number intnum.

See Also:  _chain_intr,_dos_keep,_dos_setvect

Example:  
#include <stdio.h>
#include <dos.h>

volatile int clock_ticks;
void (__interrupt __far *prev_int_1c)();
#define BLIP_COUNT (5*18) /* 5 seconds */

void __interrupt __far timer_rtn()
{
    ++clock_ticks;
    _chain_intr( prev_int_1c );
}

int delays = 0;

int compile_a_line()
{
    if( delays > 15 ) return( 0 );
    delay( 1000 ); /* delay for 1 second */
    printf( "Delayed for 1 second\n" );
    delays++;
    return( 1 );
}

void main()
{
    prev_int_1c = _dos_getvect( 0x1c );
    _dos_setvect( 0x1c, timer_rtn );
    while( compile_a_line() )
    {
        if( clock_ticks >= BLIP_COUNT )
            putchar( '.' );
        clock_ticks -= BLIP_COUNT;
    }
    _dos_setvect( 0x1c, prev_int_1c );
}

Classification: WATCOM

Systems:  DOS, Windows, DOS/PM

180  Library Functions and Macros
**Synopsis:**

```c
#include <dos.h>
void _dos_keep( unsigned retcode, unsigned memsize );
```

**Description:** The `_dos_keep` function is used to install terminate-and-stay-resident programs ("TSR’s") in memory. The amount of memory kept for the program is `memsize` paragraphs (a paragraph is 16 bytes) from the Program Segment Prefix which is stored in the variable `_psp`. The value of `retcode` is returned to the parent process.

**Returns:** The `_dos_keep` function does not return.

**See Also:** `_chain_intr`, `_dos_getvect`, `_dos_setvect`

**Example:**

```c
#include <dos.h>

void permanent()
{
    /* . */
    /* . */
    /* . */
}

void transient()
{
    /* . */
    /* . */
    /* . */
}

void main()
{
    /* initialize our TSR */
    transient();
    /*
        now terminate and keep resident
        the non-transient portion
    */
    _dos_keep( 0, (FP_OFF( transient ) + 15) >> 4 );
}
```

**Classification:** DOS
Synopsis:  
#include <dos.h>
#include <fcntl.h>
#include <share.h>
unsigned _dos_open( const char *path,
                     unsigned mode,
                     int *handle );

Description:  The _dos_open function uses system call 0x3D to open the file specified by path, which must be an existing file.  The mode argument specifies the file’s access, sharing and inheritance permissions. The access mode must be one of:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>O_RDONLY</td>
<td>Read only</td>
</tr>
<tr>
<td>O_WRONLY</td>
<td>Write only</td>
</tr>
<tr>
<td>O_RDWR</td>
<td>Both read and write</td>
</tr>
</tbody>
</table>

The sharing permissions, if specified, must be one of:

<table>
<thead>
<tr>
<th>Permission</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SH_COMPAT</td>
<td>Set compatibility mode.</td>
</tr>
<tr>
<td>SH_DENYRW</td>
<td>Prevent read or write access to the file.</td>
</tr>
<tr>
<td>SH_DENYWR</td>
<td>Prevent write access of the file.</td>
</tr>
<tr>
<td>SH_DENYRD</td>
<td>Prevent read access to the file.</td>
</tr>
<tr>
<td>SH_DENYNO</td>
<td>Permit both read and write access to the file.</td>
</tr>
</tbody>
</table>

The inheritance permission, if specified, is:

<table>
<thead>
<tr>
<th>Permission</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>O_NOINHERIT</td>
<td>File is not inherited by a child process</td>
</tr>
</tbody>
</table>

Returns:  The _dos_open function returns zero if successful. Otherwise, it returns an MS-DOS error code and sets errno to one of the following values:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EACCESS</td>
<td>Access denied because path specifies a directory or a volume ID, or opening a read-only file for write access</td>
</tr>
<tr>
<td>EINVAL</td>
<td>A sharing mode was specified when file sharing is not installed, or access-mode value is invalid</td>
</tr>
<tr>
<td>EMFILE</td>
<td>No more handles available, (too many open files)</td>
</tr>
<tr>
<td>ENOENT</td>
<td>Path or file not found</td>
</tr>
</tbody>
</table>

See Also:  _dos_close, _dos_creat, _dos_creatnew, _dos_read, _dos_write, fdopen, fopen, freopen, _fsopen, _grow_handles, _hdopen, open, _open_osfhandle, _popen, sopen

182  Library Functions and Macros
Example:  
#include <stdio.h>  
#include <dos.h>  
#include <fcntl.h>  
#include <share.h>  

void main()  
{  
    int handle;  

    if( _dos_open( "file", O_RDONLY, &handle ) != 0 ) {  
        printf( "Unable to open file\n" );  
    } else {  
        printf( "Open succeeded\n" );  
        _dos_close( handle );  
    }  
}  

Classification: DOS  

Systems:   DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM
Synopsis:

```
#include <dos.h>
unsigned _dos_read( int handle, void __far *buffer,
                   unsigned count, unsigned *bytes );
```

Description: The `_dos_read` function uses system call 0x3F to read `count` bytes of data from the file specified by `handle` into the buffer pointed to by `buffer`. The number of bytes successfully read will be stored in the unsigned integer pointed to by `bytes`.

Returns: The `_dos_read` function returns zero if successful. Otherwise, it returns an OS error code and sets `errno` accordingly.

See Also: `_dos_close`, `_dos_open`, `_dos_write`

Example:

```
#include <stdio.h>
#include <dos.h>
#include <fcntl.h>

void main()
{
    unsigned len_read;
    int      handle;
    auto char buffer[80];

    if( _dos_open( "file", O_RDONLY, &handle ) != 0 ) {
        printf( "Unable to open file\n" );
    } else {
        printf( "Open succeeded\n" );
        _dos_read( handle, buffer, 80, &len_read );
        _dos_close( handle );
    }
}
```

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM
Synopsis: #include <dos.h>
unsigned _dos_setblock( unsigned size,
        unsigned segment,
        unsigned *maxsize );

Description: The _dos_setblock function uses system call 0x4A to change the size of segment, which was previously allocated by _dos_allocmem, to size paragraphs. If the request fails, the maximum number of paragraphs that this memory block can be changed to is returned in the word pointed to by maxsize.

For 32-bit DOS applications, it is recommended that the corresponding DPMI services be used.

Returns: The _dos_setblock function returns zero if successful. Otherwise, it returns an OS error code and sets errno to ENOMEM indicating a bad segment value, insufficient memory or corrupted memory.

See Also: _dos_allocmem, _dos_freemem, realloc

Example: #include <stdio.h>
#include <dos.h>

void main( void )
{
#if defined(__NT__) ||
    ( defined(__OS2__) &&
        (defined(__386__) || defined(__PPC__)))
    void *segment;
#else
    unsigned segment;
#endif
/* Try to allocate 100 paragraphs, then free them */
if( _dos_allocmem( 100, &segment ) != 0 ) { 
    printf( "_dos_allocmem failed\n" );
    printf( "Only %u paragraphs available\n", segment);
} else {
    printf( "_dos_allocmem succeeded\n" );
#
#if defined(__DOS__)
    { unsigned maxsize = 0;
/* Try to increase it to 200 paragraphs */
    if( _dos_setblock( 200, segment, &maxsize ) != 0 ) { 
        printf( "_dos_setblock failed: max=%u, err=%s\n", 
                    maxsize, strerror( errno ) );
    } else { 
        printf( "_dos_setblock succeeded\n" );
    }
#}
```c
if( _dos_freemem( segment ) != 0 ) {
    printf( "_dos_freemem failed\n" );
} else {
    printf( "_dos_freemem succeeded\n" );
}
```

**Classification:** DOS

**Systems:** DOS, DOS/PM
Synopsis:  

```c
#include <dos.h>
unsigned _dos_setdate( struct dosdate_t *date );
```

```c
struct dosdate_t {
    unsigned char day;      /* 1-31 */
    unsigned char month;    /* 1-12 */
    unsigned short year;    /* 1980-2099 */
    unsigned char dayofweek;/* 0-6 (0=Sunday) */
};
```

Description:  The `_dos_setdate` function uses system call 0x2B to set the current system date. The date information is passed in a `dosdate_t` structure pointed to by `date`.

Returns:  The `_dos_setdate` function returns zero if successful. Otherwise, it returns an OS error code and sets `errno` accordingly.

See Also:  `_dos_getdate`, `_dos_gettime`, `_dos_settime`, `gmtime`, `localtime`, `mktime`, `time`

Example:  

```c
#include <stdio.h>
#include <dos.h>

void main()
{
    struct dosdate_t date;
    struct dostime_t time;

    /* Get and display the current date and time */
    _dos_getdate( &date );
    _dos_gettime( &time );
    printf( "The date (MM-DD-YYYY) is: %d-%d-%d\n",
            date.month, date.day, date.year );
    printf( "The time (HH:MM:SS) is: %.2d:%.2d:%.2d\n",
            time.hour, time.minute, time.second );

    /* Change it to the turn of the century */
    date.year = 1999;
    date.month = 12;
    date.day = 31;
    time.hour = 23;
    time.minute = 59;
    _dos_setdate( &date );
    _dos_settime( &time );
    printf( "New date (MM-DD-YYYY) is: %d-%d-%d\n",
            date.month, date.day, date.year );
    printf( "New time (HH:MM:SS) is: %.2d:%.2d:%.2d\n",
            time.hour, time.minute, time.second );
}
```

produces the following:

```
The date (MM-DD-YYYY) is: 12-25-1989
The time (HH:MM:SS) is: 14:23:15
New date (MM-DD-YYYY) is: 12-31-1999
New time (HH:MM:SS) is: 23:59:16
```
_dos_setdate

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM
Synopsis:

```c
#include <dos.h>
void _dos_setdrive( unsigned drive, unsigned *total );
```

Description:
The `_dos_setdrive` function uses system call 0x0E to set the current default disk drive to be the
drive specified by `drive`, where 1 = drive A, 2 = drive B, etc. The total number of disk drives is returned
in the word pointed to by `total`. For DOS versions 3.0 or later, the minimum number of drives returned
is 5.

Returns:
The `_dos_setdrive` function has no return value. If an invalid drive number is specified, the
function fails with no error indication. You must use the `_dos_getdrive` function to check that the
desired drive has been set.

See Also:

`_dos_getdiskfree`, `_dos_getdrive`, `_getdiskfree`, `_getdrive`

Example:

```c
#include <stdio.h>
#include <dos.h>

void main()
{
    unsigned drive1, drive2, total;

    _dos_getdrive( &drive1 );
    printf( "Current drive is %c\n", 'A' + drive1 - 1 );
    /* try to change to drive C */
    _dos_setdrive( 3, &total );
    _dos_getdrive( &drive2 );
    printf( "Current drive is %c\n", 'A' + drive2 - 1 );
    /* go back to original drive */
    _dos_setdrive( drive1, &total );
    _dos_getdrive( &drive1 );
    printf( "Current drive is %c\n", 'A' + drive1 - 1 );
    printf( "Total number of drives is %u\n", total );
}
```

produces the following:

```
Current drive is D
Current drive is C
Total number of drives is 6
```

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM
Synopsis:  #include <dos.h>
unsigned _dos_setfileattr( const char *path,
unsigned attributes );

Description:  The _dos_setfileattr function uses system call 0x43 to set the attributes of the file or directory that path points to.  The possible attributes are:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>_A_NORMAL</td>
<td>Indicates a normal file.  File can be read or written without any restrictions.</td>
</tr>
<tr>
<td>_A_RDONLY</td>
<td>Indicates a read-only file.  File cannot be opened for &quot;write&quot;.</td>
</tr>
<tr>
<td>_A_HIDDEN</td>
<td>Indicates a hidden file.  This file will not show up in a normal directory search.</td>
</tr>
<tr>
<td>_A_SYSTEM</td>
<td>Indicates a system file.  This file will not show up in a normal directory search.</td>
</tr>
<tr>
<td>_A_VOLID</td>
<td>Indicates a volume-ID.</td>
</tr>
<tr>
<td>_A_SUBDIR</td>
<td>Indicates a sub-directory.</td>
</tr>
<tr>
<td>_A_ARCH</td>
<td>This is the archive flag.  It is set whenever the file is modified, and is cleared by the MS-DOS BACKUP command and other backup utility programs.</td>
</tr>
</tbody>
</table>

Returns:  The _dos_setfileattr function returns zero if successful.  Otherwise, it returns an OS error code and sets errno accordingly.

See Also:  _dos_getfileattr

Example:  #include <stdio.h>
#include <dos.h>

print_attribute() {
    unsigned attribute;
    _dos_getfileattr( "file", &attribute );
    printf( "File attribute is %x\n", attribute );
    if( attribute & _A_RDONLY ) {
        printf( "This is a read-only file\n" );
    } else {
        printf( "This is not a read-only file\n" );
    }
}

void main() {
    int      handle;
if( _dos_creat( "file", _A_RDONLY, &handle ) != 0 ){
    printf( "Error creating file\n" );
}
print_attribute();
_dos_setfileattr( "file", _A_NORMAL );
print_attribute();
_dos_close( handle );

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM
Synopsis:  
#include <dos.h>
unsigned _dos_setftime( int handle,
              unsigned date,
              unsigned time );

Description: The _dos_setftime function uses system call 0x57 to set the date and time that the file associated with handle was last modified. The date consists of the year, month and day packed into 16 bits as follows:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bits 0-4</td>
<td>Day (1-31)</td>
</tr>
<tr>
<td>bits 5-8</td>
<td>Month (1-12)</td>
</tr>
<tr>
<td>bits 9-15</td>
<td>Year (0-119 representing 1980-2099)</td>
</tr>
</tbody>
</table>

The time consists of the hour, minute and seconds/2 packed into 16 bits as follows:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bits 0-4</td>
<td>Seconds/2 (0-29)</td>
</tr>
<tr>
<td>bits 5-10</td>
<td>Minutes (0-59)</td>
</tr>
<tr>
<td>bits 11-15</td>
<td>Hours (0-23)</td>
</tr>
</tbody>
</table>

Returns: The _dos_setftime function returns zero if successful. Otherwise, it returns an OS error code and sets errno accordingly.

See Also: _dos_getftime

Example:  
#include <stdio.h>
#include <dos.h>
#include <fcntl.h>
#define YEAR(t)   (((t & 0xFE00) >> 9) + 1980)
#define MONTH(t)  ((t & 0x01E0) >> 5)
#define DAY(t)    (t & 0x001F)
#define HOUR(t)   ((t & 0xF800) >> 11)
#define MINUTE(t) ((t & 0x07E0) >> 5)
#define SECOND(t) ((t & 0x001F) << 1)

void main( void )
{
    int      handle;
    unsigned short date, time;
if( _dos_open( "file", O_RDWR, &handle ) != 0 ) {
    printf( "Unable to open file\n" );
} else {
    printf( "Open succeeded\n" );
    _dos_gettime( handle, &date, &time );
    printf( "The file was last modified on %d/%d/%d",
            MONTH(date), DAY(date), YEAR(date) );
    printf( " at %.2d:%.2d:%.2d\n",
            HOUR(time), MINUTE(time), SECOND(time) );
    /* set the time to 12 noon */
    time = (12 << 11) + (0 << 5) + 0;
    _dos_setftime( handle, &date, &time );
    _dos_gettime( handle, &date, &time );
    printf( "The file was last modified on %d/%d/%d",
            MONTH(date), DAY(date), YEAR(date) );
    printf( " at %.2d:%.2d:%.2d\n",
            HOUR(time), MINUTE(time), SECOND(time) );
    _dos_close( handle );
}

produces the following:

Open succeeded
The file was last modified on 12/29/1989 at 14:32:46
The file was last modified on 12/29/1989 at 12:00:00

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM
Synopsis:

```c
#include <dos.h>

unsigned _dos_settime( struct dostime_t *time );

struct dostime_t {
    unsigned char hour;     /* 0-23 */
    unsigned char minute;   /* 0-59 */
    unsigned char second;   /* 0-59 */
    unsigned char hsecond;  /* 1/100 second; 0-99 */
};
```

Description: The `_dos_settime` function uses system call 0x2D to set the current system time. The time information is passed in a `dostime_t` structure pointed to by `time`.

Returns: The `_dos_settime` function returns zero if successful. Otherwise, it returns a non-zero value and sets `errno` to `EINVAL` indicating that an invalid time was given.

See Also: `_dos_getdate`, `_dos_setdate`, `_dos_gettime`, `gmtime`, `localtime`, `mktime`, `time`

Example: 

```c
#include <stdio.h>
#include <dos.h>

void main()
{
    struct dosdate_t date;
    struct dostime_t time;

    /* Get and display the current date and time */
    _dos_getdate( &date );
    _dos_gettime( &time );
    printf( "The date (MM-DD-YYYY) is: %d-%d-%d\n",
            date.month, date.day, date.year );
    printf( "The time (HH:MM:SS) is: %.2d:%.2d:%.2d\n",
            time.hour, time.minute, time.second );

    /* Change it to the turn of the century */
    date.year = 1999;
    date.month = 12;
    date.day = 31;
    time.hour = 23;
    time.minute = 59;
    _dos_setdate( &date );
    _dos_settime( &time );
    printf( "New date (MM-DD-YYYY) is: %d-%d-%d\n",
            date.month, date.day, date.year );
    printf( "New time (HH:MM:SS) is: %.2d:%.2d:%.2d\n",
            time.hour, time.minute, time.second );
}
```

produces the following:

The date (MM-DD-YYYY) is: 12-25-1989
The time (HH:MM:SS) is: 14:23:15
New date (MM-DD-YYYY) is: 12-31-1999
New time (HH:MM:SS) is: 23:59:16

Classification: DOS
Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM
**_dos_setvect_**

**Synopsis:**
```
#include <dos.h>
void _dos_setvect( unsigned int num,
    void (__interrupt __far *handler)() );
```

**Description:** The `_dos_setvect` function sets interrupt vector number `intnum` to point to the interrupt handling function pointed to by `handler`.

**Returns:** The `_dos_setvect` function does not return a value.

**See Also:** `_chain_intr`, `_dos_getvect`, `_dos_keep`

**Example:**
```
#include <stdio.h>
#include <dos.h>

volatile int clock_ticks;
void (__interrupt __far *prev_int_1c)();
#define BLIP_COUNT (5*18) /* 5 seconds */

void __interrupt __far timer_rtn()
{
    ++clock_ticks;
    _chain_intr( prev_int_1c );
}

int compile_a_line()
{
    static int delays = 0;
    if( delays > 15 ) return( 0 );
    delay( 1000 ); /* delay for 1 second */
    printf( "Delayed for 1 second\n" );
    delays++;
    return( 1 );
}

void main()
{
    prev_int_1c = _dos_getvect( 0x1c );
    _dos_setvect( 0x1c, timer_rtn );
    while( compile_a_line() ) {
        if( clock_ticks >= BLIP_COUNT ) {
            putchar( '.' );
            clock_ticks -= BLIP_COUNT;
        }
        _dos_setvect( 0x1c, prev_int_1c );
    }
}
```

**Classification:** WATCOM

**Systems:** DOS, Windows, DOS/PM

---

196  Library Functions and Macros
Synopsis:  
#include <dos.h>
unsigned _dos_write( int handle, void const __far *buffer,
                    unsigned count, unsigned *bytes );

Description:  The _dos_write function uses system call 0x40 to write count bytes of data from the buffer pointed to by buffer to the file specified by handle. The number of bytes successfully written will be stored in the unsigned integer pointed to by bytes.

Returns:    The _dos_write function returns zero if successful. Otherwise, it returns an OS error code and sets errno accordingly.

See Also:   _dos_close, _dos_open, _dos_read

Example:    #include <stdio.h>
            #include <dos.h>
            #include <fcntl.h>

            char buffer[] = "This is a test for _dos_write."

            void main()
            {
                unsigned len_written;
                int      handle;

                if( _dos_creat( "file", _A_NORMAL, &handle ) != 0 ) {
                    printf( "Unable to create file\n" );
                } else {
                    printf( "Create succeeded\n" );
                    _dos_write( handle, buffer, sizeof(buffer),
                                 &len_written);
                    _dos_close( handle );
                }
            }

Classification:   DOS

Systems:    DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, DOS/PM
Synopsis:  
#include <io.h>
int dup( int handle );
int _dup( int handle );

Description:  The dup function duplicates the file handle given by the argument handle. The new file handle refers to the same open file handle as the original file handle, and shares any locks. The new file handle is identical to the original in that it references the same file or device, it has the same open mode (read and/or write) and it will have file position identical to the original. Changing the position with one handle will result in a changed position in the other.

The _dup function is identical to dup. Use _dup for ANSI/ISO naming conventions.

Returns:  If successful, the new file handle is returned to be used with the other functions which operate on the file. Otherwise, -1 is returned and errno is set to indicate the error.

Errors:  When an error has occurred, errno contains a value indicating the type of error that has been detected.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBADF</td>
<td>The argument handle is not a valid open file handle.</td>
</tr>
<tr>
<td>EMFILE</td>
<td>The number of file handles would exceed {OPEN_MAX}.</td>
</tr>
</tbody>
</table>

See Also:  chsize, close, creat, dup2, eof, exec..., fdopen, filelength, fileno, fstat, _grow_handles, isatty, lseek, open, read, setmode, sopen, stat, tell, write, umask

Example:  
#include <fcntl.h>
#include <io.h>

void main( void )
{
    int handle, dup_handle;

    handle = open( "file",
                   O_WRONLY | O_CREAT | O_TRUNC | O_TEXT,
                   S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );
    if( handle != -1 ) {
        dup_handle = dup( handle );
        if( dup_handle != -1 ) {
            /* process file */
            close( dup_handle );
        } else {
            close( handle );
        }
    }
}

Classification: dup is POSIX 1003.1
_dup is not POSIX
_dup conforms to ANSI/ISO naming conventions

Systems:  dup - All, Netware
_dup - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <io.h>
int dup2( int handle, int handle2 );
int _dup2( int handle, int handle2 );

Description:  The dup2 function duplicates the file handle given by the argument handle. The new file handle is identical to the original in that it references the same file or device, it has the same open mode (read and/or write) and it will have identical file position to the original (changing the position with one handle will result in a changed position in the other).

The number of the new handle is handle2. If a file already is opened with this handle, the file is closed before the duplication is attempted.

The _dup2 function is identical to dup2. Use _dup2 for ANSI/ISO naming conventions.

Returns:  The dup2 function returns zero if successful. Otherwise, -1 is returned and errno is set to indicate the error.

Errors:  When an error has occurred, errno contains a value indicating the type of error that has been detected.

Constant Meaning
EBADF The argument handle is not a valid open file handle or handle2 is out of range.
EMFILE The number of file handles would exceed {OPEN_MAX}, or no file handles above handle2 are available.

See Also:  chsize, close, creat, dup, eof, exec..., fopen, filelength, fileno, fstat, _grow_handles, isatty, lseek, open, read, setmode, sopen, stat, tell, write, umask

Example:  
#include <fcntl.h>
#include <io.h>

void main()
{
  int handle, dup_handle;
  
  handle = open( "file",
         O_WRONLY | O_CREAT | O_TRUNC | O_TEXT,
         S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );
  if( handle != -1 ) {
    dup_handle = 4;
    if( dup2( handle, dup_handle ) != -1 ) {
      /* process file */
      close( dup_handle );
    }
  }
  close( handle );
}

Classification: dup2 is POSIX 1003.1
_dup2 is not POSIX
Systems:  

dup2 - All, Netware
_dup2 - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_dwDeleteOnClose

Synopsis:  
#include <wdefwin.h>
int _dwDeleteOnClose( int handle );

Description:  The _dwDeleteOnClose function tells the console window that it should close itself when the corresponding file is closed. The argument handle is the handle associated with the opened console.

The _dwDeleteOnClose function is one of the support functions that can be called from an application using Watcom’s default windowing support.

Returns:  The _dwDeleteOnClose function returns 1 if it was successful and 0 if not.

See Also:  _dwSetAboutDlg, _dwSetAppTitle, _dwSetConTitle, _dwShutDown, _dwYield

Example:  
#include <wdefwin.h>
#include <stdio.h>

void main()
{
    FILE *sec;

    _dwSetAboutDlg( "Hello World About Dialog",
                    "About Hello World
                    Copyright 1994 by WATCOM
                    ");
    _dwSetAppTitle( "Hello World Application Title" );
    _dwSetConTitle( 0, "Hello World Console Title" );
    printf( "Hello World\n" );
    sec = fopen( "CON", "r+" );
    _dwSetConTitle( fileno( sec ),
                    "Hello World Second Console Title" );
    _dwDeleteOnClose( fileno( sec ) );
    fprintf( sec, "Hello to second console\n" );
    fprintf( sec, "Press Enter to close this console\n" );
    fflush( sec );
    fgetc( sec );
    fclose( sec );
}

Classification:  WATCOM

Systems:  Windows, Win386, Win32, OS/2-32

202 Library Functions and Macros
Synopsis:  
#include <wdefwin.h>
int _dwSetAboutDlg( const char *title, const char *text );

Description:  The _dwSetAboutDlg function sets the "About" dialog box of the default windowing system. The argument title points to the string that will replace the current title. If title is NULL then the title will not be replaced. The argument text points to a string which will be placed in the "About" box. To get multiple lines, embed a new line after each logical line in the string. If text is NULL, then the current text in the "About" box will not be replaced.

The _dwSetAboutDlg function is one of the support functions that can be called from an application using Watcom’s default windowing support.

Returns:  The _dwSetAboutDlg function returns 1 if it was successful and 0 if not.

See Also:  _dwDeleteOnClose, _dwSetAppTitle, _dwSetConTitle, _dwShutDown, _dwYield

Example:  
#include <wdefwin.h>
#include <stdio.h>

void main()
{
    FILE *sec;

    _dwSetAboutDlg( "Hello World About Dialog",
                    "About Hello World\n"
                    "Copyright 1994 by WATCOM\n" );
    _dwSetAppTitle( "Hello World Application Title" );
    _dwSetConTitle( 0, "Hello World Console Title" );
    printf( "Hello World\n" );
    sec = fopen( "CON", "r+" );
    _dwSetConTitle( fileno( sec ),
                    "Hello World Second Console Title" );
    _dwDeleteOnClose( fileno( sec ) );
    fprintf( sec, "Hello to second console\n" );
    fprintf( sec, "Press Enter to close this console\n" );
    fflush( sec );
    fgetc( sec );
    fclose( sec );
}

Classification: WATCOM

Systems:  Windows, Win386, Win32, OS/2-32
_dwSetAppTitle

Synopsis:  
#include <wdefwin.h>
int _dwSetAppTitle( const char *title );

Description:  The _dwSetAppTitle function sets the main window’s title.  The argument title points to the string that will replace the current title.

The _dwSetAppTitle function is one of the support functions that can be called from an application using Watcom’s default windowing support.

Returns:  The _dwSetAppTitle function returns 1 if it was successful and 0 if not.

See Also:  _dwDeleteOnClose, _dwSetAboutDlg, _dwSetConTitle, _dwShutDown, _dwYield

Example:  
#include <wdefwin.h>
#include <stdio.h>

void main()
{

    FILE *sec;

    _dwSetAboutDlg( "Hello World About Dialog",
        "About Hello World
        "Copyright 1994 by WATCOM
    );
    _dwSetAppTitle( "Hello World Application Title" );
    _dwSetConTitle( 0, "Hello World Console Title" );
    printf( "Hello World\n" );
    sec = fopen( "CON", "r+" );
    _dwSetConTitle( fileno( sec ),
        "Hello World Second Console Title" );
    _dwDeleteOnClose( fileno( sec ) );
    fprintf( sec, "Hello to second console\n" );
    fprintf( sec, "Press Enter to close this console\n" );
    fflush( sec );
    fgetc( sec );
    fclose( sec );
}

Classification:  WATCOM

Systems:  Windows, Win386, Win32, OS/2-32

204  Library Functions and Macros
Synopsis:  
#include <wdefwin.h>  
int _dwSetConTitle( int handle, const char *title );

Description:  The _dwSetConTitle function sets the console window’s title which corresponds to the handle passed to it. The argument handle is the handle associated with the opened console. The argument title points to the string that will replace the current title.

The _dwSetConTitle function is one of the support functions that can be called from an application using Watcom’s default windowing support.

Returns:  The _dwSetConTitle function returns 1 if it was successful and 0 if not.

See Also:  _dwDeleteOnClose, _dwSetAboutDlg, _dwSetAppTitle, _dwShutDown, _dwYield

Example:  
#include <wdefwin.h>  
#include <stdio.h>

void main()  
{
    FILE *sec;

    _dwSetAboutDlg( "Hello World About Dialog",  
        "About Hello World\n"  
        "Copyright 1994 by WATCOM\n" );
    _dwSetAppTitle( "Hello World Application Title" );
    _dwSetConTitle( 0, "Hello World Console Title" );
    printf( "Hello World\n" );
    sec = fopen( "CON", "r+" );
    _dwSetConTitle( fileno( sec ),  
        "Hello World Second Console Title" );
    _dwDeleteOnClose( fileno( sec ) );
    fprintf( sec, "Hello to second console\n" );
    fprintf( sec, "Press Enter to close this console\n" );
    fflush( sec );
    fgetc( sec );
    fclose( sec );
}

Classification: WATCOM

Systems:  Windows, Win386, Win32, OS/2-32
_dwShutDown

Synopsis:    

#include <wdefwin.h>
int _dwShutDown( void );

Description: The _dwShutDown function shuts down the default windowing I/O system. The application will continue to execute but no windows will be available for output. Care should be exercised when using this function since any subsequent output may cause unpredictable results.

When the application terminates, it will not be necessary to manually close the main window.

The _dwShutDown function is one of the support functions that can be called from an application using Watcom’s default windowing support.

Returns: The _dwShutDown function returns 1 if it was successful and 0 if not.

See Also: _dwDeleteOnClose, _dwSetAboutDlg, _dwSetAppTitle, _dwSetConTitle, _dwYield

Example:    

#include <wdefwin.h>
#include <stdio.h>

void main()
{

    FILE *sec;

    _dwSetAboutDlg( "Hello World About Dialog",
                    "About Hello World\n"
                    "Copyright 1994 by WATCOM\n" );
    _dwSetAppTitle( "Hello World Application Title" );
    _dwSetConTitle( 0, "Hello World Console Title" );
    printf( "Hello World\n" );

    sec = fopen( "CON", "r+" );
    _dwSetConTitle( fileno( sec ),
                    "Hello World Second Console Title" );
    _dwDeleteOnClose( fileno( sec ) );
    fprintf( sec, "Hello to second console\n" );
    fprintf( sec, "Press Enter to close this console\n" );
    fflush( sec );
    fgetc( sec );
    fclose( sec );
    _dwShutDown();
    /*
     * do more computing that does not involve
     * console input/output
     */
}

Classification: WATCOM

Systems: Windows, Win386, Win32, OS/2-32

206  Library Functions and Macros
Synopsis:  
#include <wdefwin.h>  
int _dwYield( void );

Description:  
The _dwYield function yields control back to the operating system, thereby giving other processes a chance to run.

The _dwYield function is one of the support functions that can be called from an application using Watcom’s default windowing support.

Returns:  
The _dwYield function returns 1 if it was successful and 0 if not.

See Also:  
_dwDeleteOnClose, _dwSetAboutDlg, _dwSetAppTitle, _dwSetConTitle, _dwShutDown

Example:  
#include <wdefwin.h>  
#include <stdio.h>  

void main()  
{
   int i;
   for( i = 0; i < 1000; i++ ) {  
      /* give other processes a chance to run */  
      _dwYield();  
      /* do CPU-intensive calculation */  
      /* . */  
      /* . */  
   }  
}

Classification:  WATCOM

Systems:  
Windows, Win386, Win32, OS/2-32
Synopsis:
#include <stdlib.h>
char *ecvt( double value,
    int ndigits,
    int *dec,
    int *sign );
char *__ecvt( double value,
    int ndigits,
    int *dec,
    int *sign );
wchar_t *__wecvt( double value,
    int ndigits,
    int *dec,
    int *sign );

Description: The ecvt function converts the floating-point number value into a character string. The parameter ndigits specifies the number of significant digits desired. The converted number will be rounded to ndigits of precision.

The character string will contain only digits and is terminated by a null character. The integer pointed to by dec will be filled in with a value indicating the position of the decimal point relative to the start of the string of digits. A zero or negative value indicates that the decimal point lies to the left of the first digit. The integer pointed to by sign will contain 0 if the number is positive, and non-zero if the number is negative.

The __ecvt function is identical to ecvt. Use __ecvt for ANSI/ISO naming conventions.

The __wecvt function is identical to ecvt except that it produces a wide-character string.

Returns: The ecvt function returns a pointer to a static buffer containing the converted string of digits. Note: ecvt and fcvt both use the same static buffer.

See Also: fcvt, gcvt, printf

Example: #include <stdio.h>
#include <stdlib.h>

void main()
{
    char *str;
    int dec, sign;

    str = ecvt( 123.456789, 6, &dec, &sign );
    printf( "str=%s, dec=%d, sign=%d\n", str, dec, sign );
}

produces the following:
str=123457, dec=3, sign=0

Classification: WATCOM
_ecvt conforms to ANSI/ISO naming conventions

Systems: ecvt - Math
__ecvt - Math

208  Library Functions and Macros
wecvt - Math
**_ellipse Functions**

**Synopsis:**

```
#include <graph.h>
short _FAR _ellipse( short fill, short x1, short y1,
                      short x2, short y2 );

short _FAR _ellipse_w( short fill, double x1, double y1,
                      double x2, double y2 );

short _FAR _ellipse_wxy( short fill,
                        struct _wxycoord _FAR *p1,
                        struct _wxycoord _FAR *p2 );
```

**Description:**

The _ellipse functions draw ellipses. The _ellipse function uses the view coordinate system. The _ellipse_w and _ellipse_wxy functions use the window coordinate system.

The center of the ellipse is the center of the rectangle established by the points \((x1,y1)\) and \((x2,y2)\).

The argument `fill` determines whether the ellipse is filled in or has only its outline drawn. The argument can have one of two values:

- **_GFILLINTERIOR** fill the interior by writing pixels with the current plot action using the current color and the current fill mask
- **_GBORDER** leave the interior unchanged; draw the outline of the figure with the current plot action using the current color and line style

When the coordinates \((x1,y1)\) and \((x2,y2)\) establish a line or a point (this happens when one or more of the x-coordinates or y-coordinates are equal), nothing is drawn.

**Returns:**

The _ellipse functions return a non-zero value when the ellipse was successfully drawn; otherwise, zero is returned.

**See Also:**

_arc, _rectangle, _setcolor, _setfillmask, _setlinestyle, _setplotaction

**Example:**

```
#include <conio.h>
#include <graph.h>

main()
{
    _setvideomode( _VRES16COLOR );
    _ellipse( _GBORDER, 120, 90, 520, 390 );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

produces the following:
**Classification:** _ellipse_ is PC Graphics

**Systems:**
- _ellipse_ - DOS, QNX
- _ellipse_w_ - DOS, QNX
- _ellipse_wxy_ - DOS, QNX
Synopsis:  
#include <i86.h>  
void _enable( void );

Description:  
The _enable function causes interrupts to become enabled.

The _enable function would be used in conjunction with the _disable function to make sure that a sequence of instructions are executed without any intervening interrupts occurring.

Returns:  
The _enable function returns no value.

See Also:  
_disable

Example:  
#include <stdio.h>  
#include <stdlib.h>  
#include <i86.h>

struct list_entry {  
    struct list_entry *next;  
    int data;  
};
struct list_entry *ListHead = NULL;  
struct list_entry *ListTail = NULL;

void insert( struct list_entry *new_entry )  
{  
    /* insert new_entry at end of linked list */  
    new_entry->next = NULL;  
    _disable();  
    /* disable interrupts */  
    if( ListTail == NULL ) {  
        ListHead = new_entry;  
    } else {  
        ListTail->next = new_entry;  
    }  
    ListTail = new_entry;  
    _enable();  
    /* enable interrupts now */  
}

void main()  
{  
    struct list_entry *p;  
    int i;

    for( i = 1; i <= 10; i++ ) {  
        p = (struct list_entry *) malloc( sizeof( struct list_entry ) );  
        if( p == NULL ) break;  
        p->data = i;  
        insert( p );  
    }  
}

Classification:  Intel

Systems:  All, Netware
Synopsis:
#include <process.h>
void _endthread(void);
void _endthreadex( unsigned retval );

Description: The _endthread function is used to terminate a thread created by _beginthread. For each operating environment under which _endthread is supported, the _endthread function uses the appropriate system call to end the current thread of execution.

The _endthreadex function is used to terminate a thread created by _beginthreadex. The thread exit code retval must be specified.

Returns: The _endthread function does not return any value.

See Also: _beginthread

Example:
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <malloc.h>
#include <process.h>
#include <dos.h>

#if defined(__386__)
#define FAR
#define STACK_SIZE    8192
#else
#define FAR __far
#define STACK_SIZE    4096
#endif

static volatile int     WaitForThread;

void FAR child( void FAR *parm )
{
    char * FAR *argv = (char * FAR *) parm;
    int  i;

    printf( "Child thread ID = %x\n", *__threadid );
    for( i = 0; argv[i]; i++ ) {
        printf( "argv[%d] = %s\n", i, argv[i] );
    }
    WaitForThread = 0;
    _endthread();
}
void main()
{
    char *args[3];
#if defined(__NT__)
    unsigned long tid;
#else
    char *stack;
    int tid;
#endif
    #endif
    args[0] = "child";
    args[1] = "parm";
    args[2] = NULL;
    WaitForThread = 1;
#if defined(__NT__)
    tid = _beginthread( child, STACK_SIZE, args );
    printf( "Thread handle = %lx\n", tid );
#else
    #if defined(__386__)
        stack = (char *) malloc( STACK_SIZE );
    #else
        stack = (char *) _nmalloc( STACK_SIZE );
    #endif
    tid = _beginthread( child, stack, STACK_SIZE, args );
    printf( "Thread ID = %x\n", tid );
#endif
    while( WaitForThread ) {
        sleep( 0 );
    }
}

Classification: WATCOM

Systems: _endthread - Win32, QNX/32, OS/2 1.x(MT), OS/2 1.x(DL), OS/2-32, Netware
         _endthreadex - Win32
Synopsis:  
#include <io.h>  
int eof( int handle );  
int _eof( int handle );

Description:  The eof function determines, at the operating system level, if the end of the file has been reached for the file whose file handle is given by handle. Because the current file position is set following an input operation, the eof function may be called to detect the end of the file before an input operation beyond the end of the file is attempted.

The _eof function is identical to eof. Use _eof for ANSI/ISO naming conventions.

Returns:  The eof function returns 1 if the current file position is at the end of the file, 0 if the current file position is not at the end. A return value of -1 indicates an error, and in this case errno is set to indicate the error.

Errors:  When an error has occurred, errno contains a value indicating the type of error that has been detected.

EBADF  The handle argument is not a valid file handle.

See Also:  read

Example:  
#include <stdio.h>  
#include <fcntl.h>  
#include <io.h>

void main( void )
{
    int handle, len;
    char buffer[100];

    handle = open( "file", O_RDONLY );
    if( handle != -1 ) {
        while( ! eof( handle ) ) {
            len = read( handle, buffer, sizeof(buffer) - 1 );
            buffer[ len ] = '\0';
            printf( "%s", buffer );
        }
    }
    close( handle );
}

Classification:  WATCOM

Systems:  eof - All, Netware  
_EOF - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  

```
#include <process.h>

int execl(   path, arg0, arg1..., argn, NULL );
int execl(   path, arg0, arg1..., argn, NULL, envp );
int execlp( file, arg0, arg1..., argn, NULL );
int execlpe( file, arg0, arg1..., argn, NULL, envp );
int execv(   path, argv );
int execve(  path, argv, envp );
int execvp(  file, argv );
int execvpe( file, argv, envp );
const char *path;             /* file name incl. path */
const char *file;             /* file name */
const char *arg0, ..., *argn; /* arguments */
const char *const argv[];     /* array of arguments */
const char *const envp[];     /* environment strings */
int _wexecl(   path, arg0, arg1..., argn, NULL );
int _wexecl(   path, arg0, arg1..., argn, NULL, envp );
int _wexeclp( file, arg0, arg1..., argn, NULL );
int _wexeclpe( file, arg0, arg1..., argn, NULL, envp );
int _wexecv(   path, argv );
int _wexecve(  path, argv, envp );
int _wexecvp(  file, argv );
int _wexecvpe( file, argv, envp );
const wchar_t *path;            /* file name incl. path */
const wchar_t *file;            /* file name */
const wchar_t *arg0, ..., *argn; /* arguments */
const wchar_t *const argv[];    /* array of arguments */
const wchar_t *const envp[];    /* environment strings */
```

Description:  

The exec... functions load and execute a new child process, named by path or file. If the child process is successfully loaded, it replaces the current process in memory. No return is made to the original program.

The program is located by using the following logic in sequence:

1. An attempt is made to locate the program in the current working directory if no directory specification precedes the program name; otherwise, an attempt is made in the specified directory.

2. If no file extension is given, an attempt is made to find the program name, in the directory indicated in the first point, with .COM concatenated to the end of the program name.

3. If no file extension is given, an attempt is made to find the program name, in the directory indicated in the first point, with .EXE concatenated to the end of the program name.

4. When no directory specification is given as part of the program name, the execlp, execlpe, execvp, and execvpe functions will repeat the preceding three steps for each of the directories specified by the PATH environment variable. The command

```
path c:\myapps;d:\lib\applns
```

indicates that the two directories

```
c:\myapps
d:\lib\applns```
are to be searched. The DOS PATH command (without any directory specification) will cause the current path definition to be displayed.

An error is detected when the program cannot be found.

Arguments are passed to the child process by supplying one or more pointers to character strings as arguments in the exec... call. These character strings are concatenated with spaces inserted to separate the arguments to form one argument string for the child process. The length of this concatenated string must not exceed 128 bytes for DOS systems.

The arguments may be passed as a list of arguments (execl, execle, execlp, and execlpe) or as a vector of pointers (execv, execve, execvp, and execvpe). At least one argument, arg0 or argv[0], must be passed to the child process. By convention, this first argument is a pointer to the name of the program.

If the arguments are passed as a list, there must be a NULL pointer to mark the end of the argument list. Similarly, if a pointer to an argument vector is passed, the argument vector must be terminated by a NULL pointer.

The environment for the invoked program is inherited from the parent process when you use the execl, execlp, execv, and execvp functions. The execle, execlpe, execve, and execvpe functions allow a different environment to be passed to the child process through the envp argument. The argument envp is a pointer to an array of character pointers, each of which points to a string defining an environment variable. The array is terminated with a NULL pointer. Each pointer locates a character string of the form

variable=value

that is used to define an environment variable. If the value of envp is NULL, then the child process inherits the environment of the parent process.

The environment is the collection of environment variables whose values have been defined with the DOS SET command or by the successful execution of the putenv function. A program may read these values with the getenv function.

The execvpe and execlpe functions are extensions to POSIX 1003.1. The wide-character _wexecl, _wexecle, _wexeclp, _wexeclpe, _wexecv, _wexecve, _wexecvp and _wexecvpe functions are similar to their counterparts but operate on wide-character strings.

**Returns:** When the invoked program is successfully initiated, no return occurs. When an error is detected while invoking the indicated program, exec... returns -1 and errno is set to indicate the error.

**Errors:** When an error has occurred, errno contains a value indicating the type of error that has been detected.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>E2BIG</td>
<td>The argument list exceeds 128 bytes, or the space required for the environment information exceeds 32K.</td>
</tr>
<tr>
<td>EACCES</td>
<td>The specified file has a locking or sharing violation.</td>
</tr>
<tr>
<td>EMFILE</td>
<td>Too many files open</td>
</tr>
</tbody>
</table>
**ENOENT** Path or file not found

**ENOMEM** Not enough memory is available to execute the child process.

**See Also:** abort, atexit, exit, _exit, getcmd, getenv, main, putenv, spawn..., system

**Example:**

```c
#include <stddef.h>
#include <process.h>

execcl( "myprog", 
        "myprog", "ARG1", "ARG2", NULL );
```

The preceding invokes "myprog" as if

```
myprog ARG1 ARG2
```

had been entered as a command to DOS. The program will be found if one of

```
myprog.
myprog.com
myprog.exe
```

is found in the current working directory.

```c
#include <stddef.h>
#include <process.h>

char *env_list[] = { "SOURCE=MYDATA",
                    "TARGET=OUTPUT",
                    "lines=65",
                    NULL
                   };

execle( "myprog",
       "myprog", "ARG1", "ARG2", NULL,
              env_list );
```

The preceding invokes "myprog" as if

```
myprog ARG1 ARG2
```

had been entered as a command to DOS. The program will be found if one of

```
myprog.
myprog.com
myprog.exe
```

is found in the current working directory. The DOS environment for the invoked program will consist of the three environment variables SOURCE, TARGET and lines.

```c
#include <stddef.h>
#include <process.h>

char *arg_list[] = { "myprog", "ARG1", "ARG2", NULL };

execv( "myprog", arg_list );
```
The preceding invokes "myprog" as if

    myprog ARG1 ARG2

had been entered as a command to DOS. The program will be found if one of

    myprog.
    myprog.com
    myprog.exe

is found in the current working directory.

**Classification:** exec... is POSIX 1003.1 with extensions
_wexec... is not POSIX

**Systems:**
execl - DOS/16, Win32, QNX, OS/2 1.x(all), OS/2-32
execle - DOS/16, Win32, QNX, OS/2 1.x(all), OS/2-32
execlp - DOS/16, Win32, QNX, OS/2 1.x(all), OS/2-32
execlpe - DOS/16, Win32, QNX, OS/2 1.x(all), OS/2-32
execv - DOS/16, Win32, QNX, OS/2 1.x(all), OS/2-32
execve - DOS/16, Win32, QNX, OS/2 1.x(all), OS/2-32
texecvp - DOS/16, Win32, QNX, OS/2 1.x(all), OS/2-32
execvpe - DOS/16, Win32, QNX, OS/2 1.x(all), OS/2-32
_wexecl - DOS/16, Win32, OS/2 1.x(all), OS/2-32
_wexecle - DOS/16, Win32, OS/2 1.x(all), OS/2-32
_wexeclp - DOS/16, Win32, OS/2 1.x(all), OS/2-32
_wexeclpe - DOS/16, Win32, OS/2 1.x(all), OS/2-32
_wexecv - DOS/16, Win32, OS/2 1.x(all), OS/2-32
_wexecve - DOS/16, Win32, OS/2 1.x(all), OS/2-32
_wexecvp - DOS/16, Win32, OS/2 1.x(all), OS/2-32
_wexecvpe - DOS/16, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <stdlib.h>
void _exit( int status );
void _Exit( int status );

Description: The _exit function causes normal program termination to occur.

1. The functions registered by the atexit or onexit functions are not called.

2. Any unopened files are not closed and any buffered output is not flushed to the associated files or devices.

3. Any files created by tmpfile are not removed.

4. The return status is made available to the parent process. Only the low order byte of status is available on DOS systems. The status value is typically set to 0 to indicate successful termination and set to some other value to indicate an error.

Returns: The _exit function does not return to its caller.

See Also: abort, atexit, _bgetcmd, exec..., exit, _Exit, getcmd, getenv, main, onexit, putenv, spawn..., system

Example:  
#include <stdio.h>
#include <stdlib.h>

void main( int argc, char *argv[] )
{
  FILE *fp;

  if( argc <= 1 ) {
    fprintf( stderr, "Missing argument\n" );
    exit( EXIT_FAILURE );
  }

  fp = fopen( argv[1], "r" );
  if( fp == NULL ) {
    fprintf( stderr, "Unable to open '%s'\n", argv[1] );
    _exit( EXIT_FAILURE );
  }
  fclose( fp );
  _exit( EXIT_SUCCESS );
}

Classification: POSIX 1003.1
_Exit is ISO C99

Systems:  
_exit - All, Netware
_Exit - All, Netware
Synopsis:  

#include <stdlib.h>
void exit( int status );

Description:  The exit function causes normal program termination to occur.

First, all functions registered by the atexit function are called in the reverse order of their registration. Next, all open files are flushed and closed, and all files created by the tmpfile function are removed. Finally, the return status is made available to the parent process. Only the low order byte of status is available on DOS systems. The status value is typically set to 0 to indicate successful termination and set to some other value to indicate an error.

Returns:  The exit function does not return to its caller.

See Also:  abort, atexit, _exit, onexit

Example:

#include <stdio.h>
#include <stdlib.h>

void main( int argc, char *argv[] )
{
  FILE *fp;

  if( argc <= 1 ) {
    fprintf( stderr, "Missing argument\n" );
    exit( EXIT_FAILURE );
  }

  fp = fopen( argv[1], "r" );
  if( fp == NULL ) {
    fprintf( stderr, "Unable to open '%s'\n", argv[1] );
    exit( EXIT_FAILURE );
  }
  fclose( fp );
  exit( EXIT_SUCCESS );
}

Classification: ANSI

Systems: All, Netware
Synopsis:  
#include <math.h>
double exp( double x );

Description:  The exp function computes the exponential function of x.  A range error occurs if the magnitude of x is too large.

Returns:  The exp function returns the exponential value. When the argument is outside the permissible range, the matherr function is called. Unless the default matherr function is replaced, it will set the global variable errno to ERANGE, and print a "RANGE error" diagnostic message using the stderr stream.

See Also:  log, matherr

Example:  
#include <stdio.h>
#include <math.h>

   void main()
   {
      printf( "%f\n", exp(.5) );
   }

produces the following:

   1.648721

Classification:  ANSI

Systems:  Math
Synopsis:  
#include <malloc.h>
void *_expand( void *mem_blk, size_t size );
void __based(void) *bexpand( __segment seg,
  void __based(void) *mem_blk,
  size_t size );
void __far  *fexpand(void __far  *mem_blk,size_t size);
void __near *nexpand(void __near *mem_blk,size_t size);

Description:  The _expand functions change the size of the previously allocated block pointed to by mem_blk by attempting to expand or contract the memory block without moving its location in the heap. The argument size specifies the new desired size for the memory block. The contents of the memory block are unchanged up to the shorter of the new and old sizes.

Each function expands the memory from a particular heap, as listed below:

<table>
<thead>
<tr>
<th>Function</th>
<th>Heap Expanded</th>
</tr>
</thead>
<tbody>
<tr>
<td>_expand</td>
<td>Depends on data model of the program</td>
</tr>
<tr>
<td>_bexpand</td>
<td>Based heap specified by seg value</td>
</tr>
<tr>
<td>_fexpand</td>
<td>Far heap (outside the default data segment)</td>
</tr>
<tr>
<td>_nexpand</td>
<td>Near heap (inside the default data segment)</td>
</tr>
</tbody>
</table>

In a small data memory model, the _expand function is equivalent to the _nexpand function; in a large data memory model, the _expand function is equivalent to the _fexpand function.

Returns:  The _expand functions return the value mem_blk if it was successful in changing the size of the block. The return value is NULL ( _NULLOFF for _bexpand) if the memory block could not be expanded to the desired size. It will be expanded as much as possible in this case.

The appropriate _msize function can be used to determine the new size of the expanded block.

See Also:  calloc Functions, free Functions, halloc, hfree, malloc Functions, _msize Functions, realloc Functions, sbrk

Example:  
#include <stdio.h>
#include <malloc.h>

void main()
{
  char *buf;
  char __far *buf2;
buf = (char *) malloc( 80 );
printf( "Size of buffer is %u\n", _msize(buf) );
if( _expand( buf, 100 ) == NULL ) {
    printf( "Unable to expand buffer\n" );
}
printf( "New size of buffer is %u\n", _msize(buf) );
buf2 = (char __far *) _fmalloc( 2000 );
printf( "Size of far buffer is %u\n", _fmsize(buf2) );
if( _fexpand( buf2, 8000 ) == NULL ) {
    printf( "Unable to expand far buffer\n" );
}
printf( "New size of far buffer is %u\n", _fmsize(buf2) );
}

produces the following:

Size of buffer is 80
Unable to expand buffer
New size of buffer is 80
Size of far buffer is 2000
New size of far buffer is 8000

Classification: WATCOM

Systems: _expand - All
          _bexpand - DOS/16, Windows, QNX/16, OS/2 1.x(all)
          _fexpand - DOS/16, Windows, QNX/16, OS/2 1.x(all)
          _nexpand - DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2 1.x(MT),
                      OS/2-32
Synopsis:  
```c
#include <math.h>
double fabs( double x );
```

Description:  The `fabs` function computes the absolute value of the argument `x`.

Returns:  The `fabs` function returns the absolute value of `x`.

See Also:  `abs`, `labs`, `imaxabs`

Example:  
```c
#include <stdio.h>
#include <math.h>

void main()
{
    printf( "%f %f\n", fabs(.5), fabs(-.5) );
}
```

produces the following:

```
0.500000 0.500000
```

Classification:  ANSI

Systems:  Math
Synopsis:  
#include <stdio.h>
int fclose( FILE *fp );

Description:  The fclose function closes the file fp. If there was any unwritten buffered data for the file, it is written out before the file is closed. Any unread buffered data is discarded. If the associated buffer was automatically allocated, it is deallocated.

Returns:  The fclose function returns zero if the file was successfully closed, or non-zero if any errors were detected. When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also:  fcloseall, fdopen, fopen, freopen, _fsopen

Example:  
#include <stdio.h>

void main()
{
    FILE *fp;

    fp = fopen( "stdio.h", "r" );
    if( fp != NULL ) {
        fclose( fp );
    }
}

Classification:  ANSI

Systems:  All, Netware
Synopsis:  
#include <stdio.h>
int fcloseall( void );

Description:  The fcloseall function closes all open stream files, except stdin, stdout, stderr, stdaux, and stdprn. This includes streams created (and not yet closed) by fdopen, fopen and freopen. The stdaux and stdprn files are not available for some Windows platforms.

Returns:  The fcloseall function returns the number of streams that were closed if no errors were encountered. When an error occurs, EOF is returned.

See Also:  fclose, fdopen, fopen, freopen, _fsopen

Example:  
#include <stdio.h>

void main()
{
    printf( "The number of files closed is %d\n", fcloseall() );
}

Classification: WATCOM

Systems:  All, Netware
**Synopsis:**

```c
#include <stdlib.h>
char *fcvt( double value,
    int ndigits,
    int *dec,
    int *sign );
char *__fcvt( double value,
    int ndigits,
    int *dec,
    int *sign );
wchar_t *__wfcvt( double value,
    int ndigits,
    int *dec,
    int *sign );
```

**Description:**

The `fcvt` function converts the floating-point number `value` into a character string. The parameter `ndigits` specifies the number of digits desired after the decimal point. The converted number will be rounded to this position.

The character string will contain only digits and is terminated by a null character. The integer pointed to by `dec` will be filled in with a value indicating the position of the decimal point relative to the start of the string of digits. A zero or negative value indicates that the decimal point lies to the left of the first digit. The integer pointed to by `sign` will contain 0 if the number is positive, and non-zero if the number is negative.

The `_fcvt` function is identical to `fcvt`. Use `_fcvt` for ANSI/ISO naming conventions.

The `_wfcvt` function is identical to `fcvt` except that it produces a wide-character string.

**Returns:**

The `fcvt` function returns a pointer to a static buffer containing the converted string of digits. Note: `ecvt` and `fcvt` both use the same static buffer.

**See Also:**

`ecvt`, `gcvt`, `printf`

**Example:**

```c
#include <stdio.h>
#include <stdlib.h>

void main()
{
    char *str;
    int dec, sign;

    str = fcvt( -123.456789, 5, &dec, &sign );
    printf( "str=%s, dec=%d, sign=%d\n", str, dec, sign );
}
```

produces the following:

```
str=12345679, dec=3, sign=-1
```

**Classification:**

_WATCOM_,

 `_fcvt` conforms to ANSI/ISO naming conventions

**Systems:**

`fcvt` - Math
 `_fcvt` - Math
_wfcvt - Math
Synopsis:

```
#include <stdio.h>
FILE *fdopen( int handle, const char *mode );
FILE *_fdopen( int handle, const char *mode );
FILE *_wfdopen( int handle, const wchar_t *mode );
```

Description:
The `fdopen` function associates a stream with the file handle `handle` which represents an opened file or device. The handle was returned by one of `creat`, `dup`, `dup2`, `open`, or `sopen`. The open mode `mode` must match the mode with which the file or device was originally opened.

The argument `mode` is described in the description of the `fopen` function.

The `_fdopen` function is identical to `fdopen`. Use `_fdopen` for ANSI/ISO naming conventions.

The `_wfdopen` function is identical to `fdopen` except that it accepts a wide character string for the second argument.

Returns:
The `fdopen` function returns a pointer to the object controlling the stream. This pointer must be passed as a parameter to subsequent functions for performing operations on the file. If the open operation fails, `fdopen` returns a NULL pointer. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `creat`, `_dos_open`, `dup`, `dup2`, `fopen`, `freopen`, `_fsopen`, `_grow_handles`, `_hdopen`, `open`, `_open_osfhandle`, `_popen`, `sopen`

Example:

```
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

void main()
{
    int handle;
    FILE *fp;

    handle = open( "file", O_RDONLY | O_TEXT );
    if( handle != -1 ) {
        fp = fdopen( handle, "r" );
        if( fp != NULL ) {
            /*
             * process the stream
             */
            fclose( fp );
        } else {
            close( handle );
        }
    }
}
```

Classification: `fdopen` is POSIX 1003.1
- `_fdopen` is not POSIX
- `_wfdopen` is not POSIX

Systems:
- `fdopen - All, Netware`
- `_fdopen - All, Netware`
- `_wfdopen - All`
Synopsis:  
#include <fenv.h>
int feclearexcept( int __excepts );

Description:  The feclearexcept function attempts to clear the supported floating-point exceptions represented by its argument.

Returns:  The feclearexcept function returns zero if the excepts argument is zero or if all the specified exceptions were successfully cleared. Otherwise, it returns a nonzero value.

See Also:  fegetexceptflag, feraiseexcept, fesetexceptflag, fetestexcept

Example:  
#include <fenv.h>

void main( void )
{
    feclearexcept( FE_OVERFLOW|FE_UNDERFLOW );
}

Classification:  C99
__fedisableexcept

Synopsis:  
#include <fenv.h>
void __fedisableexcept( int __excepts );

Description:  The __fedisableexcept function disables the specified floating point exceptions.

Returns:  No value is returned.

See Also:  __feenableexcept

Example:  
#include <fenv.h>

void main( void )
{
    __fedisableexcept( FE_DIVBYZERO );
}

Classification: WATCOM
Synopsis:
#include <fenv.h>
void __feenableexcept( int __excepts );

Description: The __feenableexcept function enables the specified floating point exceptions.

Returns: No value is returned.

See Also: __fedisableexcept

Example:
#include <fenv.h>

void main( void )
{
    __feenableexcept( FE_DIVBYZERO );
}

Classification: WATCOM
**fegetenv**

**Synopsis:**
```
#include <fenv.h>
int fegetenv( fenv_t *__envp );
```

**Description:** The `fegetenv` function attempts to store the current floating-point environment in the object pointed to by `envp`.

**Returns:** The `fegetenv` function returns zero if the environment was successfully stored. Otherwise, it returns a nonzero value.

**See Also:** `feholdexcept`, `fesetenv`, `feupdateenv`

**Example:**
```
#include <stdio.h>
#include <fenv.h>

void main( void )
{
    fenv_t env;
    fegetenv( &env );
}
```

**Classification:** C99
Synopsis: #include <fenv.h>
int fegetexceptflag( fexcept_t *__flagp, int __excepts );

Description: The fegetexceptflag function attempts to store a representation of the states of the floating-point status flags indicated by the argument excepts in the object pointed to by the argument flagp.

Valid exceptions are FE_INVALID FE_DENORMAL FE_DIVBYZERO FE_OVERFLOW
FE_UNDERFLOW FE_INEXACT

The value FE_ALL_EXCEPT is the logical OR of these values.

Returns: The fegetexceptflag function returns zero if the representation was successfully stored. Otherwise, it returns a nonzero value.

See Also: feclearexcept, feraiseexcept, fesetexceptflag, fetestexcept

Example: #include <fenv.h>

void main( void )
{
    fexcept_t flags;
    fegetexceptflag( &flags, FE_DIVBYZERO );
}

Classification: C99
**fegetround**

**Synopsis:**
```
#include <fenv.h>
int fegetround( void );
```

**Description:**
The `fegetround` function gets the current rounding direction.

**Returns:**
The `fegetround` function returns the value of the rounding direction macro representing the current rounding direction or a negative value if there is no such rounding direction macro or the current rounding direction is not determinable.

Valid rounding modes are
- `FE_TONEAREST`
- `FE_DOWNWARD`
- `FE_TOWARDZERO`
- `FE_UPWARD`

**See Also:**
`fesetround`

**Example:**
```
#include <stdio.h>
#include <fenv.h>

void main( void )
{
    int mode;
    mode = fegetround();
    if ( mode == FE_TONEAREST )
        printf( "Nearest\n" );
    else if ( mode == FE_DOWNWARD )
        printf( "Down\n" );
    else if ( mode == FE_TOWARDZERO )
        printf( "To Zero\n" );
    else if ( mode == FE_UPWARD )
        printf( "Up\n" );
}
```

**Classification:**
C99
Synopsis:  
#include <fenv.h>
int feholdexcept( fenv_t *__envp );

Description:  The feholdexcept function saves the current floating-point environment in the object pointed to by
envp, clears the floating-point status flags, and then installs a non-stop (continue on floating-point
exceptions) mode, if available, for all floating-point exceptions.

Returns:  The feholdexcept function returns zero if and only if non-stop floating-point exception handling
was successfully installed.

See Also:  fegetenv, fesetenv, feupdateenv

Example:  
#include <fenv.h>

void main( void )
{
    fenv_t env;
    feholdexcept( &env );
}

Classification:  C99
Synopsis:  
#include <stdio.h>
int feof( FILE *fp );

Description:  The feof function tests the end-of-file indicator for the stream pointed to by fp. Because this indicator is set when an input operation attempts to read past the end of the file the feof function will detect the end of the file only after an attempt is made to read beyond the end of the file. Thus, if a file contains 10 lines, the feof will not detect end of file after the tenth line is read; it will detect end of file once the program attempts to read more data.

Returns:  The feof function returns non-zero if the end-of-file indicator is set for fp.

See Also:  clearerr, ferror, fopen, freopen, perror, read, strerror

Example:  
#include <stdio.h>

void process_record( char *buf )
{
    printf( "%s\n", buf );
}

void main()
{
    FILE *fp;
    char buffer[100];

    fp = fopen( "file", "r" );
    fgets( buffer, sizeof( buffer ), fp );
    while( ! feof( fp ) ) {
        process_record( buffer );
        fgets( buffer, sizeof( buffer ), fp );
    }
    fclose( fp );
}

Classification:  ANSI

Systems:  All, Netware
Synopsis:  
#include <fenv.h>
int feraiseexcept( int __excepts );

Description: The feraiseexcept function attempts to raise the supported floating-point exceptions represented by its argument.

Returns: The feraiseexcept function returns zero if the excepts argument is zero or if all the specified exceptions were successfully raised. Otherwise, it returns a nonzero value.

See Also: feclearexcept, fegetexceptflag, fetestexcept

Example:  
#include <fenv.h>

void main( void )
{
    feraiseexcept( FE_DIVBYZERO );
}

Classification: C99
**Synopsis:**

```c
#include <stdio.h>
int ferror( FILE *fp );
```

**Description:**

The `ferror` function tests the error indicator for the stream pointed to by `fp`.

**Returns:**

The `ferror` function returns non-zero if the error indicator is set for `fp`.

**See Also:**

`clearerr`, `feof`, `perror`, `strerror`

**Example:**

```c
#include <stdio.h>

void main()
{
    FILE *fp;
    int c;

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        c = fgetc( fp );
        if( ferror( fp ) ) {
            printf( "Error reading file\n" );
        }
    }
    fclose( fp );
}
```

**Classification:** ANSI

**Systems:** All, Netware
Synopsis:  
\#include <fenv.h>
int fesetenv( const fenv_t *__envp );

Description:  The fesetenv function attempts to establish the floating-point environment represented by the object pointed to by envp. The argument envp shall point to an object set by a call to fegetenv or feholdexcept, or equal the FE_DFL_ENV macro. Note that fesetenv merely installs the state of the floating-point status flags represented through its argument, and does not raise these floating-point exceptions.

Returns:  The fesetenv function returns zero if the environment was successfully established. Otherwise, it returns a nonzero value.

See Also:  fegetenv, feholdexcept, feupdateenv

Example:  
\#include <fenv.h>

void main( void )
{
    fenv_t env;
    fegetenv( &env );
    fesetenv( FE_DFL_ENV );
    fesetenv( &env );
}

Classification: C99
**fesetexceptflag**

**Synopsis:**
```
#include <fenv.h>
int fesetexceptflag( const fexcept_t *__flagp, int __excepts );
```

**Description:** The `fesetexceptflag` function attempts to set the floating-point status flags indicated by the argument `excepts` to the states stored in the object pointed to by `flagp`. The value of `*flagp` shall have been set by a previous call to `fegetexceptflag` whose second argument represented at least those floating-point exceptions represented by the argument `excepts`. This function does not raise floating-point exceptions, but only sets the state of the flags.

**Returns:** The `fesetexceptflag` function returns zero if the `excepts` argument is zero or if all the specified flags were successfully set to the appropriate state. Otherwise, it returns a nonzero value.

**See Also:** `feclearexcept`, `fegetexceptflag`, `fetestexcept`

**Example:**
```
#include <fenv.h>

void main( void )
{
    fexcept_t flags;
    fgetexceptflag( &flags, FE_DENORMAL|FE_INVALID );
    fsetexceptflag( &flags, FE_INVALID );
}
```

**Classification:** C99
Synopsis:  #include <fenv.h>
int fesetround( int __round );

Description:  The fesetround function establishes the rounding direction represented by its argument round. If the argument is not equal to the value of a rounding direction macro, the rounding direction is not changed.

Returns:  The fesetround function returns a zero value if and only if the requested rounding direction was established.

See Also:  fegetround

Example:  #include <fenv.h>

void main( void )
{
    fesetround( FE_UPWARD );
}

Classification:  C99
Synopsis: #include <fenv.h>
    int fetestexcept( int __excepts );

Description: The fetestexcept function determines which of a specified subset of the floatingpoint exception flags are currently set. The excepts argument specifies the floating point status flags to be queried.

Returns: The fetestexcept function returns the value of the bitwise OR of the floating-point exception macros corresponding to the currently set floating-point exceptions included in excepts.

See Also: feclearexcept, fegetexceptflag, feraiseexcept, fesetexceptflag

Example: #include <stdio.h>
    #include <fenv.h>

    void main( void )
    {
        int excepts;
        feclearexcept( FE_DIVBYZERO );

        ...code that may cause a divide by zero exception

        excepts = fetestexcept( FE_DIVBYZERO );
        if ( excepts & FE_DIVBYZERO)
            printf( "Divide by zero occurred\n" );
    }

Classification: C99
Synopsis:  
```
#include <fenv.h>
int feupdateenv( const fenv_t *__envp );
```

Description:  The `feupdateenv` function attempts to save the currently raised floating-point exceptions in its automatic storage, installs the floating-point environment represented by the object pointed to by `envp`, and then raises the saved floating-point exceptions. The argument `envp` shall point to an object set by a call to `feholdexcept` or `fegetenv`, or equal a floating-point environment macro.

Returns:  The `feupdateenv` function returns zero if all the actions were successfully carried out. Otherwise, it returns a nonzero value.

See Also:  `fegetenv`, `feholdexcept`, `fesetenv`

Example:  
```
#include <fenv.h>

void main( void )
{
    fenv_t env;
    fegetenv( &env );
    fesetenv( FE_DFL_ENV );
    feupdateenv( &env );
}
```

Classification: C99
**Synopsis:**

```c
#include <stdio.h>
int fflush( FILE *fp );
```

**Description:** If the file `fp` is open for output or update, the `fflush` function causes any unwritten data to be written to the file. If the file `fp` is open for input or update, the `fflush` function undoes the effect of any preceding `ungetc` operation on the stream. If the value of `fp` is `NULL`, then all files that are open will be flushed.

**Returns:** The `fflush` function returns non-zero if a write error occurs and zero otherwise. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

**See Also:** `fgetc`, `fgets`, `flushall`, `fopen`, `getc`, `gets`, `setbuf`, `setvbuf`, `ungetc`

**Example:**

```c
#include <stdio.h>
#include <conio.h>

void main()
{
    printf( "Press any key to continue..." );
    fflush( stdout );
    getch();
}
```

**Classification:** ANSI

**Systems:** All, Netware
Synopsis:  #include <strings.h>
            int ffs( int i );

Description: The ffs finds the first bit set, beginning with the least
            significant bit, in i.  Bits are numbered starting
            at one (the least significant bit).

Returns:   The ffs function returns the index of the first bit set. If i is 0, ffs returns zero.

See Also:  _lrotl, _lrotr, _rotl, _rotr

Example:  #include <stdio.h>
           #include <strings.h>

           int main( void )
           {
               printf( "%d\n", ffs( 0 ) );
               printf( "%d\n", ffs( 16 ) );
               printf( "%d\n", ffs( 127 ) );
               printf( "%d\n", ffs( -16 ) );
               return( 0 );
           }

           produces the following:

           0
           5
           1
           5

Classification: POSIX

Systems:   All, Netware
Synopsis:
#include <stdio.h>
int fgetc( FILE *fp );
#include <stdio.h>
#include <wchar.h>
wint_t fgetwc( FILE *fp );

Description: The fgetc function gets the next character from the file designated by fp. The character is signed.
The fgetwc function is identical to fgetc except that it gets the next multibyte character (if present) from the input stream pointed to by fp and converts it to a wide character.

Returns: The fgetc function returns the next character from the input stream pointed to by fp. If the stream is at end-of-file, the end-of-file indicator is set and fgetc returns EOF. If a read error occurs, the error indicator is set and fgetc returns EOF.
The fgetwc function returns the next wide character from the input stream pointed to by fp. If the stream is at end-of-file, the end-of-file indicator is set and fgetwc returns WEOF. If a read error occurs, the error indicator is set and fgetwc returns WEOF. If an encoding error occurs, errno is set to EILSEQ and fgetwc returns WEOF.

When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also: fgetchar, fgets, fopen, getc, getchar, gets, ungetc

Example: #include <stdio.h>

    void main()
    {
        FILE *fp;
        int c;

        fp = fopen( "file", "r" );
        if( fp != NULL ) {
            while( (c = fgetc( fp )) != EOF )
                fputc( c, stdout );
            fclose( fp );
        }
    }

Classification: fgetc is ANSI
                fgetwc is ANSI

Systems: fgetc - All, Netware
         fgetwc - All

248 Library Functions and Macros
Synopsis:    #include <stdio.h>
    int fgetchar( void );
    int _fgetchar( void );
    wint_t _fgetwchar( void );

Description: The fgetchar function is equivalent to fgetc with the argument stdin.
              The _fgetchar function is identical to fgetchar. Use _fgetchar for ANSI naming
              conventions.
              The _fgetwchar function is identical to fgetchar except that it gets the next multibyte character
              (if present) from the input stream pointed to by stdin and converts it to a wide character.

Returns:    The fgetchar function returns the next character from the input stream pointed to by stdin. If the
              stream is at end-of-file, the end-of-file indicator is set and fgetchar returns EOF. If a read error
              occurs, the error indicator is set and fgetchar returns EOF.

              The _fgetwchar function returns the next wide character from the input stream pointed to by
              stdin. If the stream is at end-of-file, the end-of-file indicator is set and _fgetwchar returns
              WEOF. If a read error occurs, the error indicator is set and _fgetwchar returns WEOF. If an
              encoding error occurs, errno is set to EILSEQ and _fgetwchar returns WEOF.

              When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also: fgetc, fgets, fopen, getc, getchar, gets, ungetc

Example:    #include <stdio.h>
              void main()
              {
                FILE *fp;
                int c;

                fp = freopen( "file", "r", stdin );
                if( fp != NULL ) {
                  while( (c = fgetchar()) != EOF )
                    fputchar(c);
                }
                fclose( fp );
              }

Classification: WATCOM

Systems:    fgetchar - All, Netware
              _fgetchar - All, Netware
              _fgetwchar - All
Synopsis:  
#include <stdio.h>
int fgetpos( FILE *fp, fpos_t *pos );

Description:  The fgetpos function stores the current position of the file fp in the object pointed to by pos. The value stored is usable by the fsetpos function for repositioning the file to its position at the time of the call to the fgetpos function.

Returns:  The fgetpos function returns zero if successful, otherwise, the fgetpos function returns a non-zero value. When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also:  fopen, fseek, fsetpos, ftell

Example:  
#include <stdio.h>

void main()
{
    FILE *fp;
    fpos_t position;
    auto char buffer[80];

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        fgetpos( fp, &position ); /* get position */
        fgets( buffer, 80, fp );  /* read record */
        fsetpos( fp, &position ); /* set position */
        fgets( buffer, 80, fp );  /* read same record */
        fclose( fp );
    }
}

Classification:  ANSI

Systems:  All, Netware
Synopsis:

```
#include <stdio.h>
char *fgets( char *buf, int n, FILE *fp );
#include <stdio.h>
#include <wchar.h>
wchar_t *fgetws( wchar_t *buf, int n, FILE *fp );
```

Description: The `fgets` function gets a string of characters from the file designated by `fp` and stores them in the array pointed to by `buf`. The `fgets` function stops reading characters when end-of-file is reached, or when a newline character is read, or when `n-1` characters have been read, whichever comes first. The new-line character is not discarded. A null character is placed immediately after the last character read into the array.

The `fgetws` function is identical to `fgets` except that it gets a string of multibyte characters (if present) from the input stream pointed to by `fp`, converts them to wide characters, and stores them in the wide-character array pointed to by `buf`. In this case, `n` specifies the number of wide characters, less one, to be read.

A common programming error is to assume the presence of a new-line character in every string that is read into the array. A new-line character will not be present when more than `n-1` characters occur before the new-line. Also, a new-line character may not appear as the last character in a file, just before end-of-file.

The `gets` function is similar to `fgets` except that it operates with `stdin`, it has no size argument, and it replaces a newline character with the null character.

Returns: The `fgets` function returns `buf` if successful. `NULL` is returned if end-of-file is encountered, or a read error occurs. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fgetc`, `fgetchar`, `fopen`, `getc`, `getchar`, `gets`, `ungetc`

Example: 

```
#include <stdio.h>

void main()
{
    FILE *fp;
    char buffer[80];
    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        while( fgets( buffer, 80, fp ) != NULL )
            fputs( buffer, stdout );
    fclose( fp );
}
```

Classification: `fgets` is ANSI
`fgetws` is ANSI

Systems: `fgets` - All, Netware
`fgetws` - All
Synopsis:  
#include <math.h>
extern int _fieeetomsbin( float *src, float *dest );

Description: The _fieeetomsbin function loads the float pointed to by src in IEEE format and converts it to Microsoft binary format, storing the result into the float pointed to by dest.

For _fieeetomsbin, IEEE Nan’s and Infinities will cause overflow. IEEE denormals will be converted if within range. Otherwise, they will be converted to 0 in the Microsoft binary format.

The range of Microsoft binary format floats is 2.938736e-39 to 1.701412e+38. The range of Microsoft binary format doubles is 2.938735877056e-39 to 1.701411834605e+38.

Microsoft Binary Format was used by early versions of Microsoft QuickBASIC before coprocessors became standard.

Returns: The _fieeetomsbin function returns 0 if the conversion was successful. Otherwise, it returns 1 if conversion would cause an overflow.

See Also: _dieeetomsbin, _dmsbintoieee, _fmsbintoieee

Example:  
#include <stdio.h>
#include <math.h>

void main()
{  
    float fieee, fmsb;
    double dieee, dmsb;

    fieee = 0.5;
    dieee = -2.0;

    /* Convert IEEE format to Microsoft binary format */
    _fieeetomsbin( &fieee, &fmsb );
    _dieeetomsbin( &dieee, &dmsb );

    /* Convert Microsoft binary format back to IEEE format */
    _fmsbintoieee( &fmsb, &fieee );
    _dmsbintoieee( &dmsb, &dieee );

    /* Display results */
    printf( "fieee = %f, dieee = %f\n", fieee, dieee );
}

produces the following:

fieee = 0.500000, dieee = -2.000000

Classification: WATCOM

Systems: All, Netware

252 Library Functions and Macros
Synopsis:  

```
#include <io.h>
long filelength( int handle );
long _filelength( int handle );
__int64 _filelengthi64( int handle );
```

Description:  
The `filelength` function returns, as a 32-bit long integer, the number of bytes in the opened file indicated by the file handle `handle`.

The `_filelengthi64` function returns, as a 64-bit integer, the number of bytes in the opened file indicated by the file handle `handle`.

The `_filelength` function is identical to `filelength`. Use `_filelength` for ANSI/ISO naming conventions.

Returns:  
If an error occurs in `filelength`, (-1L) is returned.

If an error occurs in `_filelengthi64`, (-1I64) is returned.

When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

Otherwise, the number of bytes written to the file is returned.

See Also:  
`fstat`, `lseek`, `tell`

Example:  
```
#include <sys/types.h>
#include <fcntl.h>
#include <stdio.h>
#include <io.h>

void main( void )
{
    int handle;

    /* open a file for input */
    handle = open( "file", O_RDONLY | O_TEXT );
    if( handle != -1 ) {
        printf( "Size of file is %ld bytes\n", filelength( handle ) );
        close( handle );
    }
}
```

produces the following:

Size of file is 461 bytes

Classification:  
WATCOM

Systems:  
- filelength - All, Netware
- _filelength - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
- _filelengthi64 - All
FILENAME_MAX

Synopsis:  
#include <stdio.h>
#define FILENAME_MAX 123

Description:  
The FILENAME_MAX macro is the size of an array of char big enough to hold a string naming any file that the implementation expects to open; If there is no practical file name length limit, FILENAME_MAX is the recommended size of such an array. As file name string contents must meet other system-specific constraints, some strings of length FILENAME_MAX may not work.

FILENAME_MAX typically sizes an array to hold a file name.

Returns:  
The FILENAME_MAX macro returns a positive integer value.

Example:  
#include <stdio.h>
#include <string.h>

int main( int argc, char *argv[] )
{
    if( argc ) {
        char fname[FILENAME_MAX];
        strcpy( fname, argv[0] );
        puts( fname );
    }
    return( 0 );
}

Classification: ANSI

Systems: MACRO
Synopsis: #include <stdio.h>
int fileno( FILE *stream );

Description: The `fileno` function returns the number of the file handle for the file designated by `stream`. This number can be used in POSIX input/output calls anywhere the value returned by `open` can be used. The following symbolic values in `<io.h>` define the file handles that are associated with the C language `stdin`, `stdout`, `stderr`, `stdaux`, and `stdprn` files when the application is started. The `stdaux` and `stdprn` files are not available for Win32.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>STDIN_FILENO</td>
<td>Standard input file number, <code>stdin</code> (0)</td>
</tr>
<tr>
<td>STDOUT_FILENO</td>
<td>Standard output file number, <code>stdout</code> (1)</td>
</tr>
<tr>
<td>STDERR_FILENO</td>
<td>Standard error file number, <code>stderr</code> (2)</td>
</tr>
<tr>
<td>STDAUX_FILENO</td>
<td>Standard auxiliary file number, <code>stdaux</code> (3)</td>
</tr>
<tr>
<td>STDPRN_FILENO</td>
<td>Standard printer file number, <code>stdprn</code> (4)</td>
</tr>
</tbody>
</table>

Returns: The `fileno` function returns the number of the file handle for the file designated by `stream`. If an error occurs, a value of -1 is returned and `errno` is set to indicate the error.

See Also: `open`

Example: #include <stdio.h>

```c
void main()
{
    FILE *stream;

    stream = fopen( "file", "r" );
    printf( "File number is %d\n", fileno( stream ) );
    fclose( stream );
}
```

produces the following:

```
File number is 7
```

Classification: POSIX 1003.1

Systems: All, Netware
Synopsis: 
#include <io.h>
int _findclose( long handle );

Description: The _findclose function closes the directory of filenames established by a call to the _findfirst function. The handle argument was returned by the _findfirst function.

Returns: If successful, _findclose returns 0 otherwise, _findclose and returns -1 and sets errno to one of the following values:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENOENT</td>
<td>No matching files</td>
</tr>
</tbody>
</table>

See Also: _dos_find..., _findfirst, _findnext, closedir, opendir, readdir

Example: #include <stdio.h>
#include <io.h>

void main()
{
    struct _finddata_t fileinfo;
    long handle;
    int rc;

    /* Display name and size of *.c files */
    handle = _findfirst( "*.c", &fileinfo );
    rc = handle;
    while( rc != -1 ) {
        printf( "%14s %10ld\n", fileinfo.name, fileinfo.size );
        rc = _findnext( handle, &fileinfo );
    }
    _findclose( handle );
}

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

```c
#include <io.h>
long _findfirst( const char *filespec,  
    struct _finddata_t *fileinfo );
long _findfirsti64( const char *filespec,  
    struct _finddatai64_t *fileinfo );
long _wfindfirst( const wchar_t *filespec,  
    struct _wfinddata_t *fileinfo );
long _wfindfirsti64( const wchar_t *filespec,  
    struct _wfinddatai64_t *fileinfo );
```

Description:
The `_findfirst` function returns information on the first file whose name matches the `filespec` argument. The `filespec` argument may contain wildcard characters ('?' and '*'). The information is returned in a `_finddata_t` structure pointed to by `fileinfo`.

```c
struct _finddata_t {
    unsigned attrib;
    time_t time_create; /* -1 for FAT file systems */
    time_t time_access; /* -1 for FAT file systems */
    time_t time_write;
    _fsize_t size;
    char name[_MAX_PATH];
};
```

The `_findfirsti64` function returns information on the first file whose name matches the `filespec` argument. It differs from the `_findfirst` function in that it returns a 64-bit file size. The `filespec` argument may contain wildcard characters ('?' and '*'). The information is returned in a `_finddatai64_t` structure pointed to by `fileinfo`.

```c
struct _finddatai64_t {
    unsigned attrib;
    time_t time_create; /* -1 for FAT file systems */
    time_t time_access; /* -1 for FAT file systems */
    time_t time_write;
    __int64 size;        /* 64-bit size info */
    char name[_MAX_PATH];
};
```

The wide-character `_wfindfirst` function is similar to the `_findfirst` function but operates on wide-character strings.

```c
struct _wfinddata_t {
    unsigned attrib;
    time_t time_create; /* -1 for FAT file systems */
    time_t time_access; /* -1 for FAT file systems */
    time_t time_write;
    _fsize_t size;
    wchar_t name[_MAX_PATH];
};
```

The wide-character `_wfindfirsti64` function is similar to the `_findfirsti64` function but operates on wide-character strings. It differs from the `_wfindfirst` function in that it returns a 64-bit file size.

```c
struct _wfinddatai64_t {
    unsigned attrib;
    time_t time_create; /* -1 for FAT file systems */
    time_t time_access; /* -1 for FAT file systems */
    time_t time_write;
    __int64 size;        /* 64-bit size info */
    wchar_t name[_MAX_PATH];
};
```
struct _wfinddata64_t {
    unsigned       attrib;
    time_t         time_create;  /* -1 for FAT file systems */
    time_t         time_access;  /* -1 for FAT file systems */
    time_t         time_write;
    __int64        size;        /* 64-bit size info */
    wchar_t        name[_MAX_PATH];
};

Returns: If successful, _findfirst returns a unique search handle identifying the file or group of files matching the filespec specification, which can be used in a subsequent call to _findnext or to _findclose. Otherwise, _findfirst and returns -1 and sets errno to one of the following values:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENOENT</td>
<td>No matching files</td>
</tr>
<tr>
<td>EINVAL</td>
<td>Invalid filename specification</td>
</tr>
</tbody>
</table>

See Also: _dos_find..., _findclose, _findnext, closedir, opendir, readdir

Example: #include <stdio.h>
#include <io.h>

void main()
{
    struct _finddata_t fileinfo;
    long handle;
    int rc;

    /* Display name and size of "*.c" files */
    handle = _findfirst( "*.c", &fileinfo );
    rc = handle;
    while( rc != -1 ) {
        printf( "%14s %10ld
", fileinfo.name,
                fileinfo.size );
        rc = _findnext( handle, &fileinfo );
    }
    _findclose( handle );
}

Classification: _findfirst is DOS
_findfirst is not DOS
_findfirst64 is not DOS
_wfindfirst64 is not DOS

Systems: _findfirst - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_findfirst64 - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_wfindfirst - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_wfindfirst64 - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

258 Library Functions and Macros
Synopsis:

```c
#include <io.h>
int _findnext( long handle,
    struct _finddata_t *fileinfo );
int _findnexti64( long handle,
    struct _finddatai64_t *fileinfo );
int _wfindnext( long handle,
    struct _wfinddata_t *fileinfo );
int _wfindnexti64( long handle,
    struct _wfinddatai64_t *fileinfo );
```

Description:
The _findnext function returns information on the next file whose name matches the `filespec` argument that was specified in a call to the _findfirst function. The `handle` argument was returned by the _findfirst function. The information is returned in a _finddata_t structure pointed to by `fileinfo`.

```c
struct _finddata_t {
    unsigned attrib;
    time_t time_create; /* -1 for FAT file systems */
    time_t time_access; /* -1 for FAT file systems */
    time_t time_write;
    _fsize_t size;
    char name[_MAX_PATH];
};
```

The _findnexti64 function returns information on the next file whose name matches the `filespec` argument that was specified in a call to the _findfirsti64 function. It differs from the _findnext function in that it returns a 64-bit file size. The `handle` argument was returned by the _findfirsti64 function. The information is returned in a _finddatai64_t structure pointed to by `fileinfo`.

```c
struct _finddatai64_t {
    unsigned attrib;
    time_t time_create; /* -1 for FAT file systems */
    time_t time_access; /* -1 for FAT file systems */
    time_t time_write;
    __int64 size; /* 64-bit size info */
    char name[_MAX_PATH];
};
```

The wide-character _wfindnext function is similar to the _findnext function but operates on wide-character strings.

```c
struct _wfinddata_t {
    unsigned attrib;
    time_t time_create; /* -1 for FAT file systems */
    time_t time_access; /* -1 for FAT file systems */
    time_t time_write;
    _fsize_t size;
    wchar_t name[_MAX_PATH];
};
```

The wide-character _wfindnexti64 function is similar to the _findnexti64 function but operates on wide-character strings. It differs from the _wfindnext function in that it returns a 64-bit file size.
struct _wfinddatai64_t {
  unsigned  attrib;
  time_t    time_create;  /* -1 for FAT file systems */
  time_t    time_access;  /* -1 for FAT file systems */
  time_t    time_write;
  __int64   size; /* 64-bit size info */
  wchar_t   name[_MAX_PATH];
};

Returns: If successful, _findnext returns 0 otherwise, _findnext and returns -1 and sets errno to one of the following values:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENOENT</td>
<td>No matching files</td>
</tr>
</tbody>
</table>

See Also: _dos_find..., _findclose, _findfirst, closedir, opendir, readdir

Example:
```
#include <stdio.h>
#include <io.h>

void main()
{
  struct _finddata_t  fileinfo;
  long                handle;
  int                 rc;

  /* Display name and size of "*.c" files */
  handle = _findfirst( "*.c", &fileinfo );
  rc = handle;
  while( rc != -1 ) {
    printf( "%14s %10ld\n", fileinfo.name,
            fileinfo.size );
    rc = _findnext( handle, &fileinfo );
  }
  _findclose( handle );
}
```

Classification: _findnext is DOS
_findnext is not DOS
_findnexti64 is not DOS
_wfindnexti64 is not DOS

Systems:
_findnext - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_findnexti64 - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_wfindnext - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_wfindnexti64 - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <float.h>  
int _finite( double x );

Description:  The _finite function determines whether the double precision floating-point argument is a valid number (i.e., not infinite and not a NAN).

Returns:  The _finite function returns 0 if the number is not valid and non-zero otherwise.

See Also:  _clear87, _control87, _controlfp, _fpreset, printf, _status87

Example:  
#include <stdio.h>  
#include <float.h>

void main()  
{
    printf( "%s\n", (_finite( 1.797693134862315e+308 ) )
        ? "Valid" : "Invalid" );
    printf( "%s\n", (_finite( 1.797693134862320e+308 ) )
        ? "Valid" : "Invalid" );
}

produces the following:

Valid
Invalid

Classification:  WATCOM

Systems:  Math


Synopsis:
#include <graph.h>
short _FAR _floodfill( short x, short y,
short stop_color );

short _FAR _floodfill_w( double x, double y,
short stop_color );

Description:
The _floodfill functions fill an area of the screen. The _floodfill function uses the view
coordinate system. The _floodfill_w function uses the window coordinate system.
The filling starts at the point (x, y) and continues in all directions: when a pixel is filled, the
neighbouring pixels (horizontally and vertically) are then considered for filling. Filling is done using
the current color and fill mask. No filling will occur if the point (x, y) lies outside the clipping region.
If the argument stop_color is a valid pixel value, filling will occur in each direction until a pixel is
encountered with a pixel value of stop_color. The filled area will be the area around (x, y), bordered
by stop_color. No filling will occur if the point (x, y) has the pixel value stop_color.
If stop_color has the value (-1), filling occurs until a pixel is encountered with a pixel value different
from the pixel value of the starting point (x, y). No filling will occur if the pixel value of the point
(x, y) is the current color.

Returns: The _floodfill functions return zero when no filling takes place; a non-zero value is returned to
indicate that filling has occurred.

See Also: _setcliprgn, _setcolor, _setfillmask, _setplotaction

Example:
#include <conio.h>
#include <graph.h>

main()
{
    _setvideomode( _VRES16COLOR );
    _setcolor( 1 );
    _ellipse( _GBORDER, 120, 90, 520, 390 );
    _setcolor( 2 );
    _floodfill( 320, 240, 1 );
    getch();
    _setvideomode( _DEFAULTMODE );
}

Classification: _floodfill is PC Graphics

Systems: _floodfill - DOS, QNX
_floodfill_w - DOS, QNX
Synopsis:  
#include <math.h>

double floor( double x );

Description:  The floor function computes the largest integer not greater than x.

Returns:  The floor function computes the largest integer not greater than x, expressed as a double.

See Also:  ceil, fmod

Example:  
#include <stdio.h>
#include <math.h>

void main()
{
    printf( "%f\n", floor( -3.14 ) );
    printf( "%f\n", floor( -3. ) );
    printf( "%f\n", floor( 0. ) );
    printf( "%f\n", floor( 3.14 ) );
    printf( "%f\n", floor( 3. ) );
}

produces the following:

-4.000000
-3.000000
0.000000
3.000000
3.000000

Classification:  ANSI

Systems:  Math
flushall

Synopsis:     #include <stdio.h>
            int flushall( void );

Description: The flushall function clears all buffers associated with input streams and writes any buffers
             associated with output streams. A subsequent read operation on an input file causes new data to be read
             from the associated file or device.

             Calling the flushall function is equivalent to calling the fflush for all open stream files.

Returns:     The flushall function returns the number of open streams. When an output error occurs while
             writing to a file, the errno global variable will be set.

See Also:    fopen, fflush

Example:     #include <stdio.h>
             void main()
             {
                 printf( "The number of open files is %d\n",
                        flushall() );
             }

             produces the following:

             The number of open files is 4

Classification: WATCOM

Systems:     All, Netware
Synopsis:  
#include <math.h>
double fmod( double x, double y );

Description:  
The fmod function computes the floating-point remainder of x/y, even if the quotient x/y is not representable.

Returns:  
The fmod function returns the value x - (i * y), for some integer i such that, if y is non-zero, the result has the same sign as x and magnitude less than the magnitude of y. If the value of y is zero, then the value returned is zero.

See Also:  
ceil, fabs, floor

Example:  
#include <stdio.h>
#include <math.h>

void main()
{
    printf( "%f\n", fmod(  4.5,  2.0 ) );
    printf( "%f\n", fmod( -4.5,  2.0 ) );
    printf( "%f\n", fmod(  4.5, -2.0 ) );
    printf( "%f\n", fmod( -4.5, -2.0 ) );
}

produces the following:

0.500000
-0.500000
0.500000
-0.500000

Classification:  ANSI

Systems:  Math
Synopsis:  
#include <math.h>
extern int _fmsbintoieee( float *src, float *dest );

Description:  The _fmsbintoieee function loads the float pointed to by src in Microsoft binary format and converts it to IEEE format, storing the result into the float pointed to by dest.

The range of Microsoft binary format floats is 2.938736e-39 to 1.701412e+38. The range of Microsoft binary format doubles is 2.938735877056e-39 to 1.701411834605e+38.

Microsoft Binary Format was used by early versions of Microsoft QuickBASIC before coprocessors became standard.

Returns:  The _fmsbintoieee function returns 0 if the conversion was successful. Otherwise, it returns 1 if conversion would cause an overflow.

See Also:  _dieeetomsbin, _dmsbintoieee, _fieeetomsbin

Example:  
#include <stdio.h>
#include <math.h>

void main()
{
    float fieee, fmsb;
    double dieee, dmsb;

    fieee = 0.5;
    dieee = -2.0;

    /* Convert IEEE format to Microsoft binary format */
    _fieeetomsbin( &fieee, &fmsb );
    _dieeetomsbin( &dieee, &dmsb );

    /* Convert Microsoft binary format back to IEEE format */
    _fmsbintoieee( &fmsb, &fieee );
    _dmsbintoieee( &dmsb, &dieee );

    /* Display results */
    printf( "fieee = %f, dieee = %f\n", fieee, dieee );
}

produces the following:

fieee = 0.500000, dieee = -2.000000

Classification:  WATCOM

Systems:  All, Netware
fnmatch

Synopsis:

```c
#include <fnmatch.h>
int fnmatch( const char *pattern,
            const char *string, int flags );
```

Description: The `fnmatch` function checks the string specified by the `string` argument to see if it matches the pattern specified by the `pattern` argument.

The `flag` argument is a bitwise inclusive OR of the bits described below. It modifies the interpretation of `pattern` and `string`.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FNM_PATHNAME</td>
<td>If set, a path separator in <code>string</code> is explicitly matched by a slash in <code>pattern</code>. It isn't matched by either the asterisk or question mark special characters, or by a bracket expression.</td>
</tr>
<tr>
<td>FNM_PERIOD</td>
<td>If set, a leading period in <code>string</code> matches a period in <code>pattern</code>, where the definition of &quot;leading&quot; depends on FNM_PATHNAME:</td>
</tr>
<tr>
<td></td>
<td>- If FNM_PATHNAME is set, a period is leading if it’s the first character in <code>string</code>, or if it immediately follows a path separator.</td>
</tr>
<tr>
<td></td>
<td>- If FNM_PATHNAME isn’t set, a period is leading only if it’s the first character in <code>string</code>.</td>
</tr>
<tr>
<td>FNM_NOESCAPE</td>
<td>If set, disables backslash escaping:</td>
</tr>
<tr>
<td></td>
<td>- If FNM_NOESCAPE isn’t set in flags, a backslash character () in <code>pattern</code> followed by any other character matches that second character in <code>string</code>. In particular, \ matches a backslash in <code>string</code>.</td>
</tr>
<tr>
<td></td>
<td>- If FNM_NOESCAPE is set, a backslash character is treated as an ordinary character.</td>
</tr>
<tr>
<td>FNM_IGNORECASE</td>
<td>If set, the matching is case-insensitive.</td>
</tr>
<tr>
<td>FNM_CASEFOLD</td>
<td>A synonym for FNM_IGNORECASE.</td>
</tr>
<tr>
<td>FNM_LEADING_DIR</td>
<td>If set, the final path separator and any following characters in <code>string</code> are ignored during matching.</td>
</tr>
</tbody>
</table>

A pattern-matching special character that is quoted is a pattern that matches the special character itself. When not quoted, such special characters have special meaning in the specification of patterns. The pattern-matching special characters and the contexts in which they have their special meaning are as follows:

- `?`  a `?` is a pattern that matches any printable or nonprintable character except `<newline>`.
- `*`  the `*` matches any string, including the null string.
- `[br_exp]`  a pattern that matches a single character as per Regular Expression Bracket Expressions (1003.2 2.9.1.2) except that
• The exclamation point character (!) replaces the circumflex character (^) in its role as a nonmatching list in the regular expression notation.
• The backslash is used as an escape character within bracket expressions.

The ?, * and [ characters aren’t special when used inside a bracket expression.

The concatenation of patterns matching a single character is a valid pattern that matches the concatenation of the single characters matched by each of the concatenated patterns. For example, the pattern a[bc] matches the strings ab and ac.

The concatenation of one or more patterns matching a single character with one or more asterisks (*) is a valid pattern. In such patterns, each asterisk matches a string of zero or more characters, up to the first character that matches the character following the asterisk in the pattern. For example, the pattern a*d matches the strings ad, abd, and abcd, but not the string abc.

When asterisk is the first or last character in a pattern, it matches zero or more characters that precede or follow the characters matched by the remainder of the pattern. For example, the pattern a*d* matches the strings ad, abcd, abcdef, aaaaad and adddd. The pattern *a*d matches the strings ad, abd, efabcd, aaaaad and adddd.

Returns: The fnmatch function returns zero when string matches the pattern specified by pattern. If there is no match, FNM_NOMATCH is returned. If an error occurs, fnmatch returns another non-zero value.

Example:
```c
#include <stdio.h>
#include <fnmatch.h>
#include <stdlib.h>
#include <limits.h>

int main( int argc, char **argv )
{
    int     i;
    char    buffer[PATH_MAX+1];

    while( gets( buffer ) ) {
        for( i = 1; i < argc; i++ ) {
            if( fnmatch( argv[i], buffer, 0 ) == 0 ) {
                printf( "'%s' matches pattern '%s'\n", buffer, argv[i] );
                break;
            }
        }
    }
    return( EXIT_SUCCESS );
}
```

Classification: POSIX 1003.2

Systems: All, Netware
Synopsis:  

```c
#include <stdio.h>
FILE *fopen( const char *filename, const char *mode );
FILE * _wfopen( const wchar_t *filename, const wchar_t *mode );
```

Safer C: The Safer C Library extension provides the `fopen_s` function which is a safer alternative to `fopen`. This newer `fopen_s` function is recommended to be used instead of the traditional "unsafe" `fopen` function.

Description: The `fopen` function opens the file whose name is the string pointed to by `filename`, and associates a stream with it. The argument `mode` points to a string beginning with one of the following sequences:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;r&quot;</td>
<td>open file for reading</td>
</tr>
<tr>
<td>&quot;w&quot;</td>
<td>create file for writing, or truncate to zero length</td>
</tr>
<tr>
<td>&quot;a&quot;</td>
<td>append: open file or create for writing at end-of-file</td>
</tr>
<tr>
<td>&quot;r+&quot;</td>
<td>open file for update (reading and/or writing)</td>
</tr>
<tr>
<td>&quot;w+&quot;</td>
<td>create file for update, or truncate to zero length</td>
</tr>
<tr>
<td>&quot;a+&quot;</td>
<td>append: open file or create for update, writing at end-of-file</td>
</tr>
</tbody>
</table>

In addition to the above characters, you can also include one of the following characters in `mode` to specify the translation mode for newline characters:

- `t` The letter "t" may be added to any of the above sequences in the second or later position to indicate that the file is (or must be) a text file. It also overrides the global translation mode flag if you link your program with `BINMODE.OBJ`. The global translation mode flag default is "text" unless you explicitly link your program with `BINMODE.OBJ`.

  When neither "t" nor "b" is specified, the value of the global variable `_fmode` establishes whether the file is to treated as a binary or a text file. Unless this value is changed by the program or you have linked your program with `BINMODE.OBJ`, the default will be text mode.

- `b` The letter "b" may be added to any of the above sequences in the second or later position to indicate that the file is (or must be) a binary file (an ANSI requirement for portability to systems that make a distinction between text and binary files).

You can also include one of the following characters to enable or disable the "commit" flag for the associated file.

- `c` The letter "c" may be added to any of the above sequences in the second or later position to indicate that any output is committed by the operating system whenever a flush (`fflush` or `flushall`) is done.

  This option is not supported under Netware.
The letter "n" may be added to any of the above sequences in the second or later position to indicate that the operating system need not commit any output whenever a flush is done. It also overrides the global commit flag if you link your program with COMMODE.OBJ. The global commit flag default is "no-commit" unless you explicitly link your program with COMMODE.OBJ.

This option is not supported under Netware.

The "t", "c", and "n" mode options are extensions for fopen and _fdopen and should not be used where ANSI portability is desired.

Opening a file with read mode ('r' as the first character in the mode argument) fails if the file does not exist or it cannot be read. Opening a file with append mode ('a' as the first character in the mode argument) causes all subsequent writes to the file to be forced to the current end-of-file, regardless of previous calls to the fseek function. When a file is opened with update mode ('+' as the second or later character of the mode argument), both input and output may be performed on the associated stream.

When a stream is opened in update mode, both reading and writing may be performed. However, writing may not be followed by reading without an intervening call to the fflush function or to a file positioning function (fseek, fsetpos, rewind). Similarly, reading may not be followed by writing without an intervening call to a file positioning function, unless the read resulted in end-of-file.

The _wfopen function is identical to fopen except that it accepts wide-character string arguments for filename and mode.

Returns: The fopen function returns a pointer to the object controlling the stream. This pointer must be passed as a parameter to subsequent functions for performing operations on the file. If the open operation fails, fopen returns NULL. When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also: _dos_open, fclose, fcloseall, fdopen, fopen_s, freopen, freopen_s, _fsopen, _grow_handles, _hdopen, open, _open_osfhandle, _popen, sopen

Example:
#include <stdio.h>

void main()
{
    FILE *fp;

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        /* rest of code goes here */
        fclose( fp );
    }
}

Classification: ANSI, ('t', 'c', 'n' are Watcom extensions)

Systems: fopen - All, Netware
         _wfopen - All
Synopsis:
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
errno_t fopen_s( FILE * restrict * restrict streamptr,
    const char * restrict filename,
    const char * restrict mode);
errno_t _wfopen_s( FILE * restrict * restrict streamptr,
    const wchar_t * restrict filename,
    const wchar_t * restrict mode);

Constraints:
If any of the following runtime-constraints is violated, the currently active runtime-constraint handler
will be invoked and fopen_s will return a non-zero value to indicate an error, or the
runtime-constraint handler aborts the program.

None of streamptr, filename, or mode shall be a null pointer. If there is a runtime-constraint violation,
fopen_s does not attempt to open a file. Furthermore, if streamptr is not a null pointer, fopen_s
sets *streamptr to the null pointer.

Description:
The fopen_s function opens the file whose name is the string pointed to by filename, and associates a
stream with it. The mode string shall be as described for fopen, with the addition that modes starting
with the character 'w' or 'a' may be preceded by the character 'u', see below:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;uw&quot;</td>
<td>truncate to zero length or create text file for writing, default permissions</td>
</tr>
<tr>
<td>&quot;ua&quot;</td>
<td>append; open or create text file for writing at end-of-file, default permissions</td>
</tr>
<tr>
<td>&quot;uwb&quot;</td>
<td>truncate to zero length or create binary file for writing, default permissions</td>
</tr>
<tr>
<td>&quot;uab&quot;</td>
<td>append; open or create binary file for writing at end-of-file, default permissions</td>
</tr>
<tr>
<td>&quot;uw+&quot;</td>
<td>truncate to zero length or create text file for update, default permissions</td>
</tr>
<tr>
<td>&quot;ua+&quot;</td>
<td>append; open or create text file for update, writing at end-of-file, default permissions</td>
</tr>
<tr>
<td>&quot;uw+b or uwb+&quot;</td>
<td>truncate to zero length or create binary file for update, default permissions</td>
</tr>
<tr>
<td>&quot;uab+&quot;</td>
<td>append; open or create binary file for update, writing at end-of-file, default permissions</td>
</tr>
</tbody>
</table>

To the extent that the underlying system supports the concepts, files opened for writing shall be opened
with exclusive (also known as non-shared) access. If the file is being created, and the first character of
the mode string is not 'u', to the extent that the underlying system supports it, the file shall have a file
permission that prevents other users on the system from accessing the file. If the file is being created
and first character of the mode string is 'u', then by the time the file has been closed, it shall have the
system default file access permissions. If the file was opened successfully, then the pointer to FILE
pointed to by streamptr will be set to the pointer to the object controlling the opened file. Otherwise,
the pointer to FILE pointed to by streamptr will be set to a null pointer.

In addition to the above characters, you can also include one of the following characters in mode to
specify the translation mode for newline characters:
The letter "t" may be added to any of the above sequences in the second or later position to indicate that the file is (or must be) a text file. It also overrides the global translation mode flag if you link your program with BINMODE.OBJ. The global translation mode flag default is "text" unless you explicitly link your program with BINMODE.OBJ.

When neither "t" nor "b" is specified, the value of the global variable _fmode establishes whether the file is to treated as a binary or a text file. Unless this value is changed by the program or you have linked your program with BINMODE.OBJ, the default will be text mode.

The letter "b" may be added to any of the above sequences in the second or later position to indicate that the file is (or must be) a binary file (an ANSI requirement for portability to systems that make a distinction between text and binary files).

You can also include one of the following characters to enable or disable the "commit" flag for the associated file.

The letter "c" may be added to any of the above sequences in the second or later position to indicate that any output is committed by the operating system whenever a flush (fflush or flushall) is done.

This option is not supported under Netware.

The letter "n" may be added to any of the above sequences in the second or later position to indicate that the operating system need not commit any output whenever a flush is done. It also overrides the global commit flag if you link your program with COMMODE.OBJ. The global commit flag default is "no-commit" unless you explicitly link your program with COMMODE.OBJ.

This option is not supported under Netware.

The "t", "c", and "n" mode options are extensions for fopen_s and should not be used where ANSI portability is desired.

Opening a file with read mode (r as the first character in the mode argument) fails if the file does not exist or it cannot be read. Opening a file with append mode (a as the first character in the mode argument) causes all subsequent writes to the file to be forced to the current end-of-file, regardless of previous calls to the fseek function. When a file is opened with update mode (+ as the second or later character of the mode argument), both input and output may be performed on the associated stream.

When a stream is opened in update mode, both reading and writing may be performed. However, writing may not be followed by reading without an intervening call to the fflush function or to a file positioning function (fseek, fsetpos, rewind). Similarly, reading may not be followed by writing without an intervening call to a file positioning function, unless the read resulted in end-of-file.

The _wfopen_s function is identical to fopen_s except that it accepts wide-character string arguments for filename and mode.

Returns: The fopen_s function returns zero if it opened the file. If it did not open the file or if there was a runtime-constraint violation, fopen_s returns a non-zero value.

See Also: _dos_open, fclose, fcloseall, fdopen, fopen, freopen, freopen_s, _fsopen, _grow_handles, _hdopen, open, _open_osfhandle, _popen, sopen
Example:
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>

void main()
{
    errno_t rc;
    FILE    *fp;

    rc = fopen_s( &fp, "file", "r" );
    if( fp != NULL ) {
        /* rest of code goes here */
        fclose( fp );
    }
}

Classification:
fopen_s is TR 24371
_wfopen_s is WATCOM

Systems:
fopen_s - All, Netware
_wfopen_s - All
Synopsis:  
#include <i86.h>
unsigned FP_OFF( void __far *far_ptr );

Description:  The FP_OFF macro can be used to obtain the offset portion of the far pointer value given in far_ptr.

Returns:  The macro returns an unsigned integer value which is the offset portion of the pointer value.

See Also:  FP_SEG, MK_FP, segread

Example:  
#include <stdio.h>
#include <i86.h>

char ColourTable[256][3];

void main()
{
    union REGPACK r;
    int i;

    /* read block of colour registers */
    r.h.ah = 0x10;
    r.h.al = 0x17;
#if defined(__386__)
    r.x.ebx = 0;
    r.x.ecx = 256;
    r.x.edx = FP_OFF( ColourTable );
    r.w.ds = r.w.fs = r.w.gs = FP_SEG( &r );
#else
    r.w.bx = 0;
    r.w.cx = 256;
    r.w.dx = FP_OFF( ColourTable );
#endif
    r.w.es = FP_SEG( ColourTable );
    intr( 0x10, &r );

    for( i = 0; i < 256; i++ ) {
        printf( "Colour index = %d "
            "{ Red=%d, Green=%d, Blue=%d }\n",
            i,
            ColourTable[i][0],
            ColourTable[i][1],
            ColourTable[i][2] );
    }
}

Classification:  Intel

Systems:  MACRO
Synopsis:    #include <i86.h>
    unsigned FP_SEG( void__far *far_ptr );

Description: The FP_SEG macro can be used to obtain the segment portion of the far pointer value given in far_ptr.

Returns:    The macro returns an unsigned integer value which is the segment portion of the pointer value.

See Also:   FP_OFF, MK_FP, segread

Example:    #include <stdio.h>
            #include <i86.h>

            char ColourTable[256][3];

            void main()
            {
                union REGPACK r;
                int i;

                /* read block of colour registers */
                r.h.ah = 0x10;
                r.h.al = 0x17;
                #if defined(__386__)
                    r.x.ebx = 0;
                    r.x.ecx = 256;
                    r.x.edx = FP_OFF( ColourTable );
                    r.w.ds = r.w.fs = r.w.gs = FP_SEG( &r );
                #else
                    r.w.bx = 0;
                    r.w.cx = 256;
                    r.w.dx = FP_OFF( ColourTable );
                #endif
                r.w.es = FP_SEG( ColourTable );
                intr( 0x10, &r );

                for( i = 0; i < 256; i++ ) {
                    printf( "Colour index = %d "
                            "( Red=%d, Green=%d, Blue=%d )\n",
                            i,
                            ColourTable[i][0],
                            ColourTable[i][1],
                            ColourTable[i][2] );
                }
            }

Classification: Intel

Systems:    MACRO
Synopsis:    
#include <math.h>
int fpclassify( x );

Description:   The fpclassify macro classifies its argument x as NaN, infinite, normal, subnormal, or zero. First, an argument represented in a format wider than its semantic type is converted to its semantic type. Then classification is based on the type of the argument.

The argument x must be an expression of real floating type.

The possible return values of fpclassify and their meanings are listed below.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP_INFINITE</td>
<td>positive or negative infinity</td>
</tr>
<tr>
<td>FP_NAN</td>
<td>NaN (not-a-number)</td>
</tr>
<tr>
<td>FP_NORMAL</td>
<td>normal number (neither zero, subnormal, NaN, nor infinity)</td>
</tr>
<tr>
<td>FP_SUBNORMAL</td>
<td>subnormal number</td>
</tr>
<tr>
<td>FP_ZERO</td>
<td>positive or negative zero</td>
</tr>
</tbody>
</table>

Returns:   The fpclassify macro returns the value of the number classification macro appropriate to the value of its argument x.

See Also:   isfinite, isinf, isnan, isnormal, signbit

Example:   #include <math.h>
#include <stdio.h>

    void main( void )
    {
        printf( "infinity %s a normal number\n",
            fpclassify( INFINITY ) == FP_NORMAL ?
            "is" : "is not" );
    }

produces the following:

    infinity is not a normal number

Classification:   ANSI

Systems:   MACRO
Synopsis:  

```
#include <float.h>
void _fpreset( void );
```

Description:  

The `_fpreset` function resets the floating-point unit to the default state that the math library requires for correct function. After a floating-point exception, it may be necessary to call the `_fpreset` function before any further floating-point operations are attempted.

In multi-threaded environments, `_fpreset` only affects the current thread.

Returns:  

No value is returned.

See Also:  

`_clear87`, `_control87`, `_controlfp`, `_finite`, `_status87`

Example:  

```
#include <stdio.h>
#include <float.h>

char *status[2] = { "No", " " };

void main( void )
{
    unsigned int fp_status;
    fp_status = _status87();

    printf( "80x87 status\n" );
    printf( "%s invalid operation\n", status[ (fp_status & SW_INVALID) == 0 ] );
    printf( "%s denormalized operand\n", status[ (fp_status & SW_DENORMAL) == 0 ] );
    printf( "%s divide by zero\n", status[ (fp_status & SW_ZERODIVIDE) == 0 ] );
    printf( "%s overflow\n", status[ (fp_status & SW_OVERFLOW) == 0 ] );
    printf( "%s underflow\n", status[ (fp_status & SW_UNDERFLOW) == 0 ] );
    printf( "%s inexact result\n", status[ (fp_status & SW_INEXACT) == 0 ] );
    _fpreset();
}
```

Classification:  Intel

Systems:  All, Netware
**Synopsis:**

```c
#include <stdio.h>
int fprintf( FILE *fp, const char *format, ... );
#include <stdio.h>
#include <wchar.h>
int fwprintf( FILE *fp, const wchar_t *format, ... );
```

**Safer C:**

The Safer C Library extension provides the `fprintf_s` function which is a safer alternative to `fprintf`. This newer `fprintf_s` function is recommended to be used instead of the traditional "unsafe" `fprintf` function.

**Description:**

The `fprintf` function writes output to the file pointed to by `fp` under control of the argument `format`. The `format` string is described under the description of the `printf` function.

The `fwprintf` function is identical to `fprintf` except that it accepts a wide-character string argument for `format`.

**Returns:**

The `fprintf` function returns the number of characters written, or a negative value if an output error occurred. The `fwprintf` function returns the number of wide characters written, or a negative value if an output error occurred. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

**See Also:** `_bprintf`, `cprintf`, `printf`, `sprintf`, `_vbprintf`, `vcprintf`, `vfprintf`, `vprintf`, `vscanf`  

**Example:**

```c
#include <stdio.h>

char *weekday = { "Saturday" };  
char *month = { "April" };  

void main( void )
{
    fprintf( stdout, "%s, %s %d, %d\n",  
        weekday, month, 18, 1987 );
}
```

produces the following:

```
Saturday, April 18, 1987
```

**Classification:**

- `fprintf` is ANSI
- `fwprintf` is ANSI

**Systems:**

- `fprintf` - All, Netware
- `fwprintf` - All
Synopsis:
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
int fprintf_s( FILE * restrict stream,
               const char * restrict format, ... );
#include <wchar.h>
int fwprintf_s( FILE * restrict stream,
               const wchar_t * restrict format, ... );

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and fprintf_s will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither stream nor format shall be a null pointer. The %n specifier (modified or not by flags, field width, or precision) shall not appear in the string pointed to by format. Any argument to fprintf_s corresponding to a %s specifier shall not be a null pointer.

If there is a runtime-constraint violation, the fprintf_s function does not attempt to produce further output, and it is unspecified to what extent fprintf_s produced output before discovering the runtime-constraint violation.

Description: The fprintf_s function is equivalent to the fprintf function except for the explicit runtime-constraints listed above.

The fwprintf_s function is identical to fprintf_s except that it accepts a wide-character string argument for format.

Returns: The fprintf_s function returns the number of characters written, or a negative value if an output error or runtime-constraint violation occurred.

The fwprintf_s function returns the number of wide characters written, or a negative value if an output error or runtime-constraint violation occurred.

See Also: _bprintf, cprintf, fprintf, printf, sprintf, _vbprintf, vcprintf, vfprintf, vprintf, vsprintf

Example: #define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>

char *weekday = { "Friday" };
char *month = { "August" };

void main( void )
{
    fprintf_s( stdout, "%s, %s %d, %d
",
               weekday, month, 13, 2004 );
}

produces the following:

Friday, August 13, 2004

Classification: fprintf_s is TR 24731
fwprintf_s is TR 24731
fprintf_s, fwprintf_s

Systems:  
fprintf_s - All, Netware  
fwprintf_s - All
Synopsis: 

```c
#include <stdio.h>
int fputc( int c, FILE *fp );
#include <stdio.h>
#include <wchar.h>
wint_t fputwc( wint_t c, FILE *fp );
```

Description: The `fputc` function writes the character specified by the argument `c` to the output stream designated by `fp`

The `fputwc` function is identical to `fputc` except that it converts the wide character specified by `c` to a multibyte character and writes it to the output stream.

Returns: The `fputc` function returns the character written or, if a write error occurs, the error indicator is set and `fputc` returns `EOF`.

The `fputwc` function returns the wide character written or, if a write error occurs, the error indicator is set and `fputwc` returns `WEOF`. If an encoding error occurs, `errno` is set to `EILSEQ` and `fputwc` returns `WEOF`.

When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fopen`, `fputchar`, `fputs`, `putc`, `putchar`, `puts`, `ferror`

Example: 

```c
#include <stdio.h>

void main()
{
    FILE *fp;
    int c;

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        while( (c = fgetc( fp )) != EOF )
            fputc( c, stdout );
        fclose( fp );
    }
}
```

Classification: `fputc` is ANSI
`fputwc` is ANSI

Systems: `fputc` - All, Netware
`fputwc` - All
Synopsis:  
#include <stdio.h>
int fputchar( int c );
int _fputchar( int c );
wint_t _fputwchar( wint_t c );

Description: The fputchar function writes the character specified by the argument c to the output stream stdout. This function is identical to the putchar function.

The function is equivalent to:

    fputc( c, stdout );

The _fputchar function is identical to fputchar. Use _fputchar for ANSI naming conventions.

The _fputwchar function is identical to fputchar except that it converts the wide character specified by c to a multibyte character and writes it to the output stream.

Returns: The fputchar function returns the character written or, if a write error occurs, the error indicator is set and fputchar returns EOF.

The _fputwchar function returns the wide character written or, if a write error occurs, the error indicator is set and _fputwchar returns WEOF.

When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also: fopen, fputc, fgets, putc, putchar, puts, ferror

Example:  
#include <stdio.h>

void main()
{
    FILE *fp;
    int c;

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        c = fgetc( fp );
        while( c != EOF ) {
            _fputchar( c );
            c = fgetc( fp );
        }
        fclose( fp );
    }
}

Classification: WATCOM

Systems:  
_fputchar - All, Netware
.fpputchar - All, Netware
.fpputwchar - All
Synopsis:

```c
#include <stdio.h>
int fputs( const char *buf, FILE *fp );
#include <stdio.h>
#include <wchar.h>
int fputws( const wchar_t *buf, FILE *fp );
```

Description:
The `fputs` function writes the character string pointed to by `buf` to the output stream designated by `fp`. The terminating null character is not written.

The `fputws` function is identical to `fputs` except that it converts the wide character string specified by `buf` to a multibyte character string and writes it to the output stream.

Returns:
The `fputs` function returns `EOF` if an error occurs; otherwise, it returns a non-negative value (the amount written including the new-line character). The `fputws` function returns `WEOF` if a write or encoding error occurs; otherwise, it returns a non-negative value (the amount written including the new-line character). When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fopen`, `fputc`, `fputchar`, `putc`, `putchar`, `puts`, `ferror`

Example:

```c
#include <stdio.h>

void main()
{
    FILE *fp;
    char buffer[80];

    fp = fopen( "file", "r" );
    if ( fp != NULL ) {
        while( fgets( buffer, 80, fp ) != NULL )
            fputs( buffer, stdout );
        fclose( fp );
    }
}
```

Classification: `fputs` is ANSI
`fputws` is ANSI

Systems: `fputs` - All, Netware
`fputws` - All
Synopsis:

```c
#include <stdio.h>
size_t fread( void *buf,
        size_t elsize,
        size_t nelem,
        FILE *fp );
```

Description: The `fread` function reads `nelem` elements of `elsize` bytes each from the file specified by `fp` into the buffer specified by `buf`.

Returns: The `fread` function returns the number of complete elements successfully read. This value may be less than the requested number of elements.

The `feof` and `ferror` functions can be used to determine whether the end of the file was encountered or if an input/output error has occurred. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fopen`, `feof`, `ferror`

Example: The following example reads a simple student record containing binary data. The student record is described by the `struct student_data` declaration.

```c
#include <stdio.h>

struct student_data {
    int student_id;
    unsigned char marks[10];
};

size_t read_data( FILE *fp, struct student_data *p )
{
    return( fread( p, sizeof(*p), 1, fp ) );
}

void main()
{
    FILE *fp;
    struct student_data std;
    int i;

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        while( read_data( fp, &std ) != 0 ) {
            printf( "id=%d ", std.student_id );
            for( i = 0; i < 10; i++ )
                printf( "%3d ", std.marks[ i ] );
            printf( "\n" );
        }
    fclose( fp );
    }
}
```

Classification: ANSI

Systems: All, Netware

284 Library Functions and Macros
Synopsis:

```c
#include <stdlib.h>  For ANSI compatibility (free only)
#include <malloc.h>  Required for other function prototypes
void free( void *ptr );
void _bfree( __segment seg, void __based(void) *ptr );
void _ffree( void __far *ptr );
void _nfree( void __near *ptr );
```

Description:

When the value of the argument `ptr` is NULL, the `free` function does nothing otherwise, the `free` function deallocates the memory block located by the argument `ptr` which points to a memory block previously allocated through a call to the appropriate version of `calloc`, `malloc` or `realloc`. After the call, the freed block is available for allocation.

Each function deallocates memory from a particular heap, as listed below:

<table>
<thead>
<tr>
<th>Function</th>
<th>Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>free</code></td>
<td>Depends on data model of the program</td>
</tr>
<tr>
<td><code>_bfree</code></td>
<td>Based heap specified by <code>seg</code> value</td>
</tr>
<tr>
<td><code>_ffree</code></td>
<td>Far heap (outside the default data segment)</td>
</tr>
<tr>
<td><code>_nfree</code></td>
<td>Near heap (inside the default data segment)</td>
</tr>
</tbody>
</table>

In a large data memory model, the `free` function is equivalent to the `_ffree` function; in a small data memory model, the `free` function is equivalent to the `_nfree` function.

Returns:

The `free` functions return no value.

See Also: `calloc` Functions, `_expand` Functions, `halloc`, `hfree`, `malloc` Functions, `_msize` Functions, `realloc` Functions, `sbrk` Functions.

Example:

```c
#include <stdio.h>
#include <stdlib.h>

void main()
{
    char *buffer;

    buffer = (char *)malloc( 80 );
    if( buffer == NULL ) {
        printf( "Unable to allocate memory\n" );
    } else {

        /* rest of code goes here */

        free( buffer );  /* deallocate buffer */
    }
}
```

Classification: `free` is ANSI
- `_ffree` is not ANSI
- `_bfree` is not ANSI
- `_nfree` is not ANSI
free Functions

Systems:

free - All, Netware
__bfree - DOS/16, Windows, QNX/16, OS/2 1.x(all)
__ffree - DOS/16, Windows, QNX/16, OS/2 1.x(all)
__nfree - DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2 1.x(MT), OS/2-32
Synopsis:  
#include <malloc.h>
unsigned int _freect( size_t size );

Description:  The _freect function returns the number of times that _nmalloc (or malloc in small data models) can be called to allocate an item of size bytes. In the tiny, small and medium memory models, the default data segment is only extended as needed to satisfy requests for memory allocation. Therefore, you will need to call _nheapgrow in these memory models before calling _freect in order to get a meaningful result.

Returns:  The _freect function returns the number of calls as an unsigned integer.

See Also:  calloc, _heapgrow Functions, malloc Functions, _memavl, _memmax

Example:  
#include <stdio.h>
#include <malloc.h>

void main()
{
    int  i;

    printf( "Can allocate %u longs before _nheapgrow\n", _freect( sizeof(long) ) );
    _nheapgrow();
    printf( "Can allocate %u longs after _nheapgrow\n", _freect( sizeof(long) ) );
    for( i = 1; i < 1000; i++ ) {
        _nmalloc( sizeof(long) );
    }
    printf( "After allocating 1000 longs:\n" );
    printf( "Can still allocate %u longs\n", _freect( sizeof(long) ) );
}

produces the following:

Can allocate 0 longs before _nheapgrow
Can allocate 10447 longs after _nheapgrow
After allocating 1000 longs:
Can still allocate 9447 longs

Classification: WATCOM

Systems:  All
#include <stdio.h>

FILE *freopen( const char *filename, 
    const char *mode, 
    FILE *fp );

FILE * _wfreopen( const wchar_t *filename, 
    const wchar_t *mode, 
    FILE *fp );

Safer C: The Safer C Library extension provides the freopen_s function which is a safer alternative to freopen. This newer freopen_s function is recommended to be used instead of the traditional "unsafe" freopen function.

Description: The stream located by the fp pointer is closed. The freopen function opens the file whose name is the string pointed to by filename, and associates a stream with it. The stream information is placed in the structure located by the fp pointer.

The argument mode is described in the description of the fopen function.

The _wfreopen function is identical to freopen except that it accepts wide-character string arguments for filename and mode.

Returns: The freopen function returns a pointer to the object controlling the stream. This pointer must be passed as a parameter to subsequent functions for performing operations on the file. If the open operation fails, freopen returns NULL. When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also: _dos_open, fclose, fcloseall, fdopen, fopen, fopen_s, freopen_s, _fsopen, _grow_handles, _hdopen, open, _open_osfhandle, _popen, sopen

Example: #include <stdio.h>

void main()
{
    FILE *fp;
    int c;

    fp = freopen( "file", "r", stdin );
    if( fp != NULL ) 
        
    while( (c = fgetchar()) != EOF )
        
    fclose( fp );
}

Classification: freopen is ANSI
    _wfreopen is not ANSI

Systems: freopen - All, Netware
    _wfreopen - All
Synopsis:
#include <stdio.h>
#define __STDC_WANT_LIB_EXT1__ 1
FILE *freopen( const char *filename,
               const char *mode,
               FILE *fp );
FILE *__wfreopen( const wchar_t *filename,
                 const wchar_t *mode,
                 FILE *fp );

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler
will be invoked and freopen_s will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

None of newstreamptr, mode, and stream shall be a null pointer. If there is a runtime-constraint violation, freopen_s neither attempts to close any file associated with stream nor attempts to open a file. Furthermore, if newstreamptr is not a null pointer, freopen_s sets *newstreamptr to the null pointer.

Description: The freopen_s function opens the file whose name is the string pointed to by filename and associates the stream pointed to by stream with it. The mode argument has the same meaning as in the fopen_s function (including the mode’s effect on exclusive access and file permissions). If filename is a null pointer, the freopen_s function attempts to change the mode of the stream to that specified by mode, as if the name of the file currently associated with the stream had been used. It is implementation-defined which changes of mode are permitted (if any), and under what circumstances. The freopen_s function first attempts to close any file that is associated with stream. Failure to close the file is ignored. The error and end-of-file indicators for the stream are cleared. If the file was opened successfully, then the pointer to FILE pointed to by newstreamptr will be set to the value of stream. Otherwise, the pointer to FILE pointed to by newstreamptr will be set to a null pointer.

The _wfreopen_s function is identical to freopen_s except that it accepts wide-character string arguments for filename and mode.

Returns: The freopen_s function returns zero if it opened the file. If it did not open the file or there was a runtime-constraint violation, freopen_s returns a non-zero value.

See Also: _dos_open, fclose, fcloseall, fdopen, fopen, fopen_s, freopen, _fsopen, _grow_handles, _hdopen, open, __open_osfhandle, _popen, sopen

Example:
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>

void main()
{
    errno_t rc;
    FILE    *fp;
    int     c;

    rc = freopen_s( &fp, "file", "r", stdin );
    if( rc == 0 ) {
        while( (c = fgetc( fp )) != EOF )
            putchar(c);
        fclose( fp );
    }
}


Classification: \texttt{freopen\_s} is TR 24371
\texttt{\_wfreopen\_s} is WATCOM

Systems: \texttt{freopen\_s} - All, Netware
\texttt{\_wfreopen\_s} - All
Synopsis:  
#include <math.h>
double frexp( double value, int *exp );

Description:  The frexp function breaks a floating-point number into a normalized fraction and an integral power of 2. It stores the integral power of 2 in the int object pointed to by exp.

Returns:  The frexp function returns the value of x, such that x is a double with magnitude in the interval [0.5,1) or zero, and value equals x times 2 raised to the power *exp. If value is zero, then both parts of the result are zero.

See Also:  ldexp, modf

Example:  
#include <stdio.h>
#include <math.h>

void main()
{
    int    expon;
    double value;

    value = frexp(  4.25, &expon );
    printf( "%f %d\n", value, expon );
    value = frexp( -4.25, &expon );
    printf( "%f %d\n", value, expon );
}

produces the following:

0.531250 3
-0.531250 3

Classification:  ANSI

Systems:  Math
fscanf, fwscanf

Synopsis:

```c
#include <stdio.h>
int fscanf( FILE *fp, const char *format, ... );
#include <stdio.h>
#include <wchar.h>
int fwscanf( FILE *fp, const wchar_t *format, ... );
```

Safer C: The Safer C Library extension provides the `fscanf_s` function which is a safer alternative to `fscanf`. This newer `fscanf_s` function is recommended to be used instead of the traditional "unsafe" `fscanf` function.

Description: The `fscanf` function scans input from the file designated by `fp` under control of the argument `format`. Following the format string is a list of addresses to receive values. The `format` string is described under the description of the `scanf` function.

The `fwscanf` function is identical to `fscanf` except that it accepts a wide-character string argument for `format`.

Returns: The `fscanf` function returns EOF if an input failure occurred before any conversion. Otherwise, the number of input arguments for which values were successfully scanned and stored is returned. When a file input error occurs, the `errno` global variable may be set.

See Also: `cscanf`, `scanf`, `sscanf`, `vcscanf`, `vfscanf`, `vscanf`, `vsscanf`

Example: To scan a date in the form "Saturday April 18 1987":

```c
#include <stdio.h>

void main( void )
{
    int day, year;
    char weekday[10], month[10];
    FILE *in_data;

    in_data = fopen( "file", "r" );
    if( in_data != NULL ) {
        fscanf( in_data, "%s %s %d %d", weekday, month, &day, &year );
        printf( "Weekday=%s Month=%s Day=%d Year=%d\n", weekday, month, day, year );
    }
    fclose( in_data );
}
```

Classification: `fscanf` is ISO C90
`fwscanf` is ISO C95

Systems: `fscanf` - All, Netware
`fwscanf` - All
Synopsis:

```c
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
int fscanf_s( FILE * restrict stream,
const char * restrict format, ... );
#include <stdio.h>
#include <wchar.h>
int fwscanf_s( FILE * restrict stream,
const wchar_t * restrict format, ... );
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `fscanf_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither `stream` nor `format` shall be a null pointer. Any argument indirected through in order to store converted input shall not be a null pointer.

If there is a runtime-constraint violation, the `fscanf_s` function does not attempt to perform further input, and it is unspecified to what extent `fscanf_s` performed input before discovering the runtime-constraint violation.

Description: The `fscanf_s` function is equivalent to `fscanf` except that the `c`, `s`, and `[` conversion specifiers apply to a pair of arguments (unless assignment suppression is indicated by a `*`). The first of these arguments is the same as for `fscanf`. That argument is immediately followed in the argument list by the second argument, which has type `size_t` and gives the number of elements in the array pointed to by the first argument of the pair. If the first argument points to a scalar object, it is considered to be an array of one element.

A matching failure occurs if the number of elements in a receiving object is insufficient to hold the converted input (including any trailing null character).

The `fwscanf_s` function is identical to `fscanf_s` except that it accepts a wide-character string argument for `format`.

Returns: The `fscanf_s` function returns `EOF` if an input failure occurred before any conversion or if there was a runtime-constraint violation. Otherwise, the `fscanf_s` function returns the number of input items successfully assigned, which can be fewer than provided for, or even zero.

When a file input error occurs, the `errno` global variable may be set.

See Also: `cscanf`, `fscanf`, `scanf`, `sscanf`, `vcscanf`, `vfscanf`, `vscanf`, `vsscanf`

Example: To scan a date in the form "Friday August 13 2004":

```c
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>

void main( void )
{
    int day, year;
    char weekday[10], month[10];
    FILE *in_data;
```
```c
in_data = fopen( "file", "r" );
if( in_data != NULL ) {
    fscanf_s( in_data, "%s %s %d %d",
        weekday, sizeof( weekday ),
        month, sizeof( month ),
        &day, &year );
    printf_s( "Weekday=%s Month=%s Day=%d Year=%d\n",
        weekday, month, day, year );
    fclose( in_data );
}
```

**Classification:** fscanf_s is TR 24731
    fwscanf_s is TR 24731

**Systems:**
    fscanf_s - All, Netware
    fwscanf_s - All
Synopsis:

```c
#include <stdio.h>
int fseek( FILE *fp, long int offset, int where );
```

Description: The `fseek` function changes the read/write position of the file specified by `fp`. This position defines the character that will be read or written on the next I/O operation on the file. The argument `fp` is a file pointer returned by `fopen` or `freopen`. The argument `offset` is the position to seek to relative to one of three positions specified by the argument `where`. Allowable values for `where` are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEEK_SET</td>
<td>The new file position is computed relative to the start of the file. The value of <code>offset</code> must not be negative.</td>
</tr>
<tr>
<td>SEEK_CUR</td>
<td>The new file position is computed relative to the current file position. The value of <code>offset</code> may be positive, negative or zero.</td>
</tr>
<tr>
<td>SEEK_END</td>
<td>The new file position is computed relative to the end of the file.</td>
</tr>
</tbody>
</table>

The `fseek` function clears the end-of-file indicator and undoes any effects of the `ungetc` function on the same file.

The `ftell` function can be used to obtain the current position in the file before changing it. The position can be restored by using the value returned by `ftell` in a subsequent call to `fseek` with the `where` parameter set to `SEEK_SET`.

Returns: The `fseek` function returns zero if successful, non-zero otherwise. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fgetpos`, `fopen`, `fsetpos`, `ftell`

Example: The size of a file can be determined by the following example which saves and restores the current position of the file.

```c
#include <stdio.h>

long int filesize( FILE *fp )
{
    long int save_pos, size_of_file;

    save_pos = ftell( fp );
    fseek( fp, 0L, SEEK_END );
    size_of_file = ftell( fp );
    fseek( fp, save_pos, SEEK_SET );
    return( size_of_file );
}
```
## fseek

```c
void main()
{
    FILE *fp;

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        printf( "File size=%ld\n", filesize( fp ) );
        fclose( fp );
    }
}
```

**Classification:** ANSI

**Systems:** All, Netware
Synopsis:  
#include <stdio.h>
int fsetpos( FILE *fp, fpos_t *pos );

Description:  The fsetpos function positions the file fp according to the value of the object pointed to by pos, which shall be a value returned by an earlier call to the fseekpos function on the same file.

Returns:  The fsetpos function returns zero if successful, otherwise, the fsetpos function returns a non-zero value. When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also:  fgetpos, fopen, fseek, ftell

Example:  #include <stdio.h>

void main()
{
    FILE *fp;
    fpos_t position;
    auto char buffer[80];

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        fgetpos( fp, &position ); /* get position */
        fgets( buffer, 80, fp ); /* read record */
        fsetpos( fp, &position ); /* set position */
        fgets( buffer, 80, fp ); /* read same record */
        fclose( fp );
    }
}

Classification: ANSI

Systems:  All, Netware
Synopsis:  
#include <stdio.h>
FILE *__fsopen( const char *filename,
               const char *mode, int share );
FILE *__wfsopen( const wchar_t *filename,
                const wchar_t *mode, int share );

Description:  The _fsopen function opens the file whose name is the string pointed to by filename, and associates a stream with it. The arguments mode and share control shared reading or writing. The argument mode points to a string beginning with one of the following sequences:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;r&quot;</td>
<td>open file for reading; use default file translation</td>
</tr>
<tr>
<td>&quot;w&quot;</td>
<td>create file for writing, or truncate to zero length; use default file translation</td>
</tr>
<tr>
<td>&quot;a&quot;</td>
<td>append: open text file or create for writing at end-of-file; use default file translation</td>
</tr>
<tr>
<td>&quot;rb&quot;</td>
<td>open binary file for reading</td>
</tr>
<tr>
<td>&quot;rt&quot;</td>
<td>open text file for reading</td>
</tr>
<tr>
<td>&quot;wb&quot;</td>
<td>create binary file for writing, or truncate to zero length</td>
</tr>
<tr>
<td>&quot;wt&quot;</td>
<td>create text file for writing, or truncate to zero length</td>
</tr>
<tr>
<td>&quot;ab&quot;</td>
<td>append; open binary file or create for writing at end-of-file</td>
</tr>
<tr>
<td>&quot;at&quot;</td>
<td>append; open text file or create for writing at end-of-file</td>
</tr>
<tr>
<td>&quot;r+&quot;</td>
<td>open file for update (reading and/or writing); use default file translation</td>
</tr>
<tr>
<td>&quot;w+&quot;</td>
<td>create file for update, or truncate to zero length; use default file translation</td>
</tr>
<tr>
<td>&quot;a+&quot;</td>
<td>append; open file or create for update, writing at end-of-file; use default file translation</td>
</tr>
<tr>
<td>&quot;r+b&quot;, &quot;rb+&quot;</td>
<td>open binary file for update (reading and/or writing)</td>
</tr>
<tr>
<td>&quot;r+t&quot;, &quot;rt+&quot;</td>
<td>open text file for update (reading and/or writing)</td>
</tr>
<tr>
<td>&quot;w+b&quot;, &quot;wb+&quot;</td>
<td>create binary file for update, or truncate to zero length</td>
</tr>
<tr>
<td>&quot;w+t&quot;, &quot;wt+&quot;</td>
<td>create text file for update, or truncate to zero length</td>
</tr>
<tr>
<td>&quot;a+b&quot;, &quot;ab+&quot;</td>
<td>append; open binary file or create for update, writing at end-of-file</td>
</tr>
</tbody>
</table>

When default file translation is specified, the value of the global variable _fmode establishes whether the file is to be treated as a binary or a text file. Unless this value is changed by the program, the default will be text mode.
Opening a file with read mode (‘r’ as the first character in the mode argument) fails if the file does not exist or it cannot be read. Opening a file with append mode (‘a’ as the first character in the mode argument) causes all subsequent writes to the file to be forced to the current end-of-file, regardless of previous calls to the fseek function. When a file is opened with update mode (‘+’ as the second or third character of the mode argument), both input and output may be performed on the associated stream.

When a stream is opened in update mode, both reading and writing may be performed. However, writing may not be followed by reading without an intervening call to the fflush function or to a file positioning function (fseek, fsetpos, rewind). Similarly, reading may not be followed by writing without an intervening call to a file positioning function, unless the read resulted in end-of-file.

The shared access for the file, share, is established by a combination of bits defined in the <share.h> header file. The following values may be set:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SH_COMPAT</td>
<td>Set compatibility mode.</td>
</tr>
<tr>
<td>SH_DENYRW</td>
<td>Prevent read or write access to the file.</td>
</tr>
<tr>
<td>SH_DENYWR</td>
<td>Prevent write access of the file.</td>
</tr>
<tr>
<td>SH_DENYRD</td>
<td>Prevent read access to the file.</td>
</tr>
<tr>
<td>SH_DENYNO</td>
<td>Permit both read and write access to the file.</td>
</tr>
</tbody>
</table>

You should consult the technical documentation for the DOS system that you are using for more detailed information about these sharing modes.

The _wfsopen function is identical to _fsopen except that it accepts wide-character string arguments for filename and mode.

Returns: The _fsopen function returns a pointer to the object controlling the stream. This pointer must be passed as a parameter to subsequent functions for performing operations on the file. If the open operation fails, _fsopen returns NULL. When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also: _dos_open, fclose, fcloseall, fdopen, fopen, freopen, _grow_handles, _hdopen, open, _open_osfhandle, _popen, sopen

Example:
```c
#include <stdio.h>
#include <share.h>

void main()
{
    FILE *fp;

    /*
        open a file and prevent others from writing to it
    */
    fp = _fsopen( "report.dat", "w", SH_DENYWR );
    if( fp != NULL ) {
        /* rest of code goes here */
        fclose( fp );
    }
}
```
_fsopen, _wfsopen

Classification: WATCOM

Systems: _fsopen - All, Netware
         _wfsopen - All
Synopsis:

```
#include <sys/types.h>
#include <sys/stat.h>
int fstat(int handle, struct stat *buf);
int _fstat(int handle, struct stat *buf);
int _fstati64(int handle, struct _stati64 *buf);
int _wfstat(int handle, struct _stat *buf);
int _wfstati64(int handle, struct _stati64 *buf);
```

Description:
The `fstat` functions obtain information about an open file whose file handle is `handle`. This information is placed in the structure located at the address indicated by `buf`.

The file `<sys/stat.h>` contains definitions for the structure `stat`.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type/ Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>st_dev</code></td>
<td>(dev_t) the disk drive the file resides on</td>
</tr>
<tr>
<td><code>st_ino</code></td>
<td>(ino_t) this inode’s number (not used for DOS)</td>
</tr>
<tr>
<td><code>st_mode</code></td>
<td>(unsigned short) file mode</td>
</tr>
<tr>
<td><code>st_nlink</code></td>
<td>(short) number of hard links</td>
</tr>
<tr>
<td><code>st_uid</code></td>
<td>(unsigned long) user-id (always 'root' for DOS)</td>
</tr>
<tr>
<td><code>st_gid</code></td>
<td>(short) group-id (always 'root' for DOS)</td>
</tr>
<tr>
<td><code>st_rdev</code></td>
<td>(dev_t) this should be the device type but it is the same as <code>st_dev</code> for the time being</td>
</tr>
<tr>
<td><code>st_size</code></td>
<td>(off_t) total file size</td>
</tr>
<tr>
<td><code>st_atime</code></td>
<td>(time_t) this should be the file &quot;last accessed&quot; time if the file system supports it</td>
</tr>
<tr>
<td><code>st_mtime</code></td>
<td>(time_t) the file &quot;last modified&quot; time</td>
</tr>
<tr>
<td><code>st_ctime</code></td>
<td>(time_t) this should be the file &quot;last status change&quot; time if the file system supports it</td>
</tr>
</tbody>
</table>

The following fields are Netware only:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type/ Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>st_btime</code></td>
<td>(time_t) the file &quot;last archived&quot; time</td>
</tr>
<tr>
<td><code>st_attr</code></td>
<td>(unsigned long) the file’s attributes</td>
</tr>
<tr>
<td><code>st_archivedID</code></td>
<td>(unsigned long) the user/object ID that last archived file</td>
</tr>
<tr>
<td><code>st_updatedID</code></td>
<td>(unsigned long) the user/object ID that last updated file</td>
</tr>
<tr>
<td><code>st_inheritedRightsMask</code></td>
<td>(unsigned short) the inherited rights mask</td>
</tr>
<tr>
<td><code>st_originatingNameSpace</code></td>
<td>(unsigned char) the originating name space</td>
</tr>
</tbody>
</table>

The structure `_stati64` differs from `stat` in the following way:
The value \( m \) supplied to the macros is the value of the \( \text{st}\_\text{mode} \) field of a \text{stat} structure. The macro evaluates to a non-zero value if the test is true and zero if the test is false.

The following bits are encoded within the \( \text{st}\_\text{mode} \) field of a \text{stat} structure.

### Mask Owner Permissions

<table>
<thead>
<tr>
<th>Mask</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>_S_IRWXU</td>
<td>Read, write, search (if a directory), or execute (otherwise)</td>
</tr>
<tr>
<td>_S_IRUSR</td>
<td>Read permission bit</td>
</tr>
<tr>
<td>_S_IWUSR</td>
<td>Write permission bit</td>
</tr>
<tr>
<td>_S_IXUSR</td>
<td>Search/execute permission bit</td>
</tr>
<tr>
<td>_S_IREAD</td>
<td>( \equiv _S_IRUSR ) (for Microsoft compatibility)</td>
</tr>
<tr>
<td>_S_IWRITE</td>
<td>( \equiv _S_IWUSR ) (for Microsoft compatibility)</td>
</tr>
<tr>
<td>_S_IEXEC</td>
<td>( \equiv _S_IXUSR ) (for Microsoft compatibility)</td>
</tr>
</tbody>
</table>

\( \_S\_IRWXU \) is the bitwise inclusive OR of \( \_S\_IRUSR, \_S\_IWUSR, \) and \( \_S\_IXUSR \).

### Mask Group Permissions (same as owner's on DOS, OS/2 or Windows)

<table>
<thead>
<tr>
<th>Mask</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>_S_IRWXG</td>
<td>Read, write, search (if a directory), or execute (otherwise)</td>
</tr>
<tr>
<td>_S_IRGRP</td>
<td>Read permission bit</td>
</tr>
<tr>
<td>_S_IWGRP</td>
<td>Write permission bit</td>
</tr>
<tr>
<td>_S_IXGRP</td>
<td>Search/execute permission bit</td>
</tr>
</tbody>
</table>

\( \_S\_IRWXG \) is the bitwise inclusive OR of \( \_S\_IRGRP, \_S\_IWGRP, \) and \( \_S\_IXGRP \).

### Mask Other Permissions (same as owner’s on DOS, OS/2 or Windows)

<table>
<thead>
<tr>
<th>Mask</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>_S_IRWXO</td>
<td>Read, write, search (if a directory), or execute (otherwise)</td>
</tr>
<tr>
<td>_S_IROTH</td>
<td>Read permission bit</td>
</tr>
<tr>
<td>_S_IWOTH</td>
<td>Write permission bit</td>
</tr>
<tr>
<td>_S_IXOTH</td>
<td>Search/execute permission bit</td>
</tr>
</tbody>
</table>

\( \_S\_IRWXO \) is the bitwise inclusive OR of \( \_S\_IROTH, \_S\_IWOTH, \) and \( \_S\_IXOTH \).
### Mask  Meaning

<table>
<thead>
<tr>
<th>Mask</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_ISUID</td>
<td>(Not supported by DOS, OS/2 or Windows) Set user ID on execution. The</td>
</tr>
<tr>
<td></td>
<td>process’s effective user ID shall be set to that of the owner of the</td>
</tr>
<tr>
<td></td>
<td>file when the file is run as a program. On a regular file, this bit</td>
</tr>
<tr>
<td></td>
<td>should be cleared on any write.</td>
</tr>
<tr>
<td>S_ISGID</td>
<td>(Not supported by DOS, OS/2 or Windows) Set group ID on execution. Set</td>
</tr>
<tr>
<td></td>
<td>effective group ID on the process to the file’s group when the file</td>
</tr>
<tr>
<td></td>
<td>is run as a program. On a regular file, this bit should be cleared on</td>
</tr>
<tr>
<td></td>
<td>any write.</td>
</tr>
</tbody>
</table>

The `_fstat` function is identical to `fstat`. Use `_fstat` for ANSI/ISO naming conventions. The `_fstat64`, `_wfstat`, and `_wfstat64` functions differ from `fstat` in the type of structure that they are asked to fill in. The `_wfstat` and `_wfstat64` functions deal with wide character strings. The differences in the structures are described above.

### Returns:
All forms of the `fstat` function return zero when the information is successfully obtained. Otherwise, -1 is returned.

### Errors:
When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

### Constant  Meaning

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBADF</td>
<td>The <code>handle</code> argument is not a valid file handle.</td>
</tr>
</tbody>
</table>

### See Also:
creat, dup, dup2, open, sopen, stat

### Example:
```c
#include <stdio.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <fcntl.h>
#include <io.h>

void main()
{
    int handle, rc;
    struct stat buf;

    handle = open( "file", O_RDONLY );
    if( handle != -1 ) {
        rc = fstat( handle, &buf );
        if( rc != -1 )
            printf( "File size = %d\n", buf.st_size );
        close( handle );
    }
}
```

### Classification:
- `fstat` is POSIX
- `_fstat` is not POSIX
- `_fstat64` is not POSIX
- `_wfstat` is not POSIX
- `_wfstat64` is not POSIX
- `fstat` conforms to ANSI/ISO naming conventions

### Systems:
- `fstat` - All, Netware
- `_fstat` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
- `_fstat64` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_wfstat - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_wfstat64 - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

```c
#include <io.h>
int fsync( int fd );
```

Description:
The `fsync` function writes to disk all the currently queued data for the open file specified by `fd`. All necessary file system information required to retrieve the data is also written to disk. The file access times are also updated.

The `fsync` function is used when you wish to ensure that both the file data and file system information required to recover the complete file have been written to the disk.

The `fsync` function does not return until the transfer is completed.

Returns:
The `fsync` function returns zero if successful. Otherwise, it returns -1 and `errno` is set to indicate the error. If the `fsync` function fails, outstanding i/o operations are not guaranteed to have been completed.

Errors:
When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>EBADF</code></td>
<td>The <code>fd</code> argument is not a valid file handle.</td>
</tr>
<tr>
<td><code>EINVAL</code></td>
<td>Synchronized i/o is not supported for this file.</td>
</tr>
<tr>
<td><code>EIO</code></td>
<td>A physical I/O error occurred (e.g., a bad block). The precise meaning is device dependent.</td>
</tr>
<tr>
<td><code>ENOSYS</code></td>
<td>The <code>fsync</code> function is not supported.</td>
</tr>
</tbody>
</table>

See Also:
`fstat`, `open`, `stat`, `write`

Example:

```c
/* Write a file and make sure it is on disk. */
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <io.h>

char buf[512];

void main()
{
    int handle;
    int i;

    handle = creat( "file", S_IWRITE | S_IREAD );
    if( handle == -1 ) {
        perror( "Error creating file" );
        exit( EXIT_FAILURE );
    }
```

Library Functions and Macros  305
for( i = 0; i < 255; ++i ) {
    memset( buf, i, sizeof( buf ) );
    if( write( handle, buf, sizeof(buf) ) != sizeof(buf) ) {
        perror( "Error writing file" );
        exit( EXIT_FAILURE );
    }
}

if( fsync( handle ) == -1 ) {
    perror( "Error sync’ing file" );
    exit( EXIT_FAILURE );
}

close( handle );
exit( EXIT_SUCCESS );

Classification: POSIX 1003.4

Systems: All, Netware
Synopsis:  
#include <stdio.h>  
long int ftell( FILE *fp );

Description:  
The ftell function returns the current read/write position of the file specified by fp. This position defines the character that will be read or written by the next I/O operation on the file. The value returned by ftell can be used in a subsequent call to fseek to set the file to the same position.

Returns:  
The ftell function returns the current read/write position of the file specified by fp. When an error is detected, -1L is returned. When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also:  
fgetpos, fopen, fsetpos, fseek

Example:  
#include <stdio.h>  

long int filesize( FILE *fp )  
{  
  long int save_pos, size_of_file;
  
  save_pos = ftell( fp );
  fseek( fp, 0L, SEEK_END );
  size_of_file = ftell( fp );
  fseek( fp, save_pos, SEEK_SET );
  return( size_of_file );
}

void main()  
{  
  FILE *fp;
  
  fp = fopen( "file", "r" );
  if( fp != NULL ) {  
    printf( "File size=%ld\n", filesize( fp ) );
    fclose( fp );
  }  
}

Classification: ANSI

Systems:  
All, Netware
ftime

Synopsis:  

```c
#include <sys/timeb.h>
int ftime( struct timeb *timeptr );
```

```c
struct timeb {
    time_t time;  /* time in seconds since Jan 1, 1970 UTC */
    unsigned short millitm; /* milliseconds */
    short timezone; /* difference in minutes from UTC */
    short dstflag;  /* nonzero if in daylight savings time */
};
```

Description:  The `ftime` function gets the current time and stores it in the structure pointed to by `timeptr`.

Returns:  The `ftime` function fills in the fields of the structure pointed to by `timeptr`. The `ftime` function returns -1 if not successful, and no useful value otherwise.

See Also:  `asctime` Functions, `asctime_s`, `clock`, `ctime` Functions, `ctime_s`, `difftime`, `gmtime`, `gmtime_s`, `localtime`, `localtime_s`, `mktime`, `strftime`, `time`, `tzset`

Example:  

```c
#include <stdio.h>
#include <time.h>
#include <sys/timeb.h>

void main()
{
    struct timeb timebuf;
    char   *tod;
    ftime( &timebuf );
    tod = ctime( &timebuf.time );
    printf( "The time is %.19s.%hu %s",
            tod, timebuf.millitm, &tod[20] );
}
```

produces the following:

```
The time is Tue Dec 25 15:58:42.870 1990
```

Classification:  WATCOM

Systems:  All
Synopsis:

```
#include <stdlib.h>
char *fullpath( char *buffer,
    const char *path,
    size_t size );
wchar_t *wfullpath( wchar_t *buffer ,
    const wchar_t *path,
    size_t size );
```

Description: The _fullpath function returns the full pathname of the file specification in path in the specified buffer buffer of length size.

The maximum size that might be required for buffer is __MAX_PATH. If the buffer provided is too small, NULL is returned and errno is set.

If buffer is NULL then a buffer of size __MAX_PATH is allocated using malloc. This buffer may be freed using the free function.

If path is NULL or points to a null string ("") then the current working directory is returned in buffer.

The _wfullpath function is a wide-character version of _fullpath that operates with wide-character strings.

Returns: The _fullpath function returns a pointer to the full path specification if no error occurred. Otherwise, NULL is returned.

Errors: When an error has occurred, errno contains a value indicating the type of error that has been detected.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENOENT</td>
<td>The current working directory could not be obtained.</td>
</tr>
<tr>
<td>ENOMEM</td>
<td>The buffer could not be allocated.</td>
</tr>
<tr>
<td>ERANGE</td>
<td>The buffer passed was too small.</td>
</tr>
</tbody>
</table>

See Also: _makepath, _splitpath

Example:

```
#include <stdio.h>
#include <stdlib.h>

void main( int argc, char *argv[] )
{
    int i;
    char buff[ PATH_MAX ];

    for( i = 1; i < argc; ++i ) {
        puts( argv[i] );
        if( _fullpath( buff, argv[i], PATH_MAX ) ) {
            puts( buff );
        } else {
            puts( "FAIL!" );
        }
    }
}
```
Classification: WATCOM

Systems:  
   _fullpath - All, Netware  
   _wfullpath - DOS, Windows, Win386, Win32, OS/2 1.x (all), OS/2-32
Synopsis:  

```c
#include <stdio.h>
#include <wchar.h>
int fwide( FILE *fp, int mode );
```

Description:  The `fwide` function determines the orientation of the stream pointed to by `fp`. If `mode` is greater than zero, the function first attempts to make the stream wide oriented. If `mode` is less than zero, the function first attempts to make the stream byte oriented. Otherwise, `mode` is zero and the `fwide` function does not alter the orientation of the stream.

Returns:  The `fwide` function returns a value greater than zero if, after the call, the stream has wide orientation, a value less than zero if the stream has byte orientation, or zero if the stream has no orientation.

See Also:  `fopen`, `freopen`

Example:  

```c
#include <stdio.h>
#include <wchar.h>

void main( void )
{
    FILE  *fp;
    int   mode;

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        mode = fwide( fp, -33 );
        printf( "orientation: %s\n",
                mode > 0 ? "wide" :
                mode < 0 ? "byte" : "none" );
    }
}
```

produces the following:

```
orientation: byte
```

Classification:  ANSI

Systems:  All
fwrite

Synopsis:
#include <stdio.h>
size_t fwrite( const void *buf,
    size_t elsize,
    size_t nelem,
    FILE *fp );

Description: The fwrite function writes nelem elements of elsize bytes each to the file specified by fp.

Returns: The fwrite function returns the number of complete elements successfully written. This value will be less than the requested number of elements only if a write error occurs. When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also: ferror, fopen

Example:
#include <stdio.h>

struct student_data {
    int  student_id;
    unsigned char marks[10];
};

void main()
{
    FILE *fp;
    struct student_data std;
    int i;

    fp = fopen( "file", "w" );
    if( fp != NULL ) {
        std.student_id = 1001;
        for( i = 0; i < 10; i++ )
            std.marks[ i ] = (unsigned char) (85 + i);

        /* write student record with marks */
        i = fwrite( &std, sizeof(std), 1, fp );
        printf( "%d record written\n", i );
        fclose( fp );
    }
}

Classification: ANSI

Systems: All, Netware

312  Library Functions and Macros
Synopsis:

```c
#include <stdlib.h>
char *gcvt( double value,
            int ndigits,
            char *buffer );
char *__gcvt( double value,
             int ndigits,
             char *buffer );
wchar_t *__wgcvt( double value,
               int ndigits,
               wchar_t *buffer );
```

Description:

The `gcvt` function converts the floating-point number `value` into a character string and stores the result in `buffer`. The parameter `ndigits` specifies the number of significant digits desired. The converted number will be rounded to this position.

If the exponent of the number is less than -4 or is greater than or equal to the number of significant digits wanted, then the number is converted into E-format, otherwise the number is formatted using F-format.

The `__gcvt` function is identical to `gcvt`. Use `__gcvt` for ANSI/ISO naming conventions.

The `__wgcvt` function is identical to `gcvt` except that it produces a wide-character string (which is twice as long).

Returns:

The `gcvt` function returns a pointer to the string of digits.

See Also:

`ecvt`, `fcvt`, `printf`

Example:

```c
#include <stdio.h>
#include <stdlib.h>

void main()
{
    char buffer[80];

    printf( "%s\n", gcvt( -123.456789, 5, buffer ) );
    printf( "%s\n", gcvt( 123.456789E+12, 5, buffer ) );
}
```

produces the following:

```
-123.46
1.2346E+014
```

Classification:

WATCOM

- `gcvt` conforms to ANSI/ISO naming conventions

Systems:

- `gcvt` - Math
- `__gcvt` - Math
- `__wgcvt` - Math


**Synopsis:**

```
#include <graph.h>
short _FAR _getactivepage( void );
```

**Description:**

The `_getactivepage` function returns the number of the currently selected active graphics page.

Only some combinations of video modes and hardware allow multiple pages of graphics to exist. When multiple pages are supported, the active page may differ from the visual page. The graphics information in the visual page determines what is displayed upon the screen. Animation may be accomplished by alternating the visual page. A graphics page can be constructed without affecting the screen by setting the active page to be different than the visual page.

The number of available video pages can be determined by using the `_getvideoconfig` function. The default video page is 0.

**Returns:**

The `_getactivepage` function returns the number of the currently selected active graphics page.

**See Also:**

`_setactivepage`, `setvisualpage`, `_getvisualpage`, `_getvideoconfig`.

**Example:**

```
#include <conio.h>
#include <graph.h>

main()
{
    int old_apage;
    int old_vpage;

    _setvideomode( _HRES16COLOR );
    old_apage = _getactivepage();
    old_vpage = _getvisualpage();
    /* draw an ellipse on page 0 */
    _setactivepage( 0 );
    _setvisualpage( 0 );
    _ellipse( _GFILLINTERIOR, 100, 50, 540, 150 );
    /* draw a rectangle on page 1 */
    _setactivepage( 1 );
    _rectangle( _GFILLINTERIOR, 100, 50, 540, 150 );
    getch();
    /* display page 1 */
    _setvisualpage( 1 );
    getch();
    _setactivepage( old_apage );
    _setvisualpage( old_vpage );
    _setvideomode( _DEFAULTMODE );
}
```

**Classification:**

`_getactivepage` is PC Graphics

**Systems:**

DOS, QNX
Synopsis:

```c
#include <graph.h>
short _FAR _getarcinfo( struct xycoord _FAR *start_pt,
                        struct xycoord _FAR *end_pt,
                        struct xycoord _FAR *inside_pt );
```

Description:
The `getarcinfo` function returns information about the arc most recently drawn by the `arc` or `pie` functions. The arguments `start_pt` and `end_pt` are set to contain the endpoints of the arc. The argument `inside_pt` will contain the coordinates of a point within the pie. The points are all specified in the view coordinate system.

The endpoints of the arc can be used to connect other lines to the arc. The interior point can be used to fill the pie.

Returns:
The `getarcinfo` function returns a non-zero value when successful. If the previous arc or pie was not successfully drawn, zero is returned.

See Also:
`arc`, `pie`

Example:

```c
#include <conio.h>
#include <graph.h>

main()
{
    struct xycoord start_pt, end_pt, inside_pt;

    _setvideomode( _VRES16COLOR );
    _arc( 120, 90, 520, 390, 520, 90, 120, 390 );
    _getarcinfo( &start_pt, &end_pt, &inside_pt );
    _moveto( start_pt.xcoord, start_pt.ycoord );
    _lineto( end_pt.xcoord, end_pt.ycoord );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

produces the following:
Classification: PC Graphics

Systems: DOS, QNX
Synopsis:  
#include <graph.h>
long _FAR _getbkcolor( void );

Description:  The _getbkcolor function returns the current background color. In text modes, the background color controls the area behind each individual character. In graphics modes, the background refers to the entire screen. The default background color is 0.

Returns:  The _getbkcolor function returns the current background color.

See Also: _setbkcolor, _remappalette

Example:  
#include <conio.h>
#include <graph.h>

long colors[ 16 ] = {
    _BLACK, _BLUE, _GREEN, _CYAN,
    _RED, _MAGENTA, _BROWN, _WHITE,
    _GRAY, _LIGHTBLUE, _LIGHTGREEN, _LIGHTCYAN,
    _LIGHTRED, _LIGHTMAGENTA, _YELLOW, _BRIGHTWHITE
};

main()
{
    long old_bk;
    int bk;

    _setvideomode( _VRES16COLOR );
    old_bk = _getbkcolor();
    for( bk = 0; bk < 16; ++bk ) {
        _setbkcolor( colors[ bk ] );
        getch();
    }
    _setbkcolor( old_bk );
    _setvideomode( _DEFAULTMODE );
}

Classification:  PC Graphics

Systems:  DOS, QNX
**getc, getwc**

**Synopsis:**

```c
#include <stdio.h>
int getc( FILE *fp );
#include <stdio.h>
#include <wchar.h>
wint_t getwc( FILE *fp );
```

**Description:**

The `getc` function gets the next character from the file designated by `fp`. The character is returned as an `int` value. The `getc` function is equivalent to `fgetc`, except that it may be implemented as a macro.

The `getwc` function is identical to `getc` except that it gets the next multibyte character (if present) from the input stream pointed to by `fp` and converts it to a wide character.

**Returns:**

The `getc` function returns the next character from the input stream pointed to by `fp`. If the stream is at end-of-file, the end-of-file indicator is set and `getc` returns `EOF`. If a read error occurs, the error indicator is set and `getc` returns `EOF`.

The `getwc` function returns the next wide character from the input stream pointed to by `fp`. If the stream is at end-of-file, the end-of-file indicator is set and `getwc` returns `WEOF`. If a read error occurs, the error indicator is set and `getwc` returns `WEOF`. If an encoding error occurs, `errno` is set to `EILSEQ` and `getwc` returns `WEOF`.

When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

**See Also:**

`fgetc`, `fgetchar`, `fgets`, `fopen`, `getchar`, `gets`, `ungetc`

**Example:**

```c
#include <stdio.h>

void main()
{
    FILE *fp;
    int c;

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        while( (c = getc( fp )) != EOF )
            putchar(c);

        fclose( fp );
    }
}
```

**Classification:**

- `getc` is ANSI
- `getwc` is ANSI

**Systems:**

- `getc` - All, Netware
- `getwc` - All

318  Library Functions and Macros
Synopsis: #include <conio.h>
int getch( void );

Description: The `getch` function obtains the next available keystroke from the console. Nothing is echoed on the screen (the function `getche` will echo the keystroke, if possible). When no keystroke is available, the function waits until a key is depressed.

The `kbhit` function can be used to determine if a keystroke is available.

Returns: A value of `EOF` is returned when an error is detected; otherwise the `getch` function returns the value of the keystroke (or character).

When the keystroke represents an extended function key (for example, a function key, a cursor-movement key or the ALT key with a letter or a digit), zero is returned and the next call to `getch` returns a value for the extended function.

See Also: `getche`, `kbhit`, `putch`, `ungetch`

Example: #include <stdio.h>
#include <conio.h>

void main()
{
    int c;
    printf( "Press any key\n" );
    c = getch();
    printf( "You pressed %c(%d)\n", c, c );
}

Classification: WATCOM

Systems: All, Netware
**Synopsis:**

```c
#include <stdio.h>
int getchar( void );
#include <wchar.h>
wint_t getwchar( void );
```

**Description:**

The `getchar` function is equivalent to `getc` with the argument `stdin`.

The `getwchar` function is similar to `getchar` except that it is equivalent to `getwc` with the argument `stdin`.

**Returns:**

The `getchar` function returns the next character from the input stream pointed to by `stdin`. If the stream is at end-of-file, the end-of-file indicator is set and `getchar` returns `EOF`. If a read error occurs, the error indicator is set and `getchar` returns `EOF`.

The `getwchar` function returns the next wide character from the input stream pointed to by `stdin`. If the stream is at end-of-file, the end-of-file indicator is set and `getwchar` returns `WEOF`. If a read error occurs, the error indicator is set and `getwchar` returns `WEOF`. If an encoding error occurs, `errno` is set to `EILSEQ` and `getwchar` returns `WEOF`.

When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

**See Also:**

`fgetc`, `fgetchar`, `fgets`, `fopen`, `getc`, `gets`, `ungetc`

**Example:**

```c
#include <stdio.h>

void main()
{
    FILE *fp;
    int c;

    fp = freopen( "file", "r", stdin );
    while( (c = getchar()) != EOF )
        putchar(c);
    fclose( fp );
}
```

**Classification:**

`getchar` is ANSI

`getwchar` is ANSI

**Systems:**

`getchar` - All, Netware

`getwchar` - All
Synopsis:  
#include <conio.h>
int getche( void );

Description:  The getche function obtains the next available keystroke from the console. The function will wait until a keystroke is available. That character is echoed on the screen at the position of the cursor (use getch when it is not desired to echo the keystroke).

The kbhit function can be used to determine if a keystroke is available.

Returns:  A value of EOF is returned when an error is detected; otherwise, the getche function returns the value of the keystroke (or character).

When the keystroke represents an extended function key (for example, a function key, a cursor-movement key or the ALT key with a letter or a digit), zero is returned and the next call to getche returns a value for the extended function.

See Also:  getch, kbhit, putch, ungetch

Example:  
#include <stdio.h>
#include <conio.h>

void main()
{
  int c;

  printf( "Press any key\n" );
  c = getche();
  printf( "You pressed %c(%d)\n", c, c );
}

Classification:  WATCOM

Systems:  All, Netware
Synopsis:  
#include <graph.h>
void _FAR _getcliplrgrn( short _FAR *x1, short _FAR *y1,
                        short _FAR *x2, short _FAR *y2 );

Description:  The _getcliprgrn function returns the location of the current clipping region. A clipping region is defined with the _setcliplrgrn or _setviewport functions. By default, the clipping region is the entire screen.

The current clipping region is a rectangular area of the screen to which graphics output is restricted. The top left corner of the clipping region is placed in the arguments (x1, y1). The bottom right corner of the clipping region is placed in (x2, y2).

Returns:  The _getcliprgrn function returns the location of the current clipping region.

See Also:  _setcliplrgrn, _setviewport

Example:  
#include <conio.h>
#include <graph.h>

main()
{
    short x1, y1, x2, y2;

    _setvideomode( _VRES16COLOR );
    _getcliplrgrn( &x1, &y1, &x2, &y2 );
    _setcliplrgrn( 130, 100, 510, 380 );
    _ellipse( _GBORDER, 120, 90, 520, 390 );
    getch();
    _setcliplrgrn( x1, y1, x2, y2 );
    _setvideomode( _DEFAULTMODE );
}

Classification:  PC Graphics

Systems:  DOS, QNX
Synopsis:  
#include <process.h>  
char *getcmd( char *cmd_line );

Description:  
The getcmd function causes the command line information, with the program name removed, to be copied to cmd_line. The information is terminated with a '\0' character. This provides a method of obtaining the original parameters to a program unchanged (with the white space intact).

This information can also be obtained by examining the vector of program parameters passed to the main function in the program.

Returns:  
The address of the target cmd_line is returned.

See Also:  
abort, atexit, _bgetcmd, exec..., exit, _Exit, _exit, getenv, main, onexit, putenv, spawn..., system

Example:  
Suppose a program were invoked with the command line

    myprog arg-1 ( my stuff ) here

where that program contains

    #include <stdio.h>  
    #include <process.h>

    void main()
    {
        char cmds[128];

        printf( "%s\n", getcmd( cmds ) );
    }

produces the following:

    arg-1 ( my stuff ) here

Classification: WATCOM

Systems: All, Netware
Synopsis:  
#include <graph.h>
short _FAR _getcolor( void );

Description:  The _getcolor function returns the pixel value for the current color. This is the color used for displaying graphics output. The default color value is one less than the maximum number of colors in the current video mode.

Returns:  The _getcolor function returns the pixel value for the current color.

See Also:  _setcolor

Example:  
#include <conio.h>
#include <graph.h>

main()
{
  int col, old_col;
  _setvideomode( _VRES16COLOR );
  old_col = _getcolor();
  for( col = 0; col < 16; ++col ) {
    _setcolor( col );
    _rectangle( _GFILLINTERIOR, 100, 100, 540, 380 );
    getch();
  }
  _setcolor( old_col );
  _setvideomode( _DEFAULTMODE );
}

Classification:  PC Graphics

Systems:  DOS, QNX
Synopsis:  
#include <graph.h>  
struct xycoord _FAR _getcurrentposition( void );  
struct _wxycoord _FAR _getcurrentposition_w( void );

Description:  The _getcurrentposition functions return the current output position for graphics. The _getcurrentposition function returns the point in view coordinates. The _getcurrentposition_w function returns the point in window coordinates.

The current position defaults to the origin, (0,0), when a new video mode is selected. It is changed by successful calls to the _arc, _moveto and _lineto functions as well as the _setviewport function.

Note that the output position for graphics output differs from that for text output. The output position for text output can be set by use of the _settextposition function.

Returns:  The _getcurrentposition functions return the current output position for graphics.

See Also:  _moveto, _settextposition

Example:  
#include <conio.h>  
#include <graph.h>

main()
{
    struct xycoord old_pos;
    _setvideomode(_VRES16COLOR);
    old_pos = _getcurrentposition();
    _moveto(100, 100);
    _lineto( 540, 100 );
    _lineto( 320, 380 );
    _lineto( 100, 100 );
    _moveto( old_pos.xcoord, old_pos.ycoord );
    getch();
    _setvideomode(_DEFAULTMODE);
}

Classification: PC Graphics

Systems:  _getcurrentposition - DOS, QNX  
_getcurrentposition_w - DOS, QNX
**getcwd, _wgetcwd**

**Synopsis:**
```
#include <direct.h>
char *getcwd( char *buffer, size_t size );
wchar_t * _wgetcwd( wchar_t *buffer, size_t size );
```

**Description:** The getcwd function returns the name of the current working directory. The `buffer` address is either `NULL` or is the location at which a string containing the name of the current working directory is placed. In the latter case, the value of `size` is the length (including the delimiting `\0` character) which can be used to store this name.

The maximum size that might be required for `buffer` is `PATH_MAX + 1` bytes.

**Extension:** When `buffer` has a value of `NULL`, a string is allocated using `malloc` to contain the name of the current working directory. This string may be freed using the `free` function. The `_wgetcwd` function is identical to `getcwd` except that it returns the name of the current working directory as a wide-character string (which is twice as long).

**Returns:** The `getcwd` function returns the address of the string containing the name of the current working directory, unless an error occurs, in which case `NULL` is returned.

**Errors:** When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EINVAL</td>
<td>The argument <code>size</code> is negative.</td>
</tr>
<tr>
<td>ENOMEM</td>
<td>Not enough memory to allocate a buffer.</td>
</tr>
<tr>
<td>ERANGE</td>
<td>The buffer is too small (specified by <code>size</code>) to contain the name of the current working directory.</td>
</tr>
</tbody>
</table>

**See Also:** `chdir`, `chmod`, `_getdcwd`, `mkdir`, `rmdir`

**Example:**
```
#include <stdio.h>
#include <stdlib.h>
#include <direct.h>

void main()
{
    char *cwd;
    
    cwd = getcwd( NULL, 0 );
    if( cwd != NULL ) {
        printf( "My working directory is %s\n", cwd );
        free( cwd );
    }
}
```
produces the following:
```
My working directory is C:\PROJECT\C
```

**Classification:** `getcwd` is POSIX 1003.1 with extensions

_wgetcwd is not POSIX
Systems: getcwd - All, Netware
         _wgetcwd - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:
#include <direct.h>
char * _getdcwd( int drive, char *buffer, size_t maxlen );
wchar_t * _wgetdcwd( int drive, wchar_t *buffer, 
size_t maxlen );

Description: The _getdcwd function gets the full path of the current working directory on the specified drive. The drive argument specifies the drive (0 = default drive, 1 = A, 2 = B, etc.). The buffer address is either NULL or is the location at which a string containing the name of the current working directory is placed. In the latter case, the value of maxlen is the length (including the terminating ‘\0’ character) which can be be used to store this name. An error occurs if the length of the path (including the terminating ‘\0’ character) exceeds maxlen.

The maximum size that might be required for buffer is PATH_MAX + 1 bytes.

When buffer has a value of NULL, a string is allocated using malloc to contain the name of the current working directory. This string may be freed using the free function. The _wgetdcwd function is identical to _getdcwd except that it returns the name of the current working directory as a wide-character string (which is twice as long).

Returns: The _getdcwd function returns the address of the string containing the name of the current working directory on the specified drive, unless an error occurs, in which case NULL is returned.

Errors: When an error has occurred, errno contains a value indicating the type of error that has been detected.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENODEV</td>
<td>The drive cannot be accessed.</td>
</tr>
<tr>
<td>ENOMEM</td>
<td>Not enough memory to allocate a buffer.</td>
</tr>
<tr>
<td>ERANGE</td>
<td>The buffer is too small (specified by size)</td>
</tr>
<tr>
<td></td>
<td>to contain the name of the current working</td>
</tr>
<tr>
<td></td>
<td>directory.</td>
</tr>
</tbody>
</table>

See Also: chdir, chmod, getcwd, mkdir, rmdir

Example: #include <stdio.h>
#include <stdlib.h>
#include <direct.h>

void main()
{
    char * cwd;

    cwd = _getdcwd( 3, NULL, 0 );
    if( cwd != NULL ) {
        printf( "The current directory on drive C is %s\n", 
                cwd );
        free( cwd );
    }
}

produces the following:

The current directory on drive C is C:\PROJECT\C
Classification: WATCOM

Systems:  
_getcwd - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32  
_wgetcwd - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <direct.h>
unsigned _getdiskfree( unsigned drive,
          struct diskfree_t *diskspace );

struct diskfree_t {
    unsigned short total_clusters;
    unsigned short avail_clusters;
    unsigned short sectors_per_cluster;
    unsigned short bytes_per_sector;
};

Description: The _getdiskfree function uses system call 0x36 to obtain useful information on the disk drive specified by drive. Specify 0 for the default drive, 1 for drive A, 2 for drive B, etc. The information about the drive is returned in the structure diskfree_t pointed to by diskspace.

Returns: The _getdiskfree function returns zero if successful. Otherwise, it returns a non-zero value and sets errno to EINVAL indicating an invalid drive was specified.

See Also: _dos_getdiskfree,_dos_getdrive,_dos_setdrive,_getdrive

Example:  
#include <stdio.h>
#include <direct.h>

void main()
{
    struct diskfree_t disk_data;

    /* get information about drive 3 (the C drive) */
    if( _getdiskfree( 3, &disk_data ) == 0 ) {
        printf( "total clusters: %u\n",
                disk_data.total_clusters );
        printf( "available clusters: %u\n",
                disk_data.avail_clusters );
        printf( "sectors/cluster: %u\n",
                disk_data.sectors_per_cluster );
        printf( "bytes per sector: %u\n",
                disk_data.bytes_per_sector );
    } else {
        printf( "Invalid drive specified\n" );
    }
}

produces the following:

total clusters: 16335
available clusters: 510
sectors/cluster: 4
bytes per sector: 512

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis: #include <direct.h>
int _getdrive( void );

Description: The _getdrive function returns the current (default) drive number.

Returns: A value of 1 is drive A, 2 is drive B, 3 is drive C, etc.

See Also: _dos_getdiskfree, _dos_getdrive, _dos_setdrive, _getdiskfree

Example: #include <stdio.h>
#include <direct.h>

void main( void )
{
    printf( "The current drive is %c\n", 'A' + _getdrive() - 1 );
}

produces the following:
The current drive is C

Classification: DOS

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

```c
#include <stdlib.h>
char *getenv( const char *name );
wchar_t * _wgetenv( const wchar_t *name );
```

Safer C:
The Safer C Library extension provides the `getenv_s` function which is a safer alternative to `getenv`. This newer `getenv_s` function is recommended to be used instead of the traditional "unsafe" `getenv` function.

Description:
The `getenv` function searches the environment list for an entry matching the string pointed to by `name`. The matching is case-insensitive; all lowercase letters are treated as if they were in upper case.

Entries can be added to the environment list with the DOS `set` command or with the `putenv` or `setenv` functions. All entries in the environment list can be displayed by using the DOS `set` command with no arguments.

To assign a string to a variable and place it in the environment list:

```bash
C>SET INCLUDE=C:\WATCOM\H
```

To see what variables are in the environment list, and their current assignments:

```bash
C>SET
COMSPEC=C:\COMMAND.COM
PATH=C:\;C:\WATCOM
INCLUDE=C:\WATCOM\H
```

`_wgetenv` is a wide-character version of `getenv` the argument and return value of `_wgetenv` are wide-character strings.

Returns:
The `getenv` function returns a pointer to the string assigned to the environment variable if found, and NULL if no match was found. Note: the value returned should be duplicated if you intend to modify the contents of the string.

See Also:
`clearenv`, `exec...`, `getenv_s`, `putenv`, `_searchenv`, `setenv`, `spawn...`, `system`

Example:

```c
#include <stdio.h>
#include <stdlib.h>

void main( void )
{
    char *path;

    path = getenv( "INCLUDE" );
    if( path != NULL )
        printf( "INCLUDE=%s\n", path );
}
```

Classification:
- `getenv` is ANSI
- `_wgetenv` is not ANSI

Systems:
- `getenv` - All, Netware
- `_wgetenv` - All
### Synopsis:
```c
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
errno_t getenv_s( size_t * restrict len,
    char * restrict value,
    rsize_t maxsize,
    const char * restrict name );
```

### Constraints:
If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `getenv_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

- `name` shall not be a null pointer. `maxsize` shall neither be equal to zero nor be greater than `RSIZE_MAX`. If `maxsize` is not equal to zero, then `value` shall not be a null pointer.

If there is a runtime-constraint violation, the integer pointed to by `len` (if `len` is not null) is set to zero, and the environment list is not searched.

### Description:
The `getenv_s` function searches the environment list for an entry matching the string pointed to by `name`.

If that entry is found, `getenv_s` performs the following actions. If `len` is not a null pointer, the length of the string associated with the matched entry is stored in the integer pointed to by `len`. If the length of the associated string is less than `maxsize`, then the associated string is copied to the array pointed to by `value`.

If that entry is not found, `getenv_s` performs the following actions. If `len` is not a null pointer, zero is stored in the integer pointed to by `len`. If `maxsize` is greater than zero, then `value[0]` is set to the null character.

The matching is case-insensitive; all lowercase letters are treated as if they were in upper case.

Entries can be added to the environment list with the DOS `set` command or with the `putenv` or `setenv` functions. All entries in the environment list can be displayed by using the DOS `set` command with no arguments.

To assign a string to a variable and place it in the environment list:
```c
C>SET INCLUDE=C:\WATCOM\H
```

To see what variables are in the environment list, and their current assignments:
```c
C>SET
COMSPEC=C:\COMMAND.COM
PATH=C:\;C:\WATCOM
INCLUDE=C:\WATCOM\H
```

### Returns:
The `getenv_s` function returns zero if the environment string specified by `name` was found and successfully stored in the buffer pointed to by `value`. Otherwise, a non-zero value is returned.

### See Also:
`clearenv`, `exec...`, `getenv`, `putenv`, `_searchenv`, `setenv`, `spawn...`, `system`
Example:

```c
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
#include <stdio.h>

void main( void )
{
    char    buffer[128];
    size_t  len;

    if( getenv_s( &len, buffer, sizeof( buffer ), "INCLUDE" ) == 0 )
        printf( "INCLUDE=%s
", buffer );
}
```

Classification: TR 24731

Systems: All, Netware
Synopsis:  
#include <graph.h>
unsigned char _FAR * _FAR  
_getfillmask( unsigned char _FAR *mask );

Description:  The _getfillmask function copies the current fill mask into the area located by the argument mask. The fill mask is used by the _ellipse, _floodfill, _pie, _polygon and _rectangle functions that fill an area of the screen.

The fill mask is an eight-byte array which is interpreted as a square pattern (8 by 8) of 64 bits. Each bit in the mask corresponds to a pixel. When a region is filled, each point in the region is mapped onto the fill mask. When a bit from the mask is one, the pixel value of the corresponding point is set using the current plotting action with the current color; when the bit is zero, the pixel value of that point is not affected.

When the fill mask is not set, a fill operation will set all points in the fill region to have a pixel value of the current color.

Returns:  If no fill mask has been set, NULL is returned; otherwise, the _getfillmask function returns mask.

See Also:  _floodfill, _setfillmask, _setplotaction

Example:  
#include <conio.h>
#include <graph.h>

char old_mask[ 8 ];
char new_mask[ 8 ] = { 0x81, 0x42, 0x24, 0x18,
                      0x18, 0x24, 0x42, 0x81 };

main()
{
    _setvideomode( _VRES16COLOR );
    _getfillmask( old_mask );
    _setfillmask( new_mask );
    _rectangle( _GFILLINTERIOR, 100, 100, 540, 380 );
    _setfillmask( old_mask );
    getch();
    _setvideomode( _DEFAULTMODE );
}

Classification:  PC Graphics

Systems:  DOS, QNX
Synopsis:

```
#include <graph.h>
short _FAR _getfontinfo( struct _fontinfo _FAR *info );
```

Description: The `_getfontinfo` function returns information about the currently selected font. Fonts are selected with the `_setfont` function. The font information is returned in the `_fontinfo` structure indicated by the argument `info`. The structure contains the following fields:

- **type**: 1 for a vector font, 0 for a bit-mapped font
- **ascent**: distance from top of character to baseline in pixels
- **pixwidth**: character width in pixels (0 for a proportional font)
- **pixheight**: character height in pixels
- **avgwidth**: average character width in pixels
- **filename**: name of the file containing the current font
- **facename**: name of the current font

Returns: The `_getfontinfo` function returns zero if the font information is returned successfully; otherwise a negative value is returned.

See Also: `_registerfonts`, `_unregisterfonts`, `_setfont`, `_outgtext`, `_getgtextextent`, `_setgtextvector`, `_getgtextvector`

Example:

```c
#include <conio.h>
#include <graph.h>

main()
{
    int width;
    struct _fontinfo info;

    _setvideomode( _VRES16COLOR );
    _getfontinfo( &info );
    _moveto( 100, 100 );
    _outgtext( "WATCOM Graphics" );
    width = _getgtextextent( "WATCOM Graphics" );
    _rectangle( _GBORDER, 100, 100,
                100 + width, 100 + info.pixheight );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX
Synopsis:  
#include <graph.h>
short _FAR _getgtextextent( char _FAR *text );

Description:  The _getgtextextent function returns the length in pixels of the argument text as it would be
displayed in the current font by the function _outgtext. Note that the text is not displayed on the
screen, only its length is determined.

Returns:  The _getgtextextent function returns the length in pixels of a string.

See Also:  _registerfonts, _unregisterfonts, _setfont, _getfontinfo, _outgtext,
_setgtextvector, _getgtextvector

Example:  #include <conio.h>
#include <graph.h>

main()
{
  int width;
  struct _fontinfo info;

  _setvideomode( _VRES16COLOR );
  _getfontinfo( &info );
  _moveto( 100, 100 );
  _outgtext( "WATCOM Graphics" );
  width = _getgtextextent( "WATCOM Graphics" );
  _rectangle( _GBORDER, 100, 100,
   100 + width, 100 + info.pixheight );
  getch();
  _setvideomode( _DEFAULTMODE );
}

Classification:  PC Graphics

Systems:  DOS, QNX
Synopsis: #include <graph.h>
struct xycoord _FAR _getgtextvector( void );

Description: The _getgtextvector function returns the current value of the text orientation vector. This is the direction used when text is displayed by the _outgtext function.

Returns: The _getgtextvector function returns, as an xycoord structure, the current value of the text orientation vector.

See Also: _registerfonts, _unregisterfonts, _setfont, _getfontinfo, _outgtext, _getgtextextent, _setgtextvector

Example: #include <conio.h>
#include <graph.h>

main()
{
    struct xycoord old_vec;

    _setvideomode( _VRES16COLOR );
    old_vec = _getgtextvector();
    _setgtextvector( 0, -1 );
    _moveto( 100, 100 );
    _outgtext( "WATCOM Graphics" );
    _setgtextvector( old_vec.xcoord, old_vec.ycoord );
    getch();
    _setvideomode( _DEFAULTMODE );
}

Classification: PC Graphics

Systems: DOS, QNX
Synopsis: 

```c
#include <graph.h>
void _FAR _getimage( short x1, short y1,
                     short x2, short y2,
                     char _HUGE *image );

void _FAR _getimage_w( double x1, double y1,
                       double x2, double y2,
                       char _HUGE *image );

void _FAR _getimage_wxy( struct _wxycoord _FAR *p1,
                        struct _wxycoord _FAR *p2,
                        char _HUGE *image );
```

Description: The `_getimage` functions store a copy of an area of the screen into the buffer indicated by the `image` argument. The `_getimage` function uses the view coordinate system. The `_getimage_w` and `_getimage_wxy` functions use the window coordinate system.

The screen image is the rectangular area defined by the points `(x1,y1)` and `(x2,y2)`. The buffer `image` must be large enough to contain the image (the size of the image can be determined by using the `_imagesize` function). The image may be displayed upon the screen at some later time by using the `_putimage` functions.

Returns: The `_getimage` functions do not return a value.

See Also: `_imagesize`, `_putimage`

Example: 

```c
#include <conio.h>
#include <graph.h>
#include <malloc.h>

main()
{
    char *buf;
    int y;

    _setvideomode( _VRES16COLOR );
    _ellipse( _GFILLINTERIOR, 100, 100, 200, 200 );
    buf = (char*) malloc(
        _imagesize( 100, 100, 201, 201 ) );
    if( buf != NULL ) {
        _getimage( 100, 100, 201, 201, buf );
        _putimage( 260, 200, buf, _GPSET );
        _putimage( 420, 100, buf, _GPSET );
        for( y = 100; y < 300; ) {
            _putimage( 420, y, buf, _GXOR );
            y += 20;
            _putimage( 420, y, buf, _GXOR );
        }
        free( buf );
    }
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics
_getimage Functions

Systems:

_getimage - DOS, QNX
_getimage_w - DOS, QNX
_getimage_wxy - DOS, QNX
Synopsis:  
#include <graph.h>
unsigned short _FAR _getlinestyle( void );

Description:  The _getlinestyle function returns the current line-style mask.

The line-style mask determines the style by which lines and arcs are drawn. The mask is treated as an array of 16 bits. As a line is drawn, a pixel at a time, the bits in this array are cyclically tested. When a bit in the array is 1, the pixel value for the current point is set using the current color according to the current plotting action; otherwise, the pixel value for the point is left unchanged. A solid line would result from a value of 0xFFFF and a dashed line would result from a value of 0xF0F0

The default line style mask is 0xFFFF

Returns:  The _getlinestyle function returns the current line-style mask.

See Also:  _lineto, _pie, _rectangle, _polygon, _setlinestyle

Example:  
#include <conio.h>
#include <graph.h>

#define DASHED 0xf0f0

main()
{
    unsigned old_style;

    _setvideomode( _VRES16COLOR );
    old_style = _getlinestyle();
    _setlinestyle( DASHED );
    _rectangle( _GBORDER, 100, 100, 540, 380 );
    _setlinestyle( old_style );
    getch();
    _setvideomode( _DEFAULTMODE );
}

Classification:  PC Graphics

Systems:  DOS, QNX
Synopsis:  
#include <mbctype.h>  
int _getmbcp( void );

Description:  The _getmbcp function returns the current multibyte code page number.

Returns:  The _getmbcp function returns the current multibyte code page. A return value of zero indicates that
a single byte code page is in use.

See Also:  _mbbtombc,_mbcjistojms,_mbcjstojis,_mbctombb,_ismbbalnum,_ismbbalpha,  
_ismbbgraph,_ismbbkalnum,_ismbbkalpha,_ismbbkana,_ismbbbkprint,  
_ismbbbkpunc,_ismbbblead,_ismbbprint,_ismbbpunct,_ismbbtrail,_mbbtombc,  
_mbctombc,_mbcjistojms,_mbcjstojis,_mbctombb,_mbbtype,_setmbcp

Example:  
#include <stdio.h>  
#include <mbctype.h>

void main()  
{  
    printf( "%d\n", _setmbcp( 932 ) );
    printf( "%d\n", _getmbcp() );
}

produces the following:

0
932

Classification:  WATCOM

Systems:  DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis: 
```c
#include <unistd.h>

int getopt( int argc, char * const argv[],
             const char *optstring );
```

```c
char   *optarg;
int    optind, opterr, optopt;
```

Description: The **getopt** function is a command-line parser that can be used by applications that follow Utility Syntax Guidelines 3, 4, 5, 6, 7, 9 and 10 in the Base Definitions volume of IEEE Std 1003.1-2001, Section 12.2, Utility Syntax Guidelines.

The parameters *argc* and *argv* are the argument count and argument array as passed to *main*. The argument *optstring* is a string of recognised option characters; if a character is followed by a colon, the option takes an argument. All option characters allowed by Utility Syntax Guideline 3 are allowed in *optstring*.

The global variable *optind* is the index of the next element of the *argv[]* vector to be processed. It is initialised to 1 by the system, and *getopt* updates it when it finishes with each element of *argv[].* When an element of *argv[]* contains multiple option characters, *getopt* uses a static variable to determine which options have already been processed.

The *getopt* function returns the next option character (if one is found) from *argv* that matches a character in *optstring*, if there is one that matches. If the option takes an argument, *getopt* sets the variable *optarg* to point to the option-argument as follows:

If the option was the last character in the string pointed to by an element of *argv*, then *optarg* contains the next element of *argv* and *optind* is incremented by 2. If the resulting value of *optind* is not less than *argc*, this indicates a missing option-argument, and *getopt* returns an error indication.

Otherwise, *optarg* points to the string following the option character in that element of *argv*, and *optind* is incremented by 1.

If, when *getopt* is called:

- *argv[optind]* is a null pointer
- *argv[optind]* is not the character '-'
- *argv[optind]* points to the string "-

*getopt* returns *-1* without changing *optind*. If *argv[optind]* points to the string "--", *getopt* returns *-1* after incrementing *optind*.

If *getopt* encounters an option character that is not contained in *optstring*, it returns the question-mark (?) character. If it detects a missing option-argument, it returns the colon character (:) if the first character of *optstring* was a colon, or a question-mark character (?) otherwise. In either case, *getopt* will set the global variable *opterr* to the option character that caused the error. If the application has not set the global variable *opterr* to 0 and the first character of *optstring* is not a colon, *getopt* also prints a diagnostic message to *stderr*.

The *getopt* function is not re-entrant and hence not thread-safe.
getopt
Returns:

The getopt function returns the next option character specified on the command line.
A colon (:) is returned if getopt detects a missing argument and the first character of optstring was a
colon (:).
A question mark (?) is returned if getopt encounters an option character not in optstring or detects a
missing argument and the first character of optstring was not a colon (:).
Otherwise, getopt returns -1 when all command line options are parsed.

See Also:

abort, atexit, _bgetcmd, exec..., exit, _Exit, _exit, getcmd, getenv, main,
onexit, putenv, spawn..., system

Example:

#include <stdio.h>
#include <unistd.h>
int main( int argc, char **argv )
{
int
c;
char
*ifile;
char
*ofile;
while( (c = getopt( argc, argv, ":abf:o:" )) != -1 ) {
switch( c ) {
case ’a’:
printf( "option a is set\n" );
break;
case ’b’:
printf( "option b is set\n" );
break;
case ’f’:
ifile = optarg;
printf( "input filename is ’%s’\n", ifile );
break;
case ’o’:
ofile = optarg;
printf( "output filename is ’%s’\n", ofile );
break;
case ’:’:
printf( "-%c without filename\n", optopt );
break;
case ’?’:
printf( "usage: %s -ab -f <filename> -o <filename>\n", ar
gv[0] );
break;
}
}
return( 0 );
}
produces the following:
option a is set
input filename is ’in’
output filename is ’out’

344 Library Functions and Macros


when the program is executed with the command

<program name> -afin -o out

Classification: POSIX

Systems: All
Synopsis: 
#include <io.h>
long _get_osfhandle( int posixhandle );

Description: The _get_osfhandle function returns the operating system’s internal file handle that corresponds to the POSIX-level file handle specified by posixhandle.

The value returned by _get_osfhandle can be used as an argument to the _open_osfhandle function which can be used to connect a second POSIX-level handle to an open file.

The example below demonstrates the use of these two functions. Note that the example shows how the dup2 function can be used to obtain almost identical functionality.

When the POSIX-level file handles associated with one OS file handle are closed, the first one closes successfully but the others return an error (since the first call close the file and released the OS file handle). So it is important to call close at the right time, i.e., after all I/O operations are completed to the file.

Returns: If successful, _get_osfhandle returns an operating system file handle corresponding to posixhandle. Otherwise, it returns -1 and sets errno to EBADF, indicating an invalid file handle.

See Also: close, dup2, fdopen, _hdopen, open, _open_osfhandle, _os_handle

Example: #include <stdio.h>
#include <stdlib.h>
#include <io.h>
#include <fcntl.h>

void main()
{
    long os_handle;
    int fh1, fh2, rc;

    fh1 = open( "file",
        O_WRONLY | O_CREAT | O_TRUNC | O_BINARY,
        S_IRUSR | S_IWUSR | S_IRGRP | S_IWRGP );
    if( fh1 == -1 ) {
        printf( "Could not open output file\n" );
        exit( EXIT_FAILURE );
    }
    printf( "First POSIX handle %d\n", fh1 );
}
#if defined(USE_DUP2)
    fh2 = 6;
    if( dup2( fh1, fh2 ) == -1 ) fh2 = -1;
#else
    os_handle = _get_osfhandle( fh1 );
    printf( "OS Handle %ld\n", os_handle );
    fh2 = _open_osfhandle( os_handle, O_WRONLY | O_BINARY );
#endif
    if( fh2 == -1 ) {
        printf( "Could not open with second handle\n" );
        exit( EXIT_FAILURE );
    }
    printf( "Second POSIX handle %d\n", fh2 );
    rc = write( fh2, "trash\x0d\x0a", 7 );
    printf( "Write file using second handle %d\n", rc );
    rc = close( fh2 );
    printf( "Closing second handle %d\n", rc );
    rc = close( fh1 );
    printf( "Closing first handle %d\n", rc );
}

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, Netware
Synopsis: #include <graph.h>
struct xycoord _FAR _getphyscoord( short x, short y );

Description: The getphyscoord function returns the physical coordinates of the position with view coordinates (x,y). View coordinates are defined by the _setvieworg and _setviewport functions.

Returns: The _getphyscoord function returns the physical coordinates, as an xycoord structure, of the given point.

See Also: _getviewcoord, _setvieworg, _setviewport

Example: #include <conio.h>
#include <graph.h>
#include <stdlib.h>

main()
{
    struct xycoord pos;
    _setvideomode( _VRES16COLOR );
    _setvieworg( rand() % 640, rand() % 480 );
    pos = _getphyscoord( 0, 0 );
    _rectangle( _GBORDER, - pos.xcoord, - pos.ycoord, 639 - pos.xcoord, 479 - pos.ycoord );
    getch();
    _setvideomode( _DEFAULTMODE );
}

Classification: _getphyscoord is PC Graphics

Systems: DOS, QNX
Synopsis:  
#include <process.h>
int getpid(void);

Description:  The getpid function returns the process id for the current process.

Returns:  The getpid function returns the process id for the current process.

Example:  
#include <stdio.h>
#include <process.h>

void main()
{
    unsigned int process_id;
    auto char filename[13];

    process_id = getpid();
    /* use this to create a unique file name */
    sprintf( filename, "TMP%4.4x.TMP", process_id );
}

Classification:  POSIX 1003.1

Systems:  All
_getpixel Functions

Synopsis:    
#include <graph.h> 
short _FAR _getpixel( short x, short y ); 
short _FAR _getpixel_w( double x, double y ); 

Description: The _getpixel functions return the pixel value for the point with coordinates (x,y). The _getpixel function uses the view coordinate system. The _getpixel_w function uses the window coordinate system.

Returns: The _getpixel functions return the pixel value for the given point when the point lies within the clipping region; otherwise, (-1) is returned.

See Also: _setpixel

Example:    
#include <conio.h> 
#include <graph.h> 
#include <stdlib.h> 

main() { 
    int x, y; 
    unsigned i; 
    _setvideomode( _VRES16COLOR ); 
    _rectangle( _GBORDER, 100, 100, 540, 380 ); 
    for( i = 0; i <= 60000; ++i ) {
        x = 101 + rand() % 439; 
        y = 101 + rand() % 279; 
        _setcolor( _getpixel( x, y ) + 1 ); 
        _setpixel( x, y ); 
    } 
    getch(); 
    _setvideomode( _DEFAULTMODE ); 
} 

Classification: PC Graphics

Systems: _getpixel - DOS, QNX 
         _getpixel_w - DOS, QNX

350  Library Functions and Macros
Synopsis: #include <graph.h>
short _FAR _getplotaction( void );

Description: The _getplotaction function returns the current plotting action.

The drawing functions cause pixels to be set with a pixel value. By default, the value to be set is obtained by replacing the original pixel value with the supplied pixel value. Alternatively, the replaced value may be computed as a function of the original and the supplied pixel values.

The plotting action can have one of the following values:

_GPSET replace the original screen pixel value with the supplied pixel value

_GAND replace the original screen pixel value with the bitwise and of the original pixel value and the supplied pixel value

_GOR replace the original screen pixel value with the bitwise or of the original pixel value and the supplied pixel value

_GXOR replace the original screen pixel value with the bitwise exclusive-or of the original pixel value and the supplied pixel value. Performing this operation twice will restore the original screen contents, providing an efficient method to produce animated effects.

Returns: The _getplotaction function returns the current plotting action.

See Also: _setplotaction

Example: #include <conio.h>
#include <graph.h>

main()
{
    int old_act;
    _setvideomode( _VRES16COLOR );
    old_act = _getplotaction();
    _setplotaction( _GPSET );
    _rectangle( _GFILLINTERIOR, 100, 100, 540, 380 );
    getch();
    _setplotaction( _GXOR );
    _rectangle( _GFILLINTERIOR, 100, 100, 540, 380 );
    getch();
    _setplotaction( old_act );
    _setvideomode( _DEFAULTMODE );
}

Classification: PC Graphics

Systems: DOS, QNX

Library Functions and Macros 351
Synopsis:

```
#include <stdio.h>
char *gets( char *buf );
#include <stdio.h>
wchar_t *__getws( wchar_t *buf );
```

Description:
The `gets` function gets a string of characters from the file designated by `stdin` and stores them in the array pointed to by `buf` until end-of-file is encountered or a new-line character is read. Any new-line character is discarded, and a null character is placed immediately after the last character read into the array.

The `_getws` function is identical to `gets` except that it gets a string of multibyte characters (if present) from the input stream pointed to by `stdin`, converts them to wide characters, and stores them in the wide-character array pointed to by `buf` until end-of-file is encountered or a wide-character new-line character is read.

It is recommended that `fgets` be used instead of `gets` because data beyond the array `buf` will be destroyed if a new-line character is not read from the input stream `stdin` before the end of the array `buf` is reached.

A common programming error is to assume the presence of a new-line character in every string that is read into the array. A new-line character may not appear as the last character in a file, just before end-of-file.

Returns:
The `gets` function returns `buf` if successful. `NULL` is returned if end-of-file is encountered, or if a read error occurs. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fgetc`, `fgetchar`, `fgets`, `fopen`, `getc`, `getchar`, `ungetc`

Example:
```
#include <stdio.h>

void main()
{
    char buffer[80];

    while( gets( buffer ) != NULL )
        puts( buffer );
}
```

Classification: `gets` is ANSI

`__getws` is not ANSI

Systems: `gets` - All, Netware

`__getws` - All
Synopsis:  
#define __STDC_WANT_LIB_EXT1__ 1  
#include <stdio.h>  
char *gets_s( char *s, rsize_t n );

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and gets_s will set s[0] to be the null character, and characters are read and discarded from stdin until a new-line character is read, or end-of-file or a read error occurs.

s shall not be a null pointer. n shall neither be equal to zero nor be greater than RSIZE_MAX. A new-line character, end-of-file, or read error shall occur within reading n-1 characters from stdin.

Description: The gets_s function gets a string of characters from the file designated by stdin and stores them in the array pointed to by s until end-of-file is encountered or a new-line character is read. Size of the array s is specified by the argument n, this information is used to protect buffer from overflow. If buffer s is about to be overflown, runtime-constraint is activated. Any new-line character is discarded, and a null character is placed immediately after the last character read into the array.

Returns: The gets_s function returns s if successful. NULL is returned if there was a runtime-constraint violation, or if end-of-file is encountered and no characters have been read into the array, or if a read error occurs.

See Also: fgetc, fgetchar, fgets, fopen, getc, getchar, gets, ungetc

Example:  
#define __STDC_WANT_LIB_EXT1__ 1  
#include <stdio.h>

int main()
{
    char buffer[80];

    while( gets_s( buffer, sizeof( buffer ) ) != NULL )
        puts( buffer );
}

Classification: TR 24731
Synopsis:  
#include <graph.h>
short _FAR _gettextcolor( void );

Description:  The _gettextcolor function returns the pixel value of the current text color. This is the color used for displaying text with the _outtext and _outmem functions. The default text color value is set to 7 whenever a new video mode is selected.

Returns:  The _gettextcolor function returns the pixel value of the current text color.

See Also:  _settextcolor, _setcolor, _outtext, _outmem

Example:  
#include <conio.h>
#include <graph.h>
main()
{
  int old_col;
  long old_bk;
  _setvideomode( _TEXTC80 );
  old_col = _gettextcolor();
  old_bk = _getbkcolor();
  _settextcolor( 7 );
  _setbkcolor( _BLUE );
  _outtext( " WATCOM \nGraphics" );
  _settextcolor( old_col );
  _setbkcolor( old_bk );
  getch();
  _setvideomode( _DEFAULTMODE );
}

Classification:  PC Graphics

Systems:  DOS, QNX
Synopsis: #include <graph.h>
short _FAR _gettextcursor( void );

Description: The _gettextcursor function returns the current cursor attribute, or shape. The cursor shape is set
with the _settextcursor function. See the _settextcursor function for a description of the
value returned by the _gettextcursor function.

Returns: The _gettextcursor function returns the current cursor shape when successful; otherwise, (-1) is
returned.

See Also: _settextcursor, _displaycursor

Example: #include <conio.h>
#include <graph.h>

main()
{
    int old_shape;

    old_shape = _gettextcursor();
    _settextcursor( 0x0007 );
    _outtext( "\nBlock cursor" );
    getch();
    _settextcursor( 0x0407 );
    _outtext( "\nHalf height cursor" );
    getch();
    _settextcursor( 0x2000 );
    _outtext( "\nNo cursor" );
    getch();
    _settextcursor( old_shape );
}

Classification: PC Graphics

Systems: DOS, QNX
### _gettextextent

**Synopsis:**
```
#include <graph.h>

void _FAR _gettextextent( short x, short y,
                          char _FAR *text,
                          struct xycoord _FAR *concat,
                          struct xycoord _FAR *extent );
```

**Description:** The `_gettextextent` function simulates the effect of using the `_grtext` function to display the text string `text` at the position `(x, y)`, using the current text settings. The concatenation point is returned in the argument `concat`. The text extent parallelogram is returned in the array `extent`.

The concatenation point is the position to use to output text after the given string. The text extent parallelogram outlines the area where the text string would be displayed. The four points are returned in counter-clockwise order, starting at the upper-left corner.

**Returns:** The `_gettextextent` function does not return a value.

**See Also:** `_grtext, _gettextsettings`

**Example:**
```
#include <conio.h>
#include <graph.h>

main()
{
    struct xycoord concat;
    struct xycoord extent[ 4 ];

    _setvideomode( _VRES16COLOR );
    _grtext( 100, 100, "hot" );
    _gettextextent( 100, 100, "hot", &concat, extent );
    _polygon( _GBORDER, 4, extent );
    _grtext( concat.xcoord, concat.ycoord, "dog" );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

produces the following:
Classification: PC Graphics

Systems: DOS, QNX


**Synopsis:**

```c
#include <graph.h>
struct rcord _FAR _gettextposition( void );
```

**Description:**

The `_gettextposition` function returns the current output position for text. This position is in terms of characters, not pixels.

The current position defaults to the top left corner of the screen, `(1,1)`, when a new video mode is selected. It is changed by successful calls to the `outtext`, `outmem`, `settextposition` and `settextwindow` functions.

Note that the output position for graphics output differs from that for text output. The output position for graphics output can be set by use of the `moveto` function.

**Returns:**

The `_gettextposition` function returns, as an `rcoord` structure, the current output position for text.

**See Also:**

`outtext`, `outmem`, `settextposition`, `settextwindow`, `moveto`

**Example:**

```c
#include <conio.h>
#include <graph.h>

main()
{
    struct rcord old_pos;

    _setvideomode( _TEXTC80 );
    old_pos = _gettextposition();
    settextposition( 10, 40 );
    outtext( "WATCOM Graphics" );
    settextposition( old_pos.row, old_pos.col );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

**Classification:**

PC Graphics

**Systems:**

DOS, QNX
Synopsis:  
#include <graph.h>
struct textsettings _FAR * _FAR _gettextsettings
( struct textsettings _FAR *settings );

Description:  The _gettextsettings function returns information about the current text settings used when text is displayed by the _grtext function. The information is stored in the textsettings structure indicated by the argument settings. The structure contains the following fields (all are short fields):

- **basevectorx**  x-component of the current base vector
- **basevectory**  y-component of the current base vector
- **path**  current text path
- **height**  current text height (in pixels)
- **width**  current text width (in pixels)
- **spacing**  current text spacing (in pixels)
- **horizalign** horizontal component of the current text alignment
- **vertalign**  vertical component of the current text alignment

Returns:  The _gettextsettings function returns information about the current graphics text settings.

See Also:  _grtext, _setcharsize, _setcharspacing, _settextalign, _settextpath, _settextorient

Example:  
#include <conio.h>
#include <graph.h>

main()
{
    struct textsettings ts;
    _setvideomode( _VRES16COLOR );
    _gettextsettings( &ts );
    _grtext( 100, 100, "WATCOM" );
    _setcharsize( 2 * ts.height, 2 * ts.width );
    _grtext( 100, 300, "Graphics" );
    _setcharsize( ts.height, ts.width );
    getch();
    _setvideomode( _DEFAULTMODE );
}

Classification:  PC Graphics

Systems:  DOS, QNX
Synopsis:

```c
#include <graph.h>
void _FAR _gettextwindow(
    short _FAR *row1, short _FAR *col1,
    short _FAR *row2, short _FAR *col2);
```

Description: The `_gettextwindow` function returns the location of the current text window. A text window is defined with the `_settextwindow` function. By default, the text window is the entire screen.

The current text window is a rectangular area of the screen. Text display is restricted to be within this window. The top left corner of the text window is placed in the arguments `(row1, col1)`. The bottom right corner of the text window is placed in `(row2, col2)`.

Returns: The `_gettextwindow` function returns the location of the current text window.

See Also: `_settextwindow`, `_outtext`, `_outmem`, `_settextposition`, `_scrolltextwindow`

Example:

```c
#include <conio.h>
#include <stdio.h>
main()
{
    int i;
    short r1, c1, r2, c2;
    char buf[ 80 ];

    _setvideomode( _TEXTC80 );
    _gettextwindow( &r1, &c1, &r2, &c2 );
    _settextwindow( 5, 20, 20, 40 );
    for( i = 1; i <= 20; ++i ) {
        sprintf( buf, "Line %d\n", i );
        _outtext( buf );
    }
    getch();
    _settextwindow( r1, c1, r2, c2 );
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX
Synopsis:  

```c
#include <graph.h>

struct videoconfig _FAR * _FAR _getvideoconfig
  ( struct videoconfig _FAR *config );
```

Description:  
The `_getvideoconfig` function returns information about the current video mode and the hardware configuration. The information is returned in the `videoconfig` structure indicated by the argument `config`. The structure contains the following fields (all are `short` fields):

- `numxpixes`  number of pixels in x-axis
- `numypxels`  number of pixels in y-axis
- `numtextcols`  number of text columns
- `numtextrows`  number of text rows
- `numcolors`  number of actual colors
- `bitsperpixel`  number of bits in a pixel value
- `numvideopages`  number of video pages
- `mode`  current video mode
- `adapter`  adapter type
- `monitor`  monitor type
- `memory`  number of kilobytes (1024 characters) of video memory

The `adapter` field will contain one of the following values:

- `_NODISPLAY`  no display adapter attached
- `_UNKNOWN`  unknown adapter/monitor type
- `_MDPA`  Monochrome Display/Printer Adapter
- `_CGA`  Color Graphics Adapter
- `_HERCULES`  Hercules Monochrome Adapter
- `_MCGA`  Multi-Color Graphics Array
- `_EGA`  Enhanced Graphics Adapter
- `_VGA`  Video Graphics Array
- `_SVGA`  SuperVGA Adapter
The `getvideoconfig` field will contain one of the following values:

- `_MONO` regular monochrome
- `_COLOR` regular color
- `_ENHANCED` enhanced color
- `_ANALOGMONO` analog monochrome
- `_ANALOGCOLOR` analog color

The amount of memory reported by `getvideoconfig` will not always be correct for SuperVGA adapters. Since it is not always possible to determine the amount of memory, `getvideoconfig` will always report 256K, the minimum amount.

**Returns:** The `getvideoconfig` function returns information about the current video mode and the hardware configuration.

**See Also:** `setvideomode`, `setvideomoderows`
Example:
```c
#include <conio.h>
#include <graph.h>
#include <stdio.h>
#include <stdlib.h>

main()
{
    int mode;
    struct videoconfig vc;
    char buf[ 80 ];

    _getvideoconfig( &vc );
    /* select "best" video mode */
    switch( vc.adapter ) {
        case _VGA :
        case _SVGA :
            mode = _VRES16COLOR;
            break;
        case _MCGA :
            mode = _MRES256COLOR;
            break;
        case _EGA :
            if( vc.monitor == _MONO ) { /* select "best" video mode */
                mode = _ERESNOCOLOR;
            } else {
                mode = _ERESCOLOR;
            }
            break;
        case _CGA :
            mode = _MRES4COLOR;
            break;
        case _HERCULES :
            mode = _HERCMONO;
            break;
        default :
            puts( "No graphics adapter" );
            exit( 1 );
    }

    if( _setvideomode( mode ) ) {
        _getvideoconfig( &vc );
        sprintf( buf, "%d x %d x %d
", vc.numxpixels,
                 vc.numypixels, vc.numcolors );
        _outtext( buf );
        getch();
        _setvideomode( _DEFAULTMODE );
    }
}
```

Classification: PC Graphics

Systems: DOS, QNX
Synopsis:  
#include <graph.h>  
struct xycoord _FAR _getviewcoord( short x, short y );

struct xycoord _FAR _getviewcoord_w( double x, double y );

struct xycoord _FAR _getviewcoord_wxy(
        struct _wxycoord _FAR *p );

Description: The _getviewcoord functions translate a point from one coordinate system to viewport coordinates.  
The _getviewcoord function translates the point (x,y) from physical coordinates. The  
_getviewcoord_w and _getviewcoord_wxy functions translate the point from the window  
coordinate system.

Viewport coordinates are defined by the _setvieworg and _setviewport functions. Window  
coordinates are defined by the _setwindow function.

Note: In previous versions of the software, the _getviewcoord function was called  
_getlogcoord. uindex=2

Returns:  
The _getviewcoord functions return the viewport coordinates, as an xycoord structure, of the  
given point.

See Also:  
_getphyscoord, _setvieworg, _setviewport, _setwindow

Example:  
#include <conio.h>  
#include <graph.h>  
#include <stdlib.h>

main()
{
 struct xycoord pos1, pos2;

 _setvideomode( _VRES16COLOR );
 _setvieworg( rand() % 640, rand() % 480 );
 pos1 = _getviewcoord( 0, 0 );
 pos2 = _getviewcoord( 639, 479 );
 _rectangle( _GBORDER, pos1.xcoord, pos1.ycoord,  
            pos2.xcoord, pos2.ycoord );
 _setwindowfunc( _GBORDER );

 Classification:  
PC Graphics

Systems:  
_getviewcoord - DOS, QNX  
_getviewcoord_w - DOS, QNX  
_getviewcoord_wxy - DOS, QNX

364 Library Functions and Macros
Synopsis:
#include <graph.h>
short _FAR _getvisualpage(void);

Description:
The _getvisualpage function returns the number of the currently selected visual graphics page.

Only some combinations of video modes and hardware allow multiple pages of graphics to exist. When multiple pages are supported, the active page may differ from the visual page. The graphics information in the visual page determines what is displayed upon the screen. Animation may be accomplished by alternating the visual page. A graphics page can be constructed without affecting the screen by setting the active page to be different than the visual page.

The number of available video pages can be determined by using the _getvideoconfig function. The default video page is 0.

Returns:
The _getvisualpage function returns the number of the currently selected visual graphics page.

See Also:
_setvisualpage, _setactivepage, _getactivepage, _getvideoconfig

Example:
#include <conio.h>
#include <graph.h>

main()
{
    int old_apage;
    int old_vpage;
    _setvideomode(_HRES16COLOR);
    old_apage = _getactivepage();
    old_vpage = _getvisualpage();
    /* draw an ellipse on page 0 */
    _setactivepage(0);
    _setvisualpage(0);
    _ellipse(_GFILLINTERIOR, 100, 50, 540, 150);
    /* draw a rectangle on page 1 */
    _setactivepage(1);
    _rectangle(_GFILLINTERIOR, 100, 50, 540, 150);
    getch();
    /* display page 1 */
    _setvisualpage(1);
    getch();
    _setactivepage(old_apage);
    _setvisualpage(old_vpage);
    _setvideomode(_DEFAULTMODE);
}

Classification: PC Graphics

Systems: DOS, QNX
Synopsis: #include <stdio.h>
int _getw( int binint, FILE *fp );

Description: The _getw function reads a binary value of type int from the current position of the stream fp and increments the associated file pointer to point to the next unread character in the input stream. _getw does not assume any special alignment of items in the stream.

_getw is provided primarily for compatibility with previous libraries. Portability problems may occur with _getw because the size of an int and the ordering of bytes within an int differ across systems.

Returns: The _getw function returns the integer value read or, if a read error or end-of-file occurs, the error indicator is set and _getw returns EOF. Since EOF is a legitimate value to read from fp, use ferror to verify that an error has occurred.

See Also: ferror, fgetc, fgetchar, fgets, fopen, getc, getchar, gets, _putw, ungetc

Example: #include <stdio.h>

void main()
{
    FILE *fp;
    int c;

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        while( (c = _getw( fp )) != EOF )
            _putw( c, stdout );
    }
    fclose( fp );
}

Classification: WATCOM

Systems: All, Netware
Synopsis:  
#include <graph.h>  
struct _wxycoord _FAR _getwindowcoord( short x, short y );

Description:  The _getwindowcoord function returns the window coordinates of the position with view coordinates (x,y). Window coordinates are defined by the _setwindow function.

Returns:  The _getwindowcoord function returns the window coordinates, as a _wxycoord structure, of the given point.

See Also:  _setwindow, _getviewcoord

Example:  
#include <conio.h>  
#include <graph.h>

main()
{
    struct xycoord centre;
    struct _wxycoord pos1, pos2;

    /* draw a box 50 pixels square */
    /* in the middle of the screen */
    _setvideomode( _MAXRESMODE );
    centre = _getviewcoord( 0.5, 0.5 );
    pos1 = _getwindowcoord( centre.xcoord - 25,
                            centre.ycoord - 25 );
    pos2 = _getwindowcoord( centre.xcoord + 25,
                            centre.ycoord + 25 );
    _rectangle_wxy( _GBORDER, &pos1, &pos2 );
    getch();
    _setvideomode( _DEFAULTMODE );
}

Classification:  _getwindowcoord is PC Graphics

Systems:  DOS, QNX
gmtime Functions

Synopsis:
```c
#include <time.h>

struct tm * gmtime( const time_t *timer );
struct tm *__gmtime( const time_t *timer,
                      struct tm *tmbuf );
```

```c
struct tm {
    int tm_sec;   /* seconds after the minute -- [0,61] */
    int tm_min;   /* minutes after the hour   -- [0,59] */
    int tm_hour;  /* hours after midnight -- [0,23] */
    int tm_mday;  /* day of the month   -- [1,31] */
    int tm_mon;   /* months since January -- [0,11] */
    int tm_year;  /* years since 1900                   */
    int tm_wday;  /* days since Sunday       -- [0,6]  */
    int tm_yday;  /* days since January 1   -- [0,365]*/
    int tm_isdst; /* Daylight Savings Time flag */
};
```

Safer C: The Safer C Library extension provides the gmtime_s function which is a safer alternative to gmtime. This newer gmtime_s function is recommended to be used instead of the traditional "unsafe" gmtime function.

Description: The gmtime functions convert the calendar time pointed to by timer into a broken-down time, expressed as Coordinated Universal Time (UTC) (formerly known as Greenwich Mean Time (GMT)). The function __gmtime places the converted time in the tm structure pointed to by tmbuf, and the gmtime gmtime places the converted time in a static structure that is re-used each time gmtime is called.

The time set on the computer with the DOS time command and the DOS date command reflects the local time. The environment variable TZ is used to establish the time zone to which this local time applies. See the section The TZ Environment Variable for a discussion of how to set the time zone.

Returns: The gmtime functions return a pointer to a structure containing the broken-down time.

See Also: asctime Functions, asctime_s, clock, ctime Functions, ctime_s, difftime, gmtime_s, localtime, localtime_s, mktime, strftime, time, tzset

Example:
```c
#include <stdio.h>
#include <time.h>

void main()
{
    time_t time_of_day;
    auto char buf[26];
    auto struct tm tmbuf;

    time_of_day = time( NULL );
    __gmtime( &time_of_day, &tmbuf );
    printf( "It is now: %.24s GMT\n",
            __asctime( &tmbuf, buf ) );
}
```

produces the following:
gmtime Functions

It is now: Fri Dec 25 15:58:27 1987 GMT

Classification: gmtime is ANSI
               _gmtime is not ANSI

Systems:      gmtime - All, Netware
               _gmtime - All
Synopsis:  
#define __STDC_WANT_LIB_EXT1__ 1
#include <time.h>
struct tm * gmtime_s( const time_t * restrict timer,
    struct tm * restrict result );

struct tm {
    int tm_sec;   /* seconds after the minute -- [0,61] */
    int tm_min;   /* minutes after the hour   -- [0,59] */
    int tm_hour;  /* hours after midnight     -- [0,23] */
    int tm_mday;  /* day of the month         -- [1,31] */
    int tm_mon;   /* months since January     -- [0,11] */
    int tm_year;  /* years since 1900                   */
    int tm_wday;  /* days since Sunday        -- [0,6]  */
    int tm_yday;  /* days since January 1      -- [0,365]*/
    int tm_isdst; /* Daylight Savings Time flag */
};

Constraints:  If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and gmtime_s will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither timer nor result shall be a null pointer. If there is a runtime-constraint violation, there is no attempt to convert the time.

Description:  The gmtime_s function converts the calendar time pointed to by timer into a broken-down time, expressed as UTC. The broken-down time is stored in the structure pointed to by result.

Returns:  The gmtime_s function returns result, or a null pointer if the specified time cannot be converted to UTC or there is a runtime-constraint violation.

See Also: asctime Functions, asctime_s, clock, ctime Functions, ctime_s, difftime, gmtime, localtime, localtime_s, mktime, strftime, time, tzset

Example:  
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <time.h>

void main()
{
    time_t time_of_day;
    auto char buf[26];
    auto struct tm tmbuf;

    time_of_day = time( NULL );
    gmtime_s( &time_of_day, &tmbuf );
    asctime_s( buf, sizeof( buf ), &tmbuf );
    printf( "It is now: %.24s GMT\n", buf );
}

produces the following:

It is now: Thu Jan 31 15:12:27 2006 GMT

Classification: TR 24731
Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, Netware
**Synopsis:**
#include <stdio.h>
int _grow_handles( int new_count );

**Description:**
The `_grow_handles` function increases the number of POSIX level files that are allowed to be open at one time. The parameter `new_count` is the new requested number of files that are allowed to be opened. The return value is the number that is allowed to be opened after the call. This may be less than, equal to, or greater than the number requested. If the number is less than, an error has occurred and the `errno` variable should be consulted for the reason. If the number returned is greater than or equal to the number requested, the call was successful.

Note that even if `_grow_handles` returns successfully, you still might not be able to open the requested number of files due to some system limit (e.g. FILES= in the CONFIG.SYS file under DOS) or because some file handles are already in use (stdin, stdout, stderr, etc.).

The number of file handles that the run-time system can open by default is described by `__NFILES` in `<stdio.h>` but this can be changed by the application developer. To change the number of file handles available during execution, follow the steps outlined below.

1. Let `n` represent the number of files to be opened concurrently. Ensure that the stdin, stdout, and stderr files are included in the count. Also include stdaux and stdprn files in the count for some versions of DOS. The stdaux and stdprn files are not available for Win32.

2. For DOS-based systems, change the `CONFIG.SYS` file to include "FILES=n" where "n" is the number of file handles required by the application plus an additional 5 handles for the standard files. The number of standard files that are opened by DOS varies from 3 to 5 depending on the version of DOS that you are using.

   If you are running a network such as Novell’s NetWare, this will also affect the number of available file handles. In this case, you may have to increase the number specified in the "FILES=n" statement.

3. Add a call to `_grow_handles` in your application similar to that shown in the example below.

**Returns:**
The `_grow_handles` function returns the maximum number of file handles which the run-time system can accommodate. This number can exceed an operating system limit such as that imposed by the "FILES=" statement under DOS. This limit will be the determining factor in how many files can be open concurrently.

**Errors:**
When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

**See Also:**
_dos_open, fdopen, fileno, fopen, freopen, _fsopen, _hdopen, open,
_open_osfhandle, _popen, sopen, tmpfile

**Example:**
#include <stdio.h>

FILE *fp[ 50 ];

void main()
{
    int hndl_count;
    int i;
}
```c
hndl_count = _NFILES;
if( hndl_count < 50 ) {
    hndl_count = _grow_handles( 50 );
}
for( i = 0; i < hndl_count; i++ ) {
    fp[ i ] = tmpfile();
    if( fp[ i ] == NULL ) break;
    printf( "File %d successfully opened\n", i );
}
printf( "%d files were successfully opened\n", i );
```

Classification: WATCOM

Systems: All
Synopsis: #include <graph.h>
short _FAR _grstatus( void );

Description: The _grstatus function returns the status of the most recently called graphics library function. The function can be called after any graphics function to determine if any errors or warnings occurred. The function returns 0 if the previous function was successful. Values less than 0 indicate an error occurred; values greater than 0 indicate a warning condition.

The following values can be returned:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>_GROK</td>
<td>0</td>
<td>no error</td>
</tr>
<tr>
<td>_GRERROR</td>
<td>-1</td>
<td>graphics error</td>
</tr>
<tr>
<td>_GRMOTIONOT_SUPPORTED</td>
<td>-2</td>
<td>video mode not supported</td>
</tr>
<tr>
<td>_GRNOTINPROPERMODE</td>
<td>-3</td>
<td>function n/a in this mode</td>
</tr>
<tr>
<td>_GRINVALIDPARAMETER</td>
<td>-4</td>
<td>invalid parameter(s)</td>
</tr>
<tr>
<td>_GRINSUFFICIENTMEMORY</td>
<td>-5</td>
<td>out of memory</td>
</tr>
<tr>
<td>_GRFONTFILENOTFOUND</td>
<td>-6</td>
<td>can’t open font file</td>
</tr>
<tr>
<td>_GRINVALIDFONTFILE</td>
<td>-7</td>
<td>font file has invalid format</td>
</tr>
<tr>
<td>_GRNOOUTPUT</td>
<td>1</td>
<td>nothing was done</td>
</tr>
<tr>
<td>_GRCLIPPED</td>
<td>2</td>
<td>output clipped</td>
</tr>
</tbody>
</table>

Returns: The _grstatus function returns the status of the most recently called graphics library function.

Example: #include <conio.h>
#include <graph.h>
#include <stdlib.h>

main()
{
  int x, y;
  _setvideomode( _VRES16COLOR );
  while( _grstatus() == _GROK ) {
    x = rand() % 700;
    y = rand() % 500;
    _setpixel( x, y );
  }
  getch();
  _setvideomode( _DEFAULTMODE );
}

Classification: _grstatus is PC Graphics

Systems: DOS, QNX

374 Library Functions and Macros
Synopsis:

```c
#include <graph.h>
short _FAR _grtext( short x, short y,
                    char _FAR *text );
short _FAR _grtext_w( double x, double y,
                      char _FAR *text );
```

Description:

The _grtext functions display a character string. The _grtext function uses the view coordinate system. The _grtext_w function uses the window coordinate system.

The character string text is displayed at the point \((x, y)\). The string must be terminated by a null character (\'\0\'). The text is displayed in the current color using the current text settings.

The graphics library can display text in three different ways.

1. The _outtext and _outmem functions can be used in any video mode. However, this variety of text can be displayed in only one size.
2. The _grtext function displays text as a sequence of line segments, and can be drawn in different sizes, with different orientations and alignments.
3. The _outgtext function displays text in the currently selected font. Both bit-mapped and vector fonts are supported; the size and type of text depends on the fonts that are available.

Returns:

The _grtext functions return a non-zero value when the text was successfully drawn; otherwise, zero is returned.

See Also:

_outtext, _outmem, _outgtext, _setcharsize, _settextalign, _settextpath, _settextorient, _setcharspacing

Example:

```c
#include <conio.h>
#include <graph.h>

main()
{
    _setvideomode( _VRES16COLOR );
    _grtext( 200, 100, "WATCOM" );
    _grtext( 200, 200, "Graphics" );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

produces the following:
**Classification:** PC Graphics

**Systems:**

- `_grtext` - DOS, QNX
- `_grtext_w` - DOS, QNX
Synopsis:    #include <malloc.h>
        void __huge *halloc( long int numb, size_t size );

Description:  The halloc function allocates space for an array of numb objects of size bytes each and initializes
each object to 0. When the size of the array is greater than 64K bytes, then the size of an array element
must be a power of 2 since an object could straddle a segment boundary.

Returns:  The halloc function returns a far pointer (of type void huge *) to the start of the allocated
memory. The NULL value is returned if there is insufficient memory available. The NULL value is also
returned if the size of the array is greater than 64K bytes and the size of an array element is not a power
of 2.

See Also:  calloc Functions, _expand Functions, free Functions, hfree, malloc Functions, _msize
Functions, realloc Functions, sbrk

Example:  #include <stdio.h>
        #include <malloc.h>

        void main()
        {
            long int __huge *big_buffer;

            big_buffer = (long int __huge *)
                halloc( 1024L, sizeof(long) );
            if( big_buffer == NULL ) {
                printf( "Unable to allocate memory\n" );
            } else {
                /* rest of code goes here */

                hfree( big_buffer );  /* deallocate */
            }
        }

Classification:  WATCOM

Systems:  DOS/16, Windows, QNX/16, OS/2 1.x(all)
Synopsis: 
#include <dos.h>
void _harderr( int (__far *handler)() );
void _hardresume( int action );
void _hardretn( int error );

Description: The _harderr routine installs a critical error handler (for INT 0x24) to handle hardware errors. This critical error handler will call the user-defined function specified by handler when a critical error occurs (for example, attempting to open a file on a floppy disk when the drive door is open). The parameters to this function are as follows:

    int handler( unsigned deverror,
                 unsigned errcode,
                 unsigned __far *devhdr );

The low-order byte of errcode can be one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>Attempt to write to a write-protected disk</td>
</tr>
<tr>
<td>0x01</td>
<td>Unknown unit</td>
</tr>
<tr>
<td>0x02</td>
<td>Drive not ready</td>
</tr>
<tr>
<td>0x03</td>
<td>Unknown command</td>
</tr>
<tr>
<td>0x04</td>
<td>CRC error in data</td>
</tr>
<tr>
<td>0x05</td>
<td>Bad drive-request structure length</td>
</tr>
<tr>
<td>0x06</td>
<td>Seek error</td>
</tr>
<tr>
<td>0x07</td>
<td>Unknown media type</td>
</tr>
<tr>
<td>0x08</td>
<td>Sector not found</td>
</tr>
<tr>
<td>0x09</td>
<td>Printer out of paper</td>
</tr>
<tr>
<td>0x0A</td>
<td>Write fault</td>
</tr>
<tr>
<td>0x0B</td>
<td>Read fault</td>
</tr>
<tr>
<td>0x0C</td>
<td>General failure</td>
</tr>
</tbody>
</table>

The devhdr argument points to a device header control-block that contains information about the device on which the error occurred. Your error handler may inspect the information in this control-block but must not change it.

If the error occurred on a disk device, bit 15 of the deverror argument will be 0 and the deverror argument will indicate the following:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 15</td>
<td>0 indicates disk error</td>
</tr>
<tr>
<td>bit 14</td>
<td>not used</td>
</tr>
<tr>
<td>bit 13</td>
<td>0 indicates &quot;Ignore&quot; response not allowed</td>
</tr>
<tr>
<td>bit 12</td>
<td>0 indicates &quot;Retry&quot; response not allowed</td>
</tr>
<tr>
<td>bit 11</td>
<td>0 indicates &quot;Fail&quot; response not allowed</td>
</tr>
<tr>
<td>bit 9,10</td>
<td>location of error</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>MS-DOS</td>
</tr>
<tr>
<td>01</td>
<td>File Allocation Table (FAT)</td>
</tr>
</tbody>
</table>
10 Directory
11 Data area

bit 8
0 indicates read error, 1 indicates write error

The low-order byte of `deverror` indicates the drive where the error occurred; (0 = drive A, 1 = drive B, etc.).

The handler is very restricted in the type of system calls that it can perform. System calls 0x01 through 0x0C, and 0x59 are the only system calls allowed to be issued by the handler. Therefore, many of the standard C run-time functions such as stream I/O and low-level I/O cannot be used by the handler. Console I/O is allowed (e.g., `cprintf`, `cputs`).

The handler must indicate what action to take by returning one of the following values or calling `_hardresume` with one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_HARDERR_IGNORE</code></td>
<td>Ignore the error</td>
</tr>
<tr>
<td><code>_HARDERR_RETRY</code></td>
<td>Retry the operation</td>
</tr>
<tr>
<td><code>_HARDERR_ABORT</code></td>
<td>Abort the program issuing INT 0x23</td>
</tr>
<tr>
<td><code>_HARDERR_FAIL</code></td>
<td>Fail the system call that is in progress (DOS 3.0 or higher)</td>
</tr>
</tbody>
</table>

Alternatively, the handler can return directly to the application program rather than returning to DOS by using the `_hardretn` function. The application program resumes at the point just after the failing I/O function request. The `_hardretn` function should be called only from within a user-defined hardware error-handler function.

The `error` argument of `_hardretn` should be a DOS error code. See *The MS-DOS Encyclopedia* or *Programmer’s PC Sourcebook, 2nd Edition*, for more detailed information on DOS error codes that may be returned by a given DOS function call.

If the failing I/O function request is an INT 0x21 function greater than or equal to function 0x38, `_hardretn` will return to the application with the carry flag set and the AX register set to the `_hardretn` `error` argument. If the failing INT 0x21 function request is less than function 0x38 and the function can return an error, the AL register will be set to 0xFF on return to the application. If the failing INT 0x21 function does not have a way of returning an error condition (which is true of certain INT 0x21 functions below 0x38), the `error` argument of `_hardretn` is not used, and no error code is returned to the application.

**Returns:** These functions do not return a value. The `_hardresume` and `_hardretn` functions do not return to the caller.

**See Also:** `_chain_intr`, `_dos_getvect`, `_dos_setvect`

**Example:**

Library Functions and Macros 379


```c
#include <stdio.h>
#include <conio.h>
#include <dos.h>

#if defined(__DOS__) && defined(__386__)
#define FAR __far
#else
#if defined(__386__)
#define FAR
#else
#define FAR __far
#endif
#endif

int FAR critical_error_handler( unsigned deverr,
                                  unsigned errcode,
                                  unsigned FAR *devhdr )
{
    cprintf( "Critical error: " );
    cprintf( "deverr=%4.4X errcode=%d\n", deverr, errcode );
    cprintf( "devhdr = %Fp\n", devhdr );
    return( _HARDERR_IGNORE );
}

main()
{
    FILE *fp;

    _harderr( critical_error_handler );
    fp = fopen( "a:tmp.tmp", "r" );
    printf( "fp = %p\n", fp );
}
```

produces the following:

```
Critical error: deverr=1A00 errcode=2
devhdr = 0070:01b6
fp = 0000
```

**Classification:** DOS

**Systems:**
- _harderr - DOS
- _hardresume - DOS
- _hardretn - DOS/16
Synopsis:  
#include <io.h>  
int _hdopen( int os_handle, int mode );

Description:  The _hdopen function takes a previously opened operating system file handle specified by os_handle and opened with access and sharing specified by mode, and creates a POSIX-style file handle.

Returns:  The _hdopen function returns the new POSIX-style file handle if successful. Otherwise, it returns -1.

See Also:  close, _dos_open, fdopen, fopen, freopen, _fsopen, _grow_handles, open, _open_osfhandle, _os_handle, _popen, sopen

Example:  
#include <stdio.h>  
#include <dos.h>  
#include <fcntl.h>  
#include <io.h>  
#include <windows.h>

void main()  
{  
    HANDLE os_handle;
    DWORD desired_access, share_mode;
    int handle;

    os_handle = CreateFileA( "file", GENERIC_WRITE, 0, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL );

    if( os_handle == INVALID_HANDLE_VALUE ) {  
        printf( "Unable to open file\n" );
    } else {  
        handle = _hdopen( os_handle, O_RDONLY );
        if( handle != -1 ) {  
            write( handle, "hello\n", 6 );
            close( handle );
        } else {  
            CloseHandle( os_handle );
        }
    }

}

Classification:  WATCOM

Systems:  All, Netware
_heapchk Functions

Synopsis:  
```c
#include <malloc.h>
int _heapchk( void );
int _bheapchk( __segment seg );
int _fheapchk( void );
int _nheapchk( void );
```

Description: The _heapchk functions along with _heapset and _heapwalk are provided for debugging heap related problems in programs.

The _heapchk functions perform a consistency check on the unallocated memory space or "heap". The consistency check determines whether all the heap entries are valid. Each function checks a particular heap, as listed below:

<table>
<thead>
<tr>
<th>Function</th>
<th>Heap Checked</th>
</tr>
</thead>
<tbody>
<tr>
<td>_heapchk</td>
<td>Depends on data model of the program</td>
</tr>
<tr>
<td>_bheapchk</td>
<td>Based heap specified by seg value; _NULLSEG specifies all based heaps</td>
</tr>
<tr>
<td>_fheapchk</td>
<td>Far heap (outside the default data segment)</td>
</tr>
<tr>
<td>_nheapchk</td>
<td>Near heap (inside the default data segment)</td>
</tr>
</tbody>
</table>

In a small data memory model, the _heapchk function is equivalent to the _nheapchk function; in a large data memory model, the _heapchk function is equivalent to the _fheapchk function.

Returns: All four functions return one of the following manifest constants which are defined in <malloc.h>.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>_HEAPOK</td>
<td>The heap appears to be consistent.</td>
</tr>
<tr>
<td>_HEAPEMPTY</td>
<td>The heap is empty.</td>
</tr>
<tr>
<td>_HEAPBADBEGIN</td>
<td>The heap has been damaged.</td>
</tr>
<tr>
<td>_HEAPBADNODE</td>
<td>The heap contains a bad node, or is damaged.</td>
</tr>
</tbody>
</table>

See Also: _heapenable, _heapgrow, _heapmin, _heapset, _heapshrink, _heapwalk

Example:  
```c
#include <stdio.h>
#include <malloc.h>

void main()
{
    char *buffer;
```
buffer = (char *)malloc( 80 );
malloc( 1024 );
free( buffer );
switch ( _heapchk() ) {
    case _HEAPOK:
        printf( "OK - heap is good\n" );
        break;
    case _HEAPEMPTY:
        printf( "OK - heap is empty\n" );
        break;
    case _HEAPBADBEGIN:
        printf( "ERROR - heap is damaged\n" );
        break;
    case _HEAPBADNODE:
        printf( "ERROR - bad node in heap\n" );
        break;
}

Classification: WATCOM

Systems:  _heapchk - All
          _bheapchk - DOS/16, Windows, QNX/16, OS/2 1.x(all)
          _fheapchk - DOS/16, Windows, QNX/16, OS/2 1.x(all)
          _nheapchk - DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2 1.x(MT), OS/2-32
Synopsis:    #include <malloc.h>
            int _heapenable( int enabled );

Description: The _heapenable function is used to control attempts by the heap allocation manager to request
more memory from the operating system’s memory pool. If enabled is 0 then all further allocations
which would normally go to the operating system for more memory will instead fail and return NULL.
If enabled is 1 then requests for more memory from the operating system’s memory pool are re-enabled.

This function can be used to impose a limit on the amount of system memory that is allocated by an
application. For example, if an application wishes to allocate no more than 200K bytes of memory, it
could allocate 200K and immediately free it. It can then call _heapenable to disable any further
requests from the system memory pool. After this, the application can allocate memory from the 200K
pool that it has already obtained.

Returns:    The return value is the previous state of the system allocation flag.

See Also:   _heapchk, _heapgrow, _heapmin, _heapset, _heapshrink, _heapwalk

Example:    #include <stdio.h>
            #include <malloc.h>

            void main()
            {
                char *p;
                p = malloc( 200*1024 );
                if( p != NULL ) free( p );
                _heapenable( 0 );
                /*
                   allocate memory from a pool that
                   has been capped at 200K
                */
            }

Classification: WATCOM

Systems:    All
Heapgrow Functions

Synopsis:
#include <malloc.h>
void _heapgrow( void );
void _nheapgrow( void );
void _fheapgrow( void );

Description:
The _nheapgrow function attempts to grow the near heap to the maximum size of 64K. You will want to do this in the small data models if you are using both malloc and _fmalloc or halloc. Once a call to _fmalloc or halloc has been made, you may not be able to allocate any memory with malloc unless space has been reserved for the near heap using either malloc, sbrk or _nheapgrow.

The _fheapgrow function doesn’t do anything to the heap because the far heap will be extended automatically when needed. If the current far heap cannot be extended, then another far heap will be started.

In a small data memory model, the _heapgrow function is equivalent to the _nheapgrow function; in a large data memory model, the _heapgrow function is equivalent to the _fheapgrow function.

Returns:
These functions do not return a value.

See Also:
_heapchk, _heapenable, _heapmin, _heapset, _heapshrink, _heapwalk

Example:
#include <stdio.h>
#include <malloc.h>

void main()
{
    char *p, *fmt_string;
    fmt_string = "Amount of memory available is %u\n";
    printf( fmt_string, _memavl() );
    _nheapgrow();
    printf( fmt_string, _memavl() );
    p = (char *) malloc( 2000 );
    printf( fmt_string, _memavl() );
}

produces the following:

Amount of memory available is 0
Amount of memory available is 62732
Amount of memory available is 60730

Classification: WATCOM

Systems:
_heapgrow - All
_fheapgrow - DOS/16, Windows, QNX/16, OS/2 1.x(all)
_nheapgrow - DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2 1.x (MT), OS/2-32
Synopsis:
#include <malloc.h>
int _heapmin( void );
int _bheapmin( __segment seg );
int _fheapmin( void );
int _nheapmin( void );

Description: The _heapmin functions attempt to shrink the specified heap to its smallest possible size by returning all free entries at the end of the heap back to the system. This can be used to free up as much memory as possible before using the system function or one of the spawn functions.

The various _heapmin functions shrink the following heaps:

<table>
<thead>
<tr>
<th>Function</th>
<th>Heap Minimized</th>
</tr>
</thead>
<tbody>
<tr>
<td>_heapmin</td>
<td>Depends on data model of the program</td>
</tr>
<tr>
<td>_bheapmin</td>
<td>Based heap specified by seg value; _NULLSEG specifies all based heaps</td>
</tr>
<tr>
<td>_fheapmin</td>
<td>Far heap (outside the default data segment)</td>
</tr>
<tr>
<td>_nheapmin</td>
<td>Near heap (inside the default data segment)</td>
</tr>
</tbody>
</table>

In a small data memory model, the _heapmin function is equivalent to the _nheapmin function; in a large data memory model, the _heapmin function is equivalent to the _fheapmin function. It is identical to the _heapshrink function.

Returns: These functions return zero if successful, and non-zero if some error occurred.

See Also: _heapchk, _heapenable, _heapgrow, _heapset, _heapshrink, _heapwalk

Example:
#include <stdlib.h>
#include <malloc.h>

void main()
{
    _heapmin();
    system( "chdir c:\watcomc" );
}

Note the use of two adjacent backslash characters (\) within character-string constants to signify a single backslash.

Classification: WATCOM

Systems:
_heapmin - All
_bheapmin - DOS/16, Windows, QNX/16, OS/2 1.x(all)
_fheapmin - DOS/16, Windows, QNX/16, OS/2 1.x(all)
_nheapmin - DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2 1.x(MT), OS/2-32
_heapset Functions

Synopsis:  
#include <malloc.h>

int _heapset( unsigned char fill_char );
int _bheapset( __segment seg, unsigned char fill_char );
int _fheapset( unsigned char fill_char );
int _nheapset( unsigned char fill_char );

Description:  The _heapset functions along with _heapchk and _heapwalk are provided for debugging heap related problems in programs.

The _heapset functions perform a consistency check on the unallocated memory space or "heap" just as _heapchk does, and sets the heap's free entries with the fill_char value.

Each function checks and sets a particular heap, as listed below:

<table>
<thead>
<tr>
<th>Function</th>
<th>Heap Filled</th>
</tr>
</thead>
<tbody>
<tr>
<td>_heapset</td>
<td>Depends on data model of the program</td>
</tr>
<tr>
<td>_bheapset</td>
<td>Based heap specified by seg value: _NULLSEG specifies all based heaps</td>
</tr>
<tr>
<td>_fheapset</td>
<td>Far heap (outside the default data segment)</td>
</tr>
<tr>
<td>_nheapset</td>
<td>Near heap (inside the default data segment)</td>
</tr>
</tbody>
</table>

In a small data memory model, the _heapset function is equivalent to the _nheapset function; in a large data memory model, the _heapset function is equivalent to the _fheapset function.

Returns:  The _heapset functions return one of the following manifest constants which are defined in <malloc.h>.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>_HEAPOK</td>
<td>The heap appears to be consistent.</td>
</tr>
<tr>
<td>_HEAPEMPTY</td>
<td>The heap is empty.</td>
</tr>
<tr>
<td>_HEAPBADBEGIN</td>
<td>The heap has been damaged.</td>
</tr>
<tr>
<td>_HEAPBADNODE</td>
<td>The heap contains a bad node, or is damaged.</td>
</tr>
</tbody>
</table>

See Also:  _heapchk, _heapenable, _heapgrow, _heapmin, _heapshrink, _heapwalk

Example:  
#include <stdio.h>
#include <malloc.h>

void main()
{
    int heap_status;
    char *buffer;
buffer = (char *)malloc( 80 );
malloc( 1024 );
free( buffer );
heap_status = _heapset( 0xff );
switch( heap_status ) {
case _HEAPOK:
    printf( "OK - heap is good\n" );
    break;
case _HEAPEMPTY:
    printf( "OK - heap is empty\n" );
    break;
case _HEAPBADBEGIN:
    printf( "ERROR - heap is damaged\n" );
    break;
case _HEAPBADNODE:
    printf( "ERROR - bad node in heap\n" );
    break;
}

Classification: WATCOM

Systems:
  _heapset - All
  _bheapset - DOS/16, Windows, QNX/16, OS/2 1.x(all)
  _fheapset - DOS/16, Windows, QNX/16, OS/2 1.x(all)
  _nheapset - DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2 1.x(MT),
               OS/2-32
Synopsis:  
#include <malloc.h>
int _heapshrink( void );
int _bheapshrink( __segment seg );
int _fheapshrink( void );
int _nheapshrink( void );

Description:  The _heapshrink functions attempt to shrink the heap to its smallest possible size by returning all free entries at the end of the heap back to the system. This can be used to free up as much memory as possible before using the system function or one of the spawn functions.

The various _heapshrink functions shrink the following heaps:

<table>
<thead>
<tr>
<th>Function</th>
<th>Heap Shrinked</th>
</tr>
</thead>
<tbody>
<tr>
<td>_heapshrink</td>
<td>Depends on data model of the program</td>
</tr>
<tr>
<td>_bheapshrink</td>
<td>Based heap specified by seg value; _NULLSEG specifies all based heaps</td>
</tr>
<tr>
<td>_fheapshrink</td>
<td>Far heap (outside the default data segment)</td>
</tr>
<tr>
<td>_nheapshrink</td>
<td>Near heap (inside the default data segment)</td>
</tr>
</tbody>
</table>

In a small data memory model, the _heapshrink function is equivalent to the _nheapshrink function; in a large data memory model, the _heapshrink function is equivalent to the _fheapshrink function. It is identical to the _heapmin function.

Returns:  These functions return zero if successful, and non-zero if some error occurred.

See Also:  _heapchk, _heapenable, _heapgrow, _heapmin, _heapset, _heapwalk

Example:  
#include <stdlib.h>
#include <malloc.h>

void main()
{
  _heapshrink();
  system( "chdir c:\watcomc" );
}

Note the use of two adjacent backslash characters (\) within character-string constants to signify a single backslash.

Classification:  WATCOM

Systems:  _heapshrink - All
          _bheapshrink - DOS/16, Windows, QNX/16, OS/2 1.x(all)
          _fheapshrink - DOS/16, Windows, QNX/16, OS/2 1.x(all)
          _nheapshrink - DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2 1.x (MT), OS/2-32
_heapwalk Functions

Synopsis:
#include <malloc.h>
int _heapwalk( struct _heapinfo *entry );
int _bheapwalk( __segment seg, struct _heapinfo *entry );
int _fheapwalk( struct _heapinfo *entry );
int _nheapwalk( struct _heapinfo *entry );

struct _heapinfo {
    void __far * _pentry;   /* heap pointer */
    size_t     _size;      /* heap entry size */
    int        _useflag;   /* heap entry 'in-use' flag */
};
#define _USEDENTRY      0
#define _FREEENTRY      1

Description: The _heapwalk functions along with _heapchk and _heapset are provided for debugging heap related problems in programs.

The _heapwalk functions walk through the heap, one entry per call, updating the _heapinfo structure with information on the next heap entry. The structure is defined in <malloc.h>. You must initialize the _pentry field with NULL to start the walk through the heap.

Each function walks a particular heap, as listed below:

<table>
<thead>
<tr>
<th>Function</th>
<th>Heap Walked</th>
</tr>
</thead>
<tbody>
<tr>
<td>_heapwalk</td>
<td>Depends on data model of the program</td>
</tr>
<tr>
<td>_bheapwalk</td>
<td>Based heap specified by seg value; _NULLSEG specifies all based heaps</td>
</tr>
<tr>
<td>_fheapwalk</td>
<td>Far heap (outside the default data segment)</td>
</tr>
<tr>
<td>_nheapwalk</td>
<td>Near heap (inside the default data segment)</td>
</tr>
</tbody>
</table>

In a small data memory model, the _heapwalk function is equivalent to the _nheapwalk function; in a large data memory model, the _heapwalk function is equivalent to the _fheapwalk function.

Returns: These functions return one of the following manifest constants which are defined in <malloc.h>.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>_HEAPOK</td>
<td>The heap is OK so far, and the _heapinfo structure contains information about the next entry in the heap.</td>
</tr>
<tr>
<td>_HEAPEMPTY</td>
<td>The heap is empty.</td>
</tr>
<tr>
<td>_HEAPBADPTR</td>
<td>The _pentry field of the entry structure does not contain a valid pointer into the heap.</td>
</tr>
<tr>
<td>_HEAPBADBEGIN</td>
<td>The header information for the heap was not found or has been damaged.</td>
</tr>
<tr>
<td>_HEAPBADNODE</td>
<td>The heap contains a bad node, or is damaged.</td>
</tr>
<tr>
<td>_HEAPEND</td>
<td>The end of the heap was reached successfully.</td>
</tr>
</tbody>
</table>
See Also: _heapchk, _heapenable, _heapgrow, _heapmin, _heapset, _heapshrink

Example:

```c
#include <stdio.h>
#include <malloc.h>

heap_dump()
{
    struct _heapinfo h_info;
    int heap_status;

    h_info._pentry = NULL;
    for(;;) {
        heap_status = _heapwalk( &h_info );
        if( heap_status != _HEAPOK ) break;
        printf( " %s block at %Fp of size %4.4X
",
            (h_info._useflag == _USEDENTRY ? "USED" : "FREE"),
            h_info._pentry, h_info._size );
    }

    switch( heap_status ) {
    case _HEAPEND:
        printf( "OK - end of heap\n" );
        break;
    case _HEAPEMPTY:
        printf( "OK - heap is empty\n" );
        break;
    case _HEAPBADBEGIN:
        printf( "ERROR - heap is damaged\n" );
        break;
    case _HEAPBADPTR:
        printf( "ERROR - bad pointer to heap\n" );
        break;
    case _HEAPBADNODE:
        printf( "ERROR - bad node in heap\n" );
    }
}

void main()
{
    char *p;
    heap_dump();   p = (char *) malloc( 80 );
    heap_dump();   free( p );
    heap_dump();
}
```

produces the following:

On 16-bit 80x86 systems, the following output is produced:

OK - heap is empty
    USED block at 23f8:0ab6 of size 0202
    USED block at 23f8:0cb8 of size 0052
    FREE block at 23f8:0d0a of size 1DA2
OK - end of heap
    USED block at 23f8:0ab6 of size 0202
    FREE block at 23f8:0cb8 of size 1DF4
OK - end of heap
On 32-bit 80386/486 systems, the following output is produced:

OK - heap is empty
USED block at 0014:00002a7c of size 0204
USED block at 0014:00002c80 of size 0054
FREE block at 0014:00002cd4 of size 1D98
OK - end of heap
USED block at 0014:00002a7c of size 0204
FREE block at 0014:00002c80 of size 1DEC
OK - end of heap

Classification: WATCOM

Systems:
_heapwalk - All
_bheapwalk - DOS/16, Windows, QNX/16, OS/2 1.x(all)
_fheapwalk - DOS/16, Windows, QNX/16, OS/2 1.x(all)
_nheapwalk - DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2 1.x(MT), OS/2-32
Synopsis:    #include <malloc.h>
              void hfree( void __huge *ptr );

Description:   The hfree function deallocates a memory block previously allocated by the halloc function. The argument ptr points to a memory block to be deallocated. After the call, the freed block is available for allocation.

Returns:   The hfree function returns no value.

See Also:    calloc Functions, _expand Functions, free Functions, halloc, malloc Functions, _msize Functions, realloc Functions, sbrk

Example:    #include <stdio.h>
             #include <malloc.h>
             void main()
             {
               long int __huge *big_buffer;
               big_buffer = (long int __huge *)
                             halloc( 1024L, sizeof(long) );
               if( big_buffer == NULL ) {
                 printf( "Unable to allocate memory\n" );
               } else {
                 /* rest of code goes here */
                 hfree( big_buffer ); /* deallocate */
               }
            }

Classification:   WATCOM

Systems:   DOS/16, Windows, QNX/16, OS/2 1.x(all)
Synopsis:  
#include <math.h>
double hypot( double x, double y );

Description: The hypot function computes the length of the hypotenuse of a right triangle whose sides are \( x \) and \( y \) adjacent to that right angle. The calculation is equivalent to

\[
\sqrt{x^2 + y^2}
\]

The computation may cause an overflow, in which case the matherr function will be invoked.

Returns: The value of the hypotenuse is returned. When an error has occurred, errno contains a value indicating the type of error that has been detected.

Example:  
#include <stdio.h>  
#include <math.h>

void main()
{
  printf( "\%f\n", hypot( 3.0, 4.0 ) );
}

produces the following:

5.000000

Classification: WATCOM

Systems: Math
 ignore_handler_s

Synopsis:  #define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
void ignore_handler_s(
    const char * restrict msg,
    void * restrict ptr,
    errno_t error );

Description: A pointer to the ignore_handler_s function may be passed as an argument to the
set_constraint_handler_s function. The ignore_handler_s function simply returns to
its caller.

Returns: The ignore_handler_s function does not return a value.

See Also: abort_handler_s, set_constraint_handler_s

Example: #define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
#include <stdio.h>

void main( void )
{
    constraint_handler_t old_handler;

    old_handler =
        set_constraint_handler_s( ignore_handler_s );
    if( getenv_s( NULL, NULL, 0, NULL ) ) {
        printf( "getenv_s failed\n" );
    }
    set_constraint_handler_s( old_handler );
}

produces the following:

getenv_s failed

Classification: TR 24731

Systems: All, Netware
Synopsis:  
#include <graph.h>
long _FAR _imagesize( short x1, short y1,
                       short x2, short y2 );

long _FAR _imagesize_w( double x1, double y1,
                         double x2, double y2 );

long _FAR _imagesize_wxy( struct _wxycoord _FAR *p1,
                          struct _wxycoord _FAR *p2 );

Description:  The _imagesize functions compute the number of bytes required to store a screen image. The _imagesize function uses the view coordinate system. The _imagesize_w and _imagesize_wxy functions use the window coordinate system.

The screen image is the rectangular area defined by the points (x1,y1) and (x2,y2). The storage area used by the _getimage functions must be at least this large (in bytes).

Returns:  The _imagesize functions return the size of a screen image.

See Also:  _getimage, _putimage

Example:  
#include <conio.h>
#include <graph.h>
#include <malloc.h>

main()
{
    char *buf;
    int y;

    _setvideomode( _VRES16COLOR );
    _ellipse( _GFILLINTERIOR, 100, 100, 200, 200 );
    buf = (char*) malloc( _imagesize( 100, 100, 201, 201 ) );
    if( buf != NULL ) {
        _getimage( 100, 100, 201, 201, buf );
        _putimage( 260, 200, buf, _GPSET );
        _putimage( 420, 100, buf, _GPSET );
        for( y = 100; y < 300; ) {
            _putimage( 420, y, buf, _GXOR );
            y += 20;
            _putimage( 420, y, buf, _GXOR );
        }
        free( buf );
    }
    getch();
    _setvideomode( _DEFAULTMODE );
}

Classification:  PC Graphics

Systems:  _imagesize - DOS, QNX
          _imagesize_w - DOS, QNX
          _imagesize_wxy - DOS, QNX
Synopsis:  
#include <inttypes.h>
intmax_t imaxabs( intmax_t j );

Description:  The `imaxabs` function returns the absolute value of its maximum-size integer argument `j`.

Returns:  The `imaxabs` function returns the absolute value of its argument.

See Also:  labs, llabs, abs, fabs

Example:  
#include <stdio.h>
#include <inttypes.h>

void main( void )
{
    intmax_t x, y;
    x = -500000000000;
    y = imaxabs( x );
    printf( "imaxabs(%jd) = %jd\n", x, y );
}

produces the following:

imaxabs(-500000000000) = 500000000000

Classification: ISO C99

Systems:  All, Netware
**Synopsis:**

```c
#include <stdlib.h>
imaxdiv_t imaxdiv( intmax_t numer, intmax_t denom );
```

typedef struct {
  intmax_t quot; /* quotient */
  intmax_t rem;  /* remainder */
} imaxdiv_t;

**Description:**
The `imaxdiv` function calculates the quotient and remainder of the division of the numerator `numer` by the denominator `denom`.

**Returns:**
The `imaxdiv` function returns a structure of type `imaxdiv_t` that contains the fields `quot` and `rem`, which are both of type `intmax_t`.

**See Also:**
`div`, `ldiv`, `lldiv`

**Example:**

```c
#include <stdio.h>
#include <inttypes.h>

void print_time( intmax_t ticks )
{
  imaxdiv_t sec_ticks;
  imaxdiv_t min_sec;

  sec_ticks = imaxdiv( ticks, 1000000 );
  min_sec   = imaxdiv( sec_ticks.quot, 60 );
  printf( "It took %jd minutes and %jd seconds\n",
          min_sec.quot, min_sec.rem );
}

void main( void )
{
  print_time( 9876543210 );
}
```

produces the following:

```
It took 164 minutes and 36 seconds
```

**Classification:**
ISO C99

**Systems:**
All, Netware
Synopsis:    #include <conio.h>
            unsigned int inp( int port );

Description: The inp function reads one byte from the 80x86 hardware port whose number is given by port.

A hardware port is used to communicate with a device. One or two bytes can be read and/or written from each port, depending upon the hardware. Consult the technical documentation for your computer to determine the port numbers for a device and the expected usage of each port for a device.

Returns:    The value returned is the byte that was read.

See Also:   inpd, inpw, outp, outpd, outpw

Example:    #include <conio.h>
            void main()
            {
                /* turn off speaker */
                outp( 0x61, inp( 0x61 ) & 0xFC );
            }

Classification: Intel

Systems:    All, Netware
Synopsis: #include <conio.h>
unsigned long inpd( int port );

Description: The inpd function reads a double-word (four bytes) from the 80x86 hardware port whose number is given by port.

A hardware port is used to communicate with a device. One or two bytes can be read and/or written from each port, depending upon the hardware. Consult the technical documentation for your computer to determine the port numbers for a device and the expected usage of each port for a device.

Returns: The value returned is the double-word that was read.

See Also: inp, inpw, outp, outpd, outpw

Example: #include <conio.h>
#define DEVICE 34

void main()
{
    unsigned long transmitted;

    transmitted = inpd( DEVICE );
}

Classification: Intel

Synopsis:  
#include <conio.h>
unsigned int inpw( int port );

Description:  
The inpw function reads a word (two bytes) from the 80x86 hardware port whose number is given by port.

A hardware port is used to communicate with a device. One or two bytes can be read and/or written from each port, depending upon the hardware. Consult the technical documentation for your computer to determine the port numbers for a device and the expected usage of each port for a device.

Returns:  
The value returned is the word that was read.

See Also:  
inp, inpd, outp, outpd, outpw

Example:  
#include <conio.h>
#define DEVICE 34

void main()
{
    unsigned int transmitted;

    transmitted = inpw( DEVICE );
}

Classification: Intel

Systems: All, Netware
Synopsis:

```c
#include <i86.h>

int int386( int inter_no,
            const union REGS *in_regs,
            union REGS *out_regs );
```

Description: The `int386` function causes the computer’s central processor (CPU) to be interrupted with an interrupt whose number is given by `inter_no`. This function is present in the 386 C libraries and may be executed on 80386/486 systems. Before the interrupt, the CPU registers are loaded from the structure located by `in_regs`. Following the interrupt, the structure located by `out_regs` is filled with the contents of the CPU registers. These structures may be located at the same location in memory.

You should consult the technical documentation for the computer that you are using to determine the expected register contents before and after the interrupt in question.

Returns: The `int386` function returns the value of the CPU EAX register after the interrupt.

See Also: bdos, int386x, int86, int86x, intdos, intdosx, intr, segread

Example:

```c
/*
 * This example clears the screen on DOS
 */
#include <i86.h>

void main()
{
    union REGS  regs;
    
    regs.w.cx = 0;
    regs.w.dx = 0x1850;
    regs.h.bh = 7;
    regs.w.ax = 0x0600;
    #if defined(__386__) && defined(__DOS__)
        int386( 0x10, &regs, &regs );
    #else
        int86( 0x10, &regs, &regs );
    #endif
}
```

Classification: Intel

Systems: DOS/32, QNX/32, Netware
Synopsis:  
#include <i86.h>
int int386x( int inter_no,
    const union REGS *in_regs,
    union REGS *out_regs,
    struct SREGS *seg_regs );

Description:  The int386x function causes the computer’s central processor (CPU) to be interrupted with an interrupt whose number is given by inter_no. This function is present in the 32-bit C libraries and may be executed on Intel 386 compatible systems. Before the interrupt, the CPU registers are loaded from the structure located by in_regs and the DS, ES, FS and GS segment registers are loaded from the structure located by seg_regs. All of the segment registers must contain valid values. Failure to do so will cause a segment violation when running in protect mode. If you don’t care about a particular segment register, then it can be set to 0 which will not cause a segment violation. The function segread can be used to initialize seg_regs to their current values.

Following the interrupt, the structure located by out_regs is filled with the contents of the CPU registers. The in_regs and out_regs structures may be located at the same location in memory. The original values of the DS, ES, FS and GS registers are restored. The structure seg_regs is updated with the values of the segment registers following the interrupt.

You should consult the technical documentation for the computer that you are using to determine the expected register contents before and after the interrupt in question.

Returns:  The int386x function returns the value of the CPU EAX register after the interrupt.

See Also:  bdos, int386, int86, int86x, intdos, intdosx, intr, segread

Example:  
#include <stdio.h>
#include <i86.h>

/* get current mouse interrupt handler address */

void main()
{
    union REGS r;
    struct SREGS s;

    s.ds = s.es = s.fs = s.gs = FP_SEG( &s );

    #if defined(__PHARLAP__)
        r.w.ax = 0x2503;    /* get real-mode vector */
        r.h.cl = 0x33;      /* interrupt vector 0x33 */
        int386( 0x21, &r, &r );
        printf( "mouse handler real-mode address=
                "\%lx\n", r.x.ebx );
        r.w.ax = 0x2502;    /* get protected-mode vector */
        r.h.cl = 0x33;      /* interrupt vector 0x33 */
        int386x( 0x21, &r, &r, &s );
        printf( "mouse handler protected-mode address="
                "\%x:\%lx\n", s.es, r.x.ebx );
    #endif
}
#else
    r.h.ah = 0x35; /* get vector */
    r.h.al = 0x33; /* vector 0x33 */
    int386x( 0x21, &r, &r, &s );
    printf( "mouse handler protected-mode address="
            "%lx:%lx\n", s.es, r.x.ebx );
#endif

Classification: Intel

Systems: DOS/32, QNX/32, Netware
Synopsis:  
#include <i86.h>
int int86( int inter_no,
    const union REGS *in_regs,
    union REGS *out_regs );

Description:  
The int86 function causes the computer’s central processor (CPU) to be interrupted with an interrupt
whose number is given by inter_no. Before the interrupt, the CPU registers are loaded from the
structure located by in_regs. Following the interrupt, the structure located by out_regs is filled with the
contents of the CPU registers. These structures may be located at the same location in memory.
You should consult the technical documentation for the computer that you are using to determine the
expected register contents before and after the interrupt in question.

Returns:  
The int86 function returns the value of the CPU AX register after the interrupt.

See Also:  
bdos, int386, int386x, int86x, intdos, intdosx, intr, segread

Example:  
/*
 * This example clears the screen on DOS
 */
#include <i86.h>
void main()
{
    union REGS  regs;
    regs.w.cx = 0;
    regs.w.dx = 0x1850;
    regs.h.bh = 7;
    regs.w.ax = 0x0600;
    #if defined(__386__) && defined(__DOS__)
        int386( 0x10, &regs, &regs );
    #else
        int86( 0x10, &regs, &regs );
    #endif
}

Classification:  Intel

Systems:  DOS/16, Windows, Win386, QNX/16, DOS/PM
#include <i86.h>

int int86x( int inter_no,
            const union REGS *in_regs,
            union REGS *out_regs,
            struct SREGS *seg_regs );

The int86x function causes the computer's central processor (CPU) to be interrupted with an interrupt whose number is given by inter_no. Before the interrupt, the CPU registers are loaded from the structure located by in_regs and the DS and ES segment registers are loaded from the structure located by seg_regs. All of the segment registers must contain valid values. Failure to do so will cause a segment violation when running in protect mode. If you don’t care about a particular segment register, then it can be set to 0 which will not cause a segment violation. The function segread can be used to initialize seg_regs to their current values.

Following the interrupt, the structure located by out_regs is filled with the contents of the CPU registers. The in_regs and out_regs structures may be located at the same location in memory. The original values of the DS and ES registers are restored. The structure seg_regs is updated with the values of the segment registers following the interrupt.

You should consult the technical documentation for the computer that you are using to determine the expected register contents before and after the interrupt in question.

The function returns the value of the CPU AX register after the interrupt.

See Also: bdos, int386, int386x, int86, intdos, intdosx, intr, segread

Example:
#include <stdio.h>
#include <i86.h>

/* get current mouse interrupt handler address */

void main()
{
    union REGS r;
    struct SREGS s;

    r.h.ah = 0x35; /* DOS get vector */
    r.h.al = 0x33; /* interrupt vector 0x33 */
    int86x( 0x21, &r, &r, &s );
    printf( "mouse handler address=%4.4x:%4.4x\n",
            s.es, r.w.bx );
}

Classification: Intel

Systems: DOS/16, Windows, Win386, QNX/16, DOS/PM
Synopsis:  
#include <dos.h>

int intdos( const union REGS *in_regs,
            union REGS *out_regs );

Description:  The intdos function causes the computer’s central processor (CPU) to be interrupted with an interrupt number hexadecimal 21 (0x21), which is a request to invoke a specific DOS function. Before the interrupt, the CPU registers are loaded from the structure located by in_regs. The AH register contains a number indicating the function requested. Following the interrupt, the structure located by out_regs is filled with the contents of the CPU registers. These structures may be located at the same location in memory.

You should consult the technical documentation for the DOS operating system that you are using to determine the expected register contents before and after the interrupt in question.

Returns:  The function returns the value of the AX (EAX in 386 library) register after the interrupt has completed. The CARRY flag (when set, an error has occurred) is copied into the structure located by out_regs. When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also:  bdos, int386, int386x, int86, int86x, intdosx, intr, segread

Example:  
#include <dos.h>

#define DISPLAY_OUTPUT  2

void main()
{
    union REGS  in_regs, out_regs;
    int         rc;

    in_regs.h.ah = DISPLAY_OUTPUT;
    in_regs.h.al = 0;

    in_regs.w.dx = 'I';
    rc = intdos( &in_regs, &out_regs );
    in_regs.w.dx = 'N';
    rc = intdos( &in_regs, &out_regs );
    in_regs.w.dx = 'T';
    rc = intdos( &in_regs, &out_regs );
    in_regs.w.dx = 'D';
    rc = intdos( &in_regs, &out_regs );
    in_regs.w.dx = 'O';
    rc = intdos( &in_regs, &out_regs );
    in_regs.w.dx = 'S';
    rc = intdos( &in_regs, &out_regs );
}

Classification:  DOS

Systems:  DOS, Windows, Win386, DOS/PM
Synopsis: 
#include <dos.h>
int intdosx( const union REGS *in_regs,
union REGS *out_regs,
struct SREGS *seg_regs );

Description: The intdosx function causes the computer’s central processor (CPU) to be interrupted with an interrupt number hexadecimal 21 (0x21), which is a request to invoke a specific DOS function. Before the interrupt, the CPU registers are loaded from the structure located by in_regs and the segment registers DS and ES are loaded from the structure located by seg_regs. The AH register contains a number indicating the function requested. All of the segment registers must contain valid values. Failure to do so will cause a segment violation when running in protect mode. If you don’t care about a particular segment register, then it can be set to 0 which will not cause a segment violation. The function segread can be used to initialize seg_regs to their current values.

Following the interrupt, the structure located by out_regs is filled with the contents of the CPU registers. The in_regs and out_regs structures may be located at the same location in memory. The original values for the DS and ES registers are restored. The structure seg_regs is updated with the values of the segment registers following the interrupt.

You should consult the technical documentation for the DOS operating system that you are using to determine the expected register contents before and after the interrupt in question.

Returns: The intdosx function returns the value of the AX (EAX in 32-bit library) register after the interrupt has completed. The CARRY flag (when set, an error has occurred) is copied into the structure located by out_regs. When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also: bdos, int386, int386x, int86, int86x, intdos, intr, segread

Example: 
#include <stdio.h>
#include <dos.h>

/* get current mouse interrupt handler address */

void main()
{
    union REGS r;
    struct SREGS s;

#if defined(__386__)
    s.ds = s.es = s.fs = s.gs = FP_SEG( &s );
#endif
    r.h.ah = 0x35; /* get vector */
    r.h.al = 0x33; /* vector 0x33 */
    intdosx( &r, &r, &s );
#if defined(__386__)
    printf( "mouse handler address=%4.4x:%lx\n",
            s.es, r.x.ebx );
#else
    printf( "mouse handler address=%4.4x:%4.4x\n",
            s.es, r.x.bx );
#endif
}

408 Library Functions and Macros
Classification: DOS

Systems: DOS, Windows, Win386, DOS/PM
Synopsis: 
```
#include <i86.h>
void intr( int inter_no, union REGPACK *regs );
```

Description: The `intr` function causes the computer’s central processor (CPU) to be interrupted with an interrupt whose number is given by `inter_no`. Before the interrupt, the CPU registers are loaded from the structure located by `regs`. All of the segment registers must contain valid values. Failure to do so will cause a segment violation when running in protect mode. If you don’t care about a particular segment register, then it can be set to 0 which will not cause a segment violation. Following the interrupt, the structure located by `regs` is filled with the contents of the CPU registers.

This function is similar to the `int86x` function, except that only one structure is used for the register values and that the BP (EBP in 386 library) register is included in the set of registers that are passed and saved.

You should consult the technical documentation for the computer that you are using to determine the expected register contents before and after the interrupt in question.

Returns: The `intr` function does not return a value.

See Also: `bdos`, `int386`, `int386x`, `int86`, `int86x`, `intdos`, `intdosx`, `segread`

Example: 
```
#include <stdio.h>
#include <string.h>
#include <i86.h>

void main() /* Print location of Break Key Vector */ {
    union REGPACK regs;
    memset( &regs, 0, sizeof(union REGPACK) );
    regs.w.ax = 0x3523;
    intr( 0x21, &regs );
    printf( "Break Key vector is 
#if defined(__386__)" "%x:%lx\n", regs.w.es, regs.x.ebx );
#else
    "%x:%x\n", regs.w.es, regs.x.bx );
#endif
}
```

produces the following:

```
Break Key vector is eef:13c
```

Classification: Intel

Systems: DOS, Windows, Win386, QNX, DOS/PM, Netware

410  Library Functions and Macros
Synopsis:  
#include <ctype.h>  
int isalnum( int c );  
#include <wctype.h>  
int iswalnum( wint_t c );

Description:  The isalnum function tests if the argument c is an alphanumeric character ('a' to 'z', 'A' to 'Z', or '0' to '9'). An alphanumeric character is any character for which isalpha or isdigit is true.

The iswalnum function is similar to isalnum except that it accepts a wide-character argument.

Returns:  The isalnum function returns zero if the argument is neither an alphabetic character (A-Z or a-z) nor a digit (0-9). Otherwise, a non-zero value is returned. The iswalnum function returns a non-zero value if either iswalpha or iswdigit is true for c.

See Also:  isalpha, isblank, iscntrl, isdigit, isgraph, isleadbyte, islower, isprint, ispunct, isspace, isupper, iswctype, isxdigit, tolower, toupper, towctrans

Example:  
#include <stdio.h>  
#include <ctype.h>  

void main()
{
    if( isalnum( getchar() ) ) {
        printf( "is alpha-numeric\n" );
    }
}

Classification:  isalnum is ANSI  
isalnum is ANSI  
isalnum - All, Netware  
isalnum - All, Netware
Synopsis:  
```c
#include <ctype.h>
int isalpha( int c );
#include <wctype.h>
int iswalpha( wint_t c );
```

Description: The `isalpha` function tests if the argument `c` is an alphabetic character (’a’ to ’z’ and ’A’ to ’Z’). An alphabetic character is any character for which `isupper` or `islower` is true.

The `iswalpha` function is similar to `isalpha` except that it accepts a wide-character argument.

Returns: The `isalpha` function returns zero if the argument is not an alphabetic character (A-Z or a-z); otherwise, a non-zero value is returned. The `iswalpha` function returns a non-zero value only for wide characters for which `iswupper` or `iswlower` is true, or any wide character that is one of an implementation-defined set for which none of `iswcntrl`, `iswdigit`, `iswpunct`, or `iswspace` is true.

See Also: isalnum, isblank, iscntrl, isdigit, isgraph, isleadbyte, islower, isprint, ispunct, isspace, isupper, iswctype, isxdigit, tolower, toupper, towctrans

Example: 
```c
#include <stdio.h>
#include <ctype.h>

void main()
{
    if( isalpha( getchar() ) ) {
        printf( "is alphabetic\n" );
    }
}
```

Classification: isalpha is ANSI  
iswalpha is ANSI

Systems:  
isalpha - All, Netware  
iswalpha - All, Netware
Synopsis:  
#include <ctype.h>
int isascii( int c );
int __isascii( int c );
#include <wctype.h>
int iswascii( wint_t c );

Description:  The isascii function tests for a character in the range from 0 to 127.

The __isascii function is identical to isascii. Use __isascii for ANSI/ISO naming conventions.

The iswascii function is similar to isascii except that it accepts a wide-character argument.

Returns:  The isascii function returns a non-zero value when the character is in the range 0 to 127; otherwise, zero is returned. The iswascii function returns a non-zero value when c is a wide-character representation of an ASCII character.

See Also:  isalnum, isalpha, isblank, iscntrl, isdigit, isgraph, isleadbyte, islower, isprint, ispunct, isspace, isupper, iswctype, isxdigit, tolower, toupper, towctrans

Example:  
#include <stdio.h>
#include <ctype.h>

char chars[] = {
   'A',
   0x80,
   'Z'
};

#define SIZE sizeof( chars ) / sizeof( char )

void main()
{
   int   i;
   for( i = 0; i < SIZE; i++ ) {
      printf( "Char %c is %san ASCII character\n", chars[i],
               ( isascii( chars[i] ) ) ? "" : "not " );
   }
}

produces the following:

Char A is an ASCII character
Char is not an ASCII character
Char Z is an ASCII character

Classification:  WATCOM
   __isascii conforms to ANSI/ISO naming conventions

Systems:  isascii - All, Netware
   __isascii - All, Netware
isascii, __isascii, iswascii

iswascii - All, Netware
Synopsis:  
#include <io.h>  
int isatty( int handle );  
int _isatty( int handle );

Description:  The isatty function tests if the opened file or device referenced by the file handle handle is a character device (for example, a console, printer or port).

The _isatty function is identical to isatty. Use _isatty for ANSI/ISO naming conventions.

Returns:  The isatty function returns zero if the device or file is not a character device; otherwise, a non-zero value is returned. When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also:  open

Example:  
#include <stdio.h>  
#include <io.h>  

void main( void )  
{  
    printf( "stdin is a %s\n",  
            ( isatty( fileno( stdin ) ) )  
            ? "" : "not " );  
}

Classification:  isatty is POSIX 1003.1
_isatty is not POSIX
_isatty conforms to ANSI/ISO naming conventions

Systems:  isatty - All, Netware
_isatty - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
isblank, iswblank

Synopsis:  
#include <ctype.h>
int isblank( int c );
#include <wctype.h>
int iswblank( wint_t c );

Description:  The isblank function tests for the following blank characters:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>' '</td>
<td>space</td>
</tr>
<tr>
<td>'\t'</td>
<td>horizontal tab</td>
</tr>
</tbody>
</table>

The iswblank function is similar to isblank except that it accepts a wide-character argument.

Returns:  The isblank function returns a non-zero character when the argument is one of the indicated blank characters. The iswblank function returns a non-zero value when the argument is a wide character that corresponds to a standard blank character or is one of an implementation-defined set of wide characters for which iswalnum is false. Otherwise, zero is returned.

See Also:  isalnum, isalpha, iscntr1, isdigit, isgraph, isleadbyte, islower, isprint, ispunct, isspace, isupper, iswctype, isxdigit, tolower, toupper, towctrans

Example:  
#include <stdio.h>
#include <ctype.h>

char chars[] = {
    'A',
    0x09,
    ' ',
    0x7d
};

#define SIZE sizeof( chars ) / sizeof( char )

void main()
{
    int i;

    for( i = 0; i < SIZE; i++ )
    {
        printf( "Char %s is %sa blank character\n",
            chars[i],
            isblank( chars[i] ) ? "" : "not " );
    }
}

produces the following:

Char A is not a blank character
Char   is a blank character
Char   is a blank character
Char ) is not a blank character

Classification:  isblank is ANSI
iswblank is ANSI
Systems:
isblank - All, Netware
iswblank - All, Netware
Synopsis:    #include <ctype.h>
    int iscntrl( int c );
    #include <wchar.h>
    int iswcntrl( wint_t c );

Description: The iscntrl function tests for any control character. A control character is any character whose
              value is from 0 through 31.

        The iswcntrl function is similar to iscntrl except that it accepts a wide-character argument.

Returns:    The iscntrl function returns a non-zero value when the argument is a control character. The
              iswcntrl function returns a non-zero value when the argument is a control wide character.
              Otherwise, zero is returned.

See Also:  isalnum, isalpha, isblank, isdigit, isgraph, isleadbyte, islower, isprint, ispunct, ispace, isupper, iswctype, isxdigit, tolower, toupper, towctrans

Example:    #include <stdio.h>
    #include <ctype.h>

    char chars[] = { 'A', 0x09, 'Z' };

    #define SIZE sizeof( chars ) / sizeof( char )

    void main()
    { int i;
      for( i = 0; i < SIZE; i++ ) {
        printf( "Char %c is %sa Control character\n",
                 chars[i],
                 ( iscntrl( chars[i] ) ) ? "" : "not " );
      }

    }

produces the following:

    Char A is not a Control character
    Char     is a Control character
    Char Z is not a Control character

Classification: iscntrl is ANSI
                 iswcntrl is ANSI

Systems:     iscntrl - All, Netware
             iswcntrl - All, Netware
Synopsis:
#include <ctype.h>
int iscsym( int c );
int __iscsym( int c );
#include <wctype.h>
int __iswcsym( wint_t c );

Description:
The **iscsym** function tests for a letter, underscore or digit.

The **__iscsym** function is identical to **iscsym**. Use **__iscsym** for ANSI/ISO naming conventions.

The **__iswcsym** function is similar to **iscsym** except that it accepts a wide-character argument.

Returns:
A non-zero value is returned when the character is a letter, underscore or digit; otherwise, zero is returned. The **__iswcsym** function returns a non-zero value when **c** is a wide-character representation of a letter, underscore or digit character.

See Also:

Example:
#include <stdio.h>
#include <ctype.h>

char chars[] = {
    'A',
    0x80,
    '_',
    '9',
    '+'
};
#define SIZE sizeof( chars ) / sizeof( char )

void main()
{
    int i;

    for( i = 0; i < SIZE; i++ ) {
        printf( "Char %c is %sa C symbol character\n",
                chars[i],
                ( __iscsym( chars[i] ) ) ? "" : "not " ;
        }
    }

produces the following:

Char A is a C symbol character
Char   is not a C symbol character
Char _ is a C symbol character
Char 9 is a C symbol character
Char + is not a C symbol character

Classification: WATCOM

**__iscsym** conforms to ANSI/ISO naming conventions
iscsym, __iscsym, __iswcsym

Systems:
iscsym - All, Netware
__iscsym - All, Netware
__iswcsym - All, Netware
Synopsis:  
#include <ctype.h>
int iscsymf( int c );
int __iscsymf( int c );
#include <wctype.h>
int __iswcsymf( wint_t c );

Description:  The iscsymf function tests for a letter or underscore.

The __iscsymf function is identical to iscsymf. Use __iscsymf for ANSI/ISO naming conventions.

The __iswcsymf function is similar to iscsymf except that it accepts a wide-character argument.

Returns:  A non-zero value is returned when the character is a letter or underscore; otherwise, zero is returned.
The __iswcsymf function returns a non-zero value when c is a wide-character representation of a letter or underscore character.

See Also:  isalpha, isalnum, iscntrl, isdigit, isgraph, islower, isprint, ispunct, isspace, isupper, isxdigit, tolower, toupper

Example:  
#include <stdio.h>
#include <ctype.h>

char chars[] = {
    'A',
    0x80,
    '_'
};

#define SIZE sizeof( chars ) / sizeof( char )

void main()
{
    int i;
    for( i = 0; i < SIZE; i++ ) {
        printf( "Char %c is %s csymf character\n",
            chars[i],
            ( __iscsymf( chars[i] ) ? "" : "not " );
    }
}

produces the following:

Char A is a csymf character
Char is not a csymf character
Char _ is a csymf character
Char 9 is not a csymf character
Char + is not a csymf character

Classification:  WATCOM
__iscsymf conforms to ANSI/ISO naming conventions
iscsymf, __iscsymf, __iswcsymf

Systems:

iscsymf - All, Netware
__iscsymf - All, Netware
__iswcsymf - All, Netware
Synopsis:  
#include <ctype.h>
int isdigit( int c );
#include <wctype.h>
int iswdigit( wint_t c );

Description:  The isdigit function tests for any decimal-digit character '0' through '9'.

The iswdigit function is similar to isdigit except that it accepts a wide-character argument.

Returns:  The isdigit function returns a non-zero value when the argument is a decimal-digit character. The iswdigit function returns a non-zero value when the argument is a wide character corresponding to a decimal-digit character. Otherwise, zero is returned.

See Also:  isalnum, isalpha, isblank, iscntrl, isgraph, isleadbyte, islower, isprint, ispunct, isspace, isupper, iswctype, isxdigit, tolower, toupper, towctrans

Example:  
#include <stdio.h>
#include <ctype.h>

char chars[] = {
    'A',
    '5',
    '$'
};

#define SIZE sizeof( chars ) / sizeof( char )

void main()
{
    int i;

    for( i = 0; i < SIZE; i++ ) {
        printf( "Char %c is %sa digit character\n",
                chars[i],
                ( isdigit( chars[i] ) ) ? "" : "not " );
    }
}

produces the following:

Char A is not a digit character
Char 5 is a digit character
Char $ is not a digit character

Classification:  isdigit is ANSI
               iswdigit is ANSI

Systems:  isdigit - All, Netware
           iswdigit - All, Netware
isfinite

Synopsis:  
#include <math.h>
int isfinite( x );

Description:  The isfinite macro determines whether its argument x has a finite value (zero, subnormal, or normal, and not infinite or NaN). First, an argument represented in a format wider than its semantic type is converted to its semantic type. Then determination is based on the type of the argument.

The argument x must be an expression of real floating type.

Returns:  The isfinite macro returns a nonzero value if and only if its argument has a finite value.

See Also:  fpclassify, isinf, isnan, isnormal, signbit

Example:  #include <math.h>
#include <stdio.h>

    void main( void )
    {
        printf( "zero %s a finite number\n",
            isfinite( 0.0 ) ? "is" : "is not" );
    }

produces the following:

    zero is a finite number

Classification:  ANSI

Systems:  MACRO

424  Library Functions and Macros
Synopsis:

```c
#include <ctype.h>
int isgraph( int c );
#include <wctype.h>
int iswgraph( wint_t c );
```

Description:
The `isgraph` function tests for any printable character except space (‘ ’). The `isprint` function is similar, except that the space character is also included in the character set being tested.

The `iswgraph` function is similar to `isgraph` except that it accepts a wide-character argument.

Returns:
The `isgraph` function returns non-zero when the argument is a printable character (except a space). The `iswgraph` function returns a non-zero value when the argument is a printable wide character (except a wide-character space). Otherwise, zero is returned.

See Also:
`isalnum`, `isalpha`, `isblank`, `iscntrl`, `isdigit`, `isleadbyte`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `toupper`, `towctrans`

Example:

```c
#include <stdio.h>
#include <ctype.h>

char chars[] = {
  'A',
  0x09,
  '
',
  0x7d
};

#define SIZE sizeof( chars ) / sizeof( char )

void main()
{
  int   i;
  for( i = 0; i < SIZE; i++ ) {
    printf( "Char %c is %sa printable character\n",
    chars[i],
    ( isgraph( chars[i] ) ) ? "" : "not " );
  }
}
```

produces the following:

```
Char A is a printable character
Char     is not a printable character
Char   is not a printable character
Char } is a printable character
```

Classification: `isgraph` is ANSI
   `iswgraph` is ANSI

Systems:
   `isgraph` - All, Netware
   `iswgraph` - All, Netware
isinf

Synopsis:    #include <math.h>
            int isinf( x );

Description: The isinf macro determines whether its argument value is an infinity (positive or negative). First, an argument represented in a format wider than its semantic type is converted to its semantic type. Then determination is based on the type of the argument.

            The argument x must be an expression of real floating type.

Returns:    The isinf macro returns a nonzero value if and only if its argument has an infinite value.

See Also:   fpclassify, isfinite, isnan, isnormal, signbit

Example:    #include <math.h>
            #include <stdio.h>

            void main( void )
            {
                printf( "zero %s an infinite number\n",
                        isinf( 0.0 ) ? "is" : "is not" );
            }

            produces the following:

            zero is not an infinite number

Classification: ANSI

Systems:    MACRO
Synopsis:
#include <ctype.h>
int isleadbyte( int ch )

Description: The isleadbyte function tests if the argument \emph{ch} is a valid first byte of a multibyte character in the current code page.

For example, in code page 932, a valid lead byte is any byte in the range 0x81 through 0x9F or 0xE0 through 0xFC.

Returns: The isleadbyte function returns a non-zero value when the argument is a valid lead byte. Otherwise, zero is returned.

See Also: isalnum, isalpha, isblank, iscntrl, isdigit, isgraph, islower, isprint, ispunct, isspace, isupper, iswctype, isxdigit, tolower, toupper, towctrans

Example:
#include <stdio.h>
#include <ctype.h>
#include <mbctype.h>

const unsigned char chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x81,0x40, /* double-byte space */
    0x82,0x60, /* double-byte A */
    0x82,0xA6, /* double-byte Hiragana */
    0x83,0x42, /* double-byte Katakana */
    0xA1, /* single-byte Katakana punctuation */
    0xA6, /* single-byte Katakana alphabetic */
    0xDF, /* single-byte Katakana alphabetic */
    0xE0,0xA1, /* double-byte Kanji */
    0x00
};

#define SIZE sizeof( chars ) / sizeof( char )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%2.2x is %sa valid lead byte\n",
            chars[i],
            ( isleadbyte( chars[i] ) ? "" : "not " );
    }
}

produces the following:
isleadbyte

20 is not a valid lead byte
2e is not a valid lead byte
31 is not a valid lead byte
41 is not a valid lead byte
81 is a valid lead byte
40 is not a valid lead byte
82 is a valid lead byte
60 is not a valid lead byte
82 is a valid lead byte
a6 is not a valid lead byte
83 is a valid lead byte
42 is not a valid lead byte
a1 is not a valid lead byte
a6 is not a valid lead byte
df is not a valid lead byte
e0 is a valid lead byte
a1 is not a valid lead byte
00 is not a valid lead byte

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <ctype.h>
int islower( int c );
#include <wctype.h>
int iswlower( wint_t c );

Description:  The islower function tests for any lowercase letter 'a' through 'z'.

The iswlower function is similar to islower except that it accepts a wide-character argument.

Returns:  The islower function returns a non-zero value when argument is a lowercase letter. The iswlower function returns a non-zero value when the argument is a wide character that corresponds to a lowercase letter, or if it is one of an implementation-defined set of wide characters for which none of iswcntrl, iswdigit, iswpunct, or iswspace is true. Otherwise, zero is returned.

See Also:  isalnum, isalpha, isblank, iscntrl, isdigit, isgraph, isleadbyte, isprint, ispunct, isspace, isupper, iswctype, isxdigit, tolower, toupper, towctrans

Example:  
#include <stdio.h>
#include <ctype.h>

char chars[] = 
    { 
    'A',
    'a',
    'z',
    'Z'
    };

#define SIZE sizeof( chars ) / sizeof( char )

void main()
{
    int i;

    for( i = 0; i < SIZE; i++ ) {
        printf( "Char %c is %sa lowercase character\n", 
                chars[i],
                ( islower( chars[i] ) ) ? "" : "not " );
    }
}

produces the following:

Char A is not a lowercase character
Char a is a lowercase character
Char z is a lowercase character
Char Z is not a lowercase character

Classification:  islower is ANSI
                iswlower is ANSI

Systems:  islower - All, Netware
           iswlower - All, Netware
Synopsis:  

```
#include <mbctype.h>
int _ismbbalnum( unsigned int ch );
```

Description: 

The `_ismbbalnum` function tests if the argument `ch` satisfies the condition that one of `isalnum` or `_ismbbkalnum` is true.

**Note:** The argument `ch` must represent a single-byte value (i.e., `0 <= ch <= 255`). Incorrect results occur if the argument is a double-byte character.

Returns: 

The `_ismbbalnum` function returns a non-zero value if the argument satisfies the condition; otherwise, a zero value is returned.

See Also: 

`_getmbcp`, `_mbbtombc`, `_mbcjjstojms`, `_mbcjstojis`, `_mbctombb`, `_ismbbalpha`, `_ismbbgraph`, `_ismbbkalnum`, `_ismbbkalpha`, `_ismbbkana`, `_ismbbkprint`, `_ismbbkpunct`, `_ismbblead`, `_ismbbprint`, `_ismbbpunct`, `_ismbbtrail`, `_mbbtombc`, `_mbcjjjstojms`, `_mbcjstojis`, `_mbctombb`, `_mbbtype`, `_setmbcp`

Example: 

```
#include <stdio.h>
#include <mbctype.h>

unsigned int chars[] = {
    ' ',
    '.',
    ',',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6,   /* single-byte Katakana alphabetic */
    0xDF,   /* single-byte Katakana alphabetic */
    0xE0A1  /* double-byte Kanji */
};

define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%#6.4x is %sa single-byte alphanumeric\n"  
            " or Katakana non-punctuation character\n", 
            chars[i],
            ( _ismbbalnum( chars[i] ) ? "" : "not " );
    }
}
```

produces the following:
0x0020 is not a single-byte alphanumeric or Katakana non-punctuation character
0x002e is not a single-byte alphanumeric or Katakana non-punctuation character
0x0031 is a single-byte alphanumeric or Katakana non-punctuation character
0x0041 is a single-byte alphanumeric or Katakana non-punctuation character
0x8140 is not a single-byte alphanumeric or Katakana non-punctuation character
0x8260 is not a single-byte alphanumeric or Katakana non-punctuation character
0x82a6 is a single-byte alphanumeric or Katakana non-punctuation character
0x8342 is a single-byte alphanumeric or Katakana non-punctuation character
0x00a1 is not a single-byte alphanumeric or Katakana non-punctuation character
0x00a6 is a single-byte alphanumeric or Katakana non-punctuation character
0x00df is a single-byte alphanumeric or Katakana non-punctuation character
0xe0a1 is not a single-byte alphanumeric or Katakana non-punctuation character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis: 
#include <mbctype.h>
int _ismbbalpha( unsigned int ch );

Description: The _ismbbalpha function tests if the argument ch satisfies the condition that one of isalpha or _ismbbkalpha is true.

For example, in code page 932, _ismbbalpha tests if the argument ch is a single-byte alphabetic character ("a" to "z" or "A" to "Z") or single-byte Katakana non-punctuation character.

Note: The argument ch must represent a single-byte value (i.e., 0 <= ch <= 255). Incorrect results occur if the argument is a double-byte character.

Returns: The _ismbbalpha function returns a non-zero value if the argument satisfies the condition; otherwise a zero value is returned.

See Also: _getmbcp, _mbbtombc, _mbcjistojms, _mbctombb, _ismbbalnum, _ismbbgraph, _ismbbkalnum, _ismbbkalpha, _ismbbkprint, _ismbbkpunct, _ismbblead, _ismbbprint, _ismbbpunct, _ismbbtrail, _mbbtombc, _mbcjistojms, _mbcjistojms, _mbctombb, _mbctype, _setmbcp

Example: 
#include <stdio.h>
#include <mbctype.h>

unsigned int chars[] = {
    '\n', '\n', '\n', '1', 'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1, /* single-byte Katakana punctuation */
    0xA6, /* single-byte Katakana alphabetic */
    0xDF, /* single-byte Katakana alphabetic */
    0xE0A1 /* double-byte Kanji */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%#6.4x is %sa single-byte alphabetic or Katakana alphabetic character\n",
                chars[i],
                ( _ismbbalpha( chars[i] ) ) ? "" : "not " );
    }
}

produces the following:
0x0020 is not a single-byte alphabetic or Katakana alphabetic character
0x002e is not a single-byte alphabetic or Katakana alphabetic character
0x0031 is not a single-byte alphabetic or Katakana alphabetic character
0x0041 is a single-byte alphabetic or Katakana alphabetic character
0x8140 is not a single-byte alphabetic or Katakana alphabetic character
0x8260 is not a single-byte alphabetic or Katakana alphabetic character
0x82a6 is a single-byte alphabetic or Katakana alphabetic character
0x8342 is a single-byte alphabetic or Katakana alphabetic character
0x00a1 is not a single-byte alphabetic or Katakana alphabetic character
0x00a6 is a single-byte alphabetic or Katakana alphabetic character
0x00df is a single-byte alphabetic or Katakana alphabetic character
0xe0a1 is not a single-byte alphabetic or Katakana alphabetic character

**Classification:** WATCOM

**Systems:** DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <mbctype.h>
int _ismbbgraph( unsigned int ch );

Description:  The _ismbbgraph function tests if the argument ch satisfies the condition that one of isgraph or
_ismbbkprint is true.

For example, in code page 932, _ismbbgraph tests if the argument ch is a single-byte printable
character excluding space (" ") or single-byte Katakana character.

Note: The argument ch must represent a single-byte value (i.e., 0 <= ch <= 255 ). Incorrect results
occur if the argument is a double-byte character. This is shown by the example below.

Returns:  The _ismbbgraph function returns a non-zero value if the argument satisfies the condition; otherwise
a zero value is returned.

See Also:  _getmbcp, _mbbtombc, _mbcjistojms, _mbcjmsstoijs, _mbctombb, _ismbbalnum,
_ismbbalpha, _ismbbkalnum, _ismbbkalpha, _ismbbkana, _ismbbkprint,
_ismbbkpunct, _ismbblead, _ismbbprint, _ismbbpunct, _ismbbtrail, _mbbtombc,  
_mbcjistojms, _mbcjmsstoijs, _mbctombb, _mbbtype, _setmbcp

Example:  
#include <stdio.h>
#include <mbctype.h>

unsigned int chars[] = {
    ' ',
    '.',
    'l',
    'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6,   /* single-byte Katakana alphabetic */
    0xDF,   /* single-byte Katakana alphabetic */
    0xE0A1  /* double-byte Kanji */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "\%6.4x is %s single-byte printable "
                "non-space character\n",
                chars[i],
                ( _ismbbgraph( chars[i] ) ) ? "" : "not " );
    }
}

produces the following:

434  Library Functions and Macros
0x0020 is not a single-byte printable non-space character
0x002e is a single-byte printable non-space character
0x0031 is a single-byte printable non-space character
0x0041 is a single-byte printable non-space character
0x8140 is a single-byte printable non-space character
0x8260 is a single-byte printable non-space character
0x82a6 is a single-byte printable non-space character
0x8342 is a single-byte printable non-space character
0x00a1 is a single-byte printable non-space character
0x00a6 is a single-byte printable non-space character
0x00df is a single-byte printable non-space character
0xe0a1 is a single-byte printable non-space character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <mbctype.h>
int _ismbbkalnum( unsigned int ch );

Description:  
The _ismbbkalnum function tests if the argument ch is a non-ASCII text symbol other than punctuation.

For example, in code page 932, _ismbbkalnum tests for a single-byte Katakana character (excluding the Katakana punctuation characters). Note that there are no Katakana digit characters. A single-byte Katakana non-punctuation character is any character for which the following expression is true:

0xA6 <= ch <= 0xDF

Note:  The argument ch must represent a single-byte value (i.e., 0 <= ch <= 255 ). Incorrect results occur if the argument is a double-byte character. This is shown by the example below.

Returns:  
The _ismbbkalnum function returns a non-zero value if the argument satisfies the condition; otherwise a zero value is returned.

See Also:  
_getmbcp, _mbbtombc, _mbcjistojms, _mbcjmstojis, _mbctombb, _ismbbalnum, _ismbbalpha, _ismbbgraph, _ismbbkalpaha, _ismbbkana, _ismbbkprint, _ismbbkpoint, _ismbblead, _ismbbprint, _ismbbpunct, _ismbbtrail, _mbbtombc, _mbcjistojms, _mbcjmstojis, _mbctombb, _mbbtype, _setmbcp

Example:  
#include <stdio.h>
#include <mbctype.h>

unsigned int chars[] = {
   ' ',
   ',',
   '.',
   '1',
   'A',
   0x8140, /* double-byte space */
   0x8260, /* double-byte A */
   0x82A6, /* double-byte Hiragana */
   0x8342, /* double-byte Katakana */
   0xA1,   /* single-byte Katakana punctuation */
   0xA6,   /* single-byte Katakana alphabetic */
   0xDF,   /* single-byte Katakana alphabetic */
   0xE0A1  /* double-byte Kanji */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
   int i;

   _setmbcp( 932 );
   for( i = 0; i < SIZE; i++ ) {
      printf( "\%#6.4x is %sa single-byte 
" "Katakana non-punctuation character\n",
         chars[i],
         ( _ismbbkalnum( chars[i] ) ) ? "" : "not " );
   }
}
produces the following:

0x0020 is not a single-byte Katakana non-punctuation character
0x002e is not a single-byte Katakana non-punctuation character
0x0031 is not a single-byte Katakana non-punctuation character
0x0041 is not a single-byte Katakana non-punctuation character
0x8140 is not a single-byte Katakana non-punctuation character
0x8260 is not a single-byte Katakana non-punctuation character
0x82a6 is a single-byte Katakana non-punctuation character
0x8342 is not a single-byte Katakana non-punctuation character
0x00a1 is not a single-byte Katakana non-punctuation character
0x00a6 is a single-byte Katakana non-punctuation character
0x00df is a single-byte Katakana non-punctuation character
0xe0a1 is not a single-byte Katakana non-punctuation character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_ismbbkana

Synopsis:  
#include <mbctype.h>  
int _ismbbkana( unsigned int ch );

Description:  
The _ismbbkana function tests if the argument \textit{ch} is a single-byte Katakana character. A single-byte Katakana character is any character for which the following expression is true:

\[
0xA1 <= \textit{ch} <= 0xDF
\]

Note: The argument \textit{ch} must represent a single-byte value (i.e., \(0 \leq \textit{ch} \leq 255\)). Incorrect results occur if the argument is a double-byte character. This is shown by the example below.

Returns:  
The _ismbbkana function returns non-zero if the argument is a single-byte Katakana character; otherwise, a zero value is returned.

See Also:  
_getmbcp, _mbbtombc, _mbcjistojms, _mbcjmstojis, _mbctombb, _ismbbalnum, _ismbbalpha, _ismbbgraph, _ismbbkalnum, _ismbbkalpha, _ismbbkprint, _ismbbkpunct, _ismbblead, _ismbbprint, _ismbbpunct, _ismbbtrail, _mbbtombc, _mbcjistojms, _mbcjmstojis, _mbctombb, _mbbtype, _setmbcp

Example:  
#include <stdio.h>  
#include <mbctype.h>

unsigned int chars[] = {
    ',', '.', ',', '1', 'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1, /* single-byte Katakana punctuation */
    0xA6, /* single-byte Katakana alphabetic */
    0xDF, /* single-byte Katakana alphabetic */
    0xE0A1 /* double-byte Kanji */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "\%6.4x is %s a single-byte Katakana character
",
            chars[i],
            ( _ismbbkana( chars[i] ) ) ? "" : "not " );
    }
}

produces the following:
0x0020 is not a single-byte Katakana character
0x002e is not a single-byte Katakana character
0x0031 is not a single-byte Katakana character
0x0041 is not a single-byte Katakana character
0x8140 is not a single-byte Katakana character
0x8260 is not a single-byte Katakana character
0x82a6 is a single-byte Katakana character
0x8342 is not a single-byte Katakana character
0x00a1 is a single-byte Katakana character
0x00a6 is a single-byte Katakana character
0x00df is a single-byte Katakana character
0xe0a1 is a single-byte Katakana character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:
#include <mbctype.h>
int _ismbbkalpha( unsigned int ch );

Description: The _ismbbkalpha function tests if the argument ch is a non-ASCII text symbol other than digits or punctuation.

For example, in code page 932, _ismbbkalpha tests for a single-byte Katakana character (excluding the Katakana punctuation characters). Note that there are no Katakana digit characters. A single-byte Katakana non-punctuation character is any character for which the following expression is true:

\[ 0xA6 \leq ch \leq 0xDF \]

Note: The argument ch must represent a single-byte value (i.e., 0 \leq ch \leq 255 ). Incorrect results occur if the argument is a double-byte character. This is shown by the example below.

Returns: The _ismbbkalpha function returns a non-zero value if the argument satisfies the condition; otherwise a zero value is returned.

See Also: _getmbcp, _mbbtombc, _mbcjistojms, _mbctombb, _ismbbalnum, _ismbbalpha, _ismbbgraph, _ismbbkalnum, _ismbbkana, _ismbbkprint, _ismbbkpunct, _ismbblead, _ismbbprint, _ismbbtrail, _mbbtombc, _mbcjistojms, _mbcjstojis, _mbctombb, _mbbtype, _setmbcp

Example:
#include <stdio.h>
#include <mbctype.h>

unsigned int chars[] = {
    '
', '
',
    '.', '.',
    '1', 'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6,   /* single-byte Katakana alphabetic */
    0xDF,   /* single-byte Katakana alphabetic */
    0xE0A1  /* double-byte Kanji */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "\x6.4x is %sa single-byte\n",
            "Katakana alphabetic character\n",
            chars[i],
            ( _ismbbkalpha( chars[i] ) ) ? "" : "not " );
    }
}
produces the following:

0x0020 is not a single-byte Katakana alphabetic character
0x002e is not a single-byte Katakana alphabetic character
0x0031 is not a single-byte Katakana alphabetic character
0x0041 is not a single-byte Katakana alphabetic character
0x8140 is not a single-byte Katakana alphabetic character
0x8260 is not a single-byte Katakana alphabetic character
0x82a6 is a single-byte Katakana alphabetic character
0x8342 is not a single-byte Katakana alphabetic character
0x00a1 is not a single-byte Katakana alphabetic character
0x00df is a single-byte Katakana alphabetic character
0xe0a1 is not a single-byte Katakana alphabetic character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <mbctype.h>
int _ismbbkprint( unsigned int ch );

Description: The _ismbbkprint function tests if the argument ch is a non-ASCII text or non-ASCII punctuation symbol.

For example, in code page 932, _ismbbkprint tests if the argument ch is a single-byte Katakana character. A single-byte Katakana character is any character for which the following expression is true:

    0xA1 <= ch <= 0xDF

Note: The argument ch must represent a single-byte value (i.e., 0 <= ch <= 255 ). Incorrect results occur if the argument is a double-byte character. This is shown by the example below.

Returns: The _ismbbkprint function returns a non-zero value if the argument satisfies the condition; otherwise a zero value is returned.

See Also: _getmbcp, _mbbtombc, _mbcjistojms, _mbctombb, _ismbbalnum, _ismbbalph, _ismbbkalpha, _ismbbkalnum, _ismbbkana, _ismbbkprint, _ismbbkpunct, _ismbblead, _ismbbprint, _ismbbpunct, _ismbbtrail, _mbbtombc, _mbcjistojms, _mbctombb, _mbbtype, _setmbcp

Example: #include <stdio.h>
#include <mbctype.h>

unsigned int chars[] = {
    ', ',
    '.', ',
    '1', '
    'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1, /* single-byte Katakana punctuation */
    0xA6, /* single-byte Katakana alphabetic */
    0xDF, /* single-byte Katakana alphabetic */
    0xE0A1 /* double-byte Kanji */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%5.4x is %sa single-byte
             "Katakana printable character\n",
            "Kakana printable character\n",
           =ismbbkprint( chars[i] ) ? "" : "not " );
    }
}
produces the following:

0x0020 is not a single-byte Katakana printable character
0x002e is not a single-byte Katakana printable character
0x0031 is not a single-byte Katakana printable character
0x0041 is not a single-byte Katakana printable character
0x8140 is not a single-byte Katakana printable character
0x8260 is not a single-byte Katakana printable character
0x82a6 is a single-byte Katakana printable character
0x8342 is not a single-byte Katakana printable character
0x00a1 is a single-byte Katakana printable character
0x00a6 is a single-byte Katakana printable character
0x00df is a single-byte Katakana printable character
0xe0a1 is a single-byte Katakana printable character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:
#include <mbctype.h>
int _ismbkbkpunct( unsigned int ch );

Description: The _ismbkbkpunct function tests if the argument \textit{ch} is a non-ASCII punctuation character.

For example, in code page 932, \texttt{ismbkbkpunct} tests if the argument \textit{ch} is a single-byte Katakana punctuation character. A single-byte Katakana punctuation character is any character for which the following expression is true:

\[ 0xA1 \leq \textit{ch} \leq 0xA5 \]

Note: The argument \textit{ch} must represent a single-byte value (i.e., \( 0 \leq \textit{ch} \leq 255 \)). Incorrect results occur if the argument is a double-byte character. This is shown by the example below.

Returns: The _ismbkbkpunct function returns a non-zero value if the argument satisfies the condition; otherwise a zero value is returned.

See Also: \_getmbcp, \_mbbtombc, \_mbcjistojms, \_mbctombb, \_ismbbalnum, \_ismbbalpha, \_ismbbgraph, \_ismbbkalnum, \_ismbbkkana, \_ismbbkprint, \_ismbblead, \_ismbbprint, \_ismbbpunct, \_ismbbtrail, \_mbbtombc, \_mbcjistojms, \_mbcjstojms, \_mbctombb, \_mbbtype, \_setmbcp

Example:
#include <stdio.h>
#include <mbctype.h>

unsigned int chars[] = {
    ',',
    '.',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6,   /* single-byte Katakana alphabetic */
    0xDF,   /* single-byte Katakana alphabetic */
    0xE0A1, /* double-byte Kanji */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "\%6.4x is %sa single-byte "
            "Katakana punctuation character\n",
            chars[i],
            ( _ismbkbkpunct( chars[i] ) ) ? "" : "not " );
    }
}

444 Library Functions and Macros
produces the following:

0x0020 is not a single-byte Katakana punctuation character
0x002e is not a single-byte Katakana punctuation character
0x0031 is not a single-byte Katakana punctuation character
0x0041 is not a single-byte Katakana punctuation character
0x8140 is not a single-byte Katakana punctuation character
0x8260 is not a single-byte Katakana punctuation character
0x82a6 is not a single-byte Katakana punctuation character
0x8342 is not a single-byte Katakana punctuation character
0x00a1 is a single-byte Katakana punctuation character
0x00a6 is not a single-byte Katakana punctuation character
0x00df is not a single-byte Katakana punctuation character
0xe0a1 is a single-byte Katakana punctuation character
0x00a6 is not a single-byte Katakana punctuation character
0x00df is not a single-byte Katakana punctuation character
0xe0a1 is a single-byte Katakana punctuation character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <mbctype.h>
int _ismbblead( unsigned int ch );

Description:  The _ismbblead function tests if the argument \textit{ch} is a valid first byte of a multibyte character.

For example, in code page 932, valid ranges are 0x81 through 0x9F and 0xE0 through 0xFC.

\textit{Note}: The argument \textit{ch} must represent a single-byte value (i.e., \(0 \leq \textit{ch} \leq 255\)). Incorrect results occur if the argument is a double-byte character.

Returns:  _ismbblead returns a non-zero value if the argument is valid as the first byte of a multibyte character; otherwise zero is returned.

See Also:  _getmbcp, _mbbtombc, _mbcjistojms, _mbcjstojis, _mbctombb, _ismbbalnum, _ismbbalpha, _ismbbgraph, _ismbbkalpha, _ismbbkana, _ismbbkprint, _ismbbkpunct, _ismbbpunct, _ismbbtrail, _mbbtombc, _mbcjistojms, _mbcjstojis, _mbctombb, _mbbtype, _setmbcp

Example:  
#include <stdio.h>
#include <mbctype.h>

unsigned int chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6,   /* single-byte Katakana alphabetic */
    0xDF,   /* single-byte Katakana alphabetic */
    0xE0A1  /* double-byte Kanji */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "#%6.4x does %shave a valid first byte\n",
                chars[i],
                ( _ismbblead( chars[i]>>8 ) ) ? "" : "not " );
    }
}

produces the following:
0x0020 does not have a valid first byte
0x002e does not have a valid first byte
0x0031 does not have a valid first byte
0x0041 does not have a valid first byte
0x8140 does have a valid first byte
0x8260 does have a valid first byte
0x82a6 does have a valid first byte
0x8342 does have a valid first byte
0x00a1 does not have a valid first byte
0x00a6 does not have a valid first byte
0x00df does not have a valid first byte
0xe0a1 does have a valid first byte

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <mbctype.h>
int _ismbbprint( unsigned int ch );

Description:  The _ismbbprint function tests if the argument ch is a single-byte printable character including space (" ").

For example, in code page 932, _ismbbprint tests if the argument ch is a single-byte printable character including space (" ") or a single-byte Katakana character. These are any characters for which the following expression is true:

\[
\text{isprint}(ch) \text{ or } \text{_ismbbkprint}(ch)
\]

Note: The argument ch must represent a single-byte value (i.e., \(0 \leq ch \leq 255\)). Incorrect results occur if the argument is a double-byte character. This is shown by the example below.

Returns:  The _ismbbprint function returns a non-zero value if the argument satisfies the condition; otherwise a zero value is returned.

See Also:  _getmbcp, _mbbtombc, _mbcjistojms, _mbcjmstojis, _mbctombb, _ismbbalnum, _ismbbalpha, _ismbbgraph, _ismbbkalnum, _ismbbkalpha, _ismbbkana, _ismbbkprint, _ismbbkpunct, _ismbblead, _ismbbtrail, _mbbtombc, _mbcjistojms, _mbcjmstojis, _mbctombb, _mbbtype, _setmbcp

Example:  
#include <stdio.h>
#include <mbctype.h>

unsigned int chars[] = {
  0x0D,
  '"',
  '1',
  'A',
  0x8140, /* double-byte space */
  0x8260, /* double-byte A */
  0x82A6, /* double-byte Hiragana */
  0x8342, /* double-byte Katakana */
  0xA1,   /* single-byte Katakana punctuation */
  0xA6,   /* single-byte Katakana alphabetic */
  0xDF,   /* single-byte Katakana alphabetic */
  0xE0A1  /* double-byte Kanji */
};
#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
  int i;
  _setmbcp( 932 );
  for( i = 0; i < SIZE; i++ ) {
    printf( "%#6.4x is %sa single-byte %
             "printable character\n",
             chars[i],
             ( _ismbbprint( chars[i] ) ) ? "" : "not " );
  }
}
produces the following:

0x000d is not a single-byte printable character
0x002e is a single-byte printable character
0x0031 is a single-byte printable character
0x0041 is a single-byte printable character
0x8140 is a single-byte printable character
0x8260 is a single-byte printable character
0x82a6 is a single-byte printable character
0x8342 is a single-byte printable character
0x00a1 is a single-byte printable character
0x00a6 is a single-byte printable character
0x00df is a single-byte printable character
0xe0a1 is a single-byte printable character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <mbctype.h>  
int _ismbbpunct( unsigned int ch );  

Description:  The _ismbbpunct function tests if the argument ch is a single-byte punctuation character.  
For example, in code page 932, _ismbbpunct tests if the argument ch is a single-byte punctuation character or a single-byte Katakana punctuation character.  These are any characters for which the following expression is true:  
ispunct(ch) || _ismbbkpunct(ch)  

Note:  The argument ch must represent a single-byte value (i.e., 0 <= ch <= 255 ). Incorrect results occur if the argument is a double-byte character. This is shown by the example below.  

Returns:  The _ismbbpunct function returns a non-zero value if the argument satisfies the condition; otherwise a zero value is returned.  

See Also:  _getmbcp, _mbbtombc, _mbcjistojms, _mbctombb, _ismbbalnum,  
_ismbbalpha, _ismbbgraph, _ismbbkalnum, _ismbbkalpha, _ismbbkana,  
_ismbbkprint, _ismbbpunct, _ismbblead, _ismbbprint, _ismbbtrail, _mbbtombc,  
_mbcjistojms, _mbcjmsjstojis, _mbctombb, _mbctype, _setmbcp  

Example:  
#include <stdio.h>  
#include <mbctype.h>  

unsigned int chars[] = {  
',',  
',',  
',',  
',1',  
'A',  
0x8140, /* double-byte space */  
0x8260, /* double-byte A */  
0x82A6, /* double-byte Hiragana */  
0x8342, /* double-byte Katakana */  
0xA1, /* single-byte Katakana punctuation */  
0xA6, /* single-byte Katakana alphabetic */  
0xDF, /* single-byte Katakana alphabetic */  
0xE0A1 /* double-byte Kanji */  
};  
#define SIZE sizeof( chars ) / sizeof( unsigned int )  

void main()  
{  
  int i;  
  _setmbcp( 932 );  
  for( i = 0; i < SIZE; i++ ) {  
    printf( "#%6.4x is %sa single-byte "  
      "punctuation character\n",  
    chars[i],  
    ( _ismbbpunct( chars[i] ) ) ? "" : "not " );  
  }  
}  

450 Library Functions and Macros
produces the following:

0x0020 is not a single-byte punctuation character
0x002e is a single-byte punctuation character
0x0031 is not a single-byte punctuation character
0x0041 is not a single-byte punctuation character
0x8140 is a single-byte punctuation character
0x8260 is a single-byte punctuation character
0x82a6 is not a single-byte punctuation character
0x8342 is not a single-byte punctuation character
0x00a1 is a single-byte punctuation character
0x00a6 is not a single-byte punctuation character
0x00df is not a single-byte punctuation character
0xe0a1 is a single-byte punctuation character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <mbstring.h>
int _ismbbtrail( unsigned int ch );

Description:  The _ismbbtrail function tests if ch is a valid second byte of a multibyte character.

For example, in code page 932, valid ranges are 0x40 through 0x7E and 0x80 through 0xFC.

Note: Only the least significant (trailing) byte of the argument ch is tested. If the argument is a
double-byte character, the leading byte is ignored and may be invalid. This is shown by the example
below.

Returns:  _ismbbtrail returns a non-zero value if the argument is valid as the second byte of a multibyte
character; otherwise zero is returned.

See Also: _getmbcp, _mbbtombc, _mbcjistojms, _mbcjstojis, _mbctombb, _ismbbalnum,
_ismbbalpha, _ismbbgraph, _ismbbkalnum, _ismbbkalpha, _ismbbkana,
_ismbbkprint, _ismbbkpunct, _ismbblead, _ismbbprint, _ismbbpunct, _mbbtombc,
_mbcjistojms, _mbcjstojis, _mbctombb, _mbctype, _setmbcp

Example:  
#include <stdio.h>
#include <mbctype.h>

unsigned int chars[] = {
    ' ',
    '.',
    'l',
    'A',
    0x8140, /* double-byte space */
    0x8260, /* double-byte A */
    0x82A6, /* double-byte Hiragana */
    0x8342, /* double-byte Katakana */
    0xA1,   /* single-byte Katakana punctuation */
    0xA6,   /* single-byte Katakana alphabetic */
    0xDF,   /* single-byte Katakana alphabetic */
    0xE0A1  /* double-byte Kanji */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;
    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%6.4x does %shave a valid second byte\n",
            chars[i],
            ( _ismbbtrail(chars[i]&0xff) ) ? "" : "not " );
    }
}

produces the following:

452  Library Functions and Macros
0x0020 does not have a valid second byte
0x002e does not have a valid second byte
0x0031 does not have a valid second byte
0x0041 does have a valid second byte
0x8140 does have a valid second byte
0x8260 does have a valid second byte
0x82a6 does have a valid second byte
0x8342 does have a valid second byte
0x00a1 does have a valid second byte
0x00a6 does have a valid second byte
0x00df does have a valid second byte
0xe0a1 does have a valid second byte

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <mbstring.h>

int _ismbcalnum( unsigned int ch );

Description:  
The _ismbcalnum function tests if the multibyte character argument ch is an alphanumeric character. For example, in code page 932, 'A' through 'Z', 'a' through 'z', or '0' through '9' and its corresponding double-byte versions are alphanumeric (among others). An alphanumeric character is any character for which _ismbcalpha or _ismbcdigit is true.

Returns:  
The _ismbcalnum function returns zero if the argument is not an alphanumeric character; otherwise, a non-zero value is returned.

See Also:  
_getmbcp, _ismbcalpha, _ismbccntrl, _ismbcdigit, _ismbcgraph, _ismbchira, _ismbckata, _ismbcl0, _ismbcl1, _ismbcl2, _ismbclegal, _ismbclower, _ismbcprint, _ismbcpunct, _ismbcspace, _ismbcsymbol, _ismbcupper, _ismbcxdigit, _mbbtype, _setmbcp

Example:  
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
    ',',
    '1',
    'A',
    0x8143, /* double-byte , */
    0x8254, /* double-byte 5 */
    0x8260, /* double-byte A */
    0x8279, /* double-byte Z */
    0x8281, /* double-byte a */
    0x829A, /* double-byte z */
    0x829F, /* double-byte Hiragana */
    0x8340, /* double-byte Katakana */
    0x837F, /* illegal double-byte character */
    0x889E, /* double-byte L0 character */
    0x889F, /* double-byte L1 character */
    0x989F, /* double-byte L2 character */
    0xA6    /* single-byte Katakana */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "\%6.4x is %sa valid "
            "multibyte alphanumeric character\n",
            chars[i],
            ( _ismbcalnum( chars[i] ) ) ? "" : "not " );
    }
}

produces the following:
\_ismbcalnum

\begin{verbatim}
0x002e is not a valid multibyte alphanumeri c character
0x0031 is a valid multibyte alphanumeri c character
0x0041 is a valid multibyte alphanumeri c character
0x8143 is not a valid multibyte alphanumeri c character
0x8254 is a valid multibyte alphanumeri c character
0x8260 is a valid multibyte alphanumeri c character
0x8279 is a valid multibyte alphanumeri c character
0x8281 is a valid multibyte alphanumeri c character
0x829a is a valid multibyte alphanumeri c character
0x829f is a valid multibyte alphanumeri c character
0x8340 is a valid multibyte alphanumeri c character
0x837f is not a valid multibyte alphanumeri c character
0x889e is not a valid multibyte alphanumeri c character
0x889f is a valid multibyte alphanumeri c character
0x989f is a valid multibyte alphanumeri c character
0x00a6 is a valid multibyte alphanumeri c character
\end{verbatim}

**Classification:** WATCOM

**Systems:** DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:    #include <mbstring.h>
int _ismbcalpha( unsigned int ch );

Description: The _ismbcalpha function tests if the multibyte character argument ch is an alphabetic character. For example, in code page 932, 'A' through 'Z' or 'a' through 'z' and its corresponding double-byte versions and the Katakana letters (0xA6 through 0xDF) are alphabetic.

Returns: The _ismbcalpha function returns zero if the argument is not an alphabetic character; otherwise, a non-zero value is returned.

See Also: _getmbcp, _ismbcalnum, _ismbccntrl, _ismbcdigit, _ismbcgaph, _ismbchira, _ismbckata, _ismbc10, _ismbcl1, _ismbcl2, _ismbclegal, _ismbclower, _ismbcprint, _ismbcpunct, _ismbcspac, _ismbcsymb, _ismbcupper, _ismbckata, _ismbcl0, _ismbcl1, _ismbcl2, _ismbclegal, _ismbclower, _ismbcprint, _ismbcpunct, _ismbcspac, _ismbcsymb, _ismbcupper, _ismbckata, _ismbcl0, _ismbcl1, _ismbcl2, _ismbclegal, _ismbclower, _ismbcprint, _ismbcpunct, _ismbcspac, _ismbcsymb, _ismbcupper

Example:    #include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
    ',',
    '1',
    'A',
    0x8143, /* double-byte , */
    0x8254, /* double-byte 5 */
    0x8260, /* double-byte A */
    0x8279, /* double-byte Z */
    0x8281, /* double-byte a */
    0x829A, /* double-byte z */
    0x829F, /* double-byte Hiragana */
    0x8340, /* double-byte Katakana */
    0x837F, /* illegal double-byte character */
    0x889E, /* double-byte L0 character */
    0x889F, /* double-byte L1 character */
    0x989F, /* double-byte L2 character */
    0xA6    /* single-byte Katakana */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf("%6.4x is %sa valid 
"multibyte alphabetic character\n",
chars[i],
    ( _ismbcalpha( chars[i] ) ) ? "" : "not " );
    }

produces the following:
Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <mbstring.h>
int _ismbccccntl( unsigned int ch );

Description:  The _ismbccccntl function tests for any multibyte control character.

Returns:  The _ismbccccntl function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also:  _getmbcp, _ismbcalnum, _ismbcalpha, _ismbcdigit, _ismbcgraph, _ismbchira, _ismbckata, _ismbc10, _ismbc11, _ismbc12, _ismbclegal, _ismbclower, _ismbcprint, _ismbcpunct, _ismbcspace, _ismbcsymbo1, _ismbcupper, _ismbcxdigit, _mbbtype, _setmbcp

Example:  
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
  0x0D,  /* newline */
  0x09,  /* horizontal tab */
  0x20,  /* space */
  1,  /* decimal digit */
  'A',  /* uppercase letter */
  0x8140, /* double-byte space */
  0x8143, /* double-byte , */
  0x8254, /* double-byte 5 */
  0x8260, /* double-byte A */
  0x8279, /* double-byte Z */
  0x8281, /* double-byte a */
  0x829A, /* double-byte z */
  0x989F, /* double-byte L2 character */
  0xA6
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
  int i;
  _setmbcp( 932 );
  for( i = 0; i < SIZE; i++ ) {
    printf( "\%6.4x is %s \n", "multibyte control character\n",
            chars[i],
            ( _ismbccccntl( chars[i] ) ) ? "" : "not " );
  }
}

produces the following:
0x000d is a valid multibyte control character
0x002e is not a valid multibyte control character
0x0020 is not a valid multibyte control character
0x0031 is not a valid multibyte control character
0x0041 is not a valid multibyte control character
0x8140 is a valid multibyte control character
0x8143 is a valid multibyte control character
0x8254 is not a valid multibyte control character
0x8260 is not a valid multibyte control character
0x8279 is not a valid multibyte control character
0x8281 is not a valid multibyte control character
0x829a is not a valid multibyte control character
0x989f is not a valid multibyte control character
0x00a6 is not a valid multibyte control character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
```
#include <mbstring.h>
int _ismbcdigit( unsigned int ch );
```

Description:  
The _ismbcdigit function tests for any multibyte decimal-digit character '0' through '9'. In code page 932, this includes the corresponding double-byte versions of these characters.

Returns:  
The _ismbcdigit function returns a non-zero value when the argument is a decimal-digit character. Otherwise, zero is returned.

See Also:  
_getmbcp, _ismbcalnum, _ismbcalpha, _ismbccntrl, _ismbcspace, _ismbcsymbol, _ismbcupper, _ismbcxdigit, _mbctype, _setmbcp

Example:  
```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
    ',',
    '1',
    'A',
    0x8143, /* double-byte , */
    0x8183, /* double-byte < */
    0x8254, /* double-byte 5 */
    0x8277, /* double-byte X */
    0xA6
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "\%#6.4x is %sa valid multibyte digit character\n",
                chars[i],
                ( _ismbcdigit( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

0x002e is not a valid multibyte digit character
0x0031 is a valid multibyte digit character
0x0041 is not a valid multibyte digit character
0x8143 is not a valid multibyte digit character
0x8183 is not a valid multibyte digit character
0x8254 is a valid multibyte digit character
0x8277 is not a valid multibyte digit character
0x00a6 is not a valid multibyte digit character

460 Library Functions and Macros
Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <mbstring.h>
int _ismbcgraph( unsigned int ch );

Description:  The _ismbcgraph function tests for any printable multibyte character except space (' '). The _ismbcpint function is similar, except that the space character is also included in the character set being tested.

Returns:  The _ismbcgraph function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also:  _getmbcp, _ismbcalnum, _ismbcalpha, _ismbccntrl, _ismbcdigit, _ismbchira, _ismbckata, _ismbcl0, _ismbcl1, _ismbcl2, _ismbclegal, _ismbclower, _ismbcprint, _ismbcpunct, _ismbcspace, _ismbcsymbol, _ismbcupper, _ismbcxdigit, _mbbtype, _setmbcp

Example:  
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
    ',', ',', '1', 'A',
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8254, /* double-byte 5 */
    0x8260, /* double-byte A */
    0x8279, /* double-byte Z */
    0x8281, /* double-byte a */
    0x829a, /* double-byte Z */
    0x989f, /* double-byte L2 character */
    0xa6
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "\%6.4x is %sa valid ",
            "multibyte graph character\n",
            chars[i],
            ( _ismbcgraph( chars[i] ) ) ? "" : "not " );
    }
}

produces the following:
0x002e is a valid multibyte graph character
0x0020 is not a valid multibyte graph character
0x0031 is a valid multibyte graph character
0x0041 is a valid multibyte graph character
0x8140 is not a valid multibyte graph character
0x8143 is a valid multibyte graph character
0x8254 is a valid multibyte graph character
0x8260 is a valid multibyte graph character
0x8279 is a valid multibyte graph character
0x8281 is a valid multibyte graph character
0x829a is a valid multibyte graph character
0x989f is a valid multibyte graph character
0x00a6 is a valid multibyte graph character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <mbstring.h>
int _ismbchira( unsigned int ch );

Description:  The _ismbchira function tests for a double-byte Hiragana character. A double-byte Hiragana character is any character for which the following expression is true:

0x829F <= ch <= 0x82F1

Note:  The Japanese double-byte character set includes Kanji, Hiragana, and Katakana characters - both alphabetic and numeric. Kanji is the ideogram character set of the Japanese character set. Hiragana and Katakana are two types of phonetic character sets of the Japanese character set. The Hiragana code set includes 83 characters and the Katakana code set includes 86 characters.

Returns:  The _ismbchira function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also:  _getmbcp, _ismbcalnum, _ismbcalpha, _ismbccntrl, _ismbcdigit, _ismbcgraph, _ismbckata, _ismbckat2, _ismbckat3, _ismbcctrl, _ismbcupper, _ismbcxdigit, _mbctype, _setmbcp

Example:  
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
    'A',
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8260, /* double-byte A */
    0x829F, /* double-byte Hiragana */
    0x8340, /* double-byte Katakana */
    0x837F, /* illegal double-byte character */
    0x989F, /* double-byte L2 character */
    0xA6 /* single-byte Katakana */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%6.4x is %sa valid
" 
                "Hiragana character\n",
                chars[i],
                ( _ismbchira( chars[i] ) ) ? "" : "not " );
    }
}

produces the following:
Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
**Synopsis:**

```c
#include <mbstring.h>
int _ismbckata( unsigned int ch );
```

**Description:**
The `_ismbckata` function tests for a double-byte Katakana character. A double-byte Katakana character is any character for which the following expression is true:

```
0x8340 <= ch <= 0x8396  &&  ch != 0x837F
```


**Returns:**
The `_ismbckata` function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

**See Also:**
`_getmbcp`, `_ismbcalnum`, `_ismbcalpha`, `_ismbccntrl`, `_ismbcdigit`, `_ismbcgraph`, `_ismbcchira`, `_ismbc10`, `_ismbc11`, `_ismbc12`, `_ismbclegal`, `_ismbclower`, `_ismbcpunct`, `_ismbcsym`, `_ismbcspace`, `_ismbcsym`, `_ismbcupper`, `_ismbcxdigit`, `_mbbtype`, `_setmbcp`

**Example:**

```c
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
    'A',
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8260, /* double-byte A */
    0x829F, /* double-byte Hiragana */
    0x8340, /* double-byte Katakana */
    0x837F, /* illegal double-byte character */
    0x989F, /* double-byte L2 character */
    0xA6    /* single-byte Katakana */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "#%6.4x is %sa valid 
" "Katakana character\n",
            chars[i],
            ( _ismbckata( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:
Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

```
#include <mbstring.h>
int _ismbcl0( unsigned int ch );
```

Description:
The `_ismbcl0` function tests if the argument `ch` is in the set of double-byte characters that include
Hiragana, Katakana, punctuation symbols, graphical symbols, Roman and Cyrillic alphabets, etc.
Double-byte Kanji characters are not in this set. These are any characters for which the following
expression is true:

```
0x8140 <= ch <= 0x889E && ch != 0x837F
```

The `_ismbcl0` function tests if the argument is a valid double-byte character (i.e., it checks that the
lower byte is not in the ranges 0x00 - 0x3F, 0x7F, or 0xFD - 0xFF).

Note: The Japanese double-byte character set includes Kanji, Hiragana, and Katakana characters - both
alphabetic and numeric. Kanji is the ideogram character set of the Japanese character set. Hiragana and
Katakana are two types of phonetic character sets of the Japanese character set. The Hiragana code set
includes 83 characters and the Katakana code set includes 86 characters.

Returns: The `_ismbcl0` function returns a non-zero value when the argument is a member of this set of
characters; otherwise, zero is returned.

See Also: `_getmbcp`, `_ismbcnot`, `_ismbcnotalpha`, `_ismbccntrl`, `_ismbcdigit`, `_ismbcgraph`,
`_ismbchira`, `_ismbckata`, `_ismbcl1`, `_ismbcl2`, `_ismbclegal`, `_ismbclower`,
`_ismbcprint`, `_ismbcpunct`, `_ismbcspace`, `_ismbcsymbol`, `_ismbcupper`,
`_ismbcxdigit`, `_mbbtype`, `_setmbcp`

Example:
```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
'A',
0x8140, /* double-byte space */
0x8143, /* double-byte , */
0x8260, /* double-byte A */
0x829F, /* double-byte Hiragana */
0x8340, /* double-byte Katakana */
0x837F, /* illegal double-byte character */
0x889E, /* double-byte L0 character */
0x889F, /* double-byte L1 character */
0x989F, /* double-byte L2 character */
0xA6 /* single-byte Katakana */
};
#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;
```
```c
_setmbcp( 932 );
for( i = 0; i < SIZE; i++ ) {
    printf( " %#6.4x is %sa valid " 
            "JIS L0 character\n",
            chars[i],
            ( _ismbcl0( chars[i] ) ) ? "" : "not " );
}
```

produces the following:

0x0041 is not a valid JIS L0 character
0x8140 is a valid JIS L0 character
0x8143 is a valid JIS L0 character
0x8260 is a valid JIS L0 character
0x829f is a valid JIS L0 character
0x8340 is a valid JIS L0 character
0x837f is not a valid JIS L0 character
0x889e is a valid JIS L0 character
0x889f is not a valid JIS L0 character
0x989f is not a valid JIS L0 character
0x00a6 is not a valid JIS L0 character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <mbstring.h>  
int _ismbcl1( unsigned int ch );

Description:  
The _ismbcl1 function tests if the argument ch is a JIS (Japan Industrial Standard) level 1 double-byte character code. These are any valid double-byte characters for which the following expression is true:

\[
0x889F \leq \text{ch} \leq 0x9872
\]

The _ismbcl1 function tests if the argument is a valid double-byte character (i.e., it checks that the lower byte is not in the ranges 0x00 - 0x3F, 0x7F, or 0xFD - 0xFF).

Note:  JIS establishes two levels of the Kanji double-byte character set. One is called double-byte Kanji code set level 1 and the other is called double-byte Kanji code set level 2. Usually Japanese personal computers have font ROM/RAM support for both levels.

Valid double-byte characters are those in which the first byte falls in the range 0x81 - 0x9F or 0xE0 - 0xFC and whose second byte falls in the range 0x40 - 0x7E or 0x80 - 0xFC.

Returns:  
The _ismbcl1 function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also:  
_getmbscp, _ismbcalnum, _ismbcalpha, _ismbccntrl, _ismbcdigit, _ismbcgraph,  
_ismbchira, _ismbckata, _ismbcl0, _ismbcl1, _ismbclegal, _ismbclower,  
_ismbcpunct, _ismbcpunct, _ismbcspace, _ismbcsymbol, _ismbcupper,  
_ismbcxdigit, _mbbtype, _setmbscp

Example:  
#include <stdio.h>  
#include <mbctype.h>  
#include <mbstring.h>

unsigned int chars[] = {
    'A',  /* A */
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8260, /* double-byte A */
    0x829F, /* double-byte Hiragana */
    0x8340, /* double-byte Katakana */
    0x837F, /* illegal double-byte character */
    0x889E, /* double-byte L0 character */
    0x889F, /* double-byte L1 character */
    0x989F, /* double-byte L2 character */
    0xA6, /* single-byte Katakana */

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int   i;
}
```c
_setmbcp( 932 );
for( i = 0; i < SIZE; i++ ) {
    printf( "%#6.4x is %sa valid 
    "JIS L1 character\n",
        chars[i],
        ( _ismbcll( chars[i] ) ) ? "" : "not " );
}
}
```

produces the following:

0x0041 is not a valid JIS L1 character
0x8140 is not a valid JIS L1 character
0x8143 is not a valid JIS L1 character
0x8260 is not a valid JIS L1 character
0x829f is not a valid JIS L1 character
0x8340 is not a valid JIS L1 character
0x837f is not a valid JIS L1 character
0x889e is not a valid JIS L1 character
0x889f is a valid JIS L1 character
0x989f is not a valid JIS L1 character
0x00a6 is not a valid JIS L1 character

**Classification:** WATCOM

**Systems:** DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <mbstring.h>  
int _ismbcl2( unsigned int ch );

Description:  The _ismbcl2 function tests if the argument ch is a JIS (Japan Industrial Standard) level 2 double-byte character code. These are any valid double-byte characters for which the following expression is true:

\[ 0x989F \leq ch \leq 0xEA9E \]

The _ismbcl2 function tests if the argument is a valid double-byte character (i.e., it checks that the lower byte is not in the ranges 0x00 - 0x3F, 0x7F, or 0xFD - 0xFF).

Note: JIS establishes two levels of the Kanji double-byte character set. One is called double-byte Kanji code set level 1 and the other is called double-byte Kanji code set level 2. Usually Japanese personal computers have font ROM/RAM support for both levels.

Valid double-byte characters are those in which the first byte falls in the range 0x81 - 0x9F or 0xE0 - 0xFC and whose second byte falls in the range 0x40 - 0x7E or 0x80 - 0xFC.

Returns: The _ismbcl2 function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also: _getmbcp, _ismbcalnum, _ismbcalpha, _ismbccntrl, _ismbcdigit, _ismbcgraph, _ismbchira, _ismbckata, _ismbcl0, _ismbcl1, _ismbcl2, _ismbclegal, _ismbclower, _ismbcprint, _ismbcpunct, _ismbcspace, _ismbcsymbol, _ismbcupper, _ismbcxdigit, _mbbtype, _setmbcp

Example:  
#include <stdio.h>  
#include <mbctype.h>  
#include <mbstring.h>

    unsigned int chars[] = {  
        'A',  
        0x8140, /* double-byte space */  
        0x8143, /* double-byte , */  
        0x8260, /* double-byte A */  
        0x829F, /* double-byte Hiragana */  
        0x8340, /* double-byte Katakana */  
        0x837F, /* illegal double-byte character */  
        0x889E, /* double-byte L0 character */  
        0x889F, /* double-byte L1 character */  
        0x989F, /* double-byte L2 character */  
        0xEA9E, /* double-byte L2 character */  
        0xA6 /* single-byte Katakana */  
    };

    #define SIZE sizeof( chars ) / sizeof( unsigned int )

    void main()
    {
        int i;
```c
_setmbcp( 932 );
for( i = 0; i < SIZE; i++ ) {
    printf( "\%6.4x is %sa valid "
            "JIS L2 character\n",
            chars[i],
            ( _ismbcl2( chars[i] ) ) ? "" : "not " );
}
```

produces the following:

0x0041 is not a valid JIS L2 character
0x8140 is not a valid JIS L2 character
0x8143 is not a valid JIS L2 character
0x8260 is not a valid JIS L2 character
0x829f is not a valid JIS L2 character
0x8340 is not a valid JIS L2 character
0x837f is not a valid JIS L2 character
0x889e is not a valid JIS L2 character
0x889f is not a valid JIS L2 character
0x989f is a valid JIS L2 character
0xea9e is a valid JIS L2 character
0x00a6 is not a valid JIS L2 character

**Classification:** WATCOM

**Systems:** DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis: #include <mbstring.h>
int _ismbclegal( unsigned int dbch );

Description: The _ismbclegal function tests for a valid multibyte character. Multibyte characters include both single-byte and double-byte characters. For example, in code page 932, a legal double-byte character is one in which the first byte is within the ranges 0x81 - 0x9F or 0xE0 - 0xFC, while the second byte is within the ranges 0x40 - 0x7E or 0x80 - 0xFC. This is summarized in the following diagram.

[ 1st byte ]    [ 2nd byte ]
0x81-0x9F       0x40-0x7E
or             except 0x7F
0xE0-0xFC

Returns: The _ismbclegal function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also: _getmbcp, _ismbcalnum, _ismbcalpha, _ismbcntrl, _ismbcdigit, _ismbcgraph, _ismbchira, _ismbckata, _ismbcl0, _ismbcl1, _ismbcl2, _ismbclower, _ismbcprint, _ismbcpunct, _ismbcspace, _ismbcsymbol, _ismbcupper, _ismbcxdigit, _mbbtype, _setmbcp

Example: #include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
    'A',
    0x8131, /* illegal double-byte character */
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8260, /* double-byte A */
    0x829F, /* double-byte Hiragana */
    0x8340, /* double-byte Katakana */
    0x837F, /* illegal double-byte character */
    0x889E, /* double-byte L0 character */
    0x889F, /* double-byte L1 character */
    0x989F, /* double-byte L2 character */
    0xEA9E, /* double-byte L2 character */
    0xA6    /* single-byte Katakana */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "\%6.4x is %sa legal "
            "double-byte character\n",
            chars[i],
            ( _ismbclegal( chars[i] ) ) ? "" : "not " );
    }
}
produces the following:

0x0041 is not a legal double-byte character
0x8131 is not a legal double-byte character
0x8140 is a legal double-byte character
0x8143 is a legal double-byte character
0x8260 is a legal double-byte character
0x829f is a legal double-byte character
0x8340 is a legal double-byte character
0x837f is not a legal double-byte character
0x889e is a legal double-byte character
0x889f is a legal double-byte character
0xe9af is a legal double-byte character
0x00a6 is not a legal double-byte character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <mbstring.h>
int _ismbclower( unsigned int ch );

Description:  The _ismbclower function tests for a valid lowercase multibyte character. Multibyte characters include both single-byte and double-byte characters. For example, in code page 932, a lowercase double-byte character is one for which the following expression is true:

\[ 0x8281 \leq c \leq 0x829A \]

Returns:  The _ismbclower function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also:  _getmbcp, _ismbcalnum, _ismbcalpha, _ismbccntrl, _ismbcdigit, _ismbcgraph, _ismbchira, _ismbckata, _ismbcl10, _ismbcl1, _ismbcl2, _ismbclegal, _ismbcprint, _ismbcpunct, _ismbcspace, _ismbcsymbol, _ismbcupper, _ismbcxdigit, _mbbtype, _setmbcp

Example:  
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
    '1',
    'A',
    'a',
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8254, /* double-byte 5 */
    0x8260, /* double-byte A */
    0x8279, /* double-byte Z */
    0x8281, /* double-byte a */
    0x829A, /* double-byte z */
    0x989F, /* double-byte L2 character */
    0xA6
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "%#6.4x is %sa valid "
            "multibyte lowercase character\n",
            chars[i],
            ( _ismbclower( chars[i] ) ) ? "" : "not " );
    }
}

produces the following:
Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <mbstring.h>

int _ismbcpnprint( unsigned int ch );

Description:  The _ismbcpnprint function tests for any printable multibyte character including space (" "). The _ismbcpngraph function is similar, except that the space character is not included in the character set being tested.

Returns:  The _ismbcpnprint function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also:  _getmbcp, _ismbcvalnum, _ismbcalpha, _ismbccntrl, _ismbcdigit, _ismbcgraph, _ismbchira, _ismbckata, _ismbc10, _ismbc11, _ismbc12, _ismbc2legal, _ismbcblower, _ismbcpunct, _ismbcspace, _ismbcsymbol, _ismbcupe, _ismbcxdigit, _mbbtype, _setmbcp

Example:  
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
   ',', ',',
   '1',
   'A',
   0x8140, /* double-byte space */
   0x8143, /* double-byte , */
   0x8254, /* double-byte 5 */
   0x8260, /* double-byte A */
   0x8279, /* double-byte Z */
   0x8281, /* double-byte a */
   0x829a, /* double-byte z */
   0x989f, /* double-byte L2 character */
   0xa6
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
   int i;

   _setmbcp( 932 );
   for( i = 0; i < SIZE; i++ ) {
      printf( "\%6.4x is %sa valid "
            "multibyte print character\n",
            chars[i],
            ( _ismbcpnprint( chars[i] ) ) ? "" : "not " );
   }
}

produces the following:
0x002e is a valid multibyte print character
0x0020 is a valid multibyte print character
0x0031 is a valid multibyte print character
0x0041 is a valid multibyte print character
0x8140 is a valid multibyte print character
0x8143 is a valid multibyte print character
0x8254 is a valid multibyte print character
0x8260 is a valid multibyte print character
0x8279 is a valid multibyte print character
0x8281 is a valid multibyte print character
0x989f is a valid multibyte print character
0x00a6 is a valid multibyte print character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <mbstring.h>
int _ismbcpunct( unsigned int ch );

Description:  The _ismbcpunct function tests for any multibyte punctuation character.

Returns:  The _ismbcpunct function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also:  _getmbcp, _ismbcalnum, _ismbcalpha, _ismbccntrl, _ismbcdigit, _ismbcgraph, _ismbchira, _ismbckata, _ismbcl0, _ismbcl1, _ismbcl2, _ismbclegal, _ismbclower, _ismbcprint, _ismbcspace, _ismbcsymbol, _ismbcupper, _ismbcxdigit, _mbbtype, _setmbcp

Example:  
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
  ',', ',',
  '1', 'A',
  0x8140, /* double-byte space */
  0x8143, /* double-byte , */
  0x8254, /* double-byte 5 */
  0x8260, /* double-byte A */
  0x8279, /* double-byte Z */
  0x8281, /* double-byte a */
  0x829A, /* double-byte z */
  0x889F, /* double-byte L2-character */
  0xA1,   /* single-byte Katakana punctuation */
  0xA6    /* single-byte Katakana alphabetic */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
  int i;
  _setmbc( 932 );
  for( i = 0; i < SIZE; i++ ) {
    printf( "%#6.4x is %sa valid "
      "multibyte punctuation character\n",
        chars[i],
        ( _ismbcpunct( chars[i] ) ) ? "" : "not " );
  }
}

produces the following:
0x002e is a valid multibyte punctuation character
0x0020 is not a valid multibyte punctuation character
0x0031 is not a valid multibyte punctuation character
0x0041 is not a valid multibyte punctuation character
0x8140 is not a valid multibyte punctuation character
0x8143 is a valid multibyte punctuation character
0x8254 is not a valid multibyte punctuation character
0x8260 is not a valid multibyte punctuation character
0x8279 is not a valid multibyte punctuation character
0x8281 is not a valid multibyte punctuation character
0x829a is not a valid multibyte punctuation character
0x989f is not a valid multibyte punctuation character
0x00a1 is a valid multibyte punctuation character
0x00a6 is not a valid multibyte punctuation character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <mbstring.h>
int _ismbcspace( unsigned int ch );

Description:  The _ismbcspace function tests for any multibyte space character. Multibyte characters include both single-byte and double-byte characters. For example, in code page 932, the double-byte space character is 0x8140.

Returns:  The _ismbcspace function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also:  _getmbcp, _ismbcalnum, _ismbcalpha, _ismbccntrl, _ismbcdigit, _ismbcgraph, _ismbchira, _ismbckata, _ismbcl0, _ismbcl1, _ismbcl2, _ismbclegal, _ismbclower, _ismbcprint, _ismbcpunct, _ismbcsymbol, _ismbcupper, _ismbcxdigit, _mbbtype, _setmbcp

Example:  
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
  0x09,
  ',',
  ',',
  ',1',
  ',A',
  0x8140, /* double-byte space */
  0x8143, /* double-byte , */
  0x8254, /* double-byte 5 */
  0x8260, /* double-byte A */
  0x8279, /* double-byte Z */
  0x8281, /* double-byte a */
  0x829A, /* double-byte z */
  0x989F, /* double-byte L2 character */
  0xA6
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
  int i;

  _setmbcp( 932 );

  for( i = 0; i < SIZE; i++ ) {
    printf( "\%6.4x is %sa valid multibyte space character\n",
            chars[i],
            ( _ismbcspace( chars[i] ) ) ? "" : "not " );
  }
}

produces the following:
0x0009 is a valid multibyte space character
0x002e is not a valid multibyte space character
0x0020 is a valid multibyte space character
0x0031 is not a valid multibyte space character
0x0041 is not a valid multibyte space character
0x8140 is a valid multibyte space character
0x8143 is not a valid multibyte space character
0x8254 is not a valid multibyte space character
0x8260 is not a valid multibyte space character
0x8279 is not a valid multibyte space character
0x8281 is not a valid multibyte space character
0x829a is not a valid multibyte space character
0x989f is not a valid multibyte space character
0x00a6 is not a valid multibyte space character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
**Synopsis:**

`#include <mbstring.h>
int _ismbcsymbol( unsigned int ch );`

**Description:**
The `_ismbcsymbol` function tests for a valid multibyte symbol character (punctuation and other special graphical symbols). For example, in code page 932, `_ismbcsymbol` tests for a double-byte Kigou character and returns true if and only if

```
0x8141 <= ch <= 0x81AC && ch != 0x817F
```

**Returns:**
The `_ismbcsymbol` function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

**See Also:**
`_getmbcp`, `_ismbcalph`, `_ismbcntrl`, `_ismbcdigit`, `_ismbcgraph`, `_ismbchira`, `_ismbckata`, `_ismbcl0`, `_ismbcl1`, `_ismbcl2`, `_ismbclegal`, `_ismbcnoc`, `_ismbcpunct`, `_ismbcspace`, `_ismbcupper`, `_ismbcxdigit`, `_mbbtype`, `_setmbcp`

**Example:**

```c
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
    '.', ',', ',', 
    '1', 'A',
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8254, /* double-byte 5 */
    0x8260, /* double-byte A */
    0x8279, /* double-byte Z */
    0x8281, /* double-byte a */
    0x829A, /* double-byte z */
    0x989F, /* double-byte L2 character */
    0xA6
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "\%6.4x is %sa valid "
            "multibyte symbol character\n",
            chars[i],
            ( _ismbcsymbol( chars[i] ) ) ? "" : "not "
        );
    }
}
```

produces the following:
Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <mbstring.h>  
int _ismbcupper( unsigned int ch );

Description:  The _ismbcupper function tests for a valid uppercase multibyte character. Multibyte characters include both single-byte and double-byte characters. For example, in code page 932, an uppercase double-byte character is one for which the following expression is true:

\[ 0x8260 \leq c \leq 0x8279 \]

Returns:  The _ismbcupper function returns a non-zero value when the argument is a member of this set of characters; otherwise, zero is returned.

See Also:  _getmbcp, _ismbcalnum, _ismbcalpha, _ismbccntrl, _ismbcdigit, _ismbcgraph, _ismbchira, _ismbckata, _ismbcl0, _ismbcl1, _ismbcl2, _ismbclegal, _ismbclower, _ismbcprint, _ismbcprintf, _ismbcspace, _ismbcsymbol, _ismbcxdigit, _mbbtype, _setmbcp

Example:  
#include <stdio.h>  
#include <mbctype.h>  
#include <mbstring.h>

unsigned int chars[] = {
    '1',
    'A',
    'a',
    0x8140, /* double-byte space */
    0x8143, /* double-byte , */
    0x8154, /* double-byte 5 */
    0x8260, /* double-byte A */
    0x8279, /* double-byte Z */
    0x8281, /* double-byte a */
    0x829A, /* double-byte z */
    0x989F, /* double-byte L2 character */
    0xA6
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "\%6.4x is %sa valid "
                "multibyte uppercase character\n",
                chars[i],
                ( _ismbcupper( chars[i] ) ) ? "" : "not " );
    }
}

produces the following:
0x0031 is not a valid multibyte uppercase character
0x0041 is a valid multibyte uppercase character
0x0061 is not a valid multibyte uppercase character
0x8140 is not a valid multibyte uppercase character
0x8143 is not a valid multibyte uppercase character
0x8254 is not a valid multibyte uppercase character
0x8260 is a valid multibyte uppercase character
0x8279 is a valid multibyte uppercase character
0x8281 is not a valid multibyte uppercase character
0x829a is not a valid multibyte uppercase character
0x989f is not a valid multibyte uppercase character
0x00a6 is not a valid multibyte uppercase character

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <mbstring.h>
int _ismbcxdigit( unsigned int ch );

Description:  The _ismbcxdigit function tests for any multibyte hexadecimal-digit character '0' through '9' or 'A' through 'F'. In code page 932, this includes the corresponding double-byte versions of these characters.

Returns:  The _ismbcxdigit function returns a non-zero value when the argument is a hexadecimal-digit character. Otherwise, zero is returned.

See Also:  _getmbcp, _ismbcalnum, _ismbcalpha, _ismbccntrl, _ismbcdigit, _ismbcgraph, _ismbchira, _ismbckata, _ismbcl0, _ismbcl1, _ismbcl2, _ismbclegal, _ismbclower, _ismbcprint, _ismbcpunct, _ismbcspace, _ismbcsymbol, _ismbcuppper, _mbbtype, _setmbcp

Example:  
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
    '.',
    '1',
    'A',
    '0x8143', /* double-byte "," */
    '0x8183', /* double-byte "," */
    '0x8254', /* double-byte "," */
    '0x8265', /* double-byte "," */
    '0xA6'};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;

    _setmbcp( 932 );

    for( i = 0; i < SIZE; i++ ) {
        printf( "%#6.4x is %sa valid multibyte hexadecimal digit character\n",
            _ismbcxdigit( chars[i] ) ) ;
    }
}

produces the following:

0x002e is not a valid multibyte hexadecimal digit character
0x0031 is a valid multibyte hexadecimal digit character
0x0041 is a valid multibyte hexadecimal digit character
0x08143 is not a valid multibyte hexadecimal digit character
0x08183 is not a valid multibyte hexadecimal digit character
0x08254 is a valid multibyte hexadecimal digit character
0x08265 is a valid multibyte hexadecimal digit character
0x00a6 is not a valid multibyte hexadecimal digit character

488  Library Functions and Macros
Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis: 
#include <math.h>
int isnan( x );

Description: The isnan macro determines whether its argument x is a NaN. First, an argument represented in a format wider than its semantic type is converted to its semantic type. Then determination is based on the type of the argument.

The argument x must be an expression of real floating type.

Returns: The isnan macro returns a nonzero value if and only if its argument has a NaN value.

See Also: fpclassify, isfinite, isinf, isnormal, signbit

Example: 
#include <math.h>
#include <stdio.h>

void main( void )
{
    printf( "NAN %s a NaN\n",
            isnan( NAN ) ? "is" : "is not" );
}

produces the following:

NAN is a NaN

Classification: ANSI

Systems: MACRO
Synopsis:    #include <math.h>
    int isnormal( x );

Description: The isnormal macro determines whether its argument value is normal (neither zero, subnormal,
infinite, nor NaN). First, an argument represented in a format wider than its semantic type is converted
to its semantic type. Then determination is based on the type of the argument.

The argument x must be an expression of real floating type.

Returns: The isnormal macro returns a nonzero value if and only if its argument has a normal value.

See Also: fpclassify, isnormal, isinf, isnan, signbit

Example:    #include <math.h>
    #include <stdio.h>

    void main( void )
    { 
        printf( "zero %s a normal number\n",
               isnormal( 0.0 ) ? "is" : "is not" );
    }

    produces the following:

    zero is not a normal number

Classification: ANSI

Systems: MACRO
###Synopsis:

```c
#include <ctype.h>
int isprint( int c );
#include <wctype.h>
int iswprint( wint_t c );
```

###Description:

The `isprint` function tests for any printable character including space (' '). The `isgraph` function is similar, except that the space character is excluded from the character set being tested.

The `iswprint` function is similar to `isprint` except that it accepts a wide-character argument.

###Returns:

The `isprint` function returns a non-zero value when the argument is a printable character. The `iswprint` function returns a non-zero value when the argument is a printable wide character. Otherwise, zero is returned.

###See Also:

`isalnum`, `isalpha`, `isblank`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `ispunct`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `toupper`, `towctrans`

###Example:

```c
#include <stdio.h>
#include <ctype.h>

char chars[] = {
   'A',
   0x09,
   ' ',
   0x7d
};

#define SIZE sizeof( chars ) / sizeof( char )

void main()
{
   int i;

   for( i = 0; i < SIZE; i++ ) {
      printf( "Char %c is %sa printable character\n",
              chars[i],
              ( isprint( chars[i] ) ? "" : "not " );
   }
}
```

produces the following:

Char A is a printable character
Char     is not a printable character
Char   is a printable character
Char } is a printable character

###Classification:

- isprint is ANSI
- iswprint is ANSI

###Systems:

- isprint - All, Netware
- iswprint - All, Netware
Synopsis:

```c
#include <ctype.h>
int ispunct( int c );
#include <wctype.h>
int iswpunct( wint_t c );
```

Description:
The `ispunct` function tests for any punctuation character such as a comma (,) or a period (.)

The `iswpunct` function is similar to `ispunct` except that it accepts a wide-character argument.

Returns:
The `ispunct` function returns a non-zero value when the argument is a punctuation character. The `iswpunct` function returns a non-zero value when the argument is a printable wide character that is neither the space wide character nor a wide character for which `iswalnum` is true. Otherwise, zero is returned.

See Also: `isalnum`, `isalpha`, `isblank`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `isprint`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `toupper`, `towctrans`

Example:

```c
#include <stdio.h>
#include <ctype.h>

char chars[] = {
    'A',
    '!',
    ',',
    ':',
    ';',
};

#define SIZE sizeof( chars ) / sizeof( char )

void main()
{
    int i;

    for( i = 0; i < SIZE; i++ ) {
        printf( "Char %c is %sa punctuation character\n",
                chars[i],
                ( ispunct( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

Char A is not a punctuation character
Char ! is a punctuation character
Char . is a punctuation character
Char , is a punctuation character
Char : is a punctuation character
Char ; is a punctuation character

Classification: `ispunct` is ANSI
`iswpunct` is ANSI

Systems: `ispunct` - All, Netware
ispunct, iswpunct

iswpunct - All, Netware
Synopsis:

#include <ctype.h>
int isspace( int c );

#include <wctype.h>
int iswspace( wint_t c );

Description:
The isspace function tests for the following white-space characters:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>' '</td>
<td>space</td>
</tr>
<tr>
<td>'\f'</td>
<td>form feed</td>
</tr>
<tr>
<td>'\n'</td>
<td>new-line or linefeed</td>
</tr>
<tr>
<td>'\r'</td>
<td>carriage return</td>
</tr>
<tr>
<td>'\t'</td>
<td>horizontal tab</td>
</tr>
<tr>
<td>'\v'</td>
<td>vertical tab</td>
</tr>
</tbody>
</table>

The iswspace function is similar to isspace except that it accepts a wide-character argument.

Returns:
The isspace function returns a non-zero character when the argument is one of the indicated white-space characters. The iswspace function returns a non-zero value when the argument is a wide character that corresponds to a standard white-space character or is one of an implementation-defined set of wide characters for which iswalnum is false. Otherwise, zero is returned.

See Also:
  isalnum, isalpha, isblank, iscntrl, isdigit, isgraph, isleadbyte, islower, isprint, ispunct, isupper, iswalnum, iswctype, isxdigit, tolower, toupper, towctrans

Example:

```c
#include <stdio.h>
#include <ctype.h>

char chars[] = {
    'A',
    0x09,
    ' ',
    0x7d
};

#define SIZE sizeof( chars ) / sizeof( char )

void main()
{
    int i;

    for( i = 0; i < SIZE; i++ ) {
        printf( "Char \%c is %sa space character\n",
            chars[i],
            ( isspace( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:
isspace, iswspace

Char A is not a space character
Char   is a space character
Char   is a space character
Char } is not a space character

Classification: isspace is ANSI
                   iswspace is ANSI

Systems:          isspace - All, Netware
                   iswspace - All, Netware
Synopsis: #include <ctype.h>
  int isupper( int c);
#include <wctype.h>
  int iswupper( wint_t c);

Description: The isupper function tests for any uppercase letter 'A' through 'Z'.

The iswupper function is similar to isupper except that it accepts a wide-character argument.

Returns: The isupper function returns a non-zero value when the argument is an uppercase letter. The
iswupper function returns a non-zero value when the argument is a wide character that corresponds
to an uppercase letter, or if it is one of an implementation-defined set of wide characters for which none
of iswcntrl, iswdigit, iswpunct, or iswspace is true. Otherwise, zero is returned.

See Also: isalnum, isalpha, isblank, iscntrl, isdigit, isgraph, isleadbyte, islower,
isprint, ispunct, isspace, iswctype, isxdigit, tolower, toupper, towctrans

Example: #include <stdio.h>
#include <ctype.h>

  char chars[] = {
    'A',
    'a',
    'z',
    'Z'
  };

  #define SIZE sizeof( chars ) / sizeof( char )

  void main()
  {
    int i;

    for( i = 0; i < SIZE; i++ ) {
      printf( "Char %c is %san uppercase character\n",
        chars[i],
        ( isupper( chars[i] ) ? "" : "not " ) );
    }
  }

produces the following:

  Char A is an uppercase character
  Char a is not an uppercase character
  Char z is not an uppercase character
  Char Z is an uppercase character

Classification: isupper is ANSI
               iswupper is ANSI

Systems: isupper - All, Netware
         iswupper - All, Netware
Synopsis:
#include <wctype.h>
int iswctype( wint_t wc, wctype_t desc );

Description:
The iswctype function determines whether the wide character wc has the property described by desc. Valid values of desc are defined by the use of the wctype function.

The twelve expressions listed below have a truth-value equivalent to a call to the wide character testing function shown.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>iswctype(wc, wctype(&quot;alnum&quot;))</td>
<td>iswalnum(wc)</td>
</tr>
<tr>
<td>iswctype(wc, wctype(&quot;alpha&quot;))</td>
<td>iswalpha(wc)</td>
</tr>
<tr>
<td>iswctype(wc, wctype(&quot;blank&quot;))</td>
<td>iswblank(wc)</td>
</tr>
<tr>
<td>iswctype(wc, wctype(&quot;ctrl&quot;))</td>
<td>iswctrl(wc)</td>
</tr>
<tr>
<td>iswctype(wc, wctype(&quot;digit&quot;))</td>
<td>iswdigit(wc)</td>
</tr>
<tr>
<td>iswctype(wc, wctype(&quot;graph&quot;))</td>
<td>iswgraph(wc)</td>
</tr>
<tr>
<td>iswctype(wc, wctype(&quot;lower&quot;))</td>
<td>iswlower(wc)</td>
</tr>
<tr>
<td>iswctype(wc, wctype(&quot;print&quot;))</td>
<td>iswprint(wc)</td>
</tr>
<tr>
<td>iswctype(wc, wctype(&quot;punct&quot;))</td>
<td>iswpunct(wc)</td>
</tr>
<tr>
<td>iswctype(wc, wctype(&quot;space&quot;))</td>
<td>iswspace(wc)</td>
</tr>
<tr>
<td>iswctype(wc, wctype(&quot;upper&quot;))</td>
<td>iswupper(wc)</td>
</tr>
<tr>
<td>iswctype(wc, wctype(&quot;xdigit&quot;))</td>
<td>iswxdigit(wc)</td>
</tr>
</tbody>
</table>

Returns: The iswctype function returns non-zero (true) if and only if the value of the wide character wc has the property described by desc.

See Also: isalnum, isalpha, isblank, iscntrl, isdigit, isgraph, isleadbyte, islower, isprint, ispunct, isspace, isupper, isxdigit, tolower, toupper, towctrans

Example:
```c
#include <stdio.h>
#include <wctype.h>

char *types[] = {
    "alnum",
    "alpha",
    "blank",
    "cntrl",
    "digit",
    "graph",
    "lower",
    "print",
    "punct",
    "space",
    "upper",
    "xdigit"
};

void main( void )
{
    int     i;
    wint_t  wc = 'A';

    for( i = 0; i < 12; i++ )
        if( iswctype( wc, wctype( types[i] ) ) )
            printf( "%s\n", types[i] );
}

produces the following:

alnum
alpha
graph
print
upper
xdigit

Classification:  ANSI

Systems:  All
isxdigit, iswxdigit

Synopsis:

```
#include <cctype>
int isxdigit( int c );
#include <wctype.h>
int iswxdigit( wint_t c );
```

Description:

The `isxdigit` function tests for any hexadecimal-digit character. These characters are the digits ('0' through '9') and the letters ('a' through 'f') and ('A' through 'F').

The `iswxdigit` function is similar to `isxdigit` except that it accepts a wide-character argument.

Returns:

The `isxdigit` function returns a non-zero value when the argument is a hexadecimal-digit character.

The `iswxdigit` function returns a non-zero value when the argument is a wide character that corresponds to a hexadecimal-digit character. Otherwise, zero is returned.

See Also:

`isalnum`, `isalpha`, `isblank`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `iswctype`, `tolower`, `toupper`, `towctrans`

Example:

```
#include <stdio.h>
#include <ctype.h>

cchar chars[] = {
   'A',
   '5',
   '$'
};

#define SIZE sizeof( chars ) / sizeof( char )

void main()
{
    int i;

    for( i = 0; i < SIZE; i++ ) {
        printf( "Char %c is %sa hexadecimal digit character\n", char[i],
                ( isxdigit( chars[i] ) ) ? "" : "not " );
    }
}
```

produces the following:

Char A is a hexadecimal digit character
Char 5 is a hexadecimal digit character
Char $ is not a hexadecimal digit character

Classification:

`isxdigit` is ANSI
`iswxdigit` is ANSI

Systems:

`isxdigit` - All, Netware
`iswxdigit` - All, Netware

500 Library Functions and Macros
Synopsis:
#include <stdlib.h>
char *itoa( int value, char *buffer, int radix );
char *_itoa( int value, char *buffer, int radix );
wchar_t *_itow( int value, wchar_t *buffer, 
    int radix );

Description: The itoa function converts the binary integer value into the equivalent string in base radix notation storing the result in the character array pointed to by buffer. A null character is appended to the result. The size of buffer must be at least (8 * sizeof(int) + 1) bytes when converting values in base 2. That makes the size 17 bytes on 16-bit machines, and 33 bytes on 32-bit machines. The value of radix must satisfy the condition:

2 <= radix <= 36

If radix is 10 and value is negative, then a minus sign is prepended to the result.

The _itoa function is identical to itoa. Use _itoa for ANSI/ISO naming conventions.

The _itow function is identical to itoa except that it produces a wide-character string (which is twice as long).

Returns: The itoa function returns the pointer to the result.

See Also: atoi, atol, atoll, ltoa, lltoa, sscanf, strtol,strtoll, strtoul, strtoull, strtoimax, strtoimax, ultoa, ulltoa, utoa

Example:
#include <stdio.h>
#include <stdlib.h>

void main()
{
    char buffer[20];
    int base;

    for( base = 2; base <= 16; base = base + 2 )
        printf( "%2d %s\n", base,
            itoa( 12765, buffer, base ) );
}

produces the following:

  2 11000111011101
  4 3013131
  6 135033
  8 30735
 10 12765
 12 7479
 14 491b
 16 31dd

Classification: WATCOM
    _itoa conforms to ANSI/ISO naming conventions

Systems: itoa - All, Netware
itoa, _itoa, _itow

_itoa - All, Netware
_itow - All
Synopsis: 
#include <conio.h>
int kbhit( void );
int _kbhit( void );

Description: The kbhit function tests whether or not a keystroke is currently available. When one is available, the function getch or getche may be used to obtain the keystroke in question.

With a stand-alone program, the kbhit function may be called continuously until a keystroke is available.

The _kbhit function is identical to kbhit. Use _kbhit for ANSI/ISO naming conventions.

Returns: The kbhit function returns zero when no keystroke is available; otherwise, a non-zero value is returned.

See Also: getch, getche, putch, ungetch

Example: 
/*
 * This program loops until a key is pressed
 * or a count is exceeded.
 */
#include <stdio.h>
#include <conio.h>

void main( void )
{
    unsigned long i;

    printf( "Program looping. Press any key.\n" );
    for( i = 0; i < 10000; i++ ) {
        if( kbhit( ) ) {
            getch();
            break;
        }
    }
}

Classification: WATCOM
_kbhit conforms to ANSI/ISO naming conventions

Systems: kbhit - All, Netware
_kbhit - All, Netware
Synopsis: 
#include <stdlib.h>
long int labs( long int j );

Description: The labs function returns the absolute value of its long-integer argument j.

Returns: The labs function returns the absolute value of its argument.

See Also: abs, llabs, imaxabs, fabs

Example: 
#include <stdio.h>
#include <stdlib.h>

void main( void )
{
    long x, y;

    x = -50000L;
    y = labs( x );
    printf( "labs(%ld) = %ld\n", x, y );
}

produces the following:

labs(-50000) = 50000

Classification: ISO C90

Systems: All, Netware
Synopsis:
#include <math.h>
double ldexp( double x, int exp );

Description: The ldexp function multiplies a floating-point number by an integral power of 2. A range error may occur.

Returns: The ldexp function returns the value of \( x \) times 2 raised to the power \( exp \).

See Also: frexp, modf

Example:
#include <stdio.h>
#include <math.h>

void main()
{
    double value;
    value = ldexp( 4.7072345, 5 );
    printf( "%f\n", value );
}

produces the following:

150.631504

Classification: ANSI

Systems: Math
### Synopsis:

```c
#include <stdlib.h>
ldiv_t ldiv( long int numer, long int denom );
```

```c
typedef struct {
    long int quot;     /* quotient */
    long int rem;      /* remainder */
} ldiv_t;
```

### Description:
The `ldiv` function calculates the quotient and remainder of the division of the numerator `numer` by the denominator `denom`.

### Returns:
The `ldiv` function returns a structure of type `ldiv_t` that contains the fields `quot` and `rem`, which are both of type `long int`.

### See Also:
`div`, `lldiv`, `imaxdiv`

### Example:
```c
#include <stdio.h>
#include <stdlib.h>

void print_time( long int ticks )
{
    ldiv_t sec_ticks;
    ldiv_t min_sec;

    sec_ticks = ldiv( ticks, 100L );
    min_sec   = ldiv( sec_ticks.quot, 60L );
    printf( "It took %ld minutes and %ld seconds\n", min_sec.quot, min_sec.rem );
}

void main( void )
{
    print_time( 86712L );
}
```

produces the following:

```
It took 14 minutes and 27 seconds
```

### Classification:
ISO C90

### Systems:
All, Netware
Synopsis:  
#include <search.h>
void *lfind( const void *key, /* object to search for */
            const void *base, /* base of search data */
            unsigned *num,   /* number of elements */
            unsigned width,  /* width of each element */
            int (*compare)( const void *element1,
                           const void *element2 ) );

Description: The lfind function performs a linear search for the value key in the array of num elements pointed to by base. Each element of the array is width bytes in size. The argument compare is a pointer to a user-supplied routine that will be called by lfind to determine the relationship of an array element with the key. One of the arguments to the compare function will be an array element, and the other will be key.

The compare function should return 0 if element1 is identical to element2 and non-zero if the elements are not identical.

Returns: The lfind function returns a pointer to the array element in base that matches key if it is found, otherwise NULL is returned indicating that the key was not found.

See Also: bsearch, lsearch

Example:  
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <search.h>

static const char *keywords[] = {
        "auto",
        "break",
        "case",
        "char",
        /* . */
        /* . */
        /* . */
        "while"
};

void main( int argc, const char *argv[] )
{
    unsigned num = 5;
    extern int compare( const void *, const void * );

    if( argc <= 1 ) exit( EXIT_FAILURE );
    if( lfind( &argv[1], keywords, &num, sizeof(char **),
               compare ) == NULL ) {
        printf( "'\%s' is not a C keyword\n", argv[1] );
        exit( EXIT_FAILURE );
    } else {
        printf( "'\%s' is a C keyword\n", argv[1] );
        exit( EXIT_SUCCESS );
    }
}
int compare( const void *op1, const void *op2 )
{
    const char **p1 = (const char **) op1;
    const char **p2 = (const char **) op2;
    return( strcmp( *p1, *p2 ) );
}

Classification: WATCOM
Systems: All, Netware
Synopsis:    
#include <graph.h>
short _FAR _lineto( short x, short y );
short _FAR _lineto_w( double x, double y );

Description:    The _lineto functions draw straight lines. The _lineto function uses the view coordinate system. The _lineto_w function uses the window coordinate system.

The line is drawn from the current position to the point at the coordinates (x,y). The point (x,y) becomes the new current position. The line is drawn with the current plotting action using the current line style and the current color.

Returns:    The _lineto functions return a non-zero value when the line was successfully drawn; otherwise, zero is returned.

See Also:    _moveto, _setcolor, _setlinestyle, _setplotaction

Example:    #include <conio.h>
#include <graph.h>

main()
{
  _setvideomode( _VRES16COLOR );
  _moveto( 100, 100 );
  _lineto( 540, 100 );
  _lineto( 320, 380 );
  _lineto( 100, 100 );
  getch();
  _setvideomode( _DEFAULTMODE );
}

produces the following:
Classification: PC Graphics

Systems: _lineto - DOS, QNX
        _lineto_w - DOS, QNX
Synopsis:  
#include <stdlib.h>  
long long int llabs( long long int j );

Description:  The llabs function returns the absolute value of its long long integer argument j.

Returns:  The llabs function returns the absolute value of its argument.

See Also:  abs, imaxabs, fabs

Example:  
#include <stdio.h>  
#include <stdlib.h>

void main( void )
{
    long long x, y;

    x = -5000000000;  
y = llabs( x );
    printf( "llabs(%lld) = %lld\n", x, y );
}

produces the following:

llabs(-5000000000) = 5000000000

Classification:  ISO C99
Synopsis:  
#include <stdlib.h>

```
lldiv_t lldiv( long long int numer,  
             long long int denom );
```

typedef struct {
   long long int quot; /* quotient */
   long long int rem;  /* remainder */
} lldiv_t;

Description:  The lldiv function calculates the quotient and remainder of the division of the numerator `numer` by the denominator `denom`.

Returns:  The lldiv function returns a structure of type `lldiv_t` that contains the fields `quot` and `rem`, which are both of type `long long int`.

See Also:  `div`, `imaxdiv`

Example:  
```
#include <stdio.h>  
#include <stdlib.h>

void print_time( long long int ticks )
{
   lldiv_t sec_ticks;  
   lldiv_t min_sec;

   sec_ticks = lldiv( ticks, 100 );
   min_sec   = lldiv( sec_ticks.quot, 60 );
   printf( "It took %lld minutes and %lld seconds\n",
           min_sec.quot, min_sec.rem );
}

void main( void )
{
   print_time( 73495132 );
}
```

produces the following:

```
It took 12249 minutes and 11 seconds
```

Classification:  ISO C99
Synopsis:

```c
#include <locale.h>
struct lconv *localeconv( void );
```

Description: The `localeconv` function sets the components of an object of type `struct lconv` with values appropriate for the formatting of numeric quantities according to the current locale. The components of the `struct lconv` and their meanings are as follows:

<table>
<thead>
<tr>
<th>Component</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>char *decimal_point</code></td>
<td>The decimal-point character used to format non-monetary quantities.</td>
</tr>
<tr>
<td><code>char *thousands_sep</code></td>
<td>The character used to separate groups of digits to the left of the decimal-point character in formatted non-monetary quantities.</td>
</tr>
<tr>
<td><code>char *grouping</code></td>
<td>A string whose elements indicate the size of each group of digits in formatted non-monetary quantities.</td>
</tr>
<tr>
<td><code>char *int_curr_symbol</code></td>
<td>The international currency symbol applicable to the current locale. The first three characters contain the alphabetic international currency symbol in accordance with those specified in ISO 4217 Codes for the Representation of Currency and Funds. The fourth character (immediately preceding the null character) is the character used to separate the international currency symbol from the monetary quantity.</td>
</tr>
<tr>
<td><code>char *currency_symbol</code></td>
<td>The local currency symbol applicable to the current locale.</td>
</tr>
<tr>
<td><code>char *mon_decimal_point</code></td>
<td>The decimal-point character used to format monetary quantities.</td>
</tr>
<tr>
<td><code>char *mon_thousands_sep</code></td>
<td>The character used to separate groups of digits to the left of the decimal-point character in formatted monetary quantities.</td>
</tr>
<tr>
<td><code>char *mon_grouping</code></td>
<td>A string whose elements indicate the size of each group of digits in formatted monetary quantities.</td>
</tr>
<tr>
<td><code>char *positive_sign</code></td>
<td>The string used to indicate a nonnegative-valued monetary quantity.</td>
</tr>
<tr>
<td><code>char *negative_sign</code></td>
<td>The string used to indicate a negative-valued monetary quantity.</td>
</tr>
<tr>
<td><code>char int_frac_digits</code></td>
<td>The number of fractional digits (those to the right of the decimal-point) to be displayed in an internationally formatted monetary quantity.</td>
</tr>
<tr>
<td><code>char frac_digits</code></td>
<td>The number of fractional digits (those to the right of the decimal-point) to be displayed in a formatted monetary quantity.</td>
</tr>
<tr>
<td><code>char p_cs_precedes</code></td>
<td>Set to 1 or 0 if the <code>currency_symbol</code> respectively precedes or follows the value for a nonnegative formatted monetary quantity.</td>
</tr>
<tr>
<td><code>char p_sep_by_space</code></td>
<td>Set to 1 or 0 if the <code>currency_symbol</code> respectively is or is not separated by a space from the value for a nonnegative formatted monetary quantity.</td>
</tr>
<tr>
<td><code>char n_cs_precedes</code></td>
<td>Set to 1 or 0 if the <code>currency_symbol</code> respectively precedes or follows the value for a negative formatted monetary quantity.</td>
</tr>
</tbody>
</table>
**localeconv**

*char n_sep_by_space* Set to 1 or 0 if the currency_symbol respectively is or is not separated by a space from the value for a negative formatted monetary quantity.

*char p_sign_posn* The position of the positive_sign for a nonnegative formatted monetary quantity.

*char n_sign_posn* The position of the positive_sign for a negative formatted monetary quantity.

The elements of grouping and mon_grouping are interpreted according to the following:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR_MAX</td>
<td>No further grouping is to be performed.</td>
</tr>
<tr>
<td>0</td>
<td>The previous element is to be repeatedly used for the remainder of the digits.</td>
</tr>
<tr>
<td>other</td>
<td>The value is the number of digits that comprise the current group. The next element is examined to determine the size of the next group of digits to the left of the current group.</td>
</tr>
</tbody>
</table>

The value of p_sign_posn and n_sign_posn is interpreted as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Parentheses surround the quantity and currency_symbol.</td>
</tr>
<tr>
<td>1</td>
<td>The sign string precedes the quantity and currency_symbol.</td>
</tr>
<tr>
<td>2</td>
<td>The sign string follows the quantity and currency_symbol.</td>
</tr>
<tr>
<td>3</td>
<td>The sign string immediately precedes the quantity and currency_symbol.</td>
</tr>
<tr>
<td>4</td>
<td>The sign string immediately follows the quantity and currency_symbol.</td>
</tr>
</tbody>
</table>

**Returns:** The localeconv function returns a pointer to the filled-in object.

**See Also:** setlocale

**Example:**

```c
#include <stdio.h>
#include <locale.h>

void main()
{
    struct lconv *lc;

    lc = localeconv();
    printf( "*decimal_point (%s)\n",
            lc->decimal_point );

    printf( "*thousands_sep (%s)\n",
            lc->thousands_sep );
}
```

514 Library Functions and Macros
printf( "*int_curr_symbol (%s)\n",
    lc->int_curr_symbol );

printf( "*currency_symbol (%s)\n",
    lc->currency_symbol );

printf( "*mon_decimal_point (%s)\n",
    lc->mon_decimal_point );

printf( "*mon_thousands_sep (%s)\n",
    lc->mon_thousands_sep );

printf( "*mon_grouping (%s)\n",
    lc->mon_grouping );

printf( "*grouping (%s)\n",
    lc->grouping );

printf( "*positive_sign (%s)\n",
    lc->positive_sign );

printf( "*negative_sign (%s)\n",
    lc->negative_sign );

printf( "int_frac_digits (%d)\n",
    lc->int_frac_digits );

printf( "frac_digits (%d)\n",
    lc->frac_digits );

printf( "p_cs_precedes (%d)\n",
    lc->p_cs_precedes );

printf( "p_sep_by_space (%d)\n",
    lc->p_sep_by_space );

printf( "n_cs_precedes (%d)\n",
    lc->n_cs_precedes );

printf( "n_sep_by_space (%d)\n",
    lc->n_sep_by_space );

printf( "p_sign_posn (%d)\n",
    lc->p_sign_posn );

printf( "n_sign_posn (%d)\n",
    lc->n_sign_posn );
}

Classification: ANSI

Systems: All, Netware
localtime Functions

Synopsis:  

```
#include <time.h>
struct tm * localtime( const time_t *timer );
struct tm * _localtime( const time_t *timer, 
                        struct tm *tmbuf );
```

```
struct tm {
    int tm_sec;   /* seconds after the minute -- [0,61] */
    int tm_min;   /* minutes after the hour   -- [0,59] */
    int tm_hour;  /* hours after midnight  -- [0,23] */
    int tm_mday;  /* day of the month      -- [1,31] */
    int tm_mon;   /* months since January -- [0,11] */
    int tm_year;  /* years since 1900      */
    int tm_wday;  /* days since Sunday   -- [0,6] */
    int tm_yday;  /* days since January 1-- [0,365] */
    int tm_isdst; /* Daylight Savings Time flag */
};
```

Safer C:  The Safer C Library extension provides the localtime_s function which is a safer alternative to 
localtime. This newer localtime_s function is recommended to be used instead of the 
traditional "unsafe" localtime function.

Description: The localtime functions convert the calendar time pointed to by timer into a structure of type tm, of 
time information, expressed as local time. Whenever localtime is called, the tzset function is 
also called.

The calendar time is usually obtained by using the time function. That time is Coordinated Universal 
Time (UTC) (formerly known as Greenwich Mean Time (GMT)).

The _localtime function places the converted time in the tm structure pointed to by tmbuf; and the 
localtime function places the converted time in a static structure that is re-used each time 
localtime is called.

The time set on the computer with the DOS time command and the DOS date command reflects the 
local time. The environment variable TZ is used to establish the time zone to which this local time 
applies. See the section The TZ Environment Variable for a discussion of how to set the time zone.

Returns: The localtime functions return a pointer to a tm structure containing the time information.

See Also: asctime Functions, asctime_s, clock, ctime Functions, ctime_s, difftime, localtime, 
gmtime_s, localtime_s, mktime, strftime, time, tzset

Example:  

```
#include <stdio.h>
#include <time.h>

void main()
{
    time_t time_of_day;
    auto char buf[26];
    auto struct tm tmbuf;

    time_of_day = time( NULL );
    _localtime( &time_of_day, &tmbuf );
    printf( "It is now: %s", _asctime( &tmbuf, buf ) );
}
```
produces the following:

It is now: Sat Mar 21 15:58:27 1987

Classification: localtime is ANSI
   _localtime is not ANSI

Systems:    localtime - All, Netware
   _localtime - All
Synopsis:  
#define __STDC_WANT_LIB_EXT1__ 1
#include <time.h>
struct tm * localtime_s( const time_t * restrict timer,
                        struct tm * restrict result);

struct tm {
    int tm_sec;   /* seconds after the minute -- [0,61] */
    int tm_min;   /* minutes after the hour   -- [0,59] */
    int tm_hour;  /* hours after midnight -- [0,23] */
    int tm_mday;  /* day of the month    -- [1,31] */
    int tm_mon;   /* months since January -- [0,11] */
    int tm_year;  /* years since 1900        */
    int tm_wday;  /* days since Sunday   -- [0,6]  */
    int tm_yday;  /* days since January 1  -- [0,365]*/
    int tm_isdst; /* Daylight Savings Time flag */
};

Constraints:  
If any of the following runtime-constraints is violated, the currently active runtime-constraint handler
will be invoked and localtime_s will return a non-zero value to indicate an error, or the
runtime-constraint handler aborts the program.

Neither timer nor result shall be a null pointer. If there is a runtime-constraint violation, there is no
attempt to convert the time.

Description:  
The localtime_s function converts the calendar time pointed to by timer into a broken-down time,
expressed as local time. The broken-down time is stored in the structure pointed to by result.

Returns:  
The localtime_s function returns result, or a null pointer if the specified time cannot be converted
to local time or there is a runtime-constraint violation.

See Also:  
asctime Functions, asctime_s, clock, ctime Functions, ctime_s, difftime, gmtime,
gmtime_s, localtime, mktime, strftime, time, tzset

Example:  
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <time.h>

void main()
{
    time_t time_of_day;
    auto char buf[26];
    auto struct tm tmbuf;

    time_of_day = time( NULL );
    localtime_s( &time_of_day, &tmbuf );
    asctime_s( buf, sizeof( buf ), &tmbuf );
    printf( "It is now: %s", buf );
}

produces the following:

It is now: Mon Jan 30 15:28:33 2006

Classification: TR 24731
Systems: DOS, Windows, Win32, OS/2 1.x(all), OS/2-32, Netware
Synopsis: 

```c
#include <io.h>
int lock( int handle,
         unsigned long offset,
         unsigned long nbytes );
```

Description: The lock function locks nbytes amount of data in the file designated by handle starting at byte offset in the file. This prevents other processes from reading or writing into the locked region until an unlock has been done for this locked region of the file.

Multiple regions of a file can be locked, but no overlapping regions are allowed. You cannot unlock multiple regions in the same call, even if the regions are contiguous. All locked regions of a file should be unlocked before closing a file or exiting the program.

With DOS, locking is supported by version 3.0 or later. Note that SHARE.COM or SHARE.EXE must be installed.

Returns: The lock function returns zero if successful, and -1 when an error occurs. When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also: locking, open, sopen, unlock

Example:

```c
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

void main()
{
    int handle;
    char buffer[20];

    handle = open( "file", O_RDWR | O_TEXT );
    if( handle != -1 ) {
        if( lock( handle, 0L, 20L ) ) {
            printf( "Lock failed\n" );
        } else {
            read( handle, buffer, 20 );
            /* update the buffer here */
            lseek( handle, 0L, SEEK_SET );
            write( handle, buffer, 20 );
            unlock( handle, 0L, 20L );
        }
    } else {
        close( handle );
    }
}
```

Classification: WATCOM

Systems: All, Netware
Synopsis:  
#include <sys\locking.h>
int locking( int handle, int mode, long nbyte );
int _locking( int handle, int mode, long nbyte );

Description:  The locking function locks or unlocks nbyte bytes of the file specified by handle. Locking a region of a file prevents other processes from reading or writing the locked region until the region has been unlocked. The locking and unlocking takes place at the current file position. The argument mode specifies the action to be performed. The possible values for mode are:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>_LK_LOCK, LK_LOCK</td>
<td>Locks the specified region. The function will retry to lock the region after 1 second intervals until successful or until 10 attempts have been made.</td>
</tr>
<tr>
<td>_LK_RLCK, LK_RLCK</td>
<td>Same action as _LK_LOCK.</td>
</tr>
<tr>
<td>_LK_NBLCK, LK_NBLCK</td>
<td>Non-blocking lock: makes only 1 attempt to lock the specified region.</td>
</tr>
<tr>
<td>_LK_NBRLCK, LK_NBRLCK</td>
<td>Same action as _LK_NBLCK.</td>
</tr>
<tr>
<td>_LK_UNLCK, LK_UNLCK</td>
<td>Unlocks the specified region. The region must have been previously locked.</td>
</tr>
</tbody>
</table>

Multiple regions of a file can be locked, but no overlapping regions are allowed. You cannot unlock multiple regions in the same call, even if the regions are contiguous. All locked regions of a file should be unlocked before closing a file or exiting the program.

With DOS, locking is supported by version 3.0 or later. Note that SHARE.COM or SHARE.EXE must be installed.

The _locking function is identical to locking. Use _locking for ANSI/ISO naming conventions.

Returns:  The locking function returns zero if successful. Otherwise, it returns -1 and errno is set to indicate the error.

Errors:  When an error has occurred, errno contains a value indicating the type of error that has been detected.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EACCES</td>
<td>Indicates a locking violation (file already locked or unlocked).</td>
</tr>
<tr>
<td>EBADF</td>
<td>Indicates an invalid file handle.</td>
</tr>
<tr>
<td>EDEADLOCK</td>
<td>Indicates a locking violation. This error is returned when mode is _LK_LOCK or LK_RLCK and the file cannot be locked after 10 attempts.</td>
</tr>
<tr>
<td>EINVAL</td>
<td>Indicates that an invalid argument was given to the function.</td>
</tr>
</tbody>
</table>

See Also: creat, _dos_creat, _dos_open, lock, open, sopen, unlock
locking, _locking

Example:

```c
#include <stdio.h>
#include <sys\locking.h>
#include <share.h>
#include <fcntl.h>
#include <io.h>

void main()
{
    int handle;
    unsigned nbytes;
    unsigned long offset;
    auto char buffer[512];

    nbytes = 512;
    offset = 1024;
    handle = sopen( "db.fil", O_RDWR, SH_DENYNO );
    if( handle != -1 ) {
        lseek( handle, offset, SEEK_SET );
        locking( handle, LK_LOCK, nbytes );
        read( handle, buffer, nbytes );
        /* update data in the buffer */
        lseek( handle, offset, SEEK_SET );
        write( handle, buffer, nbytes );
        lseek( handle, offset, SEEK_SET );
        locking( handle, LK_UNLCK, nbytes );
        close( handle );
    }
}
```

Classification: WATCOM

_locking conforms to ANSI/ISO naming conventions

Systems: locking - All
         _locking - All
Synopsis:    #include <math.h>
            double log( double x );

Description:  The log function computes the natural logarithm (base e) of x. A domain error occurs if the argument
              is negative. A range error occurs if the argument is zero.

Returns:      The log function returns the natural logarithm of the argument. When the argument is outside the
              permissible range, the matherr function is called. Unless the default matherr function is replaced,
              it will set the global variable errno to EDOM, and print a "DOMAIN error" diagnostic message using
              the stderr stream.

See Also:     exp, log10, log2, pow, matherr

Example:      #include <stdio.h>
              #include <math.h>

              void main()
              {
                  printf( "%f\n", log(.5) );
              }

              produces the following:

                  -0.693147

Classification:  ANSI

Systems:  Math

Library Functions and Macros  523
Synopsis:    #include <math.h>
            double log10( double x );

Description: The log10 function computes the logarithm (base 10) of x. A domain error occurs if the argument is negative. A range error occurs if the argument is zero.

Returns:    The log10 function returns the logarithm (base 10) of the argument. When the argument is outside the permissible range, the matherr function is called. Unless the default matherr function is replaced, it will set the global variable errno to EDOM, and print a "DOMAIN error" diagnostic message using the stderr stream.

See Also:    exp, log, log2, pow, matherr

Example:    #include <stdio.h>
            #include <math.h>

            void main()
            {
                printf( "%f\n", log10(.5) );
            }

            produces the following:
            -0.301030

Classification: ANSI

Systems:    Math
Synopsis:  
#include <math.h>
double log2( double x );

Description:  The log2 function computes the logarithm (base 2) of x. A domain error occurs if the argument is negative. A range error occurs if the argument is zero.

Returns:  The log2 function returns the logarithm (base 2) of the argument. When the argument is outside the permissible range, the matherr function is called. Unless the default matherr function is replaced, it will set the global variable errno to EDOM, and print a “DOMAIN error” diagnostic message using the stderr stream.

See Also:  exp, log, log10, pow, matherr

Example:  
#include <stdio.h>  
#include <math.h>  

void main()  
{  
    printf( "%f\n", log2(.25) );  
}

produces the following:

-2.000000

Classification:  WATCOM

Systems:  Math
**Synopsis:**

```c
#include <setjmp.h>
void longjmp( jmp_buf env, int return_value );
```

**Description:**

The `longjmp` function restores the environment saved by the most recent call to the `setjmp` function with the corresponding `jmp_buf` argument.

It is generally a bad idea to use `longjmp` to jump out of an interrupt function or a signal handler (unless the signal was generated by the `raise` function).

**Returns:**

After the `longjmp` function restores the environment, program execution continues as if the corresponding call to `setjmp` had just returned the value specified by `return_value`. If the value of `return_value` is 0, the value returned is 1.

**See Also:**

`setjmp`

**Example:**

```c
#include <stdio.h>
#include <setjmp.h>

jmp_buf env;

rtn()
{
    printf( "about to longjmp\n" );
    longjmp( env, 14 );
}

void main()
{
    int ret_val = 293;
    if( 0 == ( ret_val = setjmp( env ) ) ) {
        printf( "after setjmp %d\n", ret_val );
        rtn();
        printf( "back from rtn %d\n", ret_val );
    } else {
        printf( "back from longjmp %d\n", ret_val );
    }
}
```

produces the following:

```
after setjmp 0
about to longjmp
back from longjmp 14
```

**Classification:** ANSI

**Systems:** All, Netware
Synopsis:   
#include <stdlib.h>
unsigned long _lrotl( unsigned long value, unsigned int shift );

Description:  The _lrotl function rotates the unsigned long integer, determined by value, to the left by the number of bits specified in shift.

Returns:  The rotated value is returned.

See Also:  _lrotr, _rotl, _rotr

Example:   
#include <stdio.h>
#include <stdlib.h>

unsigned long mask = 0x12345678;

void main()
{
    mask = _lrotl( mask, 4 );
    printf( "%08lX\n", mask );
}

produces the following:

23456781

Classification:  WATCOM

Systems:  All, Netware
Synopsis: 
#include <stdlib.h>
unsigned long _lrotr( unsigned long value, 
unsigned int shift );

Description: The _lrotr function rotates the unsigned long integer, determined by value, to the right by the number of bits specified in shift.

Returns: The rotated value is returned.

See Also:  _lrotl, _rotl, _rotr

Example: 
#include <stdio.h>
#include <stdlib.h>

unsigned long mask = 0x12345678;

void main()
{
    mask = _lrotr( mask, 4 );
    printf( "%08lX\n", mask );
}

produces the following:

81234567

Classification: WATCOM

Systems: All, Netware

528  Library Functions and Macros
Synopsis:
#include <search.h>
void *lsearch( const void *key, /* object to search for */
    void *base,      /* base of search data */
    unsigned *num,   /* number of elements */
    unsigned width,  /* width of each element*/
    int (*compare)( const void *element1,
        const void *element2 ) );

Description:
The lsearch function performs a linear search for the value key in the array of num elements pointed
to by base. Each element of the array is width bytes in size. The argument compare is a pointer to a
user-supplied routine that will be called by lsearch to determine the relationship of an array element
with the key. One of the arguments to the compare function will be an array element, and the other will
be key.

The compare function should return 0 if element1 is identical to element2 and non-zero if the elements
are not identical.

Returns:
If the key value is not found in the array, then it is added to the end of the array and the number of
elements is incremented. The lsearch function returns a pointer to the array element in base that
matches key if it is found, or the newly added key if it was not found.

See Also:
bsearch, lfind

Example:
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <search.h>

void main( int argc, const char *argv[] )
{
    int i;
    unsigned num = 0;
    char **array = (char **)calloc( argc, sizeof(char **) );
    extern int compare( const void *, const void * );

    for( i = 1; i < argc; ++i ) {
        lsearch( &argv[i], array, &num, sizeof(char **),
            compare );
    }
    for( i = 0; i < num; ++i ) {
        printf( "%s\n", array[i] );
    }
}

int compare ( const void *op1, const void *op2 )
{
    const char **p1 = (const char **) op1;
    const char **p2 = (const char **) op2;
    return( strcmp( *p1, *p2 ) );
}

/* With input: one two one three four */

produces the following:
Isearch

one
two
three
four

Classification: WATCOM

Systems: All, Netware
Synopsis:  
#include <stdio.h>  
#include <io.h>  
off_t lseek( int handle, off_t offset, int origin );
off_t _lseek( int handle, off_t offset, int origin );
__int64 _lseeki64( int handle, __int64 offset, int origin );

Description:  The lseek function sets the current file position at the operating system level.  The file is referenced using the file handle handle returned by a successful execution of one of the creat, dup, dup2, open or sopen functions.  The value of offset is used as a relative offset from a file position determined by the value of the argument origin.

The new file position is determined in a manner dependent upon the value of origin which may have one of three possible values (defined in the <stdio.h> header file):

<table>
<thead>
<tr>
<th>Origin</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEEK_SET</td>
<td>The new file position is computed relative to the start of the file. The value of offset must not be negative.</td>
</tr>
<tr>
<td>SEEK_CUR</td>
<td>The new file position is computed relative to the current file position. The value of offset may be positive, negative or zero.</td>
</tr>
<tr>
<td>SEEK_END</td>
<td>The new file position is computed relative to the end of the file.</td>
</tr>
</tbody>
</table>

An error will occur if the requested file position is before the start of the file.

The requested file position may be beyond the end of the file. On POSIX-conforming systems, if data is later written at this point, subsequent reads of data in the gap will return bytes whose value is equal to zero until data is actually written in the gap. On systems such DOS and OS/2 that are not POSIX-conforming, data that are read in the gap have arbitrary values.

Some versions of MS-DOS allow seeking to a negative offset, but it is not recommended since it is not supported by other platforms and may not be supported in future versions of MS-DOS.

The lseek function does not, in itself, extend the size of a file (see the description of the chsize function).

The _lseek function is identical to lseek. Use _lseek for ANSI/ISO naming conventions.

The _lseeki64 function is identical to lseek except that it accepts a 64-bit value for the offset argument.

The lseek function can be used to obtain the current file position (the tell function is implemented in terms of lseek). This value can then be used with the lseek function to reset the file position to that point in the file:
off_t file_posn;
int handle;

/* get current file position */
file_posn = lseek( handle, 0L, SEEK_CUR );
/* or */
file_posn = tell( handle );

/* return to previous file position */
file_posn = lseek( handle, file_posn, SEEK_SET );

If all records in the file are the same size, the position of the n’th record can be calculated and read, as illustrated in the example included below. The function in this example assumes records are numbered starting with zero and that rec_size contains the size of a record in the file (including the record-separator character). (including the carriage-return character in text files).

**Returns:** If successful, the current file position is returned in a system-dependent manner. A value of 0 indicates the start of the file.

If an error occurs in lseek, (-1L) is returned.

If an error occurs in _lseeki64, (-1I64) is returned.

When an error has occurred, errno contains a value indicating the type of error that has been detected.

**Errors:** When an error has occurred, errno contains a value indicating the type of error that has been detected.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBADF</td>
<td>The handle argument is not a valid file handle.</td>
</tr>
<tr>
<td>EINVAL</td>
<td>The origin argument is not a proper value, or the resulting file offset would be invalid.</td>
</tr>
</tbody>
</table>

**See Also:** chsize, close, creat, dup, dup2, eof, exec..., fdopen, filelength, fileno, fstat, _grow_handles, isatty, open, read, setmode, sopen, stat, tell, write, umask

**Example:**
```c
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

int read_record( int handle,
                long rec_numb,
                int rec_size,
                char *buffer )

    { if( lseek( handle, rec_numb * rec_size, SEEK_SET )
        == -1L ) { return( -1 ); }
        return( read( handle, buffer, rec_size ) );
    }
```
void main( void )
{
    int handle;
    int size_read;
    char buffer[80];

    /* open a file for input */
    handle = open( "file", O_RDONLY | O_TEXT );
    if( handle != -1 ) {

        /* read a piece of the text */
        size_read =
            read_record( handle, 1, 80, buffer );

        /* test for error */
        if( size_read == -1 ) {
            printf( "Error reading file\n" );
        } else {
            printf( "%80s\n", buffer );
        }

        /* close the file */
        close( handle );
    }
}

Classification: lseek is POSIX 1003.1
    _lseek is not POSIX
    _lseeki64 is not POSIX
    _lseek conforms to ANSI/ISO naming conventions

Systems: lseek - All, Netware
    _lseek - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
    _lseeki64 - All
Iltoa, _lltoa, _lltow

Synopsis:

```c
#include <stdlib.h>

char *lltoa( long long int value,
            char *buffer,
            int radix );

char *_lltoa( long long int value,
             char *buffer,
             int radix );

wchar_t * _lltow( long long int value,
                 wchar_t *buffer,
                 int radix );
```

Description: The `lltoa` function converts the binary integer `value` into the equivalent string in base `radix` notation storing the result in the character array pointed to by `buffer`. A null character is appended to the result. The size of `buffer` must be at least 65 bytes when converting values in base 2. The value of `radix` must satisfy the condition:

\[ 2 \leq \text{radix} \leq 36 \]

If `radix` is 10 and `value` is negative, then a minus sign is prepended to the result.

The `_lltoa` function is identical to `lltoa`. Use `_lltoa` for ANSI/ISO naming conventions.

The `_lltow` function is identical to `lltoa` except that it produces a wide-character string (which is twice as long).

Returns: The `lltoa` function returns a pointer to the result.

See Also: `atoi`, `atol`, `atoll`, `itoa`, `ltoa`, `sscanf`, `strtol`, `strtoll`, `strto1ull`, `strtoimax`, `strto1umax`, `ultoa`, `ulltoa`, `utoa`

Example:

```c
#include <stdio.h>
#include <stdlib.h>

void print_value( long value )
{
    int base;
    char buffer[65];

    for( base = 2; base <= 16; base = base + 2 )
        printf( "%2d %s\n", base,
                lltoa( value, buffer, base ) );
}

void main()
{
    print_value( 1234098765LL );
}
```

produces the following:
Classification: WATCOM
   _lltoa conforms to ANSI/ISO naming conventions

Systems:     lltoa   - All, Netware
             _lltoa  - All, Netware
             _lltow  - All
**Synopsis:**

```c
#include <stdlib.h>
char *ltoa( long int value,  
    char *buffer,  
    int radix );
char * _ltoa( long int value,  
    char *buffer,  
    int radix );
wchar_t * _ltow( long int value,  
    wchar_t *buffer,  
    int radix );
```

**Description:**

The `ltoa` function converts the binary integer `value` into the equivalent string in base `radix` notation storing the result in the character array pointed to by `buffer`. A null character is appended to the result. The size of `buffer` must be at least 33 bytes when converting values in base 2. The value of `radix` must satisfy the condition:

```
2 <= radix <= 36
```

If `radix` is 10 and `value` is negative, then a minus sign is prepended to the result.

The `_ltoa` function is identical to `ltoa`. Use `_ltoa` for ANSI/ISO naming conventions.

The `_ltow` function is identical to `ltoa` except that it produces a wide-character string (which is twice as long).

**Returns:**

The `ltoa` function returns a pointer to the result.

**See Also:** `atoi, atol, atoll, itoa, lltoa, sscanf, strtol, strtoll, strtoul, strtoull, strtoimax, strtoumax, ultoa, ulltoa, utoa`

**Example:**

```c
#include <stdio.h>
#include <stdlib.h>

void print_value( long value )
{
    int base;
    char buffer[33];

    for( base = 2; base <= 16; base = base + 2 )
        printf( "%2d %s
", base,  
            ltoa( value, buffer, base ) );
}

void main()
{
    print_value( 12765L );
}
```

produces the following:
2 11000111011101
4 3013131
6 135033
8 30735
10 12765
12 7479
14 491b
16 31dd

Classification: WATCOM
_ltoa conforms to ANSI/ISO naming conventions

Systems: ltoa - All, Netware
        _ltoa - All, Netware
        _ltow - All
**Synopsis:**

```c
int main( void );
int main( int argc, const char *argv[] );
int wmain( void );
int wmain( int argc, wchar_t *argv[] );
int PASCAL WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow );
int PASCAL wWinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance, wcharT *lpszCmdLine, int nCmdShow );
```

**Description:**

*main* is a user-supplied function where program execution begins. The command line to the program is broken into a sequence of tokens separated by blanks and are passed to *main* as an array of pointers to character strings in the parameter *argv*. The number of arguments found is passed in the parameter *argc*. The first element of *argv* will be a pointer to a character string containing the program name. The last element of the array pointed to by *argv* will be a NULL pointer (i.e. *argv*[argc] will be NULL). Arguments that contain blanks can be passed to *main* by enclosing them within double quote characters (which are removed from that element in the *argv* vector. A literal double quote character can be passed by preceding it with a backslash. A literal backslash followed by an enclosing double quote character can be passed as a pair of backslash characters and a double quote character.

**Example:**

```c
  echo "he"l\lo world\""
```

passes the single argument "he"llo world"

The command line arguments can also be obtained in its original format by using the `getcmd` function.

Alternatively, the *main* function can be declared to return `void` (i.e., no return value). In this case, you will not be able to return an exit code from *main* using a `return` statement but must use the `exit` function to do so.

The *wmain* function is a user-defined wide-character version of *main* that operates with wide-character strings. If this function is present in the application, then it will be called by the run-time system startup code (and the *main* function, if present, will not be called).

As with *main*, the *wmain* function can be declared to return `void` and the same considerations will apply.

The *WinMain* function is called by the system as the initial entry point for a Windows-based application. The *wWinMain* function is a wide-character version of *WinMain*.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>hInstance</em></td>
<td>Identifies the current instance of the application.</td>
</tr>
<tr>
<td><em>hPrevInstance</em></td>
<td>Identifies the previous instance of the application. For an application written for Win32, this parameter is always NULL.</td>
</tr>
<tr>
<td><em>lpszCmdLine</em></td>
<td>Points to a null-terminated string specifying the command line for the application.</td>
</tr>
<tr>
<td><em>nCmdShow</em></td>
<td>Specifies how the window is to be shown. This parameter can be one of the following values:</td>
</tr>
</tbody>
</table>

538  Library Functions and Macros
<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW_HIDE</td>
<td>Hides the window and activates another window.</td>
</tr>
<tr>
<td>SW_MINIMIZE</td>
<td>Minimizes the specified window and activates the top-level window in the system’s list.</td>
</tr>
<tr>
<td>SW_RESTORE</td>
<td>Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position (same as SW_SHOWNORMAL).</td>
</tr>
<tr>
<td>SW_SHOW</td>
<td>Activates a window and displays it in its current size and position.</td>
</tr>
<tr>
<td>SW_SHOWMAXIMIZED</td>
<td>Activates a window and displays it as a maximized window.</td>
</tr>
<tr>
<td>SW_SHOWMINIMIZED</td>
<td>Activates a window and displays it as an icon.</td>
</tr>
<tr>
<td>SW_SHOWMINNOACTIVE</td>
<td>Displays a window as an icon. The active window remains active.</td>
</tr>
<tr>
<td>SW_SHOWNA</td>
<td>Displays a window in its current state. The active window remains active.</td>
</tr>
<tr>
<td>SW_SHOWNOACTIVATE</td>
<td>Displays a window in its most recent size and position. The active window remains active.</td>
</tr>
<tr>
<td>SW_SHOWNORMAL</td>
<td>Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position (same as SW_RESTORE).</td>
</tr>
</tbody>
</table>

The `WinMain` function initializes an application, and then performs a message retrieval-and-dispatch loop that is the top-level control structure for the remainder of the application’s execution. The loop terminates when a WM_QUIT message is received. At that point, `WinMain` exits the application, returning the value passed in the WM_QUIT message’s wParam parameter. If WM_QUIT was received as a result of calling `PostQuitMessage`, the value of wParam is the value of the `PostQuitMessage` function’s nExitCode parameter.

**Returns:** The main and wmain functions return an exit code to the calling program (usually the operating system).

If the WinMain function terminates before entering the message loop, it should return 0. Otherwise, it should terminate when it receives a WM_QUIT message and return the exit value contained in that message’s wParam parameter.

**See Also:** abort, atexit, _bgetcmd, exec..., exit, _Exit, _exit, getcmd, getenv, onexit, putenv, spawn..., system

**Example:**
#include <stdio.h>

int main( int argc, char *argv[] )
{
    int i;
    for( i = 0; i < argc; ++i ) {
        printf( "argv[%d] = %s\n", i, argv[i] );
    }
    return( 0 );
}
#endif

int wmain( int wargc, wchar_t *wargv[] )
{
    int i;
    for( i = 0; i < wargc; ++i ) {
        wprintf( L"wargv[%d] = %s\n", i, wargv[i] );
    }
    return( 0 );
}
#endif

produces the following:

argv[0] = C:\WATCOM\DEMO\MYPGM.EXE
argv[1] = hhhhh
argv[2] = another arg

when the program mypgm is executed with the command

mypgm hhhhh   "another arg"

A sample Windows main program is shown below.

int PASCAL WinMain( HANDLE this_inst, HANDLE prev_inst,
                    LPSTR cmdline, int cmdshow )
{
    MSG msg;

    if( !prev_inst ) {
        if( !FirstInstance( this_inst ) ) return( 0 );
    }
    if( !AnyInstance( this_inst, cmdshow ) ) return( 0 );
    /*
     * GetMessage returns FALSE when WM_QUIT is received
     */
    while( GetMessage( &msg, NULL, NULL, NULL ) ) {
        TranslateMessage( &msg );
        DispatchMessage( &msg );
    }
    return( msg.wParam );
}

Classification: main is ANSI
wmain is not ANSI
WinMain is not ANSI
wWinMain is not ANSI
Systems:

main - All, Netware
wmain - Win32, OS/2-32
WinMain - Windows, Win386, Win32
wWinMain - Win32
Synopsis:  

```c
#include <stdlib.h>

void _makepath( char *path,
    const char *drive,
    const char *dir,
    const char *fname,
    const char *ext );

void _wmakepath( wchar_t *path,
    const wchar_t *drive,
    const wchar_t *dir,
    const wchar_t *fname,
    const wchar_t *ext );
```

Description:  

The \_makepath function constructs a full pathname from the components consisting of a drive letter, directory path, file name and file name extension. The full pathname is placed in the buffer pointed to by the argument \textit{path}.

The \_wmakepath function is a wide-character version of \_makepath that operates with wide-character strings.

The maximum size required for each buffer is specified by the manifest constants \_MAX\_PATH, \_MAX\_DRIVE, \_MAX\_DIR, \_MAX\_FNAME, and \_MAX\_EXT which are defined in \texttt{<stdlib.h>}.

\textit{drive}  
The \textit{drive} argument points to a buffer containing the drive letter (A, B, C, etc.) followed by an optional colon. The \_makepath function will automatically insert a colon in the full pathname if it is missing. If \textit{drive} is a NULL pointer or points to an empty string, no drive letter or colon will be placed in the full pathname.

\textit{dir}  
The \textit{dir} argument points to a buffer containing just the pathname. Either forward slashes (/) or backslashes (/) may be used. The trailing slash is optional. The \_makepath function will automatically insert a trailing slash in the full pathname if it is missing. If \textit{dir} is a NULL pointer or points to an empty string, no slash will be placed in the full pathname.

\textit{fname}  
The \textit{fname} argument points to a buffer containing the base name of the file without any extension (suffix).

\textit{ext}  
The \textit{ext} argument points to a buffer containing the filename extension or suffix. A leading period (.) is optional. The \_makepath routine will automatically insert a period in the full pathname if it is missing. If \textit{ext} is a NULL pointer or points to an empty string, no period will be placed in the full pathname.

Returns:  
The \_makepath function returns no value.

See Also:  
\_fullpath, \_splitpath

Example:
#include <stdio.h>
#include <stdlib.h>

void main()
{
    char full_path[ _MAX_PATH ];
    char drive[ _MAX_DRIVE ];
    char dir[ _MAX_DIR ];
    char fname[ _MAX_FNAME ];
    char ext[ _MAX_EXT ];

    _makepath(full_path,"c","watcomc\h\","stdio","h");
    printf( "Full path is: %s\n\n", full_path );
    _splitpath( full_path, drive, dir, fname, ext );
    printf( "Components after _splitpath\n" );
    printf( "drive: %s\n", drive );
    printf( "dir:   %s\n", dir );
    printf( "fname: %s\n", fname );
    printf( "ext:   %s\n", ext );
}

produces the following:

Full path is: c:watcomc\h\stdio.h

Components after _splitpath

drive: c:
dir: watcomc\h\nfname: stdio
ext: .h

Note the use of two adjacent backslash characters (\) within character-string constants to signify a single backslash.

Classification: WATCOM

Systems:  _makepath - All, Netware
          _wmakepath - All
malloc Functions

Synopsis:  
#include <stdlib.h>  For ANSI compatibility (malloc only)  
#include <malloc.h>  Required for other function prototypes  
void *malloc( size_t size );  
void __based(void) * _bmalloc( __segment seg, size_t size );  
void __far * _fmalloc( size_t size );  
void __near * _nmalloc( size_t size );

Description:  
The malloc functions allocate space for an object of size bytes. Nothing is allocated when the size argument has a value of zero. Each function allocates memory from a particular heap, as listed below:

<table>
<thead>
<tr>
<th>Function</th>
<th>Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>malloc</td>
<td>Depends on data model of the program</td>
</tr>
<tr>
<td>_bmalloc</td>
<td>Based heap specified by seg value</td>
</tr>
<tr>
<td>_fmalloc</td>
<td>Far heap (outside the default data segment)</td>
</tr>
<tr>
<td>_nmalloc</td>
<td>Near heap (inside the default data segment)</td>
</tr>
</tbody>
</table>

In a small data memory model, the malloc function is equivalent to the _nmalloc function; in a large data memory model, the malloc function is equivalent to the _fmalloc function.

Returns:  
The malloc functions return a pointer to the start of the allocated memory. The malloc, _fmalloc and _nmalloc functions return NULL if there is insufficient memory available or if the requested size is zero. The _bmalloc function returns _NULLOFF if there is insufficient memory available or if the requested size is zero.

See Also:  calloc Functions, _expand Functions, free Functions, halloc, hfree, _msize Functions, realloc Functions, sbrk

Example:  
#include <stdlib.h>  
void main()
{
    char *buffer;
    
    buffer = (char *)malloc( 80 );
    if( buffer != NULL ) {
        /* body of program */
        free( buffer );
    }
}

Classification:  malloc is ANSI  
_fmalloc is not ANSI  
_bmalloc is not ANSI  
_nmalloc is not ANSI

Systems:  malloc - All, Netware
_bmalloc - DOS/16, Windows, QNX/16, OS/2 1.x(all)
_fmalloc - DOS/16, Windows, QNX/16, OS/2 1.x(all)
_nmalloc - DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2 1.x(MT), OS/2-32
Synopsis:  
```c
#include <math.h>
int matherr( struct _exception *err_info );
```

Description:  
The `matherr` function is invoked each time an error is detected by functions in the math library. The default `matherr` function supplied in the library returns zero which causes an error message to be displayed upon `stderr` and `errno` to be set with an appropriate error value. An alternative version of this function can be provided, instead of the library version, in order that the error handling for mathematical errors can be handled by an application.

A program may contain a user-written version of `matherr` to take any appropriate action when an error is detected. When zero is returned, an error message will be printed upon `stderr` and `errno` will be set as was the case with the default function. When a non-zero value is returned, no message is printed and `errno` is not changed. The value `err_info->retval` is used as the return value for the function in which the error was detected.

The `matherr` function is passed a pointer to a structure of type `struct _exception` which contains information about the error that has been detected:

```c
struct _exception
{ int type;       /* TYPE OF ERROR */
  char *name;     /* NAME OF FUNCTION */
  double arg1;    /* FIRST ARGUMENT TO FUNCTION */
  double arg2;    /* SECOND ARGUMENT TO FUNCTION */
  double retval;  /* DEFAULT RETURN VALUE */
};
```

The `type` field will contain one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOMAIN</td>
<td>A domain error has occurred, such as <code>sqrt(-1e0)</code>.</td>
</tr>
<tr>
<td>SING</td>
<td>A singularity will result, such as <code>pow(0e0,-2)</code>.</td>
</tr>
<tr>
<td>OVERFLOW</td>
<td>An overflow will result, such as <code>pow(10e0,100)</code>.</td>
</tr>
<tr>
<td>UNDERFLOW</td>
<td>An underflow will result, such as <code>pow(10e0,-100)</code>.</td>
</tr>
</tbody>
</table>
| TLOSS  | Total loss of significance will result, such as `exp(1000)`.
| PLOSS  | Partial loss of significance will result, such as `sin(10e70)`.

The `name` field points to a string containing the name of the function which detected the error. The fields `arg1` and `arg2` (if required) give the values which caused the error. The field `retval` contains the value which will be returned by the function. This value may be changed by a user-supplied version of the `matherr` function.

Returns:  
The `matherr` function returns zero when an error message is to be printed and a non-zero value otherwise.
Example:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

/* Demonstrate error routine in which negative */
/* arguments to "sqrt" are treated as positive */

void main()
{
    printf( "%e\n", sqrt( -5e0 ) );
    exit( 0 );
}

int matherr( struct _exception *err )
{
    if( strcmp( err->name, "sqrt" ) == 0 ) {
        if( err->type == DOMAIN ) {
            err->retval = sqrt( -(err->arg1) );
            return( 1 );
        } else
            return( 0 );
    } else
        return( 0 );
}
```

Classification: WATCOM

Systems: Math
Synopsis:  
#include <stdlib.h>
#define max(a,b)  (((a) > (b)) ? (a) : (b))

Description:  The `max` macro will evaluate to be the greater of two values. It is implemented as follows.

```
#define max(a,b)  (((a) > (b)) ? (a) : (b))
```

Returns:  The `max` macro will evaluate to the larger of the two values passed.

See Also:  `min`

Example:  
```c
#include <stdio.h>
#include <stdlib.h>

void main()
{
  int a;

  /*
   * The following line will set the variable "a" to 10
   * since 10 is greater than 1.
   */
  a = max( 1, 10 );
  printf( "The value is: %d\n", a );
}
```

Classification:  WATCOM

Systems:  All, Netware
Synopsis: 

```c
#include <mbstring.h>
unsigned int _mbbtombc( unsigned int ch );
```

Description: 
The _mbbtombc function returns the double-byte character equivalent to the single-byte character `ch`. The single-byte character must be in the range 0x20 through 0x7E or 0xA1 through 0xDF.

Note: This function was called `hantozen` in earlier versions.

Returns: 
The _mbbtombc function returns `ch` if there is no equivalent double-byte character; otherwise _mbbtombc returns a double-byte character.

See Also: `_getmbcp`, `_mbcji`, `_mbcm`, `_mbctombb`, `_ismbbalnum`, `_ismbbalpha`, `_ismbbgraph`, `_ismbbk`, `_ismbbkalpha`, `_ismbbkana`, `_ismbbkprint`, `_ismbbkpunct`, `_ismbblead`, `_ismbbprint`, `_ismbbtrail`, `_mbcji`, `_mbcm`, `_mbctombb`, `_mbctype`, `_setmbcp`

Example: 

```c
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

char alphabet[] = {
    "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
};

void main()
{
    int i;
    unsigned short c;

    _setmbcp( 932 );
    for( i = 0; i < sizeof( alphabet ) - 1; i++ ) {
        c = _mbbtombc( alphabet[ i ] );
        printf( "%c%c", c>>8, c );
    }
    printf( "\n" );
}
```

produces the following:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <mbstring.h>  
#include <mbctype.h>  
int _mbbtype( unsigned char ch, int type );

Description:  
The _mbbtype function determines the type of a byte in a multibyte character. If the value of type is any value except 1, _mbbtype tests for a valid single-byte or lead byte of a multibyte character. If the value of type is 1, _mbbtype tests for a valid trail byte of a multibyte character.

Note: A similar function was called chkctype in earlier versions.

Returns:  
If the value of type is not 1, the _mbbtype function returns one of the following values:

_MBC_SINGLE   
the character is a valid single-byte character (e.g., 0x20 - 0x7E, 0xA1 - 0xDF in code page 932)

_MBC_LEAD    
the character is valid lead byte character (e.g., 0x81 - 0x9F, 0xE0 - 0xFC in code page 932)

_MBC_ILLEGAL    
the character is an illegal character (e.g., any value except 0x20 - 0x7E, 0xA1 - 0xDF, 0x81 - 0x9F, 0xE0 - 0xFC in code page 932)

If the value of type is 1, the _mbbtype function returns one of the following values:

_MBC_TRAIL    
the character is a valid trailing byte character (e.g., 0x40 - 0x7E, 0x80 - 0xFC in code page 932)

_MBC_ILLEGAL    
the character is an illegal character (e.g., any character except a valid trailing byte character)

See Also:  
_getmbcp, _ismbcalnum, _ismbcalpha, _ismbccntrl, _ismbcdigit, _ismbcgraph, _ismbchira, _ismbckata, _ismbcl0, _ismbcl1, _ismbcl2, _ismbclegal, _ismbclower, _ismbcprint, _ismbcpxdigit, _ismbcspace, _ismbcsymbol, _ismbcupper, _ismbcxdigit, _setmbcp

Example:
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

const char *types[4] = {
    "ILLEGAL",
    "SINGLE",
    "LEAD",
    "TRAIL"
};

const unsigned char chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x81,0x40, /* double-byte space */
    0x82,0x60, /* double-byte A */
    0x82,0xA6, /* double-byte Hiragana */
    0x83,0x42, /* double-byte Katakana */
    0xA1,      /* single-byte Katakana punctuation */
    0xA6,      /* single-byte Katakana alphabetic */
    0xDF,      /* single-byte Katakana alphabetic */
    0xE0,0xA1, /* double-byte Kanji */
    0x00
};

#define SIZE sizeof( chars ) / sizeof( unsigned char )

void main()
{
    int     i, j, k;

    _setmbcp( 932 );
    k = 0;
    for( i = 0; i < SIZE; i++ ) {
        j = _mbbtype( chars[i], k );
        printf( "%s
", types[ 1 + j ] );
        if( j == _MBC_LEAD )
            k = 1;
        else
            k = 0;
    }
}

produces the following:

Library Functions and Macros  551
_mbbtype

SINGLE
SINGLE
SINGLE
SINGLE
LEAD
TRAIL
LEAD
TRAIL
LEAD
TRAIL
LEAD
TRAIL
SINGLE
SINGLE
SINGLE
LEAD
TRAIL
ILLEGAL

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <mbstring.h>
int _mbccmp( const unsigned char *s1,  
const unsigned char *s2 );
int _fmbccmp( const unsigned char __far *s1,  
const unsigned char __far *s2 );

Description: The _mbccmp function compares one multibyte character from s1 to one multibyte character from s2.

The _fmbccmp function is a data model independent form of the _mbccmp function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The _mbccmp and _fmbccmp functions return the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 0</td>
<td>multibyte character at s1 less than multibyte character at s2</td>
</tr>
<tr>
<td>0</td>
<td>multibyte character at s1 identical to multibyte character at s2</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>multibyte character at s1 greater than multibyte character at s2</td>
</tr>
</tbody>
</table>

See Also: _mbccpy, _mbcicmp, _mbcjistojms, _mbcjstojis, _mbclen, _mbctohira, _mbctokata, _mbctolower, _mbctombb, _mbctoupper, mblen, mbrlen, mbtowc, mbsrtowcs, mbsrtowcs_s, mbstowcs, mbstowcs_s, mbtolower, mbtoupper, wcrtomb, wcrtomb_s, wcsrtombs, wcsrtombs_s, wcstombs, wcstombs_s, wc aficionado, wcstombs_s

Example: 
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned char mb1[2] = {
  0x81, 0x43
};

unsigned char mb2[2] = {
  0x81, 0x42
};

void main()
{
  int i;
  _setmbcp( 932 );
  i = _mbccmp( mb1, mb2 );
  if( i < 0 )
    printf( "Less than\n" );
  else if( i == 0 )
    printf( "Equal to\n" );
  else
    printf( "Greater than\n" );
}

produces the following:
Greater than

Classification:
- _mbccmp is ANSI
- _mbccmp is not ANSI
- _fmbccmp is not ANSI

Systems:
- _mbccmp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
- _fmbccmp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

```c
#include <mbstring.h>

void _mbccpy( unsigned char *dest,
              const unsigned char *ch );

void _fmbccpy( unsigned char ___far *dest,
               const unsigned char ___far *ch );
```

Description:

The `_mbccpy` function copies one multibyte character from `ch` to `dest`.

The `_fmbccpy` function is a data model independent form of the `_mbccpy` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns:

The `_mbccpy` function does not return a value.

See Also:

`_mbccmp`, `_mbicmp`, `_mbcjiostojms`, `_mbcjmstojis`, `_mbclen`, `_mbctohira`,
`_mbctokata`, `_mbctolower`, `_mbctombb`, `_mbctoupper`, `mblen`, `mbrlen`, `mbrtowc`,
`mbstowcs`, `mbstowcs_s`, `mbsrtowcs`, `mbsrtowcs_s`, `mbstowcs_s`, `mbtowc`, `wctomb`,
`wcrtomb_s`, `wcrtombs`, `wcrtombs_s`, `wcsrtombs`, `wcsrtombs_s`, `wcstombs`, `wcstombs_s`, `wctob`, `wctomb`,
`wctomb_s`

Example:

```c
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned char mb1[2] = {
  0x00, 0x00
};

unsigned char mb2[4] = {
  0x81, 0x42, 0x81, 0x41
};

void main()
{
    _setmbcp( 932 );
    printf( "\%6.4x\n", mb1[0] << 8 | mb1[1] );
    _mbccpy( mb1, mb2 );
    printf( "\%6.4x\n", mb1[0] << 8 | mb1[1] );
}
```

produces the following:

```
0000
0x8142
```

Classification: WATCOM

Systems:

- `_mbccpy` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
- `_fmbccpy` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

```c
#include <mbstring.h>

int _mbcicmp( const unsigned char *s1,
              const unsigned char *s2 );

int _fmbcicmp( const unsigned char __far *s1,
               const unsigned char __far *s2 );
```

Description:
The `_mbcicmp` function compares one multibyte character from `s1` to one multibyte character from `s2` using a case-insensitive comparison.

The `_fmbcicmp` function is a data model independent form of the `_mbcicmp` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns:
The `_mbcicmp` and `_fmbcicmp` functions return the following values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 0</td>
<td>multibyte character at <code>s1</code> less than multibyte character at <code>s2</code></td>
</tr>
<tr>
<td>0</td>
<td>multibyte character at <code>s1</code> identical to multibyte character at <code>s2</code></td>
</tr>
<tr>
<td>&gt; 0</td>
<td>multibyte character at <code>s1</code> greater than multibyte character at <code>s2</code></td>
</tr>
</tbody>
</table>

See Also:
- `_mbccmp`, `_mbccpy`, `_mbcjistojs`, `_mbcjstojis`, `_mbclen`, `_mbctohira`, `_mbctokata`, `_mbctolower`, `_mbctombb`, `_mbctoupper`, `mblen`, `mbrlen`, `mbtowc`, `mbstowcs`, `mbstowcs_s`, `mbstowcs`, `mbstowcs_s`, `mbtowc`, `wctomb`, `wcrtomb`, `wcsrtombs`, `wcsrtombs_s`, `wctombs`, `wctombs_s`, `wctob`, `wctomb`, `wctomb_s`

Example:

```c
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned char mb1[2] = {
    0x41, 0x42
};

unsigned char mb2[2] = {
    0x61, 0x43
};

void main()
{
    int i;
    _setmbcp( 932 );
    i = _mbcicmp( mb1, mb2 );
    if( i < 0 )
        printf( "Less than\n" );
    else if( i == 0 )
        printf( "Equal to\n" );
    else
        printf( "Greater than\n" );
}
```
produces the following:

Equal to

Classification: WATCOM

Systems:  _mbcicmp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
          _fmbcicmp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis: 

```c
#include <mbstring.h>
unsigned int _mbcjistojms( unsigned int ch );
```

Description: The _mbcjistojms converts a JIS character set code to a shift-JIS character set code. If the argument is out of range, _mbcjistojms returns 0. Valid JIS double-byte characters are those in which the first and second byte fall in the range 0x21 through 0x7E. This is summarized in the following diagram.

```
   [ 1st byte ]    [ 2nd byte ]
 0x21-0x7E       0x21-0x7E
```

Note: The JIS character set code is a double-byte character set defined by JIS, the Japan Industrial Standard Institutes. Shift-JIS is another double-byte character set. It is defined by Microsoft for personal computers and is based on the JIS code. The first byte and the second byte of JIS codes can have values less than 0x80. Microsoft has designed shift-JIS code so that it can be mixed in strings with single-byte alphanumeric codes. Thus the double-byte shift-JIS codes are greater than or equal to 0x8140.

Note: This function was called jistojms in earlier versions.

Returns: The _mbcjistojms function returns zero if the argument is not in the range otherwise, the corresponding shift-JIS code is returned.

See Also: _getmbcp, _mbbtombc, _mbcjstojis, _mbctombb, _ismbbalnum, _ismbbalpha, _ismbbgraph, _ismbbkalnum, _ismbbkalpha, _ismbbkana, _ismbbkprint, _ismbbkpunct, _ismbblead, _ismbbprint, _ismbbpunct, _ismbtrail, _mbbtombc, _mbcjstojis, _mbctombb, _mmbtype, _setmbcp

Example: 

```c
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

void main()
{
    unsigned short c;

    _setmbcp( 932 );
    c = _mbcjistojms( 0x2152 );
    printf( "\%#6.4x\n", c );
}
```

produces the following:

```
0x8171
```

Classification: WATCOM

Systems: All
Synopsis:  
#include <mbstring.h>
unsigned int _mbcjmstojis( unsigned int ch );

Description:  The _mbcjmstojis converts a shift-JIS character set code to a JIS character set code. If the argument is out of range, _mbcjmstojis returns 0. Valid shift-JIS double-byte characters are those in which the first byte falls in the range 0x81 through 0x9F or 0xE0 through 0xFC and whose second byte falls in the range 0x40 through 0x7E or 0x80 through 0xFC. This is summarized in the following diagram.

[ 1st byte ]    [ 2nd byte ]
0x81-0x9F       0x40-0xFC
or except 0x7F
0xE0-0xFC

Note: The JIS character set code is a double-byte character set defined by JIS, the Japan Industrial Standard Institutes. Shift-JIS is another double-byte character set. It is defined by Microsoft for personal computers and is based on the JIS code. The first byte and the second byte of JIS codes can have values less than 0x80. Microsoft has designed shift-JIS code so that it can be mixed in strings with single-byte alphanumeric codes. Thus the double-byte shift-JIS codes are greater than or equal to 0x8140.

Note: This function was called jmstojis in earlier versions.

Returns:  The _mbcjmstojis function returns zero if the argument is not in the range otherwise, the corresponding shift-JIS code is returned.

See Also:  _getmbcp, _mbbtombc, _mbcjistojms, _mbctombb, _ismbbalnum, _ismbbalpha, _ismbbgraph, _ismbbkalnum, _ismbbkalpha, _ismbbkprint, _ismbbkpunct, _ismbblead, _ismbbprint, _ismbbpunct, _ismbbtrail, _mbbtombc, _mbcjistojms, _mbctombb, _mbbtype, _setmbcp

Example:  
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

void main()
{
    unsigned short c;
    _setmbcp( 932 );
    c = _mbcjmstojis( 0x8171 );
    printf( "%%6.4x\n", c );
}

produces the following:

0x2152

Classification: WATCOM

Systems:  All
Synopsis:  
#include <mbstring.h>
size_t _mbclen( const unsigned char *ch );
size_t far _fmbclen( const unsigned char __far *ch );

Description:  The _mbclen function determines the number of bytes comprising the multibyte character pointed to
by ch.

The _fmbclen function is a data model independent form of the _mbclen function that accepts far
pointer arguments. It is most useful in mixed memory model applications.

Returns:  If ch is a NULL pointer, the _mbclen function returns zero if multibyte character encodings do not
have state-dependent encoding, and non-zero otherwise. If ch is not a NULL pointer, the _mbclen
function returns:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if ch points to the null character</td>
</tr>
<tr>
<td>1</td>
<td>if ch points to a single-byte character</td>
</tr>
<tr>
<td>2</td>
<td>if ch points to a double-byte character</td>
</tr>
<tr>
<td>-1</td>
<td>if ch does not point to a valid multibyte character</td>
</tr>
</tbody>
</table>

See Also: _mbccmp, _mbccpy, _mbccicmp, _mbcjistojms, _mbcjstojis, _mbctohira,
_mbctokata, _mbctolower, _mbctombb, _mbctoupper, mblen, mbrlen, mbtowc,
mbsrtowcs, mbsrtowcs_s, mbstowcs, mbstowcs_s, mbtowc, btowc, wcrtomb,
wctomb_s, wcsrtombs, wcsrtombs_s, wcstombs, wcstombs_s, wcstowc, wcstowc,
wctomb_s

Example:  
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned char chars[] = {
    ',', '.', ',',
    '1', 'A',
    0x81,0x40, /* double-byte space */
    0x82,0x60, /* double-byte A */
    0x82,0xA6, /* double-byte Hiragana */
    0x83,0x42, /* double-byte Katakana */
    0xA1,    /* single-byte Katakana punctuation */
    0xA6,    /* single-byte Katakana alphabetic */
    0xDF,    /* single-byte Katakana alphabetic */
    0xE0,0xA1, /* double-byte Kanji */
    0x00     /* null character */
};
void main()
{
    int     i, j;

    _setmbcp( 932 );
    for( i = 0; i < sizeof(chars); i += j ) {
        j = _mbclen( &chars[i] );
        printf( "%d bytes in character\n", j );
    }
}

produces the following:

1 bytes in character
1 bytes in character
1 bytes in character
1 bytes in character
1 bytes in character
2 bytes in character
2 bytes in character
2 bytes in character
2 bytes in character
1 bytes in character
1 bytes in character
1 bytes in character
1 bytes in character
2 bytes in character
1 bytes in character
1 bytes in character
2 bytes in character
1 bytes in character

Classification: WATCOM

Systems:  _mbclen - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
          _fmbclen - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
```
#include <mbstring.h>
unsigned int _mbctolower( unsigned int c );
```

Description: The _mbctolower function converts an uppercase multibyte character to an equivalent lowercase multibyte character.

For example, in code page 932, this includes the single-byte uppercase letters A-Z and the double-byte uppercase characters such that:

```
0x8260 <= c <= 0x8279
```

Note: This function was called jtolower in earlier versions.

Returns: The _mbctolower function returns the argument value if the argument is not a double-byte uppercase character; otherwise, the equivalent lowercase character is returned.

See Also: _mbccmp, _mbcncmp, _mbcscmp, _mbcscnrtol, _mbcslnrtol, _mbcslntol, _mbctokata, _mbctokata, _mbctoupper, _mbctoupper, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mbstrlen, mb strlen
produces the following:

\texttt{abcde a b c d e}

**Classification:** WATCOM

**Systems:** DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <mbstring.h>
unsigned int _mbctoupper( unsigned int c );

Description:  The _mbctoupper function converts a lowercase multibyte character to an equivalent uppercase multibyte character.

For example, in code page 932, this includes the single-byte lowercase letters a-z and the double-byte lowercase characters such that:

\[ 0x8281 \leq c \leq 0x829A \]

Note: This function was called jtoupper in earlier versions.

Returns:  The _mbctoupper function returns the argument value if the argument is not a double-byte lowercase character; otherwise, the equivalent uppercase character is returned.

See Also:  _mbccmp, _mbcpcpy, _mbicmp, _mbcicmp, _mbclen, _mbctohira, _mbctokata, _mbctolower, _mbctouppb, mblen, mbrlen, mbtowc, mbsrtowcs, mbsrtowcs_s, mbstowcs, mbstowcs_s, mbtowc, wctomb, wctombs, wcsrtombs, wcsrtombs_s, wcstombs, wcstombs_s, wctob, wctomb, wctomb_s

Example:  
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
\'a\', /* single-byte a */
\'b\', /* single-byte b */
\'c\', /* single-byte c */
\'d\', /* single-byte d */
\'e\', /* single-byte e */
0x8281, /* double-byte a */
0x8282, /* double-byte b */
0x8283, /* double-byte c */
0x8284, /* double-byte d */
0x8285 /* double-byte e */
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int   i;
    unsigned int c;

    _setmbc( 932 );
    for( i = 0; i < SIZE; i++ ) {
        c = _mbctoupper( chars[ i ] );
        if( c > 0xff )
            printf( "%c%c", c>>8, c );
        else
            printf( "%c", c );
    }
    printf( "\n" );
}
produces the following:

ABCDE A B C D E

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

```c
#include <mbstring.h>
unsigned int _mbctohira( unsigned int ch );
```

Description: The _mbctohira converts a double-byte Katakana character to a Hiragana character. A double-byte Katakana character is any character for which the following expression is true:

```
0x8340 <= ch <= 0x8396 && ch != 0x837F
```

Any Katakana character whose value is less than 0x8393 is converted to Hiragana (there are 3 extra Katakana characters that have no equivalent).

Note: The Japanese double-byte character set includes Kanji, Hiragana, and Katakana characters - both alphabetic and numeric. Kanji is the ideogram character set of the Japanese character set. Hiragana and Katakana are two types of phonetic character sets of the Japanese character set. The Hiragana code set includes 83 characters and the Katakana code set includes 86 characters.

Note: This function was called jtohira in earlier versions.

Returns: The _mbctohira function returns the argument value if the argument is not a double-byte Katakana character; otherwise, the equivalent Hiragana character is returned.

See Also: _mbccmp, _mbccpy, _mbccmp, _mbcjstojms, _mbcmstojis, _mbclen, _mbctokata, _mbctolower, _mbctombb, _mbctoupper, mblen, mbrlen, mbrtowc, mbsrtowcs, mbsrtowcs_s, mbstowcs, mbstowcs_s, mbtowc, wcrtomb, wcrtomb_s, wcsrtombs, wcsrtombs_s, wctombs, wctombs_s, wctob, wctomb, wctomb_s

Example:

```c
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
  0x8340,
  0x8364,
  0x8396
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
  int i;

  _setmbcp( 932 );
  for( i = 0; i < SIZE; i++ ) {
    printf( "\%6.4x - \%6.4x\n", 
             chars[ i ],
             _mbctohira( chars[ i ] ) );
  }
}
```

produces the following:
Classification: WATCOM

Systems: All
Synopsis:  
#include <mbstring.h>
unsigned int _mbctokata( unsigned int ch );

Description: The _mbctokata converts a double-byte Hiragana character to a Katakana character. A double-byte Hiragana character is any character for which the following expression is true:

0x829F <= c <= 0x82F1

Note: The Japanese double-byte character set includes Kanji, Hiragana, and Katakana characters - both alphabetic and numeric. Kanji is the ideogram character set of the Japanese character set. Hiragana and Katakana are two types of phonetic character sets of the Japanese character set. The Hiragana code set includes 83 characters and the Katakana code set includes 86 characters.

Note: This function was called jtokata in earlier versions.

Returns: The _mbctokata function returns the argument value if the argument is not a double-byte Hiragana character; otherwise, the equivalent Katakana character is returned.

See Also: _mbccmp, _mbccpy, _mbcicmp, _mbcjistojms, _mbcjstojis, _mbcclen, _mbctohira, _mbctolower, _mbctoupper, mblen, mbrlen, mbrtowc, mbsrtowcs, mbsrtowcs_s, mbstowcs, mbstowcs_s, mbtowc, wcrtomb, wcrtomb_s, wcsrtombs, wcsrtombs_s, wcstombs, wcstombs_s, wctomb, wctomb_s

Example:
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned int chars[] = {
0x829F,
0x82B0,
0x82F1
};

#define SIZE sizeof( chars ) / sizeof( unsigned int )

void main()
{
    int i;
    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "\%6.4x - \%6.4x\n",
            chars[ i ],
            _mbctokata( chars[ i ] ) );
    }
}

produces the following:
0x829f - 0x8340
0x82b0 - 0x8351
0x82f1 - 0x8393

Classification: WATCOM
Systems: All
Synopsis:  
#include <mbstring.h>
unsigned int _mbctombb( unsigned int ch );

Description:  The _mbctombb function returns the single-byte character equivalent to the double-byte character ch. The single-byte character will be in the range 0x20 through 0x7E or 0xA1 through 0xDF.

Note: This function was called zentohan in earlier versions.

Returns:  The _mbctombb function returns ch if there is no equivalent single-byte character; otherwise _mbctombb returns a single-byte character.

See Also:  _getmbcp, _mbbtombc, _mbcjiostojms, _mbcjstojis, _ismbbalnum, _ismbbalpha, _ismbbgraph, _ismbbkalnum, _ismbbkana, _ismbbkprint, _ismbbkpunct, _ismbblead, _ismbbprint, _ismbbpunct, _ismbbtrail, _mbbtombc, _mbcjiostojms, _mbcjstojis, _mbbtype, _setmbcp

Example:  
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

#define ZEN(x) 130*256+(x-1+32)

unsigned int alphabet[26] = {
    ZEN('A'),ZEN('B'),ZEN('C'),ZEN('D'),ZEN('E'),
    ZEN('F'),ZEN('G'),ZEN('H'),ZEN('I'),ZEN('J'),
    ZEN('K'),ZEN('L'),ZEN('M'),ZEN('N'),ZEN('O'),
    ZEN('P'),ZEN('Q'),ZEN('R'),ZEN('S'),ZEN('T'),
    ZEN('U'),ZEN('V'),ZEN('W'),ZEN('X'),ZEN('Y'),
    ZEN('Z')
};

#define SIZE sizeof( alphabet ) / sizeof( unsigned int )

void main()
{
    int i;
    unsigned int c;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        c = _mbctombb( alphabet[ i ] );
        printf( "%c", c );
    }
    printf( "\n" );
}

produces the following:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Classification:  WATCOM

Systems:  DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

570  Library Functions and Macros
Synopsis:

```
#include <mbstring.h>

unsigned char * _mbgetcode( unsigned char *mbstr,
                           unsigned int *dbchp );

unsigned char far * _fmbgetcode( unsigned char far *mbstr,
                                unsigned int *dbchp );
```

Description: The _mbgetcode function places the next single- or double-byte character from the start of the Kanji string specified by mbstr in the wide character pointed to by dbchp. If the second-half of a double-byte character is NULL, then the returned wide character is NULL.

The _fmbgetcode function is a code and data model independent form of the _mbgetcode function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

Returns: The _mbgetcode function returns a pointer to the next character to be obtained from the string. If mbstr points at a null character then mbstr is returned.

See Also: _mbsncnt, _mbputchar

Example:

```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

unsigned char set[] = {
    "ab\x81\x41\x81\x42\cd\x81"
};

void main()
{
    unsigned int c;
    unsigned char *str;

    _setmbcp( 932 );
    str = set;
    for( ; *str != '\0'; ) {
        str = _mbgetcode( str, &c );
        printf( "Character code 0x%2.2x\n", c );
    }
}
```

produces the following:

Character code 0x61
Character code 0x62
Character code 0x8141
Character code 0x8142
Character code 0x63
Character code 0x64
Character code 0x00

Classification: WATCOM

Systems: _mbgetcode - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fmbgetcode - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:    #include <stdlib.h>
            or
            #include <mbstring.h>
            int mblen( const char *s, size_t n );
            int _fmblen( const char __far *s, size_t n );

Description:    The mblen function determines the number of bytes comprising the multibyte character pointed to by
                 s. At most n bytes of the array pointed to by s will be examined.

                 The _fmblen function is a data model independent form of the mblen function. It accepts far pointer
                 arguments and returns a far pointer. It is most useful in mixed memory model applications.

Returns:    If s is a NULL pointer, the mblen function returns zero if multibyte character encodings are not state
             dependent, and non-zero otherwise. If s is not a NULL pointer, the mblen function returns:

             Value   Meaning

             0       if s points to the null character

             len     the number of bytes that comprise the multibyte character (if the next n or fewer bytes form a
                     valid multibyte character)

             -1      if the next n bytes do not form a valid multibyte character

See Also:    _mbccmp, _mbccpy, _mbccicmp, _mbcjistoijs, _mbcjstojis, _mbclen, _mbctohira,   
             _mbctokata, _mbctolower, _mbctombb, _mbctoupper, mbrlen, mbrtowc, mbsrtowcs,   
             mbsrtowcs_s, mbstowcs, mbstowcs_s, mbtowc, btowc, wcrtomb, wcrtomb_s,         
             wcsrtombs, wcsrtombs_s, wcstombs, wcstombs_s, wctob, wctomb, wctomb_s

Example:
#include <stdio.h>
#include <mbstring.h>

const char chars[] = {
  ,
  ,
  ,
  ,
  ,
  'A',
  0x81,0x40, /* double-byte space */
  0x82,0x60, /* double-byte A */
  0x82,0xA6, /* double-byte Hiragana */
  0x83,0x42, /* double-byte Katakana */
  0xA1,      /* single-byte Katakana punctuation */
  0xA6,      /* single-byte Katakana alphabetic */
  0xDF,      /* single-byte Katakana alphabetic */
  0xE0,0xA1, /* double-byte Kanji */
  0x00
};

void main()
{
  int i, j, k;
  _setmbcp( 932 );
  printf( "Character encodings are %sstate dependent\n",
    ( mblen( NULL, MB_CUR_MAX ) ) ? "" : "not " );
  j = 1;
  for( i = 0; j > 0; i += j ) {
    j = mblen( &chars[i], MB_CUR_MAX );
    printf( "%d bytes in character ", j);
    if( j == 0 ) {
      k = 0;
    } else if ( j == 1 ) {
      k = chars[i];
    } else if( j == 2 ) {
      k = chars[i]<<8 | chars[i+1];
    }
    printf( "(%#6.4x)\n", k );
  }
}

produces the following:

Character encodings are not state dependent
1 bytes in character (0x0020)
1 bytes in character (0x002e)
1 bytes in character (0x0031)
1 bytes in character (0x0041)
2 bytes in character (0x8140)
2 bytes in character (0x8260)
2 bytes in character (0x82a6)
2 bytes in character (0x8342)
1 bytes in character (0x00a1)
1 bytes in character (0x00a6)
1 bytes in character (0x00df)
2 bytes in character (0xe0a1)
0 bytes in character ( 0000)
mblen, _mblen

Classification: mblen is ANSI
   _mblen is not ANSI

Systems:     mblen - All, Netware
             _mblen - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

```c
#include <mbstring.h>

unsigned char *_mbputchar( unsigned char *mbstr,
                           unsigned int dbch );

unsigned char far *_fmbputchar( unsigned char far *mbstr,
                              unsigned int dbch );
```

Description:

The `_mbputchar` function places the next single- or double-byte character specified by `dbch` at the start of the buffer specified by `mbstr`.

The `_fmbputchar` function is a code and data model independent form of the `_mbputchar` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

Returns:

The `_mbputchar` function returns a pointer to the next location in which to store a character.

See Also:

`_mbsnccnt`, `_mbgetcode`

Example:

```c
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

void main()
{
    unsigned int c;
    unsigned char *str1;
    unsigned char *str2;
    unsigned char buf[30];

    _setmbcp( 932 );
    str1 = "ab\x82\x62\x82\x63\ef\x81\x66";
    str2 = buf;

    for( ; *str1 != '\0'; ) {
        str1 = _mbgetcode( str1, &c );
        str2 = _mbputchar( str2, '<' );
        str2 = _mbputchar( str2, c );
        str2 = _mbputchar( str2, '>' );
    }
    *str2 = '\0';
    printf( "%s\n", buf );
}
```

produces the following:

```
<a><b>< C>< D><e><f>< G>
```

Classification: WATCOM

Systems:

- `_mbputchar` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
- `_fmbputchar` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  

```c
#include <wchar.h>
int mbrlen( const char *s, size_t n, mbstate_t *ps );
int _fmbrlen( const char far *s, size_t n, mbstate_t far *ps );
```

Description:  The `mbrlen` function determines the number of bytes comprising the multibyte character pointed to by `s`. The `mbrlen` function is equivalent to the following call:

```
mbrtowc((wchar_t *)0, s, n, ps != 0 ? ps : &internal)
```

where `&internal` is the address of the internal `mbstate_t` object for the `mbrlen` function.

The `_fmbrlen` function is a data model independent form of the `mbrlen` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The restartable multibyte/wide character conversion functions differ from the corresponding internal-state multibyte character functions (`mblen, mbtowc, and wctomb`) in that they have an extra argument, `ps`, of type pointer to `mbstate_t` that points to an object that can completely describe the current conversion state of the associated multibyte character sequence. If `ps` is a null pointer, each function uses its own internal `mbstate_t` object instead. You are guaranteed that no other function in the library calls these functions with a null pointer for `ps`, thereby ensuring the stability of the state.

Also unlike their corresponding functions, the return value does not represent whether the encoding is state-dependent.

If the encoding is state-dependent, on entry each function takes the described conversion state (either internal or pointed to by `ps`) as current. The conversion state described by the pointed-to object is altered as needed to track the shift state of the associated multibyte character sequence. For encodings without state dependency, the pointer to the `mbstate_t` argument is ignored.

Returns:  The `mbrlen` function returns a value between -2 and `n`, inclusive. The `mbrlen` function returns the first of the following that applies:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if the next <code>n</code> or fewer bytes form the multibyte character that corresponds to the null wide character.</td>
</tr>
<tr>
<td>&gt;0</td>
<td>if the next <code>n</code> or fewer bytes form a valid multibyte character; the value returned is the number of bytes that constitute that multibyte character.</td>
</tr>
<tr>
<td>-2</td>
<td>if the next <code>n</code> bytes form an incomplete (but potentially valid) multibyte character, and all <code>n</code> bytes have been processed; it is unspecified whether this can occur when the value of <code>n</code> is less than that of the <code>MB_CUR_MAX</code> macro.</td>
</tr>
<tr>
<td>-1</td>
<td>if an encoding error occurs (when the next <code>n</code> or fewer bytes do not form a complete and valid multibyte character); the value of the macro <code>EILSEQ</code> will be stored in <code>errno</code>, but the conversion state will be unchanged.</td>
</tr>
</tbody>
</table>

See Also:  `_mbccmp, _mbccpy, _mbccmp, _mbcjmpstojms, _mbcjstojms, _mbclen, _mbctohira, _mbctokata, _mbctolower, _mbctombb, _mbctoupper, mblen, mbtowc, mbsrtowcs, mbsrtowcs_s, mbstowcs, mbstowcs_s, mbtowc, btowc, wctomb, wcrtomb, wcrtomb_s, wcsrtombs, wcsrtombs_s, wcstombs, wcstombs_s, wctob, wcrtomb, wcrtomb_s`
Example:
#include <stdio.h>
#include <wchar.h>
#include <mbctyp.e.h>
#include <errno.h>

const char chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x81,0x40, /* double-byte space */
    0x82,0x60, /* double-byte A */
    0x82,0xA6, /* double-byte Hiragana */
    0x83,0x42, /* double-byte Katakana */
    0xA1,     /* single-byte Katakana punctuation */
    0xA6,     /* single-byte Katakana alphabetic */
    0xDF,     /* single-byte Katakana alphabetic */
    0xE0,0xA1, /* double-byte Kanji */
    0x00
};

void main()
{
    int i, j, k;
    _setmbcp( 932 );
    j = 1;
    for( i = 0; j > 0; i += j ) {
        j = mbrlen( &chars[i], MB_CUR_MAX, NULL );
        printf( "%d bytes in character
", j );
        if( errno == EILSEQ ) {
            printf( " - illegal multibyte character\n" );
        } else {
            if( j == 0 ) {
                k = 0;
            } else if ( j == 1 ) {
                k = chars[i];
            } else if( j == 2 ) {
                k = chars[i]<<8 | chars[i+1];
            }
            printf( "(#%6.4x)\n", k );
        }
    }
}

produces the following:
mbrlen, _fmbrlen

1 bytes in character (0x0020)
1 bytes in character (0x002e)
1 bytes in character (0x0031)
1 bytes in character (0x0041)
2 bytes in character (0x8140)
2 bytes in character (0x8260)
2 bytes in character (0x82a6)
2 bytes in character (0x8342)
1 bytes in character (0x00a1)
1 bytes in character (0x00a6)
1 bytes in character (0x00df)
2 bytes in character (0xe0a1)
0 bytes in character ( 0000)

Classification: mbrlen is ANSI
_fmbrlen is not ANSI

Systems: mbrlen - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fmbrlen - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

#include <wchar.h>
int mbrtowc( wchar_t *pwc, const char *s, size_t n, mbstate_t *ps );
int _fmbrtowc( wchar_t __far *pwc, const char __far *s, size_t n, mbstate_t __far *ps );

Description: If \( s \) is a null pointer, the \texttt{mbrtowc} function determines the number of bytes necessary to enter the initial shift state (zero if encodings are not state-dependent or if the initial conversion state is described). In this case, the value of the \texttt{pwc} argument will be ignored, and the resulting state described will be the initial conversion state.

If \( s \) is not a null pointer, the \texttt{mbrtowc} function determines the number of bytes that are contained in the multibyte character (plus any leading shift sequences) pointed to by \( s \), produces the value of the corresponding wide character and then, if \( pwc \) is not a null pointer, stores that value in the object pointed to by \( pwc \). If the corresponding wide character is the null wide character, the resulting state described will be the initial conversion state.

The \texttt{fmbrtowc} function is a data model independent form of the \texttt{mbrtowc} function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The restartable multibyte/wide character conversion functions differ from the corresponding internal-state multibyte character functions (\texttt{mblen}, \texttt{mbtowc}, and \texttt{wctomb}) in that they have an extra argument, \( ps \), of type pointer to \texttt{mbstate_t} that points to an object that can completely describe the current conversion state of the associated multibyte character sequence. If \( ps \) is a null pointer, each function uses its own internal \texttt{mbstate_t} object instead. You are guaranteed that no other function in the library calls these functions with a null pointer for \( ps \), thereby ensuring the stability of the state.

Also unlike their corresponding functions, the return value does not represent whether the encoding is state-dependent.

If the encoding is state-dependent, on entry each function takes the described conversion state (either internal or pointed to by \( ps \)) as current. The conversion state described by the pointed-to object is altered as needed to track the shift state of the associated multibyte character sequence. For encodings without state dependency, the pointer to the \texttt{mbstate_t} argument is ignored.

Returns: If \( s \) is a null pointer, the \texttt{mbrtowc} function returns the number of bytes necessary to enter the initial shift state. The value returned will not be greater than that of the \texttt{MB_CUR_MAX} macro.

If \( s \) is not a null pointer, the \texttt{mbrtowc} function returns the first of the following that applies:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if the next ( n ) or fewer bytes form the multibyte character that corresponds to the null wide character.</td>
</tr>
<tr>
<td>&gt;0</td>
<td>if the next ( n ) or fewer bytes form a valid multibyte character; the value returned is the number of bytes that constitute that multibyte character.</td>
</tr>
<tr>
<td>-2</td>
<td>if the next ( n ) bytes form an incomplete (but potentially valid) multibyte character, and all ( n ) bytes have been processed; it is unspecified whether this can occur when the value of ( n ) is less than that of the \texttt{MB_CUR_MAX} macro.</td>
</tr>
</tbody>
</table>
if an encoding error occurs (when the next \( n \) or fewer bytes do not form a complete and valid multibyte character); the value of the macro \texttt{EILSEQ} will be stored in \texttt{errno}, but the conversion state will be unchanged.

See Also: \texttt{_mbccmp}, \texttt{_mbccpy}, \texttt{_mbccicmp}, \texttt{_mbcjistojms}, \texttt{_mbcjnstojis}, \texttt{_mbclen}, \texttt{_mbctohira}, \texttt{_mbctokata}, \texttt{_mbctolower}, \texttt{_mbctoupper}, \texttt{mblen}, \texttt{mbrlen}, \texttt{mbsrtowcs}, \texttt{mbsrtowcs_s}, \texttt{mbstowcs}, \texttt{mbstowcs_s}, \texttt{mbtowc}, \texttt{wcrtomb}, \texttt{wcrtomb_s}, \texttt{wcsrtombs}, \texttt{wcsrtombs_s}, \texttt{wcstombs}, \texttt{wcstombs_s}, \texttt{wctob}, \texttt{wctomb}, \texttt{wctomb_s}

Example:

```
#include <stdio.h>
#include <wchar.h>
#include <mbctype.h>
#include <errno.h>

const char chars[] = {
    ' ',
    ',',
    '.',
    '1',
    'A',
    0x81,0x40, /* double-byte space */
    0x82,0x60, /* double-byte A */
    0x82,0xA6, /* double-byte Hiragana */
    0x83,0x42, /* double-byte Katakana */
    0xA1,     /* single-byte Katakana punctuation */
    0xA6,     /* single-byte Katakana alphabetic */
    0xDF,     /* single-byte Katakana alphabetic */
    0xE0,0xA1, /* double-byte Kanji */
    0x00
};
```
void main()
{
    int     i, j, k;
    wchar_t pwc;

    _setmbcp( 932 );
    i = mbrtowc( NULL, NULL, MB_CUR_MAX, NULL );
    printf( "Number of bytes to enter initial shift state = %d\n", i );
    j = 1;
    for( i = 0; j > 0; i += j ) {
        j = mbrtowc( &pwc, &chars[i], MB_CUR_MAX, NULL );
        printf( "%d bytes in character\n" );
        if( errno == EILSEQ ) {
            printf( " - illegal multibyte character\n" );
        } else {
            if( j == 0 ) {
                k = 0;
            } else if ( j == 1 ) {
                k = chars[i];
            } else if( j == 2 ) {
                k = chars[i]<<8 | chars[i+1];
            }
            printf( "(%#6.4x->%#6.4x)\n", k, pwc );
        }
    }
}

produces the following:

Number of bytes to enter initial shift state = 0
1 bytes in character (0x0020->0x0020)
1 bytes in character (0x002e->0x002e)
1 bytes in character (0x0031->0x0031)
1 bytes in character (0x0041->0x0041)
2 bytes in character (0x8140->0x3000)
2 bytes in character (0x8260->0xff21)
2 bytes in character (0x82a6->0x3048)
2 bytes in character (0x8342->0x30a3)
1 bytes in character (0x00a1->0xff61)
1 bytes in character (0x00a6->0xff66)
1 bytes in character (0x00df->0xff9f)
2 bytes in character (0xe0a1->0x720d)
0 bytes in character ( 0000-> 0000)

Classification: mbrtowc is ANSI
                _fmbrtowc is not ANSI

Systems:   mbrtowc - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
            _fmbrtowc - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <mbstring.h>  
#include <mbctype.h> (for manifest constants)  
int _mbsbtype( const unsigned char *mbstr, int count );  
int _fmbsbtype( const unsigned char __far *mbstr,  
               int count );

Description:  The _mbsbtype function determines the type of a byte in a multibyte character string. The function  
examines only the byte at offset count in mbstr, ignoring invalid characters before the specified byte  

Note: A similar function was called nthctype in earlier versions.

Returns:  The _mbsbtype function returns one of the following values:

_MBC_SINGLE  
the character is a valid single-byte character (e.g., 0x20 - 0x7E, 0xA1 - 0xDF  
in code page 932)

_MBC_LEAD  
the character is a valid lead byte character (e.g., 0x81 - 0x9F, 0xE0 - 0xFC in  
code page 932)

_MBC_TRAIL  
the character is a valid trailing byte character (e.g., 0x40 - 0x7E, 0x80 - 0xFC  
in code page 932)

_MBC_ILLEGAL  
the character is an illegal character (e.g., any value except 0x20 - 0x7E, 0xA1  
- 0xDF, 0x81 - 0x9F, 0xE0 - 0xFC in code page 932)

See Also: _getmbcp, _ismbcalnum, _ismbcalpha, _ismbccntr1, _ismbcatidigit, _ismbcsgraph,  
_ismbchira, _ismbckata, _ismbc10, _ismbc11, _ismbc12, _ismbclegal,  
_ismbclower, _ismbcprint, _ismbcpunct, _ismbcspace, _ismbcsymbol,  
_ismbcupper, _ismbcxdigit, _mbsbtype, _setmbcp

Example:
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

const char *types[4] = {
    "ILLEGAL",
    "SINGLE",
    "LEAD",
    "TRAIL"
};

const unsigned char chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x81,0x40, /* double-byte space */
    0x82,0x60, /* double-byte A */
    0x82,0xA6, /* double-byte Hiragana */
    0x83,0x42, /* double-byte Katakana */
    0xA1,     /* single-byte Katakana punctuation */
    0xA6,     /* single-byte Katakana alphabetic */
    0xDF,     /* single-byte Katakana alphabetic */
    0xE0,0xA1, /* double-byte Kanji */
    0x00
};

#define SIZE sizeof( chars ) / sizeof( unsigned char )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ )
        printf( "%s\n", types[ 1+_mbsbtype( chars, i ) ] );
}

produces the following:

SINGLE
SINGLE
SINGLE
SINGLE
LEAD
TRAIL
LEAD
TRAIL
LEAD
TRAIL
SINGLE
SINGLE
SINGLE
LEAD
TRAIL
ILLEGAL

Library Functions and Macros  583
Classification: WATCOM

Systems:  

\_mbsbtype - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32  
\_fmbsbtype - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

```c
#include <mbstring.h>
unsigned char *_mbsnbcat( unsigned char *dst,
    const unsigned char *src,
    size_t n );
unsigned char __far *__fmbsnbcat( unsigned char __far *dst,
    const unsigned char __far *src,
    size_t n );
```

Description:
The `_mbsnbcat` function appends not more than `n` bytes of the string pointed to by `src` to the end of the string pointed to by `dst`. If the byte immediately preceding the null character in `dst` is a lead byte, the initial byte of `src` overwrites this lead byte. Otherwise, the initial byte of `src` overwrites the terminating null character at the end of `dst`. If the last byte to be copied from `src` is a lead byte, the lead byte is not copied and a null character replaces it in `dst`. In any case, a terminating null character is always appended to the result.

The `_fmbsnbcat` function is a data model independent form of the `_mbsnbcat` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

Returns:
The `_mbsnbcat` function returns the value of `dst`.

See Also:
`_mbsnbcmp`, `_mbsnbcpy`, `_mbsnbset`, `_mbsncnt`, `strncat`, `strcat`

Example:

```c
#include <stdio.h>
#include <string.h>
#include <mbctype.h>
#include <mbstring.h>

const unsigned char str1[] = {
    0x81,0x40, /* double-byte space */
    0x82,0x60, /* double-byte A */
    0x00
};

const unsigned char str2[] = {
    0x81,0x40, /* double-byte space */
    0x82,0xA6, /* double-byte Hiragana */
    0x83,0x42, /* double-byte Katakana */
    0x00
};

void main()
{
    unsigned char big_string[10];
    int i;
    _setmbcp( 932 );
    memset( (char *) big_string, 0xee, 10 );
    big_string[9] = 0x00;
    printf( "Length of string = %d\n",
        strlen( (char *) big_string ) );
    for( i = 0; i < 10; i++ )
        printf( "%2.2x \", big_string[i] );
    printf( "\n" );
}
```
```c
_mbsnset( big_string, 0x8145, 5 );
for( i = 0; i < 10; i++ )
    printf( "%2.2x ", big_string[i] );
printf( "\n" );

big_string[0] = 0x00;
_mbsncat( big_string, str1, 3 );
for( i = 0; i < 10; i++ )
    printf( "%2.2x ", big_string[i] );
printf( "\n" );

big_string[2] = 0x84;
big_string[3] = 0x00;
for( i = 0; i < 10; i++ )
    printf( "%2.2x ", big_string[i] );
printf( "\n" );

_mbsncat( big_string, str2, 5 );
for( i = 0; i < 10; i++ )
    printf( "%2.2x ", big_string[i] );
printf( "\n" );
```

produces the following:

Length of string = 9
ee ee ee ee ee ee ee ee ee 00
81 45 81 45 81 45 81 45 20 00
81 40 00 00 81 45 81 45 20 00
81 40 84 00 81 45 81 45 20 00
81 40 81 40 82 a6 00 00 20 00

Classification: WATCOM

Systems:  
_mbsncat - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fmbsncat - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

586 Library Functions and Macros
Synopsis:  
#include <mbstring.h>
int _mbsnbcmp( const unsigned char *s1,
const unsigned char *s2,
size_t n );
int _fmbsnbcmp( const unsigned char __far *s1,
const unsigned char __far *s2,
size_t n );

Description:  The _mbsnbcmp lexicographically compares not more than n bytes from the string pointed to by s1 to the string pointed to by s2.

The _fmbsnbcmp function is a data model independent form of the _mbsnbcmp function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns:  The _mbsnbcmp function returns an integer less than, equal to, or greater than zero, indicating that the string pointed to by s1 is less than, equal to, or greater than the string pointed to by s2. _mbsnbcmp is similar to _mbsncmp, except that _mbsnbcmp compares strings by bytes rather than by characters.

See Also:  _mbsnbcat, _mbsnbicmp, strncmp, strnicmp

Example:  
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

const unsigned char str1[] = {
  0x81,0x40, /* double-byte space */
  0x82,0x60, /* double-byte A */
  0x00
};

const unsigned char str2[] = {
  0x81,0x40, /* double-byte space */
  0x82,0xA6, /* double-byte Hiragana */
  0x83,0x42, /* double-byte Katakana */
  0x00
};

void main()
{
  _setmbcp( 932 );
  printf( "%d\n", _mbsnbcmp( str1, str2, 3 ) );
}

produces the following:

produces the following:

0

Classification:  WATCOM

Systems:  _mbsnbcmp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
          _fmbsnbcmp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <mbstring.h>
size_t _mbsnbcnt( const unsigned char *string, size_t n );
size_t _fmbsnbcnt( const unsigned char __far *string, 
                    size_t n );
#include <tchar.h>
size_t _strncnt( const char *string, size_t n );
size_t _wcsncnt( const wchar_t *string, size_t n ) { 

Description:  The function counts the number of bytes in the first \( n \) multibyte characters of the string \( string \).

Note:  This function was called mtob in earlier versions.

The function is a data model independent form of the \_strncnt function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The header file \tchar.h\ defines the generic-text routine \_tcsnbcnt\ . This macro maps to if \_MBCS\ has been defined, or to the \_wcsncnt macro if \_UNICODE\ has been defined. Otherwise \_tcsnbcnt maps to \_strncnt\ . \_strncnt and \_wcsncnt\ are single-byte character string and wide-character string versions of \. The \_strncnt and \_wcsncnt macros are provided only for this mapping and should not be used otherwise.

The \_strncnt function returns the number of characters (i.e., \( n \)) in the first \( n \) bytes of the single-byte string \( string \)\ . The \_wcsncnt function returns the number of bytes (i.e., \( 2 \times n \)) in the first \( n \) wide characters of the wide-character string \( string \)\ .

Returns:  The \_strncnt functions return the number of bytes in the string up to the specified number of characters or until a null character is encountered. The null character is not included in the count. If the character preceding the null character was a lead byte, the lead byte is not included in the count.

See Also:  \_mbsnbcat, \_mbsnbcnt, \_mbsnccnt

Example:  
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

const unsigned char chars[] = {
    '\',
    '\','
    ',
    '\','
    '1','
    '\A',
    0x81,0x40, /* double-byte space */
    0x82,0x60, /* double-byte A */
    0x82,0xA6, /* double-byte Hiragana */
    0x83,0x42, /* double-byte Katakana */
    0xA1, /* single-byte Katakana punctuation */
    0xA6, /* single-byte Katakana alphabetic */
    0xDF, /* single-byte Katakana alphabetic */
    0xE0,0xA1, /* double-byte Kanji */
    0x00
};
void main()
{
    _setmbcp( 932 );
    printf( "%d bytes found\n",
            _mbsnbcnt( chars, 10 ) );
}

produces the following:

14 bytes found

**Classification:** WATCOM

**Systems:**
- _mbsnbcnt - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
- _fmbsnbcnt - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
- _strncnt - MACRO
- _wcsncnt - MACRO
Synopsis:  
#include <mbstring.h>
unsigned char * _mbsncpy ( unsigned char * dst, 
    const unsigned char * src, 
    size_t n );
unsigned char __far * _fmbsncpy ( unsigned char __far * dst, 
    const unsigned char __far * src, 
    size_t n );

Description:  The _mbsncpy function copies no more than $n$ bytes from the string pointed to by $src$ into the array pointed to by $dst$. Copying of overlapping objects is not guaranteed to work properly.

If the string pointed to by $src$ is shorter than $n$ bytes, null characters are appended to the copy in the array pointed to by $dst$, until $n$ bytes in all have been written. If the string pointed to by $src$ is longer than $n$ characters, then the result will not be terminated by a null character.

The _fmbsncpy function is a data model independent form of the _mbsncpy function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

Returns:  The _mbsncpy function returns the value of $dst$.

See Also:  strcpy, strdup

Example:
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

const unsigned char chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x81,0x40, /* double-byte space */
    0x82,0x60, /* double-byte A */
    0x82,0xA6, /* double-byte Hiragana */
    0x83,0x42, /* double-byte Katakana */
    0xA1,      /* single-byte Katakana punctuation */
    0xA6,      /* single-byte Katakana alphabetic */
    0xDF,      /* single-byte Katakana alphabetic */
    0xE0,0xA1, /* double-byte Kanji */
    0x00
};

void main()
{
    unsigned char   chars2[20];
    int             i;

    _setmbscp( 932 );
    _mbsnset( chars2, 0xFF, 20 );
    _mbsnbcpy( chars2, chars, 11 );
    for( i = 0; i < 20; i++ )
        printf( "%2.2x ", chars2[i] );
    printf( "\n" );
    _mbsnbcpy( chars2, chars, 20 );
    for( i = 0; i < 20; i++ )
        printf( "%2.2x ", chars2[i] );
    printf( "\n" );
}

produces the following:

20 2e 31 41 81 80 82 60 82 a6 83 ff ff ff ff ff ff ff ff
20 2e 31 41 81 40 82 60 82 a6 83 42 a1 a6 df e0 a1 00 00

Classification: WATCOM

Systems: _mbsnbcpy - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
        _fmbsnbcpy - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
**Synopsis:**

```
#include <mbstring.h>
int _mbsnbicmp( const unsigned char *s1,
    const unsigned char *s2,
    size_t n );
int _fmbsnbicmp( const unsigned char __far *s1,
    const unsigned char __far *s2,
    size_t n );
```

**Description:**

The `_mbsnbicmp` lexicographically compares not more than `n` bytes from the string pointed to by `s1` to the string pointed to by `s2`. The comparison is insensitive to case.

The `_fmbsnbicmp` function is a data model independent form of the `_mbsnbicmp` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

**Returns:**

The `_mbsnbicmp` function returns an integer less than, equal to, or greater than zero, indicating that the string pointed to by `s1` is less than, equal to, or greater than the string pointed to by `s2`

_mbsnbicmp is similar to _mbsncmp, except that _mbsnbicmp compares strings by bytes rather than by characters.

**See Also:**

_mbsnbcat, _mbsnbcmp, strncmp, strnicmp

**Example:**

```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

const unsigned char str1[] = {
    0x81,0x40, /* double-byte space */
    0x82,0x60, /* double-byte A */
    0x82,0x79, /* double-byte Z */
    0x00
};

const unsigned char str2[] = {
    0x81,0x40, /* double-byte space */
    0x82,0x81, /* double-byte a */
    0x82,0x9a, /* double-byte z */
    0x00
};

void main()
{
    _setmbcp( 932 );
    printf( "%d\n", _mbsnbicmp( str1, str2, 5 ) );
}
```

produces the following:

0

**Classification:** WATCOM

**Systems:**

_mbsnbicmp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fmbsnbicmp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

```c
#include <mbstring.h>
unsigned char *_mbsnbset( unsigned char *str,
                           unsigned int fill,
                           size_t count );
unsigned char __far *__fmbsnbset( unsigned char __far *str,
                                   unsigned int fill,
                                   size_t count );
```

Description:
The `_mbsnbset` function fills the string `str` with the value of the argument `fill`. When the value of `len` is greater than the length of the string, the entire string is filled. Otherwise, that number of characters at the start of the string are set to the fill character.

The `_mbsnbset` is similar to `_mbsnset`, except that it fills in `count` bytes rather than `count` characters. If the number of bytes to be filled is odd and `fill` is a double-byte character, the partial byte at the end is filled with an ASCII space character.

The `_fmbsnbset` function is a data model independent form of the `_mbsnbset` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

Returns:
The address of the original string `str` is returned.

See Also:
strnset, strset

Example:
```c
#include <stdio.h>
#include <string.h>
#include <mbctype.h>
#include <mbstring.h>

void main()
{
    unsigned char   big_string[10];
    int             i;

    _setmbcp( 932 );
    memset( (char *) big_string, 0xee, 10 );
    big_string[9] = 0x00;
    for( i = 0; i < 10; i++ )
        printf( "%2.2x ", big_string[i] );
    printf( "\n" );
    _mbsnbset( big_string, 0x8145, 5 );
    for( i = 0; i < 10; i++ )
        printf( "%2.2x ", big_string[i] );
    printf( "\n" );
}
```

produces the following:

```
ee ee ee ee ee ee ee ee ee 00
81 45 81 45 20 ee ee ee 00
```

Classification: WATCOM

Systems:

- `_mbsnbset` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
- `_fmbsnbset` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

```c
#include <mbstring.h>
size_t _mbsncnt( const unsigned char *string, size_t n );
size_t _fmbsncnt( const unsigned char __far *string, 
                 size_t n );
#include <tchar.h>
size_t _strncnt( const char *string, size_t n );
size_t _wcsncnt( const wchar_t *string, size_t n ) {
```

Description:

The function counts the number of multibyte characters in the first \( n \) bytes of the string `string`. If finds a null byte as the second byte of a double-byte character, the first (lead) byte is not included in the count.

Note: This function was called `btom` in earlier versions.

The function is a data model independent form of the `strncnt` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The header file `<tchar.h>` defines the generic-text routine `tcsncnt`. This macro maps to if `MBCS` has been defined, or to the `wcsncnt` macro if `UNICODE` has been defined. Otherwise `tcsncnt` maps to `strncnt`. `strncnt` and `wcsncnt` are single-byte character string and wide-character string versions of. The `strncnt` and `wcsncnt` macros are provided only for this mapping and should not be used otherwise.

The `strncnt` function returns the number of characters (i.e., \( n \)) in the first \( n \) bytes of the single-byte string `string`. The `wcsncnt` function returns the number of bytes (i.e., \( 2 \times n \)) in the first \( n \) wide characters of the wide-character string `string`.

Returns:

`strncnt` returns the number of characters from the beginning of the string to byte \( n \). `wcsncnt` returns the number of wide characters from the beginning of the string to byte \( n \). returns the number of multibyte characters from the beginning of the string to byte \( n \). If these functions find a null character before byte \( n \), they return the number of characters before the null character. If the string consists of fewer than \( n \) characters, these functions return the number of characters in the string.

See Also:

`mbsnbcat, mbsnbcnt, mbsncnt`

Example:

```c
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

const unsigned char chars[] = {
   ', ',
   '.',
   '1',
   'A',
   0x81,0x40, /* double-byte space */
   0x82,0x60, /* double-byte A */
   0x82,0xA6, /* double-byte Hiragana */
   0x83,0x42, /* double-byte Katakana */
   0xA1,      /* single-byte Katakana punctuation */
   0xA6,      /* single-byte Katakana alphabetic */
   0xDF,      /* single-byte Katakana alphabetic */
   0xE0,0xA1, /* double-byte Kanji */
   0x00
};
```
```c
void main()
{
    _setmbcp( 932 );
    printf( "%d characters found\n",
            _mbsncnt( chars, 10 ) );
}
```

produces the following:

7 characters found

Classification: WATCOM

Systems:

- `_mbsncnt` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
- `_fmsncnt` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
- `_strncnt` - MACRO
- `_wcsncnt` - MACRO
Synopsis:

#include <mbstring.h>
unsigned int _mbsnextc( const unsigned char *string );
unsigned int _fmbsnextc( const unsigned char __far *string );
#include <tchar.h>
unsigned int _strnextc( const char *string );
unsigned int _wcsnextc( const wchar_t *string ) {

Description:

The function returns the integer value of the next multibyte-character in string, without advancing the string pointer. Recognizes multibyte character sequences according to the multibyte code page currently in use.

The header file <tchar.h> defines the generic-text routine _tcsnextc. This macro maps to if _MBCS has been defined, or to _wcsnextc if _UNICODE has been defined. Otherwise _tcsnextc maps to _strnextc. _strnextc and _wcsnextc are single-byte character string and wide-character string versions of _strnextc and _wcsnextc are provided only for this mapping and should not be used otherwise. _strnextc returns the integer value of the next single-byte character in the string. _wcsnextc returns the integer value of the next wide character in the string.

Returns:

These functions return the integer value of the next character (single-byte, wide, or multibyte) pointed to by string.

See Also:

_mbsnextc, _strdec, _strinc, _strninc

Example:

#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

const unsigned char chars[] = {
    ',',
    '.',
    '1',
    'A',
    0x81,0x40, /* double-byte space */
    0x82,0x60, /* double-byte A */
    0x82,0xA6, /* double-byte Hiragana */
    0x83,0x42, /* double-byte Katakana */
    0xA1,      /* single-byte Katakana punctuation */
    0xA6,      /* single-byte Katakana alphabetic */
    0xDF,      /* single-byte Katakana alphabetic */
    0xE0,0xA1, /* double-byte Kanji */
    0x00
};

void main()
{
    _setmbcp( 932 );
    printf( "%-6.4x\n", _mbsnextc( &chars[2] ) );
    printf( "%-6.4x\n", _mbsnextc( &chars[4] ) );
    printf( "%-6.4x\n", _mbsnextc( &chars[12] ) );
}

produces the following:
Classification: WATCOM

Systems:
  _mbsnextc - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
  _fmbsnextc - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
  _strnextc - MACRO
  _wcsnextc - MACRO
**Synopsis:**

```c
#include <wchar.h>
size_t mbsrtowcs( wchar_t *dst,
    const char **src,
    size_t len, mbstate_t *ps );
```

```c
#include <mbstring.h>
size_t _fmbsrtowcs( wchar_t __far *dst,
    const char __far *__far *src,
    size_t len, mbstate_t __far *__far *ps );
```

**Safer C:** The Safer C Library extension provides the `mbsrtowcs_s` function which is a safer alternative to `mbsrtowcs`. This newer `mbsrtowcs_s` function is recommended to be used instead of the traditional "unsafe" `mbsrtowcs` function.

**Description:** The `mbsrtowcs` function converts a sequence of multibyte characters that begins in the shift state described by `ps` from the array indirectly pointed to by `src` into a sequence of corresponding wide characters, which, if `dst` is not a null pointer, are then stored into the array pointed to by `dst`. Conversion continues up to and including a terminating null character, but the terminating null wide character will not be stored. Conversion will stop earlier in two cases: when a sequence of bytes is reached that does not form a valid multibyte character, or (if `dst` is not a null pointer) when `len` codes have been stored into the array pointed to by `dst`. Each conversion takes place as if by a call to the `mbrtowc` function.

If `dst` is not a null pointer, the pointer object pointed to by `src` will be assigned either a null pointer (if conversion stopped due to reaching a terminating null character) or the address just past the last multibyte character converted. If conversion stopped due to reaching a terminating null character and if `dst` is not a null pointer, the resulting state described will be the initial conversion state.

The `_fmbsrtowcs` function is a data model independent form of the `mbsrtowcs` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The restartable multibyte/wide string conversion functions differ from the corresponding internal-state multibyte string functions ( `mbstowcs` and `wcstombs` ) in that they have an extra argument, `ps`, of type pointer to `mbstate_t` that points to an object that can completely describe the current conversion state of the associated multibyte character sequence. If `ps` is a null pointer, each function uses its own internal `mbstate_t` object instead. You are guaranteed that no other function in the library calls these functions with a null pointer for `ps`, thereby ensuring the stability of the state.

Also unlike their corresponding functions, the conversion source argument, `src`, has a pointer-to-pointer type. When the function is storing conversion results (that is, when `dst` is not a null pointer), the pointer object pointed to by this argument will be updated to reflect the amount of the source processed by that invocation.

If the encoding is state-dependent, on entry each function takes the described conversion state (either internal or pointed to by `ps`) as current and then, if the destination pointer, `dst`, is not a null pointer, the conversion state described by the pointed-to object is altered as needed to track the shift state of the associated multibyte character sequence. For encodings without state dependency, the pointer to the `mbstate_t` argument is ignored.

**Returns:** If the input string does not begin with a valid multibyte character, an encoding error occurs: The `mbsrtowcs` function stores the value of the macro `EILSEQ` in `errno` and returns `(size_t)-1`, but the conversion state will be unchanged. Otherwise, it returns the number of multibyte characters successfully converted, which is the same as the number of array elements modified when `dst` is not a null pointer.
See Also: _mbccmp, _mbccpy, _mbccicmp, _mbcjistojms, _mbcjmstojis, _mbclen, _mbctohira, _mbctokata, _mbctolower, _mbctombb, _mbctoupper, mblen, mbrlen, mbtowc, mbsrtowcs_s, mbstowcs, mbstowcs_s, mbtowc, btowc, wcrtomb, wcrtomb_s, wcsrtombs, wcsrtombs_s, wcstombs, wcstombs_s, wctob, wctomb, wctomb_s

Example:
#include <stdio.h>
#include <wchar.h>
#include <mbctype.h>
#include <errno.h>

const char chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x81,0x40, /* double-byte space */
    0x82,0x60, /* double-byte A */
    0x82,0xA6, /* double-byte Hiragana */
    0x83,0x42, /* double-byte Katakana */
    0xA1,      /* single-byte Katakana punctuation */
    0xA6,      /* single-byte Katakana alphabetic */
    0xDF,      /* single-byte Katakana alphabetic */
    0xE0,0xA1, /* double-byte Kanji */
    0x00
};

void main()
{
    int         i;
    size_t      elements;
    const char  *src;
    wchar_t     wc[50];
    mbstate_t   pstate;

    _setmbcp( 932 );
    src = chars;
    elements = mbsrtowcs( wc, &src, 50, &pstate );
    if( errno == EILSEQ ) {
        printf( "Error in multibyte character string\n" );
    } else {
        for( i = 0; i < elements; i++ ) {
            printf( "%#6.4x\n", wc[i] );
        }
    }
}

produces the following:
mbsrtowcs, _fmbsrtowcs

0x0020
0x002e
0x0031
0x0041
0x3000
0xff21
0x3048
0x30a3
0xff61
0xff66
0xff9f
0x720d

Classification: mbsrtowcs is ANSI
_fmbsrtowcs is not ANSI

Systems: mbsrtowcs - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fmbsrtowcs - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

```c
#define __STDC_WANT_LIB_EXT1__ 1
#include <wchar.h>
errno_t mbsrtowcs_s( size_t * restrict retval,
                     wchar_t * restrict dst, rsize_t dstmax,
                     const char ** restrict src, rsize_t len,
                     mbstate_t * restrict ps);
errno_t _fmbsrtowcs_s( size_t __far * restrict retval,
                       wchar_t __far * restrict dst, rsize_t dstmax,
                       const char __far * __far * restrict src, rsize_t len,
                       mbstate_t __far * restrict ps);
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `mbsrtowcs_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

None of `retval`, `src`, `*src`, or `ps` shall be null pointers. If `dst` is not a null pointer, then neither `len` nor `dstmax` shall be greater than `RSIZE_MAX`. If `dst` is a null pointer, then `dstmax` shall equal zero. If `dst` is not a null pointer, then `dstmax` shall not equal zero. If `dst` is not a null pointer and `len` is not less than `dstmax`, then a null character shall occur within the first `dstmax` multibyte characters of the array pointed to by `*src`.

If there is a runtime-constraint violation, then `mbsrtowcs_s` does the following. If `retval` is not a null pointer, then `mbsrtowcs_s` sets `*retval` to `(size_t)(-1)`. If `dst` is not a null pointer and `dstmax` is greater than zero and less than `RSIZE_MAX`, then `mbsrtowcs_s` sets `dst[0]` to the null wide character.

Description: The `mbsrtowcs_s` function converts a sequence of multibyte characters that begins in the conversion state described by the object pointed to by `ps`, from the array indirectly pointed to by `src` into a sequence of corresponding wide characters. If `dst` is not a null pointer, the converted characters are stored into the array pointed to by `dst`. Conversion continues up to and including a terminating null character, which is also stored.

Conversion stops earlier in two cases: when a sequence of bytes is encountered that does not form a valid multibyte character, or (if `dst` is not a null pointer) when `len` wide characters have been stored into the array pointed to by `dst`. If `dst` is not a null pointer and no null wide character was stored into the array pointed to by `dst`, then `dst[len]` is set to the null wide character. Each conversion takes place as if by a call to the `mbrtowc` function.

If `dst` is not a null pointer, the pointer object pointed to by `src` is assigned either a null pointer (if conversion stopped due to reaching a terminating null character) or the address just past the last multibyte character converted (if any). If conversion stopped due to reaching a terminating null character and if `dst` is not a null pointer, the resulting state described is the initial conversion state.

Regardless of whether `dst` is or is not a null pointer, if the input conversion encounters a sequence of bytes that do not form a valid multibyte character, an encoding error occurs: the `mbsrtowcs_s` function stores the value `(size_t)(-1)` into `*retval` and the conversion state is unspecified. Otherwise, the `mbsrtowcs_s` function stores into `*retval` the number of multibyte characters successfully converted, not including the terminating null character (if any).

All elements following the terminating null wide character (if any) written by `mbsrtowcs_s` in the array of `dstmax` wide characters pointed to by `dst` take unspecified values when `mbsrtowcs_s` returns.

If copying takes place between objects that overlap, the objects take on unspecified values.
The `_fmbstowcs_s` function is a data model independent form of the `mbsrtowcs_s` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The `mbsrtowcs_s` function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

See Also: `_mbccmp`, `_mbccpy`, `_mbccicmp`, `_mbcjistojms`, `_mbcmstojis`, `_mbclen`, `_mbctohira`, `_mbctokata`, `_mbctolower`, `_mbctoupper`, `mblen`, `mbrlen`, `mbtowc`, `mbsrtowcs`, `mbstowcs`, `mbstowcs_s`, `mbtowc`, `wcrtomb`, `wcrtomb_s`, `wcsrtombs`, `wcsrtombs_s`, `wcstombs`, `wcstombs_s`, `wctob`, `wctomb`, `wctomb_s`

Example:
```c
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <wchar.h>
#include <mbctype.h>
#include <errno.h>

const char chars[] = {
    ' ',
    ',',
    '.',
    '1',
    'A',
    0x81,0x40, /* double-byte space */
    0x82,0x60, /* double-byte A */
    0x82,0xA6, /* double-byte Hiragana */
    0x83,0x42, /* double-byte Katakana */
    0xA1,      /* single-byte Katakana punctuation */
    0xA6,      /* single-byte Katakana alphabetic */
    0xDF,      /* single-byte Katakana alphabetic */
    0xE0,0xA1, /* double-byte Kanji */
    0x00
};

int main()
{
    int i;
    size_t retval;
    const char *src;
    wchar_t *wc[50];
    mbstate_t pstate;
    errno_t rc;

    _setmbcp( 932 );
    src = chars;
    rc = mbsrtowcs( &retval, wc, 50, &src, sizeof(chars), &pstate );
    if( rc != 0 ) {
        printf( "Error in multibyte character string\n" );
    } else {
        for( i = 0; i < retval; i++ ) {
            printf( "\%6.4x\n", wc[i] );
        }
    }
    return( 0 );
}
```

Classification: `mbsrtowcs_s` is TR 24731
_fmbstowcs_s is WATCOM

Systems:  
  mbsrtowcs_s - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32  
  _fmbstowcs_s - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:
#include <stdlib.h>
size_t mbstowcs( wchar_t *pwcs, const char *s, size_t n );
#include <mbstring.h>
size_t _fmbstowcs( const wchar_t __far *pwcs,
                  char __far *s,
                  size_t n );

Safer C: The Safer C Library extension provides the mbstowcs_s function which is a safer alternative to mbstowcs. This newer mbstowcs_s function is recommended to be used instead of the traditional "unsafe" mbstowcs function.

Description: The mbstowcs function converts a sequence of multibyte characters pointed to by s into their corresponding wide character codes and stores not more than n codes into the array pointed to by pwcs. The mbstowcs function does not convert any multibyte characters beyond the null character. At most n elements of the array pointed to by pwcs will be modified.

The _fmbstowcs function is a data model independent form of the mbstowcs function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: If an invalid multibyte character is encountered, the mbstowcs function returns (size_t)-1. Otherwise, the mbstowcs function returns the number of array elements modified, not including the terminating zero code if present.

See Also: mbstowcs_s, mblen, mbtowc, wctomb, wctomb_s, wcstombs, wcstombs_s

Example: #include <stdio.h>
#include <stdlib.h>

void main()
{
    char *wc = "string";
    wchar_t wbuffer[50];
    int i, len;

    len = mbstowcs( wbuffer, wc, 50 );
    if( len != -1 ) {
        wbuffer[len] = '\0';
        printf( "%s(\%d)\n", wc, len );
        for( i = 0; i < len; i++ )
            printf( "/%4.4x", wbuffer[i] );
        printf( "\n" );
    }
}

produces the following:

string(6)
/0073/0074/0072/0069/006e/0067

Classification: mbstowcs is ANSI
_fmbstowcs is not ANSI

Systems: mbstowcs - All, Netware
_fmbstowcs - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

604 Library Functions and Macros
Synopsis:
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
errno_t mbstowcs_s( size_t * restrict retval,
                    wchar_t * restrict dst,
                    rsize_t dstmax,
                    const char * restrict src, rsize_t len);
errno_t _fmbstowcs_s( size_t __far * restrict retval,
                      wchar_t __far * restrict dst,
                      rsize_t dstmax,
                      const char __far * restrict src, rsize_t len);

Constraints:
If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and mbstowcs_s will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither retval nor src shall be a null pointer. If dst is not a null pointer, then neither len nor dstmax shall be greater than RSIZE_MAX. If dst is a null pointer, then dstmax shall equal zero. If dst is not a null pointer, then dstmax shall not equal zero. If dst is not a null pointer and len is not less than dstmax, then a null character shall occur within the first dstmax multibyte characters of the array pointed to by src.

If there is a runtime-constraint violation, then mbstowcs_s does the following. If retval is not a null pointer, then mbstowcs_s sets *retval to (size_t)(-1). If dst is not a null pointer and dstmax is greater than zero and less than RSIZE_MAX, then mbstowcs_s sets dst[0] to the null wide character.

Description:
The mbstowcs_s function converts a sequence of multibyte characters that begins in the initial shift state from the array pointed to by src into a sequence of corresponding wide characters. If dst is not a null pointer, the converted characters are stored into the array pointed to by dst.

Conversion continues up to and including a terminating null character, which is also stored. Conversion stops earlier in two cases: when a sequence of bytes is encountered that does not form a valid multibyte character, or (if dst is not a null pointer) when len wide characters have been stored into the array pointed to by dst. If dst is not a null pointer and no null wide character was stored into the array pointed to by dst, then dst[len] is set to the null wide character. Each conversion takes place as if by a call to the mbtowc function.

Regardless of whether dst is or is not a null pointer, if the input conversion encounters a sequence of bytes that do not form a valid multibyte character, an encoding error occurs: the mbstowcs_s function stores the value (size_t)(-1) into *retval. Otherwise, the mbstowcs_s function stores into *retval the number of multibyte characters successfully converted, not including the terminating null character (if any).

All elements following the terminating null wide character (if any) written by mbstowcs_s in the array of dstmax wide characters pointed to by dst take unspecified values when mbstowcs_s returns.

If copying takes place between objects that overlap, the objects take on unspecified values.

The _fmbstowcs_s function is a data model independent form of the mbstowcs_s function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns:
The mbstowcs_s function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.
mbstowcs_s, _fmbstowcs_s

See Also: mbstowcs, mblen, mbtowc, wctomb, wctomb_s, wcstombs, wcstombs_s

Example: #define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char *wc = "string";
    wchar_t wbuffer[50];
    int i;
    errno_t rc;
    size_t retval;

    rc = mbstowcs_s( &retval, wbuffer, 50, wc, 10);
    if( rc == 0 ) {
        wbuffer[retval] = L'\0';
        printf( "%s(%d)\n", wc, retval );
        for( i = 0; i < retval; i++ )
            printf( "/%4.4x", wbuffer[i] );
        printf( "\n" );
    }
    return( 0 );
}

produces the following:

string(6)
/0073/0074/0072/0069/006e/0067

Classification: mbstowcs_s is TR 24731
_fmbstowcs_s is WATCOM

Systems: mbstowcs_s - All, Netware
_fmbstowcs_s - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:    
#include <mbstring.h>
int _mbterm( const unsigned char *ch );
int _fmbterm( const unsigned char __far *ch );

Description: The _mbterm function determines if the next multibyte character in the string pointed to by ch is a null character or a valid lead byte followed by a null character.

The _fmbterm function is a data model independent form of the _mbterm function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The _mbterm function returns 1 if the multibyte character pointed to by ch is a null character. The _mbterm function returns 2 if the multibyte character pointed to by ch is a valid lead byte character followed by a null character. Otherwise, the _mbterm function returns 0.

See Also: _mbccmp, _mbccpy, _mbcicmp, _mbcjistojms, _mbcjstojis, _mbclen, _mbctohira, _mbctokata, _mbctolower, _mbctombb, _mbctoupper, mblen, mbrcnt, mbrcntombs, mbsrtowcs, mbstowcs, mbstowcs_s, mbstowcs, mbtowc, wctomb, wcrtomb_s, wcsrtombs, wcsrtombs_s, wcstombs, wcstombs_s, wctob, wctomb, wcctomb_s

Example: #include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

const unsigned char chars[] = {
    ' ',
    '.',
    '1',
    'A',
    0x81,0x40, /* double-byte space */
    0x82,0x00  /* invalid double-byte */
};

#define SIZE sizeof( chars ) / sizeof( unsigned char )

void main()
{
    int i;

    _setmbcp( 932 );
    for( i = 0; i < SIZE; i++ ) {
        printf( "0x%2.2x %d\n", chars[i], _mbterm( &chars[i] ) );
    }
}

produces the following:
Classification: WATCOM

Systems:  
_mbterm - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fmterm - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

```c
#include <stdlib.h>
int mbtowc( wchar_t *pwc, const char *s, size_t n );
#include <mbstring.h>
int _fmbtowc( wchar_t __far *pwc,
             const char __far *s,
             size_t n );
```

Description: The `mbtowc` function converts a single multibyte character pointed to by `s` into the wide character code that corresponds to that multibyte character. The code for the null character is zero. If the multibyte character is valid and `pwc` is not a NULL pointer, the code is stored in the object pointed to by `pwc`. At most `n` bytes of the array pointed to by `s` will be examined.

The `mbtowc` function does not examine more than `MB_CUR_MAX` bytes.

The `_fmbtowc` function is a data model independent form of the `mbtowc` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: If `s` is a NULL pointer, the `mbtowc` function returns zero if multibyte character encodings are not state dependent, and non-zero otherwise. If `s` is not a NULL pointer, the `mbtowc` function returns:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if <code>s</code> points to the null character</td>
</tr>
<tr>
<td><code>len</code></td>
<td>the number of bytes that comprise the multibyte character (if the next <code>n</code> or fewer bytes form a valid multibyte character)</td>
</tr>
<tr>
<td><code>-1</code></td>
<td>if the next <code>n</code> bytes do not form a valid multibyte character</td>
</tr>
</tbody>
</table>

See Also: `mblen`, `wctomb`, `mbstowcs`, `wcstombs`

Example:

```c
#include <stdio.h>
#include <stdlib.h>
#include <mbctype.h>

void main()
{
    char    *wc = "string";
    wchar_t wbuffer[10];
    int     i, len;

    _setmbcp( 932 );
    printf( "Character encodings are %sstate dependent\n",
            ( mbtowc( wbuffer, NULL, 0 )
                ? "" : "not " ));

    len = mbtowc( wbuffer, wc, MB_CUR_MAX );
    wbuffer[len] = '\0';
    printf( "%s(%d)\n", wc, len );
    for( i = 0; i < len; i++ )
        printf( "%/4.4x", wbuffer[i] );
    printf( '\n' );
}
```

Library Functions and Macros  609
produces the following:

Character encodings are not state dependent
string(l)
/0073

Classification: mbtowc is ANSI
_fmbtowc is not ANSI

Systems: mbtowc – All, Netware
_fmbtowc – DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <mbstring.h>
unsigned char *_mbvtop( unsigned int ch,
  unsigned char *addr );
unsigned char __far *_fmbvtop( unsigned int ch,
  unsigned char __far *addr );

Description:  The _mbvtop function stores the multibyte character ch into the string pointed to by addr.

The _fmbvtop function is a data model independent form of the _mbvtop function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns:  The _mbvtop function returns the value of the argument addr.

See Also:  _mbccmp, _mbccpy, _mbcicmp, _mbcjistojms, _mbcjstojis, _mbclen, _mbctohira, _mbctokata, _mbctolower, _mbctoupper, mblen, mbrlen, mbrtowc, mbstowcs, mbsrtowcs_s, mbstowcs, mbstowcs_s, mbtowc, wcrtomb, wcsrtombs, wcsrtombs_s, wcstombs, wcstombs_s, wcstowb, wcstomb, wcstomb_s

Example:  
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>
#include <ctype.h>

void main()
{
  unsigned char string[10];
  unsigned char *p;
  int i;
  _setmbcp( 932 );
  p = string;
  _mbvtop( '.', p );
  p++;
  _mbvtop( '1', p );
  p++;
  _mbvtop( 'A', p );
  p++;
  _mbvtop( 0x8140, p );
  p += 2;
  _mbvtop( 0x8260, p );
  p += 2;
  _mbvtop( 0x82a6, p );
  p += 2;
  _mbvtop( '\0', p );
  for( i = 0; i < 10; i++ )
    printf( "%2.2x ", string[i] );
  printf( "\n" );
}

produces the following:

2e 31 41 81 40 82 60 82 a6 00
_mbvtop, _fmbvtop

Classification: WATCOM

Systems:

_mbvtop - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fmbvtop - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <malloc.h>  
size_t _memavl( void );

Description:  The _memavl function returns the number of bytes of memory available for dynamic memory allocation in the near heap (the default data segment). In the tiny, small and medium memory models, the default data segment is only extended as needed to satisfy requests for memory allocation. Therefore, you will need to call _nheapgrow in these memory models before calling _memavl in order to get a meaningful result.

The number returned by _memavl may not represent a single contiguous block of memory. Use the _memmax function to find the largest contiguous block of memory that can be allocated.

Returns:  The _memavl function returns the number of bytes of memory available for dynamic memory allocation in the near heap (the default data segment).

See Also:  calloc Functions, _freect, _memmax, _heapgrow Functions, malloc Functions, realloc Functions

Example:  
#include <stdio.h>  
#include <malloc.h>  

void main()
{
    char *p;
    char *fmt = "Memory available = %u\n";

    printf( fmt, _memavl() );
    _nheapgrow();
    printf( fmt, _memavl() );
    p = (char *) malloc( 2000 );
    printf( fmt, _memavl() );
}

produces the following:

Memory available = 0  
Memory available = 62732  
Memory available = 60730

Classification:  WATCOM

Systems:  All
#memccpy, _fmemccpy

## Synopsis:
```c
#include <string.h>
void *memccpy( void *dest, const void *src, int c, size_t cnt );
void __far * _fmemccpy( void __far * dest, const void __far * src, int c, size_t cnt );
```

## Description:
The `memccpy` function copies bytes from `src` to `dest` up to and including the first occurrence of the character `c` or until `cnt` bytes have been copied, whichever comes first.

The `_fmemccpy` function is a data model independent form of the `memccpy` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

## Returns:
The `memccpy` function returns a pointer to the byte in `dest` following the character `c` if one is found and copied, otherwise it returns NULL.

## See Also:
`memcpy`, `memmove`, `memset`

## Example:
```c
#include <stdio.h>
#include <string.h>

char *msg = "This is the string: not copied";

void main()
{
  auto char buffer[80];

  memset( buffer, '\0', 80 );
  memccpy( buffer, msg, ':', 80 );
  printf( "%s
", buffer );
}
```
produces the following:

This is the string:

## Classification:
WATCOM

## Systems:
`memccpy` - All, Netware
`_fmemccpy` - All
Synopsis:
#include <string.h>

void *memchr( const void *buf, int ch, size_t length );

void __far *_fmemchr( const void __far *buf,
                        int ch,
                        size_t length );

#include <wchar.h>

wchar_t *wmemchr( const wchar_t *buf, wchar_t ch, size_t length );

Description: The `memchr` function locates the first occurrence of `ch` (converted to an unsigned char) in the first `length` characters of the object pointed to by `buf`.

The `_fmemchr` function is a data model independent form of the `memchr` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `wmemchr` wide-character function is identical to `memchr` except that it operates on characters of `wchar_t` type. The argument `length` is interpreted to mean the number of wide characters.

Returns: The `memchr` function returns a pointer to the located character, or `NULL` if the character does not occur in the object.

See Also: `memcmp`, `memcpy`, `memicmp`, `memset`

Example:
#include <stdio.h>
#include <string.h>

void main( void )
{
    char buffer[80];
    char *where;

    strcpy( buffer, "video x-rays" );
    where = (char *)memchr( buffer, 'x', 6 );
    if( where == NULL )
        printf( "'x' not found\n" );
    else
        printf( "%s\n", where );

    where = (char *)memchr( buffer, 'r', 9 );
    if( where == NULL )
        printf( "'r' not found\n" );
    else
        printf( "%s\n", where );
}

Classification: `memchr` is ANSI
`_fmemchr` is not ANSI
`wmemchr` is ANSI

Systems: `memchr` - All, Netware
`_fmemchr` - All
`wmemchr` - All
Synopsis:  
#include <string.h>
int memcmp( const void *s1, const void *s2, size_t length );
int _fmemcmp( const void __far *s1, const void __far *s2, size_t length );
#include <wchar.h>
int wmemcmp( const wchar_t *s1, const wchar_t *s2, size_t length );

Description:  The memcmp function compares the first length characters of the object pointed to by s1 to the object pointed to by s2.

The _fmemcmp function is a data model independent form of the memcmp function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The wmemcmp wide-character function is identical to memcmp except that it operates on characters of wchar_t type. The argument length is interpreted to mean the number of wide characters.

Returns:  The memcmp function returns an integer less than, equal to, or greater than zero, indicating that the object pointed to by s1 is less than, equal to, or greater than the object pointed to by s2.

See Also:  memchr, memcpy, memicmp, memset

Example:  
#include <stdio.h>
#include <string.h>

void main( void )
{
    auto char buffer[80];

    strcpy( buffer, "world" );
    if( memcmp( buffer, "Hello ", 6 ) < 0 ) {
        printf( "Less than\n" );
    }
}

Classification:  memcmp is ANSI
               _fmemcmp is not ANSI
               wmemcmp is ANSI

Systems:  memcmp - All, Netware
           _fmemcmp - All
           wmemcmp - All

616  Library Functions and Macros
Synopsis:

```c
#include <string.h>
void *memcpy( void *dst,
    const void *src,
    size_t length );

void __far *fmemcpy( void __far *dst,
    const void __far *src,
    size_t length );

#include <wchar.h>
wchar_t *wmemcpy( wchar_t *dst,
    const wchar_t *src,
    size_t length );
```

Safer C:
The Safer C Library extension provides the `memcpy_s` function which is a safer alternative to `memcpy`. This newer `memcpy_s` function is recommended to be used instead of the traditional "unsafe" `memcpy` function.

Description:
The `memcpy` function copies `length` characters from the buffer pointed to by `src` into the buffer pointed to by `dst`. Copying of overlapping objects is not guaranteed to work properly. See the `memmove` function if you wish to copy objects that overlap.

The `_fmemcpy` function is a data model independent form of the `memcpy` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `wmemcpy` wide-character function is identical to `memcpy` except that it operates on characters of `wchar_t` type. The argument `length` is interpreted to mean the number of wide characters.

Returns:
The original value of `dst` is returned.

See Also:
`memchr`, `memcmp`, `memcmp`, `memmove`, `memset`, `memcpy_s`, `memmove_s`

Example:
```
#include <stdio.h>
#include <string.h>

void main( void )
{
    auto char buffer[80];

    memcpy( buffer, "Hello", 5 );
    buffer[5] = '\0';
    printf( "%s\n", buffer );
}
```

Classification: `memcpy` is ANSI  
`_fmemcpy` is not ANSI  
`wmemcpy` is ANSI

Systems:  
`memcpy` - All, Netware  
`_fmemcpy` - All  
`wmemcpy` - All
memcpy_s, wmemcpy_s

Synopsis:
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
errno_t memcpy_s( void * restrict s1,
                 rsize_t s1max,
                 const void * restrict s2,
                 rsize_t n);
#include <wchar.h>
errno_t wmemcpy_s( wchar_t * restrict s1,
                  rsize_t s1max,
                  const wchar_t * restrict s2,
                  size_t n);

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler
will be invoked and memcpy_s will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither s1 nor s2 shall be a null pointer. Neither s1max nor n shall be greater than RSIZE_MAX. n shall not be greater than s1max. Copying shall not take place between objects that overlap.

If there is a runtime-constraint violation, the memcpy_s function stores zeros in the first s1max characters of the object pointed to by s1 if s1 is not a null pointer and s1max is not greater than RSIZE_MAX.

Description: The memcpy_s function copies n characters from the buffer pointed to by s2 into the buffer pointed to by s1. Copying between overlapping objects is not allowed. See the memmove_s function if you wish to copy objects that overlap.

The wmemcpy_s wide-character function is identical to memcpy_s except that it operates on characters of wchar_t type. The arguments s1max and n are interpreted to mean the number of wide characters.

Returns: The memcpy_s function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

See Also: memcpy, memchr, memcmp, memcpy, memicmp, memmove, memset, memmove_s

Example:
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <string.h>

void main( void )
{
    char buffer[80];

    memcpy_s( buffer, sizeof( buffer ), "Hello", 5 );
    buffer[5] = '\0';
    printf( "%s\n", buffer );
}

Classification: memcpy_s is TR 24731
wmemcpy_s is TR 24731

Systems: memcpy_s - All, Netware
wmemcpy_s - All
Synopsis:

```c
#include <string.h>
int memicmp( const void *s1, const void *s2, size_t length );
int _memicmp( const void *s1, const void *s2, size_t length );
int _fmemicmp( const void __far *s1, const void __far *s2, size_t length );
```

Description:

The `memicmp` function compares, with case insensitivity (upper- and lowercase characters are equivalent), the first `length` characters of the object pointed to by `s1` to the object pointed to by `s2`.

The `_fmemicmp` function is a data model independent form of the `memicmp` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The `_memicmp` function is identical to `memicmp`. Use `_memicmp` for ANSI/ISO naming conventions.

Returns:

The `memicmp` function returns an integer less than, equal to, or greater than zero, indicating that the object pointed to by `s1` is less than, equal to, or greater than the object pointed to by `s2`.

See Also:

`memchr`, `memcmp`, `memcpy`, `memset`

Example:

```c
#include <stdio.h>
#include <string.h>

void main()
{
    char buffer[80];
    if( memicmp( buffer, "Hello", 5 ) < 0 ) {
        printf( "Less than\n" );
    }
}
```

Classification:

`memicmp` conforms to ANSI/ISO naming conventions

Systems:

- `memicmp` - All, Netware
- `_memicomp` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
- `_fmemicmp` - All
_memmax

Synopsis:    #include <malloc.h>
            size_t _memmax( void );

Description: The _memmax function returns the size of the largest contiguous block of memory available for
dynamic memory allocation in the near heap (the default data segment). In the tiny, small and medium
memory models, the default data segment is only extended as needed to satisfy requests for memory
allocation. Therefore, you will need to call _nheapgrow in these memory models before calling
_memmax in order to get a meaningful result.

Returns:    The _memmax function returns the size of the largest contiguous block of memory available for
dynamic memory allocation in the near heap. If 0 is returned, then there is no more memory available
in the near heap.

See Also:   calloc, _freect, _memavl, _heapgrow, malloc

Example:    #include <stdio.h>
            #include <malloc.h>

            void main()
            {
                char *p;
                size_t size;

                size = _memmax();
                printf( "Maximum memory available is %u\n", size );
                _nheapgrow();
                size = _memmax();
                printf( "Maximum memory available is %u\n", size );
                p = (char *) _nmalloc( size );
                size = _memmax();
                printf( "Maximum memory available is %u\n", size );
            }

produces the following:

    Maximum memory available is 0
    Maximum memory available is 62700
    Maximum memory available is 0

Classification: WATCOM

Systems:    All
Synopsis:  
#include <string.h>  
void *memmove( void *dst,  
        const void *src,  
        size_t length );
void __far *fmemmove( void __far *dst,  
        const void __far *src,  
        size_t length );
#include <wchar.h>  
wchar_t *wmemmove( wchar_t *dst,  
        const wchar_t *src,  
        size_t length );

Safer C:  
The Safer C Library extension provides the memmove_s function which is a safer alternative to memmove. This newer memmove_s function is recommended to be used instead of the traditional "unsafe" memmove function.

Description:  
The memmove function copies length characters from the buffer pointed to by src to the buffer pointed to by dst. Copying of overlapping objects will take place properly. See the memcpy function to copy objects that do not overlap.

The _fmemmove function is a data model independent form of the memmove function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The wmemmove wide-character function is identical to memmove except that it operates on characters of wchar_t type. The argument length is interpreted to mean the number of wide characters.

Returns:  
The memmove function returns dst.

See Also:  
memchr, memcmp, memcpy, memicmp, memset, memmove_s, memcpy_s

Example:  
#include <string.h>
void main( void )
{
    char buffer[80];
    memmove( buffer + 1, buffer, 79 );
    buffer[0] = '*';
}

Classification:  
memmove is ANSI
_fmemmove is not ANSI
wmemmove is ANSI

Systems:  
memmove – All, Netware
_fmemmove – All
wmemmove – All

Library Functions and Macros  621
Synopsis:
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
errno_t memmove_s( void * restrict s1,
                  rsize_t s1max,
                  const void * restrict s2,
                  rsize_t n);
#include <wchar.h>
errno_t wmemmove_s( wchar_t * restrict s1,
                   rsize_t s1max,
                   const wchar_t * restrict s2,
                   size_t n);

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and memmove_s will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither s1 nor s2 shall be a null pointer. Neither s1max nor n shall be greater than RSIZE_MAX. n shall not be greater than s1max.

If there is a runtime-constraint violation, the memmove_s function stores zeros in the first s1max characters of the object pointed to by s1 if s1 is not a null pointer and s1max is not greater than RSIZE_MAX.

Description: The memmove_s function copies n characters from the buffer pointed to by s2 into the buffer pointed to by s1. This copying takes place as if the n characters from the buffer pointed to by s2 are first copied into a temporary array of n characters that does not overlap the objects pointed to by s1 or s2, and then the n characters from the temporary array are copied into the object pointed to by s1.

See the memcpy_s function if you wish to copy objects that do not overlap.

The wmemmove_s wide-character function is identical to memmove_s except that it operates on characters of wchar_t type. The arguments s1max and n are interpreted to mean the number of wide characters.

Returns: The memmove_s function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

See Also: memchr, memcmp, memcpy, memicmp, memmove, memset, memcpy_s

Example:
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
void main( void)
{
    char buffer[80] = "0123456789";
    memmove_s( buffer + 1, sizeof( buffer ), buffer, 79 );
    buffer[0] = '*';
    printf( buffer );
}

produces the following:
0123456789

622 Library Functions and Macros
Classification: memmove_s is TR 24731
               wmemmove_s is TR 24731

Systems:  memmove_s - All, Netware
          wmemmove_s - All
Synopsis:  
#include <mmintrin.h>
void _m_empty(void);

Description:  The _m_empty function empties the multimedia state. The values in the Multimedia Tag Word (TW) are set to empty (i.e., all ones). This will indicate that no Multimedia registers are in use.

This function is useful for applications that mix floating-point (FP) instructions with multimedia instructions. Intel maps the multimedia registers onto the floating-point registers. For this reason, you are discouraged from intermixing MM code and FP code. The recommended way to write an application with FP instructions and MM instructions is:

- Split the FP code and MM code into two separate instruction streams such that each stream contains only instructions of one type.
- Do not rely on the contents of FP/MM registers across transitions from one stream to the other.
- Leave the MM state empty at the end of an MM stream using the _m_empty function.
- Similarly, leave the FP stack empty at the end of an FP stream.

Returns:  The _m_empty function does not return a value.

See Also:  _m_from_int, _m_to_int, _m_packsswb, _m_packdq, _m_pand, _m_pcmpeqb, _m_pmaddwd, _m_psllw, _m_psraw, _m_psrlw, _m_psubb, _m_punpckhbw

Example:  
#include <stdio.h>
#include <mmintrin.h>

long featureflags( void );
#include <mmintrin.h>

#pragma aux featureflags = 
  "586"          
  "mov eax,1"     
  "cpuid"         
  "mov eax,edx"   
  modify [eax ebx ecx edx]

#define MM_EXTENSION 0x00800000

void main( void )
{
  if( featureflags() & MM_EXTENSION ) {
      /*
       * sequence of code that uses Multimedia functions
       *.
       */
      _m_empty();
  }
}
/ * 
    sequence of code that uses floating-point 
    * / 
} 

Classification: Intel  

Systems: MACRO
**Synopsis:**

```c
#include <string.h>

void *memset( void *dst, int c, size_t length );
void __far *__fmemset( void __far *dst, int c, size_t length );
wchar_t *wmemset( wchar_t *dst, wchar_t c, size_t length );
```

**Description:**

The `memset` function fills the first `length` characters of the object pointed to by `dst` with the value `c`.

The `_fmemset` function is a data model independent form of the `memset` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `wmemset` wide-character function is identical to `memset` except that it operates on characters of `wchar_t` type. The argument `length` is interpreted to mean the number of wide characters.

**Returns:**

The `memset` function returns the pointer `dst`.

**See Also:** `memchr`, `memcmp`, `memcpy`, `memcmp`, `memmove`

**Example:**

```c
#include <string.h>

void main( void )
{
    char buffer[80];
    memset( buffer, '=', 80 );
}
```

**Classification:**

`memset` is ANSI

`_fmemset` is not ANSI

`wmemset` is ANSI

**Systems:**

`memset` - All, Netware

`_fmemset` - All

`wmemset` - All
Synopsis:  
#include <mmintrin.h>
__m64 __m_from_int(int i);

Description:  The __m_from_int function forms a 64-bit MM value from an unsigned 32-bit integer value.

Returns:  The 64-bit result of loading MM0 with an unsigned 32-bit integer value is returned.

See Also:  __m_empty, __m_to_int, __m_packsswb, __m_paddb, __m_pand, __m_pcmpeqb, __m_pmaddwd, __m_pslw, __m_psraw, __m_psrlw, __m_empty, __m_psubb, __m_punpckhbw, __m_pmaddwd, __m_psllw, __m_psraw, __m_psrlw, __m_empty, __m_psubb, __m_punpckhbw

Example:  
#include <stdio.h>
#include <mmintrin.h>

__m64 a;
int k = 0xF1F2F3F4;

void main()
{
    a = __m_from_int(k);
    printf("int=%8.8lx m=%8.8lx%8.8lx\n", k, a._32[1], a._32[0]);
}

produces the following:

int=f1f2f3f4 m=00000000f1f2f3f4

Classification:  Intel

Systems:  MACRO
Synopsis: 
#include <stdlib.h>
#define min(a,b)  (((a) < (b)) ? (a) : (b))

Description:  The min macro will evaluate to be the lesser of two values. It is implemented as follows.

#define min(a,b)  (((a) < (b)) ? (a) : (b))

Returns:  The min macro will evaluate to the smaller of the two values passed.

See Also:  max

Example:  #include <stdio.h>
#include <stdlib.h>

void main()
{
  int a;

  /*
   * The following line will set the variable "a" to 1
   * since 10 is greater than 1.
   */
  a = min( 1, 10 );
  printf( "The value is: %d\n", a );
}

Classification: WATCOM

Systems:  All, Netware
Synopsis:

```
#include <sys/types.h>
#include <dirent.h>
int mkdir( const char *path );
int _mkdir( const char *path );
int _wmkdir( const wchar_t *path );
```

Description:
The `mkdir` function creates a new subdirectory with name `path`. The `path` can be either relative to the current working directory or it can be an absolute path name.

The `_mkdir` function is identical to `mkdir`. Use `_mkdir` for ANSI/ISO naming conventions.

The `_wmkdir` function is identical to `mkdir` except that it accepts a wide-character string argument.

Returns:
The `mkdir` function returns zero if successful, and a non-zero value otherwise.

Errors:
When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EACCES</td>
<td>Search permission is denied for a component of <code>path</code> or write permission is denied on the parent directory of the directory to be created.</td>
</tr>
<tr>
<td>EEXIST</td>
<td>The named file exists.</td>
</tr>
<tr>
<td>ENOENT</td>
<td>The specified <code>path</code> does not exist or <code>path</code> is an empty string.</td>
</tr>
</tbody>
</table>

See Also:
`chdir`, `chmod`, `getcwd`, `rmdir`, `stat`, `umask`

Example:

To make a new directory called `\watcom` on drive C:

```
#include <sys/types.h>
#include <direct.h>

void main( void )
{
  mkdir( "c:\watcom" );
}
```

Note the use of two adjacent backslash characters (\) within character-string constants to signify a single backslash.

Classification:
`mkdir` is POSIX 1003.1
`_mkdir` is not POSIX
`_wmkdir` is not POSIX
`_mkdir` conforms to ANSI/ISO naming conventions

Systems:
`mkdir` - All, Netware
`_mkdir` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
`_wmkdir` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <i86.h>
void __far *MK_FP( unsigned int segment,
                unsigned int offset );

Description:  The MK_FP macro can be used to obtain the far pointer value given by the segment segment value and
the offset offset value. These values may be obtained by using the FP_SEG and FP_OFF macros.

Returns:  The macro returns a far pointer.

See Also:  FP_OFF, FP_SEG, segread

Example:  
#include <i86.h>
#include <stdio.h>

void main()
{  
    unsigned short __far *bios_prtr_port_1;

    bios_prtr_port_1 =
        (unsigned short __far *) MK_FP( 0x40, 0x8 );
    printf( "Port address is %x\n", *bios_prtr_port_1 );
}

Classification: Intel

Systems: MACRO
Synopsis:  
#include <stdlib.h>  
int mkstemp( char *template );

Description:  The mkstemp function creates a file with unique name by modifying the template argument, and returns its file handle open for reading and writing in binary mode. The use of mkstemp prevents any possible race condition between testing whether the file exists and opening it for use.

The string template has the form baseXXXXXX where base is the fixed part of the generated filename and XXXXXX is the variable part of the generated filename. Each of the 6 X’s is a placeholder for a character supplied by mkstemp. Each placeholder character in template must be an uppercase “X”. mkstemp preserves base and replaces the first of the 6 trailing X’s with a unique sequence of alphanumeric characters. The string template therefore must be writable.

mkstemp checks to see if a file with the generated name already exists and if so selects another name, until it finds a file that doesn’t exist. If it is unsuccessful at finding a name for a file that does not already exist or is unable to create a file, mkstemp returns -1.

Returns:  The mkstemp function returns a file handle. When an error occurs while creating the file, -1 is returned.

See Also:  fopen, freopen, _mktemp, _tempnam, tmpfile, tmpnam

Example:  
#include <stdio.h>  
#include <string.h>  
#include <stdlib.h>  
#include <unistd.h>  
#define TEMPLATE    "_tXXXXXX"
#define MAX TEMPS   5

void main( void )  
{
    char     name[sizeof( TEMPLATE )];
    int      i;
    int      handles[MAX TEMPS];

    for( i = 0; i < MAX TEMPS; i++ ) {
        strcpy( name, TEMPLATE );
        handles[i] = mkstemp( name );
        if( handles[i] == -1 ) {
            printf( "Failed to create temporary file\n" );
        } else {
            printf( "Created temporary file ‘%s’\n", name );
        }
    }
    for( i = 0; i < MAX TEMPS; i++ ) {
        if( handles[i] != -1 ) {
            close( handles[i] );
        }
    }
}

Classification:  POSIX
mkstemp

Systems: All, Netware
Synopsis:

```
#include <io.h>
char * _mktemp( char *template );
#include <wchar.h>
wchar_t * _wmktemp( wchar_t *template );
```

Description: The `_mktemp` function creates a unique filename by modifying the `template` argument. `_mktemp` automatically handles multibyte-character string arguments as appropriate, recognizing multibyte-character sequences according to the multibyte code page currently in use by the run-time system.

The `_wmktemp` function is a wide-character version of `_mktemp` that operates with wide-character strings.

The string `template` has the form `baseXXXXXX` where `base` is the fixed part of the generated filename and `XXXXXX` is the variable part of the generated filename. Each of the 6 X’s is a placeholder for a character supplied by `_mktemp`. Each placeholder character in `template` must be an uppercase “X”. `_mktemp` preserves `base` and replaces the first of the 6 trailing X’s with a lowercase alphabetic character (a-z). `_mktemp` replaces the following 5 trailing X’s with a five-digit value this value is a unique number identifying the calling process or thread.

_mktemp checks to see if a file with the generated name already exists and if so selects another letter, in succession, from "a" to "z" until it finds a file that doesn’t exist. If it is unsuccessful at finding a name for a file that does not already exist, _mktemp returns NULL. At most, 26 unique file names can be returned to the calling process or thread.

Returns: The `_mktemp` function returns a pointer to the modified `template`. The `_mktemp` function returns NULL if `template` is badly formed or no more unique names can be created from the given template.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `fopen`, `freopen`, `mkstemp`, `_tempnam`, `tmpfile`, `tmpnam`

Example:
#include <stdio.h>
#include <string.h>
#include <io.h>

#define TMPLTE "_tXXXXXX"

void main()
{
    char name[sizeof(TMPLTE)];
    char *mknm;
    int i;
    FILE *fp;

    for( i = 0; i < 30; i++ ) {
        strcpy( name, TMPLTE );
        mknm = _mktemp( name );
        if( mknm == NULL )
            printf( "Name is badly formed\n" );
        else {
            printf( "Name is %s\n", mknm );
            fp = fopen( mknm, "w" );
            if( fp != NULL ) {
                fprintf( fp, "Name is %s\n", mknm );
                fclose( fp );
            }
        }
    }
}

Classification: WATCOM

Systems: _mktemp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
        _wmktemp - Win32
Synopsis:

```c
#include <time.h>

time_t mktime( struct tm *timeptr );
```

```c
struct tm {
    int tm_sec;   /* seconds after the minute -- [0,61] */
    int tm_min;   /* minutes after the hour   -- [0,59] */
    int tm_hour;  /* hours after midnight     -- [0,23] */
    int tm_mday;  /* day of the month         -- [1,31] */
    int tm_mon;   /* months since January     -- [0,11] */
    int tm_year;  /* years since 1900                   */
    int tm_wday;  /* days since Sunday        -- [0,6]  */
    int tm_yday;  /* days since January 1     -- [0,365]*/
    int tm_isdst; /* Daylight Savings Time flag */
};
```

Description:

The `mktime` function converts the local time information in the structure pointed to by `timeptr` into a calendar time (Coordinated Universal Time) with the same encoding used by the `time` function. The original values of the fields `tm_sec`, `tm_min`, `tm_hour`, `tm_mday`, and `tm_mon` are not restricted to ranges described for `struct tm`. If these fields are not in their proper ranges, they are adjusted so that they are in the proper ranges. Values for the fields `tm_wday` and `tm_yday` are computed after all the other fields have been adjusted.

If the original value of `tm_isdst` is negative, this field is computed also. Otherwise, a value of 0 is treated as "daylight savings time is not in effect" and a positive value is treated as "daylight savings time is in effect".

Whenever `mktime` is called, the `tzset` function is also called.

Returns:

The `mktime` function returns the converted calendar time.

See Also:

`asctime` Functions, `asctime_s`, `clock`, `ctime` Functions, `ctime_s`, `difftime`, `gmtime`, `gmtime_s`, `localtime`, `localtime_s`, `strftime`, `time`, `tzset`

Example:

```c
#include <stdio.h>
#include <time.h>

static const char *week_day[] = {
    "Sunday", "Monday", "Tuesday", "Wednesday",
    "Thursday", "Friday", "Saturday"
};

void main()
{
    struct tm new_year;
```
new_year.tm_year = 2001 - 1900;
new_year.tm_mon = 0;
new_year.tm_mday = 1;
new_year.tm_hour = 0;
new_year.tm_min = 0;
new_year.tm_sec = 0;
new_year.tm_isdst = 0;
mktime( &new_year );
printf( "The 21st century began on a %s\n",
    week_day[ new_year.tm_wday ] );
}

produces the following:

The 21st century began on a Monday

Classification: ANSI

Systems: All, Netware
Synopsis:

```
#include <math.h>

double modf( double value, double *iptr );
```

Description: The `modf` function breaks the argument `value` into integral and fractional parts, each of which has the same sign as the argument. It stores the integral part as a `double` in the object pointed to by `iptr`.

Returns: The `modf` function returns the signed fractional part of `value`.

See Also: `frexp`, `ldexp`

Example:

```
#include <stdio.h>
#include <math.h>

void main()
{
    double integral_value, fractional_part;

    fractional_part = modf( 4.5, &integral_value );
    printf( "%f %f\n", fractional_part, integral_value );
    fractional_part = modf( -4.5, &integral_value );
    printf( "%f %f\n", fractional_part, integral_value );
}
```

produces the following:

```
0.500000 4.000000
-0.500000 -4.000000
```

Classification: ANSI

Systems: Math
movedata

Synopsis:  
#include <string.h>
void movedata( unsigned int src_segment,
               unsigned int src_offset,
               unsigned int tgt_segment,
               unsigned int tgt_offset,
               size_t length );

Description:  The movedata function copies length bytes from the far pointer calculated as
(src_segment:src_offset) to a target location determined as a far pointer
(tgt_segment:tgt_offset).

Overlapping data may not be correctly copied. When the source and target areas may overlap, copy the
areas one character at a time.

The function is useful to move data when the near address(es) of the source and/or target areas are not
known.

Returns:  No value is returned.

See Also:  FP_SEG, FP_OFF, memcpy, segread

Example:  
#include <stdio.h>
#include <string.h>
#include <dos.h>

void main()
{
    char buffer[14] = {
        '*', 0x17, 'H', 0x17, 'e', 0x17, 'l', 0x17,
        'l', 0x17, 'o', 0x17, '*', 0x17 };

    movedata( FP_SEG( buffer ),
               FP_OFF( buffer ),
               0xB800,
               0x0720,
               14 );
}

Classification:  WATCOM

Systems:  All, Netware
Synopsis:  
#include <graph.h>
struct xycoord _FAR _moveto( short x, short y );

struct _wxycoord _FAR _moveto_w( double x, double y );

Description:  The _moveto functions set the current output position for graphics. The _moveto function uses the view coordinate system. The _moveto_w function uses the window coordinate system.

The current output position is set to be the point at the coordinates \((x, y)\). Nothing is drawn by the function. The _lineto function uses the current output position as the starting point when a line is drawn.

Note that the output position for graphics output differs from that for text output. The output position for text output can be set by use of the _settextposition function.

Returns:  The _moveto functions return the previous value of the output position for graphics.

See Also:  _getcurrentposition, _lineto, _settextposition

Example:  
#include <conio.h>
#include <graph.h>

main()
{
    _setvideomode( _VRES16COLOR );
    _moveto( 100, 100 );
    _lineto( 540, 100 );
    _lineto( 320, 380 );
    _lineto( 100, 100 );
    getch();
    _setvideomode( _DEFAULTMODE );
}

Classification:  PC Graphics

Systems:  _moveto - DOS, QNX
           _moveto_w - DOS, QNX
**Synopsis:**

```c
#include <mmintrin.h>
__m64 _m_packssdw(__m64 *m1, __m64 *m2);
```

**Description:** Convert signed packed double-words into signed packed words by packing (with signed saturation) the low-order words of the signed double-word elements from `m1` and `m2` into the respective signed words of the result. If the signed values in the word elements of `m1` and `m2` are smaller than 0x8000, the result elements are clamped to 0x8000. If the signed values in the word elements of `m1` and `m2` are larger than 0x7fff, the result elements are clamped to 0x7fff.

```
 m2    m1
    ----------------------------------
   | w3 : w2 | w1 : w0 |   | w3 : w2 | w1 : w0 |
    ----------------------------------
   |   |   |   |   |   |   |   |   |
       | | | | | | | | |
       v v v v
    ----------------------------------
   | w3 | w2 | w1 | w0 |
    ----------------------------------
 result
```

**Returns:** The result of packing, with signed saturation, 32-bit signed double-words into 16-bit signed words is returned.

**See Also:** 
- `_m_empty`
- `_m_packsswb`
- `_m_packuswb`

**Example:**

```c
#include <stdio.h>
#include <mmintrin.h>

#define AS_BYTES "%2.2x %2.2x %2.2x %2.2x " 
   "%2.2x %2.2x %2.2x %2.2x"
#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"
#define AS_DWORDS "%8.8lx %8.8lx"

__m64   a;
__m64   b = { 0x0000567800001234 };  
__m64   c = { 0xffffffff00010101 };

void main()
{
    a = _m_packssdw( b, c );
    printf( "m2="AS_DWORDS" 
            "m1="AS_DWORDS"
            "mm="AS_WORDS"
            ,
            c._32[1], c._32[0],
            b._32[1], b._32[0],
            a._16[3], a._16[2], a._16[1], a._16[0] );
}
```

produces the following:

```
m2=fffffffe 00010101 m1=00005678 00001234
mm=fffe 7fff 5678 1234
```
Classification: Intel
Systems: MACRO
Synopsis:  

```
#include <mmintrin.h>
__m64 _m_packsswb(__m64 *m1, __m64 *m2);
```

Description:  Convert signed packed words into signed packed bytes by packing (with signed saturation) the low-order bytes of the signed word elements from \( m1 \) and \( m2 \) into the respective signed bytes of the result. If the signed values in the word elements of \( m1 \) and \( m2 \) are smaller than 0x80, the result elements are clamped to 0x80. If the signed values in the word elements of \( m1 \) and \( m2 \) are larger than 0x7f, the result elements are clamped to 0x7f.

```

Returns:  The result of packing, with signed saturation, 16-bit signed words into 8-bit signed bytes is returned.

See Also:  _m_empty, _m_packssdw, _m_packuswb

Example:  

```
#include <stdio.h>
#include <mmintrin.h>

#define AS_BYTES "%2.2x %2.2x %2.2x %2.2x "
    "%2.2x %2.2x %2.2x %2.2x"
#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"
#define AS_DWORDS "%8.8lx %8.8lx"

__m64   a;
__m64   b = { 0x0004000300020001 };
__m64   c = { 0xff7fff800080007f };

void main()
{
    a = _m_packsswb( b, c );
    printf( "m2="AS_WORDS "\n"m1="AS_WORDS\n"
    "mm="AS_BYTES\n",
    c._16[3], c._16[2], c._16[1], c._16[0],
    b._16[3], b._16[2], b._16[1], b._16[0],
    a._8[7], a._8[6], a._8[5], a._8[4],
    a._8[3], a._8[2], a._8[1], a._8[0] );
}
```

produces the following:
_m_packsswb

m2=ff7f ff80 0080 007f m1=0004 0003 0002 0001
mm=80 80 7f 7f 04 03 02 01

Classification: Intel

Systems: MACRO
**Synopsis:**
```
#include <mmintrin.h>
__m64 _m_packuswb(__m64 *m1, __m64 *m2);
```

**Description:** Convert signed packed words into unsigned packed bytes by packing (with unsigned saturation) the low-order bytes of the signed word elements from `m1` and `m2` into the respective unsigned bytes of the result. If the signed values in the word elements of `m1` and `m2` are too large to be represented in an unsigned byte, the result elements are clamped to 0xff.

```
m2
<table>
<thead>
<tr>
<th>b7 b6 b5 b4</th>
<th>b3 b2 b1 b0</th>
</tr>
</thead>
</table>

----------------------------------------
| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
----------------------------------------

m1
<table>
<thead>
<tr>
<th>b7 b6 b5 b4</th>
<th>b3 b2 b1 b0</th>
</tr>
</thead>
</table>

----------------------------------------
| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
----------------------------------------
```

**Returns:** The result of packing, with unsigned saturation, 16-bit signed words into 8-bit unsigned bytes is returned.

**See Also:**  
`_m_empty`, `_m_packssdw`, `_m_packsswb`

**Example:**
```
#include <stdio.h>
#include <mmintrin.h>

#define AS_BYTES "%2.2x %2.2x %2.2x %2.2x "
    "%2.2x %2.2x %2.2x %2.2x"
#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"
#define AS_DWORDS "%8.8lx %8.8lx"

__m64 a;
__m64 b = { 0x0004000300020001 }
__m64 c = { 0xff7fff800080007f }

void main()
{
    a = _m_packuswb( b, c );
    printf( "m2=\"AS_WORDS\" \
            "m1=\"AS_WORDS\"\n            "mm=\"AS_BYTES\"\n            c._16[3], c._16[2], c._16[1], c._16[0],
            b._16[3], b._16[2], b._16[1], b._16[0],
            a._8[7], a._8[6], a._8[5], a._8[4],
            a._8[3], a._8[2], a._8[1], a._8[0] );
}
```

produces the following:
m2=ff7f ff80 0080 007f m1=0004 0003 0002 0001
mm=00 00 80 7f 04 03 02 01

Classification: Intel

Systems: MACRO
Synopsis: 
#include <mmintrin.h>
__m64 _m_paddb(__m64 *m1, __m64 *m2);

Description: The signed or unsigned 8-bit bytes of \texttt{m2} are added to the respective signed or unsigned 8-bit bytes of \texttt{m1} and the result is stored in memory. If any result element does not fit into 8 bits (overflow), the lower 8 bits of the result elements are stored (i.e., truncation takes place).

Returns: The result of adding the packed bytes of two 64-bit multimedia values is returned.

See Also: \_m_empty, \_m_paddd, \_m_paddsb, \_m_paddsw, \_m_paddusb, \_m_paddusw, \_m_paddw

Example: 
#include <stdio.h>
#include <mmintrin.h>

#define AS_BYTES "%2.2x %2.2x %2.2x %2.2x " \\
"%2.2x %2.2x %2.2x %2.2x"

__m64 a;
__m64 b = \{ 0x0123456789abcdef \};
__m64 c = \{ 0xfedcba9876543210 \};

void main()
{
    a = _m_paddb( b, c );
    printf( "m1=\"AS\_BYTES\"\n"
            "m2=\"AS\_BYTES\"\n"
            "mm=\"AS\_BYTES\"\n",
            b._8[7], b._8[6], b._8[5], b._8[4],
            b._8[3], b._8[2], b._8[1], b._8[0],
            c._8[7], c._8[6], c._8[5], c._8[4],
            c._8[3], c._8[2], c._8[1], c._8[0],
            a._8[7], a._8[6], a._8[5], a._8[4],
            a._8[3], a._8[2], a._8[1], a._8[0] );
}

produces the following:

\texttt{m1=01 23 45 67 89 ab cd ef}
\texttt{m2=fe dc ba 98 76 54 32 10}
\texttt{mm=ff ff ff ff ff ff ff ff}

Classification: Intel

Systems: MACRO
Synopsis:  
#include <mmintrin.h>
__m64 _m_paddd(__m64 *m1, __m64 *m2);

Description:  The signed or unsigned 32-bit double-words of \( m2 \) are added to the respective signed or unsigned 32-bit double-words of \( m1 \) and the result is stored in memory. If any result element does not fit into 32 bits (overflow), the lower 32-bits of the result elements are stored (i.e., truncation takes place).

Returns:  The result of adding the packed double-words of two 64-bit multimedia values is returned.

See Also:  _m_empty, _m_paddd, _m_paddsb, _m_paddsw, _m_paddusb, _m_paddusw, _m_paddw

Example:  
#include <stdio.h>
#include <mmintrin.h>

#define AS_DWORDS "%8.8lx %8.8lx"

__m64   a;
__m64   b = { 0x0123456789abcdef };  
__m64   c = { 0xfedcba9876543210 }; 

void main()
{
    a = _m_paddd( b, c );
    printf( "m1="AS_DWORDS"\n"
            "m2="AS_DWORDS"\n"
            "mm="AS_DWORDS"\n",
            b._32[1], b._32[0],
            c._32[1], c._32[0],
            a._32[1], a._32[0] ) ;
}

produces the following:

m1=01234567 89abcdef
m2=fedcba98 76543210
mm=ffffffff ffffffff

Classification:  Intel

Systems:  MACRO
Synopsis:  
#include <mmintrin.h>  
__m64 _m_paddsb(__m64 *m1, __m64 *m2);

Description:  The signed 8-bit bytes of \texttt{m2} are added to the respective signed 8-bit bytes of \texttt{m1} and the result is stored in memory. Saturation occurs when a result exceeds the range of a signed byte. In the case where a result is a byte larger than 0x7f (overflow), it is clamped to 0x7f. In the case where a result is a byte smaller than 0x80 (underflow), it is clamped to 0x80.

Returns:  The result of adding the packed signed bytes, with saturation, of two 64-bit multimedia values is returned.

See Also:  \_m_empty, \_m_paddb, \_m_paddw, \_m_paddsw, \_m_paddusb, \_m_paddusw, \_m_paddw

Example:  
#include <stdio.h>  
#include <mmintrin.h>

#define AS_BYTES "%2.2x %2.2x %2.2x %2.2x " \  
"%2.2x %2.2x %2.2x %2.2x"

__m64 a;
__m64 b = { 0x8aacceef02244668 };  
__m64 c = { 0x76543211fedcba98 };  

void main()
{
    a = _m_paddsb( b, c );
    printf( "m1="AS_BYTES"\n"  
        "m2="AS_BYTES"\n"  
        "mm="AS_BYTES"\n",  
        b._8[7], b._8[6], b._8[5], b._8[4],  
        b._8[3], b._8[2], b._8[1], b._8[0],  
        c._8[7], c._8[6], c._8[5], c._8[4],  
        c._8[3], c._8[2], c._8[1], c._8[0],  
        a._8[7], a._8[6], a._8[5], a._8[4],  
        a._8[3], a._8[2], a._8[1], a._8[0] );
}

produces the following:

\texttt{m1=8a ac ce ef 02 24 46 68}  
\texttt{m2=76 54 32 11 fe dc ba 98}  
\texttt{mm=00 00 00 00 00 00 00 00}

Classification: Intel

Systems: MACRO
Synopsis:
```c
#include <mmintrin.h>
__m64 _m_paddsw(__m64 *m1, __m64 *m2);
```

Description:
The signed 16-bit words of `m2` are added to the respective signed 16-bit words of `m1` and the result is stored in memory. Saturation occurs when a result exceeds the range of a signed word. In the case where a result is a word larger than 0x7fff (overflow), it is clamped to 0x7fff. In the case where a result is a word smaller than 0x8000 (underflow), it is clamped to 0x8000.

Returns:
The result of adding the packed signed words, with saturation, of two 64-bit multimedia values is returned.

See Also:
`_m_empty`, `_m_paddb`, `_m_paddw`, `_m_paddsb`, `_m_paddusb`, `_m_paddusw`, `_m_paddw`

Example:
```c
#include <stdio.h>
#include <mmintrin.h>
#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"

__m64 a;
__m64 b = { 0x8aacceef02244668 };  
__m64 c = { 0x76543211fedcba98 };  

void main()
{
    a = _m_paddsw(b, c);
    printf("m1="AS_WORDS"
         
         "m2="AS_WORDS"
         
         "mm="AS_WORDS",
         
         b._16[3], b._16[2], b._16[1], b._16[0],
         c._16[3], c._16[2], c._16[1], c._16[0],
         a._16[3], a._16[2], a._16[1], a._16[0] );
}
```

produces the following:

```
m1=8aac ceef 0224 4668
m2=7654 3211 fedc ba98
mm=0100 0100 0100 0100
```

Classification: Intel

Systems: MACRO
Synopsis:  
#include <mmintrin.h>
__m64 _m_paddusb(__m64 *m1, __m64 *m2);

Description:  The unsigned 8-bit bytes of m2 are added to the respective unsigned 8-bit bytes of m1 and the result is stored in memory. Saturation occurs when a result exceeds the range of an unsigned byte. In the case where a result is a byte larger than 0xff (overflow), it is clamped to 0xff.

Returns:  The result of adding the packed unsigned bytes, with saturation, of two 64-bit multimedia values is returned.

See Also:  _m_empty, _m_paddb, _m_paddw, _m_paddsw, _m_paddusw

Example:  
#include <stdio.h>
#include <mmintrin.h>

#define AS_BYTES "%2.2x %2.2x %2.2x %2.2x " 
    "%2.2x %2.2x %2.2x %2.2x"

__m64 a;
__m64 b = { 0x8aacceef02244668 };
__m64 c = { 0x76543211fedcba98 };

void main()
{
    a = _m_paddusb( b, c );
    printf( "m1=%s
" "m2=%s
" "mm=%s
",
        b._8[7], b._8[6], b._8[5], b._8[4],
        b._8[3], b._8[2], b._8[1], b._8[0],
        c._8[7], c._8[6], c._8[5], c._8[4],
        c._8[3], c._8[2], c._8[1], c._8[0],
        a._8[7], a._8[6], a._8[5], a._8[4],
        a._8[3], a._8[2], a._8[1], a._8[0] );
}

produces the following:

m1=8a ac ce ef 02 24 46 68
m2=76 54 32 11 fe dc ba 98
mm=ff ff ff ff ff ff ff

Classification:  Intel

Systems:  MACRO
Synopsis: #include <mmintrin.h>
__m64 _m_paddusw(__m64 *m1, __m64 *m2);

Description: The unsigned 16-bit words of m2 are added to the respective unsigned 16-bit words of m1 and the result is stored in memory. Saturation occurs when a result exceeds the range of an unsigned word. In the case where a result is a word larger than 0xffff (overflow), it is clamped to 0xffff.

Returns: The result of adding the packed unsigned words, with saturation, of two 64-bit multimedia values is returned.

See Also: _m_empty, _m_paddb, _m_paddd, _m_paddsb, _m_paddsw, _m_paddusb, _m_paddw

Example: #include <stdio.h>
#include <mmintrin.h>

#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"

__m64   a;
__m64   b = { 0x8aacceef02244668 };
__m64   c = { 0x76543211fedcba98 };

void main()
{
    a = _m_paddusw( b, c );
    printf( "m1="AS_WORDS"\n"
            "m2="AS_WORDS"\n"
            "mm="AS_WORDS"\n",
            b._16[3], b._16[2], b._16[1], b._16[0],
            c._16[3], c._16[2], c._16[1], c._16[0],
            a._16[3], a._16[2], a._16[1], a._16[0] );
}

produces the following:

m1=8aac ceef 0224 4668
m2=7654 3211 fedc ba98
mm=ffff ffff ffff ffff

Classification: Intel

Systems: MACRO
Synopsis:  
#include <mmintrin.h>
__m64 _m_paddw(__m64 *m1, __m64 *m2);

Description:  The signed or unsigned 16-bit words of \textit{m2} are added to the respective signed or unsigned 16-bit words of \textit{m1} and the result is stored in memory. If any result element does not fit into 16 bits (overflow), the lower 16 bits of the result elements are stored (i.e., truncation takes place).

Returns:  The result of adding the packed words of two 64-bit multimedia values is returned.

See Also:  _m_empty, _m_paddb, _m_paddd, _m_paddsb, _m_paddsw, _m_paddusb, _m_paddusw

Example:  
#include <stdio.h>
#include <mmintrin.h>

#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"

__m64   a;
__m64   b = { 0x0123456789abcdef };  
__m64   c = { 0xfedcba9876543210 };

void main()
{
    a = _m_paddw( b, c );
    printf( "m1="AS_WORDS"\n"
            "m2="AS_WORDS"\n"
            "mm="AS_WORDS"\n",
        b._16[3], b._16[2], b._16[1], b._16[0],
        c._16[3], c._16[2], c._16[1], c._16[0],
        a._16[3], a._16[2], a._16[1], a._16[0] );
}

produces the following:

m1=0123 4567 89ab cdef
m2=fedc ba98 7654 3210
mm=ffff ffff ffff ffff

Classification:  Intel

Systems:  MACRO
Synopsis:
#include <mmintrin.h>
__m64 _m_pand(__m64 *m1, __m64 *m2);

Description: A bit-wise logical AND is performed between 64-bit multimedia operands m1 and m2 and the result is stored in memory.

Returns: The bit-wise logical AND of two 64-bit values is returned.

See Also: _m_empty, _m_pandn, _m_por, _m_pxor

Example:
#include <stdio.h>
#include <mmintrin.h>

#define AS_QWORD "%16.16Lx"

__m64 a;
__m64 b = { 0x0123456789abcdef };  
__m64 c = { 0xfedcba9876543210 };  

void main()
{
    a = _m_pand( b, c );  
    printf( "m1=AS_QWORD\n"
            "m2=AS_QWORD\n"
            "mm=AS_QWORD\n",
            b, c, a );
}

produces the following:

m1=0123456789abcdef  
m2=fedcba9876543210  
mm=0000000000000000

Classification: Intel

Systems: MACRO
**Synopsis:**
```
#include <mmintrin.h>
__m64 _m_pandn(__m64 *m1, __m64 *m2);
```

**Description:** A bit-wise logical AND is performed on the logical inversion of 64-bit multimedia operand \( m1 \) and 64-bit multimedia operand \( m2 \) and the result is stored in memory.

**Returns:** The bit-wise logical AND of an inverted 64-bit value and a non-inverted value is returned.

**See Also:** \_m_empty, \_m_pand, \_m_por, \_m_pxor

**Example:**
```
#include <stdio.h>
#include <mmintrin.h>
#define AS_QWORD "%16.16Lx"

__m64   a;
__m64   b = { 0x0123456789abcdef };  
__m64   c = { 0xfedcba9876543210 };  

void main()
{
    a = _m_pandn( b, c );
    printf( "m1="AS_QWORD"
            "m2="AS_QWORD"
            "mm="AS_QWORD",
            b, c, a );
}
```

The following is produced:

```
m1=0123456789abcdef
m2=fedcba9876543210
mm=fedcba9876543210
```

**Classification:** Intel

**Systems:** MACRO
Synopsis:  
#include <mmintrin.h>

__m64 _m_pcmpeqb(__m64 *m1, __m64 *m2);

Description:  
If the respective bytes of m1 are equal to the respective bytes of m2, the respective bytes of the result are set to all ones, otherwise they are set to all zeros.

Returns:  
The result of comparing the packed bytes of two 64-bit multimedia values is returned as a sequence of bytes (0xff for equal, 0x00 for not equal).

See Also:  
_m_empty, _m_pcmpeqd, _m_pcmpeqw, _m_pcmpgtb, _m_pcmpgtd, _m_pcmgtw

Example:  
#include <stdio.h>
#include <mmintrin.h>

#define AS_BYTES "%2.2x %2.2x %2.2x %2.2x " 
"%2.2x %2.2x %2.2x %2.2x"

__m64   a;
__m64   b = { 0x0004000030020001 };  
__m64   c = { 0xff7ffe800080007f };  

void main()
{
    a = _m_pcmpeqb( b, c );
    printf( "m1="ASgetBytes"
    "m2="ASgetBytes"
    "mm="ASgetBytes"
        b._8[7], b._8[6], b._8[5], b._8[4],
        b._8[3], b._8[2], b._8[1], b._8[0],
        c._8[7], c._8[6], c._8[5], c._8[4],
        c._8[3], c._8[2], c._8[1], c._8[0],
        a._8[7], a._8[6], a._8[5], a._8[4],
        a._8[3], a._8[2], a._8[1], a._8[0] );
}

produces the following:

m1=00 04 00 03 00 02 00 01
m2=ff 7f ff 80 00 80 00 7f
mm=00 00 00 00 ff 00 00 00

Classification:  Intel

Systems:  MACRO
Synopsis:  

```c
#include <mmintrin.h>
__m64 _m_pcmpeqd(__m64 *m1, __m64 *m2);
```

Description: If the respective double-words of \textit{m1} are equal to the respective double-words of \textit{m2}, the respective double-words of the result are set to all ones, otherwise they are set to all zeros.

Returns: The result of comparing the 32-bit packed double-words of two 64-bit multimedia values is returned as a sequence of double-words (0xffffffff for equal, 0x00000000 for not equal).

See Also: _m_empty, _m_pcmpeqb, _m_pcmpeqw, _m_pcmpgtb, _m_pcmpgtw, _m_pcmpgtw

Example: 

```c
#include <stdio.h>
#include <mmintrin.h>
#define AS_DWORDS "%8.8lx %8.8lx"

__m64 a;
__m64 b = { 0x0004000300020001 };  
__m64 c = { 0x000400030002007f };  

void main()
{
  a = _m_pcmpeqd( b, c );
  printf( "m1="AS_DWORDS\n"
          "m2="AS_DWORDS\n"
          "mm="AS_DWORDS\n",
          b._32[1], b._32[0],
          c._32[1], c._32[0],
          a._32[1], a._32[0] );
}
```

produces the following:

```
m1=00040003 00020001
m2=00040003 0002007f
mm=ffffffff 00000000
```

Classification: Intel

Systems: MACRO
Synopsis:

```c
#include <mmintrin.h>
__m64 _m_pcmpeqw(__m64 *m1, __m64 *m2);
```

Description:

If the respective words of \( m1 \) are equal to the respective words of \( m2 \), the respective words of the result are set to all ones, otherwise they are set to all zeros.

Returns:

The result of comparing the packed words of two 64-bit multimedia values is returned as a sequence of words (0xffff for equal, 0x0000 for not equal).

See Also:

\_m_empty, \_m_pcmpeqb, \_m_pcmpeqd, \_m_pcmpgtb, \_m_pcmpgtd, \_m_pcmpgtw

Example:

```c
#include <stdio.h>
#include <mmintrin.h>

#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"

__m64   a;
__m64   b = { 0x0004000300020001 ];
__m64   c = { 0x0004ff8000800001 ];

void main()
{
    a = _m_pcmpeqw( b, c );
    printf( "m1="AS_WORDS"
             "m2="AS_WORDS"
             "mm="AS_WORDS",
             b._16[3], b._16[2], b._16[1], b._16[0],
             c._16[3], c._16[2], c._16[1], c._16[0],
             a._16[3], a._16[2], a._16[1], a._16[0] );
}
```

produces the following:

```
m1=0004 0003 0002 0001
m2=0004 ff80 0080 0001
mm=ffff 0000 0000 ffff
```

Classification: Intel

Systems: MACRO
Synopsis:  
#include <mmintrin.h>
__m64 _m_pcmpgtb(__m64 *m1, __m64 *m2);

Description:  If the respective signed bytes of \( m_1 \) are greater than the respective signed bytes of \( m_2 \), the respective bytes of the result are set to all ones, otherwise they are set to all zeros.

Returns:  The result of comparing the packed signed bytes of two 64-bit multimedia values is returned as a sequence of bytes (0xff for greater than, 0x00 for not greater than).

See Also:  _m_empty, _m_pcmpeqb, _m_pcmpeqd, _m_pcmpeqw, _m_pcmpgtd, _m_pcmpgtw

Example:  
#include <stdio.h>
#include <mmintrin.h>

#define AS_BYTES "%2.2x %2.2x %2.2x %2.2x " \\
\"%2.2x %2.2x %2.2x %2.2x"

__m64   a;
__m64   b = { 0x0004000300020001 };
__m64   c = { 0xff7fff800080007f };

void main()
{
    a = _m_pcmpgtb( b, c );
    printf( "m1="AS_BYTES\n"m2="AS_BYTES\n"mm="AS_BYTES\n",
        b._8[7], b._8[6], b._8[5], b._8[4],
        b._8[3], b._8[2], b._8[1], b._8[0],
        c._8[7], c._8[6], c._8[5], c._8[4],
        c._8[3], c._8[2], c._8[1], c._8[0],
        a._8[7], a._8[6], a._8[5], a._8[4],
        a._8[3], a._8[2], a._8[1], a._8[0] );
}

produces the following:

\( m_1=00 \ 04 \ 00 \ 03 \ 00 \ 02 \ 00 \ 01 \)
\( m_2=ff \ 7f \ ff \ 80 \ 00 \ 00 \ 00 \ 7f \)
\( mm=ff \ 00 \ ff \ ff \ 00 \ 00 \ 00 \ 00 \)

Classification:  Intel

Systems:  MACRO
Synopsis:

#include <mmintrin.h>

__m64 _m_pcmpgtd(__m64 *m1, __m64 *m2);

Description:

If the respective signed double-words of \textit{m1} are greater than the respective signed double-words of \textit{m2},
the respective double-words of the result are set to all ones, otherwise they are set to all zeros.

Returns:

The result of comparing the 32-bit packed signed double-words of two 64-bit multimedia values is returned as a sequence of double-words (0xffffffff for greater than, 0x00000000 for not greater than).

See Also:

\_m\_empty, \_m\_pcmpeqb, \_m\_pcmpeqd, \_m\_pcmpeqw, \_m\_pcmptb, \_m\_pcmptw

Example:

```c
#include <stdio.h>
#include <mmintrin.h>

#define AS_DWORDS "%8.8lx %8.8lx"

__m64 a;
__m64 b = { 0x0004000400020001 };  
__m64 c = { 0x000400030080007f };  

void main()
{
    a = _m_pcmpgtd( b, c );
    printf( "m1="AS_DWORDS"
            "m2="AS_DWORDS"
            "mm="AS_DWORDS",
            b._32[1], b._32[0],
            c._32[1], c._32[0],
            a._32[1], a._32[0] );
}
```

produces the following:

```
m1=00040004 00020001
m2=00040003 0080007f
mm=ffffffff 00000000
```

Classification: Intel

Systems: MACRO
Synopsis:  
#include <mmintrin.h>
__m64 _m_pcmpgtw(__m64 *m1, __m64 *m2);

Description: If the respective signed words of \textit{m1} are greater than the respective signed words of \textit{m2}, the respective words of the result are set to all ones, otherwise they are set to all zeros.

Returns: The result of comparing the 16-bit packed signed words of two 64-bit multimedia values is returned as a sequence of words (0xffff for greater than, 0x0000 for not greater than).

See Also: _m_empty, _m_pcmpeqb, _m_pcmpeqd, _m_pcmpeqw, _m_pcmpgtb, _m_pcmpgtd

Example:  
#include <stdio.h>
#include <mmintrin.h>

#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"

__m64 a;
__m64 b = { 0x0005000300020001 };
__m64 c = { 0x0004ff8000800001 };

void main()
{
    a = _m_pcmpgtw( b, c );
    printf( "m1=%s\n", AS_WORDS, b._16[3], b._16[2], b._16[1], b._16[0],
            c._16[3], c._16[2], c._16[1], c._16[0],
            a._16[3], a._16[2], a._16[1], a._16[0] );
}

produces the following:

m1=0005 0003 0002 0001
m2=0004 ff80 0080 0001
mm=ffff ffff 0000 0000

Classification: Intel

Systems: MACRO
**Synopsis:**

```c
#include <mmintrin.h>

__m64 _m_pmaddwd(__m64 *m1, __m64 *m2);
```

**Description:**
The signed 16-bit words of `m1` are multiplied with the respective signed 16-bit words of `m2`. The 32-bit intermediate results are summed by pairs producing two 32-bit integers.

\[
\begin{align*}
\text{MM}[63-32] &= M1[63-48] \times M2[63-48] \\
&\quad + M1[47-32] \times M2[47-32] \\
\text{MM}[31-0] &= M1[31-16] \times M2[31-16] \\
&\quad + M1[15-0] \times M2[15-0]
\end{align*}
\]

In cases which overflow, the results are truncated. These two integers are packed into their respective elements of the result.

**Returns:**
The result of multiplying the packed signed 16-bit words of two 64-bit multimedia values and adding the 32-bit results pairwise is returned as packed double-words.

**See Also:**
`_m_empty`, `_m_pmulhw`, `_m_pmullw`

**Example:**

```c
#include <stdio.h>
#include <mmintrin.h>

#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"
#define AS_DWORDS "%8.8lx %8.8lx"

__m64   a;
__m64   b = { 0x0000006000123456 };  
__m64   c = { 0x0000000200010020 };  

void main()
{
    a = _m_pmaddwd( b, c );
    printf( "m1="AS_WORDS"\n"
            "m2="AS_WORDS"\n"
            "mm="AS_DWORDS"\n",
            b._16[3], b._16[2], b._16[1], b._16[0],
            c._16[3], c._16[2], c._16[1], c._16[0],
            a._32[1], a._32[0] );
}
```

produces the following:

```
m1=0000 0060 0012 3456
m2=0000 0002 0001 0020
mm=000000c0 00068ad2
```

**Classification:** Intel

**Systems:** MACRO
Synopsis:  
#include <mmintrin.h>  
__m64 _m_pmulhw(__m64 *m1, __m64 *m2);

Description:  
The signed 16-bit words of \textit{m1} are multiplied with the respective signed 16-bit words of \textit{m2}. The high-order 16-bits of each result are placed in the respective elements of the result.

Returns:  
The packed 16-bit words in \textit{m1} are multiplied with the packed 16-bit words in \textit{m2} and the high-order 16-bits of the results are returned.

See Also:  
\_m\_empty, \_m\_pmaddwd, \_m\_pmluw

Example:  
#include <stdio.h>  
#include <mmintrin.h>  

#define AS\_WORDS "%4.4x %4.4x %4.4x %4.4x"

__m64 a;
__m64 b = { 0x4000006000123456 };  
__m64 c = { 0x0008000210000020 };  

void main()
{
    a = _m_pmulhw( b, c );  
    printf( "m1=AS\_WORDS\n"  
                "m2=AS\_WORDS\n"  
                "mm=AS\_WORDS\n",  
               b._16[3], b._16[2], b._16[1], b._16[0],  
               c._16[3], c._16[2], c._16[1], c._16[0],  
               a._16[3], a._16[2], a._16[1], a._16[0] );
}

produces the following:

m1=4000 0060 0012 3456
m2=0008 0002 1000 0020
mm=0002 0000 0001 0006

Classification: Intel

Systems: MACRO
Synopsis:  
#include <mmintrin.h>
__m64 _m_pmullw(__m64 *m1, __m64 *m2);

Description:  The signed or unsigned 16-bit words of \textit{m1} are multiplied with the respective signed or unsigned 16-bit words of \textit{m2}. The low-order 16-bits of each result are placed in the respective elements of the result.

Returns:  The packed 16-bit words in \textit{m1} are multiplied with the packed 16-bit words in \textit{m2} and the low-order 16-bits of the results are returned.

See Also:  \_m_empty, \_m_pmaddwd, \_m_pmulhw

Example:  #include <stdio.h>
#include <mmintrin.h>

#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"

__m64 a;
__m64 b = { 0x4000006000123456 };  
__m64 c = { 0x0008002010000020 };  

void main()
{
    a = _m_pmullw( b, c );
    printf( "m1="AS_WORDS"
            "m2="AS_WORDS"
            "mm="AS_WORDS",
            b._16[3], b._16[2], b._16[1], b._16[0],
            c._16[3], c._16[2], c._16[1], c._16[0],
            a._16[3], a._16[2], a._16[1], a._16[0] );
}

produces the following:

m1=4000 0060 0012 3456
m2=0008 0002 1000 0020
mm=0000 00c0 2000 8ac0

Classification:  Intel

Systems:  MACRO
Synopsis:    #include <mmmintrin.h>
            __m64 _m_por(__m64 *m1, __m64 *m2);

Description:  A bit-wise logical OR is performed between 64-bit multimedia operands m1 and m2 and the result is stored in memory.

Returns:     The bit-wise logical OR of two 64-bit values is returned.

See Also:    _m_empty, _m_pand, _m_pandn, _m_pxor

Example:     #include <stdio.h>
             #include <mmmintrin.h>

             #define AS_QWORD "%16.16Lx"

             __m64   a;
             __m64   b = { 0x0123456789abcdef };  
             __m64   c = { 0xfedcba9876543210 };  

             void main()
             {  
                a = _m_por( b, c );
                printf( "m1="AS_QWORD"\n"
                        "m2="AS_QWORD"\n"
                        "mm="AS_QWORD"\n",
                        b, c, a );
             }  

             produces the following:

             ml=0123456789abcdef
             m2=fedcba9876543210
             mm=ffffffffffffffff

Classification:  Intel

Systems:     MACRO
Synopsis:  
```c
#include <mmintrin.h>
__m64 _m_pslld(__m64 *m, __m64 *count);
```

Description:  The 32-bit double-words in \( m \) are each independently shifted to the left by the scalar shift count in \( count \). The low-order bits of each element are filled with zeros. The shift count is interpreted as unsigned. Shift counts greater than 31 yield all zeros.

Returns:  Shift left each 32-bit double-word in \( m \) by an amount specified in \( count \) while shifting in zeros.

See Also:  _m_empty, _m_pslldi, _m_psllq, _m_psllqi, _m_psllw, _m_psllwi

Example:  
```c
#include <stdio.h>
#include <mmintrin.h>

#define AS_DWORDS "%8.8lx %8.8lx"
#define AS_QWORD "%16.16Lx"

__m64   a;
__m64   b = { 0x3f04800300020001 };  // Example 1
__m64   c = { 0x0000000000000002 };  // Example 2

void main()
{
    a = _m_pslld( b, c );
    printf( "m1=AS_DWORDS\n"
            "m2=AS_QWORD\n"
            "mm=AS_DWORDS\n",
            b._32[1], b._32[0],
            c,
            a._32[1], a._32[0] );
}
```

produces the following:

```
m1=3f048003 00020001
m2=0000000000000002
mm=fc12000c 00080004
```

Classification: Intel

Systems: MACRO
Synopsis:  
#include <mmintrin.h>
__m64 _m_pslldi(__m64 *m, int count);

Description:  The 32-bit double-words in m are each independently shifted to the left by the scalar shift count in count. The low-order bits of each element are filled with zeros. The shift count is interpreted as unsigned. Shift counts greater than 31 yield all zeros.

Returns:  Shift left each 32-bit double-word in m by an amount specified in count while shifting in zeros.

See Also:  _m_empty, _m_pslld, _m_pslldq, _m_pslldqi, _m_pslldw, _m_pslldwi

Example:  
#include <stdio.h>
#include <mmintrin.h>
#define AS_DWORDS "%8.8lx %8.8lx"

__m64 a;
__m64 b = { 0x3f04800300020001 }; 

void main()
{
  a = _m_pslldi( b, 2 );
  printf( "m ="AS_DWORDS"\n",
           "mm="AS_DWORDS"\n",
           b._32[1], b._32[0],
           a._32[1], a._32[0] );
}

produces the following:

m =3f048003 00020001
mm=fc12000c 00080004

Classification:  Intel

Systems:  MACRO
Synopsis:  
```
#include <mmintrin.h>
__m64 _m_psllq(__m64 *m, __m64 *count);
```

Description:  
The 64-bit quad-word in `m` is shifted to the left by the scalar shift count in `count`. The low-order bits are filled with zeros. The shift count is interpreted as unsigned. Shift counts greater than 63 yield all zeros.

Returns:  
Shift left the 64-bit quad-word in `m` by an amount specified in `count` while shifting in zeros.

See Also:  
`_m_empty`, `_m_pslld`, `_m_pslldi`, `_m_psllqi`, `_m_psllw`, `_m_psllwi`

Example:  
```
#include <stdio.h>
#include <mmintrin.h>
#define AS_QWORD "%16.16Lx"

__m64 a;
__m64 b = { 0x3f04800300020001 };  
__m64 c = { 0x0000000000000002 };  

void main()
{
    a = _m_psllq( b, c );
    printf( "m1=%AS_QWORD"
            "m2=%AS_QWORD"
            "mm=%AS_QWORD"
            , b, c, a );
}
```

produces the following:

m1=3f04800300020001
m2=0000000000000002
mm=fc12000c00080004

Classification:  
Intel

Systems:  
MACRO
Synopsis: #include <mmintrin.h>
__m64 _m_psllqi(__m64 *m, int count);

Description: The 64-bit quad-word in \textit{m} is shifted to the left by the scalar shift count in \textit{count}. The low-order bits are filled with zeros. The shift count is interpreted as unsigned. Shift counts greater than 63 yield all zeros.

Returns: Shift left the 64-bit quad-word in \textit{m} by an amount specified in \textit{count} while shifting in zeros.

See Also: \_m_empty, \_m_psllrd, \_m_pslldi, \_m_psllq, \_m_psllw, \_m_psllwi

Example: #include <stdio.h>
#include <mmintrin.h>

#define AS_QWORD "\%16.16Lx"

__m64 a;
__m64 b = { 0x3f04800300020001 };

void main()
{
  a = _m_psllqi( b, 2 );
  printf( "m ="AS_QWORD"\n"
          "mm="AS_QWORD"\n",
            b, a );
}

produces the following:

m =3f04800300020001
mm=fc12000c00080004

Classification: Intel

Systems: MACRO
Synopsis:  #include <mmintrin.h>
  __m64 _m_psllw(__m64 *m, __m64 *count);

Description:  The 16-bit words in m are each independently shifted to the left by the scalar shift count in count. The low-order bits of each element are filled with zeros. The shift count is interpreted as unsigned. Shift counts greater than 15 yield all zeros.

Returns:  Shift left each 16-bit word in m by an amount specified in count while shifting in zeros.

See Also:  _m_empty, _m_pslld, _m_pslldi, _m_psllq, _m_psllqi, _m_psllwi

Example:  #include <stdio.h>
#include <mmintrin.h>

#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"
#define AS_QWORD "%16.16Lx"

__m64   a;
__m64   b = { 0x3f04800300020001 };
__m64   c = { 0x0000000000000002 };

void main()
{
  a = _m_psllw( b, c );
  printf( "m1="%AS_WORDS"
    "m2="%AS_QWORD"
    "mm="%AS_WORDS"
    ,
    b._16[3], b._16[2], b._16[1], b._16[0],
    c,
    a._16[3], a._16[2], a._16[1], a._16[0] );
}

produces the following:

m1=3f04 8003 0002 0001
m2=0000000000000002
mm=fc10 000c 0008 0004

Classification:  Intel

Systems:  MACRO
Synopsis:  
```
#include <mmintrin.h>
__m64 _m_psllwi(__m64 *m, int count);
```

Description:  The 16-bit words in m are each independently shifted to the left by the scalar shift count in count. The low-order bits of each element are filled with zeros. The shift count is interpreted as unsigned. Shift counts greater than 15 yield all zeros.

Returns:  Shift left each 16-bit word in m by an amount specified in count while shifting in zeros.

See Also:  _m_empty, _m_pslld, _m_pslldi, _m_psllq, _m_psllqi, _m_psllw

Example:  
```
#include <stdio.h>
#include <mmintrin.h>
#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"

__m64 a;
__m64 b = { 0x3f04800300020001 };

void main()
{
    a = _m_psllwi( b, 2 );
    printf( "m ="AS_WORDS\n"
          "mm="AS_WORDS\n",
          b._16[3], b._16[2], b._16[1], b._16[0],
          a._16[3], a._16[2], a._16[1], a._16[0] );
}
```

produces the following:
```
m =3f04 8003 0002 0001
mm=fc10 000c 0008 0004
```

Classification:  Intel

Systems:  MACRO
Synopsis:  
#include <mmintrin.h>  
__m64 _m_psrad(__m64 *m, __m64 *count);

Description:  
The 32-bit signed double-words in m are each independently shifted to the right by the scalar shift count in count. The high-order bits of each element are filled with the initial value of the sign bit of each element. The shift count is interpreted as unsigned. Shift counts greater than 31 yield all ones or zeros depending on the initial value of the sign bit.

Returns:  
Shift right each 32-bit double-word in m by an amount specified in count while shifting in sign bits.

See Also:  
_m_empty, _m_psradi, _m_psraw, _m_psrawi

Example:  
#include <stdio.h>  
#include <mmintrin.h>  
#define AS_DWORDS "%8.8lx %8.8lx"  
#define AS_QWORD "%16.16Lx"  

__m64 a;  
__m64 b = { 0x3f04800300020001 };  
__m64 c = { 0x0000000000000002 };  

void main()  
{  
a = _m_psrad( b, c );  
printf( "m1="AS_DWORDS"\n"  
"m2="AS_QWORD"\n"  
"mm="AS_DWORDS"\n",  
b._32[1], b._32[0],  
c,  
a._32[1], a._32[0] );  
}

produces the following:

ml=3f048003 00020001  
m2=0000000000000002  
mm=0fc12000 00008000

Classification:  Intel

Systems:  MACRO
Synopsis: 
#include <mmintrin.h>
__m64 _m_psradi(__m64 *m, int count);

Description: The 32-bit signed double-words in m are each independently shifted to the right by the scalar shift count in count. The high-order bits of each element are filled with the initial value of the sign bit of each element. The shift count is interpreted as unsigned. Shift counts greater than 31 yield all ones or zeros depending on the initial value of the sign bit.

Returns: Shift right each 32-bit double-word in m by an amount specified in count while shifting in sign bits.

See Also: __m_empty, __m_psrad, __m_psraw, __m_psrawi

Example: 
#include <stdio.h>
#include <mmintrin.h>

#define AS_DWORDS "%8.8lx %8.8lx"

__m64   a;
__m64   b = { 0x3f04800300020001 };

void main()
{
    a = __m_psradi( b, 2 );
    printf( "m="AS_DWORDS"\n"
            "mm="AS_DWORDS"\n",
            b._32[1], b._32[0],
            a._32[1], a._32[0] );
}

produces the following:

m =3f048003 00020001
mm=0fc12000 00008000

Classification: Intel

Systems: MACRO
Synopsis:  
#include <mmintrin.h>
__m64 _m_psraw(__m64 *m, __m64 *count);

Description:  The 16-bit signed words in m are each independently shifted to the right by the scalar shift count in count. The high-order bits of each element are filled with the initial value of the sign bit of each element. The shift count is interpreted as unsigned. Shift counts greater than 15 yield all ones or zeros depending on the initial value of the sign bit.

Returns:  Shift right each 16-bit word in m by an amount specified in count while shifting in sign bits.

See Also:  _m_empty, _m_psrad, _m_psradi, _m_psrawi

Example:  
#include <stdio.h>
#include <mmintrin.h>
#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"
#define AS_QWORD "%16.16Lx"

__m64   a;
__m64   b = { 0x3f04800300040001 };  
__m64   c = { 0x0000000000000002 };  

void main()
{
    a = _m_psraw( b, c );
    printf( "m1="%AS_WORDS"
            "m2="%AS_QWORD"
            "mm="%AS_WORDS",
            b._16[3], b._16[2], b._16[1], b._16[0],
            c,
            a._16[3], a._16[2], a._16[1], a._16[0] );
}

produces the following:

m1=3f04 8003 0004 0001
m2=0000000000000002
mm=0fc1 e000 0001 0000

Classification:  Intel

Systems:  MACRO
Synopsis:  
#include <mmintrin.h>
__m64 _m_psrawi(__m64 *m, int count);

Description: The 16-bit signed words in m are each independently shifted to the right by the scalar shift count in count. The high-order bits of each element are filled with the initial value of the sign bit of each element. The shift count is interpreted as unsigned. Shift counts greater than 15 yield all ones or zeros depending on the initial value of the sign bit.

Returns: Shift right each 16-bit word in m by an amount specified in count while shifting in sign bits.

See Also: _m_empty, _m_psrad, _m_psradi, _m_psraw

Example:  
#include <stdio.h>
#include <mmintrin.h>

#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"

__m64 a;
__m64 b = { 0x3f04800300040001 };

void main()
{
    a = _m_psrawi( b, 2 );
    printf( "m ="AS_WORDS"\n"
            "mm="AS_WORDS"\n",
            b._16[3], b._16[2], b._16[1], b._16[0],
            a._16[3], a._16[2], a._16[1], a._16[0] );
}

produces the following:

m =3f04 8003 0004 0001
mm=0fc1 e000 0001 0000

Classification: Intel

Systems: MACRO
Synopsis:  
#include <mmintrin.h>
__m64 _m_psrld(__m64 *m, __m64 *count);

Description:  The 32-bit double-words in \textit{m} are each independently shifted to the right by the scalar shift count in \textit{count}. The high-order bits of each element are filled with zeros. The shift count is interpreted as unsigned. Shift counts greater than 31 yield all zeros.

Returns:  Shift right each 32-bit double-word in \textit{m} by an amount specified in \textit{count} while shifting in zeros.

See Also:  \_m\_empty, \_m\_psrldi, \_m\_psrlq, \_m\_psrlqi, \_m\_psrlw, \_m\_psrlwi

Example:  
#include <stdio.h>
#include <mmintrin.h>

#define AS_DWORDS "%8.8lx %8.8lx"
#define AS_QWORD "%16.16Lx"

__m64   a;
__m64   b = { 0x3f04800300020001 };
__m64   c = { 0x0000000000000002 };

void main()
{
    a = _m_psrld( b, c );
    printf( "m1="AS_DWORDS\n"
            "m2="AS_QWORD\n"
            "mm="AS_DWORDS\n",
            b._32[1], b._32[0],
            c,
            a._32[1], a._32[0] );
}

produces the following:

m1=3f048003 00020001
m2=0000000000000002
mm=0fc12000 00008000

Classification:  Intel

Systems:  MACRO
Synopsis:  
#include <mmintrin.h>
__m64 _m_psrldi(__m64 *m, int count);

Description:  The 32-bit double-words in m are each independently shifted to the right by the scalar shift count in count. The high-order bits of each element are filled with zeros. The shift count is interpreted as unsigned. Shift counts greater than 31 yield all zeros.

Returns:  Shift right each 32-bit double-word in m by an amount specified in count while shifting in zeros.

See Also:  _m_empty, _m_psrld, _m_psrlq, _m_psrlqi, _m_psrlw, _m_psrlwi

Example:  
#include <stdio.h>
#include <mmintrin.h>

#define AS_DWORDS "%8.8lx %8.8lx"

__m64   a;
__m64   b = { 0x3f04800300020001 };

void main()
{
    a = _m_psrldi( b, 2 );
    printf( "m =AS_DWORDS\n"
        "mm=AS_DWORDS\n",
        b._32[1], b._32[0],
        a._32[1], a._32[0] );
}

produces the following:

m =3f048003 00020001
mm=0fc12000 00008000

Classification:  Intel

Systems:  MACRO
Synopsis:  
#include <mmintrin.h>  
__m64 _m_psrlq(__m64 *m, __m64 *count);

Description:  
The 64-bit quad-word in \textit{m} is shifted to the right by the scalar shift count in \textit{count}. The high-order bits are filled with zeros. The shift count is interpreted as unsigned. Shift counts greater than 63 yield all zeros.

Returns:  
Shift right the 64-bit quad-word in \textit{m} by an amount specified in \textit{count} while shifting in zeros.

See Also:  
\_m\_empty, \_m\_psrld, \_m\_psrldi, \_m\_psrlq, \_m\_psrlw, \_m\_psrlwi

Example:  
#include <stdio.h>  
#include <mmintrin.h>
#define AS_QWORD "%16.16Lx"

__m64   a;
__m64   b = { 0x3f04800300020001 };  
__m64   c = { 0x0000000000000002 };  
void main()
{
    a = _m_psrlq( b, c );
    printf( "m1="AS_QWORD"\n"
            "m2="AS_QWORD"\n"
            "mm="AS_QWORD"\n",
            b, c, a );
}

produces the following:

\texttt{ml=3f04800300020001}
\texttt{m2=0000000000000002}
\texttt{mm=0fc12000c0008000}

Classification:  Intel

Systems:  MACRO
Synopsis:  
#include <mmintrin.h>
__m64 _m_psrlqi(__m64 *m, int count);

Description:  The 64-bit quad-word in \textit{m} is shifted to the right by the scalar shift count in \textit{count}. The high-order bits are filled with zeros. The shift count is interpreted as unsigned. Shift counts greater than 63 yield all zeros.

Returns:  Shift right the 64-bit quad-word in \textit{m} by an amount specified in \textit{count} while shifting in zeros.

See Also:  _m_empty, _m_psrl, _m_psrlq, _m_psrlw

Example:  
#include <stdio.h>
#include <mmintrin.h>

#define AS_QWORD "%16.16Lx"

__m64 a;
__m64 b = { 0x3f04800300020001 };

void main()
{
    a = _m_psrlqi( b, 2 );
    printf( "m="AS_QWORD"\n"
        "mm="AS_QWORD"\n",
        b, a );
}

produces the following:

m =3f04800300020001
mm=0fc12000c0008000

Classification:  Intel

Systems:  MACRO
Synopsis:

```
#include <mmintrin.h>
__m64 _m_psrlw(__m64 *m, __m64 *count);
```

Description: The 16-bit words in `m` are each independently shifted to the right by the scalar shift count in `count`. The high-order bits of each element are filled with zeros. The shift count is interpreted as unsigned. Shift counts greater than 15 yield all zeros.

Returns: Shift right each 16-bit word in `m` by an amount specified in `count` while shifting in zeros.

See Also: `_m_empty`, `_m_psrl`, `_m_psrlid`, `_m_psrlq`, `_m_psrlqi`, `_m_psrlw`

Example:

```
#include <stdio.h>
#include <mmintrin.h>

#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"
#define AS_QWORD "%16.16Lx"

__m64 a;
__m64 b = { 0x3f04800300040001 };
__m64 c = { 0x0000000000000002 };

void main()
{
    a = _m_psrlw( b, c );
    printf( "m1="AS_WORDS"
             "m2="AS_QWORD"
             "mm="AS_WORDS",
             b._16[3], b._16[2], b._16[1], b._16[0],
             c,
             a._16[3], a._16[2], a._16[1], a._16[0] );
}
```

produces the following:

```
m1=3f04 8003 0004 0001
m2=0000000000000002
mm=0fc1 2000 0001 0000
```

Classification: Intel

Systems: MACRO
Synopsis:  
#include <mmmintrin.h>
__m64 _m_psrlwi(__m64 *m, int count);

Description:  The 16-bit words in \textit{m} are each independently shifted to the right by the scalar shift count in \textit{count}. The high-order bits of each element are filled with zeros. The shift count is interpreted as unsigned. Shift counts greater than 15 yield all zeros.

Returns:  Shift right each 16-bit word in \textit{m} by an amount specified in \textit{count} while shifting in zeros.

See Also:  \_m_empty, \_m_psrld, \_m_psrldi, \_m_psrqlq, \_m_psrqli, \_m_psrllw

Example:  
#include <stdio.h>
#include <mmmintrin.h>
#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"

__m64   a;
__m64   b = { 0x3f04800300040001 };

void main()
{
    a = _m_psrlwi( b, 2 );
    printf( "m ="AS_WORDS"\n"
            "mm="AS_WORDS"\n",
            b._16[3], b._16[2], b._16[1], b._16[0],
            a._16[3], a._16[2], a._16[1], a._16[0] );
}

produces the following:

\textit{m} =3f04 8003 0004 0001
\textit{mm}=0fc1 2000 0001 0000

Classification:  Intel

Systems:  MACRO
Synopsis:  
#include <mmintrin.h>

__m64 _m_psubb(__m64 *m1, __m64 *m2);

Description:  The signed or unsigned 8-bit bytes of \texttt{m2} are subtracted from the respective signed or unsigned 8-bit bytes of \texttt{m1} and the result is stored in memory. If any result element does not fit into 8 bits (underflow or overflow), the lower 8 bits of the result elements are stored (i.e., truncation takes place).

Returns:  The result of subtracting the packed bytes of one 64-bit multimedia value from another is returned.

See Also:  \_m_empty, \_m_psubd, \_m_psubsb, \_m_psubsw, \_m_psubusb, \_m_psubusw, \_m_psubw

Example:  
#include <stdio.h>
#include <mmintrin.h>

#define AS_BYTES "%2.2x %2.2x %2.2x %2.2x " \n"%2.2x %2.2x %2.2x %2.2x"

__m64   a;
__m64   b = { 0x0123456789abcdef };
__m64   c = { 0xfedcba9876543210 };

void main()
{
    a = _m_psubb( b, c );
    printf( "m1="AS_BYTES"
            "m2="AS_BYTES"
            "mm="AS_BYTES"
            ,
            b._8[7], b._8[6], b._8[5], b._8[4],
            b._8[3], b._8[2], b._8[1], b._8[0],
            c._8[7], c._8[6], c._8[5], c._8[4],
            c._8[3], c._8[2], c._8[1], c._8[0],
            a._8[7], a._8[6], a._8[5], a._8[4],
            a._8[3], a._8[2], a._8[1], a._8[0] );
}

produces the following:

ml=01 23 45 67 89 ab cd ef
m2=fe dc ba 98 76 54 32 10
mm=03 47 8b cf 13 57 9b df

Classification:  Intel

Systems:  MACRO
Synopsis: 

```c
#include <mmintrin.h>
__m64 _m_psubd(__m64 *m1, __m64 *m2);
```

Description: The signed or unsigned 32-bit double-words of \textit{m2} are subtracted from the respective signed or unsigned 32-bit double-words of \textit{m1} and the result is stored in memory. If any result element does not fit into 32 bits (underflow or overflow), the lower 32-bits of the result elements are stored (i.e., truncation takes place).

Returns: The result of subtracting one set of packed double-words from a second set of packed double-words is returned.

See Also: \_m_empty, _m_psubb, _m_psubsb, _m_psubsw, _m_psubusb, _m_psubusw, _m_psubw

Example: 

```c
#include <stdio.h>
#include <mmintrin.h>
#define AS_DWORDS "%8.8lx %8.8lx"

__m64 a;
__m64 b = { 0x0123456789abcdef };  
__m64 c = { 0xfedcba9876543210 }; 

void main()
{
    a = _m_psubd( b, c );
    printf( "m1="AS_DWORDS\n"
            "m2="AS_DWORDS\n"
            "mm="AS_DWORDS\n",
            "m1="AS_DWORDS,  
            "m2="AS_DWORDS,  
            "mm="AS_DWORDS,  
            b._32[1], b._32[0],
            c._32[1], c._32[0],
            a._32[1], a._32[0] );
}
```

produces the following:

```
m1=01234567 89abcdef
m2=fedcba98 76543210
mm=02468acf 13579bdf
```

Classification: Intel

Systems: MACRO
Synopsis:  
#include <mmintrin.h>
__m64 _m_psubsb(__m64 *m1, __m64 *m2);

Description:  
The signed 8-bit bytes of \texttt{m2} are subtracted from the respective signed 8-bit bytes of \texttt{m1} and the result is stored in memory. Saturation occurs when a result exceeds the range of a signed byte. In the case where a result is a byte larger than 0x7f (overflow), it is clamped to 0x7f. In the case where a result is a byte smaller than 0x80 (underflow), it is clamped to 0x80.

Returns:  
The result of subtracting the packed signed bytes, with saturation, of one 64-bit multimedia value from a second multimedia value is returned.

See Also:  
\_m\_empty, \_m\_psubb, \_m\_psubd, \_m\_psubsw, \_m\_psubusb, \_m\_psubusw, \_m\_psubw

Example:  
#include <stdio.h>
#include <mmintrin.h>

#define AS\_BYTES "%2.2x %2.2x %2.2x %2.2x " \n"%2.2x %2.2x %2.2x %2.2x"

__m64 a;
__m64 b = { 0x8aacceef02244668 };
__m64 c = { 0x76543211fedcba98 };

void main()
{
    a = _m_psubsb( b, c );
    printf( "m1="AS\_BYTES"\n"  
            "m2="AS\_BYTES"\n"  
            "mm="AS\_BYTES"\n"
            ,
            b._8[7], b._8[6], b._8[5], b._8[4],
            b._8[3], b._8[2], b._8[1], b._8[0],
            c._8[7], c._8[6], c._8[5], c._8[4],
            c._8[3], c._8[2], c._8[1], c._8[0],
            a._8[7], a._8[6], a._8[5], a._8[4],
            a._8[3], a._8[2], a._8[1], a._8[0] );
}

produces the following:

m1=8a ac ce ef 02 24 46 68
m2=76 54 32 11 fe dc ba 98
mm=80 80 9c de 04 48 7f 7f

Classification:  
Intel

Systems:  
MACRO
Synopsis:  
#include <mmintrin.h>
__m64 _m_psubsw(__m64 *m1, __m64 *m2);

Description:  The signed 16-bit words of m2 are subtracted from the respective signed 16-bit words of m1 and the result is stored in memory. Saturation occurs when a result exceeds the range of a signed word. In the case where a result is a word larger than 0x7fff (overflow), it is clamped to 0x7fff. In the case where a result is a word smaller than 0x8000 (underflow), it is clamped to 0x8000.

Returns:  The result of subtracting the packed signed words, with saturation, of one 64-bit multimedia value from a second multimedia value is returned.

See Also:  _m_empty, _m_psubb, _m_psubd, _m_psubsb, _m_psubusb, _m_psubusw, _m_psubw

Example:  
#include <stdio.h>
#include <mmintrin.h>

#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"

__m64   a;
__m64   b = { 0x8aacceef02244668 };
__m64   c = { 0x76543211fedcba98 };

void main()
{
    a = _m_psubsw( b, c );
    printf( "m1="AS_WORDS"
            "m2="AS_WORDS"
            "mm="AS_WORDS",
            b._16[3], b._16[2], b._16[1], b._16[0],
            c._16[3], c._16[2], c._16[1], c._16[0],
            a._16[3], a._16[2], a._16[1], a._16[0] );
}

produces the following:

m1=8aac ceef 0224 4668
m2=7654 3211 fedc ba98
mm=8000 9cde 0348 7fff

Classification:  Intel

Systems:  MACRO
Synopsis:  
#include <mmintrin.h>  
__m64 _m_psubusb(__m64 *m1, __m64 *m2);

Description:  
The unsigned 8-bit bytes of m2 are subtracted from the respective unsigned 8-bit bytes of m1 and the result is stored in memory.  Saturation occurs when a result is less than zero.  If a result is less than zero, it is clamped to 0xff.

Returns:  
The result of subtracting the packed unsigned bytes, with saturation, of one 64-bit multimedia value from a second multimedia value is returned.

See Also:  
_m_empty, _m_psubb, _m_psubd, _m_psubsb, _m_psubsw, _m_psubusw, _m_psubw

Example:  
#include <stdio.h>  
#include <mmintrin.h>

#define AS_BYTES "%2.2x %2.2x %2.2x %2.2x "  
"%2.2x %2.2x %2.2x %2.2x"

__m64 a;  
__m64 b = { 0x8aacceef02244668 };  
__m64 c = { 0x76543211fedcba98 };  

void main()
{
    a = _m_psubusb( b, c );  
    printf( "m1="AS_BYTES"
    "m2="AS_BYTES"
    "mm="AS_BYTES"
    b._8[7], b._8[6], b._8[5], b._8[4],  
b._8[3], b._8[2], b._8[1], b._8[0],  
c._8[7], c._8[6], c._8[5], c._8[4],  
c._8[3], c._8[2], c._8[1], c._8[0],  
a._8[7], a._8[6], a._8[5], a._8[4],  
a._8[3], a._8[2], a._8[1], a._8[0] );
}

produces the following:

m1=8a ac ce ef 02 24 46 68  
m2=76 54 32 11 fe dc ba 98  
mm=14 58 9c de 00 00 00 00

Classification:  
Intel

Systems:  
MACRO
Synopsis:  
#include <mmintrin.h>
__m64 __m_psubusw(__m64 *m1, __m64 *m2);

Description:  The unsigned 16-bit words of m2 are subtracted from the respective unsigned 16-bit words of m1 and the result is stored in memory. Saturation occurs when a result is less than zero. If a result is less than zero, it is clamped to 0xffff.

Returns:  The result of subtracting the packed unsigned words, with saturation, of one 64-bit multimedia value from a second multimedia value is returned.

See Also:  __m_empty, __m_psubb, __m_psubd, __m_psubsw, __m_psubusb, __m_psubw

Example:  
#include <stdio.h>
#include <mmintrin.h>
#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"

__m64 a;
__m64 b = { 0x8aacceef02244668 };
__m64 c = { 0x76543211fedcba98 };

void main()
{
    a = __m_psubusw( b, c );
    printf( "m1="AS_WORDS\n"
            "m2="AS_WORDS\n"
            "mm="AS_WORDS\n",
            b._16[3], b._16[2], b._16[1], b._16[0],
            c._16[3], c._16[2], c._16[1], c._16[0],
            a._16[3], a._16[2], a._16[1], a._16[0] );
}

produces the following:

m1=8aac ceef 0224 4668
m2=7654 3211 fedc ba98
mm=1458 9cde 0000 0000

Classification:  Intel

Systems:  MACRO
Synopsis: 

```c
#include <mmintrin.h>
__m64 _m_psubw(__m64 *m1, __m64 *m2);
```

Description: The signed or unsigned 16-bit words of `m2` are subtracted from the respective signed or unsigned 16-bit words of `m1` and the result is stored in memory. If any result element does not fit into 16 bits (underflow or overflow), the lower 16 bits of the result elements are stored (i.e., truncation takes place).

Returns: The result of subtracting the packed words of two 64-bit multimedia values is returned.

See Also: `_m_empty`, `_m_psubb`, `_m_psubd`, `_m_psubsb`, `_m_psubsw`, `_m_psubusb`, `_m_psubusw`

Example: 

```c
#include <stdio.h>
#include <mmintrin.h>

#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"

__m64   a;
__m64   b = { 0x0123456789abcdef };  
__m64   c = { 0xfedcba9876543210 };  

void main()
{
    a = _m_psubw( b, c );
    printf( "m1="AS_WORDS"
    "m2="AS_WORDS"
    "mm="AS_WORDS",
    b._16[3], b._16[2], b._16[1], b._16[0],
    c._16[3], c._16[2], c._16[1], c._16[0],
    a._16[3], a._16[2], a._16[1], a._16[0] );
}
```

produces the following:

```
m1=0123 4567 89ab cdef
m2=fedc ba98 7654 3210
mm=0247 8acf 1357 9bdf
```

Classification: Intel

Systems: MACRO
Synopsis:  
```c
#include <mmintrin.h>
__m64 __m_punpckhbw(__m64 *m1, __m64 *m2);
```

Description:  The `__m_punpckhbw` function performs an interleaved unpack of the high-order data elements of `m1` and `m2`. It ignores the low-order bytes. When unpacking from a memory operand, the full 64-bit operand is accessed from memory but only the high-order 32 bits are utilized. By choosing `m1` or `m2` to be zero, an unpacking of byte elements into word elements is performed.

```
\[ \begin{array}{c|c|c|c|c|c|c|c|}
\hline
| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | \\
\hline
b7 & b5 & b3 & b1 \\
\hline
\end{array} \quad \begin{array}{c|c|c|c|c|c|c|c|}
\hline
| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | \\
\hline
b6 & b4 & b2 & b0 \\
\hline
\end{array} \quad \begin{array}{c|c|c|c|c|c|c|c|}
\hline
| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | \\
\hline
\end{array} \quad \begin{array}{c|c|c|c|c|c|c|c|}
\hline
\end{array}
\]
```

Returns: The result of the interleaved unpacking of the high-order bytes of two multimedia values is returned.

See Also:  `_m_empty`, `_m_punpckhdq`, `_m_punpckhwd`, `_m_punpckhbw`, `_m_punpckldq`, `_m_punpcklwd`

Example:  
```c
#include <stdio.h>
#include <mmintrin.h>
#define AS_BYTES "%2.2x %2.2x %2.2x %2.2x " \
    "%2.2x %2.2x %2.2x %2.2x"

__m64   a;
__m64   b = { 0x0004000300020001 };  
__m64   c = { 0xff7fff800080007f }; 

void main()
{
    a = __m_punpckhbw( b, c );
    printf( "m2=AS_BYTES " \
        "m1=AS_BYTES\n" \
        "mm=AS_BYTES\n", \
        c._8[7], c._8[6], c._8[5], c._8[4], \
        c._8[3], c._8[2], c._8[1], c._8[0], \
        b._8[7], b._8[6], b._8[5], b._8[4], \
        b._8[3], b._8[2], b._8[1], b._8[0], \
        a._8[7], a._8[6], a._8[5], a._8[4], \
        a._8[3], a._8[2], a._8[1], a._8[0] );
}
```

produces the following:

```
m2=ff 7f ff 00 00 00 00 7f m1=00 04 00 03 00 02 00 01
mm=ff 00 7f 04 ff 00 80 03
```
Classification: Intel

Systems: MACRO
Synopsis:  

```c
#include <mmintrin.h>
__m64 _m_punpckhdq(__m64 *m1, __m64 *m2);
```

Description:  The `_m_punpckhdq` function performs an interleaved unpack of the high-order data elements of `m1` and `m2`. It ignores the low-order double-words. When unpacking from a memory operand, the full 64-bit operand is accessed from memory but only the high-order 32 bits are utilized.

<table>
<thead>
<tr>
<th>m2</th>
<th>m1</th>
</tr>
</thead>
<tbody>
<tr>
<td>d1</td>
<td>d0</td>
</tr>
<tr>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>d1</td>
<td>d0</td>
</tr>
</tbody>
</table>

result

Returns:  The result of the interleaved unpacking of the high-order double-words of two multimedia values is returned.

See Also:  `_m_empty`, `_m_punpckhbw`, `_m_punpckhwd`, `_m_punpckldq`, `_m_punpcklwd`

Example:  

```c
#include <stdio.h>
#include <mmintrin.h>
#define AS_DWORDS "%8.8lx %8.8lx"

__m64   a;
__m64   b = { 0x0004000300020001 };
__m64   c = { 0xff7fff800080007f };

void main()
{
    a = _m_punpckhdq( b, c );
    printf( "m2=%s m1=%s mm=%s\n",
            AS_DWORDS, b._32[1], b._32[0],
            AS_DWORDS, c._32[1], c._32[0],
            AS_DWORDS, a._32[1], a._32[0] );
}
```

produces the following:

```
m2=ff7fff80 0080007f m1=00040003 00020001
mm=ff7fff80 00040003
```

Classification:  Intel

Systems:  MACRO
Synopsis:  
#include <mmintrin.h>
__m64 _m_punpckhwd(__m64 *m1, __m64 *m2);

Description: The _m_punpckhwd function performs an interleaved unpack of the high-order data elements of m1 and m2. It ignores the low-order words. When unpacking from a memory operand, the full 64-bit operand is accessed from memory but only the high-order 32 bits are utilized. By choosing m1 or m2 to be zero, an unpacking of word elements into double-word elements is performed.

<table>
<thead>
<tr>
<th>m2</th>
<th>m1</th>
</tr>
</thead>
<tbody>
<tr>
<td>w3</td>
<td>w2</td>
</tr>
<tr>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>w3</td>
<td>w1</td>
</tr>
</tbody>
</table>

result

| w3 | w2 | w1 | w0 |
|------------------------- |

Returns: The result of the interleaved unpacking of the high-order words of two multimedia values is returned.

See Also: _m_empty, _m_punpckhbw, _m_punpckhdq, _m_punpcklw, _m_punpcklwd

Example:  
#include <stdio.h>
#include <mmintrin.h>
#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"

__m64 a;
__m64 b = { 0x0004000300020001 ];
__m64 c = { 0xff7fff800080007f ];

void main()
{
    a = _m_punpckhwd( b, c );
    printf( "m2=%s m1=%s mm=%s\n",
            AS_WORDS, b._16[3], b._16[2], b._16[1], b._16[0],
            AS_WORDS, c._16[3], c._16[2], c._16[1], c._16[0],
            AS_WORDS, a._16[3], a._16[2], a._16[1], a._16[0] );
}

produces the following:

m2=ff7f ff80 0080 007f m1=0004 0003 0002 0001
mm=ff7f 0004 ff80 0003

Classification: Intel

Systems: MACRO
Synopsis: #include <mmintrin.h>

__m64 _m_punpcklbw(__m64 *m1, __m64 *m2);

Description: The _m_punpcklbw function performs an interleaved unpack of the low-order data elements of m1 and m2. It ignores the high-order bytes. When unpacking from a memory operand, 32 bits are accessed and all are utilized by the instruction. By choosing m1 or m2 to be zero, an unpacking of byte elements into word elements is performed.

+------------------------+------------------------+
| m2                     | m1                     |
+------------------------+------------------------+
| b6|b5|b4|b3|b2|b1|b0| | b7|b6|b5|b4|b3|b2|b1|b0|
| b7|b5|b3|b1|   |   |   | | b6|b4|b2|b0|
+------------------------+------------------------+

Returns: The result of the interleaved unpacking of the low-order bytes of two multimedia values is returned.

See Also: _m_empty, _m_punpckhbw, _m_punpckhdq, _m_punpckhwd, _m_punpckldq, _m_punpcklwd

Example:

```c
#include <stdio.h>
#include <mmintrin.h>

#define AS_BYTES "%2.2x %2.2x %2.2x %2.2x " \
    "%2.2x %2.2x %2.2x %2.2x"

__m64   a;
__m64   b = { 0x000200013478bcf0 };  
__m64   c = { 0x00800007f12569ade };    

void main()
{
    a = _m_punpcklbw( b, c );
    printf("m2=\"AS_BYTES\" "  
        "m1=\"AS_BYTES\"n"
        "mm=\"AS_BYTES\"n",
        c._8[7], c._8[6], c._8[5], c._8[4],
        c._8[3], c._8[2], c._8[1], c._8[0],
        b._8[7], b._8[6], b._8[5], b._8[4],
        b._8[3], b._8[2], b._8[1], b._8[0],
        a._8[7], a._8[6], a._8[5], a._8[4],
        a._8[3], a._8[2], a._8[1], a._8[0] );
}
```

produces the following:

```
m2=00 80 00 7f 12 56 9a de m1=00 02 00 01 34 78 bc f0
mm=12 34 56 78 9a bc de f0
```
Classification: Intel

Systems: MACRO
**Synopsis:**

```c
#include <mmintrin.h>
__m64 _m_punpckldq(__m64 *m1, __m64 *m2);
```

**Description:**

The `_m_punpckldq` function performs an interleaved unpack of the low-order data elements of `m1` and `m2`. It ignores the high-order double-words. When unpacking from a memory operand, 32 bits are accessed and all are utilized by the instruction.

![Diagram](image)

**Returns:**

The result of the interleaved unpacking of the low-order double-words of two multimedia values is returned.

**See Also:**

`_m_empty`, `_m_punpckhbw`, `_m_punpckhdq`, `_m_punpckhwd`, `_m_punpcklbw`, `_m_punpcklwd`

**Example:**

```c
#include <stdio.h>
#include <mmintrin.h>

#define AS_DWORDS "%8.8lx %8.8lx"

__m64   a;
__m64   b = { 0x0004000300020001 };  
__m64   c = { 0xff7fff800080007f };  

void main()
{
    a = _m_punpckldq( b, c );
    printf( "m2=%s m1=%s
            mm=%s
", 
            d2._32[1], c._32[0],
            b._32[1], b._32[0],
            a._32[1], a._32[0] );
}
```

produces the following:

```
m2=ff7fff80 0080007f m1=00040003 00020001
```

**Classification:** Intel

**Systems:** MACRO
Synopsis:    #include <mmintrin.h>
__m64 _m_punpcklwd(__m64 *m1, __m64 *m2);

Description: The _m_punpcklwd function performs an interleaved unpack of the low-order data elements of m1 and m2. It ignores the high-order words. When unpacking from a memory operand, 32 bits are accessed and all are utilized by the instruction. By choosing m1 or m2 to be zero, an unpacking of word elements into double-word elements is performed.

```
| w3 | w2 | w1 | w0 |
V   V
w3  w1
```

result

Returns: The result of the interleaved unpacking of the low-order words of two multimedia values is returned.

See Also: _m_empty, _m_punpckhbw, _m_punpckhdq, _m_punpckhwd, _m_punpcklbdq

Example: #include <stdio.h>
#include <mmintrin.h>

#define AS_WORDS "%4.4x %4.4x %4.4x %4.4x"

__m64 a;
__m64 b = { 0x0004000300020001 };  
__m64 c = { 0xff7fff800080007f };  

void main() 
{
    a = _m_punpcklwd( b, c );  
    printf( "m2=",AS_WORDS "m1=",AS_WORDS "mm=",AS_WORDS  
            c._16[3], c._16[2], c._16[1], c._16[0],
            b._16[3], b._16[2], b._16[1], b._16[0],
            a._16[3], a._16[2], a._16[1], a._16[0] );
}

produces the following:

m2=ff7f ff80 0080 007f m1=0004 0003 0002 0001
mm=0080 0002 007f 0001

Classification: Intel

Systems: MACRO

Synopsis: #include <mmintrin.h>
__m64 _m_pxor(__m64 *m1, __m64 *m2);

Description: A bit-wise logical XOR is performed between 64-bit multimedia operands m1 and m2 and the result is stored in memory.

Returns: The bit-wise logical exclusive OR of two 64-bit values is returned.

See Also: _m_empty, _m_pand, _m_pandn, _m_por

Example: #include <stdio.h>
#include <mmintrin.h>

#define AS_QWORD "%16.16Lx"

__m64   a;
__m64   b = { 0x0123456789abcdef };  
__m64   c = { 0xfedcba9876543210 };  

void main()
{
    a = _m_pxor( b, c );
    printf( "m1=AS_QWORD\n"
            "m2=AS_QWORD\n"
            "mm=AS_QWORD\n"
            , b, c, a );
}

produces the following:

m1=0123456789abcdef
m2=fedcba9876543210
mm=ffffffffffffffff

Classification: Intel

Systems: MACRO
**Synopsis:**

```c
#include <malloc.h>

size_t _msize( void *buffer );
size_t _bmsize( __segment seg, void __based(void) *buffer );
size_t _fmsize( void __far *buffer );
size_t _nmsize( void __near *buffer );
```

**Description:**

The `_msize` functions return the size of the memory block pointed to by `buffer` that was allocated by a call to the appropriate version of the `calloc`, `malloc`, or `realloc` functions.

You must use the correct `_msize` function as listed below depending on which heap the memory block belongs to.

<table>
<thead>
<tr>
<th>Function</th>
<th>Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_msize</code></td>
<td>Depends on data model of the program</td>
</tr>
<tr>
<td><code>_bmsize</code></td>
<td>Based heap specified by <code>seg</code> value</td>
</tr>
<tr>
<td><code>_fmsize</code></td>
<td>Far heap (outside the default data segment)</td>
</tr>
<tr>
<td><code>_nmsize</code></td>
<td>Near heap (inside the default data segment)</td>
</tr>
</tbody>
</table>

In small data models (small and medium memory models), `_msize` maps to `_nmsize`. In large data models (compact, large and huge memory models), `_msize` maps to `_fmsize`.

**Returns:** The `_msize` functions return the size of the memory block pointed to by `buffer`.

**See Also:**

`calloc Functions`, `_expand Functions`, `free Functions`, `halloc`, `hfree`, `malloc Functions`, `realloc Functions`, `sbrk`

**Example:**

```c
#include <stdio.h>
#include <malloc.h>

void main()
{
    void *buffer;

    buffer = malloc( 999 );
    printf( "Size of block is %u bytes\n",
            _msize( buffer ) );
}
```

produces the following:

```
Size of block is 1000 bytes
```

**Classification:** WATCOM

**Systems:**

- `_msize` - All, Netware
- `_bmsize` - DOS/16, Windows, QNX/16, OS/2 1.x (all)
- `_fmsize` - DOS/16, Windows, QNX/16, OS/2 1.x (all)
- `_nmsize` - DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2 1.x (MT), OS/2-32
Synopsis:  #include <mmintrin.h>
int _m_to_int(__m64 *__m);

Description:  The _m_to_int function returns the low-order 32 bits of a multimedia value.

Returns:  The low-order 32 bits of a multimedia value are fetched and returned as the result.

See Also:  _m_empty, _m_from_int, _m_packsswb, _m_paddb, _m_pand, _m_empty, _m_pcmpeqb, _m_pmaddwd, _m_psllw, _m_psraw, _m_psrlw, _m_empty, _m_psubb, _m_punpckhbw, _m_pmaddwd

Example:  #include <stdio.h>
#include <mmintrin.h>

__m64 b = { 0x0123456789abcdef };
int j;

void main()
{
    j = _m_to_int( b );
    printf( "m=%16.16Lx int=%8.8lx
", b, j );
}

produces the following:

m=0123456789abcdef int=89abcdef

Classification:  Intel

Systems:  MACRO
Synopsis: 
#include <i86.h>
void nosound( void );

Description: The nosound function turns off the PC’s speaker.

Returns: The nosound function has no return value.

See Also: delay, sound

Example: 
#include <i86.h>

    void main()
    {
        sound( 200 );
        delay( 500 ); /* delay for 1/2 second */
        nosound();
    }

Classification: Intel

Systems: DOS, Windows, Win386, QNX
Synopsis:
#include <stddef.h>
size_t offsetof( composite, name );

Description: The offsetof macro returns the offset of the element name within the struct or union composite. This provides a portable method to determine the offset.

Returns: The offsetof function returns the offset of name.

Example:
#include <stdio.h>
#include <stddef.h>

struct new_def
{
  char *first;
  char second[10];
  int third;
};

tvoid main()
{
  printf( "%first:%d second:%d third:%d\n", 
    offsetof( struct new_def, first ),
    offsetof( struct new_def, second ),
    offsetof( struct new_def, third ) );
}

produces the following:

In a small data model, the following would result:

first:0 second:2 third:12

In a large data model, the following would result:

first:0 second:4 third:14

Classification: ANSI

Systems: MACRO
Synopsis:  
#include <stdlib.h>
onexit_t onexit( onexit_t func );

Description:  The onexit function is passed the address of function func to be called when the program terminates normally. Successive calls to onexit create a list of functions that will be executed on a "last-in, first-out" basis. No more than 32 functions can be registered with the onexit function.

The functions have no parameters and do not return values.

NOTE: The onexit function is not an ANSI function. The ANSI standard function atexit does the same thing that onexit does and should be used instead of onexit where ANSI portability is concerned.

Returns:  The onexit function returns func if the registration succeeds, NULL if it fails.

See Also:  abort, atexit, exit, _exit

Example:  
#include <stdio.h>
#include <stdlib.h>

void main()
{
 extern void func1(void), func2(void), func3(void);

 onexit( func1 );
 onexit( func2 );
 onexit( func3 );
 printf( "Do this first.\n" );
}

void func1(void) { printf( "last.\n" ); } 
void func2(void) { printf( "this " ); } 
void func3(void) { printf( "Do " ); }

produces the following:

Do this first.
Do this last.

Classification:  WATCOM

Systems:  All, Netware
Synopsis:  
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open( const char *path, int access, ... );
int _open( const char *path, int access, ... );
int _wopen( const wchar_t *path, int access, ... );

Description:  
The open function opens a file at the operating system level. The name of the file to be opened is given by path. The file will be accessed according to the access mode specified by access. The optional argument is the file permissions to be used when the O_CREAT flag is on in the access mode.

The _open function is identical to open. Use _open for ANSI/ISO naming conventions.

The _wopen function is identical to open except that it accepts a wide character string argument for path.

The access mode is established by a combination of the bits defined in the <fcntl.h> header file. The following bits may be set:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>O_RDONLY</td>
<td>permit the file to be only read.</td>
</tr>
<tr>
<td>O_WRONLY</td>
<td>permit the file to be only written.</td>
</tr>
<tr>
<td>O_RDWR</td>
<td>permit the file to be both read and written.</td>
</tr>
<tr>
<td>O_APPEND</td>
<td>causes each record that is written to be written at the end of the file.</td>
</tr>
<tr>
<td>O_CREAT</td>
<td>has no effect when the file indicated by filename already exists; otherwise, the file is created;</td>
</tr>
<tr>
<td>O_TRUNC</td>
<td>causes the file to be truncated to contain no data when the file exists; has no effect when the file does not exist.</td>
</tr>
<tr>
<td>O_BINARY</td>
<td>causes the file to be opened in binary mode which means that data will be transmitted to and from the file unchanged.</td>
</tr>
<tr>
<td>O_TEXT</td>
<td>causes the file to be opened in text mode which means that carriage-return characters are written before any linefeed character that is written and causes carriage-return characters to be removed when encountered during reads.</td>
</tr>
<tr>
<td>O_NOINHERIT</td>
<td>indicates that this file is not to be inherited by a child process.</td>
</tr>
<tr>
<td>O_EXCL</td>
<td>indicates that this file is to be opened for exclusive access. If the file exists and O_CREAT was also specified then the open will fail (i.e., use O_EXCL to ensure that the file does not already exist).</td>
</tr>
</tbody>
</table>

When neither O_TEXT nor O_BINARY are specified, the default value in the global variable _fmode is used to set the file translation mode. When the program begins execution, this variable has a value of O_TEXT.

O_CREAT must be specified when the file does not exist and it is to be written.
When the file is to be created (O_CREAT is specified), an additional argument must be passed which contains the file permissions to be used for the new file. The access permissions for the file or directory are specified as a combination of bits (defined in the `<sys/stat.h>` header file).

The following bits define permissions for the owner.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_IRWXU</td>
<td>Read, write, execute/search</td>
</tr>
<tr>
<td>S_IRUSR</td>
<td>Read permission</td>
</tr>
<tr>
<td>S_IWUSR</td>
<td>Write permission</td>
</tr>
<tr>
<td>S_IXUSR</td>
<td>Execute/search permission</td>
</tr>
</tbody>
</table>

The following bits define permissions for the group.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_IRWXG</td>
<td>Read, write, execute/search</td>
</tr>
<tr>
<td>S_IRGRP</td>
<td>Read permission</td>
</tr>
<tr>
<td>S_IWGRP</td>
<td>Write permission</td>
</tr>
<tr>
<td>S_IXGRP</td>
<td>Execute/search permission</td>
</tr>
</tbody>
</table>

The following bits define permissions for others.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_IRWXO</td>
<td>Read, write, execute/search</td>
</tr>
<tr>
<td>S_IROTH</td>
<td>Read permission</td>
</tr>
<tr>
<td>S_IWOTH</td>
<td>Write permission</td>
</tr>
<tr>
<td>S_IXOTH</td>
<td>Execute/search permission</td>
</tr>
</tbody>
</table>

The following bits define miscellaneous permissions used by other implementations.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_IREAD</td>
<td>is equivalent to S_IRUSR (read permission)</td>
</tr>
<tr>
<td>S_IWRITE</td>
<td>is equivalent to S_IWUSR (write permission)</td>
</tr>
<tr>
<td>S_IEXEC</td>
<td>is equivalent to S_IXUSR (execute/search permission)</td>
</tr>
</tbody>
</table>

All files are readable with DOS; however, it is a good idea to set S_IREAD when read permission is intended for the file.

The `open` function applies the current file permission mask to the specified permissions (see `umask`).

**Returns:** If successful, `open` returns a handle for the file. When an error occurs while opening the file, -1 is returned.

**Errors:** When an error has occurred, `errno` contains a value indicating the type of error that has been detected.
open, _open, _wopen

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EACCES</td>
<td>Access denied because path specifies a directory or a volume ID, or attempting to open a read-only file for writing</td>
</tr>
<tr>
<td>EMFILE</td>
<td>No more handles available (too many open files)</td>
</tr>
<tr>
<td>ENOENT</td>
<td>Path or file not found</td>
</tr>
</tbody>
</table>

See Also: chsize, close, creat, dup, dup2, eof, exec..., fdopen, filelength, fileno, fstat, _grow_handles, isatty, lseek, read, setmode, sopen, stat, tell, write, umask

Example:
```c
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>

void main()
{
  int handle;

  /* open a file for output                  */
  /* replace existing file if it exists     */
  handle = open( "file",
                 O_WRONLY | O_CREAT | O_TRUNC,
                 S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );

  /* read a file which is assumed to exist  */
  handle = open( "file", O_RDONLY );

  /* append to the end of an existing file  */
  /* write a new file if file does not exist*/
  handle = open( "file",
                 O_WRONLY | O_CREAT | O_APPEND,
                 S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );
}
```

Classification: open is POSIX 1003.1
_open is not POSIX
_wopen is not POSIX
_open conforms to ANSI/ISO naming conventions

Systems: open - All, Netware
_open - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_wopen - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

704 Library Functions and Macros
Synopsis:  

```c
#include <direct.h>
struct dirent *opendir( const char *dirname );
struct _wdirent *_wopendir( const wchar_t *dirname );
```

Description:  

The opendir function is used in conjunction with the functions readdir and closedir to obtain the list of file names contained in the directory specified by dirname. The path indicated by dirname can be either relative to the current working directory or it can be an absolute path name. As an extension to POSIX, the last part of dirname can contain the characters ‘?’ and ‘*’ for matching multiple files within a directory.

The file `<direct.h>` contains definitions for the structure dirent.

```c
#if defined(__OS2__) || defined(__NT__)
#define NAME_MAX 255 /* maximum for HPFS or NTFS */
#else
#define NAME_MAX  12 /* 8 chars + '.' + 3 chars */
#endif

typedef struct dirent {
    char    d_dta[ 21 ];        /* disk transfer area */
    char    d_attr;             /* file's attribute */
    unsigned short int d_time; /* file's time */
    unsigned short int d_date; /* file's date */
    long    d_size;             /* file's size */
    char    d_name[ NAME_MAX + 1 ]; /* file's name */
    unsigned short d_ino;       /* serial number */
    char    d_first;            /* flag for 1st time */
} DIR;
```

The file attribute field d_attr field is a set of bits representing the following attributes.

- `_A_RDONLY` /* Read-only file */
- `_A_HIDDEN` /* Hidden file */
- `_A_SYSTEM` /* System file */
- `_A_VOLID` /* Volume-ID entry (only MSFT knows) */
- `_A_SUBDIR` /* Subdirectory */
- `_A_ARCH` /* Archive file */

If the _A_RDONLY bit is off, then the file is read/write.

The format of the d_time field is described by the following structure (this structure is not defined in any Watcom header file).

```c
typedef struct {
    unsigned short twosecs : 5;  /* seconds / 2 */
    unsigned short minutes : 6;  /* minutes (0,59) */
    unsigned short hours : 5;    /* hours (0,23) */
} ftime_t;
```

The format of the d_date field is described by the following structure (this structure is not defined in any Watcom header file).
typedef struct {
    unsigned short day : 5; /* day (1,31) */
    unsigned short month : 4; /* month (1,12) */
    unsigned short year : 7; /* 0 is 1980 */
} fdate_t;

See the sample program below for an example of the use of these structures.

More than one directory can be read at the same time using the opendir, readdir, and closedir functions.

The _wopendir function is identical to opendir except that it accepts a wide-character string argument and returns a pointer to a _wdirent structure that can be used with the _wreaddir and _wclosedir functions.

The file <dirent.h> contains definitions for the structure _wdirent.

struct _wdirent {
    char        d_dta[21];      /* disk transfer area */
    char        d_attr;         /* file’s attribute */
    unsigned short int d_time; /* file’s time */
    unsigned short int d_date; /* file’s date */
    long         d_size;         /* file’s size */
    wchar_t      d_name[NAME_MAX+1]; /* file’s name */
    unsigned short d_ino;       /* serial number (not used) */
    char         d_first;        /* flag for 1st time */
};

Returns: The opendir function, if successful, returns a pointer to a structure required for subsequent calls to readdir to retrieve the file names matching the pattern specified by dirname. The opendir function returns NULL if dirname is not a valid pathname, or if there are no files matching dirname.

Errors: When an error has occurred, errno contains a value indicating the type of error that has been detected.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EACCESS</td>
<td>Search permission is denied for a component of dirname or read permission is denied for dirname.</td>
</tr>
<tr>
<td>ENOENT</td>
<td>The named directory does not exist.</td>
</tr>
</tbody>
</table>

See Also: closedir, _dos_find..., readdir, reindir

Example: To get a list of files contained in the directory \watcom\h on your default disk:
#include <stdio.h>
#include <direct.h>

typedef struct {
  unsigned short  twosecs : 5;    /* seconds / 2 */
  unsigned short  minutes : 6;
  unsigned short  hours   : 5;
} ftime_t;

typedef struct {
  unsigned short  day     : 5;
  unsigned short  month   : 4;
  unsigned short  year    : 7;
} fdate_t;

void main()
{
  DIR *dirp;
  struct dirent *direntp;
  ftime_t  *f_time;
  fdate_t  *f_date;

  dirp = opendir( "\\watcom\\h" );
  if( dirp != NULL ) { 
    for(;;) {
      direntp = readdir( dirp );
      if( direntp == NULL ) break;
      f_time = (ftime_t *)&direntp->d_time;
      f_date = (fdate_t *)&direntp->d_date;
      printf( "%-12s %d/%2.2d/%2.2d 
       %2.2d:%2.2d:%2.2d 
", 
        direntp->d_name,
        f_date->year + 1980,
        f_date->month,
        f_date->day,
        f_time->hours,
        f_time->minutes,
        f_time->twosecs * 2 );
    }
  }
  closedir( dirp );
}

Note the use of two adjacent backslash characters (\) within character-string constants to signify a single backslash.

**Classification:**
- opendir is POSIX 1003.1
- _wopendir is not POSIX

**Systems:**
- opendir - All, Netware
- _wopendir - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  

#include <io.h>  
int _open_osfhandle( long osfhandle, int access );

Description:  The _open_osfhandle function allocates a POSIX-level file handle and sets it to point to the operating system’s internal file handle specified by osfhandle. The value returned by _get_osfhandle can be used as an argument to the _open_osfhandle function.

The access mode is established by a combination of the bits defined in the <fcntl.h> header file. The following bits may be set:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>O_RDONLY</td>
<td>permit the file to be only read.</td>
</tr>
<tr>
<td>O_WRONLY</td>
<td>permit the file to be only written.</td>
</tr>
<tr>
<td>O_RDWR</td>
<td>permit the file to be both read and written.</td>
</tr>
<tr>
<td>O_APPEND</td>
<td>causes each record that is written to be written at the end of the file.</td>
</tr>
<tr>
<td>O_CREAT</td>
<td>has no effect when the file indicated by filename already exists; otherwise, the file is created;</td>
</tr>
<tr>
<td>O_TRUNC</td>
<td>causes the file to be truncated to contain no data when the file exists; has no effect when the file does not exist.</td>
</tr>
<tr>
<td>O_BINARY</td>
<td>causes the file to be opened in binary mode which means that data will be transmitted to and from the file unchanged.</td>
</tr>
<tr>
<td>O_TEXT</td>
<td>causes the file to be opened in text mode which means that carriage-return characters are written before any linefeed character that is written and causes carriage-return characters to be removed when encountered during reads.</td>
</tr>
<tr>
<td>O_NOINHERIT</td>
<td>indicates that this file is not to be inherited by a child process.</td>
</tr>
<tr>
<td>O_EXCL</td>
<td>indicates that this file is to be opened for exclusive access. If the file exists and O_CREAT was also specified then the open will fail (i.e., use O_EXCL to ensure that the file does not already exist).</td>
</tr>
</tbody>
</table>

When neither O_TEXT nor O_BINARY are specified, the default value in the global variable _fmode is used to set the file translation mode. When the program begins execution, this variable has a value of O_TEXT.

O_CREAT must be specified when the file does not exist and it is to be written.

When two or more manifest constants are used to form the flags argument, the constants are combined with the bitwise-OR operator (|).

The example below demonstrates the use of the _get_osfhandle and _open_osfhandle functions. Note that the example shows how the dup2 function can be used to obtain almost identical functionality.
When the POSIX-level file handles associated with one OS file handle are closed, the first one closes successfully but the others return an error (since the first call close the file and released the OS file handle). So it is important to call \texttt{close} at the right time, i.e., after all I/O operations are completed to the file.

\textbf{Returns:} If successful, \texttt{open\_osfhandle} returns a POSIX-style file handle. Otherwise, it returns -1.

\textbf{See Also:} \texttt{close, \_dos\_open, dup2, fdopen, fopen, freopen, \_fsopen, \_get\_osfhandle, \_grow\_handles, \_hdopen, open, \_os\_handle, \_popen, sopen}

\textbf{Example:}

```c
#include <stdio.h>
#include <stdlib.h>
#include <io.h>
#include <fcntl.h>

void main()
{
    long os_handle;
    int fh1, fh2, rc;

    fh1 = open( "file",
                   O_WRONLY | O_CREAT | O_TRUNC | O_BINARY,
                   S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );
    if( fh1 == -1 ) { 
      printf( "Could not open output file\n" );
      exit( EXIT_FAILURE );
    }
    printf( "First POSIX handle %d\n", fh1 );

    #if defined(USE_DUP2)
    fh2 = 6;
    if( dup2( fh1, fh2 ) == -1 ) fh2 = -1;
    #else
    os_handle = \_get\_osfhandle( fh1 );
    printf( "OS Handle %ld\n", os_handle );
    
    fh2 = \_open\_osfhandle( os_handle, O_WRONLY |
                          O_BINARY );
    #endif
    if( fh2 == -1 ) { 
      printf( "Could not open with second handle\n" );
      exit( EXIT_FAILURE );
    }
    printf( "Second POSIX handle %d\n", fh2 );

    rc = write( fh2, "trash\x0d\x0a", 7 );
    printf( "Write file using second handle %d\n", rc );

    rc = close( fh2 );
    printf( "Closing second handle %d\n", rc );
    rc = close( fh1 );
    printf( "Closing first handle %d\n", rc );
}
```

\textbf{Classification:} WATCOM
_open_osfhandle

Systems: All, Netware
Synopsis:  
```c
#include <io.h>
int _os_handle( int handle );
```

Description:  The _os_handle function takes a POSIX-style file handle specified by handle. It returns the corresponding operating system level handle.

Returns:  
The _os_handle function returns the operating system handle that corresponds to the specified POSIX-style file handle.

See Also:  
close, fdopen, _get_osfhandle, _hdopen, open, _open_osfhandle

Example:  
```c
#include <stdio.h>
#include <io.h>

void main()
{
    int handle;
    FILE *fp;

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        handle = _os_handle( fileno( fp ) );
        fclose( fp );
    }
}
```

Classification:  WATCOM

Systems:  
DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, Netware
Synopsis:  
#include <graph.h>
void _FAR _outgtext( char _FAR *text );

Description:  The _outgtext function displays the character string indicated by the argument text. The string must be terminated by a null character ('\0').

The string is displayed starting at the current position (see the _moveto function) in the current color and in the currently selected font (see the _setfont function). The current position is updated to follow the displayed text.

When no font has been previously selected with _setfont, a default font will be used. The default font is an 8-by-8 bit-mapped font.

The graphics library can display text in three different ways.

1. The _outtext and _outmem functions can be used in any video mode. However, this variety of text can be displayed in only one size.

2. The _grtext function displays text as a sequence of line segments, and can be drawn in different sizes, with different orientations and alignments.

3. The _outgtext function displays text in the currently selected font. Both bit-mapped and vector fonts are supported; the size and type of text depends on the fonts that are available.

Returns:  The _outgtext function does not return a value.

See Also:  _registerfonts, _unregisterfonts, _setfont, _getfontinfo, _getgtextextent, _setgtextvector, _getgtextvector, _outtext, _outmem, _grtext

Example:  #include <conio.h>
#include <stdio.h>
#include <graph.h>

main()
{
    int i, n;
    char buf[ 10 ];

    _setvideomode( _VRES16COLOR );
n = _registerfonts( "*.fon" );
for( i = 0; i < n; ++i ) {
    sprintf( buf, "n%d", i );
    _setfont( buf );
    _moveto( 100, 100 );
    _outgtext( "WATCOM Graphics" );
    getch();
    _clearscreen( _GCLEARSCREEN );
}

    _unregisterfonts();
    _setvideomode( _DEFAULTMODE );
}

Classification:  _outgtext is PC Graphics

712 Library Functions and Macros
Systems: DOS, QNX
Synopsis:  
#include <graph.h>  
void _FAR _outmem( char _FAR *text, short length );

Description:  The _outmem function displays the character string indicated by the argument text. The argument length specifies the number of characters to be displayed. Unlike the _outtext function, _outmem will display the graphical representation of characters such as ASCII 10 and 0, instead of interpreting them as control characters.

The text is displayed using the current text color (see the _settextcolor function), starting at the current text position (see the _settextposition function). The text position is updated to follow the end of the displayed text.

The graphics library can display text in three different ways.

1. The _outtext and _outmem functions can be used in any video mode. However, this variety of text can be displayed in only one size.

2. The _grtext function displays text as a sequence of line segments, and can be drawn in different sizes, with different orientations and alignments.

3. The _outgtext function displays text in the currently selected font. Both bit-mapped and vector fonts are supported; the size and type of text depends on the fonts that are available.

Returns:  The _outmem function does not return a value.

See Also:  _settextcolor, _settextposition, _settextwindow, _grtext, _outtext, _outgtext

Example:  
#include <conio.h>
#include <graph.h>

main()
{
 int i;
 char buf[ 1 ];

 _clearscreen( _GCLEARSCREEN );
 for( i = 0; i <= 255; ++i ) {
   _settextposition( 1 + i % 16,
                     1 + 5 * ( i / 16 ) );
   buf[ 0 ] = i;
   _outmem( buf, 1 );
 }
 getch();
}

Classification:  PC Graphics

Systems:  DOS, QNX

714  Library Functions and Macros
Synopsis:  
#include <conio.h>
unsigned int outp( int port, int value );

Description:  The outp function writes one byte, determined by value, to the 80x86 hardware port whose number is given by port.

A hardware port is used to communicate with a device. One or two bytes can be read and/or written from each port, depending upon the hardware. Consult the technical documentation for your computer to determine the port numbers for a device and the expected usage of each port for a device.

Returns:  The value transmitted is returned.

See Also:  inp, inpd, inpw, outpd, outpw

Example:  
#include <conio.h>

void main()
{
    /* turn off speaker */
    outp( 0x61, inp( 0x61 ) & 0xFC );
}

Classification:  Intel

Systems:  All, Netware
Synopsis:     #include <conio.h>
              unsigned long outpd( int port,
                                      unsigned long value );

Description: The outpd function writes a double-word (four bytes), determined by value, to the 80x86 hardware port whose number is given by port.

    A hardware port is used to communicate with a device. One or two bytes can be read and/or written from each port, depending upon the hardware. Consult the technical documentation for your computer to determine the port numbers for a device and the expected usage of each port for a device.

Returns:     The value transmitted is returned.

See Also:    inp, inpd, inpw, outp, outpw

Example:     #include <conio.h>
              #define DEVICE 34
              
              void main()
              {
                  outpd( DEVICE, 0x12345678 );
              }

Classification: Intel

Synopsis:    
#include <conio.h>
unsigned int outpw( int port,
                  unsigned int value );

Description: The outpw function writes a word (two bytes), determined by value, to the 80x86 hardware port whose number is given by port.

A hardware port is used to communicate with a device. One or two bytes can be read and/or written from each port, depending upon the hardware. Consult the technical documentation for your computer to determine the port numbers for a device and the expected usage of each port for a device.

Returns:    The value transmitted is returned.

See Also:   inp, inpd, inpw, outp, outpd

Example:    #include <conio.h>
#define DEVICE 34

t void main()
{
    outpw( DEVICE, 0x1234 );
}

Classification: Intel

Systems:   All, Netware
Synopsis:  
#include <graph.h>
void _FAR _outtext( char _FAR *text );

Description:  The _outtext function displays the character string indicated by the argument text. The string must be terminated by a null character ('\0'). When a line-feed character ('\n') is encountered in the string, the characters following will be displayed on the next row of the screen.

The text is displayed using the current text color (see the _settextcolor function), starting at the current text position (see the _settextposition function). The text position is updated to follow the end of the displayed text.

The graphics library can display text in three different ways.

1. The _outtext and _outmem functions can be used in any video mode. However, this variety of text can be displayed in only one size.

2. The _grtext function displays text as a sequence of line segments, and can be drawn in different sizes, with different orientations and alignments.

3. The _outgtext function displays text in the currently selected font. Both bit-mapped and vector fonts are supported; the size and type of text depends on the fonts that are available.

Returns:  The _outtext function does not return a value.

See Also: _settextcolor, _settextposition, _settextwindow, _grtext, _outmem, _outgtext

Example:  
#include <conio.h>
#include <graph.h>

main()
{
   _setvideomode( _TEXTC80 );
   _settextposition( 10, 30 );
   _outtext( "WATCOM Graphics" );
   getch();
   _setvideomode( _DEFAULTMODE );
}

Classification:  PC Graphics

Systems:  DOS, QNX
Synopsis:  
#include <stdio.h>  
int _pclose( FILE *fp );

Description:  The _pclose function closes the pipe associated with fp and waits for the subprocess created by _popen to terminate.

Returns:  The _pclose function returns the termination status of the command language interpreter. If an error occurred, _pclose returns (-1) with errno set appropriately.

Errors:  When an error has occurred, errno contains a value indicating the type of error that has been detected.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EINTR</td>
<td>The _pclose function was interrupted by a signal while waiting for the child process to terminate.</td>
</tr>
<tr>
<td>ECHILD</td>
<td>The _pclose function was unable to obtain the termination status of the child process.</td>
</tr>
</tbody>
</table>

See Also:  perror, _pipe, _popen

Example:  See example provided with _popen.

Classification:  WATCOM

Systems:  Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <stdio.h>
void perror( const char *prefix );
void _wperror( const wchar_t *prefix );

Description:  The perror function prints, on the file designated by stderr, the error message corresponding to the error number contained in errno. The perror function writes first the string pointed to by prefix to stderr. This is followed by a colon (":") , a space, the string returned by strerror(errno), and a newline character.

The _wperror function is identical to perror except that it accepts a wide-character string argument and produces wide-character output.

Returns: The perror function returns no value. Because perror uses the fprintf function, errno can be set when an error is detected during the execution of that function.

See Also: clearerr, feof, ferror, strerror

Example:  
#include <stdio.h>

void main()
{
    FILE *fp;

    fp = fopen( "data.fil", "r" );
    if( fp == NULL ) {
        perror( "Unable to open file" );
    }
}

Classification: perror is ANSI
                _wperror is not ANSI

Systems: perror - All, Netware
                _wperror - All
Synopsis:  
#include <pgchart.h>
short _FAR _pg_analyzechart( chartenv _FAR *env,
   char _FAR * _FAR *cat,
   float _FAR *values, short n );

short _FAR _pg_analyzechartms( chartenv _FAR *env,
   char _FAR * _FAR *cat,
   float _FAR *values,
   short nseries,
   short n, short dim,
   char _FAR * _FAR *labels );

Description:  The _pg_analyzechart functions analyze either a single-series or a multi-series bar, column or line chart. These functions calculate default values for chart elements without actually displaying the chart.

The _pg_analyzechart function analyzes a single-series bar, column or line chart. The chart environment structure env is filled with default values based on the type of chart and the values of the cat and values arguments. The arguments are the same as for the _pg_chart function.

The _pg_analyzechartms function analyzes a multi-series bar, column or line chart. The chart environment structure env is filled with default values based on the type of chart and the values of the cat, values and labels arguments. The arguments are the same as for the _pg_chartms function.

Returns:  The _pg_analyzechart functions return zero if successful; otherwise, a non-zero value is returned.

See Also:  _pg_defaultchart,_pg_initchart,_pg_chart,_pg_chartpie,_pg_chartscatter,_pg_analyzepie,_pg_analyzescatter
Example:
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#if defined ( __386__ )
#define _FAR
#else
#define _FAR __far
#endif

#define NUM_VALUES 4

char _FAR *categories[ NUM_VALUES ] = {
   "Jan", "Feb", "Mar", "Apr"
};

float values[ NUM_VALUES ] = {
   20, 45, 30, 25
};

main()
{
    chartenv env;

    _setvideomode( _VRES16COLOR );
    _pg_initchart();
    _pg_defaultchart( &env,
        _PG_COLUMNCHART, _PG_PLAINBARS );
    strcpy( env.maintitle.title, "Column Chart" );
    _pg_analyzechart( &env,
         categories, values, NUM_VALUES );
    /* use manual scaling */
    env.yaxis.autoscale = 0;
    env.yaxis.scalemin = 0.0;
    env.yaxis.scalemax = 100.0;
    env.yaxis.ticinterval = 25.0;
    _pg_chart( &env, categories, values, NUM_VALUES );
    getch();
    _setvideomode( _DEFAULTMODE );
}

Classification: _pg_analyzechart is PC Graphics

Systems: _pg_analyzechart - DOS, QNX
          _pg_analyzechartms - DOS, QNX
Synopsis:  
#include <pgchart.h>
short _FAR _pg_analyzepie( chartenv _FAR *env,
                        char _FAR * _FAR *cat,
                        float _FAR *values,
                        short _FAR *explode, short n );

Description:  The _pg_analyzepie function analyzes a pie chart. This function calculates default values for chart elements without actually displaying the chart.

The chart environment structure env is filled with default values based on the values of the cat, values and explode arguments. The arguments are the same as for the _pg_chartpie function.

Returns:  The _pg_analyzepie function returns zero if successful; otherwise, a non-zero value is returned.

See Also:  _pg_defaultchart, _pg_initchart, _pg_chart, _pg_chartpie, _pg_chartscatter, _pg_analyzechart, _pg_analyzescatter
Example:

```c
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#if defined ( __386__ )
#define _FAR
#else
#define _FAR __far
#endif

#define NUM_VALUES 4

char _FAR *categories[ NUM_VALUES ] = {
    "Jan", "Feb", "Mar", "Apr"
};

float values[ NUM_VALUES ] = {
    20, 45, 30, 25
};

short explode[ NUM_VALUES ] = {
    1, 0, 0, 0
};

main()
{
    chartenv env;

    _setvideomode( _VRES16COLOR );
    _pg_initchart();
    _pg_defaultchart( &env,
        _PG_PIECHART, _PG_NOPERCENT );
    strcpy( env.maintitle.title, "Pie Chart" );
    env.legend.place = _PG_BOTTOM;
    _pg_analyzepie( &env, categories,
        values, explode, NUM_VALUES );
    /* make legend window same width as data window */
    env.legend.autosize = 0;
    env.legend.legendwindow.x1 = env.datawindow.x1;
    env.legend.legendwindow.x2 = env.datawindow.x2;
    _pg_chartpie( &env, categories,
        values, explode, NUM_VALUES );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX
Synopsis:

```
#include <pgchart.h>
short _FAR _pg_analyzescatter( chartenv _FAR *env,
       float _FAR *x,
       float _FAR *y, short n );
```

```
short _FAR _pg_analyzescatterms(
       chartenv _FAR *env,
       float _FAR *x, float _FAR *y,
       short nseries, short n, short dim,
       char _FAR * _FAR *labels );
```

Description: The _pg_analyzescatter functions analyze either a single-series or a multi-series scatter chart. These functions calculate default values for chart elements without actually displaying the chart.

The _pg_analyzescatter function analyzes a single-series scatter chart. The chart environment structure env is filled with default values based on the values of the x and y arguments. The arguments are the same as for the _pg_chartscatter function.

The _pg_analyzescatterterms function analyzes a multi-series scatter chart. The chart environment structure env is filled with default values based on the values of the x, y and labels arguments. The arguments are the same as for the _pg_chartscatterterms function.

Returns: The _pg_analyzescatter functions return zero if successful; otherwise, a non-zero value is returned.

See Also: _pg_defaultchart, _pg_initchart, _pg_chart, _pg_chartpie,
           _pg_chartscatter, _pg_analyzechart, _pg_analyzepie
Example:

```c
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#if defined ( __386__ )
#define _FAR
#else
#define _FAR    __far
#endif

#define NUM_VALUES 4
#define NUM_SERIES 2

char _FAR *labels[ NUM_SERIES ] = {
    "Jan", "Feb"
};

float x[ NUM_SERIES ][ NUM_VALUES ] = {
    5, 15, 30, 40, 10, 20, 30, 45
};

float y[ NUM_SERIES ][ NUM_VALUES ] = {
    10, 15, 30, 45, 40, 30, 15, 5
};

main()
{
    chartenv env;

    _setvideomode( _VRES16COLOR );
    _pg_initchart();
    _pg_defaultchart( &env, _PG_SCATTERCHART, _PG_POINTANDLINE );
    strcpy( env.maintitle.title, "Scatter Chart" );
    _pg_analyzescatterms( &env, x, y, NUM_SERIES, NUM_VALUES, NUM_VALUES, labels );
    /* display x-axis labels with 2 decimal places */
    env.xaxis.autoscale = 0;
    env.xaxis.ticdecimals = 2;
    _pg_chartscatterms( &env, x, y, NUM_SERIES, NUM_VALUES, NUM_VALUES, labels );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: _pg_analyzescatter - DOS, QNX
         _pg_analyzescatterms - DOS, QNX
Synopsis:    #include <pgchart.h>
short _FAR _pg_chart( chartenv _FAR *env,
                   char _FAR * _FAR *cat,
                   float _FAR *values, short n );

short _FAR _pg_chartms( chartenv _FAR *env,
                   char _FAR * _FAR *cat,
                   float _FAR *values, short nseries,
                   short n, short dim,
                   char _FAR * _FAR *labels );

Description:  The _pg_chart functions display either a single-series or a multi-series bar, column or line chart. The type of chart displayed and other chart options are contained in the env argument. The argument cat is an array of strings. These strings describe the categories against which the data in the values array is charted.

The _pg_chart function displays a bar, column or line chart from the single series of data contained in the values array. The argument n specifies the number of values to chart.

The _pg_chartms function displays a multi-series bar, column or line chart. The argument nseries specifies the number of series of data to chart. The argument values is assumed to be a two-dimensional array defined as follows:

    float values[ nseries ][ dim ];

The number of values used from each series is given by the argument n, where n is less than or equal to dim. The argument labels is an array of strings. These strings describe each of the series and are used in the chart legend.

Returns:    The _pg_chart functions return zero if successful; otherwise, a non-zero value is returned.

See Also:   _pg_defaultchart,_pg_initchart,_pg_chartpie,_pg_chartscatter,
            _pg_analyzechart,_pg_analyzepie,_pg_analyzescatter
Example:

```c
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#if defined ( __386__ )
#define _FAR
#else
#define _FAR    __far
#endif

#define NUM_VALUES 4

char _FAR *categories[ NUM_VALUES ] = {
    "Jan", "Feb", "Mar", "Apr"
};

float values[ NUM_VALUES ] = {
    20, 45, 30, 25
};

main()
{
    chartenv env;

    _setvideomode( _VRES16COLOR );
    _pg_initchart();
    _pg_defaultchart( &env,
        _PG_COLUMNCHART, _PG_PLAINBARS );
    strcpy( env.maintitle.title, "Column Chart" );
    _pg_chart( &env, categories, values, NUM_VALUES );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

produces the following:
Classification: PC Graphics

Systems:  
_pg_chart - DOS, QNX  
_pg_chartms - DOS, QNX
Synopsis:

```c
#include <pgchart.h>
short _FAR _pg_chartpie( chartenv _FAR *env,
                        char _FAR * _FAR *cat,
                        float _FAR *values,
                        short _FAR *explode, short n );
```

Description:
The `_pg_chartpie` function displays a pie chart. The chart is displayed using the options specified in the `env` argument.

The pie chart is created from the data contained in the `values` array. The argument `n` specifies the number of values to chart.

The argument `cat` is an array of strings. These strings describe each of the pie slices and are used in the chart legend. The argument `explode` is an array of values corresponding to each of the pie slices. For each non-zero element in the array, the corresponding pie slice is drawn "exploded", or slightly offset from the rest of the pie.

Returns:
The `_pg_chartpie` function returns zero if successful; otherwise, a non-zero value is returned.

See Also:
`_pg_defaultchart, _pg_initchart, _pg_chart, _pg_chartscatter, _pg_analyzechart, _pg_analyzepie, _pg_analyzescatter`
Example:

```c
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#if defined ( __386__ )
#define _FAR
#else
#define _FAR    __far
#endif

#define NUM_VALUES 4

char _FAR *categories[NUM_VALUES] = {
    "Jan", "Feb", "Mar", "Apr"
};

float values[NUM_VALUES] = {
    20, 45, 30, 25
};

short explode[NUM_VALUES] = {
    1, 0, 0, 0
};

main()
{
    chartenv env;

    _setvideomode(_VRES16COLOR);
    _pg_initchart();
    _pg_defaultchart(&env,
                     _PG_PIECHART, _PG_NOPERCENT);
    strcpy(env.maintitle.title, "Pie Chart");
    _pg_chartpie(&env, categories,
                 values, explode, NUM_VALUES);
    getch();
    _setvideomode(_DEFAULTMODE);
}
```

produces the following:
Classification: PC Graphics

Systems: DOS, QNX
Synopsis:  
```c
#include <pgchart.h>
short _FAR _pg_chartscatter( chartenv _FAR *env,
                          float _FAR *x,
                          float _FAR *y, short n );

short _FAR _pg_chartscatterms( chartenv _FAR *env,
                               float _FAR *x,
                               float _FAR *y,
                               short nseries,
                               short n, short dim,
                               char _FAR * _FAR *labels );
```

Description:  The _pg_chartscatter functions display either a single-series or a multi-series scatter chart. The chart is displayed using the options specified in the `env` argument.

The _pg_chartscatter function displays a scatter chart from the single series of data contained in the arrays `x` and `y`. The argument `n` specifies the number of values to chart.

The _pg_chartscatterms function displays a multi-series scatter chart. The argument `nseries` specifies the number of series of data to chart. The arguments `x` and `y` are assumed to be two-dimensional arrays defined as follows:

```c
float x[ nseries ][ dim ];
```

The number of values used from each series is given by the argument `n`, where `n` is less than or equal to `dim`. The argument `labels` is an array of strings. These strings describe each of the series and are used in the chart legend.

Returns:  The _pg_chartscatter functions return zero if successful; otherwise, a non-zero value is returned.

See Also:  _pg_defaultchart, _pg_initchart, _pg_chart, _pg_chartpie, _pg_analyzechart, _pg_analyzepie, _pg_analyzescatter
Example:

```c
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#if defined ( __386__ )
#define _FAR
#else
#define _FAR    __far
#endif

#define NUM_VALUES 4
#define NUM_SERIES 2

char _FAR *labels[NUM_SERIES] = {
    "Jan", "Feb"
};

float x[NUM_SERIES][NUM_VALUES] = {
    5, 15, 30, 40, 10, 20, 30, 45
};

float y[NUM_SERIES][NUM_VALUES] = {
    10, 15, 30, 45, 40, 30, 15, 5
};

main()
{
    chartenv env;

    _setvideomode(_VRES16COLOR);
    _pg_initchart();
    _pg_defaultchart( &env, _PG_SCATTERCHART, _PG_POINTANDLINE );
    strcpy( env.maintitle.title, "Scatter Chart" );
    _pg_chartscatterms( &env, x, y, NUM_SERIES, NUM_VALUES, labels );
    getch();
    _setvideomode(_DEFAULTMODE);
}
```

produces the following:
Classification: PC Graphics

Systems:  
- `_pg_chartscatter` - DOS, QNX  
- `_pg_chartscatterms` - DOS, QNX
Synopsis:  

```c
#include <pgchart.h>
short _FAR _pg_defaultchart( chartenv _FAR *env,
    short type, short style );
```

Description:  The `_pg_defaultchart` function initializes the chart structure `env` to contain default values before a chart is drawn. All values in the chart structure are initialized, including blanking of all titles. The chart type in the structure is initialized to the value `type`, and the chart style is initialized to `style`.

The argument `type` can have one of the following values:

- **_PG_BARCHART**  
  Bar chart (horizontal bars)

- **_PG_COLUMNCHART**  
  Column chart (vertical bars)

- **_PG_LINECHART**  
  Line chart

- **_PG_SCATTERCHART**  
  Scatter chart

- **_PG_PIECHART**  
  Pie chart

Each type of chart can be drawn in one of two styles. For each chart type the argument `style` can have one of the following values:

<table>
<thead>
<tr>
<th>Type</th>
<th>Style 1</th>
<th>Style 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bar</td>
<td>_PG_PLAINBARS</td>
<td>_PG_STACKEDBARS</td>
</tr>
<tr>
<td>Column</td>
<td>_PG_PLAINBARS</td>
<td>_PG_STACKEDBARS</td>
</tr>
<tr>
<td>Line</td>
<td>_PG_POINTANDLINE</td>
<td>_PG_POINTONLY</td>
</tr>
<tr>
<td>Scatter</td>
<td>_PG_POINTANDLINE</td>
<td>_PG_POINTONLY</td>
</tr>
<tr>
<td>Pie</td>
<td>_PG_PERCENT</td>
<td>_PG_NOPERCENT</td>
</tr>
</tbody>
</table>

For single-series bar and column charts, the chart style is ignored. The "plain" (clustered) and "stacked" styles only apply when there is more than one series of data. The "percent" style for pie charts causes percentages to be displayed beside each of the pie slices.

Returns:  The `_pg_defaultchart` function returns zero if successful; otherwise, a non-zero value is returned.

See Also:  `_pg_initchart`, `_pg_chart`, `_pg_chartpie`, `_pg_chartscatter`
Example:

```c
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#if defined ( __386__ )
#define _FAR
#else
#define _FAR    __far
#endif

#define NUM_VALUES 4

char _FAR *categories[ NUM_VALUES ] = {
    "Jan", "Feb", "Mar", "Apr"
};

float values[ NUM_VALUES ] = {
    20, 45, 30, 25
};

main()
{
    chartenv env;
    _setvideomode( _VRES16COLOR );
    _pg_initchart();
    _pg_defaultchart( &env,
        _PG_COLUMNCHART, _PG_PLAINBARS );
    strcpy( env.maintitle.title, "Column Chart" );
    _pg_chart( &env, categories, values, NUM_VALUES );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX
Synopsis:
#include <pgchart.h>
short _FAR _pg_getchardef( short ch,
    unsigned char _FAR *def );

Description: The _pg_getchardef function retrieves the current bit-map definition for the character ch. The bit-map is placed in the array def. The current font must be an 8-by-8 bit-mapped font.

Returns: The _pg_getchardef function returns zero if successful; otherwise, a non-zero value is returned.

See Also: _pg_defaultchart, _pg_initchart, _pg_chart, _pg_chartpie,
          _pg_charscatter, _pg_setchardef

Example:
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#define NUM_VALUES 4

float x[ NUM_VALUES ] = {
    5, 25, 45, 65
};

float y[ NUM_VALUES ] = {
    5, 45, 25, 65
};

char diamond[ 8 ] = {
    0x10, 0x28, 0x44, 0x82, 0x44, 0x28, 0x10, 0x00
};

main()
{
    chartenv env;
    char old_def[ 8 ];

    _setvideomode( _VRES16COLOR );
    _pg_initchart();
    _pg_defaultchart( &env,
                      _PG_SCATTERCHART, _PG_POINTANDLINE );
    strcpy( env.maintitle.title, "Scatter Chart" );
    /* change asterisk character to diamond */
    _pg_getchardef( '*', old_def );
    _pg_setchardef( '*', diamond );
    _pg_charscatter( &env, x, y, NUM_VALUES );
    _pg_setchardef( '*', old_def );
    getch();
    _setvideomode( _DEFAULTMODE );
}

Classification: PC Graphics

Systems: DOS, QNX

738 Library Functions and Macros
Synopsis:  
#include <pgchart.h>
short _FAR _pg_getpalette ( paletteentry _FAR *pal );

Description:  
The _pg_getpalette function retrieves the internal palette of the presentation graphics system.
The palette controls the colors, line styles, fill patterns and plot characters used to display each series of
data in a chart.

The argument pal is an array of palette structures that will contain the palette. Each element of the
palette is a structure containing the following fields:

color  color used to display series
style  line style used for line and scatter charts
fill  fill pattern used to fill interior of bar and pie sections
plotchar  character plotted on line and scatter charts

Returns:  
The _pg_getpalette function returns zero if successful; otherwise, a non-zero value is returned.

See Also:  
/pg_defaultchart, /pg_initchart, /pg_chart, /pg_chartpie, 
/pg_chartscatter, /pg_setpalette, /pg_resetpalette
Example:
```c
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#if defined ( __386__ )
#define _FAR
#else
#define _FAR    __far
#endif

#define NUM_VALUES 4

char _FAR *categories[ NUM_VALUES ] = {
    "Jan", "Feb", "Mar", "Apr"
};

float values[ NUM_VALUES ] = {
    20, 45, 30, 25
};

char bricks[ 8 ] = {
    0xff, 0x80, 0x80, 0x80, 0xff, 0x08, 0x08, 0x08
};

main()
{
    chartenv env;
    palettetype pal;

    _setvideomode( _VRES16COLOR );
    _pg_initchart();
    _pg_defaultchart( &env,
                    _PG_COLUMNCHART, _PG_PLAINBARS );
    strcpy( env.maintitle.title, "Column Chart" );
    /* get default palette and change 1st entry */
    _pg_getpalette( &pal );
    pal[ 1 ].color = 12;
    memcpy( pal[ 1 ].fill, bricks, 8 );
    /* use new palette */
    _pg_setpalette( &pal );
    _pg_chart( &env, categories, values, NUM_VALUES );
    /* reset palette to default */
    _pg_resetpalette();
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: PC Graphics

Systems: DOS, QNX
Synopsis:  
#include <pgchart.h>  
void _FAR _pg_getstyleset( unsigned short _FAR *style );

Description:  
The _pg_getstyleset function retrieves the internal style-set of the presentation graphics system.  
The style-set is a set of line styles used for drawing window borders and grid-lines.  The argument style  
is an array that will contain the style-set.

Returns:  
The _pg_getstyleset function does not return a value.

See Also:  
.pg_defaultchart,.pg_initchart,.pg_chart,.pg_chartpie,  
.pg_chartscatter,.pg_setstyleset,.pg_resetstyleset

Example:
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#if defined ( __386__ )
#define _FAR
#else
#define _FAR __far
#endif

#define NUM_VALUES 4

char _FAR *categories[ NUM_VALUES ] = {
    "Jan", "Feb", "Mar", "Apr"
};

float values[ NUM_VALUES ] = {
    20, 45, 30, 25
};

main()
{
    chartenv env;
    styleset style;

    _setvideomode ( _VRES16COLOR );
    _pg_initchart();
    _pg_defaultchart( &env,
        _PG_COLUMNCHART, _PG_PLAINBARS );
    strcpy( env.maintitle.title, "Column Chart" );
    /* turn on yaxis grid, and use style 2 */
    env.yaxis.grid = 1;
    env.yaxis.gridstyle = 2;
    /* get default style-set and change entry 2 */
    _pg_getstyleset( &style );
    style[ 2 ] = 0x8888;
    /* use new style-set */
    _pg_setstyleset( &style );
    _pg_chart( &env, categories, values, NUM_VALUES );
    /* reset style-set to default */
    _pg_resetstyleset();
    getch();
    _setvideomode( _DEFAULTMODE );
}

Classification: PC Graphics

Systems: DOS, QNX
Synopsis:  

```c
#include <pgchart.h>
short _FAR _pg_hlabelchart( chartenv _FAR *env,
    short x, short y,
    short color,
    char _FAR *label );
```

Description:  The _pg_hlabelchart function displays the text string label on the chart described by the env chart structure. The string is displayed horizontally starting at the point \((x, y)\), relative to the upper left corner of the chart. The color specifies the palette color used to display the string.

Returns: The _pg_hlabelchart function returns zero if successful; otherwise, a non-zero value is returned.

See Also:  _pg_defaultchart, _pg_initchart, _pg_chart, _pg_chartpie, _pg_chartscatter, _pg_vlabelchart

Example:  

```c
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#if defined ( __386__ )
    #define _FAR
#else
    #define _FAR    __far
#endif

#define NUM_VALUES 4

char _FAR *categories[ NUM_VALUES ] = {
    "Jan", "Feb", "Mar", "Apr"
};

float values[ NUM_VALUES ] = {
    20, 45, 30, 25
};

main()
{
    chartenv env;

    _setvideomode( _VRES16COLOR );
    _pg_initchart();
    _pg_defaultchart( &env,
        _PG_COLUMNCHART, _PG_PLAINBARS );
    strcpy( env.maintitle.title, "Column Chart" );
    _pg_chart( &env, categories, values, NUM_VALUES );
    _pg_hlabelchart( &env, 64, 32, 1, "Horizontal label" );
    _pg_vlabelchart( &env, 48, 32, 1, "Vertical label" );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification:  PC Graphics

Systems:  DOS, QNX
Synopsis: #include <pgchart.h>
short _FAR _pg_initchart( void );

Description: The _pg_initchart function initializes the presentation graphics system. This includes initializing the internal palette and style-set used when drawing charts. This function must be called before any of the other presentation graphics functions.

The initialization of the presentation graphics system requires that a valid graphics mode has been selected. For this reason the _setvideomode function must be called before _pg_initchart is called. If a font has been selected (with the _setfont function), that font will be used when text is displayed in a chart. Font selection should also be done before initializing the presentation graphics system.

Returns: The _pg_initchart function returns zero if successful; otherwise, a non-zero value is returned.

See Also: _pg_defaultchart, _pg_chart, _pg_chartpie, _pg_chartscatter, _setvideomode, _setfont, _registerfonts

Example: #include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#if defined ( __386__ )
#define _FAR
#else
#define _FAR    __far
#endif

#define NUM_VALUES 4
char _FAR *categories[ NUM_VALUES ] = {
   "Jan", "Feb", "Mar", "Apr"
};

float values[ NUM_VALUES ] = {
   20, 45, 30, 25
};

main()
{
   chartenv env;

   _setvideomode( _VRES16COLOR );
   _pg_initchart();
   _pg_defaultchart( &env,
      _PG_COLUMNCHART, _PG_PLAINBARS );
   strcpy( env.maintitle.title, "Column Chart" );
   _pg_chart( &env, categories, values, NUM_VALUES );
   getch();
   _setvideomode( _DEFAULTMODE );
}

Classification: PC Graphics
Systems: DOS, QNX
Synopsis:  

#include <pgchart.h>  
short _FAR _pg_resetpalette( void );

Description:  The _pg_resetpalette function resets the internal palette of the presentation graphics system to default values. The palette controls the colors, line styles, fill patterns and plot characters used to display each series of data in a chart. The default palette chosen is dependent on the current video mode.

Returns:  The _pg_resetpalette function returns zero if successful; otherwise, a non-zero value is returned.

See Also:  _pg_defaultchart, _pg_initchart, _pg_chart, _pg_chartpie, _pg_chartscatter, _pg_getpalette, _pg_setpalette
Example:
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#if defined ( __386__ )
#define _FAR
#else
#define _FAR    __far
#endif

#define NUM_VALUES 4

char _FAR *categories[ NUM_VALUES ] = {
    "Jan", "Feb", "Mar", "Apr"
};

float values[ NUM_VALUES ] = {
    20, 45, 30, 25
};

char bricks[ 8 ] = {
    0xff, 0x80, 0x80, 0x80, 0xff, 0x08, 0x08, 0x08
};

main()
{
    chartenv env;
    palettetype pal;

    _setvideomode( _VRES16COLOR );
    _pg_initchart();
    _pg_defaultchart( &env,
                        _PG_COLUMNCHART, _PG_PLAINBARS );
    strcpy( env.maintitle.title, "Column Chart" );
    /* get default palette and change 1st entry */
    _pg_getpalette( &pal );
    pal[ 1 ].color = 12;
    memcpy( pal[ 1 ].fill, bricks, 8 );
    /* use new palette */
    _pg_setpalette( &pal );
    _pg_chart( &env, categories, values, NUM_VALUES );
    /* reset palette to default */
    _pg_resetpalette();
    getch();
    _setvideomode( _DEFAULTMODE );
}

Classification: PC Graphics

Systems: DOS, QNX
Synopsis:  
#include <pgchart.h>
void _FAR _pg_resetstyleset( void );

Description:  The _pg_resetstyleset function resets the internal style-set of the presentation graphics system to default values. The style-set is a set of line styles used for drawing window borders and grid-lines.

Returns:  The _pg_resetstyleset function does not return a value.

See Also:  _pg_defaultchart, _pg_initchart, _pg_chart, _pg_chartpie, _pg_chartscatter, _pg_getstyleset, _pg_setstyleset

Example:  
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#if defined ( __386__ )
#define _FAR
#else
#define _FAR    __far
#endif

#define NUM_VALUES 4

char _FAR *categories[ NUM_VALUES ] = {
    "Jan", "Feb", "Mar", "Apr"
};

float values[ NUM_VALUES ] = {
    20, 45, 30, 25
};

main()
{
    chartenv env;
    styleset style;

    _setvideomode( _VRES16COLOR );
    _pg_initchart();
    _pg_defaultchart( &env,
        _PG_COLUMNCHART, _PGPLAINBARS );
    strcpy( env.maintitle.title, "Column Chart" );
    env.yaxis.grid = 1;
    env.yaxis.gridstyle = 2;
    _pg_getstyleset( &style );
    style[ 2 ] = 0x8888;
    _pg_setstyleset( &style );
    _pg_chart( &env, categories, values, NUM_VALUES );
    _pg_resetstyleset();
    getch();
    _setvideomode( _DEFAULTMODE );
}
Classification: PC Graphics

Systems: DOS, QNX
Synopsis:  
#include <pgchart.h>  
short _FAR _pg_setchardef( short ch,  
unsigned char _FAR *def );

Description: The _pg_setchardef function sets the current bit-map definition for the character ch. The bit-map is contained in the array def. The current font must be an 8-by-8 bit-mapped font.

Returns: The _pg_setchardef function returns zero if successful; otherwise, a non-zero value is returned.

See Also: _pg_defaultchart, _pg_initchart, _pg_chart, _pg_chartpie, _pg_chartscatter, _pg_getchardef

Example:  
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#define NUM_VALUES 4

float x[ NUM_VALUES ] = {  
  5, 25, 45, 65  
};

float y[ NUM_VALUES ] = {  
  5, 45, 25, 65  
};

char diamond[ 8 ] = {  
  0x10, 0x28, 0x44, 0x82, 0x44, 0x28, 0x10, 0x00  
};

main()  
{  
  chartenv env;
  char old_def[ 8 ];

  _setvideomode( _VRES16COLOR );
  _pg_initchart();
  _pg_defaultchart( &env,  
                   _PG_SCATTERCHART, _PG_POINTANDLINE );
  strcpy( env.maintitle.title, "Scatter Chart" );
  /* change asterisk character to diamond */
  _pg_getchardef( '*', old_def );
  _pg_setchardef( '*', diamond );
  _pg_chartscatter( &env, x, y, NUM_VALUES );
  _pg_setchardef( '*', old_def );
  getch();
  _setvideomode( _DEFAULTMODE );
}

Classification: PC Graphics

Systems: DOS, QNX

750  Library Functions and Macros
Synopsis:

```c
#include <pgchart.h>
short _FAR _pg_setpalette( paletteentry _FAR *pal );
```

Description: The `_pg_setpalette` function sets the internal palette of the presentation graphics system. The palette controls the colors, line styles, fill patterns and plot characters used to display each series of data in a chart.

The argument `pal` is an array of palette structures containing the new palette. Each element of the palette is a structure containing the following fields:

- `color`: color used to display series
- `style`: line style used for line and scatter charts
- `fill`: fill pattern used to fill interior of bar and pie sections
- `plotchar`: character plotted on line and scatter charts

Returns: The `_pg_setpalette` function returns zero if successful; otherwise, a non-zero value is returned.

See Also: `_pg_defaultchart`, `_pg_initchart`, `_pg_chart`, `_pg_chartpie`, `_pg_chartscatter`, `_pg_getpalette`, `_pg_resetpalette`
Example:

```c
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#if defined ( __386__ )
#define _FAR
#else
#define _FAR    __far
#endif

#define NUM_VALUES 4

char _FAR *categories[ NUM_VALUES ] = {
    "Jan", "Feb", "Mar", "Apr"
};

float values[ NUM_VALUES ] = {
    20, 45, 30, 25
};

char bricks[ 8 ] = {
    0xff, 0x80, 0x80, 0x80, 0xff, 0x08, 0x08, 0x08
};

main()
{
    chartenv env;
    palettetype pal;

    _setvideomode( _VRES16COLOR );
    _pg_initchart();
    _pg_defaultchart( &env, _PG_COLUMNCHART, _PG_PLAINBARS );
    strcpy( env.maintitle.title, "Column Chart" );
    /* get default palette and change 1st entry */
    _pg_getpalette( &pal );
    pal[ 1 ].color = 12;
    memcpy( pal[ 1 ].fill, bricks, 8 );
    /* use new palette */
    _pg_setpalette( &pal );
    _pg_chart( &env, categories, values, NUM_VALUES );
    /* reset palette to default */
    _pg_resetpalette();
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

**Classification:** PC Graphics

**Systems:** DOS, QNX
Synopsis:  
#include <pgchart.h>
void _FAR _pg_setstyleset( unsigned short _FAR *style );

Description:  The _pg_setstyleset function retrieves the internal style-set of the presentation graphics system. The style-set is a set of line styles used for drawing window borders and grid-lines. The argument style is an array containing the new style-set.

Returns:  The _pg_setstyleset function does not return a value.

See Also:  _pg_defaultchart, _pg_initchart, _pg_chart, _pg_chartpie, _pg_chartscatter, _pg_getstyleset, _pg_resetstyleset

Example:
```c
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#if defined ( __386__ )
#define _FAR
#else
#define _FAR __far
#endif

#define NUM_VALUES 4

char _FAR *categories[ NUM_VALUES ] = {
    "Jan", "Feb", "Mar", "Apr"
};

float values[ NUM_VALUES ] = {
    20, 45, 30, 25
};

main()
{
    chartenv env;
    styleset style;

    _setvideomode( _VRES16COLOR );
    _pg_initchart();
    _pg_defaultchart( &env,
                       _PG_COLUMNCHART, _PG_PLAINBARS );
    strcpy( env.maintitle.title, "Column Chart" );
    /* turn on yaxis grid, and use style 2 */
    env.yaxis.grid = 1;
    env.yaxis.gridstyle = 2;
    /* get default style-set and change entry 2 */
    _pg_getstyleset( &style );
    style[ 2 ] = 0x8888;
    /* use new style-set */
    _pg_setstyleset( &style );
    _pg_chart( &env, categories, values, NUM_VALUES );
    /* reset style-set to default */
    _pg_resetstyleset();
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

**Classification:** PC Graphics

**Systems:** DOS, QNX
Synopsis:  
#include <pgchart.h>
short _FAR _pg_vlabelchart( chartenv _FAR *env,
    short x, short y,
    short color,
    char _FAR *label );

Description: The _pg_vlabelchart function displays the text string label on the chart described by the env chart structure. The string is displayed vertically starting at the point (x, y), relative to the upper left corner of the chart. The color specifies the palette color used to display the string.

Returns: The _pg_vlabelchart function returns zero if successful; otherwise, a non-zero value is returned.

See Also:  _pg_defaultchart, _pg_initchart, _pg_chart, _pg_chartpie, _pg_chartscatter, _pg_hlabelchart

Example:  
#include <graph.h>
#include <pgchart.h>
#include <string.h>
#include <conio.h>

#if defined ( __386__ )
    #define _FAR
#else
    #define _FAR __far
#endif

#define NUM_VALUES 4
char _FAR *categories[ NUM_VALUES ] = {
   "Jan", "Feb", "Mar", "Apr"
};

float values[ NUM_VALUES ] = {
   20, 45, 30, 25
};

main()
{
    chartenv env;
    _setvideomode( _VRES16COLOR );
    _pg_initchart();
    _pg_defaultchart( &env,
        _PG_COLUMNCHART, _PG_PLAINBARS );
    strcpy( env.maintitle.title, "Column Chart" );
    _pg_chart( &env, categories, values, NUM_VALUES );
    _pg_hlabelchart( &env, 64, 32, 1, "Horizontal label" );
    _pg_vlabelchart( &env, 48, 32, 1, "Vertical label" );
    getch();
    _setvideomode( _DEFAULTMODE );
}

Classification: PC Graphics

Systems:  DOS, QNX
_pie Functions

Synopsis:
#include <graph.h>
short _FAR _pie( short fill, short x1, short y1,
    short x2, short y2,
    short x3, short y3,
    short x4, short y4 );

short _FAR _pie_w( short fill, double x1, double y1,
    double x2, double y2,
    double x3, double y3,
    double x4, double y4 );

short _FAR _pie_wxy( short fill,
    struct _wxycoord _FAR *p1,
    struct _wxycoord _FAR *p2,
    struct _wxycoord _FAR *p3,
    struct _wxycoord _FAR *p4 );

Description: The _pie functions draw pie-shaped wedges. The _pie function uses the view coordinate system. The _pie_w and _pie_wxy functions use the window coordinate system.

The pie wedges are drawn by drawing an elliptical arc (in the way described for the _arc functions) and then joining the center of the rectangle that contains the ellipse to the two endpoints of the arc.

The elliptical arc is drawn with its center at the center of the rectangle established by the points (x1, y1) and (x2, y2). The arc is a segment of the ellipse drawn within this bounding rectangle. The arc starts at the point on this ellipse that intersects the vector from the centre of the ellipse to the point (x3, y3). The arc ends at the point on this ellipse that intersects the vector from the centre of the ellipse to the point (x4, y4). The arc is drawn in a counter-clockwise direction with the current plot action using the current color and the current line style.

The following picture illustrates the way in which the bounding rectangle and the vectors specifying the start and end points are defined.
When the coordinates \((x_1, y_1)\) and \((x_2, y_2)\) establish a line or a point (this happens when one or more of the x-coordinates or y-coordinates are equal), nothing is drawn.

The argument \(fill\) determines whether the figure is filled in or has only its outline drawn. The argument can have one of two values:

\[ \textit{GFILLINTERIOR} \]
fill the interior by writing pixels with the current plot action using the current color and the current fill mask

\[ \textit{GBORDER} \]
leave the interior unchanged; draw the outline of the figure with the current plot action using the current color and line style

**Returns:** The \_pie functions return a non-zero value when the figure was successfully drawn; otherwise, zero is returned.

**See Also:** \_arc, \_ellipse, \_setcolor, \_setfillmask, \_setlinestyle, \_setplotaction

**Example:**
```c
#include <conio.h>
#include <graph.h>

main()
{
    _setvideomode( _VRES16COLOR );
    _pie( _GBORDER, 120, 90, 520, 390,
          140, 20, 190, 460 );
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

produces the following:
_pie Functions

Classification: PC Graphics

Systems:  _pie - DOS, QNX
          _pie_w - DOS, QNX
          _pie_wxy - DOS, QNX
Synopsis:    
#include <io.h>
int _pipe( int *phandles, unsigned psize, int textmode );

Description:  The _pipe function creates a pipe (an unnamed FIFO) and places a file descriptor for the read end of
the pipe in phandles[0] and a file descriptor for the write end of the pipe in phandles[1]. Their integer
values are the two lowest available at the time of the _pipe function call. The O_NONBLOCK flag is
cleared for both file descriptors. (The fcntl call can be used to set the O_NONBLOCK flag.)

Data can be written to file descriptor phandles[1] and read from file descriptor phandles[0]. A read on
file descriptor phandles[0] returns the data written to phandles[1] on a first-in-first-out (FIFO) basis.

This function is typically used to connect together standard utilities to act as filters, passing the write
end of the pipe to the data producing process as its STDOUT_FILENO and the read end of the pipe to
the data consuming process as its STDIN_FILENO. (either via the traditional fork/dup2/exec or the
more efficient spawn calls).

If successful, _pipe marks for update the st_mtime, st_ctime, st_atime and st_atime fields of the pipe
for updating.

Returns:    The _pipe function returns zero on success. Otherwise, (-1) is returned and errno is set to indicate
the error.

Errors:     When an error has occurred, errno contains a value indicating the type of error that has been detected.
If any of the following conditions occur, the _pipe function shall return (-1) and set errno to the
 corresponding value:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMFILE</td>
<td>The calling process does not have at least 2 unused file descriptors available.</td>
</tr>
<tr>
<td>ENFILE</td>
<td>The number of simultaneously open files in the system would exceed the configured limit.</td>
</tr>
<tr>
<td>ENOSPC</td>
<td>There is insufficient space available to allocate the pipe buffer.</td>
</tr>
<tr>
<td>EROFS</td>
<td>The pipe pathname space is a read-only filesystem.</td>
</tr>
</tbody>
</table>

See Also:   open, _pclose, perror, _popen, read, write

Example:    
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <fcntl.h>
#include <io.h>
#include <process.h>

static int handles[2] = { 0, 0 };
static int pid;

Library Functions and Macros  759
create_pipe()
{
    if( _pipe( (int *)&handles, 2048, _O_BINARY ) == -1 ) {
        perror( "create_pipe" );
        exit( EXIT_FAILURE );
    }
}

create_child( char *name )
{
    char buff[10];

    itoa( handles[0], buff, 10 );
    pid = spawnl( P_NOWAIT, name,
                    "_pipe", buff, NULL );
    close( handles[0] );
    if( pid == -1 ) {
        perror( "create_child" );
        close( handles[1] );
        exit( EXIT_FAILURE );
    }
}

fill_pipe()
{
    int i;
    int rc;

    for( i = 1; i <= 10; i++ ) {
        printf( "Child, what is 5 times %d\n", i );
        rc = write( handles[1], &i, sizeof( int ) );
        if( rc < sizeof( int ) ) {
            perror( "fill_pipe" );
            close( handles[1] );
            exit( EXIT_FAILURE );
        }
    }
    /* indicate that we are done */
    i = -1;
    write( handles[1], &i, sizeof( int ) );
    close( handles[1] );
}
empty_pipe( int in_pipe )
{
    int i;
    int amt;
    for(;;) {
        amt = read( in_pipe, &i, sizeof( int ) );
        if( amt != sizeof( int ) || i == -1 )
            break;
        printf( "Parent, 5 times %d is %d\n", i, 5*i );
    }
    if( amt == -1 ) {
        perror( "empty_pipe" );
        exit( EXIT_FAILURE );
    }
    close( in_pipe );
}

void main( int argc, char *argv[] )
{
    if( argc <= 1 ) {
        /* we are the spawning process */
        create_pipe();
        create_child( argv[0] );
        fill_pipe();
    } else {
        /* we are the spawned process */
        empty_pipe( atoi( argv[1] ) );
    }
    exit( EXIT_SUCCESS );
}

produces the following:

Child, what is 5 times 1
Child, what is 5 times 2
Parent, 5 times 1 is 5
Parent, 5 times 2 is 10
Child, what is 5 times 3
Child, what is 5 times 4
Parent, 5 times 3 is 15
Parent, 5 times 4 is 20
Child, what is 5 times 5
Child, what is 5 times 6
Parent, 5 times 5 is 25
Parent, 5 times 6 is 30
Child, what is 5 times 7
Child, what is 5 times 8
Parent, 5 times 7 is 35
Parent, 5 times 8 is 40
Child, what is 5 times 9
Child, what is 5 times 10
Parent, 5 times 9 is 45
Parent, 5 times 10 is 50

Classification: WATCOM

Systems: Win32, OS/2 1.x(all), OS/2-32
**Synopsis:**

```c
#include <graph.h>
short _FAR _polygon( short fill, short numpts,
                      struct xycoord _FAR *points );

short _FAR _polygon_w( short fill, short numpts,
                       double _FAR *points );

short _FAR _polygon_wxy( short fill, short numpts,
                         struct _wxycoord _FAR *points );
```

**Description:**

The `_polygon` functions draw polygons. The `_polygon` function uses the view coordinate system. The `_polygon_w` and `_polygon_wxy` functions use the window coordinate system.

The polygon is defined as containing `numpts` points whose coordinates are given in the array `points`.

The argument `fill` determines whether the polygon is filled in or has only its outline drawn. The argument can have one of two values:

- `_GFILLINTERIOR` fill the interior by writing pixels with the current plot action using the current color and the current fill mask
- `_GBORDER` leave the interior unchanged; draw the outline of the figure with the current plot action using the current color and line style

**Returns:**

The `_polygon` functions return a non-zero value when the polygon was successfully drawn; otherwise, zero is returned.

**See Also:**

_setcolor, _setfillmask, _setlinestyle, _setplotaction

**Example:**

```c
#include <conio.h>
#include <graph.h>

struct xycoord points[ 5 ] = {
   319, 140, 224, 209, 261, 320,
   378, 320, 415, 209
};

main()
{
   _setvideomode( _VRES16COLOR );
   _polygon( _GBORDER, 5, points );
   getch();
   _setvideomode( _DEFAULTMODE );
}
```

produces the following:
Classification: _polygon is PC Graphics

Systems:  
_polygon - DOS, QNX  
_polygon_w - DOS, QNX  
_polygon_wxy - DOS, QNX
Synopsis: 

```c
#include <stdio.h>
FILE *_popen( const char *command, const char *mode );
FILE *wpopen( const wchar_t *command, const wchar_t *mode );
```

Description: The `_popen` function executes the command specified by `command` and creates a pipe between the calling process and the executed command.

Depending on the `mode` argument, the stream pointer returned may be used to read from or write to the pipe.

The executed command has an environment the same as its parents. The command will be started as follows: `spawnl(<shell_path>, <shell>, "-c", command, (char *)NULL);`

where `<shell_path>` is an unspecified path for the shell utility and `<shell>` is one of "command.com" (DOS, Windows 95) or "cmd.exe" (Windows NT/2000, OS/2).

The `mode` argument to `_popen` is a string that specifies an I/O mode for the pipe.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;r&quot;</td>
<td>The calling process will read from the standard output of the child process using the stream pointer returned by <code>_popen</code>.</td>
</tr>
<tr>
<td>&quot;w&quot;</td>
<td>The calling process will write to the standard input of the child process using the stream pointer returned by <code>_popen</code>.</td>
</tr>
</tbody>
</table>

The letter "t" may be added to any of the above modes to indicate that the file is (or must be) a text file (i.e., CR/LF pairs are converted to newline characters).

The letter "b" may be added to any of the above modes to indicate that the file is (or must be) a binary file (an ANSI requirement for portability to systems that make a distinction between text and binary files).

When default file translation is specified (i.e., no "t" or "b" is specified), the value of the global variable `_fmode` establishes whether the file is to treated as a binary or a text file. Unless this value is changed by the program, the default will be text mode.

A stream opened by `_popen` should be closed by the `pclose` function.

Returns: The `_popen` function returns a non-NULL stream pointer upon successful completion. If `_popen` is unable to create either the pipe or the subprocess, a NULL stream pointer is returned and `errno` is set appropriately.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EINVAL</td>
<td>The <code>mode</code> argument is invalid.</td>
</tr>
</tbody>
</table>

`_popen` may also set `errno` values as described by the `_pipe` and `spawnl` functions.

See Also: `_grow_handles`, `pclose`, `perror`, `_pipe`
Example:

```c
/*
* Executes a given program, converting all
* output to upper case.
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

char   buffer[256];

void main( int argc, char **argv )
{
    int   i;
    int   c;
    FILE  *f;

    for( i = 1; i < argc; ++i ) {
        strcat( buffer, argv[i] );
        strcat( buffer, " " );
    }

    if( ( f = _popen( buffer, "r" ) ) == NULL ) {
        perror( "_popen" );
        exit( 1 );
    }

    while( ( c = getc(f) ) != EOF ) {
        if( islower( c ) )
            c = toupper( c );
        putchar( c );
    }

    _pclose( f );
}
```

Classification: WATCOM

Systems: 
- _popen - Win32, OS/2 1.x(all), OS/2-32
- _wpopen - Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <math.h>
double pow( double x, double y );

Description:  The pow function computes $x$ raised to the power $y$. A domain error occurs if $x$ is zero and $y$ is less than or equal to 0, or if $x$ is negative and $y$ is not an integer. A range error may occur.

Returns:  The pow function returns the value of $x$ raised to the power $y$. When the argument is outside the permissible range, the matherr function is called. Unless the default matherr function is replaced, it will set the global variable errno to EDOM, and print a “DOMAIN error” diagnostic message using the stderr stream.

See Also:  exp, log, sqrt

Example:  
#include <stdio.h>  
#include <math.h>  

void main()  
{  
    printf( "%f\n", pow( 1.5, 2.5 ) );  
}  

produces the following:

2.755676

Classification:  ANSI

Systems:  Math
Synopsis:    
#include <stdio.h>
int printf( const char *format, ... );
#include <wchar.h>
int wprintf( const wchar_t *format, ... );

Safer C:    
The Safer C Library extension provides the printf_s function which is a safer alternative to printf. This newer printf_s function is recommended to be used instead of the traditional "unsafe" printf function.

Description:    
The printf function writes output to the file designated by stdout under control of the argument format. The format string is described below.

The wprintf function is identical to printf except that it accepts a wide-character string argument for format.

Returns:    
The printf function returns the number of characters written, or a negative value if an output error occurred.

The wprintf function returns the number of wide characters written, or a negative value if an output error occurred. When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also:    
_bprintf, cprintf, fprintf, sprintf, _vbprintf, vcprintf, vfprintf, vprintf, vsprintf

Example:    
#include <stdio.h>

void main( void )
{
    char *weekday, *month;

    weekday = "Saturday";
    month = "April";
    printf( "%s, %s %d, %d\n",
            weekday, month, 18, 1987 );
    printf( "%f1 = %8.4f f2 = %10.2E x = %#08x i = %d\n",
            23.45,      3141.5926,   0x1db,     -1 );
}

produces the following:

Saturday, April 18, 1987
f1 =  23.4500 f2 =  3.14E+003 x = 0x0001db i = -1

Format Control String:    
The format control string consists of ordinary characters, that are written exactly as they occur in the format string, and conversion specifiers, that cause argument values to be written as they are encountered during the processing of the format string. An ordinary character in the format string is any character, other than a percent character (%), that is not part of a conversion specifier. A conversion specifier is a sequence of characters in the format string that begins with a percent character (%) and is followed, in sequence, by the following:
The valid format control flags are:

```
-" " the formatted item is left-justified within the field; normally, items are right-justified

"+" a signed, positive object will always start with a plus character (+); normally, only negative items begin with a sign

" " a signed, positive object will always start with a space character; if both "+" and " " are specified, "+" overrides ""

"##" an alternate conversion form is used:

  - for "b" (unsigned binary) and "o" (unsigned octal) conversions, the precision is incremented, if necessary, so that the first digit is "0".

  - for "x" or "X" (unsigned hexadecimal) conversions, a non-zero value is prepended with "0x" or "0X" respectively.

  - for "e", "E", "f", "F", "g" or "G" (any floating-point) conversions, the result always contains a decimal-point character, even if no digits follow it; normally, a decimal-point character appears in the result only if there is a digit to follow it.

  - in addition to the preceding, for "g" or "G" conversions, trailing zeros are not removed from the result.

If no field width is specified, or if the value that is given is less than the number of characters in the converted value (subject to any precision value), a field of sufficient width to contain the converted value is used. If the converted value has fewer characters than are specified by the field width, the value is padded on the left (or right, subject to the left-justification flag) with spaces or zero characters ("0"). If the field width begins with "0" and no precision is specified, the value is padded with zeros; otherwise the value is padded with spaces. If the field width is "/", a value of type `int` from the argument list is used (before a precision argument or a conversion argument) as the minimum field width. A negative field width value is interpreted as a left-justification flag, followed by a positive field width.

As with the field width specifier, a precision specifier of "/" causes a value of type `int` from the argument list to be used as the precision specifier. If no precision value is given, a precision of 0 is used. The precision value affects the following conversions:
• For "b", "d", "i", "o", "u", "x" and "X" (integer) conversions, the precision specifies the minimum number of digits to appear.

• For "e", "E", "f" and "F" (fixed-precision, floating-point) conversions, the precision specifies the number of digits to appear after the decimal-point character.

• For "g" and "G" (variable-precision, floating-point) conversions, the precision specifies the maximum number of significant digits to appear.

• For "s" or "S" (string) conversions, the precision specifies the maximum number of characters to appear.

A type length specifier affects the conversion as follows:

• "hh" causes a "b", "d", "i", "o", "u", "x" or "X" (integer) format conversion to treat the argument as a signed char or unsigned char argument. Note that, although the argument may have been promoted to an int as part of the function call, the value is converted to the smaller type before it is formatted.

• "hh" causes an "n" (converted length assignment) operation to assign the converted length to an object of type signed char.

• "h" causes a "b", "d", "i", "o", "u", "x" or "X" (integer) format conversion to treat the argument as a short int or unsigned short int argument. Note that, although the argument may have been promoted to an int as part of the function call, the value is converted to the smaller type before it is formatted.

• "h" causes an "f" format conversion to interpret a long argument as a fixed-point number consisting of a 16-bit signed integer part and a 16-bit unsigned fractional part. The integer part is in the high 16 bits and the fractional part is in the low 16 bits.

```c
struct fixpt {
    unsigned short fraction; /* Intel architecture! */
    signed short integral;
};

struct fixpt foo1 = 0x8000, 1234; /* represents 1234.5 */
struct fixpt foo2 = 0x8000, -1; /* represents -0.5 (-1+.5) */
```

The value is formatted with the same rules as for floating-point values. This is a Watcom extension.

• "h" causes an "n" (converted length assignment) operation to assign the converted length to an object of type short int.

• "h" causes an "s" operation to treat the argument string as an ASCII character string composed of 8-bit characters.

For printf and related byte input/output functions, this specifier is redundant. For wprintf and related wide character input/output functions, this specifier is required if the argument string is to be treated as an 8-bit ASCII character string; otherwise it will be treated as a wide character string.

```
struct fixpt {
    unsigned short fraction; /* Intel architecture! */
    signed short integral;
};

struct fixpt foo1 = 0x8000, 1234; /* represents 1234.5 */
struct fixpt foo2 = 0x8000, -1; /* represents -0.5 (-1+.5) */
```
\begin{verbatim}
printf(    "%s%d", "Num=", 12345 );
wprintf( L "%hs%d", "Num=", 12345 );
\end{verbatim}

- "l" causes a "b", "d", "i", "o", "u", "x" or "X" (integer) conversion to process a `long int` or `unsigned long int` argument.

- "l" causes an "n" (converted length assignment) operation to assign the converted length to an object of type `long int`.

- "l" or "w" cause an "s" operation to treat the argument string as a wide character string (a string composed of characters of type `wchar_t`).

For `printf` and related byte input/output functions, this specifier is required if the argument string is to be treated as a wide character string; otherwise it will be treated as an 8-bit ASCII character string. For `wprintf` and related wide character input/output functions, this specifier is redundant.

\begin{verbatim}
printf( L "%ls%d", L "Num=", 12345 );
wprintf( L "%ls%d", L "Num=", 12345 );
\end{verbatim}

- "ll" causes a "b", "d", "i", "o", "u", "x" or "X" (integer) conversion to process a `long long` or `unsigned long long` argument (e.g., `%lld`).

- "ll" causes an "n" (converted length assignment) operation to assign the converted length to an object of type `long long int`.

- "j" causes a "b", "d", "i", "o", "u", "x" or "X" (integer) conversion to process an `intmax_t` or `uintmax_t` argument.

- "j" causes an "n" (converted length assignment) operation to assign the converted length to an object of type `intmax_t`.

- "z" causes a "b", "d", "i", "o", "u", "x" or "X" (integer) conversion to process a `size_t` or the corresponding signed integer type argument.

- "z" causes an "n" (converted length assignment) operation to assign the converted length to an object of signed integer type corresponding to `size_t`.

- "t" causes a "b", "d", "i", "o", "u", "x" or "X" (integer) conversion to process a `ptrdiff_t` or the corresponding unsigned integer type argument.

- "l64" causes a "b", "d", "i", "o", "u", "x" or "X" (integer) conversion to process an `__int64` or `unsigned __int64` argument (e.g., `%I64d`).

- "L" causes an "e", "E", "f", "F", "g", "G" (double) conversion to process a `long double` argument.

- "W" causes the pointer associated with "n", "p", "s" conversions to be treated as a far pointer.

- "N" causes the pointer associated with "n", "p", "s" conversions to be treated as a near pointer.
The valid conversion type specifiers are:

- **b**: An argument of type `int` is converted to an unsigned binary notation and written to the output stream. The default precision is 1, but if more digits are required, leading zeros are added.

- **c**: An argument of type `int` is converted to a value of type `char` and the corresponding ASCII character code is written to the output stream.

- **C**: An argument of type `wchar_t` is converted to a multibyte character and written to the output stream.

- **d, i**: An argument of type `int` is converted to a signed decimal notation and written to the output stream. The default precision is 1, but if more digits are required, leading zeros are added.

- **e, E**: An argument of type `double` is converted to a decimal notation in the form \([-\]d.dde[+|\-]ddd\) similar to FORTRAN exponential (E) notation. The leading sign appears (subject to the format control flags) only if the argument is negative. If the argument is non-zero, the digit before the decimal-point character is non-zero. The precision is used as the number of digits following the decimal-point character. If the precision is not specified, a default precision of six is used. If the precision is 0, the decimal-point character is suppressed. The value is rounded to the appropriate number of digits. For "E" conversions, the exponent begins with the character "E" rather than "e". The exponent sign and a three-digit number (that indicates the power of ten by which the decimal fraction is multiplied) are always produced.

- **f, F**: An argument of type `double` is converted to a decimal notation in the form \([-\]ddd.dd\) similar to FORTRAN fixed-point (F) notation. The leading sign appears (subject to the format control flags) only if the argument is negative. The precision is used as the number of digits following the decimal-point character. If the precision is not specified, a default precision of six is used. If the precision is 0, the decimal-point character is suppressed, otherwise, at least one digit is produced before the decimal-point character. The value is rounded to the appropriate number of digits.

- **g, G**: An argument of type `double` is converted using either the "f" or "e" (or "F" or "E", for a "G" conversion) style of conversion depending on the value of the argument. In either case, the precision specifies the number of significant digits that are contained in the result. "e" style conversion is used only if the exponent from such a conversion would be less than -4 or greater than the precision. Trailing zeros are removed from the result and a decimal-point character only appears if it is followed by a digit.

- **n**: The number of characters that have been written to the output stream is assigned to the integer pointed to by the argument. No output is produced.

- **o**: An argument of type `int` is converted to an unsigned octal notation and written to the output stream. The default precision is 1, but if more digits are required, leading zeros are added.

- **p, P**: An argument of type `void *` is converted to a value of type `int` and the value is formatted as for a hexadecimal ("x") conversion.

- **s**: Characters from the string specified by an argument of type `char *` or `wchar_t *`, up to, but not including the terminating null character (\"\0\"), are written to the output stream. If a precision is specified, no more than that many characters (bytes) are written (e.g., \%.7s)

For `printf`, this specifier refers to an ASCII character string unless the "l" or "w" modifiers are used to indicate a wide character string.
For wprintf, this specifier refers to a wide character string unless the "h" modifier is used to indicate an ASCII character string.

S Characters from the string specified by an argument of type wchar_t *, up to, but not including the terminating null wide character (L'\0'), are converted to multibyte characters and written to the output stream. If a precision is specified, no more than that many characters (bytes) are written (e.g., %.7S)

u An argument of type int is converted to an unsigned decimal notation and written to the output stream. The default precision is 1, but if more digits are required, leading zeros are added.

x, X An argument of type int is converted to an unsigned hexadecimal notation and written to the output stream. The default precision is 1, but if more digits are required, leading zeros are added. Hexadecimal notation uses the digits "0" through "9" and the characters "a" through "f" or "A" through "F" for "x" or "X" conversions respectively, as the hexadecimal digits. Subject to the alternate-form control flag, "0x" or "0X" is prepended to the output.

Any other conversion type specifier character, including another percent character (%), is written to the output stream with no special interpretation.

The arguments must correspond with the conversion type specifiers, left to right in the string; otherwise, indeterminate results will occur.

If the value corresponding to a floating-point specifier is infinity, or not a number (NaN), then the output will be "inf" or "-inf" for infinity, and "nan" or "-nan" for NaN's. If the conversion specifier is an uppercase character (ie. "E", "F", or "G"), the output will be uppercase as well ("INF", "NAN"), otherwise the output will be lowercase as noted above.

The pointer size specification ("N" or "W") is only effective on platforms that use a segmented memory model, although it is always recognized.

For example, a specifier of the form "%8.*f" will define a field to be at least 8 characters wide, and will get the next argument for the precision to be used in the conversion.

Classification: ANSI (except for N, W pointer size modifiers and b, I64 specifiers)

Systems: printf - All, Netware
          wprintf - All
Synopsis:
#include <stdio.h>
int printf_s( const char * restrict format, ... );
#define __STDC_WANT_LIB_EXT1__ 1
#include <wchar.h>
int wprintf_s( const wchar_t * restrict format, ... );

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and printf_s will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

The format argument shall not be a null pointer. The %n specifier (modified or not by flags, field width, or precision) shall not appear in the string pointed to by format. Any argument to printf_s corresponding to a %s specifier shall not be a null pointer.

If there is a runtime-constraint violation, the printf_s function does not attempt to produce further output, and it is unspecified to what extent printf_s produced output before discovering the runtime-constraint violation.

Description: The printf_s function is equivalent to the printf function except for the explicit runtime-constraints listed above.

The wprintf_s function is identical to printf_s except that it accepts a wide-character string argument for format.

Returns: The printf_s function returns the number of characters written, or a negative value if an output error or runtime-constraint violation occurred.

The wprintf_s function returns the number of wide characters written, or a negative value if an output error or runtime-constraint violation occurred.

See Also: _bprintf, cprintf, fprintf, printf, sprintf, _vbprintf, vcprintf, vfprintf, vprintf, vsprintf

Example:
#include <stdio.h>
#define __STDC_WANT_LIB_EXT1__ 1

void main( void )
{
    char *weekday, *month;
    weekday = "Saturday";
    month = "April";
    printf_s( "%s, %s %d, %d\n",
              weekday, month, 18, 1987 );
    printf_s( "%f = %8.4f %f = %10.2E x = %#08x i = %d\n",
              23.45, 3141.5926, 0x1db, -1 );
}

produces the following:

Saturday, April 18, 1987
f1 = 23.4500 f2 = 3.14E+003 x = 0x1db i = -1

Classification: printf_s is TR 24731
printf_s, wprintf_s

wprintf_s is TR 24731

Systems:

printf_s - All, Netware
wprintf_s - All
Synopsis:  #include <stdio.h>
int putc( int c, FILE *fp );
#include <stdio.h>
#include <wchar.h>
wint_t putwc( wint_t c, FILE *fp );

Description:  The putc function is equivalent to fputc, except it may be implemented as a macro. The putc function writes the character specified by the argument c to the output stream designated by fp.

The putwc function is identical to putc except that it converts the wide character specified by c to a multibyte character and writes it to the output stream.

Returns:  The putc function returns the character written or, if a write error occurs, the error indicator is set and putc returns EOF.

The putwc function returns the wide character written or, if a write error occurs, the error indicator is set and putwc returns WEOF.

When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also:  fopen, fputc, fputc, fputs, putchar, puts, ferror

Example:  #include <stdio.h>

void main()
{
    FILE *fp;
    int c;

    fp = fopen( "file", "r" );
    if( fp != NULL ) {
        while( (c = fgetc( fp )) != EOF )
            putc( c, stdout );
        fclose( fp );
    }
}

Classification:  putc is ANSI
putwc is ANSI

Systems:  putc - All, Netware
putwc - All
Synopsis:  #include <conio.h>  
  int putch( int c );

Description:  The putch function writes the character specified by the argument c to the console.

Returns:  The putch function returns the character written.

See Also:  getch, getche, kbhit, ungetch

Example:  #include <conio.h>  
#include <stdio.h>  

  void main()
  {
    FILE *fp;
    int c;

    fp = fopen( "file", "r" );
    if ( fp != NULL ) {
      while( (c = fgetc( fp )) != EOF )
        putch( c );
    }
    fclose( fp );
  }

Classification:  WATCOM

Systems:  All, Netware
Synopsis:  #include <stdio.h>
int putchar( int c );
#include <wchar.h>
wint_t putwchar( wint_t c );

Description:  The putchar function writes the character specified by the argument \texttt{c} to the output stream \texttt{stdout}.

The function is equivalent to

\begin{verbatim}
fputc( c, stdout );
\end{verbatim}

The putwchar function is identical to putchar except that it converts the wide character specified by \texttt{c} to a multibyte character and writes it to the output stream.

Returns:  The putchar function returns the character written or, if a write error occurs, the error indicator is set and putchar returns EOF.

The putwchar function returns the wide character written or, if a write error occurs, the error indicator is set and putwchar returns WEOF.

When an error has occurred, \texttt{errno} contains a value indicating the type of error that has been detected.

See Also:  fopen, fputc, fpputchar, fputs, putc, puts, ferror

Example:  #include <stdio.h>

\begin{verbatim}
void main()
{
    FILE *fp;
    int c;

    fp = fopen( "file", "r" );
    c = fgetc( fp );
    while( c != EOF ) {
        putchar( c );
        c = fgetc( fp );
    }
    fclose( fp );
}
\end{verbatim}

Classification:  putchar is ANSI
putwchar is ANSI

Systems:  putchar - All, Netware
putwchar - All
Synopsis:  

```c
#include <stdlib.h>
int putenv( const char *env_name );
int _putenv( const char *env_name );
int _wputenv( const wchar_t *env_name );
```

Description:   

The environment list consists of a number of environment names, each of which has a value associated with it. Entries can be added to the environment list with the DOS `set` command or with the `putenv` function. All entries in the environment list can be displayed by using the DOS `set` command with no arguments. A program can obtain the value for an environment variable by using the `getenv` function.

When the value of `env_name` has the format

```
env_name=value
```

an environment name and its value is added to the environment list. When the value of `env_name` has the format

```
env_name=
```

the environment name and value is removed from the environment list.

The matching is case-insensitive; all lowercase letters are treated as if they were in upper case.

The space into which environment names and their values are placed is limited. Consequently, the `putenv` function can fail when there is insufficient space remaining to store an additional value.

The `_putenv` function is identical to `putenv`. Use `_putenv` for ANSI naming conventions.

The `_wputenv` function is a wide-character version of `putenv` the `env_name` argument to `_wputenv` is a wide-character string.

`putenv` and `_wputenv` affect only the environment that is local to the current process; you cannot use them to modify the command-level environment. That is, these functions operate only on data structures accessible to the run-time library and not on the environment "segment" created for a process by the operating system. When the current process terminates, the environment reverts to the level of the calling process (in most cases, the operating-system level). However, the modified environment can be passed to any new processes created by `spawn`, `exec`, or `system`, and these new processes get any new items added by `putenv` and `_wputenv`.

With regard to environment entries, observe the following cautions:

- Do not change an environment entry directly; instead, use `putenv` or `_wputenv` to change it. To modify the return value of `putenv` or `_wputenv` without affecting the environment table, use `_strdup` or `strcpy` to make a copy of the string.

- If the argument `env_name` is not a literal string, you should duplicate the string, since `putenv` does not copy the value; for example,

  ```c
  putenv( _strdup( buffer ) );
  ```

- Never free a pointer to an environment entry, because the environment variable will then point to freed space. A similar problem can occur if you pass `putenv` or `_wputenv` a pointer to a local variable, then exit the function in which the variable is declared.
To assign a string to a variable and place it in the environment list:

C>SET INCLUDE=C:\WATCOM\H

To see what variables are in the environment list, and their current assignments:

C>SET
COMSPEC=C:\COMMAND.COM
PATH=C:\;C:\WATCOM
INCLUDE=C:\WATCOM\H
C>

Returns: The `putenv` function returns zero when it is successfully executed and returns -1 when it fails.

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

`ENOMEM` Not enough memory to allocate a new environment string.

See Also: `clearenv`, `getenv`, `setenv`

Example: The following gets the string currently assigned to `INCLUDE` and displays it, assigns a new value to it, gets and displays it, and then removes the environment name and value.

```c
#include <stdio.h>
#include <stdlib.h>

void main()
{
    char *path;
    path = getenv( "INCLUDE" );
    if( path != NULL )
        printf( "INCLUDE=%s\n", path );
    if( putenv( "INCLUDE=mylib;yourlib" ) != 0 )
        printf( "putenv failed" );
    path = getenv( "INCLUDE" );
    if( path != NULL )
        printf( "INCLUDE=%s\n", path );
    if( putenv( "INCLUDE=" ) != 0 )
        printf( "putenv failed" );
}

```

produces the following:

```
INCLUDE=C:\WATCOM\H
INCLUDE=mylib;yourlib
```

Classification: `putenv` is POSIX 1003.1
`_putenv` is not POSIX
`_wputenv` is not POSIX

Systems: `putenv` - All
`_putenv` - All
`_wputenv` - All
Synopsis:  
#include <graph.h>
void _FAR _putimage( short x, short y,
                     char _HUGE *image, short mode );

void _FAR _putimage_w( double x, double y,
                       char _HUGE *image, short mode );

Description:  The _putimage functions display the screen image indicated by the argument image. The _putimage function uses the view coordinate system. The _putimage_w function uses the window coordinate system.

The image is displayed upon the screen with its top left corner located at the point with coordinates (x,y). The image was previously saved using the _getimage functions. The image is displayed in a rectangle whose size is the size of the rectangular image saved by the _getimage functions.

The image can be displayed in a number of ways, depending upon the value of the mode argument. This argument can have the following values:

_GPSET                   replace the rectangle on the screen by the saved image
_GPRESET                 replace the rectangle on the screen with the pixel values of the saved image inverted; this produces a negative image
_GAND                    produce a new image on the screen by ANDing together the pixel values from the screen with those from the saved image
_GOR                     produce a new image on the screen by ORing together the pixel values from the screen with those from the saved image
_GXOR                    produce a new image on the screen by exclusive ORing together the pixel values from the screen with those from the saved image; the original screen is restored by two successive calls to the _putimage function with this value, providing an efficient method to produce animated effects

Returns:  The _putimage functions do not return a value.

See Also:  _getimage, _imagesize
Example:

```c
#include <conio.h>
#include <graph.h>
#include <malloc.h>

main()
{
    char *buf;
    int y;

    _setvideomode( _VRES16COLOR );
    _ellipse( _GFILLINTERIOR, 100, 100, 200, 200 );
    buf = (char*) malloc( 
        _imagesize( 100, 100, 201, 201 ) );
    if( buf != NULL ) {
        _getimage( 100, 100, 201, 201, buf );
        _putimage( 260, 200, buf, _GPSET );
        _putimage( 420, 100, buf, _GPSET );
        for( y = 100; y < 300; ) {
            _putimage( 420, y, buf, _GXOR );
            y += 20;
            _putimage( 420, y, buf, _GXOR );
        }
        free( buf );
    }
    getch();
    _setvideomode( _DEFAULTMODE );
}
```

Classification: _putimage is PC Graphics

Systems: _putimage - DOS, QNX
          _putimage_w - DOS, QNX
**puts, _putws**

**Synopsis:**

```c
#include <stdio.h>
int puts( const char *buf );
#include <stdio.h>
int _putws( const wchar_t *bufs );
```

**Description:** The `puts` function writes the character string pointed to by `buf` to the output stream designated by `stdout`, and appends a new-line character to the output. The terminating null character is not written.

The `_putws` function is identical to `puts` except that it converts the wide character string specified by `buf` to a multibyte character string and writes it to the output stream.

**Returns:** The `puts` function returns `EOF` if an error occurs; otherwise, it returns a non-negative value (the amount written including the new-line character). The `_putws` function returns `WEOF` if a write or encoding error occurs; otherwise, it returns a non-negative value (the amount written including the new-line character). When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

**See Also:** `fopen`, `fputc`, `fputchar`, `fputs`, `putc`, `putchar`, `ferror`

**Example:**

```c
#include <stdio.h>

void main()
{
    FILE *fp;
    char buffer[80];

    fp = freopen( "file", "r", stdin );
    while( gets( buffer ) != NULL ) {
        puts( buffer );
    }
    fclose( fp );
}
```

**Classification:**

- `puts` is ANSI
- `_putws` is not ANSI

**Systems:**

- `puts` - All, Netware
- `_putws` - All

---

782 Library Functions and Macros
Synopsis:
#include <stdio.h>
int _putw( int binint, FILE *fp );

Description: The _putw function writes a binary value of type int to the current position of the stream fp. _putw does not affect the alignment of items in the stream, nor does it assume any special alignment.

_putw is provided primarily for compatibility with previous libraries. Portability problems may occur with _putw because the size of an int and the ordering of bytes within an int differ across systems.

Returns: The _putw function returns the value written or, if a write error occurs, the error indicator is set and _putw returns EOF. Since EOF is a legitimate value to write to fp, use ferror to verify that an error has occurred.

See Also: ferror, fopen, fputc, fputchar, fputs, putc, putchar, puts

Example:  #include <stdio.h>

    void main()
    {
        FILE *fp;
        int c;

        fp = fopen( "file", "r" );
        if( fp != NULL )
        {
            while( (c = _getw( fp )) != EOF )
            {
                _putw( c, stdout );
            }
        }
        fclose( fp );
    }

Classification: WATCOM

Systems: All, Netware
Synopsis:  
#include <stdlib.h>
void qsort( void *base,
    size_t num,
    size_t width,
    int (*compar) ( const void *,
                    const void *) );

Safer C:  The Safer C Library extension provides the qsort_s function which is a safer alternative to qsort. This newer qsort_s function is recommended to be used instead of the traditional "unsafe" qsort function.

Description:  The qsort function sorts an array of num elements, which is pointed to by base, using a modified version of Sedgewick’s Quicksort algorithm. Each element in the array is width bytes in size. The comparison function pointed to by compar is called with two arguments that point to elements in the array. The comparison function shall return an integer less than, equal to, or greater than zero if the first argument is less than, equal to, or greater than the second argument.

The version of the Quicksort algorithm that is employed was proposed by Jon Louis Bentley and M. Douglas McIlroy in the article "Engineering a sort function" published in Software -- Practice and Experience, 23(11):1249-1265, November 1993.

Returns:  The qsort function returns no value.

See Also:  qsort_s, bsearch, bsearch_s

Example:  
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char *CharVect[] = { "last", "middle", "first" };

int compare( const void *op1, const void *op2 )
{
    const char **p1 = (const char **) op1;
    const char **p2 = (const char **) op2;
    return( strcmp( *p1, *p2 ) );
}

void main()
{
    qsort( CharVect, sizeof(CharVect)/sizeof(char *),
           sizeof(char *), compare );
    printf( "%s %s %s\n",
            CharVect[0], CharVect[1], CharVect[2] );
}

produces the following:

first last middle

Classification:  ANSI

Systems:  All, Netware
Synopsis:
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
errno_t qsort_s( void *base, 
    rsize_t nmemb, 
    rsize_t size, 
    int (*compar)( const void *x, const void *y, void *context ), 
    void *context );

Constraints:
If any of the following runtime-constraints is violated, the currently active runtime-constraint handler 
will be invoked and qsort_s will return a non-zero value to indicate an error, or the 
runtime-constraint handler aborts the program.

Neither nmemb nor size shall be greater than RSIZE_MAX. If nmemb is not equal to zero, then neither 
base nor compar shall be a null pointer. If there is a runtime-constraint violation, the qsort_s 
function does not sort the array.

Description:
The qsort_s function sorts an array of nmemb objects, the initial element of which is pointed to by 
base. The size of each object is specified by size. The contents of the array are sorted into ascending 
order according to a comparison function pointed to by compar, which is called with three arguments. 
The first two point to the objects being compared. The function shall return an integer less than, equal 
to, or greater than zero if the first argument is considered to be respectively less than, equal to, or 
greater than the second. The third argument to the comparison function is the context argument passed 
to qsort_s The sole use of context by qsort_s is to pass it to the comparison function. If two 
elements compare as equal, their relative order in the resulting sorted array is unspecified.

Returns:
The qsort_s function returns zero if there was no runtime-constraint violation. Otherwise, a 
non-zero value is returned.

See Also:
qsort, bsearch, bsearch_s

Example:
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char *CharVect[] = { "last", "middle", "first" }; 

int compare( const void *op1, const void *op2, void *context ) 
{
    const char **p1 = (const char **) op1;
    const char **p2 = (const char **) op2;
    return( strcmp( *p1, *p2 ) );
}

void main()
{
    void * context = NULL;
    qsort_s( CharVect, sizeof(CharVect)/sizeof(char *), 
            sizeof(char *), compare, context );
    printf( "%s %s %s\n", 
            CharVect[0], CharVect[1], CharVect[2] );
}

produces the following:
first last middle

Classification: TR 24731

Systems: All, Netware
Synopsis:    #include <signal.h>
            int raise( int condition );

Description: The `raise` function signals the exceptional condition indicated by the `condition` argument. The possible conditions are defined in the `<signal.h>` header file and are documented with the `signal` function. The `signal` function can be used to specify the action which is to take place when such a condition occurs.

Returns: The `raise` function returns zero when the condition is successfully raised and a non-zero value otherwise. There may be no return of control following the function call if the action for that condition is to terminate the program or to transfer control using the `longjmp` function.

See Also: `signal`

Example:    /*
* This program waits until a SIGINT signal
* is received.
*/
#include <stdio.h>
#include <signal.h>

sig_atomic_t signal_count;

static void alarm_handler( int signum )
{
    ++signal_count;
    signal_number = signum;
}

void main()
{
    unsigned long i;

    signal_count = 0;
    signal_number = 0;
    signal( SIGINT, alarm_handler );

    printf("Signal will be auto-raised on iteration ",
        "10000 or hit CTRL-C.
    ");
    printf("Iteration: 
    ");
    for( i = 0; i < 100000; ++i )
    {
        printf("\b\b\b\b\b%*d", 5, i);

        if( i == 10000 ) raise(SIGINT);
        if( signal_count > 0 ) break;
    }


if ( i == 100000 ) {
    printf("\nNo signal was raised.\n");
} else if ( i == 10000 ) {
    printf("\nSignal %d was raised by the "
          "raise() function.\n", signal_number);
} else {
    printf("\nUser raised the signal.\n",
           signal_number);
}

Classification: ANSI

Systems: All, Netware
Synopsis:  
#include <stdlib.h>
int rand( void );

Description: The rand function computes a sequence of pseudo-random integers in the range 0 to RAND_MAX (32767). The sequence can be started at different values by calling the srand function.

Returns: The rand function returns a pseudo-random integer.

See Also: srand

Example:  
#include <stdio.h>
#include <stdlib.h>

void main()
{
   int i;
   for( i=1; i < 10; ++i ) {
      printf( "%d\n", rand() );
   }
}

Classification: ANSI

Systems: All, Netware
Synopsis:

#include <io.h>
int read( int handle, void *buffer, unsigned len );
int _read( int handle, void *buffer, unsigned len );

Description:
The `read` function reads data at the operating system level. The number of bytes transmitted is given by `len` and the data is transmitted starting at the address specified by `buffer`.

The `handle` value is returned by the `open` function. The access mode must have included either `O_RDONLY` or `O_RDWR` when the `open` function was invoked. The data is read starting at the current file position for the file in question. This file position can be determined with the `tell` function and can be set with the `lseek` function.

When `O_BINARY` is included in the access mode, the data is transmitted unchanged. When `O_TEXT` is included in the access mode, the data is transmitted with the extra carriage return character removed before each linefeed character encountered in the original data.

The `_read` function is identical to `read`. Use `_read` for ANSI/ISO naming conventions.

Returns:
The `read` function returns the number of bytes of data transmitted from the file to the buffer (this does not include any carriage-return characters that were removed during the transmission). Normally, this is the number given by the `len` argument. When the end of the file is encountered before the read completes, the return value will be less than the number of bytes requested.

A value of -1 is returned when an input/output error is detected. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: close, creat, fread, open, write

Example:

#include <stdio.h>
#include <fcntl.h>
#include <io.h>

void main( void ){
    int  handle;
    int  size_read;
    char buffer[80];

    /* open a file for input */
    handle = open( "file", O_RDONLY | O_TEXT );
    if( handle != -1 ) {
        /* read the text */
        size_read = read( handle, buffer, sizeof( buffer ) );

        /* test for error */
        if( size_read == -1 ) {
            printf( "Error reading file\n" );
        }
    }
}
/ * close the file          */
         close( handle );
}

Classification:  read is POSIX 1003.1
                   _read is not POSIX
                   _read conforms to ANSI/ISO naming conventions

Systems:        read - All, Netware
                _read - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <direct.h>
struct dirent *readdir( struct dirent *dirp );
struct _wdirent * _wreaddir( _wdirent * dirp );

Description:  
The readdir function obtains information about the next matching file name from the argument dirp.
The argument dirp is the value returned from the opendir function. The readdir function can be called repeatedly to obtain the list of file names contained in the directory specified by the pathname given to opendir. The function closedir must be called to close the directory and free the memory allocated by opendir.

The file <direct.h> contains definitions for the structure dirent.

#if defined(__OS2__) || defined(__NT__)
#define NAME_MAX 255 /* maximum for HPFS or NTFS */
#else
#define NAME_MAX 12 /* 8 chars + '.' + 3 chars */
#endif

typedef struct dirent {
    char   d_dta[ 21 ];        /* disk transfer area */
    char   d_attr;             /* file's attribute */
    unsigned short int d_time; /* file's time */
    unsigned short int d_date; /* file's date */
    long    d_size;            /* file's size */
    char    d_name[ NAME_MAX + 1 ]; /* file's name */
    unsigned short d_ino;      /* serial number */
    char    d_first;           /* flag for 1st time */
} DIR;

The file attribute field d_attr field is a set of bits representing the following attributes.

&_A_RDONLY       /* Read-only file */
&_A_HIDDEN       /* Hidden file */
&_A_SYSTEM       /* System file */
&_A_VOLID        /* Volume-ID entry (only MSFT knows) */
&_A_SUBDIR       /* Subdirectory */
&_A_ARCHIVE      /* Archive file */

If the _A_RDONLY bit is off, then the file is read/write.

The format of the d_time field is described by the following structure (this structure is not defined in any Watcom header file).

typedef struct {
    unsigned short twoseconds : 5; /* seconds / 2 */
    unsigned short minutes : 6;    /* minutes (0,59) */
    unsigned short hours : 5;      /* hours (0,23) */
} ftime_t;

The format of the d_date field is described by the following structure (this structure is not defined in any Watcom header file).
typedef struct {
    unsigned short day     : 5;    /* day (1,31) */
    unsigned short month   : 4;    /* month (1,12) */
    unsigned short year    : 7;    /* 0 is 1980 */
} fdate_t;

See the sample program below for an example of the use of these structures.

The _wreaddir function is identical to readdir except that it reads a directory of wide-character filenames.

The file <direct.h> contains definitions for the structure _wdirent.

struct _wdirent {
    char d_dta[21];      /* disk transfer area */
    char d_attr;         /* file’s attribute */
    unsigned short int d_time; /* file’s time */
    unsigned short int d_date; /* file’s date */
    long d_size;         /* file’s size */
    wchar_t d_name[NAME_MAX+1]; /* file’s name */
    unsigned short d_ino;   /* serial number (not used) */
    char d_first;        /* flag for 1st time */
};

Returns: When successful, readdir returns a pointer to an object of type struct dirent. When an error occurs, readdir returns the value NULL and errno is set to indicate the error. When the end of the directory is encountered, readdir returns the value NULL and errno is unchanged.

When successful, _wreaddir returns a pointer to an object of type struct _wdirent. When an error occurs, _wreaddir returns the value NULL and errno is set to indicate the error. When the end of the directory is encountered, _wreaddir returns the value NULL and errno is unchanged.

Errors: When an error has occurred, errno contains a value indicating the type of error that has been detected.

**EBADF** The argument dirp does not refer to an open directory stream.

See Also: closedir, _dos_find..., opendir, rewinddir

Example: To get a list of files contained in the directory \watcom\h on your default disk:
#include <stdio.h>
#include <dirent.h>

typedef struct {
    unsigned short  twosecs : 5;    /* seconds / 2 */
    unsigned short  minutes : 6;
    unsigned short  hours   : 5;
} ftime_t;

typedef struct {
    unsigned short  day     : 5;
    unsigned short  month   : 4;
    unsigned short  year    : 7;
} fdate_t;

void main()
{
    DIR *dirp;
    struct dirent *direntp;
    ftime_t *f_time;
    fdate_t *f_date;

    dirp = opendir( "\\watcom\h" );
    if( dirp != NULL ) {
        for(;;) {
            direntp = readdir( dirp );
            if( direntp == NULL ) break;
            f_time = (ftime_t *)&direntp->d_time;
            f_date = (fdate_t *)&direntp->d_date;
            printf( "%-12s %d/%2.2d/%2.2d \
" ,
                    direntp->d_name,
                    f_date->year + 1980,
                    f_date->month,
                    f_date->day,
                    f_time->hours,
                    f_time->minutes,
                    f_time->twosecs * 2 );
        }
        closedir( dirp );
    }
}

Note the use of two adjacent backslash characters (\) within character-string constants to signify a single backslash.

Classification: readdir is POSIX 1003.1
_wreaddir is not POSIX

Systems: readdir - All, Netware
_wreaddir - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:
```c
#include <stdlib.h>  For ANSI compatibility (realloc only)
#include <malloc.h>  Required for other function prototypes
void * realloc( void *old_blk, size_t size );
void __based(void) * _brealloc( __segment seg,
                          void __based(void) *old_blk,
                          size_t size );
void __far  * _frealloc( void __far  *old_blk,
                          size_t size );
void __near * _nrealloc( void __near *old_blk,
                          size_t size );
```

Description: When the value of the `old_blk` argument is `NULL`, a new block of memory of `size` bytes is allocated.

If the value of `size` is zero, the corresponding `free` function is called to release the memory pointed to by `old_blk`.

Otherwise, the `realloc` function re-allocates space for an object of `size` bytes by either:

- shrinking the allocated size of the allocated memory block `old_blk` when `size` is sufficiently smaller than the size of `old_blk`.
- extending the allocated size of the allocated memory block `old_blk` if there is a large enough block of unallocated memory immediately following `old_blk`.
- allocating a new block and copying the contents of `old_blk` to the new block.

Because it is possible that a new block will be allocated, any pointers into the old memory should not be maintained. These pointers will point to freed memory, with possible disastrous results, when a new block is allocated.

The function returns `NULL` when the memory pointed to by `old_blk` cannot be re-allocated. In this case, the memory pointed to by `old_blk` is not freed so care should be exercised to maintain a pointer to the old memory block.

```
buffer = (char *) realloc( buffer, 100 );
```

In the above example, `buffer` will be set to `NULL` if the function fails and will no longer point to the old memory block. If `buffer` was your only pointer to the memory block then you will have lost access to this memory.

Each function reallocates memory from a particular heap, as listed below:

<table>
<thead>
<tr>
<th>Function</th>
<th>Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>realloc</code></td>
<td>Depends on data model of the program</td>
</tr>
<tr>
<td><code>_brealloc</code></td>
<td>Based heap specified by <code>seg</code> value</td>
</tr>
<tr>
<td><code>_frealloc</code></td>
<td>Far heap (outside the default data segment)</td>
</tr>
<tr>
<td><code>_nrealloc</code></td>
<td>Near heap (inside the default data segment)</td>
</tr>
</tbody>
</table>

In a small data memory model, the `realloc` function is equivalent to the `_nrealloc` function; in a large data memory model, the `realloc` function is equivalent to the `_frealloc` function.
realloc Functions

Returns: The realloc functions return a pointer to the start of the re-allocated memory. The return value is NULL if there is insufficient memory available or if the value of the *size* argument is zero. The _brealloc function returns _NULLOFF if there is insufficient memory available or if the requested size is zero.

See Also: calloc Functions, _expand Functions, free Functions, halloc, hfree, malloc Functions, _msize Functions, sbrk

Example: 

```c
#include <stdlib.h>
#include <malloc.h>

void main()
{
    char *buffer;
    char *new_buffer;

    buffer = (char *) malloc( 80 );
    new_buffer = (char *) realloc( buffer, 100 );
    if( new_buffer == NULL ) {
        /* not able to allocate larger buffer */
    } else {
        buffer = new_buffer;
    }
}
```

Classification: realloc is ANSI
     _frealloc is not ANSI
     _brealloc is not ANSI
     _nrealloc is not ANSI

Systems: realloc - All, Netware
     _brealloc - DOS/16, Windows, QNX/16, OS/2 1.x(all)
     _frealloc - DOS/16, Windows, QNX/16, OS/2 1.x(all)
     _nrealloc - DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2 1.x(MT), OS/2-32
Synopsis:
#include <graph.h>
short _FAR _rectangle( short fill,
    short x1, short y1,
    short x2, short y2 );

short _FAR _rectangle_w( short fill,
    double x1, double y1,
    double x2, double y2 );

short _FAR _rectangle_wxy( short fill,
    struct _wxycoord _FAR *p1,
    struct _wxycoord _FAR *p2 );

Description:
The _rectangle functions draw rectangles. The _rectangle function uses the view coordinate system. The _rectangle_w and _rectangle_wxy functions use the window coordinate system.

The rectangle is defined with opposite corners established by the points (x1,y1) and (x2,y2).

The argument fill determines whether the rectangle is filled in or has only its outline drawn. The argument can have one of two values:

_GFILLINTERIOR fill the interior by writing pixels with the current plot action using the current color and the current fill mask

_GBORDER leave the interior unchanged; draw the outline of the figure with the current plot action using the current color and line style

Returns: The _rectangle functions return a non-zero value when the rectangle was successfully drawn; otherwise, zero is returned.

See Also: _setcolor, _setfillmask, _setlinestyle, _setplotaction

Example:
#include <conio.h>
#include <graph.h>

main()
{
    _setvideomode( _VRES16COLOR );
    _rectangle( _GBORDER, 100, 100, 540, 380 );
    getch();
    _setvideomode( _DEFAULTMODE );
}

produces the following:
Classification: _rectangle is PC Graphics

Systems:
  _rectangle - DOS, QNX
  _rectangle_w - DOS, QNX
  _rectangle_wxy - DOS, QNX
Synopsis:  
#include <graph.h>
short _FAR _registerfonts( char _FAR *path );

Description:  The _registerfonts function initializes the font graphics system. Fonts must be registered, and a font selected, before text can be displayed with the _outgtext function.

The argument path specifies the location of the font files. This argument is a file specification, and can contain drive and directory components and may contain wildcard characters. The _registerfonts function opens each of the font files specified and reads the font information. Memory is allocated to store the characteristics of the font. These font characteristics are used by the _setfont function when selecting a font.

Returns:  The _registerfonts function returns the number of fonts that were registered if the function is successful; otherwise, a negative number is returned.

See Also:  _unregisterfonts, _setfont, _getfontinfo, _outgtext, _getgtexttextent, _setgtextvector, _getgtextvector

Example:  
#include <conio.h>
#include <stdio.h>
#include <graph.h>

main()
{
    int i, n;
    char buf[10];

    _setvideomode(_VRES16COLOR);
    n = _registerfonts("*.fon");
    for( i = 0; i < n; ++i ) {
        sprintf(buf, "n%d", i);
        _setfont(buf);
        _moveto(100, 100);
        _outgtext("WATCOM Graphics");
        getch();
        _clearscreen(_GCLEARSCREEN);
    }
    _unregisterfonts();
    _setvideomode(_DEFAULTMODE);
}

Classification:  PC Graphics

Systems:  DOS, QNX
_remapallpalette

Synopsis:  
#include <graph.h>
short _FAR _remapallpalette( long _FAR *colors );

Description:  The _remapallpalette function sets (or remaps) all of the colors in the palette. The color values in the palette are replaced by the array of color values given by the argument colors. This function is supported in all video modes, but only works with EGA, MCGA and VGA adapters.

The array colors must contain at least as many elements as there are supported colors. The newly mapped palette will cause the complete screen to change color wherever there is a pixel value of a changed color in the palette.

The representation of colors depends upon the hardware being used. The number of colors in the palette can be determined by using the _getvideoconfig function.

Returns:  The _remapallpalette function returns (-1) if the palette is remapped successfully and zero otherwise.

See Also:  _remappalette, _getvideoconfig

Example:  
#include <conio.h>
#include <graph.h>

long colors[ 16 ] = {
  _BRIGHTWHITE, _YELLOW, _LIGHTMAGENTA, _LIGHTRED,  
  _LIGHTCYAN, _LIGHTGREEN, _LIGHTBLUE, _GRAY, _WHITE,  
  _BROWN, _MAGENTA, _RED, _CYAN, _GREEN, _BLUE, _BLACK, 
};

main()
{  
  int x, y;
  _setvideomode( _VRES16COLOR );  
  for( y = 0; y < 4; ++y ) {  
    for( x = 0; x < 4; ++x ) { 
      _setcolor( x + 4 * y );  
      _rectangle( _GFILLINTERIOR,  
                 x * 160, y * 120,  
                 ( x + 1 ) * 160, ( y + 1 ) * 120 );
    }
  }
  getch();  
  _remapallpalette( colors );  
  getch();  
  _setvideomode( _DEFAULTMODE );
}

Classification:  PC Graphics

Systems:  DOS, QNX

800  Library Functions and Macros
Synopsis:  
#include <graph.h>
long _FAR _remappalette( short pixval, long color );

Description:  The _remappalette function sets (or remaps) the palette color pixval to be the color color. This function is supported in all video modes, but only works with EGA, MCGA and VGA adapters.

The argument pixval is an index in the color palette of the current video mode. The argument color specifies the actual color displayed on the screen by pixels with pixel value pixval. Color values are selected by specifying the red, green and blue intensities that make up the color. Each intensity can be in the range from 0 to 63, resulting in 262144 possible different colors. A given color value can be conveniently specified as a value of type long. The color value is of the form 0x00bbgrr, where bb is the blue intensity, gg is the green intensity and rr is the red intensity of the selected color. The file graph.h defines constants containing the color intensities of each of the 16 default colors.

The _remappalette function takes effect immediately. All pixels on the complete screen which have a pixel value equal to the value of pixval will now have the color indicated by the argument color.

Returns:  The _remappalette function returns the previous color for the pixel value if the palette is remapped successfully; otherwise, (-1) is returned.

See Also:  _remapallpalette, _setvideomode

Example:  
#include <conio.h>
#include <graph.h>

long colors[ 16 ] = {
    _BLACK, _BLUE, _GREEN, _CYAN,
    _RED, _MAGENTA, _BROWN, _WHITE,
    _GRAY, _LIGHTBLUE, _LIGHTGREEN, _LIGHTCYAN,
    _LIGHTRED, _LIGHTMAGENTA, _YELLOW, _BRIGHTWHITE
};

main()
{
    int col;

    _setvideomode( _VRES16COLOR );
    for( col = 0; col < 16; ++col ) {
        _remappalette( 0, colors[ col ] );
        getch();
    }
    _setvideomode( _DEFAULTMODE );
}

Classification:  PC Graphics

Systems:  DOS, QNX
**Synopsis:**

```c
#include <stdio.h>
int remove( const char *filename );
int _wremove( const wchar_t *filename );
```

**Description:**

The `remove` function deletes the file whose name is the string pointed to by `filename`.

The `_wremove` function is identical to `remove` except that it accepts a wide-character string argument.

**Returns:**

The `remove` function returns zero if the operation succeeds, non-zero if it fails. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

**Example:**

```c
#include <stdio.h>

void main()
{
    remove( "vm.tmp" );
}
```

**Classification:**

- `remove` is ANSI
- `_wremove` is not ANSI

**Systems:**

- `remove` - All, Netware
- `_wremove` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
rename, _wrename

Synopsis:  
#include <stdio.h>
int rename( const char *old, const char *new );
int _wrename( const wchar_t *old, const wchar_t *new );

Description:  The rename function causes the file whose name is indicated by the string old to be renamed to the name given by the string new. The _wrename function is identical to rename except that it accepts wide-character string arguments.

Returns:  The rename function returns zero if the operation succeeds, a non-zero value if it fails. When an error has occurred, errno contains a value indicating the type of error that has been detected.

Example:  
#include <stdio.h>

void main()
{
    rename( "old.dat", "new.dat" );
}

Classification: rename is ANSI
_wrename is not ANSI

Systems:  rename - All, Netware
_wrename - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
**Synopsis:**

```c
#include <stdio.h>
void rewind( FILE *fp );
```

**Description:**

The `rewind` function sets the file position indicator for the stream indicated by `fp` to the beginning of the file. It is equivalent to

```c
fseek( fp, 0L, SEEK_SET );
```

except that the error indicator for the stream is cleared.

**Returns:**

The `rewind` function returns no value.

**See Also:**

`fopen`, `clearerr`

**Example:**

```c
#include <stdio.h>

static assemble_pass( int passno )
{
    printf( "Pass %d\n", passno );
}

void main()
{
    FILE *fp;
    
    if( (fp = fopen( "program.asm", "r" )) != NULL ) {
        assemble_pass( 1 );
        rewind( fp );
        assemble_pass( 2 );
        fclose( fp );
    }
}
```

**Classification:** ANSI

**Systems:** All, Netware
Synopsis:
#include <sys/types.h>
#include <direct.h>
void rewinddir( struct dirent *dirp );
void _wrewinddir( _wdirent *dirp );

Description: The rewinddir function resets the position of the directory stream to which dirp refers to the beginning of the directory. It also causes the directory stream to refer to the current state of the corresponding directory, as a call to opendir would have done.

The _wrewinddir function is identical to rewinddir except that it rewinds a directory of wide-character filenames opened by _wopendir.

Returns: The rewinddir function does not return a value.

See Also: closedir, _dos_find..., opendir, readdir

Example: The following example lists all the files in a directory, creates a new file, and then relists the directory.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <direct.h>

void main()
{  
  DIR *dirp;
  struct dirent *direntp;
  int handle;

  dirp = opendir( "\watcom\h\*.*" );
  if( dirp != NULL ) {  
    printf( "Old directory listing\n" );
    for(;;) {  
      direntp = readdir( dirp );
      if( direntp == NULL )  
        break;
      printf( "%s\n", direntp->d_name );
    }
  
  handle = creat( "\watcom\h\file.new",
               S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );
  close( handle );

  rewinddir( dirp );
  printf( "New directory listing\n" );
  for(;;) {  
    direntp = readdir( dirp );
    if( direntp == NULL )  
      break;
    printf( "%s\n", direntp->d_name );
  }
  closedir( dirp );
}  
```
rewinddir, _wrewinddir

Note the use of two adjacent backslash characters (\) within character-string constants to signify a single backslash.

**Classification:** rewinddir is POSIX 1003.1  
_wrewinddir is not POSIX

**Systems:**  
rewinddir - All  
_wrewinddir - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis: 
#include <sys/types.h>
#include <direct.h>
int rmdir( const char *path );
int _rmdir( const char *path );
int _wrmdir( const wchar_t *path );

Description: The rmdir function removes (deletes) the specified directory. The directory must not contain any files or directories. The path can be either relative to the current working directory or it can be an absolute path name.

The _rmdir function is identical to rmdir. Use _rmdir for ANSI/ISO naming conventions.

The _wrmdir function is identical to rmdir except that it accepts a wide-character string argument.

Returns: The rmdir function returns zero if successful and -1 otherwise.

Errors: When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also: chdir, chmod, getcwd, mkdir, stat, umask

Example: To remove the directory called \watcom on drive C:

```c
#include <sys/types.h>
#include <direct.h>

void main( void )
{
    rmdir( "c:\watcom" );
}
```

Note the use of two adjacent backslash characters (\) within character-string constants to signify a single backslash.

Classification: rmdir is POSIX 1003.1
_rmdir is not POSIX
_wrmdir is not POSIX
_rmdir conforms to ANSI/ISO naming conventions

Systems: rmdir - All, Netware
_rmdir - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_wrmdir - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <stdlib.h>  
unsigned int _rotl( unsigned int value,  
                    unsigned int shift );

Description:  The _rotl function rotates the unsigned integer, determined by value, to the left by the number of bits specified in shift. If you port an application using _rotl between a 16-bit and a 32-bit environment, you will get different results because of the difference in the size of integers.

Returns:  The rotated value is returned.

See Also:  _lrotl, _lrotr, _rotr

Example:  
#include <stdio.h>  
#include <stdlib.h>  

unsigned int mask = 0x0F00;  

void main()  
{  
    mask = _rotl( mask, 4 );  
    printf( "%04X\n", mask );  
}

produces the following:

F000

Classification:  WATCOM

Systems:  All, Netware
Synopsis:  
#include <stdlib.h>  
unsigned int _rotr( unsigned int value,  
                    unsigned int shift );

Description:  The _rotr function rotates the unsigned integer, determined by value, to the right by the number of bits specified in shift. If you port an application using _rotr between a 16-bit and a 32-bit environment, you will get different results because of the difference in the size of integers.

Returns:  The rotated value is returned.

See Also:  _lrotl, _lrotr, _rotl

Example:  
#include <stdio.h>  
#include <stdlib.h>  

    unsigned int mask = 0x1230;
    void main()
    {
        mask = _rotr( mask, 4 );
        printf( "%04X\n", mask );
    }

produces the following:

    0123

Classification:  WATCOM

Systems:  All, Netware
Synopsis: #include <stdlib.h>
void *sbrk( int increment );

Description: Under 16-bit DOS and Phar Lap’s 386|DOS-Extender, the data segment is grown contiguously. The "break" value is the address of the first byte of unallocated memory. When a program starts execution, the break value is placed following the code and constant data for the program. As memory is allocated, this pointer will advance when there is no freed block large enough to satisfy an allocation request. The sbrk function can be used to set a new "break" value for the program by adding the value of increment to the current break value. This increment may be positive or negative.

Under other systems, heap allocation is discontiguous. The sbrk function can only be used to allocate additional discontiguous blocks of memory. The value of increment is used to determine the minimum size of the block to be allocated and may not be zero or negative. The actual size of the block that is allocated is rounded up to a multiple of 4K.

The variable _ambksiz defined in <stdlib.h> contains the default increment by which the "break" pointer for memory allocation will be advanced when there is no freed block large enough to satisfy a request to allocate a block of memory. This value may be changed by a program at any time.

Under 16-bit DOS, a new process started with one of the spawn... or exec... functions is loaded following the break value. Consequently, decreasing the break value leaves more space available to the new process. Similarly, for a resident program (a program which remains in memory while another program executes), increasing the break value will leave more space available to be allocated by the resident program after other programs are loaded.

Returns: If the call to sbrk succeeds, a pointer to the start of the new block of memory is returned. Under 16-bit DOS, this corresponds to the old break value. If the call to sbrk fails, -1 is returned. When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also: calloc Functions, _expand Functions, free Functions, halloc, hfree, malloc Functions, _msize Functions, realloc Functions

Example: #include <stdio.h>
#include <stdlib.h>

#if defined(M_I86)
#define alloc( x, y ) sbrk( x ); y = sbrk( 0 );
#else
#define alloc( x, y ) y = sbrk( x );
#endif

void main()
{
    void *brk;
#if defined(M_I86)
alloc( 0x0000, brk );
/* calling printf will cause an allocation */
printf( "Original break value %p\n", brk );
printf( "Current amblksiz value %x\n", _amblksiz );
alloc( 0x0000, brk );
printf( "New break value after printf \t\t%p\n", brk );
#endif
alloc( 0x3100, brk );
printf( "New break value after sbrk( 0x3100 ) \t%p\n", brk );
alloc( 0x0200, brk );
printf( "New break value after sbrk( 0x0200 ) \t%p\n", brk );
#if defined(M_I86)
alloc( -0x0100, brk );
printf( "New break value after sbrk( -0x0100 ) \t%p\n", brk );
#endif

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, QNX, OS/2 1.x, OS/2 1.x(MT), OS/2-32
**Synopsis:**

```c
#include <stdio.h>
int scanf( const char *format, ... );
#include <wchar.h>
int wscanf( const wchar_t *format, ... );
```

**Safer C:**

The Safer C Library extension provides the `scanf_s` function which is a safer alternative to `scanf`. This newer `scanf_s` function is recommended to be used instead of the traditional "unsafe" `scanf` function.

**Description:**

The `scanf` function scans input from the file designated by `stdin` under control of the argument `format`. The `format` string is described below. Following the format string is the list of addresses of items to receive values.

The `wscanf` function is identical to `scanf` except that it accepts a wide-character string argument for `format`.

**Returns:**

The `scanf` function returns `EOF` if an input failure occurred before any conversion. Otherwise, the number of input arguments for which values were successfully scanned and stored is returned.

**See Also:**
`cscanf`, `fscanf`, `sscanf`, `vcscanf`, `vfscanf`, `vscanf`, `vsscanf`

**Example:**

To scan a date in the form "Saturday April 18 1987":

```c
#include <stdio.h>

void main( void )
{
    int day, year;
    char weekday[10], month[10];

    scanf( "%s %s %d %d", weekday, month, &day, &year );
}
```

**Format Control String:**

The format control string consists of zero or more format directives that specify acceptable input file data. Subsequent arguments are pointers to various types of objects that are assigned values as the format string is processed.

A format directive can be a sequence of one or more white-space characters, an ordinary character, or a conversion specifier. An ordinary character in the format string is any character, other than a white-space character or the percent character (%), that is not part of a conversion specifier. A conversion specifier is a sequence of characters in the format string that begins with a percent character (%) and is followed, in sequence, by the following:

- an optional assignment suppression indicator: the asterisk character (*);
- an optional decimal integer that specifies the maximum field width to be scanned for the conversion;
- an optional pointer-type specification: one of "N" or "W";
- an optional type length specification: one of "hh", "h", "l", "ll", "j", "z", "t", "L" or "I64";
- a character that specifies the type of conversion to be performed: one of the characters "cCdeEfGinopsSuxX[".

---

812 Library Functions and Macros
As each format directive in the format string is processed, the directive may successfully complete, fail because of a lack of input data, or fail because of a matching error as defined by the particular directive. If end-of-file is encountered on the input data before any characters that match the current directive have been processed (other than leading white-space where permitted), the directive fails for lack of data. If end-of-file occurs after a matching character has been processed, the directive is completed (unless a matching error occurs), and the function returns without processing the next directive. If a directive fails because of an input character mismatch, the character is left unread in the input stream. Trailing white-space characters, including new-line characters, are not read unless matched by a directive. When a format directive fails, or the end of the format string is encountered, the scanning is completed and the function returns.

When one or more white-space characters (space " ", horizontal tab "\t", vertical tab "\v", form feed "\f", carriage return "\r", new line or linefeed "\n") occur in the format string, input data up to the first non-white-space character is read, or until no more data remains. If no white-space characters are found in the input data, the scanning is complete and the function returns.

An ordinary character in the format string is expected to match the same character in the input stream.

A conversion specifier in the format string is processed as follows:

- for conversion types other than "l", "c", "C" and "n", leading white-space characters are skipped
- for conversion types other than "n", all input characters, up to any specified maximum field length, that can be matched by the conversion type are read and converted to the appropriate type of value; the character immediately following the last character to be matched is left unread; if no characters are matched, the format directive fails
- unless the assignment suppression indicator ("*") was specified, the result of the conversion is assigned to the object pointed to by the next unused argument (if assignment suppression was specified, no argument is skipped); the arguments must correspond in number, type and order to the conversion specifiers in the format string

A pointer-type specification is used to indicate the type of pointer used to locate the next argument to be scanned:

- \textbf{W} pointer is a far pointer
- \textbf{N} pointer is a near pointer

The pointer-type specification is only effective on platforms that use a segmented memory model, although it is always recognized.

The pointer type defaults to that used for data in the memory model for which the program has been compiled.

A type length specifier affects the conversion as follows:

- "hh" causes a "d", "i", "o", "u" or "x" (integer) conversion to assign the converted value to an object of type \texttt{signed char} or \texttt{unsigned char}.
- "hh" causes an "n" (read length assignment) operation to assign the number of characters that have been read to an object of type \texttt{signed char}. 

\textbf{Library Functions and Macros} 813
"h" causes a "d", "i", "o", "u" or "x" (integer) conversion to assign the converted value to an object of type short int or unsigned short int.

"h" causes an "f" conversion to assign a fixed-point number to an object of type long consisting of a 16-bit signed integer part and a 16-bit unsigned fractional part. The integer part is in the high 16 bits and the fractional part is in the low 16 bits.

```
struct fixpt {
    unsigned short fraction; /* Intel architecture! */
    signed short integral;
};
```

```
struct fixpt foo1 =
    { 0x8000, 1234 }; /* represents 1234.5 */
struct fixpt foo2 =
    { 0x8000, -1 };   /* represents -0.5 (-1+.5) */
```

"h" causes an "n" (read length assignment) operation to assign the number of characters that have been read to an object of type short int.

"h" causes an "s" operation to convert the input string to an ASCII character string. For scanf, this specifier is redundant. For wscanf, this specifier is required if the wide character input string is to be converted to an ASCII character string; otherwise it will not be converted.

"l" causes a "d", "i", "o", "u" or "x" (integer) conversion to assign the converted value to an object of type long int or unsigned long int.

"l" causes an "n" (read length assignment) operation to assign the number of characters that have been read to an object of type long int.

"l" causes an "e", "f" or "g" (floating-point) conversion to assign the converted value to an object of type double.

"l" or "w" cause an "s" operation to convert the input string to a wide character string. For scanf, this specifier is required if the input ASCII string is to be converted to a wide character string; otherwise it will not be converted.

"ll" causes a "d", "i", "o", "u" or "x" (integer) conversion to assign the converted value to an object of type long long or unsigned long long (e.g., %lld).

"ll" causes an "n" (read length assignment) operation to assign the number of characters that have been read to an object of type long long int.

"j" causes a "d", "i", "o", "u" or "x" (integer) conversion to assign the converted value to an object of type intmax_t or uintmax_t.

"j" causes an "n" (read length assignment) operation to assign the number of characters that have been read to an object of type intmax_t.

"z" causes a "d", "i", "o", "u" or "x" (integer) conversion to assign the converted value to an object of type size_t or the corresponding signed integer type.

"z" causes an "n" (read length assignment) operation to assign the number of characters that have been read to an object of signed integer type corresponding to size_t.
• "t" causes a "d", "i", "o", "u" or "x" (integer) conversion to assign the converted value to an object of type `ptrdiff_t` or the corresponding unsigned integer type.

• "t" causes an "n" (read length assignment) operation to assign the number of characters that have been read to an object of type `ptrdiff_t`.

• "%64" causes a "d", "i", "o", "u" or "x" (integer) conversion to assign the converted value to an object of type `__int64` or `unsigned __int64` (e.g., `%I64d`).

• "L" causes an "e", "f" or "g" (floating-point) conversion to assign the converted value to an object of type `long double`.

The valid conversion type specifiers are:

- \texttt{c} Any sequence of characters in the input stream of the length specified by the field width, or a single character if no field width is specified, is matched. The argument is assumed to point to the first element of a character array of sufficient size to contain the sequence, without a terminating null character ('\texttt{\0}').
  For a single character assignment, a pointer to a single object of type `char` is sufficient.

- \texttt{C} A sequence of multibyte characters in the input stream is matched. Each multibyte character is converted to a wide character of type `wchar_t`. The number of wide characters matched is specified by the field width (1 if no field width is specified). The argument is assumed to point to the first element of an array of `wchar_t` of sufficient size to contain the sequence. No terminating null wide character (L'\texttt{\0}') is added. For a single wide character assignment, a pointer to a single object of type `wchar_t` is sufficient.

- \texttt{d} A decimal integer, consisting of an optional sign, followed by one or more decimal digits, is matched. The argument is assumed to point to an object of type `int`.

- \texttt{e, f, g} A floating-point number, consisting of an optional sign ("+" or "-"), followed by one or more decimal digits, optionally containing a decimal-point character, followed by an optional exponent of the form "e" or "E", an optional sign and one or more decimal digits, is matched. The exponent, if present, specifies the power of ten by which the decimal fraction is multiplied. The argument is assumed to point to an object of type `float`.

- \texttt{i} An optional sign, followed by an octal, decimal or hexadecimal constant is matched. An octal constant consists of "0" and zero or more octal digits. A decimal constant consists of a non-zero decimal digit and zero or more decimal digits. A hexadecimal constant consists of the characters "0x" or "0X" followed by one or more (upper- or lowercase) hexadecimal digits. The argument is assumed to point to an object of type `int`.

- \texttt{n} No input data is processed. Instead, the number of characters that have already been read is assigned to the object of type `unsigned int` that is pointed to by the argument. The number of items that have been scanned and assigned (the return value) is not affected by the "n" conversion type specifier.

- \texttt{o} An octal integer, consisting of an optional sign, followed by one or more (zero or non-zero) octal digits, is matched. The argument is assumed to point to an object of type `int`.

- \texttt{p} A hexadecimal integer, as described for "%x" conversions below, is matched. The converted value is further converted to a value of type `void*` and then assigned to the object pointed to by the argument.
A sequence of non-white-space characters is matched. The argument is assumed to point to the first element of a character array of sufficient size to contain the sequence and a terminating null character, which is added by the conversion operation.

A sequence of multibyte characters is matched. None of the multibyte characters in the sequence may be single byte white-space characters. Each multibyte character is converted to a wide character. The argument is assumed to point to the first element of an array of \texttt{wchar_t} of sufficient size to contain the sequence and a terminating null wide character, which is added by the conversion operation.

An unsigned decimal integer, consisting of one or more decimal digits, is matched. The argument is assumed to point to an object of type \texttt{unsigned int}.

A hexadecimal integer, consisting of an optional sign, followed by an optional prefix "0x" or "0X", followed by one or more (upper- or lowercase) hexadecimal digits, is matched. The argument is assumed to point to an object of type \texttt{int}.

The longest, non-empty sequence of characters, consisting of any of the characters \texttt{c1, c2, ...} called the \texttt{scanset}, in any order, is matched. \texttt{c1} cannot be the caret character ('^'). If \texttt{c1} is "]", that character is considered to be part of the scanset and a second "]" is required to end the format directive. The argument is assumed to point to the first element of a character array of sufficient size to contain the sequence and a terminating null character, which is added by the conversion operation.

The longest, non-empty sequence of characters, consisting of any characters \texttt{other than} the characters between the "^" and "]", is matched. As with the preceding conversion, if \texttt{c1} is "]", it is considered to be part of the scanset and a second "]" ends the format directive. The argument is assumed to point to the first element of a character array of sufficient size to contain the sequence and a terminating null character, which is added by the conversion operation.

For example, the specification \texttt{%[^\n]} will match an entire input line up to but not including the newline character.

A conversion type specifier of \texttt{"%"} is treated as a single ordinary character that matches a single "\texttt{\%}" character in the input data. A conversion type specifier other than those listed above causes scanning to terminate and the function to return.

Conversion type specifiers \texttt{"E", "F", "G", "X"} have meaning identical to their lowercase equivalents.

The line

\begin{verbatim}
scanf( "%s%f%hx%d", name, &hexnum, &decnum )
\end{verbatim}

with input

\begin{verbatim}
some_string 34.555e-3 abc1234
\end{verbatim}

will copy "some_string" into the array name, skip 34.555e-3, assign 0xabc to hexnum and 1234 to decnum. The return value will be 3.
```c
#include <stdio.h>

void main( void )
{
    char string1[80], string2[80];
    scanf( "%[abcdefghijklmnopqrstuvwxyzABCDEFHIJKLMNOPQRSTUVWZ ]%*2s%^\n",
            string1, string2 );
    printf( "%s\n%s\n", string1, string2 );
}
```

with input

They may look alike, but they don’t perform alike.

will assign

"They may look alike"

to string1, skip the comma (the "%*2s" will match only the comma; the following blank terminates that field), and assign

" but they don’t perform alike."

to string2.

**Classification:** scanf is ISO C90

wscanf is ISO C95

The N, W pointer size modifiers and the I64 modifier are extensions to ISO C.

**Systems:**

scanf – All, Netware

wscanf – All
Synopsis:  
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
int scanf_s( const char * restrict format, ... );
#include <wchar.h>
int wscanf_s( const wchar_t * restrict format, ... );

Constraints:  If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and scanf_s will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

The format argument shall not be a null pointer. Any argument indirected through in order to store converted input shall not be a null pointer.

If there is a runtime-constraint violation, the scanf_s function does not attempt to perform further input, and it is unspecified to what extent scanf_s performed input before discovering the runtime-constraint violation.

Description:  The scanf_s function is equivalent to fscanf_s with the argument stdin interposed before the arguments to scanf_s

The wscanf_s function is identical to scanf_s except that it accepts a wide-character string argument for format.

Returns:  The scanf_s function returns EOF if an input failure occurred before any conversion or if there was a runtime-constraint violation. Otherwise, the scanf_s function returns the number of input items successfully assigned, which can be fewer than provided for, or even zero.

When a file input error occurs, the errno global variable may be set.

See Also:  cscansf, fscansf, scansf, sscansf, vcscansf, vfscansf, vscansf, vsscansf

Example:  To scan a date in the form "Friday August 13 2004":

#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>

void main( void )
{
    int day, year;
    char weekday[10], month[10];

    scanf_s( "%s %s %d %d",
        weekday, sizeof( weekday ),
        month, sizeof( month ),
        &day, &year );
}

Classification:  scanf_s is TR 24731
wscanf_s is TR 24731

Systems:  scanf_s - All, Netware
wscanf_s - All
Synopsis:  
#include <graph.h>  
void _FAR _scrolltextwindow( short rows );

Description:  The _scrolltextwindow function scrolls the lines in the current text window. A text window is defined with the _settextwindow function. By default, the text window is the entire screen.

The argument rows specifies the number of rows to scroll. A positive value means to scroll the text window up or towards the top of the screen. A negative value means to scroll the text window down or towards the bottom of the screen. Specifying a number of rows greater than the height of the text window is equivalent to clearing the text window with the _clearscreen function.

Two constants are defined that can be used with the _scrolltextwindow function:

_GSCROLLUP  the contents of the text window are scrolled up (towards the top of the screen) by one row

_GSCROLLDOWN the contents of the text window are scrolled down (towards the bottom of the screen) by one row

Returns:  The _scrolltextwindow function does not return a value.

See Also:  _settextwindow, _clearscreen, _outtext, _outmem, _settextposition

Example:  
#include <conio.h>  
#include <graph.h>  
#include <stdio.h>  

main()  
{  
    int i;  
    char buf[ 80 ];
    _setvideomode( _TEXTC80 );
    _settextwindow( 5, 20, 20, 40 );
    for( i = 1; i <= 10; ++i )  
        sprintf( buf, "Line %d\n", i );
        _outtext( buf );
    }  
    getch();  
    _scrolltextwindow( _GSCROLLDOWN );
    getch();
    _scrolltextwindow( _GSCROLLUP );
    getch();
    _setvideomode( _DEFAULTMODE );
}

Classification: _scrolltextwindow is PC Graphics

Systems:  DOS, QNX
Synopsis:  
#include <stdlib.h>  
void _searchenv( const char *name,  
    const char *env_var,  
    char *pathname );  
void _wsearchenv( const wchar_t *name,  
    const wchar_t *env_var,  
    wchar_t *pathname );

Description:  
The _searchenv function searches for the file specified by name in the list of directories assigned to the environment variable specified by env_var. Common values for env_var are PATH, LIB and INCLUDE.

The current directory is searched first to find the specified file. If the file is not found in the current directory, each of the directories specified by the environment variable is searched.

The full pathname is placed in the buffer pointed to by the argument pathname. If the specified file cannot be found, then pathname will contain an empty string.

The _wsearchenv function is a wide-character version of _searchenv that operates with wide-character strings.

Returns:  
The _searchenv function returns no value.

See Also:  
getenv, setenv, _splitpath, putenv

Example:  
#include <stdio.h>  
#include <stdlib.h>

void display_help( FILE *fp )  
{  
    printf( "display_help T.B.I.\n" );  
}

void main()  
{  
    FILE *help_file;  
    char full_path[ _MAX_PATH ];  

    _searchenv( "watcomc.hlp", "PATH", full_path );  
    if( full_path[0] == '\0' ) {  
        printf( "Unable to find help file\n" );  
    } else {  
        help_file = fopen( full_path, "r" );  
        display_help( help_file );  
        fclose( help_file );  
    }  
}

Classification: WATCOM

Systems:  
_searchenv - All  
_wsearchenv - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <i86.h>
void segread( struct SREGS *seg_regs );

Description:  The segread function places the values of the segment registers into the structure located by seg_regs.

Returns:  No value is returned.

See Also:  FP_OFF, FP_SEG, MK_FP

Example:  
#include <stdio.h>
#include <i86.h>

void main()
{
    struct SREGS sregs;
    segread( &sregs );
    printf( "Current value of CS is %04X\n", sregs.cs );
}

Classification:  WATCOM

Systems:  All, Netware
Synopsis:  
#include <graph.h>  
short _FAR _selectpalette( short palnum );

Description:  The _selectpalette function selects the palette indicated by the argument palnum from the color palettes available. This function is only supported by the video modes _MRES4COLOR and _MRESNOCOLOR.

Mode _MRES4COLOR supports four palettes of four colors. In each palette, color 0, the background color, can be any of the 16 possible colors. The color values associated with the other three pixel values, (1, 2 and 3), are determined by the selected palette.

The following table outlines the available color palettes:

<table>
<thead>
<tr>
<th>Palette Number</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>green</td>
<td>red</td>
<td>brown</td>
</tr>
<tr>
<td>1</td>
<td>cyan</td>
<td>magenta</td>
<td>white</td>
</tr>
<tr>
<td>2</td>
<td>light green</td>
<td>light red</td>
<td>yellow</td>
</tr>
<tr>
<td>3</td>
<td>light cyan</td>
<td>light magenta</td>
<td>bright white</td>
</tr>
</tbody>
</table>

Returns:  The _selectpalette function returns the number of the previously selected palette.

See Also:  _setvideomode, _getvideoconfig

Example:  
#include <conio.h>  
#include <graph.h>

main()
{
  int x, y, pal;
  _setvideomode( _MRES4COLOR );
  for( y = 0; y < 2; ++y ) {
    for( x = 0; x < 2; ++x ) {
      _setcolor( x + 2 * y );
      _rectangle( _GFILLINTERIOR,
                   x * 160, y * 100,
                   ( x + 1 ) * 160, ( y + 1 ) * 100 );
    }
  }
  for( pal = 0; pal < 4; ++pal ) {
    _selectpalette( pal );
    getch();
  }
  _setvideomode( _DEFAULTMODE );
}

Classification:  PC Graphics

Systems:  DOS, QNX
Synopsis:

```c
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
constraint_handler_t set_constraint_handler_s(
    constraint_handler_t handler );
```

Description:

The `set_constraint_handler_s` function sets the runtime-constraint handler to be `handler`. The runtime-constraint handler is the function called when a library function detects a runtime-constraint violation. Only the most recent handler registered with `set_constraint_handler_s` is called when a runtime-constraint violation occurs.

When the handler is called, it is passed the following arguments:

1. A pointer to a character string describing the runtime-constraint violation.
2. A null pointer or a pointer to an implementation defined object. This implementation passes a null pointer.
3. If the function calling the handler has a return type declared as `errno_t`, the return value of the function is passed. Otherwise, a positive value of type `errno_t` is passed.

If no calls to the `set_constraint_handler_s` function have been made, a default constraint handler is used. This handler will display an error message and abort the program.

If the `handler` argument to `set_constraint_handler_s` is a null pointer, the default handler becomes the current constraint handler.

Returns:

The `set_constraint_handler_s` function returns a pointer to the previously registered handler.

See Also:

`abort_handler_s`, `ignore_handler_s`

Example:

```c
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
#include <stdio.h>

void my_handler( const char *msg, void *ptr, errno_t error )
{
    fprintf( stderr, "rt-constraint violation caught: ");
    fprintf( stderr, msg );
    fprintf( stderr, "\n" );
}

void main( void )
{
    constraint_handler_t   old_handler;

    old_handler = set_constraint_handler_s( my_handler );
    if( getenv_s( NULL, NULL, 0, NULL ) )
    {
        printf( "getenv_s failed\n" );
    }
    set_constraint_handler_s( old_handler );
}
```

produces the following:
rt-constraint violation caught: getenv_s, name == NULL.
getenv_s failed

Classification: TR 24731

Systems: All, Netware
Synopsis:  
#include <graph.h>
short _FAR _setactivepage( short pagenum );

Description:  The _setactivepage function selects the page (in memory) to which graphics output is written.  The page to be selected is given by the pagenum argument.

Only some combinations of video modes and hardware allow multiple pages of graphics to exist. When multiple pages are supported, the active page may differ from the visual page. The graphics information in the visual page determines what is displayed upon the screen. Animation may be accomplished by alternating the visual page. A graphics page can be constructed without affecting the screen by setting the active page to be different than the visual page.

The number of available video pages can be determined by using the _getvideoconfig function. The default video page is 0.

Returns:  The _setactivepage function returns the number of the previous page when the active page is set successfully; otherwise, a negative number is returned.

See Also:  _getactivepage, _setvisualpage, _getvisualpage, _getvideoconfig

Example:  
#include <conio.h>
#include <graph.h>

main()
{
  int old_apage;
  int old_vpage;
  
  _setvideomode( _HRES16COLOR );
  old_apage = _getactivepage();
  old_vpage = _getvisualpage();
  /* draw an ellipse on page 0 */
  _setactivepage( 0 );
  _setvisualpage( 0 );
  _ellipse( _GFILLINTERIOR, 100, 50, 540, 150 );
  /* draw a rectangle on page 1 */
  _setactivepage( 1 );
  _rectangle( _GFILLINTERIOR, 100, 50, 540, 150 );
  getch();
  /* display page 1 */
  _setvisualpage( 1 );
  getch();
  _setactivepage( old_apage );
  _setvisualpage( old_vpage );
  _setvideomode( _DEFAULTMODE );
}

Classification:  PC Graphics

Systems:  DOS, QNX
Synopsis:  
#include <graph.h>
long _FAR _setbkcolor( long color );

Description:  The _setbkcolor function sets the current background color to be that of the color argument. In text modes, the background color controls the area behind each individual character. In graphics modes, the background refers to the entire screen. The default background color is 0.

When the current video mode is a graphics mode, any pixels with a zero pixel value will change to the color of the color argument. When the current video mode is a text mode, nothing will immediately change; only subsequent output is affected.

Returns:  The _setbkcolor function returns the previous background color.

See Also:  _getbkcolor

Example:  #include <conio.h>
#include <graph.h>

long colors[ 16 ] = {
    _BLACK, _BLUE, _GREEN, _CYAN,
    _RED, _MAGENTA, _BROWN, _WHITE,
    _GRAY, _LIGHTBLUE, _LIGHTGREEN, _LIGHTCYAN,
    _LIGHTRED, _LIGHTMAGENTA, _YELLOW, _BRIGHTWHITE
};

main()
{
    long old_bk;
    int bk;

    _setvideomode( _VRES16COLOR );
    old_bk = _getbkcolor();
    for( bk = 0; bk < 16; ++bk ) {
        _setbkcolor( colors[ bk ] );
        getch();
    }
    _setbkcolor( old_bk );
    _setvideomode( _DEFAULTMODE );
}

Classification:  PC Graphics

Systems:  DOS, QNX
Synopsis:  
#include <stdio.h>
void setbuf( FILE *fp, char *buffer );

Description:  The setbuf function can be used to associate a buffer with the file designated by fp. If this function is used, it must be called after the file has been opened and before it has been read or written. If the argument buffer is NULL, then all input/output for the file fp will be completely unbuffered. If the argument buffer is not NULL, then it must point to an array that is at least BUFSIZ characters in length, and all input/output will be fully buffered.

Returns:  The setbuf function returns no value.

See Also:  fopen, setvbuf

Example:  
#include <stdio.h>
#include <stdlib.h>

void main()
{
  char *buffer;
  FILE *fp;

  fp = fopen( "file", "r" );
  buffer = (char *) malloc( BUFSIZ );
  setbuf( fp, buffer );
  /* . */
  /* . */
  /* . */
  fclose( fp );
}

Classification:  ANSI

Systems:  All, Netware
Synopsis:  #include <graph.h>
void _FAR _setcharsize( short height, short width );

void _FAR _setcharsize_w( double height, double width );

Description:  The _setcharsize functions set the character height and width to the values specified by the arguments height and width. For the _setcharsize function, the arguments height and width represent a number of pixels. For the _setcharsize_w function, the arguments height and width represent lengths along the y-axis and x-axis in the window coordinate system.

These sizes are used when displaying text with the _grtext function. The default character sizes are dependent on the graphics mode selected, and can be determined by the _gettextsettings function.

Returns:  The _setcharsize functions do not return a value.

See Also:  _grtext, _gettextsettings

Example:  #include <conio.h>
#include <graph.h>

main()
{
    struct textsettings ts;
    _setvideomode( _VRES16COLOR );
    _gettextsettings( &ts );
    _grtext( 100, 100, "WATCOM" );
    _setcharsize( 2 * ts.height, 2 * ts.width );
    _grtext( 100, 300, "Graphics" );
    _setcharsize( ts.height, ts.width );
    getch();
    _setvideomode( _DEFAULTMODE );
}

produces the following:
Classification: PC Graphics

Systems:

- _setcharsize - DOS, QNX
- _setcharsize_w - DOS, QNX
_setcharspacing Functions

Synopsis:  
#include <graph.h>
void _FAR _setcharspacing( short space );

void _FAR _setcharspacing_w( double space );

Description:  The _setcharspacing functions set the current character spacing to have the value of the argument space.  For the _setcharspacing function, space represents a number of pixels.  For the _setcharspacing_w function, space represents a length along the x-axis in the window coordinate system.

The character spacing specifies the additional space to leave between characters when a text string is displayed with the _grtext function.  A negative value can be specified to cause the characters to be drawn closer together.  The default value of the character spacing is 0.

Returns:  The _setcharspacing functions do not return a value.

See Also:  _grtext, _gettextsettings

Example:  
#include <conio.h>
#include <graph.h>

main()
{
   _setvideomode( _VRES16COLOR );
   _grtext( 100, 100, "WATCOM" );
   _setcharspacing( 20 );
   _grtext( 100, 300, "Graphics" );
   getch();
   _setvideomode( _DEFAULTMODE );
}

produces the following:
Classification: PC Graphics

Systems:  
_setcharspacing - DOS, QNX  
_setcharspacing_w - DOS, QNX
Synopsis: 
#include <graph.h>
void _FAR _setcliprgn( short x1, short y1,
                        short x2, short y2 );

Description: The _setcliprgn function restricts the display of graphics output to the clipping region. This region is a rectangle whose opposite corners are established by the physical points \((x1, y1)\) and \((x2, y2)\).

The _setcliprgn function does not affect text output using the _outtext and _outmem functions. To control the location of text output, see the _settextwindow function.

Returns: The _setcliprgn function does not return a value.

See Also: _settextwindow, _setvieworg, _setviewport

Example: #include <conio.h>
#define include <graph.h>

main()
{
    short x1, y1, x2, y2;
    _setvideomode( _VRES16COLOR );
    _getcliprgn( &x1, &y1, &x2, &y2 );
    _setcliprgn( 130, 100, 510, 380 );
    _ellipse( _GBORDER, 120, 90, 520, 390 );
    getch();
    _setcliprgn( x1, y1, x2, y2 );
    _setvideomode( _DEFAULTMODE );
}

Classification: PC Graphics

Systems: DOS, QNX
Synopsis: #include <graph.h>
short _FAR _setcolor( short pixval );

Description: The _setcolor function sets the pixel value for the current color to be that indicated by the pixval argument. The current color is only used by the functions that produce graphics output; text output with _outtext uses the current text color (see the _settextcolor function). The default color value is one less than the maximum number of colors in the current video mode.

Returns: The _setcolor function returns the previous value of the current color.

See Also: _getcolor, _settextcolor

Example: #include <conio.h>
#include <graph.h>

main()
{
    int col, old_col;
    _setvideomode( _VRES16COLOR );
    old_col = _getcolor();
    for( col = 0; col < 16; ++col ) {
        _setcolor( col );
        _rectangle( _GFILLINTERIOR, 100, 100, 540, 380 );
        getch();
    }
    _setcolor( old_col );
    _setvideomode( _DEFAULTMODE );
}

Classification: PC Graphics

Systems: DOS, QNX
**setenv, _setenv, _wsetenv**

**Synopsis:**

```c
#include <env.h>
int setenv( const char *name,
            const char *newvalue,
            int overwrite );
int _setenv( const char *name,
            const char *newvalue,
            int overwrite );
int _wsetenv( const wchar_t *name,
             const wchar_t *newvalue,
            int overwrite );
```

**Description:**

The environment list consists of a number of environment names, each of which has a value associated with it. Entries can be added to the environment list with the DOS `set` command or with the `setenv` function. All entries in the environment list can be displayed by using the DOS `set` command with no arguments. A program can obtain the value for an environment variable by using the `getenv` function.

The `setenv` function searches the environment list for an entry of the form `name=value`. If no such string is present, `setenv` adds an entry of the form `name=newvalue` to the environment list. Otherwise, if the `overwrite` argument is non-zero, `setenv` either will change the existing value to `newvalue` or will delete the string `name=value` and add the string `name=newvalue`.

If the `newvalue` pointer is NULL, all strings of the form `name=value` in the environment list will be deleted.

The value of the pointer `environ` may change across a call to the `setenv` function.

The `setenv` function will make copies of the strings associated with `name` and `newvalue`.

The matching is case-insensitive; all lowercase letters are treated as if they were in upper case.

Entries can also be added to the environment list with the DOS `set` command or with the `putenv` or `setenv` functions. All entries in the environment list can be obtained by using the `getenv` function.

To assign a string to a variable and place it in the environment list:

```bash
C>SET INCLUDE=C:\WATCOM\H
```

To see what variables are in the environment list, and their current assignments:

```bash
C>SET
COMSPEC=C:\COMMAND.COM
PATH=C:\;C:\WATCOM
INCLUDE=C:\WATCOM\H
C>
```

The `_setenv` function is identical to `setenv`. Use `_setenv` for ANSI naming conventions.

The `_wsetenv` function is a wide-character version of `setenv` that operates with wide-character strings.

**Returns:**

The `setenv` function returns zero upon successful completion. Otherwise, it will return a non-zero value and set `errno` to indicate the error.
setenv, _setenv, _wsetenv

Errors: When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

`ENOMEM` Not enough memory to allocate a new environment string.

See Also: `clearenv`, `exec...`, `getenv`, `getenv_s`, `putenv`, `_searchenv`, `spawn...`, `system`

Example: The following will change the string assigned to `INCLUDE` and then display the new string.

```c
#include <stdio.h>
#include <stdlib.h>
#include <env.h>

void main()
{
    char *path;

    if( setenv( "INCLUDE", "D:\WATCOM\H", 1 ) == 0 )
        if( (path = getenv( "INCLUDE" )) != NULL )
            printf( "INCLUDE=%s\n", path );
}
```

Classification: WATCOM

Systems: `setenv` - All
        `_setenv` - All
        `_wsetenv` - All
Synopsis: #include <graph.h>
void _FAR _setfillmask( char _FAR *mask );

Description: The _setfillmask function sets the current fill mask to the value of the argument mask. When the value of the mask argument is NULL, there will be no fill mask set.

The fill mask is an eight-byte array which is interpreted as a square pattern (8 by 8) of 64 bits. Each bit in the mask corresponds to a pixel. When a region is filled, each point in the region is mapped onto the fill mask. When a bit from the mask is one, the pixel value of the corresponding point is set using the current plotting action with the current color; when the bit is zero, the pixel value of that point is not affected.

When the fill mask is not set, a fill operation will set all points in the fill region to have a pixel value of the current color. By default, no fill mask is set.

Returns: The _setfillmask function does not return a value.

See Also: _getfillmask, _ellipse, _floodfill, _rectangle, _polygon, _pie, _setcolor, _setplotaction

Example: #include <conio.h>
#include <graph.h>

char old_mask[ 8 ];
char new_mask[ 8 ] = { 0x81, 0x42, 0x24, 0x18,
                      0x18, 0x24, 0x42, 0x81 };

main()
{
   _setvideomode( _VRES16COLOR );
   _getfillmask( old_mask );
   _setfillmask( new_mask );
   _rectangle( _GFILLINTERIOR, 100, 100, 540, 380 );
   _setfillmask( old_mask );
   getch();
   _setvideomode( _DEFAULTMODE );
}

produces the following:
Classification: _setfillmask is PC Graphics

Systems: DOS, QNX
Synopsis:  
#include <graph.h>
short _FAR _setfont( char _FAR *opt );

Description:  The _setfont function selects a font from the list of registered fonts (see the _registerfonts function). The font selected becomes the current font and is used whenever text is displayed with the _outgtext function. The function will fail if no fonts have been registered, or if a font cannot be found that matches the given characteristics.

The argument opt is a string of characters specifying the characteristics of the desired font. These characteristics determine which font is selected. The options may be separated by blanks and are not case-sensitive. Any number of options may be specified and in any order. The available options are:

- **hX** character height X (in pixels)
- **wX** character width X (in pixels)
- **f** choose a fixed-width font
- **p** choose a proportional-width font
- **r** choose a raster (bit-mapped) font
- **v** choose a vector font
- **b** choose the font that best matches the options
- **nX** choose font number X (the number of fonts is returned by the _registerfonts function)
- **'t'facename'** choose a font with specified facename

The facename option is specified as a "t" followed by a facename enclosed in single quotes. The available facenames are:

- **Courier** fixed-width raster font with serifs
- **Helv** proportional-width raster font without serifs
- **Tms Rmn** proportional-width raster font with serifs
- **Script** proportional-width vector font that appears similar to hand-writing
- **Modern** proportional-width vector font without serifs
- **Roman** proportional-width vector font with serifs

When "nX" is specified to select a particular font, the other options are ignored.

If the best fit option ("b") is specified, _setfont will always be able to select a font. The font chosen will be the one that best matches the options specified. The following precedence is given to the options when selecting a font:

1. Pixel height (higher precedence is given to heights less than the specified height)
2. Facename

3. Pixel width

4. Font type (fixed or proportional)

When a pixel height or width does not match exactly and a vector font has been selected, the font will be stretched appropriately to match the given size.

Returns: The _setfont function returns zero if successful; otherwise, (-1) is returned.

See Also: _registerfonts, _unregisterfonts, _getfontinfo, _outgtext, _getgtextextent, _setgtextvector, _getgtextvector

Example:
```c
#include <conio.h>
#include <stdio.h>
#include <graph.h>

main()
{
    int i, n;
    char buf[10];

    _setvideomode(_VRES16COLOR);
    n = _registerfonts("*.fon");
    for ( i = 0; i < n; ++i ) {
        sprintf(buf, "n%d", i);
        _setfont(buf);
        _moveto(100, 100);
        _outgtext("WATCOM Graphics");
        getch();
        _clearscreen(_GCLEARSCREEN);
    }
    _unregisterfonts();
    _setvideomode(_DEFAULTMODE);
}
```

Classification: PC Graphics

Systems: DOS, QNX
Synopsis:  
#include <graph.h>
struct xycoord _FAR _setgtextvector( short x, short y );

Description:  The _setgtextvector function sets the orientation for text output used by the _outgtext
function to the vector specified by the arguments (x,y). Each of the arguments can have a value of
-1, 0 or 1, allowing for text to be displayed at any multiple of a 45-degree angle. The default text
orientation, for normal left-to-right text, is the vector (1,0).

Returns:  The _setgtextvector function returns, as an xycoord structure, the previous value of the text
orientation vector.

See Also:  _registerfonts, _unregisterfonts, _setfont, _getfontinfo, _outgtext,
_getgtextextent, _getgtextvector

Example:  
#include <conio.h>
#include <graph.h>

main()
{
    struct xycoord old_vec;

    _setvideomode( _VRES16COLOR );
    old_vec = _getgtextvector();
    _setgtextvector( 0, -1 );
    _moveto( 100, 100 );
    _outgtext( "WATCOM Graphics" );
    _setgtextvector( old_vec.xcoord, old_vec.ycoord );
    getch();
    _setvideomode( _DEFAULTMODE );
}

Classification:  PC Graphics

Systems:  DOS, QNX
Synopsis:  
#include <setjmp.h>
int setjmp(jmp_buf env);

Description:  The setjmp function saves its calling environment in its jmp_buf argument, for subsequent use by the longjmp function.

In some cases, error handling can be implemented by using setjmp to record the point to which a return will occur following an error. When an error is detected in a called function, that function uses longjmp to jump back to the recorded position. The original function which called setjmp must still be active (it cannot have returned to the function which called it).

Special care must be exercised to ensure that any side effects that are left undone (allocated memory, opened files, etc.) are satisfactorily handled.

Returns:  The setjmp function returns zero when it is initially called. The return value will be non-zero if the return is the result of a call to the longjmp function. An if statement is often used to handle these two returns. When the return value is zero, the initial call to setjmp has been made; when the return value is non-zero, a return from a longjmp has just occurred.

See Also:  longjmp

Example:  
#include <stdio.h>
#include <setjmp.h>

jmp_buf env;

rtn()
{
    printf( "about to longjmp\n" );
    longjmp( env, 14 );
}

void main()
{
    int ret_val = 293;

    if( 0 == ( ret_val = setjmp( env ) ) ) {
        printf( "after setjmp %d\n", ret_val );
        rtn();
        printf( "back from rtn %d\n", ret_val );
    } else {
        printf( "back from longjmp %d\n", ret_val );
    }
}

produces the following:

after setjmp 0
about to longjmp
back from longjmp 14

Classification:  ANSI

Systems:  MACRO
_setlinestyle

Synopsis: #include <graph.h>
void _FAR _setlinestyle( unsigned short style );

Description: The _setlinestyle function sets the current line-style mask to the value of the style argument.

The line-style mask determines the style by which lines and arcs are drawn. The mask is treated as an array of 16 bits. As a line is drawn, a pixel at a time, the bits in this array are cyclically tested. When a bit in the array is 1, the pixel value for the current point is set using the current color according to the current plotting action; otherwise, the pixel value for the point is left unchanged. A solid line would result from a value of 0xFFFF and a dashed line would result from a value of 0xF0F0.

The default line style mask is 0xFFFF.

Returns: The _setlinestyle function does not return a value.

See Also: _getlinestyle, _lineto, _rectangle, _polygon, _setplotaction

Example: #include <conio.h>
#include <graph.h>
#define DASHED 0xf0f0

main()
{
    unsigned old_style;
    _setvideomode( _VRES16COLOR );
    old_style = _getlinestyle();
    _setlinestyle( DASHED );
    _rectangle( _GBORDER, 100, 100, 540, 380 );
    _setlinestyle( old_style );
    getch();
    _setvideomode( _DEFAULTMODE );
}

produces the following:
Classification: PC Graphics

Systems: DOS, QNX
setlocale, _wsetlocale

Synopsis:    #include <locale.h>
char *setlocale(int category, const char *locale);
wchar_t *__wsetlocale(int category, const wchar_t *locale);

Description:  The setlocale function selects a portion of a program’s locale according to the category given by
category and the locale specified by locale. A locale affects the collating sequence (the order in which
characters compare with one another), the way in which certain character-handling functions operate,
the decimal-point character that is used in formatted input/output and string conversion, and the format
and names used in the time string produced by the strftime function.

Potentially, there may be many such environments. Watcom C/C++ supports only the "C" locale and
so invoking this function will have no effect upon the behavior of a program at present.

The possible values for the argument category are as follows:

<table>
<thead>
<tr>
<th>Category</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>LC_ALL</td>
<td>select entire environment</td>
</tr>
<tr>
<td>LC_COLLATE</td>
<td>select collating sequence</td>
</tr>
<tr>
<td>LC_CTYPE</td>
<td>select the character-handling</td>
</tr>
<tr>
<td>LC_MONETARY</td>
<td>select monetary formatting information</td>
</tr>
<tr>
<td>LC_NUMERIC</td>
<td>select the numeric-format environment</td>
</tr>
<tr>
<td>LC_TIME</td>
<td>select the time-related environment</td>
</tr>
</tbody>
</table>

At the start of a program, the equivalent of the following statement is executed.

    setlocale( LC_ALL, "C" );

The _wsetlocale function is a wide-character version of setlocale that operates with
wide-character strings.

Returns:  If the selection is successful, a string is returned to indicate the locale that was in effect before the
function was invoked; otherwise, a NULL pointer is returned.

See Also: strcoll, strftime, strxfrm

Example:    #include <stdio.h>
#include <string.h>
#include <locale.h>

    char src[] = { "A sample STRING" };
    char dst[20];

    void main()
    {
        char *prev_locale;
        size_t len;

844  Library Functions and Macros
/* set native locale */
prev_locale = setlocale( LC_ALL, "" );
printf( "%s\n", prev_locale );
len = strxfrm( dst, src, 20 );
printf( "%s (%u)\n", dst, len );
}

produces the following:

C
A sample STRING (15)

Classification: setlocale is ANSI, POSIX 1003.1
_wsetlocale is not ANSI

Systems: setlocale - All, Netware
_wsetlocale - All
Synopsis:  
#include <math.h>

void _set_matherr( int (*rtn)( struct _exception *err_info ) )

Description:  The default matherr function supplied in the library can be replaced so that the application can handle mathematical errors. To do this, the _set_matherr function must be called with the address of the new mathematical error handling routine.

Note: Under some systems, the default math error handler can be replaced by providing a user-written function of the same name, matherr, and using linking strategies to replace the default handler. Under PenPoint, the default handler is bound into a dynamic link library and can only be replaced by notifying the C library with a call to the _set_matherr function.

A program may contain a user-written version of matherr to take any appropriate action when an error is detected. When zero is returned by the user-written routine, an error message will be printed upon stderr and errno will be set as was the case with the default function. When a non-zero value is returned, no message is printed and errno is not changed. The value err_info->retval is used as the return value for the function in which the error was detected.

When called, the user-written math error handler is passed a pointer to a structure of type struct _exception which contains information about the error that has been detected:

```
struct _exception
{ int type;     /* TYPE OF ERROR */
  char *name;   /* NAME OF FUNCTION */
  double arg1;  /* FIRST ARGUMENT TO FUNCTION */
  double arg2;  /* SECOND ARGUMENT TO FUNCTION */
  double retval; /* DEFAULT RETURN VALUE */
};
```

The type field will contain one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOMAIN</td>
<td>A domain error has occurred, such as sqrt(-1e0).</td>
</tr>
<tr>
<td>SING</td>
<td>A singularity will result, such as pow(0e0,-2).</td>
</tr>
<tr>
<td>OVERFLOW</td>
<td>An overflow will result, such as pow(10e0,100).</td>
</tr>
<tr>
<td>UNDERFLOW</td>
<td>An underflow will result, such as pow(10e0,-100).</td>
</tr>
<tr>
<td>TLOSS</td>
<td>Total loss of significance will result, such as exp(1000).</td>
</tr>
<tr>
<td>PLOSS</td>
<td>Partial loss of significance will result, such as sin(10e70).</td>
</tr>
</tbody>
</table>

The name field points to a string containing the name of the function which detected the error. The fields arg1 and arg2 (if required) give the values which caused the error. The field retval contains the value which will be returned by the function. This value may be changed by a user-supplied version of the _set_matherr function.

Returns:  The _set_matherr function returns no value.
Example:
#include <stdio.h>
#include <string.h>
#include <math.h>

/* Demonstrate error routine in which negative */
/* arguments to "sqrt" are treated as positive */

int my_matherr( struct _exception *err )
{
    if( strcmp( err->name, "sqrt" ) == 0 ) {
        if( err->type == DOMAIN ) {
            err->retval = sqrt( -(err->arg1) );
            return( 1 );
        } else
            return( 0 );
    } else
        return( 0 );
}

void main( void )
{
    _set_matherr( &my_matherr );
    printf( "%e\n", sqrt( -5e0 ) );
    exit( 0 );
}

Classification: WATCOM
Systems: Math
Synopsis: 
#include <mbctype.h>
int _setmbcp( int codepage );

Description: The _setmbcp function sets the current code page number.

Returns: The _setmbcp function returns zero if the code page is set successfully. If an invalid code page value is supplied for codepage, the function returns -1 and the code page setting is unchanged.

See Also: _getmbcp, _mbbtombc, _mbcjmstojis, _mbcjmstojms, _mbctombb, _ismbbalnum, _ismbbalpha, _ismbbgraph, _ismbbkalnum, _ismbbkalpha, _ismbbkana, _ismbbkprint, _ismbbkpunct, _ismbblead, _ismbbprint, _ismbbpunct, _ismbbtrail, _mbbtombc, _mbcjmstojis, _mbcjmstojms, _mbctombb, _mbctype

Example: #include <stdio.h>
#include <mbctype.h>

void main()
{
    printf( "%d\n", _setmbcp( 932 ) );
    printf( "%d\n", _getmbcp() );
}

produces the following:

0
932

Classification: WATCOM

Systems: DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
setmode, _setmode

Synopsis:    #include <io.h>
#include <fcntl.h>
int setmode( int handle, int mode );
int _setmode( int handle, int mode );

Description: The setmode function sets, at the operating system level, the translation mode to be the value of mode for the file whose file handle is given by handle. The mode, defined in the <fcntl.h> header file, can be one of:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>O_TEXT</td>
<td>On input, a carriage-return character that immediately precedes a linefeed character is removed from the data that is read. On output, a carriage-return character is inserted before each linefeed character.</td>
</tr>
<tr>
<td>O_BINARY</td>
<td>Data is read or written unchanged.</td>
</tr>
</tbody>
</table>

Returns: If successful, the setmode function returns the previous mode that was set for the file; otherwise, -1 is returned. When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also: chsize, close, creat, dup, dup2, eof, exec..., fdopen, filelength, fileno, fstat, _grow_handles, isatty, lseek, open, read, sopen, stat, tell, write, umask

Example: #include <stdio.h>
#include <fcntl.h>
#include <io.h>

void main( void )
{
    FILE *fp;
    long count;

    fp = fopen( "file", "rb" );
    if( fp != NULL ) {
        setmode( fileno( fp ), O_BINARY );
        count = 0L;
        while( fgetc( fp ) != EOF ) ++count;
        printf( "File contains %lu characters\n", count );
        fclose( fp );
    }
}

Classification: WATCOM

Systems: setmode - All, Netware
_setmode - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32, Netware
**set_new_handler, _set_new_handler**

**Synopsis:**
```
#include <new.h>
PFV set_new_handler( PFV pNewHandler );
PFU _set_new_handler( PFU pNewHandler );
```

**Description:** The `set_new_handler` functions are used to transfer control to a user-defined error handler if the new operator fails to allocate memory. The argument `pNewHandler` is the name of a function of type `PFV` or `PFU`.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PFV</td>
<td>Pointer to a function that returns <code>void</code> (i.e., returns nothing) and takes an argument of type <code>void</code> (i.e., takes no argument).</td>
</tr>
<tr>
<td>PFU</td>
<td>Pointer to a function that returns <code>int</code> and takes an argument of type <code>unsigned</code> which is the amount of space to be allocated.</td>
</tr>
</tbody>
</table>

In a multi-threaded environment, handlers are maintained separately for each process and thread. Each new process lacks installed handlers. Each new thread gets a copy of its parent thread’s new handlers. Thus, each process and thread is in charge of its own free-store error handling.

**Returns:** The `set_new_handler` functions return a pointer to the previous error handler so that the previous error handler can be reinstated at a later time.

The error handler specified as the argument to `_set_new_handler` returns zero indicating that further attempts to allocate memory should be halted or non-zero to indicate that an allocation request should be re-attempted.

**See Also:** `_bfreeseg, _bheapseg, calloc, free, malloc, realloc`

**Example:**
```
#include <stdio.h>
#include <new.h>

#if defined(__386__)
const size_t MemBlock = 8192;
#else
const size_t MemBlock = 2048;
#endif

/*
   Pre-allocate a memory block for demonstration purposes. The out-of-memory handler will return it to the system so that "new" can use it.
*/
long *failsafe = new long[MemBlock];
```
/*
   Declare a customized function to handle memory
   allocation failure.
*/

int out_of_memory_handler( unsigned size )
{
    printf( "Allocation failed, " );
    printf( "%u bytes not available.\n", size );
    /* Release pre-allocated memory if we can */
    if( failsafe == NULL ) {
        printf( "Halting allocation.\n" );
        /* Tell new to stop allocation attempts */
        return( 0 );
    } else {
        delete failsafe;
        failsafe = NULL;
        printf( "Retrying allocation.\n" );
        /* Tell new to retry allocation attempt */
        return( 1 );
    }
}

void main( void )
{
    int i;

    /* Register existence of a new memory handler */
    _set_new_handler( out_of_memory_handler );
    long *pmemdump = new long[MemBlock];
    for( i=1 ; pmemdump != NULL; i++ ) {
        pmemdump = new long[MemBlock];
        if( pmemdump != NULL )
            printf( "Another block allocated %d\n", i );
    }
}

Classification: WATCOM

Systems: set_new_handler - All, Netware
          _set_new_handler - All, Netware
Synopsis:  
#include <graph.h>
short _FAR _setpixel( short x, short y );
short _FAR _setpixel_w( double x, double y );

Description:  The _setpixel function sets the pixel value of the point \((x,y)\) using the current plotting action with the current color. The _setpixel function uses the view coordinate system. The _setpixel_w function uses the window coordinate system.

A pixel value is associated with each point. The values range from 0 to the number of colors (less one) that can be represented in the palette for the current video mode. The color displayed at the point is the color in the palette corresponding to the pixel number. For example, a pixel value of 3 causes the fourth color in the palette to be displayed at the point in question.

Returns:  The _setpixel functions return the previous value of the indicated pixel if the pixel value can be set; otherwise, (-1) is returned.

See Also:  _getpixel, _setcolor, _setplotaction

Example:  
#include <conio.h>
#include <graph.h>
#include <stdlib.h>

main()
{
    int x, y;
    unsigned i;

    _setvideomode( _VRES16COLOR );
    _rectangle( _GBORDER, 100, 100, 540, 380 );
    for( i = 0; i <= 60000; ++i ) {
        x = 101 + rand() % 439;
        y = 101 + rand() % 279;
        _setcolor( _getpixel( x, y ) + 1 );
        _setpixel( x, y );
    }
    getch();
    _setvideomode( _DEFAULTMODE );
}

Classification: _setpixel is PC Graphics

Systems:  _setpixel - DOS, QNX
          _setpixel_w - DOS, QNX
Synopsis:  
#include <graph.h>  
short _FAR _setplotaction( short action );

Description: The _setplotaction function sets the current plotting action to the value of the action argument.  
The drawing functions cause pixels to be set with a pixel value. By default, the value to be set is obtained by replacing the original pixel value with the supplied pixel value. Alternatively, the replaced value may be computed as a function of the original and the supplied pixel values.

The plotting action can have one of the following values:

_GPSET  replace the original screen pixel value with the supplied pixel value

_GAND  replace the original screen pixel value with the bitwise and of the original pixel value and the supplied pixel value

_GOR  replace the original screen pixel value with the bitwise or of the original pixel value and the supplied pixel value

_GXOR  replace the original screen pixel value with the bitwise exclusive-or of the original pixel value and the supplied pixel value. Performing this operation twice will restore the original screen contents, providing an efficient method to produce animated effects.

Returns: The previous value of the plotting action is returned.

See Also: _getplotaction

Example:  
#include <conio.h>  
#include <graph.h>  

main()  
{  
    int old_act;  

    _setvideomode( _VRES16COLOR );  
    old_act = _getplotaction();  
    _setplotaction( _GPSET );  
    _rectangle( _GFILLINTERIOR, 100, 100, 540, 380 );  
    getch();  
    _setplotaction( _GXOR );  
    _rectangle( _GFILLINTERIOR, 100, 100, 540, 380 );  
    getch();  
    _setplotaction( old_act );  
    _setvideomode( _DEFAULTMODE );  
}

Classification: PC Graphics

Systems: DOS, QNX
Synopsis:  
#include <graph.h>
void _FAR _settextalign( short horiz, short vert );

Description: The _settextalign function sets the current text alignment to the values specified by the arguments horiz and vert. When text is displayed with the _grtext function, it is aligned (justified) horizontally and vertically about the given point according to the current text alignment settings.

The horizontal component of the alignment can have one of the following values:

_NORMAL       use the default horizontal alignment for the current setting of the text path
_LEFT         the text string is left justified at the given point
_CENTER       the text string is centred horizontally about the given point
_RIGHT        the text string is right justified at the given point

The vertical component of the alignment can have one of the following values:

_NORMAL       use the default vertical alignment for the current setting of the text path
_TOP          the top of the text string is aligned at the given point
_CAP          the cap line of the text string is aligned at the given point
_HALF         the text string is centred vertically about the given point
_BASE         the base line of the text string is aligned at the given point
_BOTTOM       the bottom of the text string is aligned at the given point

The default is to use _LEFT alignment for the horizontal component unless the text path is _PATH_LEFT, in which case _RIGHT alignment is used. The default value for the vertical component is _TOP unless the text path is _PATH_UP, in which case _BOTTOM alignment is used.

Returns: The _settextalign function does not return a value.

See Also:  _grtext, _gettextsettings

Example:  
#include <conio.h>
#include <graph.h>
main()
{
    _setvideomode( _VRES16COLOR );
    _grtext( 200, 100, "WATCOM" );
    _setpixel( 200, 100 );
    _settextalign( _CENTER, _HALF );
    _grtext( 200, 200, "Graphics" );
    _setpixel( 200, 200 );
    getch();
    _setvideomode( _DEFAULTMODE );
}
produces the following:

Classification: PC Graphics

Systems: DOS, QNX
Synopsis: 
#include <graph.h>
short _FAR _settextcolor( short pixval );

Description: The _settextcolor function sets the current text color to be the color indicated by the pixel value of the pixval argument. This is the color value used for displaying text with the _outtext and _outmem functions. Use the _setcolor function to change the color of graphics output. The default text color value is set to 7 whenever a new video mode is selected.

The pixel value pixval is a number in the range 0-31. Colors in the range 0-15 are displayed normally. In text modes, blinking colors are specified by adding 16 to the normal color values. The following table specifies the default colors in color text modes.

<table>
<thead>
<tr>
<th>Pixel Value</th>
<th>Color</th>
<th>Pixel Value</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Black</td>
<td>8</td>
<td>Gray</td>
</tr>
<tr>
<td>1</td>
<td>Blue</td>
<td>9</td>
<td>Light Blue</td>
</tr>
<tr>
<td>2</td>
<td>Green</td>
<td>10</td>
<td>Light Green</td>
</tr>
<tr>
<td>3</td>
<td>Cyan</td>
<td>11</td>
<td>Light Cyan</td>
</tr>
<tr>
<td>4</td>
<td>Red</td>
<td>12</td>
<td>Light Red</td>
</tr>
<tr>
<td>5</td>
<td>Magenta</td>
<td>13</td>
<td>Light Magenta</td>
</tr>
<tr>
<td>6</td>
<td>Brown</td>
<td>14</td>
<td>Yellow</td>
</tr>
<tr>
<td>7</td>
<td>White</td>
<td>15</td>
<td>Bright White</td>
</tr>
</tbody>
</table>

Returns: The _settextcolor function returns the pixel value of the previous text color.

See Also: _gettextcolor, _outtext, _outmem, _setcolor

Example: #include <conio.h>
#include <graph.h>

main()
{
    int old_col;
    long old_bk;

    _setvideomode( _TEXTC80 );
    old_col = _gettextcolor();
    old_bk = _getbkcolor();
    _settextcolor( 7 );
    _setbkcolor( _BLUE );
    _outtext( " WATCOM \nGraphics" );
    _settextcolor( old_col );
    _setbkcolor( old_bk );
    getch();
    _setvideomode( _DEFAULTMODE );
}

Classification: PC Graphics

Systems: DOS, QNX
Synopsis:  
#include <graph.h>  
short _FAR _settextcursor( short cursor );

Description:  The _settextcursor function sets the attribute, or shape, of the cursor in text modes. The argument cursor specifies the new cursor shape. The cursor shape is selected by specifying the top and bottom rows in the character matrix. The high byte of cursor specifies the top row of the cursor; the low byte specifies the bottom row.

Some typical values for cursor are:

<table>
<thead>
<tr>
<th>Cursor</th>
<th>Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0607</td>
<td>normal underline cursor</td>
</tr>
<tr>
<td>0x0007</td>
<td>full block cursor</td>
</tr>
<tr>
<td>0x0407</td>
<td>half-height block cursor</td>
</tr>
<tr>
<td>0x2000</td>
<td>no cursor</td>
</tr>
</tbody>
</table>

Returns:  The _settextcursor function returns the previous cursor shape when the shape is set successfully; otherwise, (-1) is returned.

See Also:  _gettextcursor, _displaycursor

Example:  
#include <conio.h>  
#include <graph.h>  

main()
{
    int old_shape;

    old_shape = _gettextcursor();
    _settextcursor( 0x0007 );
    _outtext( "\nBlock cursor" );
    getch();
    _settextcursor( 0x0407 );
    _outtext( "\nHalf height cursor" );
    getch();
    _settextcursor( 0x2000 );
    _outtext( "\nNo cursor" );
    getch();
    _settextcursor( old_shape );
}

Classification:  PC Graphics

Systems:  DOS, QNX
Synopsis:    
#include <graph.h>
void _FAR _settextorient( short vecx, short vecy );

Description: The _settextorient function sets the current text orientation to the vector specified by the arguments (vecx,vecy). The text orientation specifies the direction of the base-line vector when a text string is displayed with the _grtext function. The default text orientation, for normal left-to-right text, is the vector (1,0).

Returns:    The _settextorient function does not return a value.

See Also:   _grtext,_gettextsettings

Example:    #include <conio.h>
#include <graph.h>
main()
{
    _setvideomode( _VRES16COLOR );
    _grtext( 200, 100, "WATCOM" );
    _settextorient( 1, 1 );
    _grtext( 200, 200, "Graphics" );
    getch();
    _setvideomode( _DEFAULTMODE );
}

produces the following:

WATCOM

Graphics

Classification:  PC Graphics

Systems:       DOS, QNX

858   Library Functions and Macros
Synopsis:  
#include <graph.h>  
void _FAR _settextpath( short path );

Description:  The _settextpath function sets the current text path to have the value of the path argument. The text path specifies the writing direction of the text displayed by the _grtext function. The argument can have one of the following values:

(PATH_RIGHT) subsequent characters are drawn to the right of the previous character
(PATH_LEFT) subsequent characters are drawn to the left of the previous character
(PATH_UP) subsequent characters are drawn above the previous character
(PATH_DOWN) subsequent characters are drawn below the previous character

The default value of the text path is PATH_RIGHT.

Returns:  The _settextpath function does not return a value.

See Also:  _grtext, _gettextsettings

Example:  #include <conio.h>
#include <graph.h>

main()
{
    _setvideomode( _VRES16COLOR );
    _grtext ( 200, 100, "WATCOM" );
    _settextpath( _PATH_DOWN );
    _grtext ( 200, 200, "Graphics" );
    getch();
    _setvideomode( _DEFAULTMODE );
}

produces the following:
Classification: PC Graphics

Systems: DOS, QNX
Synopsis:
#include <graph.h>
struct rccoord _FAR _settextposition( short row,
                     short col );

Description: The _settextposition function sets the current output position for text to be (row, col) where
this position is in terms of characters, not pixels.

The text position is relative to the current text window. It defaults to the top left corner of the screen,
(1,1), when a new video mode is selected, or when a new text window is set. The position is
updated as text is drawn with the _outtext and _outmem functions.

Note that the output position for graphics output differs from that for text output. The output position
for graphics output can be set by use of the _moveto function.

Also note that output to the standard output file, stdout, is line buffered by default. It may be
necessary to flush the output stream using fflush( stdout ) after a printf call if your output
does not contain a newline character. Mixing of calls to _outtext and printf may cause
overlapped text since _outtext uses the output position that was set by _settextposition.

Returns: The _settextposition function returns, as an rccoord structure, the previous output position
for text.

See Also: _gettextposition, _outtext, _outmem, _settextwindow, _moveto

Example: #include <conio.h>
#include <graph.h>

main()
{
  struct rccoord old_pos;

  _setvideomode( _TEXTC80 );
  old_pos = _gettextposition();
  _settextposition( 10, 40 );
  _outtext( "WATCOM Graphics" );
  _settextposition( old_pos.row, old_pos.col );
  getch();
  _setvideomode( _DEFAULTMODE );
}

Classification: PC Graphics

Systems: DOS, QNX
Synopsis: The _settextrows function selects the number of rows of text displayed on the screen. The number of rows is specified by the argument rows. Computers equipped with EGA, MCGA and VGA adapters can support different numbers of text rows. The number of rows that can be selected depends on the current video mode and the type of monitor attached.

If the argument rows has the value _MAXTEXTROWS, the maximum number of text rows will be selected for the current video mode and hardware configuration. In text modes the maximum number of rows is 43 for EGA adapters, and 50 for MCGA and VGA adapters. Some graphics modes will support 43 rows for EGA adapters and 60 rows for MCGA and VGA adapters.

Returns: The _settextrows function returns the number of screen rows when the number of rows is set successfully; otherwise, zero is returned.

See Also: _getvideoconfig, _setvideomode, _setvideomoderows

Example: #include <conio.h>
#include <graph.h>
#include <stdio.h>

int valid_rows[] = {
    14, 25, 28, 30,
    34, 43, 50, 60
};

main()
{
    int i, j, rows;
    char buf[ 80 ];

    for( i = 0; i < 8; ++i ) {
        rows = valid_rows[ i ];
        if( _settextrows( rows ) == rows ) {
            for( j = 1; j <= rows; ++j ) {
                sprintf( buf, "Line \d", j );
                _settextposition( j, 1 );
                _outtext( buf );
            }
            getch();
        }
    }
    _setvideomode( _DEFAULTMODE );
}

Classification: PC Graphics

Systems: DOS, QNX
Synopsis:    
#include <graph.h>
void _FAR _settextwindow( short row1, short col1,
                      short row2, short col2 );

Description: The _settextwindow function sets the text window to be the rectangle with a top left corner at (row1,col1) and a bottom right corner at (row2,col2). These coordinates are in terms of characters not pixels.

The initial text output position is (1,1). Subsequent text positions are reported (by the _gettextposition function) and set (by the _outtext, _outmem and _settextposition functions) relative to this rectangle.

Text is displayed from the current output position for text proceeding along the current row and then downwards. When the window is full, the lines scroll upwards one line and then text is displayed on the last line of the window.

Returns: The _settextwindow function does not return a value.

See Also: _gettextposition, _outtext, _outmem, _settextposition

Example:    
#include <conio.h>
#include <graph.h>
#include <stdio.h>

main()
{
    int i;
    short r1, c1, r2, c2;
    char buf[ 80 ];

    _setvideomode( _TEXTC80 );
    _gettextwindow( &r1, &c1, &r2, &c2 );
    _settextwindow( 5, 20, 20, 40 );
    for( i = 1; i <= 20; ++i ) {
        sprintf( buf, "Line %d\n", i );
        _outtext( buf );
    }
    getch();
    _settextwindow( r1, c1, r2, c2 );
    _setvideomode( _DEFAULTMODE );
}

Classification: PC Graphics

Systems: DOS, QNX
Synopsis: The `setvbuf` function can be used to associate a buffer with the file designated by `fp`. If this function is used, it must be called after the file has been opened and before it has been read or written. The argument `mode` determines how the file `fp` will be buffered, as follows:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>_IOFBF</td>
<td>causes input/output to be fully buffered.</td>
</tr>
<tr>
<td>_IOLBF</td>
<td>causes output to be line buffered (the buffer will be flushed when a new-line character is written, when the buffer is full, or when input is requested on a line buffered or unbuffered stream).</td>
</tr>
<tr>
<td>_IONBF</td>
<td>causes input/output to be completely unbuffered.</td>
</tr>
</tbody>
</table>

If the argument `buf` is not `NULL`, the array to which it points will be used instead of an automatically allocated buffer. The argument `size` specifies the size of the array.

Returns: The `setvbuf` function returns zero on success, or a non-zero value if an invalid value is given for `mode` or `size`.

See Also: `fopen`, `setbuf`

Example: ```c
#include <stdio.h>
#include <stdlib.h>

void main()
{
  char *buf;
  FILE *fp;

  fp = fopen( "file", "r" );
  buf = (char *) malloc( 1024 );
  setvbuf( fp, buf, _IOFBF, 1024 );
}
```

Classification: ANSI

Systems: All, Netware
Synopsis: #include <graph.h>
short _FAR _setvideomode( short mode );

Description: The _setvideomode function sets the video mode according to the value of the mode argument. The value of mode can be one of the following:

- _MAXRESMODE (graphics mode with highest resolution)
- _MAXCOLORMODE (graphics mode with most colors)
- _DEFAULTMODE (restores screen to original mode)
- _TEXTBW40 M,T 40 x 25 16 MDPA, HGC, VGA, SVGA
- _TEXTC40 C,T 40 x 25 16 CGA, EGA, MCGA, VGA, SVGA
- _TEXTBW80 M,T 80 x 25 16 MDPA, HGC, VGA, SVGA
- _TEXTC80 C,T 80 x 25 16 CGA, EGA, MCGA, VGA, SVGA
- _MRES4COLOR C,G 320 x 200 4 CGA, EGA, MCGA, VGA, SVGA
- _MRESNOCOLOR C,G 320 x 200 4 CGA, EGA, MCGA, VGA, SVGA
- _HRESBW C,G 640 x 200 2 CGA, EGA, MCGA, VGA, SVGA
- _TEXTMONO M,T 80 x 25 16 MDPA, HGC, VGA, SVGA
- _HERCMONO M,G 720 x 350 2 HGC
- _RES16COLOR C,G 320 x 200 16 EGA, VGA, SVGA
- _RES16COLOR C,G 640 x 200 16 EGA, VGA, SVGA
- _RESNOCOLOR M,G 640 x 350 4 EGA, VGA, SVGA
- _RES16COLOR C,G 640 x 350 4/16 EGA, VGA, SVGA
- _RES2COLOR C,G 640 x 480 2 MCGA, VGA, SVGA
- _RES16COLOR C,G 640 x 480 16 VGA, SVGA
- _RES256COLOR C,G 320 x 200 256 MCGA, VGA, SVGA
- _RES256COLOR C,G 640 x 400 256 SVGA
- _RES256COLOR C,G 640 x 480 256 SVGA
- _SVRES16COLOR C,G 800 x 600 16 SVGA
- _SVRES256COLOR C,G 800 x 600 256 SVGA
- _RES256COLOR C,G 1024 x 768 16 SVGA
- _RES256COLOR C,G 1024 x 768 256 SVGA

In the preceding table, the Type column contains the following letters:

- \( M \) indicates monochrome; multiple colors are shades of grey
- \( C \) indicates color
- \( G \) indicates graphics mode; size is in pixels
- \( T \) indicates text mode; size is in columns and rows of characters

The Adapter column contains the following codes:

- \( MDPA \) IBM Monochrome Display/Printer Adapter
- \( CGA \) IBM Color Graphics Adapter
- \( EGA \) IBM Enhanced Graphics Adapter
- \( VGA \) IBM Video Graphics Array
- \( MCGA \) IBM Multi-Color Graphics Array
- \( HGC \) Hercules Graphics Adapter
**_setvideomode_**

**SVGA SuperVGA adapters**

The modes _MAXRESMODE_ and _MAXCOLORMODE_ will select from among the video modes supported by the current graphics adapter the one that has the highest resolution or the greatest number of colors. The video mode will be selected from the standard modes, not including the SuperVGA modes.

Selecting a new video mode resets the current output positions for graphics and text to be the top left corner of the screen. The background color is reset to black and the default color value is set to be one less than the number of colors in the selected mode.

**Returns:** The _setvideomode_ function returns the number of text rows when the new mode is successfully selected; otherwise, zero is returned.

**See Also:** _getvideoconfig, _settextrows, _setvideomoderows_
Example:

```c
#include <conio.h>
#include <graph.h>
#include <stdio.h>
#include <stdlib.h>

main()
{
    int mode;
    struct videoconfig vc;
    char buf[ 80 ];

    _getvideoconfig( &vc );
    /* select "best" video mode */
    switch( vc.adapter ) {
    case _VGA :
    case _SVGA :
        mode = _VRES16COLOR;
        break;
    case _MCGA :
        mode = _MRES256COLOR;
        break;
    case _EGA :
        if( vc.monitor == _MONO ) {
            mode = _ERESNOCOLOR;
        } else {
            mode = _ERESCOLOR;
        }
        break;
    case _CGA :
        mode = _MRES4COLOR;
        break;
    case _HERCULES :
        mode = _HERCMONO;
        break;
    default :
        puts( "No graphics adapter" );
        exit( 1 );
    }
    if( _setvideomode( mode ) ) {
        _getvideoconfig( &vc );
        sprintf( buf, "%d x %d x %d\n", vc.numxpixels,
                 vc.numypixels, vc.numcolors );
        _outtext( buf );
        getch();
        _setvideomode( _DEFAULTMODE );
    }
}
```

Classification: PC Graphics

Systems: DOS, QNX
Synopsis:  
#include <graph.h>
short _FAR _setvideomoderows( short mode, short rows );

Description:  
The _setvideomoderows function selects a video mode and the number of rows of text displayed on the screen. The video mode is specified by the argument mode and is selected with the _setvideomode function. The number of rows is specified by the argument rows and is selected with the _settextrows function.

Computers equipped with EGA, MCGA and VGA adapters can support different numbers of text rows. The number of rows that can be selected depends on the video mode and the type of monitor attached.

Returns:  
The _setvideomoderows function returns the number of screen rows when the mode and number of rows are set successfully; otherwise, zero is returned.

See Also:  
_getvideoconfig, _setvideomode, _settextrows

Example:  
#include <conio.h>
#include <graph.h>
#include <stdio.h>

main()
{
  int rows;
  char buf[ 80 ];

  rows = _setvideomoderows( _TEXTC80, _MAXTEXTROWS );
  if( rows != 0 ) {
    sprintf( buf, "Number of rows is %d\n", rows );
    _outtext( buf );
    getch();
    _setvideomode( _DEFAULTMODE );
  }
}

Classification:  PC Graphics

Systems:  DOS, QNX
Synopsis:  
#include <graph.h>
struct xycoord _FAR _setvieworg( short x, short y );

Description:  The _setvieworg function sets the origin of the view coordinate system, (0,0), to be located at the physical point (x,y). This causes subsequently drawn images to be translated by the amount (x,y).

Note: In previous versions of the software, the _setvieworg function was called _setlogorg.

Returns:  The _setvieworg function returns, as an xycoord structure, the physical coordinates of the previous origin.

See Also:  _getviewcoord, _getphyscoord, _setcliprgn, _setviewport

Example:  
#include <conio.h>
#include <graph.h>

main()
{
  _setvideomode ( _VRES16COLOR );
  _setvieworg( 320, 240 );
  _ellipse( _GBORDER, -200, -150, 200, 150 );
  getch();
  _setvideomode ( _DEFAULTMODE );
}

Classification:  PC Graphics

Systems:  DOS, QNX
_setviewport

Synopsis:

#include <graph.h>

void _FAR _setviewport( short x1, short y1,
short x2, short y2 );

Description: The _setviewport function restricts the display of graphics output to the clipping region and then sets the origin of the view coordinate system to be the top left corner of the region. This region is a rectangle whose opposite corners are established by the physical points (x1,y1) and (x2,y2).

The _setviewport function does not affect text output using the _outtext and _outmem functions. To control the location of text output, see the _settextwindow function.

Returns: The _setviewport function does not return a value.

See Also: _setcliprgn, _setvieworg, _settextwindow, _setwindow

Example:

#include <conio.h>
#include <graph.h>

#define XSIZE 380
#define YSIZE 280

main()
{
    _setvideomode( _VRES16COLOR );
    _setviewport( 130, 100, 130 + XSIZE, 100 + YSIZE );
    _ellipse( _GBORDER, 0, 0, XSIZE, YSIZE );
    getch();
    _setvideomode( _DEFAULTMODE );
}

Classification: PC Graphics

Systems: DOS, QNX
Synopsis:  
#include <graph.h>
short _FAR _setvisualpage( short pagenum );

Description: The _setvisualpage function selects the page (in memory) from which graphics output is displayed. The page to be selected is given by the pagenum argument.

Only some combinations of video modes and hardware allow multiple pages of graphics to exist. When multiple pages are supported, the active page may differ from the visual page. The graphics information in the visual page determines what is displayed upon the screen. Animation may be accomplished by alternating the visual page. A graphics page can be constructed without affecting the screen by setting the active page to be different than the visual page.

The number of available video pages can be determined by using the _getvideoconfig function. The default video page is 0.

Returns: The _setvisualpage function returns the number of the previous page when the visual page is set successfully; otherwise, a negative number is returned.

See Also: _getvisualpage, _setactivepage, _getactivepage, _getvideoconfig

Example: 
#include <conio.h>
#include <graph.h>

main()
{
    int old_apage;
    int old_vpage;

    _setvideomode( _HRES16COLOR );
    old_apage = _getactivepage();
    old_vpage = _getvisualpage();
    /* draw an ellipse on page 0 */
    _setactivepage( 0 );
    _setvisualpage( 0 );
    _ellipse( _GFILLINTERIOR, 100, 50, 540, 150 );
    /* draw a rectangle on page 1 */
    _setactivepage( 1 );
    _rectangle( _GFILLINTERIOR, 100, 50, 540, 150 );
    getch();
    /* display page 1 */
    _setvisualpage( 1 );
    getch();
    _setactivepage( old_apage );
    _setvisualpage( old_vpage );
    _setvideomode( _DEFAULTMODE );
}

Classification: PC Graphics

Systems: DOS, QNX
Synopsis:  
#include <graph.h>
short _FAR _setwindow( short invert, 
double x1, double y1, 
double x2, double y2 );

Description:  The _setwindow function defines a window for the window coordinate system. Window coordinates are specified as a user-defined range of values. This allows for consistent pictures regardless of the video mode.

The window is defined as the region with opposite corners established by the points (x1,y1) and (x2,y2). The argument invert specifies the direction of the y-axis. If the value is non-zero, the y values increase from the bottom of the screen to the top, otherwise, the y values increase as you move down the screen.

The window defined by the _setwindow function is displayed in the current viewport. A viewport is defined by the _setviewport function.

By default, the window coordinate system is defined with the point (0.0,0.0) located at the lower left corner of the screen, and the point (1.0,1.0) at the upper right corner.

Returns:  The _setwindow function returns a non-zero value when the window is set successfully; otherwise, zero is returned.

See Also:  _setviewport

Example:  
#include <conio.h>
#include <graph.h>

main()
{
  _setvideomode( _MAXRESMODE );
  draw_house( "Default window" );
  _setwindow( 1, -0.5, -0.5, 1.5, 1.5 );
  draw_house( "Larger window" );
  _setwindow( 1, 0.0, 0.0, 0.5, 1.0 );
  draw_house( "Left side" );
  _setvideomode( _DEFAULTMODE );
}

draw_house( char *msg )
{
  _clearscreen( _GCLEARSCREEN );
  _outtext( msg );
  _rectangle_w( _GBORDER, 0.2, 0.1, 0.8, 0.6 );
  _moveto_w( 0.1, 0.5 );
  _lineto_w( 0.5, 0.9 );
  _lineto_w( 0.9, 0.5 );
  _arc_w( 0.4, 0.5, 0.6, 0.3, 0.6, 0.4, 0.4, 0.4 );
  _rectangle_w( _GBORDER, 0.4, 0.1, 0.6, 0.4 );
  getch();
}

Classification: PC Graphics
Systems: DOS, QNX
**Synopsis:**

```c
#include <signal.h>
void ( *signal(int sig, void (*func)(int)) )( int );
```

**Description:** The `signal` function is used to specify an action to take place when certain conditions are detected while a program executes. These conditions are defined to be:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGABRT</td>
<td>abnormal termination, such as caused by the <code>abort</code> function</td>
</tr>
<tr>
<td>SIGBREAK</td>
<td>an interactive attention (CTRL/BREAK on keyboard) is signalled</td>
</tr>
<tr>
<td>SIGFPE</td>
<td>an erroneous floating-point operation occurs (such as division by zero, overflow and underflow)</td>
</tr>
<tr>
<td>SIGILL</td>
<td>illegal instruction encountered</td>
</tr>
<tr>
<td>SIGINT</td>
<td>an interactive attention (CTRL/C on keyboard) is signalled</td>
</tr>
<tr>
<td>SIGSEGV</td>
<td>an illegal memory reference is detected</td>
</tr>
<tr>
<td>SIGTERM</td>
<td>a termination request is sent to the program</td>
</tr>
<tr>
<td>SIGUSR1</td>
<td>OS/2 process flag A via DosFlagProcess</td>
</tr>
<tr>
<td>SIGUSR2</td>
<td>OS/2 process flag B via DosFlagProcess</td>
</tr>
<tr>
<td>SIGUSR3</td>
<td>OS/2 process flag C via DosFlagProcess</td>
</tr>
</tbody>
</table>

An action can be specified for each of the conditions, depending upon the value of the `func` argument:

**function**

When `func` is a function name, that function will be called equivalently to the following code sequence.

```c
/* "sig_no" is condition being signalled */
signal( sig_no, SIG_DFL );
(*func)( sig_no );
```

The `func` function may terminate the program by calling the `exit` or `abort` functions or call the `longjmp` function. Because the next signal will be handled with default handling, the program must again call `signal` if it is desired to handle the next condition of the type that has been signalled.

After returning from the signal-catching function, the receiving process will resume execution at the point at which it was interrupted.

The signal catching function is described as follows:

```c
void func( int sig_no )
{
    /* body of function */
}
```
Since signal-catching functions are invoked asynchronously with process execution, the
type sig_atomic_t may be used to define variables on which an atomic operation
(e.g., incrementation, decrementation) may be performed.

**SIG_DFL**
This value causes the default action for the condition to occur.

**SIG_IGN**
This value causes the indicated condition to be ignored.

When a condition is detected, it may be handled by a program, it may be ignored, or it may be handled
by the usual default action (often causing an error message to be printed upon the stderr stream
followed by program termination).

When the program begins execution, the equivalent of

```c
    signal( SIGABRT, SIG_DFL );
    signal( SIGFPE, SIG_DFL );
    signal( SIGILL, SIG_DFL );
    signal( SIGINT, SIG_DFL );
    signal( SIGSEGV, SIG_DFL );
    signal( SIGTERM, SIG_DFL );
    signal( SIGBREAK, SIG_DFL );
    signal( SIGUSR1, SIG_IGN );
    signal( SIGUSR2, SIG_IGN );
    signal( SIGUSR3, SIG_IGN );
```

is executed.

The SIGINT signal is generated by pressing the CTRL/C or CTRL/BREAK key combination on the
keyboard. Under DOS, if "BREAK=ON", a signal will be delivered at the next DOS call; otherwise, if
"BREAK=OFF", a signal will be delivered only at the next standard input/output DOS call. The
BREAK setting is configured in the CONFIG.SYS file.

Under OS/2, the SIGBREAK signal can only be received if CTRL/BREAK is pressed and the keyboard
is in binary (raw) mode. In ASCII (cooked) mode, which is the default, both CTRL/C and
CTRL/BREAK combinations will raise the SIGINT signal.

A condition can be generated by a program using the raise function.

**Returns:**
A return value of SIG_ERR indicates that the request could not be handled, and errno is set to the
value EINVAL.

Otherwise, the previous value of func for the indicated condition is returned.

**See Also:**
break..., raise

**Example:**
```c
#include <stdio.h>
#include <signal.h>
#include <i86.h>

/* SIGINT Test */

    sig_atomic_t signal_count;
    sig_atomic_t signal_number;
```
void MyIntHandler( int signo )
{
    signal_count++;
    signal_number = signo;
}

void MyBreakHandler( int signo )
{
    signal_count++;
    signal_number = signo;
}

int main( void )
{
    int i;

    signal_count = 0;
    signal_number = 0;
    signal( SIGINT, MyIntHandler );
    signal( SIGBREAK, MyBreakHandler );
    printf( "Press Ctrl/C or Ctrl/Break\n" );
    for( i = 0; i < 50; i++ ) {
        printf( "Iteration # %d\n", i );
        delay( 500 ); /* sleep for 1/2 second */
        if( signal_count > 0 ) break;
    }
    printf( "SIGINT count %d number %d\n",
            signal_count, signal_number );

    signal_count = 0;
    signal_number = 0;
    signal( SIGINT, SIG_DFL ); /* Default action */
    signal( SIGBREAK, SIG_DFL ); /* Default action */
    printf( "Default signal handling\n" );
    for( i = 0; i < 50; i++ ) {
        printf( "Iteration # %d\n", i );
        delay( 500 ); /* sleep for 1/2 second */
        if( signal_count > 0 ) break; /* Won’t happen */
    }
    return( signal_count );
}

Classification: ANSI

Systems: All, Netware
Synopsis:  
#include <math.h>
int signbit( x );

Description:  The signbit macro determines whether the sign of its argument value is negative.

The argument x must be an expression of real floating type.

Returns:  The signbit macro returns a nonzero value if and only if the sign of its argument has value is negative.

See Also:  fpclassify, isfinite, isinf, isnan, isnormal

Example:  
#include <math.h>
#include <stdio.h>

    void main( void )
    {
        printf( "-4.5 %s negative\n", 
                signbit( -4.5 ) ? "is" : "is not" );
    }

produces the following:

-4.5 is negative

Classification:  ANSI

Systems:  MACRO
Synopsis: 
#include <math.h>
double sin( double x );

Description: The sin function computes the sine of x (measured in radians). A large magnitude argument may yield a result with little or no significance.

Returns: The sin function returns the sine value.

See Also: acos, asin, atan, atan2, cos, tan

Example: 
#include <stdio.h>
#include <math.h>

void main()
{
    printf( "%f\n", sin(.5) );
}

produces the following:

0.479426

Classification: ANSI

Systems: Math
Synopsis:  #include <math.h>
    double sinh( double x );

Description:  The sinh function computes the hyperbolic sine of x. A range error occurs if the magnitude of x is too large.

Returns:  The sinh function returns the hyperbolic sine value. When the argument is outside the permissible range, the matherr function is called. Unless the default matherr function is replaced, it will set the global variable errno to ERANGE, and print a "RANGE error" diagnostic message using the stderr stream.

See Also:  cosh, tanh, matherr

Example:  #include <stdio.h>
    #include <math.h>

    void main()
    {
        printf( "%f\n", sinh(.5) );
    }

    produces the following:

    0.521095

Classification:  ANSI

Systems:  Math
**Synopsis:**

```c
#include <wchar.h>
int mbsinit( const mbstate_t *ps );
int sisinit( const mbstate_t *ps );
```

**Description:**

If `ps` is not a null pointer, the `mbsinit` function determines whether the pointed-to `mbstate_t` object describes an initial conversion state.

**Returns:**

The `mbsinit` function returns nonzero if `ps` is a null pointer or if the pointed-to object describes an initial conversion state; otherwise, it returns zero.

**See Also:**

`_mbccmp`, `_mbccpy`, `_mbcicmp`, `_mbcjistojms`, `_mbcjstmjis`, `_mbclen`, `_mbctohira`, `_mbctokata`, `_mbctolower`, `_mbctombb`, `_mbctoupper`, `_mblen`, `_mbrlen`, `_mbtowc`, `_mbstowcs`, `_mbstowcs_s`, `_mbstowcs`, `_mbtowc`, `btcwc`, `_wctomb_s`, `_wcsrtombs`, `_wcsrtombs_s`, `_wcstombs`, `_wcstombs_s`, `_wctob`, `_wctomb`, `_wctomb_s`

**Example:**
#include <stdio.h>
#include <wchar.h>
#include <mbctypes.h>
#include <errno.h>

const char chars[] = {
    ',', '
    '.', '
    '1',
    'A',
    0xB1,0x40, /* double-byte space */
    0xB2,0x60, /* double-byte A */
    0xB2,0xA6, /* double-byte Hiragana */
    0xB3,0x42, /* double-byte Katakana */
    0xA1,      /* single-byte Katakana punctuation */
    0xA6,      /* single-byte Katakana alphabetic */
    0xDF,      /* single-byte Katakana alphabetic */
    0xE0,0xA1, /* double-byte Kanji */
    0x00
};

void main( void )
{
    int           i, j, k;
    wchar_t       pwc;
    mbstate_t     pstate = { 0 };

    _setmbcp( 932 );
    j = 1;
    for( i = 0; i < 255; i += j )
    {
        printf( "We are %sin an initial conversion state\n",
            mbsinit( &pstate ) ? "not " : "" );
        j = mbtowc( &pwc, &chars[i], MB_CUR_MAX, &pstate );
        printf( "%d bytes in character ", j );
        if( errno == EILSEQ )
            printf( " - illegal multibyte character\n" );
        else if( j == 0 )
            k = 0;
        else if ( j == 1 )
            k = chars[i];
        else if ( j == 2 )
            k = chars[i]<<8 | chars[i+1];
        printf( "( %#6.4x ---> %#6.4x )\n", k, pwc );
    }
}

produces the following:
We are in an initial conversion state
1 bytes in character (0x0020->0x0020)
We are in an initial conversion state
1 bytes in character (0x002e->0x002e)
We are in an initial conversion state
1 bytes in character (0x0031->0x0031)
We are in an initial conversion state
1 bytes in character (0x0041->0x0041)
We are in an initial conversion state
2 bytes in character (0x8140->0x3000)
We are in an initial conversion state
2 bytes in character (0x8260->0xff21)
We are in an initial conversion state
2 bytes in character (0x82a6->0x3048)
We are in an initial conversion state
2 bytes in character (0x8342->0x30a3)
We are in an initial conversion state
1 bytes in character (0x00a1->0xff61)
We are in an initial conversion state
1 bytes in character (0x00a6->0xff66)
We are in an initial conversion state
1 bytes in character (0x00df->0xff9f)
We are in an initial conversion state
2 bytes in character (0xe0a1->0x720d)
We are in an initial conversion state
0 bytes in character (  0000->  0000)

Classification: ANSI

Systems:

mbsinit - All, Netware
sisinit - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  #include <dos.h>  unsigned sleep( unsigned seconds );

Description:  The sleep function suspends execution by the specified number of seconds.

Returns:  The sleep function always returns zero.

See Also:  delay

Example:  /*
   * The following program sleeps for the
   * number of seconds specified in argv[1].
   */
   #include <stdlib.h>
   #include <dos.h>

   void main( int argc, char *argv[] )
   {
      unsigned seconds;
      seconds = (unsigned) strtol( argv[1], NULL, 0 );
      sleep( seconds );
   }

Classification:  WATCOM

Systems:  All, Netware
_snprintf, _snwprintf

Synopsis:  
#include <stdio.h>
int _snprintf( char *buf,
 size_t count,
 const char *format, ... );
#include <wchar.h>
int _snwprintf( wchar_t *buf,
 size_t count,
 const wchar_t *format, ... );

Description:  The _snprintf function is equivalent to the fprintf function, except that the argument buf specifies a character array into which the generated output is placed, rather than to a file. The maximum number of characters to store is specified by count. A null character is placed at the end of the generated character string if fewer than count characters were stored. The format string is described under the description of the printf function.

The _snwprintf function is identical to _snprintf except that the argument buf specifies an array of wide characters into which the generated output is to be written, rather than converted to multibyte characters and written to a stream. The maximum number of wide characters to store is specified by count. A null wide character is placed at the end of the generated wide character string if fewer than count wide characters were stored. The _snwprintf function accepts a wide-character string argument for format.

Returns:  The _snprintf function returns the number of characters written into the array, not counting the terminating null character, or a negative value if more than count characters were requested to be generated. An error can occur while converting a value for output. The _snwprintf function returns the number of wide characters written into the array, not counting the terminating null wide character, or a negative value if more than count wide characters were requested to be generated. When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also:  _bprintf, cprintf, fprintf, sprintf, _vbprintf, vfprintf, printf, vsprintf

Example:  
#include <stdio.h>

/* Create temporary file names using a counter */

char namebuf[13];
int TempCount = 0;

char *make_temp_name()
{
    _snprintf( namebuf, 13, "ZZ%.6o.TMP", TempCount++ );
    return( namebuf);
}

void main()
{
    FILE *tf1, *tf2;
}
tf1 = fopen( make_temp_name(), "w" );
tf2 = fopen( make_temp_name(), "w" );
fputs( "temp file 1", tf1 );
fputs( "temp file 2", tf2 );
fclose( tf1 );
fclose( tf2 );

Classification: WATCOM

Systems: _snprintf - All, Netware
         _snwprintf - All
Synopsis:  
#include <stdio.h>  
int snprintf( char *buf, 
    size_t count,  
    const char *format, ... );

#include <wchar.h>  
int snwprintf( wchar_t *buf, 
    size_t count,  
    const wchar_t *format, ... );

Safer C:  
The Safer C Library extension provides the snprintf_s function which is a safer alternative to snprintf. This newer snprintf_s function is recommended to be used instead of the traditional "unsafe" snprintf function.

Description:  
The snprintf function is equivalent to the fprintf function, except that the argument buf specifies a character array into which the generated output is placed, rather than to a file. A null character is placed at the end of the generated character string. The maximum number of characters to store, including a terminating null character, is specified by count. The format string is described under the description of the printf function.

The snwprintf function is identical to snprintf except that the argument buf specifies an array of wide characters into which the generated output is to be written, rather than converted to multibyte characters and written to a stream. The maximum number of wide characters to store, including a terminating null wide character, is specified by count. The snwprintf function accepts a wide-character string argument for format.

Returns:  
The snprintf function returns the number of characters that would have been written had count been sufficiently large, not counting the terminating null character, or a negative value if an encoding error occurred. Thus, the null-terminated output has been completely written if and only if the returned value is nonnegative and less than count. The snwprintf function returns the number of wide characters that would have been written had count been sufficiently large, not counting the terminating null wide character, or a negative value if an encoding error occurred. Thus, the null-terminated output has been completely written if and only if the returned value is nonnegative and less than count. When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also:  
_bprintf, cprintf, fprintf, printf, sprintf, _vbprintf, wcprintf, vfprintf, vprintf, vsprintf

Example:  
#include <stdio.h>  
#include <stdlib.h>  
/* Format output into a buffer after determining its size */  
void main( void )  
{
    int bufsize;  
    char *buffer;

    bufsize = snprintf( NULL, 0, "%3d %P", 42, 42 );
    buffer = malloc( bufsize + 1 );
    snprintf( buffer, bufsize + 1, "%3d %P", 42, 42 );
    free( buffer );
}

Classification:  snprintf is ANSI
snprintf is ANSI

**Systems:**
- `snprintf` - All, Netware
- `snwprintf` - All
**Synopsis:**

```c
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
int snprintf_s( char * restrict s, rsize_t n
    const char * restrict format, ... );
#include <wchar.h>
int snwprintf_s( char * restrict s, rsize_t n,
    const wchar_t * restrict format, ... );
```

**Constraints:** If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `snprintf_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither `s` nor `format` shall be a null pointer. The `n` argument shall neither equal zero nor be greater than `RSIZE_MAX`. The number of characters (including the trailing null) required for the result to be written to the array pointed to by `s` shall not be greater than `n`. The `%n` specifier (modified or not by flags, field width, or precision) shall not appear in the string pointed to by `format`. Any argument to `snprintf_s` corresponding to a `%s` specifier shall not be a null pointer. No encoding error shall occur.

If there is a runtime-constraint violation, then if `s` is not a null pointer and `n` is greater than zero and less than `RSIZE_MAX`, then the `snprintf_s` function sets `s[0]` to the null character.

**Description:** The `snprintf_s` function is equivalent to the `snprintf` function except for the explicit runtime-constraints listed above.

The `snprintf_s` function, unlike `sprintf_s`, will truncate the result to fit within the array pointed to by `s`.

The `snwprintf_s` function is identical to `snprintf_s` except that it accepts a wide-character string argument for `format` and produces wide character output.

**Returns:** The `snprintf_s` function returns the number of characters that would have been written had `n` been sufficiently large, not counting the terminating null character, or a negative value if a runtime-constraint violation occurred. Thus, the null-terminated output has been completely written if and only if the returned value is nonnegative and less than `n`.

The `snwprintf_s` function returns the number of wide characters that would have been written had `n` been sufficiently large, not counting the terminating wide null character, or a negative value if a runtime-constraint violation occurred. Thus, the null-terminated output has been completely written if and only if the returned value is nonnegative and less than `n`.

**See Also:** `_bprintf`, `cprintf`, `fprintf`, `printf`, `sprintf`, `_vbprintf`, `vfprintf`, `vprintf`, `vsprintf`

**Example:**
```c
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <stdlib.h>

/* Format output into a buffer after determining its size */

void main( void )
{
    int bufsize;
    char *buffer;

    bufsize = snprintf( NULL, 0, "%3d %P", 42, 42 ) + 1;
    buffer = malloc( bufsize );
    snprintf_s( buffer, bufsize, "%3d %P", 42, 42 );
    free( buffer );
}

Classification: snprintf_s is TR 24731
                snwprintf_s is TR 24731

Systems:       snprintf_s - All, Netware
                snwprintf_s - All
```
Synopsis:

```c
#include <io.h>
#include <fcntl.h>
#include <sys\stat.h>
#include <sys\types.h>
#include <share.h>
int sopen( const char *filename,
            int access, int share, ... );
int _sopen( const char *filename,
            int access, int share, ... );
int _wsopen( const wchar_t *filename,
            int access, int share, ... );
```

Description:
The `sopen` function opens a file at the operating system level for shared access. The name of the file to be opened is given by `filename`. The file will be accessed according to the access mode specified by `access`. When the file is to be created, the optional argument must be given which establishes the future access permissions for the file. Additionally, the sharing mode of the file is given by the `share` argument. The optional argument is the file permissions to be used when `O_CREAT` flag is on in the `access` mode.

The `_sopen` function is identical to `sopen`. Use `_sopen` for ANSI/ISO naming conventions.

The `_wsopen` function is identical to `sopen` except that it accepts a wide character string argument.

The access mode is established by a combination of the bits defined in the `<fcntl.h>` header file. The following bits may be set:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>O_RDONLY</td>
<td>permit the file to be only read.</td>
</tr>
<tr>
<td>O_WRONLY</td>
<td>permit the file to be only written.</td>
</tr>
<tr>
<td>O_RDWR</td>
<td>permit the file to be both read and written.</td>
</tr>
<tr>
<td>O_APPEND</td>
<td>causes each record that is written to be written at the end of the file.</td>
</tr>
<tr>
<td>O_CREAT</td>
<td>has no effect when the file indicated by <code>filename</code> already exists; otherwise, the file is created;</td>
</tr>
<tr>
<td>O_TRUNC</td>
<td>causes the file to be truncated to contain no data when the file exists; has no effect when the file does not exist.</td>
</tr>
<tr>
<td>O_BINARY</td>
<td>causes the file to be opened in binary mode which means that data will be transmitted to and from the file unchanged.</td>
</tr>
<tr>
<td>O_TEXT</td>
<td>causes the file to be opened in text mode which means that carriage-return characters are written before any linefeed character that is written and causes carriage-return characters to be removed when encountered during reads.</td>
</tr>
<tr>
<td>O_NOINHERIT</td>
<td>indicates that this file is not to be inherited by a child process.</td>
</tr>
<tr>
<td>O_EXCL</td>
<td>indicates that this file is to be opened for exclusive access. If the file exists and <code>O_CREAT</code> was also specified then the open will fail (i.e., use <code>O_EXCL</code> to ensure that the file does not already exist).</td>
</tr>
</tbody>
</table>
When neither \texttt{O\_TEXT} nor \texttt{O\_BINARY} are specified, the default value in the global variable \texttt{fmode} is used to set the file translation mode. When the program begins execution, this variable has a value of \texttt{O\_TEXT}.

\texttt{O\_CREAT} must be specified when the file does not exist and it is to be written.

When the file is to be created (\texttt{O\_CREAT} is specified), an additional argument must be passed which contains the file permissions to be used for the new file. The access permissions for the file or directory are specified as a combination of bits (defined in the <\texttt{sys\_stat}\.h> header file).

The following bits define permissions for the owner.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{S_IRWXU}</td>
<td>Read, write, execute/search</td>
</tr>
<tr>
<td>\texttt{S_IRUSR}</td>
<td>Read permission</td>
</tr>
<tr>
<td>\texttt{S_IWUSR}</td>
<td>Write permission</td>
</tr>
<tr>
<td>\texttt{S_IXUSR}</td>
<td>Execute/search permission</td>
</tr>
</tbody>
</table>

The following bits define permissions for the group.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{S_IRWXG}</td>
<td>Read, write, execute/search</td>
</tr>
<tr>
<td>\texttt{S_IRGRP}</td>
<td>Read permission</td>
</tr>
<tr>
<td>\texttt{S_IWGRP}</td>
<td>Write permission</td>
</tr>
<tr>
<td>\texttt{S_IXGRP}</td>
<td>Execute/search permission</td>
</tr>
</tbody>
</table>

The following bits define permissions for others.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{S_IRWKO}</td>
<td>Read, write, execute/search</td>
</tr>
<tr>
<td>\texttt{S_IROTH}</td>
<td>Read permission</td>
</tr>
<tr>
<td>\texttt{S_IWOTH}</td>
<td>Write permission</td>
</tr>
<tr>
<td>\texttt{S_IXOTH}</td>
<td>Execute/search permission</td>
</tr>
</tbody>
</table>

The following bits define miscellaneous permissions used by other implementations.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{S_IREAD}</td>
<td>is equivalent to \texttt{S_IRUSR} (read permission)</td>
</tr>
<tr>
<td>\texttt{S_IWRITE}</td>
<td>is equivalent to \texttt{S_IWUSR} (write permission)</td>
</tr>
<tr>
<td>\texttt{S_IEXEC}</td>
<td>is equivalent to \texttt{S_IXUSR} (execute/search permission)</td>
</tr>
</tbody>
</table>

All files are readable with DOS; however, it is a good idea to set \texttt{S\_IREAD} when read permission is intended for the file.

The \texttt{sopen} function applies the current file permission mask to the specified permissions (see \texttt{umask}).

The shared access for the file, \texttt{share}, is established by a combination of bits defined in the <\texttt{share}\.h> header file. The following values may be set:
### Value | Meaning
---|---
**SH_COMPAT** | Set compatibility mode.
**SH_DENYRW** | Prevent read or write access to the file.
**SH_DENYWR** | Prevent write access of the file.
**SH_DENYRD** | Prevent read access to the file.
**SH_DENYNO** | Permit both read and write access to the file.

You should consult the technical documentation for the DOS system that you are using for more detailed information about these sharing modes.

**Returns:** If successful, `sopen` returns a handle for the file. When an error occurs while opening the file, -1 is returned. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

**Errors:** When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

### Constant | Meaning
---|---
**EACCES** | Access denied because `path` specifies a directory or a volume ID, or sharing mode denied due to a conflicting open.
**EMFILE** | No more handles available (too many open files)
**ENOENT** | Path or file not found

**See Also:** `chsize`, `close`, `creat`, `dup`, `dup2`, `eof`, `exec...`, `fdopen`, `filelength`, `fileno`, `fstat`, `_grow_handles`, `isatty`, `lseek`, `open`, `read`, `setmode`, `stat`, `tell`, `write`, `umask`

**Example:**
```c
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <share.h>

void main( void )
{
    int handle;
```
/* open a file for output */
/* replace existing file if it exists */

handle = sopen("file",
    O_WRONLY | O_CREAT | O_TRUNC,
    SH_DENYWR,
    S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP);

/* read a file which is assumed to exist */
handle = sopen("file", O_RDONLY, SH_DENYWR);

/* append to the end of an existing file */
/* write a new file if file does not exist */

handle = sopen("file",
    O_WRONLY | O_CREAT | O_APPEND,
    SH_DENYWR,
    S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP);

Classification: WATCOM

Systems:
sopen - All, Netware
_sopen - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_wsopen - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  #include <i86.h>
void sound( unsigned frequency );

Description:  The sound function turns on the PC’s speaker at the specified frequency. The frequency is in Hertz (cycles per second). The speaker can be turned off by calling the nosound function after an appropriate amount of time.

Returns: The sound function has no return value.

See Also: delay, nosound

Example:  #include <i86.h>

/*
The numbers in this table are the timer divisors necessary to produce the pitch indicated in the lowest octave that is supported by the "sound" function.

To raise the pitch by N octaves, simply divide the number in the table by 2**N since a pitch which is an octave above another has double the frequency of the original pitch.

The frequency obtained by these numbers is given by 1193180 / X where X is the number obtained in the table.
*/

unsigned short Notes[] = {
  19327,    /* C b */
  18242,    /* C */
  17218,    /* C # ( D b ) */
  16252,    /* D */
  15340,    /* D # ( E b ) */
  14479,    /* E ( F b ) */
  13666,    /* F ( E # ) */
  12899,    /* F # ( G b ) */
  12175,    /* G */
  11492,    /* G # ( A b ) */
  10847,    /* A */
  10238,    /* A # ( B b ) */
  9664,     /* B ( C b ) */
  9121,     /* B # */
  0
};
#define FACTOR 1193180
#define OCTAVE 4

void main() /* play the scale */
{
    int i;
    for( i = 0; Notes[i]; ++i ) {
        sound( FACTOR / (Notes[i] / (1 << OCTAVE)) );
        delay( 200 );
        nosound();
    }
}

Classification: Intel

Systems: DOS, Windows, Win386, QNX
The `spawn...` functions create and execute a new child process, named by `pgm`. The value of `mode` determines how the program is loaded and how the invoking program will behave after the invoked program is initiated:

### Mode | Meaning
--- | ---
**P_WAIT** | The invoked program is loaded into available memory, is executed, and then the original program resumes execution. This option is supported under DOS, OS/2, Win32 and QNX.

**P_NOWAIT** | Causes the current program to execute concurrently with the new child process. This option is supported under OS/2, Win32 and QNX.

**P_NOWAITO** | Causes the current program to execute concurrently with the new child process. This option is supported under OS/2, Win32 and QNX. The `wait` and `cwait` functions cannot be used to obtain the exit code.

**P_OVERLAY** | The invoked program replaces the original program in memory and is executed. No return is made to the original program. This option is supported under DOS (16-bit only), OS/2, Win32, and QNX. This is equivalent to calling the appropriate `exec...` function.

The program is located by using the following logic in sequence:
1. An attempt is made to locate the program in the current working directory if no directory specification precedes the program name; otherwise, an attempt is made in the specified directory.

2. If no file extension is given, an attempt is made to find the program name, in the directory indicated in the first point, with .COM concatenated to the end of the program name.

3. If no file extension is given, an attempt is made to find the program name, in the directory indicated in the first point, with .EXE concatenated to the end of the program name.

4. When no directory specification is given as part of the program name, the spawnlp, spawnlpe, spawnvp, and spawnvpe functions will repeat the preceding three steps for each of the directories specified by the PATH environment variable. The command

   path c:\myapps;d:\lib\applns

indicates that the two directories

   c:\myapps
   d:\lib\applns

are to be searched. The DOS PATH command (without any directory specification) will cause the current path definition to be displayed.

An error is detected when the program cannot be found.

Arguments are passed to the child process by supplying one or more pointers to character strings as arguments in the spawn... call. These character strings are concatenated with spaces inserted to separate the arguments to form one argument string for the child process. The length of this concatenated string must not exceed 128 bytes for DOS systems.

The arguments may be passed as a list of arguments (spawnl, spawnle, spawnlp and spawnlpe) or as a vector of pointers (spawnv, spawnve, spawnvp, and spawnvpe). At least one argument, arg0 or argv[0], must be passed to the child process. By convention, this first argument is a pointer to the name of the program.

If the arguments are passed as a list, there must be a NULL pointer to mark the end of the argument list. Similarly, if a pointer to an argument vector is passed, the argument vector must be terminated by a NULL pointer.

The environment for the invoked program is inherited from the parent process when you use the spawnl, spawnlp, spawnv and spawnvp functions. The spawnle, spawnlpe, spawnve and spawnvpe functions allow a different environment to be passed to the child process through the envp argument. The argument envp is a pointer to an array of character pointers, each of which points to a string defining an environment variable. The array is terminated with a NULL pointer. Each pointer locates a character string of the form

   variable=value

that is used to define an environment variable. If the value of envp is NULL, then the child process inherits the environment of the parent process.

The environment is the collection of environment variables whose values that have been defined with the DOS SET command or by the successful execution of the putenv function. A program may read
spawn... Functions

these values with the getenv function. The wide-character _wspawnl, _wspawnle, _wspawnlp, _wspawnlpe, _wspawnv, _wspawnve, _wspawnvp and _wspawnvpe functions are similar to their counterparts but operate on wide-character strings.

The following example invokes "myprog" as if `myprog ARG1 ARG2` had been entered as a command to DOS.

```c
spawnl( P_WAIT, "myprog",
       "myprog", "ARG1", "ARG2", NULL );
```

The program will be found if one of "myprog.", "myprog.com", or "myprog.exe" is found in the current working directory.

The following example includes a new environment for "myprog".

```c
char *env_list[] = { "SOURCE=MYDATA",
                   "TARGET=OUTPUT",
                   "lines=65",
                   NULL
};

spawnle( P_WAIT, "myprog",
         "myprog", "ARG1", "ARG2", NULL,
             env_list );
```

The environment for the invoked program will consist of the three environment variables SOURCE, TARGET and lines.

The following example is another variation on the first example.

```c
char *arg_list[] = { "myprog", "ARG1", "ARG2", NULL };

spawnv( P_WAIT, "myprog", arg_list );
```

**Returns:**
When the value of mode is:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_WAIT</td>
<td>then the return value from spawn... is the exit status of the child process.</td>
</tr>
<tr>
<td>P_NOWAIT</td>
<td>then the return value from spawn... is the process id (or process handle under Win32) of the child process. To obtain the exit code for a process spawned with P_NOWAIT, you must call the wait (under OS/2 or QNX) or cwait (under OS/2 or Win32) function specifying the process id/handle. If the child process terminated normally, then the low order byte of the returned status word will be set to 0, and the high order byte will contain the low order byte of the return code that the child process passed to the DOSEXIT function.</td>
</tr>
<tr>
<td>P_NOWAITO</td>
<td>then the return value from spawn... is the process id of the child process. The exit code cannot be obtained for a process spawned with P_NOWAITO.</td>
</tr>
</tbody>
</table>

When an error is detected while invoking the indicated program, spawn... returns -1 and errno is set to indicate the error.
Errors: When an error has occurred, \texttt{errno} contains a value indicating the type of error that has been detected.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{E2BIG}</td>
<td>The argument list exceeds 128 bytes, or the space required for the environment information exceeds 32K.</td>
</tr>
<tr>
<td>\texttt{EINVAL}</td>
<td>The \texttt{mode} argument is invalid.</td>
</tr>
<tr>
<td>\texttt{ENOENT}</td>
<td>Path or file not found</td>
</tr>
<tr>
<td>\texttt{ENOMEM}</td>
<td>Not enough memory is available to execute the child process.</td>
</tr>
</tbody>
</table>

See Also: \texttt{abort, atexit, cwait, exec..., exit, _exit, getcmd, getenv, main, putenv, system, wait}

Example:

```c
#include <stdio.h>
#include <stdlib.h>
#include <process.h>
#include <errno.h>
#include <string.h>

void main()
{
    int process_id;
    #if defined(__OS2__) || defined(__NT__)
        int status, rc;
    #endif

    process_id = spawnl( P_NOWAIT, "child.exe", "child", "5", NULL );
    if( process_id == -1 ) {
        printf( "spawn failed - %s\n", strerror( errno ) );
        exit( EXIT_FAILURE );
    }
    printf( "Process id = %d\n", process_id );
}
```


#if defined(__OS2__) || defined(__NT__)  
rc = cwait( &status, process_id, WAIT_CHILD );  
if( rc == -1 ) {  
    printf( "wait failed - %s\n", strerror( errno ) );  
} else {  
    printf( "wait succeeded - %x\n", status );  
    switch( status & 0xff ) {  
    case 0:  
        printf( "Normal termination exit code = %d\n",  
            status >> 8 );  
        break;  
    case 1:  
        printf( "Hard-error abort\n" );  
        break;  
    case 2:  
        printf( "Trap operation\n" );  
        break;  
    case 3:  
        printf( "SIGTERM signal not intercepted\n" );  
        break;  
    default:  
        printf( "Bogus return status\n" );  
    }  
}  
#else  
printf( "spawn completed\n" );  
#endif

/*
[child.c]
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>

void main( int argc, char *argv[] )
{
    int delay;

    if( argc <= 1 )
        exit( EXIT_FAILURE );
    delay = atoi( argv[1] );
    printf( "I am a child going to sleep "  
        "for %d seconds\n", delay );  
    sleep( delay );
    printf( "I am a child awakening\n" );
    exit( 123 );
}
*/

Classification: WATCOM

Systems:  
spawnl - DOS, Win32, QNX, OS/2 1.x(all), OS/2-32  
spawnle - DOS, Win32, QNX, OS/2 1.x(all), OS/2-32  
spawnlp - DOS, Win32, QNX, OS/2 1.x(all), OS/2-32, Netware  
spawnlpe - DOS, Win32, QNX, OS/2 1.x(all), OS/2-32  
spawnv - DOS, Win32, QNX, OS/2 1.x(all), OS/2-32  
spawnvpe - DOS, Win32, QNX, OS/2 1.x(all), OS/2-32
spawnvp - DOS, Win32, QNX, OS/2 1.x(all), OS/2-32, Netware
spawnvpe - DOS, Win32, QNX, OS/2 1.x(all), OS/2-32
_wspawnl - DOS, Win32, OS/2 1.x(all), OS/2-32
_wspawnle - DOS, Win32, OS/2 1.x(all), OS/2-32
_wspawnlp - DOS, Win32, OS/2 1.x(all), OS/2-32
_wspawnlpe - DOS, Win32, OS/2 1.x(all), OS/2-32
_wspawnv - DOS, Win32, OS/2 1.x(all), OS/2-32
_wspawnve - DOS, Win32, OS/2 1.x(all), OS/2-32
_wspawnvp - DOS, Win32, OS/2 1.x(all), OS/2-32
_wspawnvpe - DOS, Win32, OS/2 1.x(all), OS/2-32
**Synopsis:**

```c
#include <stdlib.h>
void _splitpath( const char *path,
              char *drive,
              char *dir,
              char *fname,
              char *ext );
void _wsplitpath( const wchar_t *path,
                 wchar_t *drive,
                 wchar_t *dir,
                 wchar_t *fname,
                 wchar_t *ext );
```

**Description:**

The `_splitpath` function splits up a full pathname into four components consisting of a drive letter, directory path, file name and file name extension. The argument `path` points to a buffer containing the full pathname to be split up.

The `_wsplitpath` function is a wide-character version of `_splitpath` that operates with wide-character strings.

The maximum size required for each buffer is specified by the manifest constants `_MAX_PATH`, `_MAX_DRIVE` (or `_MAX_VOLUME` for Netware applications), `_MAX_DIR`, `_MAX_FNAME`, and `_MAX_EXT` which are defined in `<stdlib.h>`.

- **drive**: The `drive` argument points to a buffer that will be filled in with the drive letter (e.g., A, B, C, etc.) followed by a colon if a drive is specified in the full pathname (filled in by `_splitpath`).

  For Netware applications, the `drive` argument points to a buffer that will be filled in with the volume identifier (e.g., `\NAME_SPACE`) if a volume is specified in the full pathname (filled in by `_splitpath`).

- **dir**: The `dir` argument points to a buffer that will be filled in with the pathname including the trailing slash. Either forward slashes (`/`) or backslashes (`\`) may be used.

- **fname**: The `fname` argument points to a buffer that will be filled in with the base name of the file without any extension (suffix) if a file name is specified in the full pathname (filled in by `_splitpath`).

- **ext**: The `ext` argument points to a buffer that will be filled in with the filename extension (suffix) including the leading period if an extension is specified in the full pathname (filled in by `_splitpath`).

The arguments `drive`, `dir`, `fname` and `ext` will not be filled in if they are NULL pointers.

For each component of the full pathname that is not present, its corresponding buffer will be set to an empty string.

**Returns:**

The `_splitpath` function returns no value.

**See Also:** `_fullpath`, `_makepath`, `_splitpath2`

---

902 Library Functions and Macros
Example:  

```c
#include <stdio.h>
#include <stdlib.h>

void main()
{
    char full_path[ _MAX_PATH ];
    char drive[ _MAX_DRIVE ];
    char dir[ _MAX_DIR ];
    char fname[ _MAX_FNAME ];
    char ext[ _MAX_EXT ];

    _makepath(full_path,"c","watcom\h\","stdio","h");
    printf( "Full path is: %s\n\n", full_path );
    _splitpath( full_path, drive, dir, fname, ext );
    printf( "Components after _splitpath\n" );
    printf( "drive: %s\n", drive );
    printf( "dir:   %s\n", dir );
    printf( "fname: %s\n", fname );
    printf( "ext:   %s\n", ext );
}
```

produces the following:

Full path is: c:\watcom\h\stdio.h

Components after _splitpath
drive: c:
    dir:   watcom\h\n    fname: stdio
    ext:   .h

Note the use of two adjacent backslash characters (\) within character-string constants to signify a single backslash.

Classification: WATCOM

Systems:  
    _splitpath - All, Netware
    _wsplitpath - All
Synopsis:
#include <stdlib.h>
void _splitpath2( const char *inp,
                char *outp,
                char **drive,
                char **dir,
                char **fname,
                char **ext );
void _wsplitpath2( const wchar_t *inp,
                  wchar_t *outp,
                  wchar_t **drive,
                  wchar_t **dir,
                  wchar_t **fname,
                  wchar_t **ext );

Description: The _splitpath2 function splits up a full pathname into four components consisting of a drive letter, directory path, file name and file name extension.

inp The argument inp points to a buffer containing the full pathname to be split up.

outp The argument outp points to a buffer that will contain all the components of the path, each separated by a null character. The maximum size required for this buffer is specified by the manifest constant _MAX_PATH2 which is defined in <stdlib.h>.

drive The drive argument is the location that is to contain the pointer to the drive letter (e.g., A, B, C, etc.) followed by a colon if a drive is specified in the full pathname (filled in by _splitpath2).

dir The dir argument is the location that is to contain the pointer to the directory path including the trailing slash if a directory path is specified in the full pathname (filled in by _splitpath2). Either forward slashes (/) or backslashes (\) may be used.

fname The fname argument is the location that is to contain the pointer to the base name of the file without any extension (suffix) if a file name is specified in the full pathname (filled in by _splitpath2).

ext The ext argument is the location that is to contain the pointer to the filename extension (suffix) including the leading period if an extension is specified in the full pathname (filled in by _splitpath2).

The arguments drive, dir, fname and ext will not be filled in if they are NULL pointers.

For each component of the full pathname that is not present, its corresponding pointer will be set to point at a NULL string (\'\0\').

This function reduces the amount of memory space required when compared to the splitpath function.

The _wsplitpath2 function is a wide-character version of _splitpath2 that operates with wide-character strings.

Returns: The _splitpath2 function returns no value.

See Also: _fullpath, _makepath, _splitpath
Example:  
#include <stdio.h>
#include <stdlib.h>

void main()
{
    char full_path[ _MAX_PATH ];
    char tmp_path[ _MAX_PATH2 ];
    char *drive;
    char *dir;
    char *fname;
    char *ext;

    _makepath(full_path,"c","watcom\h","stdio","h");
    printf( "Full path is: %s\n\n", full_path );
    _splitpath2(full_path, tmp_path,
                &drive, &dir, &fname, &ext );
    printf( "Components after _splitpath2\n" );
    printf( "drive: %s\n", drive );
    printf( "dir: %s\n", dir );
    printf( "fname: %s\n", fname );
    printf( "ext: %s\n", ext );
}

produces the following:

Full path is: c:watcom\h\stdio.h

Components after _splitpath2
drive: c:
dir: watcom\h
fname: stdio
ext: .h

Note the use of two adjacent backslash characters (\) within character-string constants to signify a single backslash.

Classification: WATCOM

Systems:  
_splitpath2 - All
_wsplitpath2 - All
**Synopsis:**

```c
#include <stdio.h>
int sprintf( char *buf, const char *format, ... );

#include <wchar.h>
int swprintf( wchar_t *buf, size_t n, const wchar_t *format, ... );
```

**Safer C:**

The Safer C Library extension provides the `sprintf_s` function which is a safer alternative to `sprintf`. This newer `sprintf_s` function is recommended to be used instead of the traditional "unsafe" `sprintf` function.

**Description:**

The `sprintf` function is equivalent to the `fprintf` function, except that the argument `buf` specifies a character array into which the generated output is placed, rather than to a file. A null character is placed at the end of the generated character string. The `format` string is described under the description of the `printf` function.

The `swprintf` function is identical to `sprintf` except that the argument `buf` specifies an array of wide characters into which the generated output is to be written, rather than converted to multibyte characters and written to a stream. The maximum number of wide characters to write, including a terminating null wide character, is specified by `n`. The `swprintf` function accepts a wide-character string argument for `format`.

**Returns:**

The `sprintf` function returns the number of characters written into the array, not counting the terminating null character. An error can occur while converting a value for output. The `swprintf` function returns the number of wide characters written into the array, not counting the terminating null wide character, or a negative value if `n` or more wide characters were requested to be generated. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

**See Also:** `_bprintf`, `cprintf`, `fprintf`, `printf`, `_vbprintf`, `vcprintf`, `vfprintf`, `vprintf`, `vsnprintf`

**Example:**

```c
#include <stdio.h>

/* Create temporary file names using a counter */

char namebuf[13];
int TempCount = 0;

char *make_temp_name( void )
{
    sprintf( namebuf, "zz%.6o.tmp", TempCount++ );
    return( namebuf );
}

void main( void )
{
    FILE *tf1, *tf2;
```
tf1 = fopen( make_temp_name(), "w" );
.tf2 = fopen( make_temp_name(), "w" );
.fputs( "temp file 1", tf1 );
.fputs( "temp file 2", tf2 );
.fclose( tf1 );
.fclose( tf2 );
}

**Classification:** sprintf is ANSI
swprintf is ANSI

**Systems:** sprintf - All, Netware
swprintf - All
**Synopsis:**
```c
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
int sprintf_s( char * restrict s, rsize_t n
const char * restrict format, ... );
#include <wchar.h>
int swprintf_s( char * restrict s, rsize_t n,
const wchar_t * restrict format, ... );
```

**Constraints:**
If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `sprintf_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither `s` nor `format` shall be a null pointer. The `n` argument shall neither equal zero nor be greater than `RSIZE_MAX`. The number of characters (including the trailing null) required for the result to be written to the array pointed to by `s` shall not be greater than `n`. The `%n` specifier (modified or not by flags, field width, or precision) shall not appear in the string pointed to by `format`. Any argument to `sprintf_s` corresponding to a `%s` specifier shall not be a null pointer. No encoding error shall occur.

If there is a runtime-constraint violation, then if `s` is not a null pointer and `n` is greater than zero and less than `RSIZE_MAX`, then the `sprintf_s` function sets `s[0]` to the null character.

**Description:**
The `sprintf_s` function is equivalent to the `sprintf` function except for the explicit runtime-constraints listed above.

The `sprintf_s` function, unlike `snprintf_s`, treats a result too big for the array pointed to by `s` as a runtime-constraint violation.

The `swprintf_s` function is identical to `sprintf_s` except that it accepts a wide-character string argument for `format` and produces wide character output.

**Returns:**
If no runtime-constraint violation occurred, the `sprintf_s` function returns the number of characters written in the array, not counting the terminating null character. If an encoding error occurred, `sprintf_s` returns a negative value. If any other runtime-constraint violation occurred, `sprintf_s` returns zero.

If no runtime-constraint violation occurred, the `swprintf_s` function returns the number of wide characters written in the array, not counting the terminating null wide character. If an encoding error occurred or if `n` or more wide characters are requested to be written, `swprintf_s` returns a negative value. If any other runtime-constraint violation occurred, `swprintf_s` returns zero.

**See Also:** `_bprintf`, `cprintf`, `fprintf`, `printf`, `sprintf`, `_vbprintf`, `vcprintf`, `vfprintf`, `vprintf`, `vsprintf`

**Example:**
```c
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
/* Create temporary file names using a counter */

char namebuf[13];
int TempCount = 0;
```
char *make_temp_name( void )
{
    sprintf_s( namebuf, sizeof( namebuf ),
               "zz%.6o.tmp", TempCount++ );
    return( namebuf );
}

void main( void )
{
    FILE *tf1, *tf2;
    tf1 = fopen( make_temp_name(), "w" );
    tf2 = fopen( make_temp_name(), "w" );
    fputs( "temp file 1", tf1 );
    fputs( "temp file 2", tf2 );
    fclose( tf1 );
    fclose( tf2 );
}

Classification: sprintf_s is TR 24731
                swprintf_s is TR 24731

Systems:       sprintf_s - All, Netware
                swprintf_s - All
Synopsis:  
#include <math.h>
double sqrt(double x);

Description:  The sqrt function computes the non-negative square root of x. A domain error occurs if the argument is negative.

Returns:  The sqrt function returns the value of the square root. When the argument is outside the permissible range, the matherr function is called. Unless the default matherr function is replaced, it will set the global variable errno to EDOM, and print a "DOMAIN error" diagnostic message using the stderr stream.

See Also:  exp, log, pow, matherr

Example:  
#include <stdio.h>
#include <math.h>

void main()
{
    printf("%f\n", sqrt(.5));
}

produces the following:

0.707107

Classification:  ANSI

Systems:  Math
Synopsis:  
#include <stdlib.h>  
void srand( unsigned int seed );

Description:  The srand function uses the argument seed to start a new sequence of pseudo-random integers to be returned by subsequent calls to rand. A particular sequence of pseudo-random integers can be repeated by calling srand with the same seed value. The default sequence of pseudo-random integers is selected with a seed value of 1.

Returns:  The srand function returns no value.

See Also:  rand

Example:  
#include <stdio.h>  
#include <stdlib.h>

void main()
{
    int i;

    srand( 982 );
    for( i = 1; i < 10; ++i ) {
        printf( "%d\n", rand() );
    }
    srand( 982 );  /* start sequence over again */
    for( i = 1; i < 10; ++i ) {
        printf( "%d\n", rand() );
    }
}

Classification:  ANSI

Systems:  All, Netware
Synopsis:
```c
#include <stdio.h>
int sscanf( const char *in_string,
            const char *format, ... );
#include <wchar.h>
int swscanf( const wchar_t *in_string,
            const wchar_t *format, ... );
```

Safer C:
The Safer C Library extension provides the `sscanf_s` function which is a safer alternative to `sscanf`. This newer `sscanf_s` function is recommended to be used instead of the traditional "unsafe" `sscanf` function.

Description:
The `sscanf` function scans input from the character string `in_string` under control of the argument `format`. Following the format string is the list of addresses of items to receive values.

The `format` string is described under the description of the `scanf` function.

The `swscanf` function is identical to `sscanf` except that it accepts a wide-character string argument for `format` and the input string `in_string` consists of wide characters.

Returns:
The `sscanf` function returns EOF if the end of the input string was reached before any input conversion. Otherwise, the number of input arguments for which values were successfully scanned and stored is returned.

See Also:
cscanf, fscanf, scanf, vcscanf, vfscanf, vscanf, vsscanf

Example:
```c
#include <stdio.h>

/* Scan a date in the form "Saturday April 18 1987" */

void main( void )
{
    int day, year;
    char weekday[10], month[10];

    scanf( "Friday August 0014 1987",
           "%s %s %d %d",
           weekday, month, &day, &year );
    printf( "%s %s %d %d
",
            weekday, month, day, year );
}
```
produces the following:

Friday August 14 1987

Classification:
`sscanf` is ISO C90
`swscanf` is ISO C95

Systems:
`sscanf` - All, Netware
`swscanf` - All
Synopsis:
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
int sscanf_s( const char * restrict s,
const char * restrict format, ... );
#include <wchar.h>
int swscanf_s( const wchar_t * restrict s,
const wchar_t * restrict format, ... );

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and sscanf_s will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither s nor format shall be a null pointer. Any argument indrected through in order to store converted input shall not be a null pointer.

If there is a runtime-constraint violation, the sscanf_s function does not attempt to perform further input, and it is unspecified to what extent sscanf_s performed input before discovering the runtime-constraint violation.

Description: The sscanf_s function is equivalent to fscanf_s, except that input is obtained from a string (specified by the argument s) rather than from a stream. Reaching the end of the string is equivalent to encountering end-of-file for the fscanf_s function. If copying takes place between objects that overlap, the objects take on unspecified values.

The swscanf_s function is identical to sscanf_s except that it accepts wide-character string arguments for s and format.

Returns: The sscanf_s function returns EOF if an input failure occurred before any conversion or if there was a runtime-constraint violation. Otherwise, the sscanf_s function returns the number of input items successfully assigned, which can be fewer than provided for, or even zero.

See Also: cscanf, fscanf, scanf, sscanf, vcscanf, vfscanf, vscanf, vsscanf

Example:
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>

void main( void )
{
    int day, year;
    char weekday[10], month[10];

    sscanf_s( "Friday August 0013 2004",
        "%s %s %d %d",
        weekday, sizeof( weekday ),
        month, sizeof( month ),
        &day, &year );
    printf_s( "%s %s %d %d\n",
        weekday, month, day, year );
}

produces the following:

Friday August 13 2004
`sscanf_s`, `swscanf_s`

**Classification:**
- `sscanf_s` is TR 24731
- `swscanf_s` is TR 24731

**Systems:**
- `sscanf_s` - All, Netware
- `swscanf_s` - All
Synopsis:

```c
#include <malloc.h>
size_t stackavail( void );
size_t _stackavail( void );
```

Description:
The `stackavail` function returns the number of bytes currently available in the stack. This value is usually used to determine an appropriate amount to allocate using `alloca`.

The `_stackavail` function is identical to `stackavail`. Use `_stackavail` for ANSI/ISO naming conventions.

Returns:
The `stackavail` function returns the number of bytes currently available in the stack.

See Also:
`alloca`, `calloc` Functions, `malloc` Functions

Example:

```c
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <fcntl.h>
#include <io.h>

long char_count( FILE *fp )
{
    char    *buffer;
    size_t  bufsiz;
    long    count;

    /* allocate half of stack for temp buffer */
    bufsiz = stackavail() >> 1;
    buffer = (char *) alloca( bufsiz );
    setvbuf( fp, buffer, _IOFBF, bufsiz );
    count = 0L;
    while( fgetc( fp ) != EOF ) ++count;
    fclose( fp );
    return( count );
}

void main( void )
{
    FILE    *fp;

    fp = fopen( "file", "rb" );
    if( fp != NULL ) {
        setmode( fileno( fp ), O_BINARY );
        printf( "File contains %lu characters\n",
                char_count( fp ) );
        fclose( fp );
    }
}
```

Classification:
`WATCOM`

 `_stackavail` conforms to ANSI/ISO naming conventions

Systems:
`stackavail` - All, Netware
 `_stackavail` - All, Netware
Synopsis:

```
#include <sys/stat.h>
int stat( const char *path, struct stat *buf );
int _stat( const char *path, struct _stat *buf );
int _stati64( const char *path, struct _stati64 *buf );
int _wstat( const wchar_t *path, struct _stat *buf );
int _wstati64( const wchar_t *path, struct _stati64 *buf );
int lstat( const char *path, struct stat *buf );
```

Description:
The `stat` functions obtain information about the file or directory referenced in `path`. This information is placed in the structure located at the address indicated by `buf`.

The file `<sys/stat.h>` contains definitions for the structure `stat`.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type/ Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>st_dev</code></td>
<td>(dev_t) the disk drive the file resides on</td>
</tr>
<tr>
<td><code>st_ino</code></td>
<td>(ino_t) this inode’s number (not used for DOS)</td>
</tr>
<tr>
<td><code>st_mode</code></td>
<td>(unsigned short) file mode</td>
</tr>
<tr>
<td><code>st_nlink</code></td>
<td>(short) number of hard links</td>
</tr>
<tr>
<td><code>st_uid</code></td>
<td>(unsigned long) user-id (always 'root' for DOS)</td>
</tr>
<tr>
<td><code>st_gid</code></td>
<td>(short) group-id (always 'root' for DOS)</td>
</tr>
<tr>
<td><code>st_rdev</code></td>
<td>(dev_t) this should be the device type but it is the same as st_dev for the time being</td>
</tr>
<tr>
<td><code>st_size</code></td>
<td>(off_t) total file size</td>
</tr>
<tr>
<td><code>st_atime</code></td>
<td>(time_t) this should be the file &quot;last accessed&quot; time if the file system supports it</td>
</tr>
<tr>
<td><code>st_mtime</code></td>
<td>(time_t) the file &quot;last modified&quot; time</td>
</tr>
<tr>
<td><code>st_ctime</code></td>
<td>(time_t) this should be the file &quot;last status change&quot; time if the file system supports it</td>
</tr>
</tbody>
</table>

The following fields are Netware only:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type/ Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>st_btime</code></td>
<td>(time_t) the file &quot;last archived&quot; time</td>
</tr>
<tr>
<td><code>st_attr</code></td>
<td>(unsigned long) the file’s attributes</td>
</tr>
<tr>
<td><code>st_archivedID</code></td>
<td>(unsigned long) the user/object ID that last archived file</td>
</tr>
<tr>
<td><code>st_updatedID</code></td>
<td>(unsigned long) the user/object ID that last updated file</td>
</tr>
<tr>
<td><code>st_inheritedRightsMask</code></td>
<td>(unsigned short) the inherited rights mask</td>
</tr>
<tr>
<td><code>st_originatingNamespace</code></td>
<td>(unsigned char) the originating name space</td>
</tr>
</tbody>
</table>

The structure `_stati64` differs from `stat` in the following way:
st_size: (_int64) total file size (as a 64-bit value)

At least the following macros are defined in the `<sys/stat.h>` header file.

<table>
<thead>
<tr>
<th>Macro</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_ISFIFO(m)</td>
<td>Test for FIFO.</td>
</tr>
<tr>
<td>S_ISCHR(m)</td>
<td>Test for character special file.</td>
</tr>
<tr>
<td>S_ISDIR(m)</td>
<td>Test for directory file.</td>
</tr>
<tr>
<td>S_ISBLK(m)</td>
<td>Test for block special file.</td>
</tr>
<tr>
<td>S_ISREG(m)</td>
<td>Test for regular file.</td>
</tr>
</tbody>
</table>

The value `m` supplied to the macros is the value of the `st_mode` field of a `stat` structure. The macro evaluates to a non-zero value if the test is true and zero if the test is false.

The following bits are encoded within the `st_mode` field of a `stat` structure.

<table>
<thead>
<tr>
<th>Mask</th>
<th>Owner Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_IRWXU</td>
<td>Read, write, search (if a directory), or execute (otherwise)</td>
</tr>
<tr>
<td>S_IRUSR</td>
<td>Read permission bit</td>
</tr>
<tr>
<td>S_IWUSR</td>
<td>Write permission bit</td>
</tr>
<tr>
<td>S_IXUSR</td>
<td>Search/execute permission bit</td>
</tr>
<tr>
<td>S_IREAD</td>
<td>== S_IRUSR (for Microsoft compatibility)</td>
</tr>
<tr>
<td>S_IWRITE</td>
<td>== S_IWUSR (for Microsoft compatibility)</td>
</tr>
<tr>
<td>S_IEXEC</td>
<td>== S_IXUSR (for Microsoft compatibility)</td>
</tr>
</tbody>
</table>

S_IRWXU is the bitwise inclusive OR of S_IRUSR, S_IWUSR, and S_IXUSR.

<table>
<thead>
<tr>
<th>Mask</th>
<th>Group Permissions (same as owner's on DOS, OS/2 or Windows)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_IRWXG</td>
<td>Read, write, search (if a directory), or execute (otherwise)</td>
</tr>
<tr>
<td>S_IRGRP</td>
<td>Read permission bit</td>
</tr>
<tr>
<td>S_IWGRP</td>
<td>Write permission bit</td>
</tr>
<tr>
<td>S_IXGRP</td>
<td>Search/execute permission bit</td>
</tr>
</tbody>
</table>

S_IRWXG is the bitwise inclusive OR of S_IRGRP, S_IWGRP, and S_IXGRP.

<table>
<thead>
<tr>
<th>Mask</th>
<th>Other Permissions (same as owner's on DOS, OS/2 or Windows)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_IRWXO</td>
<td>Read, write, search (if a directory), or execute (otherwise)</td>
</tr>
<tr>
<td>S_IROTH</td>
<td>Read permission bit</td>
</tr>
<tr>
<td>S_IWOTH</td>
<td>Write permission bit</td>
</tr>
<tr>
<td>S_IXOTH</td>
<td>Search/execute permission bit</td>
</tr>
</tbody>
</table>

S_IRWXO is the bitwise inclusive OR of S_IROTH, S_IWOTH, and S_IXOTH.
### stat, _stat, _stati64, _wstat, _wstati64, lstat

<table>
<thead>
<tr>
<th>Mask</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_ISUID</td>
<td>(Not supported by DOS, OS/2 or Windows) Set user ID on execution. The process’s effective user ID shall be set to that of the owner of the file when the file is run as a program. On a regular file, this bit should be cleared on any write.</td>
</tr>
<tr>
<td>S_ISGID</td>
<td>(Not supported by DOS, OS/2 or Windows) Set group ID on execution. Set effective group ID on the process to the file’s group when the file is run as a program. On a regular file, this bit should be cleared on any write.</td>
</tr>
</tbody>
</table>

The _stat function is identical to stat. Use _stat for ANSI/ISO naming conventions. The _stati64, _wstat, and _wstati64 functions differ from stat in the type of structure that they are asked to fill in. The _wstat and _wstati64 functions deal with wide character strings. The differences in the structures are described above. The lstat function is identical to stat on non-UNIX platforms.

### Returns:
All forms of the stat function return zero when the information is successfully obtained. Otherwise, -1 is returned.

### Errors:
When an error has occurred, errno contains a value indicating the type of error that has been detected.

#### EACCESS
Search permission is denied for a component of path.

### See Also:
fstat

### Example:
```c
#include <stdio.h>
#include <sys/stat.h>

void main()
{
    struct stat buf;

    if( stat( "file", &buf ) != -1 ) {
        printf( "File size = %d\n", buf.st_size );
    }
}
```

### Classification:
stat is POSIX
_stat is not POSIX
_stati64 is not POSIX
_wstat is not POSIX
_wstati64 is not POSIX
_stat conforms to ANSI/ISO naming conventions

### Systems:
stat - All, Netware
_stat - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_stati64 - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_wstat - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_wstati64 - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
lstat - All, Netware
Synopsis:  
#include <float.h>
unsigned int _status87( void );

Description:  The _status87 function returns the floating-point status word which is used to record the status of 8087/80287/80387/80486 floating-point operations.

Returns:  The _status87 function returns the floating-point status word which is used to record the status of 8087/80287/80387/80486 floating-point operations. The description of this status is found in the <float.h> header file.

See Also:  _clear87, _control87, _controlfp, _finite, _fpreset

Example:  
#include <stdio.h>
#include <float.h>
#define TEST_FPU(x,y) printf( "\t%s " y "\n", 
        ((fp_status & x) ? " " : "No") )

void main()
{
    unsigned int fp_status;
    fp_status = _status87();

    printf( "80x87 status\n" );
    TEST_FPU( SW_INVALID, "invalid operation" );
    TEST_FPU( SW_DENORMAL, "denormalized operand" );
    TEST_FPU( SW_ZERODIVIDE, "divide by zero" );
    TEST_FPU( SW_OVERFLOW, "overflow" );
    TEST_FPU( SW_UNDERFLOW, "underflow" );
    TEST_FPU( SW_INEXACT, "inexact result" );
}

Classification:  Intel

Systems:  Math
**strcasecmp**

**Synopsis:**
```c
#include <strings.h>
int strcasecmp( const char *s1, const char *s2 );
```

**Description:** The `strcasecmp` function compares, with case insensitivity, the string pointed to by `s1` to the string pointed to by `s2`. All uppercase characters from `s1` and `s2` are mapped to lowercase for the purposes of doing the comparison.

The `strcasecmp` function is identical to the `stricmp` function.

**Returns:** The `strcasecmp` function returns an integer less than, equal to, or greater than zero, indicating that the string pointed to by `s1` is, ignoring case, less than, equal to, or greater than the string pointed to by `s2`.

**See Also:** `strcmp`, `strcmpi`, `stricmp`, `strncmp`, `strnicmp`, `strncasecmp`

**Example:**
```c
#include <stdio.h>
#include <strings.h>

int main( void )
{
    printf( "%d\n", strcasecmp( "AbCDEF", "abcdef" ) );
    printf( "%d\n", strcasecmp( "abcdef", "ABC" ) );
    printf( "%d\n", strcasecmp( "abc", "ABCdef" ) );
    printf( "%d\n", strcasecmp( "Abcdef", "mnopqr" ) );
    printf( "%d\n", strcasecmp( "Mnopqr", "abcdef" ) );
    return( 0 );
}
```

produces the following:

```
0
100
-100
-12
12
```

**Classification:** POSIX

**Systems:** All, Netware
Synopsis:

```
#include <string.h>
char *strcat( char *dst, const char *src );
char __far *__fstrcat( char __far *dst,
const char __far *src );

#include <wchar.h>
wchar_t *wcscat( wchar_t *dst, const wchar_t *src );

#include <mbstring.h>
unsigned char *__mbscat( unsigned char *dst,
const unsigned char *src );
unsigned char __far *__mbncat( unsigned char __far *dst,
const unsigned char __far *src );
```

Safer C:
The Safer C Library extension provides the `strcat_s` function which is a safer alternative to `strcat`. This newer `strcat_s` function is recommended to be used instead of the traditional "unsafe" `strcat` function.

Description:
The `strcat` function appends a copy of the string pointed to by `src` (including the terminating null character) to the end of the string pointed to by `dst`. The first character of `src` overwrites the null character at the end of `dst`.

The `_fstrcat` function is a data model independent form of the `strcat` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `wcscat` function is a wide-character version of `strcat` that operates with wide-character strings.

The `_mbscat` function is a multibyte character version of `strcat` that operates with multibyte character strings.

Returns:
The value of `dst` is returned.

See Also:
`strncat, strcat_s, strncat_s`

Example:
```
#include <stdio.h>
#include <string.h>

void main()
{
    char buffer[80];
    memset(buffer, 0, 80);
    strcat(buffer, "Hello ");
    printf("%s\n", buffer);
}
```

produces the following:

```
Hello world
```

Classification:
- `strcat` is ANSI
- `_fstrcat` is not ANSI
- `wcscat` is ANSI
- `_mbscat` is not ANSI
- `_fmbscat` is not ANSI
strcat, _fstrcat, wcscat, _mbscat, _fmbscat

Systems:

strcat - All, Netware
_fstrcat - All
wcscat - All
__mbscat - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
__fmbscat - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

```c
#define __STDC_WANT_LIB_EXT1__  1
#include <string.h>
errno_t strcat_s( char * restrict s1,
    rsize_t slmax,
    const char * restrict s2 );
#include <wchar.h>
errno_t wcscat_s( wchar_t * restrict s1,
    rsize_t slmax,
    const wchar_t * restrict s2 );
```

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `strcat_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Let $m$ denote the value $slmax - strnlen_s(s1, slmax)$ upon entry to `strcat_s`. Neither $s1$ nor $s2$ shall be a null pointer. $slmax$ shall not be greater than `RSIZE_MAX`. $slmax$ shall not equal zero. $m$ shall not equal zero. $m$ shall be greater than $strnlen_s(s2, m)$. Copying shall not take place between objects that overlap.

If there is a runtime-constraint violation, then if $s1$ is not a null pointer and $slmax$ is greater than zero and not greater than `RSIZE_MAX`, then `strcat_s` sets $s1[0]$ to the null character.

Description: The `strcat_s` function appends a copy of the string pointed to by $s2$ (including the terminating null character) to the end of the string pointed to by $s1$. The initial character from $s2$ overwrites the null character at the end of $s1$. All elements following the terminating null character (if any) written by `strcat_s` in the array of $slmax$ characters pointed to by $s1$ take unspecified values when `strcat_s` returns.

The `wcscat_s` function is a wide-character version of `strcat_s` that operates with wide-character strings.

Returns: The `strcat_s` function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

See Also: `strcat`, `strncat`, `strncat_s`

Example:

```c
#define __STDC_WANT_LIB_EXT1__  1
#include <stdio.h>
#include <string.h>

void main( void )
{
    char buffer[80];
    strcpy_s( buffer, sizeof( buffer ), "Hello " );
    strcat_s( buffer, sizeof( buffer ), "world" );
    printf( "%s\n", buffer );
}
```

produces the following:

```
Hello world
```

Classification: `strcat_s` is TR 24731
**strcat_s, wcscat_s**

wcscat_s is TR 24731

**Systems:**
- strcat_s - All, Netware
- wcscat_s - All
Synopsis:

```c
#include <string.h>
char *strchr( const char *s, int c );
char __far *__fstrchr( const char __far *s, int c );
#include <wchar.h>
wchar_t *wcschr( const wchar_t *s, int c );
#include <mbstring.h>
unsigned char *__mbstrchr( const unsigned char *s,
                             unsigned int c );
unsigned char __far *__fmbschr( const unsigned char __far *s,
                                unsigned int c );
```

Description: The `strchr` function locates the first occurrence of `c` (converted to a char) in the string pointed to by `s`. The terminating null character is considered to be part of the string.

The `_fstrchr` function is a data model independent form of the `strchr` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `wcschr` function is a wide-character version of `strchr` that operates with wide-character strings.

The `__mbstrchr` function is a multibyte character version of `strchr` that operates with multibyte character strings.

Returns: The `strchr` function returns a pointer to the located character, or NULL if the character does not occur in the string.

See Also: `memchr`, `strcspn`, `strstr`, `strspn`, `strstr`, `strtok`

Example:

```c
#include <stdio.h>
#include <string.h>

void main()
{
    char buffer[80];
    char *where;
    strcpy( buffer, "video x-rays" );
    where = strchr( buffer, 'x' );
    if( where == NULL ) {
        printf( "'x' not found\n" );
    }
}
```

Classification: `strchr` is ANSI
`_fstrchr` is not ANSI
`wcschr` is ANSI
`__mbstrchr` is not ANSI
`__fmbschr` is not ANSI

Systems: `strchr` - All, Netware
`_fstrchr` - All
`wcschr` - All
`__mbstrchr` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
`__fmbschr` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:
#include <string.h>
int strcmp( const char *s1, const char *s2 );
int _fstrcmp( const char __far *s1,
             const char __far *s2 );
#include <wchar.h>
int wcscmp( const wchar_t *s1, const wchar_t *s2 );
#include <mbstring.h>
int _mbscmp( const unsigned char *s1,
             const unsigned char *s2 );
int _fmbscmp( const unsigned char __far *s1,
             const unsigned char __far *s2 );

Description:
The strcmp function compares the string pointed to by s1 to the string pointed to by s2.

The _fstrcmp function is a data model independent form of the strcmp function that accepts far
pointer arguments. It is most useful in mixed memory model applications.

The wcscmp function is a wide-character version of strcmp that operates with wide-character strings.

The _mbscmp function is a multibyte character version of strcmp that operates with multibyte
character strings.

Returns:
The strcmp function returns an integer less than, equal to, or greater than zero, indicating that the
string pointed to by s1 is less than, equal to, or greater than the string pointed to by s2.

See Also: strcmpi, stricmp, strncmp, strnicmp

Example:
#include <stdio.h>
#include <string.h>

void main()
{
    printf( "%d\n", strcmp( "abcdef", "abcdef" ) );
    printf( "%d\n", strcmp( "abcdef", "abc" ) );
    printf( "%d\n", strcmp( "abc", "abcdef" ) );
    printf( "%d\n", strcmp( "abcdef", "mnopqr" ) );
    printf( "%d\n", strcmp( "mnopqr", "abcdef" ) );
}

produces the following:

0
1
-1
-1
1

Classification: strcmp is ANSI
_fstrcmp is not ANSI
wcscmp is ANSI
_mbscmp is not ANSI
_fmbscmp is not ANSI

Systems: strcmp - All, Netware
_fstrcmp - All
wcscmp - All
_mbscmp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fmbscmp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
**Synopsis:**

```
#include <string.h>
int strcmpi( const char *s1, const char *s2 );
int wcsncmpi ( const wchar_t *s1, const wchar_t *s2 );
```

**Description:**

The `strcmpi` function compares, with case insensitivity, the string pointed to by `s1` to the string pointed to by `s2`. All uppercase characters from `s1` and `s2` are mapped to lowercase for the purposes of doing the comparison. The `strcmpi` function is identical to the `stricmp` function.

The `wcsncmpi` function is a wide-character version of `strcmpi` that operates with wide-character strings.

**Returns:**

The `strcmpi` function returns an integer less than, equal to, or greater than zero, indicating that the string pointed to by `s1` is less than, equal to, or greater than the string pointed to by `s2`.

**See Also:**

`strcmp`, `stricmp`, `strncmp`, `strnicmp`

**Example:**

```
#include <stdio.h>
#include <string.h>

void main()
{
    printf( "%d\n", strcmpi( "AbCDEF", "abcdef" ) );
    printf( "%d\n", strcmpi( "abcdef", "ABC" ) );
    printf( "%d\n", strcmpi( "abc", "ABCdef" ) );
    printf( "%d\n", strcmpi( "Abcdef", "mnopqr" ) );
    printf( "%d\n", strcmpi( "Mnopqr", "abcdef" ) );
}
```

produces the following:

```
0
100
-100
-12
12
```

**Classification:** WATCOM

**Systems:**

- `strcmpi` - All, Netware
- `wcsncmpi` - All
Synopsis:
#include <string.h>
int strcoll( const char *s1, const char *s2 );
#include <wchar.h>
int wcscoll( const wchar_t *s1, const wchar_t *s2 );
#include <mbstring.h>
int _mbscoll( const unsigned char *s1, const unsigned char *s2 );

Description: The strcoll function compares the string pointed to by s1 to the string pointed to by s2. The comparison uses the collating sequence selected by the setlocale function. The function will be equivalent to the strcmp function when the collating sequence is selected from the "C" locale.

The wcscoll function is a wide-character version of strcoll that operates with wide-character strings.

The _mbscoll function is a multibyte character version of strcoll that operates with multibyte character strings.

Returns: The strcoll function returns an integer less than, equal to, or greater than zero, indicating that the string pointed to by s1 is less than, equal to, or greater than the string pointed to by s2, according to the collating sequence selected.

See Also: setlocale, strcmp, strncmp

Example:
#include <stdio.h>
#include <string.h>

char buffer[80] = "world";

void main()
{
    if( strcoll( buffer, "Hello" ) < 0 ) {
        printf( "Less than\n" );
    }
}

Classification: strcoll is ANSI
wcscoll is ANSI
_mbscoll is not ANSI

Systems: strcoll - All, Netware
wcscoll - All
_mbscoll - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

```c
#include <string.h>
char *strcpy( char *dst, const char *src );
char __far *__fstrcpy( char __far *dst,
    const char __far *src );
#include <wchar.h>
wchar_t *wcscpy( wchar_t *dst, const wchar_t *src );
#include <mbstring.h>
int _mbscpy( unsigned char *dst,
    const unsigned char *src );
int _fmbscpy( unsigned char __far *dst,
    const unsigned char __far *src );
```

Safer C:
The Safer C Library extension provides the `strcpy_s` function which is a safer alternative to `strcpy`. This newer `strcpy_s` function is recommended to be used instead of the traditional "unsafe" `strcpy` function.

Description: The `strcpy` function copies the string pointed to by `src` (including the terminating null character) into the array pointed to by `dst`. Copying of overlapping objects is not guaranteed to work properly. See the description for the `memmove` function to copy objects that overlap.

The `_fstrcpy` function is a data model independent form of the `strcpy` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `wcscpy` function is a wide-character version of `strcpy` that operates with wide-character strings.

The `_mbscpy` function is a multibyte character version of `strcpy` that operates with multibyte character strings.

Returns: The value of `dst` is returned.

See Also: `strdup`, `strncpy`, `strcpy_s`, `strncpy_s`

Example:

```c
#include <stdio.h>
#include <string.h>

void main()
{
    auto char buffer[80];
    
    strcpy( buffer, "Hello ");
    strcat( buffer, "world" );
    printf( "%s\n", buffer );
}

produces the following:

Hello world

Classification: `strcpy` is ANSI
 `_fstrcpy` is not ANSI
 `wcscpy` is ANSI
 `_mbscpy` is not ANSI
 `_fmbscpy` is not ANSI
Systems:

- `strcpy` - All, Netware
- `_fstrcpy` - All
- `wcscpy` - All
- `_mbscpy` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
- `_fmbscpy` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
**Synopsis:**

```c
#define __STDC_WANT_LIB_EXT1__  1
#include <string.h>
errno_t strcpy_s( char * restrict s1, 
                  rsize_t s1max, 
                  const char * restrict s2 );
#include <wchar.h>
errno_t wcscpy_s( wchar_t * restrict s1, 
                  rsize_t s1max, 
                  const wchar_t * restrict s2 );
```

**Constraints:**

If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `strcpy_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

- Neither `s1` nor `s2` shall be a null pointer.
- `s1max` shall not be greater than `RSIZE_MAX`. `s1max` shall not equal zero. `s1max` shall be greater than `strnlen_s(s2, s1max)`. Copying shall not take place between objects that overlap.

If there is a runtime-constraint violation, then if `s1` is not a null pointer and `s1max` is greater than zero and not greater than `RSIZE_MAX`, then `strcpy_s` sets `s1[0]` to the null character.

**Description:**

The `strcpy_s` function copies the string pointed to by `s2` (including the terminating null character) into the array pointed to by `s1`. All elements following the terminating null character (if any) written by `strcpy_s` in the array of `s1max` characters pointed to by `s1` take unspecified values when `strcpy_s` returns.

The `wcscpy_s` function is a wide-character version of `strcpy_s` that operates with wide-character strings.

**Returns:**

The `strcpy_s` function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

**See Also:**

`strcpy`, `strdup`, `strncpy`, `strncpy_s`

**Example:**

```c
#define __STDC_WANT_LIB_EXT1__  1
#include <stdio.h>
#include <string.h>

void main( void )
{
    auto char buffer[80];

    strcpy_s( buffer, sizeof( buffer ), "Hello " );
    strcat_s( buffer, sizeof( buffer ), "world" );
    printf( "%s\n", buffer );
}
```

produces the following:

Hello world

**Classification:**

`strcpy_s` is TR 24731

`wcscpy_s` is TR 24731

---

932 Library Functions and Macros
Systems:

strcpy_s - All, Netware
wcscpy_s - All
Synopsis:

```c
#include <string.h>
size_t strcspn( const char *str,
    const char *charset );
size_t _fstrcspn( const char __far *str,
    const char __far *charset );

#include <wchar.h>
size_t wcscspn( const wchar_t *str,
    const wchar_t *charset );

#include <mbstring.h>
size_t _mbscspn( const unsigned char *str,
    const unsigned char *charset );
size_t _fmbscspn( const unsigned char __far *str,
    const unsigned char __far *charset );
```

Description:
The `strcspn` function computes the length, in bytes, of the initial segment of the string pointed to by `str` which consists entirely of characters not from the string pointed to by `charset`. The terminating null character is not considered part of `str`.

The `_fstrcspn` function is a data model independent form of the `strcspn` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The `wcscspn` function is a wide-character version of `strcspn` that operates with wide-character strings.

The `_mbcspn` function is a multibyte character version of `strcspn` that operates with multibyte character strings.

Returns:
The length, in bytes, of the initial segment is returned.

See Also:
`strspn`

Example:

```c
#include <stdio.h>
#include <string.h>

void main()
{
    printf( "%d\n", strcspn( "abcbcadef", "cba" ) );
    printf( "%d\n", strcspn( "xxxbcadef", "cba" ) );
    printf( "%d\n", strcspn( "123456789", "cba" ) );
}
```

produces the following:

```
0
3
9
```

Classification:
- `strcspn` is ANSI
- `_fstrcspn` is not ANSI
- `wcscspn` is ANSI
- `_mbcspn` is not ANSI
- `_mbcspn` is not ANSI
- `_fmbscspn` is not ANSI

Systems:
- `strcspn` - All, Netware
strcspn, _fstrcspn, wcscspn, _mbscspn, _fmbscspn

_fstrcspn - All
wcscspn - All
_mbscspn - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fmbscspn - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <time.h>
char * _strdate( char *datestr )
wchar_t _wstrdate( wchar_t *datestr );

Description:  The _strdate function copies the current date to the buffer pointed to by datestr. The date is formatted as "MM/DD/YY" where "MM" is two digits representing the month, where "DD" is two digits representing the day, and where "YY" is two digits representing the year. The buffer must be at least 9 bytes long.

The _wstrdate function is a wide-character version of _strdate that operates with wide-character strings.

Returns:  The _strdate function returns a pointer to the resulting text string datestr.

See Also:  asctime Functions, ctime Functions, gmtime, localtime, mktime, _strtime, time, tzset

Example:  
#include <stdio.h>
#include <time.h>
void main()
{
       char datebuff[9];

       printf("%s\n", _strdate( datebuff ) );
}

Classification:  WATCOM

Systems:  _strdate – All
         _wstrdate – All
Synopsis:  
```
#include <tchar.h>
char *_strdec( const char *start, const char *current );
wchar_t *wcsdec( const wchar_t *start,
    const wchar_t *current );
#include <mbstring.h>
unsigned char *_mbsdec( const unsigned char *start,
    const unsigned char *current );
unsigned char *fmbsdec( const unsigned char __far *start,
    const unsigned char __far *current );
```

Description:  
The _strdec function returns a pointer to the previous character (single-byte, wide, or multibyte) in the string pointed to by start which must precede current. The current character in the string is pointed to by current. You must ensure that current does not point into the middle of a multibyte or wide character.

The function is a data model independent form of the _strdec function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The _wcsdec function is a wide-character version of _strdec that operates with wide-character strings.

The _mbsdec function is a multibyte character version of _strdec that operates with multibyte character strings.

Returns:  
The _strdec function returns a pointer to the previous character (single-byte, wide, or multibyte depending on the function used).

See Also:  
_strinc, _strninc

Example:  
```
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

const unsigned char chars[] = {
    ',',
    ',',
    ',',
    '1',
    'A',
    0x81,0x40, /* double-byte space */
    0x82,0x60, /* double-byte A */
    0x82,0xA6, /* double-byte Hiragana */
    0x83,0x42, /* double-byte Katakana */
    0xA1,      /* single-byte Katakana punctuation */
    0xA6,      /* single-byte Katakana alphabetic */
    0xDF,      /* single-byte Katakana alphabetic */
    0xE0,0xA1, /* double-byte Kanji */
    0x00
};
#define SIZE sizeof( chars ) / sizeof( unsigned char )
```
void main()
{
    int j, k;
    const unsigned char *prev;

    _setmbcp( 932 );
    prev = &chars[ SIZE - 1 ];
    do {
        prev = _mbsdec( chars, prev );
        j = mblen( prev, MB_CUR_MAX );
        if( j == 0 ) {
            k = 0;
        } else if ( j == 1 ) {
            k = *prev;
        } else if( j == 2 ) {
            k = *(prev)<<8 | *(prev+1);
        }
        printf( "Previous character %#6.4x\n", k );
    } while( prev != chars );
}

produces the following:

Previous character 0xe0a1
Previous character 0x00df
Previous character 0x00a6
Previous character 0x00a1
Previous character 0x8342
Previous character 0x82a6
Previous character 0x8260
Previous character 0x8140
Previous character 0x0041
Previous character 0x0031
Previous character 0x002e
Previous character 0x0020

Classification: WATCOM

Systems:
  _strdec - MACRO
  _wcsdec - MACRO
  _mbsdec - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
  _fmbsdec - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

```c
#include <string.h>
char *strdup( const char *src );
char *_strdup( const char *src );
char __far *_fstrdup( const char __far *src );
#include <wchar.h>
wchar_t *__wcsdup( const wchar_t *src );
#include <mbstring.h>
unsigned char *_mbsdup( unsigned char *src );
unsigned char __far *_fmbsdup( unsigned char __far *src );
```

Description:

The `strdup` function creates a duplicate copy of the string pointed to by `src` and returns a pointer to the new copy. For `strdup`, the memory for the new string is obtained by using the `malloc` function and can be freed using the `free` function. For `_fstrdup`, the memory for the new string is obtained by using the `_fmalloc` function and can be freed using the `_ffree` function.

The `_strdup` function is identical to `strdup`. Use `_strdup` for ANSI/ISO naming conventions.

The `_fstrdup` function is a data model independent form of the `strdup` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The `_wcsdup` function is a wide-character version of `strdup` that operates with wide-character strings.

The `_mbsdup` function is a multibyte character version of `strdup` that operates with multibyte character strings.

The `_fmbsdup` function is a data model independent form of the `_mbsdup` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns:

The `strdup` function returns the pointer to the new copy of the string if successful, otherwise it returns `NULL`.

See Also: `free`, `malloc`, `strcpy`, `strncpy`

Example:

```c
#include <stdio.h>
#include <string.h>

void main()
{
    char *dup;
    dup = strdup( "Make a copy" );
    printf( "%s\n", dup );
}
```

Classification: WATCOM

`strdup` conforms to ANSI/ISO naming conventions

Systems:

- `strdup` - All, Netware
- `_strdup` - All, Netware
- `_fstrdup` - All
- `_wcsdup` - All
- `_mbsdup` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
- `_fmbsdup` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
**strerror, wcserror**

**Synopsis:**
```
#include <string.h>
char *strerror( int errnum );
wchar_t *wcserror( int errnum );
```

**Safer C:** The Safer C Library extension provides the `strerror_s` function which is a safer alternative to `strerror`. This newer `strerror_s` function is recommended to be used instead of the traditional "unsafe" `strerror` function.

**Description:** The `strerror` function maps the error number contained in `errnum` to an error message.

The `wcserror` function is identical to `strerror` except that the message it points to is a wide-character string.

**Returns:** The `strerror` function returns a pointer to the error message. The array containing the error string should not be modified by the program. This array may be overwritten by a subsequent call to the `strerror` function.

**See Also:** `clearerr`, `feof`, `ferror`, `perror`, `strerror_s`, `strerrorlen_s`

**Example:**
```
#include <stdio.h>
#include <string.h>
#include <errno.h>

void main()
{
    FILE *fp;

    fp = fopen( "filename", "r" );
    if( fp == NULL ) {
        printf( "Unable to open file: %s
", strerror( errno ) );
    }
}
```

**Classification:** `strerror` is ANSI

`wcserror` is ANSI

**Systems:**
- `strerror` - All, Netware
- `wcserror` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
strerror_s, wcserror_s

Synopsis:
#define __STDC_WANT_LIB_EXT1__  1
#include <string.h>
errno_t strerror_s( char * s, 
rsize_t maxsize, 
errno_t errnum );

errno_t wcserror_s( wchar_t * s, 
rsize_t maxsize, 
errno_t errnum );

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and strerror_s will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

s shall not be a null pointer. maxsize shall not be greater than RSIZE_MAX. maxsize shall not equal zero.

If there is a runtime-constraint violation, then the array (if any) pointed to by s is not modified.

Description: The strerror_s function maps the number in errnum to a locale-specific message string. Typically, the values for errnum come from errno, but strerror_s shall map any value of type int to a message. If the length of the desired string is less than maxsize, then the string is copied to the array pointed to by s. Otherwise, if maxsize is greater than zero, then maxsize-1 characters are copied from the string to the array pointed to by s and then s[maxsize-1] is set to the null character. Then, if maxsize is greater than 3, then s[maxsize-2], s[maxsize-3], and s[maxsize-4] are set to the character period (.).

The wcserror_s function is a wide-character version of strerror_s that operates with wide-character strings.

Returns: The strerror_s function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

See Also: clearerr, feof, ferror, perror, strerror, strerrorlen_s

Example:
#define __STDC_WANT_LIB_EXT1__  1
#include <stdio.h>
#include <string.h>
#include <errno.h>

void main( void )
{
    FILE *fp;
    char emsg[ 100 ];

    fp = fopen( "file.nam", "r" );
    if( fp == NULL ) { 
        strerror_s( emsg, sizeof( emsg ), errno );
        printf( "Unable to open file: %s\n", emsg );
    }
}

Classification: strerror_s is TR 24731
    wcserror_s is TR 24731

Systems: strerror_s - All, Netware
strerror_s, wcserror_s

wcserror_s - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
size_t strerrorlen_s( errno_t errnum );
#include <wchar.h>
size_t wcserrorlen_s( errno errnum );

Constraints: None.

Description: The `strerrorlen_s` function calculates the length of the (untruncated) locale-specific message string that the `strerror_s` function maps to `errno`.

The `wcserrorlen_s` function is a wide-character version of `strerrorlen_s` that operates with wide-character strings.

Returns: The `strerrorlen_s` function returns the number of characters (not including the null character) in the full message string.

See Also: `strerror`, `strerror_s`  

Example: 
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <string.h>
#include <errno.h>

void main( void )
{
    FILE    *fp;
    char    emsg[ 100 ];
    size_t  emsglen;

    fp = fopen( "file.nam", "r" );
    if( fp == NULL ) {
        emsglen = strerrorlen_s( errno );
        printf( "Length of errormessage: %d\n", emsglen );
        strerror_s( emsg, sizeof( emsg ), errno );
        printf( "Unable to open file: %s\n", emsg );
    }
}

Classification: `strerrorlen_s` is TR 24731
`wcserrorlen_s` is TR 24731

Systems:  
`strerrorlen_s` - All, Netware  
`wcserrorlen_s` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
**Synopsis:**
```c
#include <time.h>
size_t strftime( char *s,
    size_t maxsize,
    const char *format,
    const struct tm *timeptr );

#include <wchar.h>
size_t wcsftime( wchar_t *s,
    size_t maxsize,
    const wchar_t *format,
    const struct tm *timeptr );

#include <time.h>
size_t _wstrftime_ms( wchar_t *s,
    size_t maxsize,
    const char *format,
    const struct tm *timeptr );
```

```
struct tm {
    int tm_sec;   /* seconds after the minute -- [0,61] */
    int tm_min;   /* minutes after the hour   -- [0,59] */
    int tm_hour;  /* hours after midnight     -- [0,23] */
    int tm_mday;  /* day of the month         -- [1,31] */
    int tm_mon;   /* months since January     -- [0,11] */
    int tm_year;  /* years since 1900                   */
    int tm_wday;  /* days since Sunday        -- [0,6]  */
    int tm_yday;  /* days since January 1     -- [0,365]*/
    int tm_isdst; /* Daylight Savings Time flag */
};
```

**Description:**

The `strftime` function formats the time in the argument `timeptr` into the array pointed to by the argument `s` according to the `format` argument.

The `wcsftime` function is a wide-character version of `strftime` that operates with wide-character strings.

The `_wstrftime_ms` function is identical to `wcsftime` except that the `format` is not a wide-character string.

The `format` string consists of zero or more directives and ordinary characters. A directive consists of a `'%'` character followed by a character that determines the substitution that is to take place. All ordinary characters are copied unchanged into the array. No more than `maxsize` characters are placed in the array. The format directives `%D, %h, %n, %r, %t, and %T` are from POSIX.

<table>
<thead>
<tr>
<th>Directive</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%a</code></td>
<td>locale’s abbreviated weekday name</td>
</tr>
<tr>
<td><code>%A</code></td>
<td>locale’s full weekday name</td>
</tr>
<tr>
<td><code>%b</code></td>
<td>locale’s abbreviated month name</td>
</tr>
<tr>
<td><code>%B</code></td>
<td>locale’s full month name</td>
</tr>
<tr>
<td><code>%c</code></td>
<td>locale’s appropriate date and time representation</td>
</tr>
<tr>
<td>Format</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>%C</code></td>
<td>is replaced by the year divided by 100 and truncated to an integer (00-99)</td>
</tr>
<tr>
<td><code>%d</code></td>
<td>day of the month as a decimal number (01-31)</td>
</tr>
<tr>
<td><code>%D</code></td>
<td>date in the format mm/dd/yy (POSIX)</td>
</tr>
<tr>
<td><code>%e</code></td>
<td>day of the month as a decimal number (1-31), a single digit is preceded by a blank</td>
</tr>
<tr>
<td><code>%F</code></td>
<td>is equivalent to <code>%Y-%m-%d</code> (the ISO 8601 date format)</td>
</tr>
<tr>
<td><code>%g</code></td>
<td>is replaced by the last 2 digits of the week-based year as a decimal number (00-99)</td>
</tr>
<tr>
<td><code>%G</code></td>
<td>is replaced by the week-based year as a decimal number (e.g. 2006)</td>
</tr>
<tr>
<td><code>%h</code></td>
<td>locale’s abbreviated month name (POSIX)</td>
</tr>
<tr>
<td><code>%H</code></td>
<td>hour (24-hour clock) as a decimal number (00-23)</td>
</tr>
<tr>
<td><code>%I</code></td>
<td>hour (12-hour clock) as a decimal number (01-12)</td>
</tr>
<tr>
<td><code>%j</code></td>
<td>day of the year as a decimal number (001-366)</td>
</tr>
<tr>
<td><code>%m</code></td>
<td>month as a decimal number (01-12)</td>
</tr>
<tr>
<td><code>%M</code></td>
<td>minute as a decimal number (00-59)</td>
</tr>
<tr>
<td><code>%n</code></td>
<td>newline character (POSIX)</td>
</tr>
<tr>
<td><code>%p</code></td>
<td>locale’s equivalent of either AM or PM</td>
</tr>
<tr>
<td><code>%r</code></td>
<td>12-hour clock time (01-12) using the AM/PM notation in the format HH:MM:SS (AM</td>
</tr>
<tr>
<td><code>%S</code></td>
<td>second as a decimal number (00-59)</td>
</tr>
<tr>
<td><code>%t</code></td>
<td>tab character (POSIX)</td>
</tr>
<tr>
<td><code>%T</code></td>
<td>24-hour clock time in the format HH:MM:SS (POSIX)</td>
</tr>
<tr>
<td><code>%u</code></td>
<td>is replaced by the ISO 8601 weekday as a decimal number (1-7), where Monday is 1</td>
</tr>
<tr>
<td><code>%U</code></td>
<td>week number of the year as a decimal number (00-52) where Sunday is the first day of the week</td>
</tr>
<tr>
<td><code>%V</code></td>
<td>is replaced by the ISO 8601 week number as a decimal number (01-53)</td>
</tr>
<tr>
<td><code>%w</code></td>
<td>weekday as a decimal number (0-6) where 0 is Sunday</td>
</tr>
<tr>
<td><code>%W</code></td>
<td>week number of the year as a decimal number (00-52) where Monday is the first day of the week</td>
</tr>
<tr>
<td><code>%x</code></td>
<td>locale’s appropriate date representation</td>
</tr>
</tbody>
</table>
strftime, wcsftime, _wstrft time_ms

%X locale’s appropriate time representation
%y year without century as a decimal number (00-99)
%Y year with century as a decimal number
%z offset from UTC in the ISO 8601 format ‘-0430’ (meaning 4 hours 30 minutes behind
UTC, west of Greenwich), or by no characters, if no timezone is determinable
%Z timezone name, or by no characters if no timezone exists
%% character %

When the %Z or %z directive is specified, the tzset function is called.

% g, %G, %V give values according to the ISO 8601 week-based year. In this system, weeks begin on
a monday and week 1 of the year is the week that includes January 4th, which is also the week that
includes the first Thursday of the year, and is also the first week that contains at least four days in the
year. If the first Monday of January is the 2nd, 3rd, or 4th, the preceding days are part of the last week
of the preceding year; thus, for Saturday 2nd January 1999, %G is replaced by 1998 and %V is replaced
by 53. If december 29th, 30th, or 31st is a Monday, it and any following days are part of week 1 of the
following year. Thus, for Tuesday 30th December 1997, %G is replaced by 1998 and %V is replaced
by 01.

The format modifiers E and O are ignored. (eg. %EY is the same as %Y)

Returns: If the number of characters to be placed into the array is less than maxsize, the strftime function
returns the number of characters placed into the array pointed to by s not including the terminating null
character. Otherwise, zero is returned. When an error has occurred, errno contains a value indicating
the type of error that has been detected.

See Also: setlocale, asctime Functions, asctime_s, clock, ctime Functions, ctime_s, difftime,
gmtime, gmtime_s, localtime, localtime_s, mktime, time, tzset

Example: #include <stdio.h>
#include <time.h>

void main()
{
    time_t time_of_day;
    char buffer[80];

    time_of_day = time( NULL );
    strftime( buffer, 80, "Today is %A %B %d, %Y",
        localtime( &time_of_day ) );
    printf( "%s\n", buffer );
}

produces the following:

Today is Friday December 25, 1987

Classification: strftime is ANSI, POSIX
wcsftime is ANSI

946 Library Functions and Macros
_wstrftime_ms is not ANSI

Systems:

_strftime - All, Netware
wcsftime - All
_wstrftime_ms - All
Synopsis:

```
#include <string.h>
int stricmp( const char *s1, const char *s2 );
int _stricmp( const char *s1, const char *s2 );
int _fstricmp( const char __far *s1, 
               const char __far *s2 );
#include <wchar.h>
int _wcsicmp( const wchar_t *s1, const wchar_t *s2 );
#include <mbstring.h>
int _mbsicmp( const unsigned char *s1, 
              const unsigned char *s2 );
int _fmbsicmp( const unsigned char __far *s1, 
              const unsigned char __far *s2 );
```

description:
The `stricmp` function compares, with case insensitivity, the string pointed to by `s1` to the string pointed to by `s2`. All uppercase characters from `s1` and `s2` are mapped to lowercase for the purposes of doing the comparison.

The `_stricmp` function is identical to `stricmp`. Use `_stricmp` for ANSI/ISO naming conventions.

The `_fstricmp` function is a data model independent form of the `stricmp` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The `_wcsicmp` function is a wide-character version of `stricmp` that operates with wide-character strings.

The `_mbsicmp` function is a multibyte character version of `stricmp` that operates with multibyte character strings.

The `_fmbsicmp` function is a data model independent form of the `_mbsicmp` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns:
The `stricmp` function returns an integer less than, equal to, or greater than zero, indicating that the string pointed to by `s1` is less than, equal to, or greater than the string pointed to by `s2`.

See Also:
strcmp, strncmp, strncmp, strnicmp

Example:
```
#include <stdio.h>
#include <string.h>

void main()
{
    printf( "%d\n", stricmp( "AbCDEF", "abcdef" ) );
    printf( "%d\n", stricmp( "abcdef", "ABC" ) );
    printf( "%d\n", stricmp( "abc", "ABCDEF" ) );
    printf( "%d\n", stricmp( "Abcdef", "mnopqr" ) );
    printf( "%d\n", stricmp( "Mnopqr", "abcdef" ) );
}
```

produces the following:
stricmp, _stricmp, _fstricmp, _wcscmp, _mbsicmp, _fmbsicmp

-12
12

Classification: WATCOM
_stricmp conforms to ANSI/ISO naming conventions

Systems:
stricmp - All, Netware
_stricmp - All, Netware
_fstricmp - All
_wcscmp - All
_mbsicmp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fmbsicmp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

```c
#include <string.h>
int _stricoll( const char *s1, const char *s2 );
#include <wchar.h>
int _wcsicoll( const wchar_t *s1, const wchar_t *s2 );
#include <mbstring.h>
int _mbsicoll( const unsigned char *s1, const unsigned char *s2 );
```

Description:
The `_stricoll` function performs a case insensitive comparison of the string pointed to by `s1` to the string pointed to by `s2`. The comparison uses the current code page which can be selected by the `_setmbcp` function.

The `_wcsicoll` function is a wide-character version of `_stricoll` that operates with wide-character strings.

The `_mbsicoll` function is a multibyte character version of `_stricoll` that operates with multibyte character strings.

Returns:
These functions return an integer less than, equal to, or greater than zero, indicating that the string pointed to by `s1` is less than, equal to, or greater than the string pointed to by `s2`, according to the collating sequence selected.

See Also:
`_setmbcp`, `strcoll`, `stricmp`, `strncmp`, `_strncoll`, `strnicmp`, `_strnicoll`

Example:

```c
#include <stdio.h>
#include <string.h>

char buffer[80] = "world";

void main()
{
    int test;
    test = _stricoll( buffer, "world2" );
    if( test < 0 ) {
        printf( "Less than\n" );
    } else if( test == 0 ) {
        printf( "Equal\n" );
    } else {
        printf( "Greater than\n" );
    }
}
```

Classification: WATCOM

Systems:
- `_stricoll` - All, Netware
- `_wcsicoll` - All
- `_mbsicoll` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

```c
#include <tchar.h>
char *__strinc( const char *current );
wchar_t *__wcsinc( const wchar_t *current );
#include <mbstring.h>
unsigned char *__mbsinc( const unsigned char *current );
unsigned char *__fmbsinc(
    const unsigned char __far *current );
```

Description:

The `_strinc` function returns a pointer to the next character (single-byte, wide, or multibyte) in the string pointed to by `current`. You must ensure that `current` does not point into the middle of a multibyte or wide character.

The function is a data model independent form of the `_strinc` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The `_wcsinc` function is a wide-character version of `_strinc` that operates with wide-character strings.

The `_mbsinc` function is a multibyte character version of `_strinc` that operates with multibyte character strings.

Returns:

The `_strinc` function returns a pointer to the next character (single-byte, wide, or multibyte depending on the function used).

See Also:

`_strdec, _strninc`

Example:
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

const unsigned char chars[] = {
    '
',
    '.
',
    '1',
    'A',
    0x81,0x40, /* double-byte space */
    0x82,0x60, /* double-byte A */
    0x82,0xA6, /* double-byte Hiragana */
    0x83,0x42, /* double-byte Katakana */
    0xA1,      /* single-byte Katakana punctuation */
    0xA6,      /* single-byte Katakana alphabetic */
    0xDF,      /* single-byte Katakana alphabetic */
    0xE0,0xA1, /* double-byte Kanji */
    0x00
};

#define SIZE sizeof( chars ) / sizeof( unsigned char )

void main()
{
    int j, k;
    const unsigned char *next;
    
    _setmbscp( 932 );
    next = chars;
    do {
        next = _mbsinc( next );
        j = mblen( next, MB_CUR_MAX );
        if( j == 0 ) {
            k = 0;
        } else if ( j == 1 ) {
            k = *next;
        } else if( j == 2 ) {
            k = *(next)<<8 | *(next+1);
        }
        printf( "Next character %#6.4x
", k );
    } while( next != &chars[ SIZE - 1 ] );
}

produces the following:

Next character 0x002e
Next character 0x0031
Next character 0x0041
Next character 0x8140
Next character 0x8260
Next character 0x82a6
Next character 0x8342
Next character 0x00a1
Next character 0x00a6
Next character 0xe0df
Next character 0xe0a1
Next character 0x0000
Classification: WATCOM

Systems:

_strinc - MACRO

_wcsinc - MACRO

_mbsinc - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

_fmbsinc - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
**strlcat, wcslcat**

**Synopsis:**
```c
#include <string.h>
size_t strlcat( char *dst, const char *src, size_t n );
size_t *wcslcat( wchar_t *dst, const wchar_t *src, size_t n );
```

**Description:**
The `strlcat` function appends characters of the string pointed to by `src` to the end of the string in a buffer pointed to by `dst` that can hold up to `n` characters. The first character of `src` overwrites the null character at the end of `dst`. A terminating null character is always appended to the result, unless `n` characters of `dst` are scanned and no null character is found.

The `wcslcat` function is a wide-character version of `strlcat` that operates with wide-character strings.

**Returns:**
The `strlcat` function returns the total length of string it tried to create, that is the number of characters in both `src` and `dst` strings, not counting the terminating null characters. If `n` characters of `dst` were scanned without finding a null character, `n` is returned.

**See Also:**
`strlcpy`, `strncat`, `strcat`

**Example:**
```c
#include <stdio.h>
#include <string.h>

char buffer[80];

void main( void )
{
  strcpy( buffer, "Hello ");
  strlcat( buffer, "world", 12 );
  printf( "%s\n", buffer );
  strlcat( buffer, "*************", 16 );
  printf( "%s\n", buffer );
}
```

produces the following:
```
Hello world
Hello world****
```

**Classification:** WATCOM

**Systems:**
- `strlcat` - All, Netware
- `wcslcat` - All
Synopsis:

```
#include <string.h>

size_t strlcpy( char *dst,
               const char *src,
               size_t n );

size_t wcslcpy( wchar_t *dst,
               const wchar_t *src,
               size_t n );
```

Description: The `strlcpy` function copies no more than n characters from the string pointed to by `src` into the array pointed to by `dst`. Copying of overlapping objects is not guaranteed to work properly. See the `memmove` function if you wish to copy objects that overlap.

If the string pointed to by `src` is longer than n characters, then only n - 1 characters will be copied and the result will be null terminated.

The `wcslcpy` function is a wide-character version of `strlcpy` that operates with wide-character strings.

Returns: The `strlcpy` function returns the number of characters in the `src` string, not including the terminating null character.

See Also: `strlcat`, `strncpy`, `strcpy`

Example:

```
#include <stdio.h>
#include <string.h>

void main( void )
{
    char    buffer[10];

    printf( "%d:%s\n", strlcpy( buffer,
                   "Buffer overflow", sizeof( buffer ) ), buffer );
}
```

produces the following:

```
15:'Buffer ov'
```

Classification: WATCOM

Systems: `strlcpy` - All, Netware
          `wcslcpy` - All
Synopsis:    
#include <string.h>
size_t strlen( const char *s );
size_t _fstrlen( const char __far *s );
#include <wchar.h>
size_t wcslen( const wchar_t *s );
#include <mbstring.h>
size_t _mbslen( const unsigned char *s );
size_t _fmbslen( const unsigned char __far *s );

Safer C:  The Safer C Library extension provides the function which is a safer alternative to strlen. This newer strlen_s function is recommended to be used instead of the traditional "unsafe" strlen function.

Description:  The strlen function computes the length of the string pointed to by s.

The _fstrlen function is a data model independent form of the strlen function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The wcslen function is a wide-character version of strlen that operates with wide-character strings.

The _mbslen function is a multibyte character version of strlen that operates with multibyte character strings.

The _fmbslen function is a data model independent form of the _mbslen function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The strlen function returns the number of characters that precede the terminating null character.

See Also:  strlen_s

Example:  
#include <stdio.h>
#include <string.h>

void main()
{
    printf( "\%d\n", strlen( "Howdy" ) );
    printf( "\%d\n", strlen( "Hello world\n" ) );
    printf( "\%d\n", strlen( "" ) );
}

produces the following:

5
12
0

Classification:  strlen is ANSI
    _fstrlen is not ANSI
    wcslen is ANSI
    _mbslen is not ANSI
    _fmbslen is not ANSI

Systems:  strlen - All, Netware
    _fstrlen - All
wcslen - All
_mbslen - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fmbslen - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
**strnlen_s, wcsnlen_s**

**Synopsis:**
```
#define __STDC_WANT_LIB_EXT1__  1
#include <string.h>
size_t strnlen_s( const char * s,
                 size_t maxsize );
#include <wchar.h>
size_t wcsnlen_s( const wchar_t * s,
                 size_t maxsize );
```

**Constraints:** None.

**Description:** The `strnlen_s` function calculates the length of the string pointed to by `s`.

The `wcsnlen_s` function is a wide-character version of `strnlen_s` that operates with wide-character strings.

**Returns:** If `s` is a null pointer, then the `strnlen_s` function returns zero. Otherwise, the `strnlen_s` function returns the number of characters that precede the terminating null character. If there is no null character in the first `maxsize` characters of `s` then `strnlen_s` returns `maxsize`. At most the first `maxsize` characters of `s` shall be accessed by `strnlen_s`.

**See Also:** `strlen`

**Example:**
```
#define __STDC_WANT_LIB_EXT1__  1
#include <stdio.h>
#include <string.h>

void main( void )
{
    char    buffer[ 30 ] = "Hello world.";
    size_t  len;

    len = strnlen_s( buffer, sizeof( buffer ) );
    printf( "Length of text: %d\n", len );
    printf( "Text: %s\n", buffer );
}
```

**Classification:**
- `strnlen_s` is TR 24731
- `wcsnlen_s` is TR 24731

**Systems:**
- `strnlen_s` - All, Netware
- `wcsnlen_s` - All
Synopsis:

```
#include <string.h>
char *strlwr( char *s1 );
char *__strlwr( char *s1 );
char __far *__fstrlwr( char __far *s1 );
#include <wchar.h>
wchar_t *__wcslwr( wchar_t *s1 );
#include <mbstring.h>
unsigned char *__mbslwr( unsigned char *s1 );
unsigned char __far *__fmbslwr( unsigned char __far *s1 );
```

Description: The `strlwr` function replaces the string `s1` with lowercase characters by invoking the `tolower` function for each character in the string.

The `_strlwr` function is identical to `strlwr`. Use `_strlwr` for ANSI/ISO naming conventions.

The `_fstrlwr` function is a data model independent form of the `strlwr` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `_wcslwr` function is a wide-character version of `strlwr` that operates with wide-character strings.

The `_mbslwr` function is a multibyte character version of `strlwr` that operates with multibyte character strings.

The `_fmbslwr` function is a data model independent form of the `_mbslwr` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The address of the original string `s1` is returned.

See Also: `strupr`

Example:

```
#include <stdio.h>
#include <string.h>

char source[] = { "A mixed-case STRING" };

void main()
{
    printf( "%s\n", source );
    printf( "%s\n", strlwr( source ) );
    printf( "%s\n", source );
}
```

produces the following:

```
A mixed-case STRING
a mixed-case string
a mixed-case string
```

Classification: WATCOM

`strlwr` conforms to ANSI/ISO naming conventions

Systems: `strlwr` - All, Netware
          `_strlwr` - All, Netware
strlwr, _strlwr, _fstrlwr, _wcslwr, _mbslwr, _fmbslwr

_fstrlwr - All
_wcs1wr - All
_mbslwr - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fmbslwr - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:
#include <strings.h>
int strncasecmp( const char *s1,
    const char *s2, size_t len );

Description: The `strncasecmp` function compares, without case sensitivity, the string pointed to by s1 to the string pointed to by s2, for at most len characters.

The `strncasecmp` function is identical to the `strnicmp` function.

Returns: The `strncasecmp` function returns an integer less than, equal to, or greater than zero, indicating that the string pointed to by s1 is, ignoring case, less than, equal to, or greater than the string pointed to by s2.

See Also: `strcmp`, `stricmp`, `strncmp`, `strcasecmp`

Example: #include <stdio.h>
#include <strings.h>

    int main( void )
    {
        printf( "%d\n", strncasecmp( "abcdef", "ABCXXX", 10 ) );
        printf( "%d\n", strncasecmp( "abcdef", "ABCXXX", 6 ) );
        printf( "%d\n", strncasecmp( "abcdef", "ABCXXX", 3 ) );
        printf( "%d\n", strncasecmp( "abcdef", "ABCXXX", 0 ) );
        return( 0 );
    }

produces the following:

-20
-20
0
0

Classification: POSIX

Systems: All, Netware
strncat, _fstrncat, wcsncat, _mbsncat, _fmbsncat

Synopsis:  
#include <string.h>  
char *strncat( char *dst, const char *src, size_t n );  
char __far *fstrncat( char __far *dst,  
const char __far *src,  
size_t n );  
#include <wchar.h>  
wchar_t *wcsncat( wchar_t *dst,  
const wchar_t *src,  
size_t n );  
#include <mbstring.h>  
unsigned char *mbstrncat( unsigned char *dst,  
const unsigned char *src,  
szize_t n );  
unsigned char __far *fmbsncat( unsigned char __far *dst,  
const unsigned char __far *src,  
szize_t n );

Safer C:  
The Safer C Library extension provides the strncat_s function which is a safer alternative to strncat. This newer strncat_s function is recommended to be used instead of the traditional "unsafe" strncat function.

Description:  
The strncat function appends not more than n characters of the string pointed to by src to the end of the string pointed to by dst. The first character of src overwrites the null character at the end of dst. A terminating null character is always appended to the result.

The _fstrncat function is a data model independent form of the strncat function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The wcsncat function is a wide-character version of strncat that operates with wide-character strings.

The _mbsncat function is a multibyte character version of strncat that operates with multibyte character strings.

The _fmbsncat function is a data model independent form of the _mbsncat function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns:  
The strncat function returns the value of dst.

See Also:  
strcat, strlcat, strncat_s, strcat_s

Example:  
#include <stdio.h>  
#include <string.h>

char buffer[80];

void main( void )
{
    strcpy( buffer, "Hello ");
    strncat( buffer, "world", 8 );
    printf( "%s\n", buffer );
    strncat( buffer, "*************", 4 );
    printf( "%s\n", buffer );
}
produces the following:

Hello world
Hello world****

Classification:  
strncat is ANSI  
_fstrncat is not ANSI  
wcsncat is ANSI  
_mbsncat is not ANSI  
_fmbsncat is not ANSI

Systems:  
strncat - All, Netware  
_fstrncat - All  
wcsncat - All  
_mbsncat - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32  
_fmbsncat - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
**Synopsis:**

```c
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
errno_t strncat_s( char * restrict s1,
    rsize_t s1max,
    const char * restrict s2,
    rsize_t n )
#include <wchar.h>
errno_t wcsncat_s( wchar_t * restrict s1,
    rsize_t s1max,
    const wchar_t * restrict s2,
    rsize_t n )
```

**Constraints:**

If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `strncat_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Let \( m \) denote the value \( s1max - strlen(s1, s1max) \) upon entry to `strncat_s`

Neither \( s1 \) nor \( s2 \) shall be a null pointer. Neither \( s1max \) nor \( n \) shall be greater than \( RSIZE_MAX \). \( s1max \) shall not equal zero. \( m \) shall not equal zero. If \( n \) is not less than \( m \), then \( m \) shall be greater than \( strlen(s2, m) \). Copying shall not take place between objects that overlap.

If there is a runtime-constraint violation, then if \( s1 \) is not a null pointer and \( s1max \) is greater than zero and not greater than \( RSIZE_MAX \), then `strncat_s` sets \( s1[0] \) to the null character.

**Description:**

The `strncat_s` function appends not more than \( n \) successive characters (characters that follow a null character are not copied) from the array pointed to by \( s2 \) to the end of the string pointed to by \( s1 \). The initial character from \( s2 \) overwrites the null character at the end of \( s1 \). If no null character was copied from \( s2 \), then \( s1[s1max-m+n] \) is set to a null character. All elements following the terminating null character (if any) written by `strncat_s` in the array of \( s1max \) characters pointed to by \( s1 \) take unspecified values when `strncat_s` returns.

The `wcsncat_s` function is a wide-character version of `strncat_s` that operates with wide-character strings.

**Returns:**

The `strncat_s` function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

**See Also:**

`strcat`, `strlcat`, `strcat_s`

**Example:**

```c
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <string.h>

cchar buffer[80];

do main( void )
{
   strcpy( buffer, "Hello " );
   strncat_s( buffer, sizeof( buffer ), "world", 8 );
   printf( "%s\n", buffer );
   strncat( buffer, "***********", 4 );
   printf( "%s\n", buffer );
}
```
produces the following:

Hello world
Hello world****

Classification:  strncat_s is TR 24731
               wcsncat_s is TR 24731

Systems:       strncat_s - All, Netware
               wcsncat_s - All
Synopsis: 
#include <string.h>
int strncmp( const char *s1,
const char *s2,
size_t n );
int _fstrncmp( const char __far *s1,
const char __far *s2,
size_t n );
#include <wchar.h>
int wcsncmp( const wchar_t *s1,
const wchar_t *s2,
size_t n );
#include <mbstring.h>
int _mbsncmp( const unsigned char *s1,
const unsigned char *s2,
size_t n );
int _fmbsncmp( const unsigned char __far *s1,
const unsigned char __far *s2,
size_t n );

Description: The strncmp compares not more than \( n \) characters from the string pointed to by \( s1 \) to the string pointed to by \( s2 \).

The _fstrncmp function is a data model independent form of the strncmp function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The wcsncmp function is a wide-character version of strncmp that operates with wide-character strings.

The _mbsncmp function is a multibyte character version of strncmp that operates with multibyte character strings.

The _fmbsncmp function is a data model independent form of the _mbsncmp function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The strncmp function returns an integer less than, equal to, or greater than zero, indicating that the string pointed to by \( s1 \) is less than, equal to, or greater than the string pointed to by \( s2 \).

See Also: strncmp, stricmp, strnicmp

Example: 
#include <stdio.h>
#include <string.h>

void main()
{
    printf( "%d\n",strncmp( "abcdef", "abcDEF", 10 ) );
    printf( "%d\n", strncmp( "abcdef", "abcDEF", 6 ) );
    printf( "%d\n", strncmp( "abcdef", "abcDEF", 3 ) );
    printf( "%d\n", strncmp( "abcdef", "abcDEF", 0 ) );
}

produces the following:
strncmp, _fstrncmp, wcsncmp, _mbsncmp, _mbsncmp

1
1
0
0

Classification: strncmp is ANSI
_fstrncmp is not ANSI
wcsncmp is ANSI
_mbsncmp is not ANSI
_fmbsncmp is not ANSI

Systems: strncmp - All, Netware
_fstrncmp - All
wcsncmp - All
_mbsncmp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fmbsncmp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <string.h>
int _strncoll( const char *s1,
        const char *s2,
        size_t count );

#include <wchar.h>
int _wcsncoll( const wchar_t *s1,
        const wchar_t *s2,
        size_t count );

#include <mbstring.h>
int _mbsncoll( const unsigned char *s1,
        const unsigned char *s2,
        size_t count );

Description: These functions compare the first count characters of the string pointed to by s1 to the string pointed to by s2. The comparison uses the current code page which can be selected by the _setmbcp function.

The _wcsncoll function is a wide-character version of _strncoll that operates with wide-character strings.

The _mbsncoll function is a multibyte character version of _strncoll that operates with multibyte character strings.

Returns: These functions return an integer less than, equal to, or greater than zero, indicating that the string pointed to by s1 is less than, equal to, or greater than the string pointed to by s2, according to the collating sequence selected.

See Also: _setmbcp, strcoll, stricmp, _stricoll, strncmp, strnicmp, _stricoll

Example:  
#include <stdio.h>
#include <string.h>

char buffer[80] = "world";

void main()
{
    int test;

    test = _strncoll( buffer, "world2", 5 );
    if( test < 0 ) {
        printf( "Less than\n" );
    } else if( test == 0 ) {
        printf( "Equal\n" );
    } else {
        printf( "Greater than\n" );
    }
}

Classification: WATCOM

Systems:  
_strncoll - All, Netware
_wcsncoll - All
_mbsncoll - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

968  Library Functions and Macros
SYNOPSIS:

```
#include <string.h>
char *strncpy( char *dst,
               const char *src,
               size_t n );

cchar __far *__fstrncpy( char __far *dst,
                         const char __far *src,
                         size_t n );

#include <wchar.h>
wchar_t *wcsncpy( wchar_t *dst,
                 const wchar_t *src,
                 size_t n );

#include <mbstring.h>
unsigned char *__mbsncpy( unsigned char *dst,
                         const unsigned char *src,
                         size_t n );

unsigned char __far *__fmbsncpy( unsigned char __far *dst,
                                const unsigned char __far *src,
                                size_t n );
```

Safer C:  
The Safer C Library extension provides the `strncpy_s` function which is a safer alternative to `strncpy`. This newer `strncpy_s` function is recommended to be used instead of the traditional "unsafe" `strncpy` function.

Description:  
The `strncpy` function copies no more than `n` characters from the string pointed to by `src` into the array pointed to by `dst`. Copying of overlapping objects is not guaranteed to work properly. See the `memmove` function if you wish to copy objects that overlap.

If the string pointed to by `src` is shorter than `n` characters, null characters are appended to the copy in the array pointed to by `dst`, until `n` characters in all have been written. If the string pointed to by `src` is longer than `n` characters, then the result will not be terminated by a null character.

The `_fstrncpy` function is a data model independent form of the `strncpy` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `wcsncpy` function is a wide-character version of `strncpy` that operates with wide-character strings.

The `_mbsncpy` function is a multibyte character version of `strncpy` that operates with multibyte character strings.

The `_fmbsncpy` function is a data model independent form of the `_mbsncpy` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns:  
The `strncpy` function returns the value of `dst`.

See Also:  
`strlcpy`, `strcpy`, `strdup`, `strncpy_s`, `strcpy_s`

Example:  
```
#include <stdio.h>
#include <string.h>

void main( void )
{
    char buffer[15];
```


```c
printf( "%s\n", strncpy( buffer, "abcdefg", 10 ) );
printf( "%s\n", strncpy( buffer, "1234567",  6 ) );
printf( "%s\n", strncpy( buffer, "abcdefg",  3 ) );
printf( "%s\n", strncpy( buffer, "*******",  0 ) );
```

produces the following:

```
abcdefg
123456g
abc456g
abc456g
```

**Classification:**

- `strncpy` is ANSI
- `_fstrncpy` is not ANSI
- `wcsncpy` is ANSI
- `_mbsncpy` is not ANSI
- `_fmbsncpy` is not ANSI

**Systems:**

- `strncpy` - All, Netware
- `_fstrncpy` - All
- `wcsncpy` - All
- `_mbsncpy` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
- `_fmbsncpy` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:
#define __STDC_WANT_LIB_EXT1__  1
#include <string.h>
errno_t strncpy_s( char * restrict s1,
       rsize_t s1max,
       const char * restrict s2,
       rsize_t n );
#include <wchar.h>
errno_t wcsncpy_s( wchar_t * restrict s1,
       rsize_t s1max,
       const wchar_t * restrict s2,
       rsize_t n );

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler
will be invoked and strncpy_s will return a non-zero value to indicate an error, or the
runtime-constraint handler aborts the program.

Neither s1 nor s2 shall be a null pointer. Neither s1max nor n shall be greater than RSIZE_MAX.
s1max shall not equal zero. If n is not less than s1max, then s1max shall be greater than strlen_s(s2, s1max).

Copying shall not take place between objects that overlap.

If there is a runtime-constraint violation, then if s1 is not a null pointer and s1max is greater than zero
and not greater than RSIZE_MAX, then strncpy_s sets s1[0] to the null character.

Description: The strncpy_s function copies not more than n successive characters (characters that follow a null
character are not copied) from the array pointed to by s2 to the array pointed to by s1. If no null
character was copied from s2, then s1[n] is set to a null character.

All elements following the terminating null character (if any) written by strncpy_s in the array of
s1max characters pointed to by s1 take unspecified values when strncpy_s returns.

The wcsncpy_s function is a wide-character version of strncpy_s that operates with
wide-character strings.

Returns: The strncpy_s function returns zero if there was no runtime-constraint violation. Otherwise, a
non-zero value is returned.

See Also: strncpy, strlcpy, strcpy, strdup, strcpy_s

Example:
#define __STDC_WANT_LIB_EXT1__  1
#include <stdio.h>
#include <string.h>

void main( void )
{
    char buffer[15];
**strncpy_s, wcsncpy_s**

```c
strncpy_s( buffer, sizeof( buffer ), "abcdefg", 10 );
printf( "%s\n", buffer );

strncpy_s( buffer, sizeof( buffer ), "1234567", 6 );
printf( "%s\n", buffer );

strncpy_s( buffer, sizeof( buffer ), "abcdefg", 3 );
printf( "%s\n", buffer );

strncpy_s( buffer, sizeof( buffer ), "*******", 0 );
printf( "%s\n", buffer );
```

produces the following:

```
abcdefg
123456
abc
(nothing)
```

**Classification:**
- `strncpy_s` is TR 24731
- `wcsncpy_s` is TR 24731

**Systems:**
- `strncpy_s` - All, Netware
- `wcsncpy_s` - All
Synopsis:
#include <string.h>
int strnicmp( const char *s1,
        const char *s2,
        size_t len );
int _strnicmp( const char *s1,
               const char *s2,
               size_t len );
int _fstrnicmp( const char __far *s1,
                const char __far *s2,
                size_t len );
#include <wchar.h>
int _wcsnicmp( const wchar_t *s1,
               const wchar_t *s2,
               size_t len );
#include <mbstring.h>
int _mbsnicmp( const unsigned char *s1,
               const unsigned char *s2,
               size_t n );
int _fmbsnicmp( const unsigned char __far *s1,
                const unsigned char __far *s2,
                size_t n );

Description:
The strnicmp function compares, without case sensitivity, the string pointed to by s1 to the string pointed to by s2, for at most len characters.

The _strnicmp function is identical to strnicmp. Use _strnicmp for ANSI/ISO naming conventions.

The _fstrnicmp function is a data model independent form of the strnicmp function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The _wcsnicmp function is a wide-character version of strnicmp that operates with wide-character strings.

The _mbsnicmp function is a multibyte character version of strnicmp that operates with multibyte character strings.

The _fmbsnicmp function is a data model independent form of the _mbsnicmp function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns:
The strnicmp function returns an integer less than, equal to, or greater than zero, indicating that the string pointed to by s1 is less than, equal to, or greater than the string pointed to by s2.

See Also:
strcmp, stricmp, strncmp

Example:
#include <stdio.h>
#include <string.h>

void main()
{
    printf( "%d\n", strnicmp( "abcdef", "ABCXXX", 10 ) );
    printf( "%d\n", strnicmp( "abcdef", "ABCXXX", 6 ) );
    printf( "%d\n", strnicmp( "abcdef", "ABCXXX", 3 ) );
    printf( "%d\n", strnicmp( "abcdef", "ABCXXX", 0 ) );
}
produces the following:

-20
-20
0
0

Classification: WATCOM

_strnicmp conforms to ANSI/ISO naming conventions

Systems:

_strnicmp - All, Netware
_strnicmp - All, Netware
_fstrnicmp - All
wcsnicmp - All
_mbsnicmp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fmbsnicmp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

#include <string.h>
int _strnicoll( const char *s1,
    const char *s2,
    size_t count );

#include <wchar.h>
int _wcsnicoll( const wchar_t *s1,
    const wchar_t *s2,
    size_t count );

#include <mbstring.h>
int _mbsnicoll( const unsigned char *s1,
    const unsigned char *s2,
    size_t count );

Description: These functions perform a case insensitive comparison of the first count characters of the string pointed
to by s1 to the string pointed to by s2. The comparison uses the current code page which can be
selected by the _setmbcp function.

The _wcsnicoll function is a wide-character version of _strnicoll that operates with
wide-character strings.

The _mbsnicoll function is a multibyte character version of _strnicoll that operates with
multibyte character strings.

Returns: These functions return an integer less than, equal to, or greater than zero, indicating that the string
pointed to by s1 is less than, equal to, or greater than the string pointed to by s2, according to the
collating sequence selected.

See Also: _setmbcp, strcoll, stricmp, _stricoll, strncmp, _strncoll, strnicmp

Example: #include <stdio.h>
#include <string.h>
char buffer[80] = "world";

void main()
{
    int test;
    test = _strnicoll( buffer, "World2", 5 );
    if( test < 0 ) {
        printf( "Less than\n" );
    } else if( test == 0 ) {
        printf( "Equal\n" );
    } else {
        printf( "Greater than\n" );
    }
}

Classification: WATCOM

Systems: _strnicoll - All, Netware
    _wcsnicoll - All
    _mbsnicoll - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis: #include <tchar.h>
char *_strninc( const char *str, size_t count );
wchar_t *_wcsninc( const wchar_t *str, size_t count );
#include <mbstring.h>
unsigned char * _mbsninc( const unsigned char *str, size_t count );
unsigned char __far * _fmbsninc( const unsigned char __far *str, size_t count );

Description: The _mbsninc function increments str by count multibyte characters. _mbsninc recognizes multibyte-character sequences according to the multibyte code page currently in use. The header file <tchar.h> defines the generic-text routine _tcsninc. This macro maps to _mbsninc if _MBCS has been defined, or to _wcsninc if _UNICODE has been defined. Otherwise _tcsninc maps to _strninc. _strninc and _wcsninc are single-byte-character string and wide-character string versions of _mbsninc. _wcsninc and _strninc are provided only for this mapping and should not be used otherwise.

Returns: The _strninc function returns a pointer to str after it has been incremented by count characters or NULL if str was NULL. If count exceeds the number of characters remaining in the string, the result is undefined.

See Also: _strdec, _strinc

Example:
#include <stdio.h>
#include <mbctype.h>
#include <mbstring.h>

const unsigned char chars[] = {
    ' ', 
    '.', 
    '1', 
    'A',
    0x81,0x40, /* double-byte space */
    0x82,0x60, /* double-byte A */
    0x82,0xA6, /* double-byte Hiragana */
    0x83,0x42, /* double-byte Katakana */
    0xA1,      /* single-byte Katakana punctuation */
    0xA6,      /* single-byte Katakana alphabetic */
    0xDF,      /* single-byte Katakana alphabetic */
    0xE0,0xA1, /* double-byte Kanji */
    0x00
};

#define SIZE sizeof( chars ) / sizeof( unsigned char )

void main()
{
    int j, k;
    const unsigned char *next;

    _setmbcp( 932 );
    next = chars;
    do {
        next = _mbsninc( next, 1 );
        j = mblen( next, MB_CUR_MAX );
        if( j == 0 ) {
            k = 0;
        } else if ( j == 1 ) {
            k = *next;
        } else if( j == 2 ) {
            k = *(next)<<8 | *(next+1);
        }
        printf( "Next character %#6.4x\n", k );
    } while( next != &chars[ SIZE - 1 ] );
}

produces the following:

Next character 0x002e
Next character 0x0031
Next character 0x0041
Next character 0x8140
Next character 0x8260
Next character 0x82a6
Next character 0x8342
Next character 0x00a1
Next character 0x00a6
Next character 0x00df
Next character 0xe0a1
Next character 0x0000
Classification: WATCOM

Systems:  
_strninc - MACRO
_wcsninc - MACRO
_mbsninc - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fmbsninc - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:
#include <string.h>
char *strnset( char *str, int fill, size_t count );
char *strnset( char *str, int fill, size_t count );
char __far *_fstrnset( char __far *str,
                        int fill,
                        size_t count );
#include <wchar.h>
wchar_t *_wcsnset( wchar_t *str, int fill, size_t count );
#include <mbstring.h>
unsigned char *mbsnset( unsigned char *str,
                        unsigned int fill,
                        size_t count );
unsigned char __far *fmbsnset( unsigned char __far *str,
                               unsigned int fill,
                               size_t __n );

Description:
The strnset function fills the string str with the value of the argument fill, converted to be a character value. When the value of count is greater than the length of the string, the entire string is filled. Otherwise, that number of characters at the start of the string are set to the fill character.

The _strnset function is identical to strnset. Use _strnset for ANSI naming conventions.

The _fstrnset function is a data model independent form of the strnset function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The _wcsnset function is a wide-character version of strnset that operates with wide-character strings. For _wcsnset, the value of count is the number of wide characters to fill. This is half the number of bytes.

The _mbsnset function is a multibyte character version of strnset that operates with multibyte character strings.

The _fmbsnset function is a data model independent form of the _mbsnset function that accepts far pointer arguments. It is most useful in mixed memory model applications.

For _mbsnset, the value of count is the number of multibyte characters to fill. If the number of bytes to be filled is odd and fill is a double-byte character, the partial byte at the end is filled with an ASCII space character.

Returns:
The address of the original string str is returned.

See Also:
strset

Example:
#include <stdio.h>
#include <string.h>
char source[] = { "A sample STRING" };

void main()
{
    printf( "%s\n", source );
    printf( "%s\n", strnset( source, '=', 100 ) );
    printf( "%s\n", strnset( source, '*', 7 ) );
}
produces the following:

A sample STRING
=================
***************

Classification: WATCOM

Systems:  
strnset - All, Netware
_strnset - All, Netware
_fstrnset - All
_wcsnset - All
_mbsnset - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fmbsnset - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

```c
#include <string.h>
char *strpbrk(const char *str, const char *charset);

char __far *__fstrpbrk(const char __far *str, const char __far *charset);

#include <wchar.h>
wchar_t *wcspbrk(const wchar_t *str, const wchar_t *charset);

#include <mbstring.h>
unsigned char *__mbspbrk(const unsigned char *str, const unsigned char *charset);
unsigned char __far *__fmbspbrk(const unsigned char __far *str, const unsigned char __far *charset);
```

Description:
The `strpbrk` function locates the first occurrence in the string pointed to by `str` of any character from the string pointed to by `charset`.

The `_fstrpbrk` function is a data model independent form of the `strpbrk` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `wcspbrk` function is a wide-character version of `strpbrk` that operates with wide-character strings.

The `_mbspbrk` function is a multibyte character version of `strpbrk` that operates with multibyte character strings.

The `_fmbspbrk` function is a data model independent form of the `_mbspbrk` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns:
The `strpbrk` function returns a pointer to the located character, or `NULL` if no character from `charset` occurs in `str`.

See Also:
`strchr`, `strrchr`, `strtok`

Example:
```
#include <stdio.h>
#include <string.h>

void main()
{
    char *p = "Find all vowels";
    while( p != NULL ) {
        printf( "%s\n", p );
        p = strpbrk( p+1, "aeiouAEIOU" );
    }
}
```

produces the following:

```
Find all vowels
ind all vowels
all vowels
owels
els
```
Classification:  
strpbrk is ANSI
_fstrpbrk is not ANSI
wcspbrk is ANSI
_mbstrpbrk is not ANSI
_fmbspbrk is not ANSI

Systems:  
strpbrk - All, Netware
_fstrpbrk - All
wcspbrk - All
_mbstrpbrk - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fmbspbrk - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

```c
#include <string.h>
char *strrchr( const char *s, int c );
char __far *_fstrrchr( const char __far *s, int c );
#include <wchar.h>
wchar_t *wcsrchr( const wchar_t *s, wint_t c );
#include <mbstring.h>
unsigned char *__far *_mbsrchr( const unsigned char __far *s,
                                  unsigned int c );
unsigned char __far *__mbsrchr( const unsigned char __far *s,
                                unsigned int c );
```

Description:

The `strrchr` function locates the last occurrence of `c` (converted to a char) in the string pointed to by `s`. The terminating null character is considered to be part of the string.

The `_fstrrchr` function is a data model independent form of the `strrchr` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `wcsrchr` function is a wide-character version of `strrchr` that operates with wide-character strings.

The `_mbsrchr` function is a multibyte character version of `strrchr` that operates with multibyte character strings.

The `_fmbsrchr` function is a data model independent form of the `_mbsrchr` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns:

The `strrchr` function returns a pointer to the located character, or a NULL pointer if the character does not occur in the string.

See Also:

`strchr, strpbrk`

Example:

```c
#include <stdio.h>
#include <string.h>

void main()
{
    printf( "%s\n", strrchr( "abcdeaaklnn", 'a' ) );
    if( strrchr( "abcdeaaklnn", 'x' ) == NULL )
        printf( "NULL\n" );
}

produces the following:

aklmn
NULL
```

Classification:

- `strrchr` is ANSI
- `_fstrrchr` is not ANSI
- `wcsrchr` is ANSI
- `_mbsrchr` is not ANSI
- `_fmbsrchr` is not ANSI

Systems:

- `strrchr` - All, Netware
strrchr, _fstrrchr, wcsrchr, _mbsrchr, _fmbsrchr

_fstrrchr - All
wcstrchr - All
_mbsrchr - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fmbsrchr - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

```c
#include <string.h>
char *strrev( char *s1 );
char *__strrev( char *s1 );
char __far *__fstrrev( char __far *s1 );
#include <wchar.h>
wchar_t *__wcsrev( wchar_t *s1 );
#include <mbstring.h>
unsigned char *__mbsrev( unsigned char *s1 );
unsigned char __far *__fmbsrev( unsigned char __far *s1 );
```

Description:
The `strrev` function replaces the string `s1` with a string whose characters are in the reverse order.

The `_strrev` function is identical to `strrev`. Use `_strrev` for ANSI/ISO naming conventions.

The `_fstrrev` function is a data model independent form of the `strrev` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `_wcsrev` function is a wide-character version of `strrev` that operates with wide-character strings.

The `_mbsrev` function is a multibyte character version of `strrev` that operates with multibyte character strings.

The `_fmbsrev` function is a data model independent form of the `_mbsrev` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns:
The address of the original string `s1` is returned.

Example:

```c
#include <stdio.h>
#include <string.h>

char source[] = { "A sample STRING" };

void main()
{
    printf( "%s\n", source );
    printf( "%s\n", strrev( source ) );
    printf( "%s\n", _strrev( source ) );
}
```

produces the following:

```
A sample STRING
GNIRTS elpmas A
A sample STRING
```

Classification: WATCOM

`strrev` conforms to ANSI/ISO naming conventions

Systems: `strrev` - All, Netware

`strrev` - All, Netware

`_fstrrev` - All

`_wcsrev` - All

`_mbsrev` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
strrev, _strrev, _fstrrev, _wcsrev, _mbsrev, _fmbsrev

_fmbrev - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

```
#include <string.h>
char *strset( char *s1, int fill );
char *__strset( char *s1, int fill );
char __far *__fstrset( char __far *s1, int fill );
#include <wchar.h>
wchar_t *__wcsset( wchar_t *s1, int fill );
#include <mbstring.h>
unsigned char *__mbsset( unsigned char *s1,
                        unsigned int fill );
unsigned char __far *__fmbsset( unsigned char __far *s1,
                                unsigned int fill );
```

Description:

The `strset` function fills the string pointed to by `s1` with the character `fill`. The terminating null character in the original string remains unchanged.

The `_strset` function is identical to `strset`. Use `_strset` for ANSI naming conventions.

The `_fstrset` function is a data model independent form of the `strset` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `_wcsset` function is a wide-character version of `strset` that operates with wide-character strings.

The `_mbsset` function is a multibyte character version of `strset` that operates with multibyte character strings.

The `_fmbsset` function is a data model independent form of the `_mbsset` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns:

The address of the original string `s1` is returned.

See Also: `strnset`

Example:

```
#include <stdio.h>
#include <string.h>

char source[] = { "A sample STRING" };

void main()
{
    printf( "%s\n", source );
    printf( "%s\n", strset( source, '=' ) );
    printf( "%s\n", strset( source, '*' ) );
}
```

produces the following:

```
A sample STRING
===============
***************
```

Classification: WATCOM

Systems: `strset` - All, Netware
_strset - All, Netware
_fstrset - All
_wcsset - All
_mbsset - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fmbsset - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  

```c
#include <string.h>
size_t strspn( const char *str,
               const char *charset );
size_t _fstrspn( const char __far *str,
                const char __far *charset );
```

```c
#include <wchar.h>
size_t wcsspn( const wchar_t *str,
               const wchar_t *charset );
#include <wchar.h>
size_t _mbsspn( const unsigned char *str,
               const unsigned char *charset );
size_t _fmbsspn( const unsigned char __far *str,
                const unsigned char __far *charset );
```

Description:  
The `strspn` function computes the length, in bytes, of the initial segment of the string pointed to by `str` which consists of characters from the string pointed to by `charset`. The terminating null character is not considered to be part of `charset`.

The `_fstrspn` function is a data model independent form of the `strspn` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The `wcsspn` function is a wide-character version of `strspn` that operates with wide-character strings.

The `_mbsspn` function is a multibyte character version of `strspn` that operates with multibyte character strings.

The `_fmbsspn` function is a data model independent form of the `_mbsspn` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns:  
The length, in bytes, of the initial segment is returned.

See Also:  
`strcspn`, `strspnp`

Example:  
```c
#include <stdio.h>
#include <string.h>

void main()
{
    printf( "%d\n", strspn( "out to lunch", "aeiou" ) );
    printf( "%d\n", strspn( "out to lunch", "xyz" ) );
}
```

produces the following:

```
2
0
```

Classification:  
`strspn` is ANSI  
`_fstrspn` is not ANSI  
`wcsspn` is ANSI  
`_mbsspn` is not ANSI  
`_fmbsspn` is not ANSI  

Systems:  
`strspn` - All, Netware
strspn, _fstrspn, wcspn, _mbsspn, _fmbsspn

_fstrspn - All
wcspn - All
_mbsspn - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fmbsspn - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

```
#include <string.h>
char *strspnp( const char *str, 
                const char *charset );
char __*strspnp( const char *str, 
                const char *charset );
char __far *__fstrspnp( const char __far *str, 
                        const char __far *charset );
#include <tchar.h>
wchar_t *__wcsspnp( const wchar_t *str, 
                  const wchar_t *charset );
#include <mbstring.h>
unsigned char *__mbsspnp( const unsigned char *str, 
                          const unsigned char *charset );
unsigned char __far *__fmbsspnp( 
                        const unsigned char __far *str, 
                          const unsigned char __far *charset );
```

Description: The `strspnp` function returns a pointer to the first character in `str` that does not belong to the set of characters in `charset`. The terminating null character is not considered to be part of `charset`.

The `_strspnp` function is identical to `strspnp`. Use `_strspnp` for ANSI/ISO naming conventions.

The `_fstrspnp` function is a data model independent form of the `strspnp` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The `_wcsspnp` function is a wide-character version of `strspnp` that operates with wide-character strings.

The `_mbsspnp` function is a multibyte character version of `strspnp` that operates with multibyte character strings.

The `_fmbsspnp` function is a data model independent form of the `_mbsspnp` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The `strspnp` function returns `NULL` if `str` consists entirely of characters from `charset`.

See Also: `strcspn`, `strspn`

Example: ```
#include <stdio.h>
#include <string.h>

void main()
{
    printf( "%s\n", strspnp( "out to lunch", "aeiou" ) );
    printf( "%s\n", strspnp( "out to lunch", "xyz" ) );
}
```
produces the following:

t to lunch
out to lunch

Classification: WATCOM
strspnp, _strspnp, _fstrspnp, _wcspnp, _mbsspnp, _fmbsspnp

_strspnp conforms to ANSI/ISO naming conventions

Systems:

  strspnp - All, Netware
  _strspnp - All, Netware
  _fstrspnp - All
  _wcspnp - All
  _mbsspnp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
  _fmbsspnp - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

```
#include <string.h>
char *strstr( const char *str,
               const char *substr );
char __far *__fstrstr( const char __far *str,
                       const char __far *substr );
#include <wchar.h>
wchar_t *wcsstr( const wchar_t *str,
                 const wchar_t *substr );
#include <mbstring.h>
unsigned char *__mbsstr( const unsigned char *str,
                        const unsigned char *substr );
unsigned char __far *__fmbsstr(
                        const unsigned char __far *str,
                        const unsigned char __far *substr );
```

Description:
The `strstr` function locates the first occurrence in the string pointed to by `str` of the sequence of characters (excluding the terminating null character) in the string pointed to by `substr`.

The `_fstrstr` function is a data model independent form of the `strstr` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `wcsstr` function is a wide-character version of `strstr` that operates with wide-character strings.

The `_mbsstr` function is a multibyte character version of `strstr` that operates with multibyte character strings.

The `_fmbsstr` function is a data model independent form of the `_mbsstr` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns:
The `strstr` function returns a pointer to the located string, or `NULL` if the string is not found.

See Also: `strcspn`

Example:

```
#include <stdio.h>
#include <string.h>

void main()
{
    printf( "%s\n", strstr("This is an example", "is") );
}
```

produces the following:

```
is is an example
```

Classification:

- `strstr` is ANSI
- `_fstrstr` is not ANSI
- `wcsstr` is ANSI
- `_mbsstr` is not ANSI
- `_fmbsstr` is not ANSI

Systems:

- `strstr` - All, Netware
- `_fstrstr` - All
- `wcsstr` - All
strstr, _fstrstr, wcsstr, _mbsstr, _fmbsstr

_mbsstr - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
_fmbsstr - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:    
#include <time.h>
char * _strtime( char * timestr )
wchar_t _wstrtime( wchar_t * timestr );

Description: The _strtime function copies the current time to the buffer pointed to by timestr. The time is formatted as "HH:MM:SS" where "HH" is two digits representing the hour in 24-hour notation, where "MM" is two digits representing the minutes past the hour, and where "SS" is two digits representing seconds. The buffer must be at least 9 bytes long.

The _wstrtime function is a wide-character version of _strtime that operates with wide-character strings.

Returns: The _strtime function returns a pointer to the resulting text string timestr.

See Also:asctime Functions, ctime Functions, gmtime, localtime, mktime, _strdate, time, tzset

Example:    
#include <stdio.h>
#include <time.h>

void main()
{
    char timebuff[9];

    printf( "%s\n", _strtime( timebuff ) );
}

Classification: WATCOM

Systems: _strtime - All
         _wstrtime - All
Synopsis: #include <stdlib.h>
double strtod( const char *ptr, char **endptr );
#include <wchar.h>
double wcstod( const wchar_t *ptr, wchar_t **endptr );

Description: The `strtod` function converts the string pointed to by `ptr` to `double` representation. First, it decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters (as specified by the `isspace` function), a subject sequence resembling a floating-point constant or representing an infinity or NaN; and a final string of one or more unrecognized characters, including the terminating null character of the input string. Then, it attempts to convert the subject sequence to a floating-point number, and return the result.

The expected form of the subject sequence is an optional plus or minus sign, then one of the following:

- a decimal floating-point number
- a hexadecimal floating-point number
- `INF` or `INFINITY`, ignoring case
- `NAN`, ignoring case, optionally followed by a sequence of digits and nondigits (upper- or lowercase characters or underscore) enclosed in parentheses.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-whitespace character, that is of the expected form. The subject sequence contains no characters if the input string is not of the expected form.

A decimal floating-point number recognized by `strtod` (after optional sign was processed) is a string containing:

- a sequence of digits containing an optional decimal point,
- an optional 'e' or 'E' followed by an optionally signed sequence of digits.

A hexadecimal floating-point number recognized by `strtod` (after optional sign was processed) is a string containing:

- a `0X` prefix, ignoring case,
- a sequence of hexadecimal digits containing an optional decimal point,
- an optional 'p' or 'P' followed by an optionally signed sequence of decimal digits.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is not of the expected form.

If the subject sequence contains `NAN`, a NaN (with appropriate sign) will be returned the optional digit-nondigit sequence is ignored. If the subject sequence contains `INF`, the value of infinity (with appropriate sign) will be returned. This case can be distinguished from overflow by checking `errno`.

For a hexadecimal floating-point number, the optional exponent is binary (that is, denotes a power of two), not decimal.

A pointer to the final string (following the subject sequence) will be stored in the object to which `endptr` points if `endptr` is not NULL. By comparing the "end" pointer with `ptr`, it can be determined how much of the string, if any, was scanned by the `strtod` function.

The `wcstod` function is a wide-character version of `strtod` that operates with wide-character strings.
Returns: The `strtod` function returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value would cause overflow, plus or minus `HUGE_VAL` is returned according to the sign, and `errno` is set to `ERANGE`. If the correct value would cause underflow, then zero is returned, and `errno` is set to `ERANGE`. Zero is returned when the input string cannot be converted. In this case, `errno` is not set. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `atof`

Example: 
```c
#include <stdio.h>
#include <stdlib.h>

void main( void )
{
  double pi;

  pi = strtod( "3.141592653589793", NULL );
  printf( "pi=%17.15f\n",pi );
}
```

Classification: `strtod` is ISO C90
`wcstod` is ISO C95

Systems: `strtod` — Math
`wcstod` — Math
Synopsis:

```c
#include <string.h>
char *strtok( char *s1, const char *s2 );
char __far *__fstrtok( char __far *s1, 
                     const char __far *s2 );
#include <wchar.h>
wchar_t *wcstok( wchar_t *s1, const wchar_t *s2, 
                wchar_t **ptr );
#include <mbstring.h>
unsigned char *__mbstok( unsigned char *s1, 
                        const unsigned char *s2 );
unsigned char __far *__fmbstok( unsigned char __far *s1, 
                               const unsigned char __far *s2 );
```

Safer C: The Safer C Library extension provides the `strtok_s` function which is a safer alternative to `strtok`. This newer `strtok_s` function is recommended to be used instead of the traditional "unsafe" `strtok` function.

Description: The `strtok` function is used to break the string pointed to by `s1` into a sequence of tokens, each of which is delimited by a character from the string pointed to by `s2`. The first call to `strtok` will return a pointer to the first token in the string pointed to by `s1`. Subsequent calls to `strtok` must pass a NULL pointer as the first argument, in order to get the next token in the string. The set of delimiters used in each of these calls to `strtok` can be different from one call to the next.

The first call in the sequence searches `s1` for the first character that is not contained in the current delimiter string `s2`. If no such character is found, then there are no tokens in `s1` and the `strtok` function returns a NULL pointer. If such a character is found, it is the start of the first token.

The `strtok` function then searches from there for a character that is contained in the current delimiter string. If no such character is found, the current token extends to the end of the string pointed to by `s1`. If such a character is found, it is overwritten by a null character, which terminates the current token. The `strtok` function saves a pointer to the following character, from which the next search for a token will start when the first argument is a NULL pointer.

Because `strtok` may modify the original string, that string should be duplicated if the string is to be re-used.

The `_fstrtok` function is a data model independent form of the `strtok` function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The `wcstok` function is a wide-character version of `strtok` that operates with wide-character strings. The third argument `ptr` points to a caller-provided `wchar_t` pointer into which the `wcstok` function stores information necessary for it to continue scanning the same wide string.

On the first call in the sequence of calls to `wcstok`, `s1` points to a wide string. In subsequent calls for the same string, `s1` must be NULL. If `s1` is NULL, the value pointed to by `ptr` matches that set by the previous call to `wcstok` for the same wide string. Otherwise, the value of `ptr` is ignored. The list of delimiters pointed to by `s2` may be different from one call to the next. The tokenization of `s1` is similar to that for the `strtok` function.

The `_mbstok` function is a multibyte character version of `strtok` that operates with multibyte character strings.

The `_fmbstok` function is a data model independent form of the `_mbstok` function that accepts far pointer arguments. It is most useful in mixed memory model applications.
Returns: The `strtok` function returns a pointer to the first character of a token or NULL if there is no token found.

See Also: `strcspn`, `strpbrk`, `strtok_s`

Example: ```c
#include <stdio.h>
#include <string.h>

void main()
{
    char *p;
    char *buffer;
    char *delims = { " .," };

    buffer = strdup( "Find words, all of them." );
    printf( "%s\n", buffer );
    p = strtok( buffer, delims );
    while( p != NULL ) {
        printf( "word: %s\n", p );
        p = strtok( NULL, delims );
    }
    printf( "%s\n", buffer );
}
``` produces the following:

Find words, all of them.
word: Find
word: words
word: all
word: of
word: them
Find

Classification: `strtok` is ANSI
`_fstrtok` is not ANSI
`wcstok` is ANSI
`_mbstok` is not ANSI
`_fmbstok` is not ANSI

Systems: `strtok` - All, Netware
`_fstrtok` - All
`wcstok` - All
`_mbstok` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
`_fmbstok` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
**Synopsis:**

```c
#define __STDC_WANT_LIB_EXT1__ 1
#include <string.h>
char *strtok_s( char * restrict s1, 
               rsize_t * restrict s1max, 
               const char * restrict s2, 
               char ** restrict ptr);

#include <wchar.h>
wchar_t *wcstok_s( wchar_t * restrict s1, 
                  rsize_t * restrict s1max, 
                  const wchar_t * restrict s2, 
                  wchar_t ** restrict ptr);
```

**Constraints:**

If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `strtok_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

None of `s1max`, `s2`, or `ptr` shall be a null pointer. If `s1` is a null pointer, then `*ptr` shall not be a null pointer. The value of `*s1max` shall not be greater than `RSIZE_MAX`. The end of the token found shall occur within the first `*s1max` characters of `s1` for the first call, and shall occur within the first `*s1max` characters of where searching resumes on subsequent calls. If there is a runtime-constraint violation, the `strtok_s` function does not indirect through the `s1` or `s2` pointers, and does not store a value in the object pointed to by `ptr`.

**Description:**

A sequence of calls to the `strtok_s` function breaks the string pointed to by `s1` into a sequence of tokens, each of which is delimited by a character from the string pointed to by `s2`. The fourth argument points to a caller-provided char pointer into which the `strtok_s` function stores information necessary for it to continue scanning the same string. The first call in a sequence has a non-null first argument and `s1max` points to an object whose value is the number of elements in the character array pointed to by the first argument. The first call stores an initial value in the object pointed to by `ptr` and updates the value pointed to by `s1max` to reflect the number of elements that remain in relation to `ptr`. Subsequent calls in the sequence have a null first argument and the objects pointed to by `s1max` and `ptr` are required to have the values stored by the previous call in the sequence, which are then updated. The separator string pointed to by `s2` may be different from call to call. The first call in the sequence searches the string pointed to by `s1` for the first character that is not contained in the current separator string pointed to by `s2`. If no such character is found, then there are no tokens in the string pointed to by `s1` and the `strtok_s` function returns a null pointer. If such a character is found, it is the start of the first token. The `strtok_s` function then searches from there for the first character in `s1` that is contained in the current separator string. If no such character is found, the current token extends to the end of the string pointed to by `s1`, and subsequent searches in the same string for a token return a null pointer. If such a character is found, it is overwritten by a null character, which terminates the current token. In all cases, the `strtok_s` function stores sufficient information in the pointer pointed to by `ptr` so that subsequent calls, with a null pointer for `s1` and the unmodified pointer value for `ptr`, shall start searching just past the element overwritten by a null character (if any).

The `wcstok_s` function is a wide-character version of `strtok_s` that operates with wide-character strings.

**Returns:**

The `strtok_s` function returns a pointer to the first character of a token, or a null pointer if there is no token or there is a runtime-constraint violation.

**See Also:** `strtok`, `strcspn`, `strpbrk`
Example:  
#define __STDC_WANT_LIB_EXT1__ 1  
#include <stdio.h>  
#include <string.h>  

void main( void )  
{  
    char    *p;  
    char    *buffer;  
    char    *delims = { " .," };  
    size_t  buflen;  
    char    *ptr;  

    buffer = strdup( "Find words, all of them." );  
    printf( "%s\n", buffer );  
    buflen = strlen( buffer );  
    p = strtok_s( buffer, &buflen, delims, &ptr );  
    while( p != NULL ) {  
        printf( "word: %s\n", p );  
        p = strtok_s( NULL, &buflen, delims, &ptr );  
    }  
    printf( "%s\n", buffer );  
}

produces the following:

Find words, all of them.
word: Find
word: words
word: all
word: of
word: them
Find

Classification:  strtok_s is TR 24731
wcstok_s is TR 24731

Systems:  strtok_s - All, Netware
wcstok_s - All
Synopsis:

```c
#include <stdlib.h>
long int strtol( const char *ptr,
                 char **endptr,
                 int base );

#include <wchar.h>
long int wcstol( const wchar_t *ptr,
                 wchar_t **endptr,
                 int base );
```

Description: The `strtol` function converts the string pointed to by `ptr` to an object of type `long int`. The `strtol` function recognizes a string containing:

- optional white space,
- an optional plus or minus sign,
- a sequence of digits and letters.

The conversion ends at the first unrecognized character. A pointer to that character will be stored in the object to which `endptr` points if `endptr` is not `NULL`.

If `base` is zero, the first characters after the optional sign determine the base used for the conversion. If the first characters are "0x" or "0X" the digits are treated as hexadecimal. If the first character is '0', the digits are treated as octal. Otherwise the digits are treated as decimal.

If `base` is not zero, it must have a value between 2 and 36. The letters a-z and A-Z represent the values 10 through 35. Only those letters whose designated values are less than `base` are permitted. If the value of `base` is 16, the characters "0x" or "0X" may optionally precede the sequence of letters and digits.

The `wcstol` function is a wide-character version of `strtol` that operates with wide-character strings.

Returns: The `strtol` function returns the converted value. If the correct value would cause overflow, `LONG_MAX` or `LONG_MIN` is returned according to the sign, and `errno` is set to `ERANGE`. If `base` is out of range, zero is returned and `errno` is set to `EDOM`.

See Also: `atoi`, `atol`, `atoll`, `itoa`, `ltoa`, `lltoa`, `sscanf`, `strtol`, `strtol`, `strtoull`, `strtoimax`, `strtoimax`, `ultoa`, `ulltoa`, `utoa`

Example:

```c
#include <stdlib.h>

void main()
{
    long int v;

    v = strtol( "12345678", NULL, 10 );
}
```

Classification: `strtol` is ANSI
`wcstol` is ANSI

Systems: `strtol` - All, Netware
`wcstol` - All
Synopsis:

```c
#include <stdlib.h>
long long int strtoll( const char *ptr,
                        char **endptr,
                        int base );

#include <wchar.h>
long long int wcstoll( const wchar_t *ptr,
                        wchar_t **endptr,
                        int base );
```

Description: The `strtoll` function converts the string pointed to by `ptr` to an object of type `long long int`. The `strtoll` function recognizes a string containing:

- optional white space,
- an optional plus or minus sign,
- a sequence of digits and letters.

The conversion ends at the first unrecognized character. A pointer to that character will be stored in the object to which `endptr` points if `endptr` is not `NULL`.

If `base` is zero, the first characters after the optional sign determine the base used for the conversion. If the first characters are "0x" or "0X" the digits are treated as hexadecimal. If the first character is '0', the digits are treated as octal. Otherwise the digits are treated as decimal.

If `base` is not zero, it must have a value between 2 and 36. The letters a-z and A-Z represent the values 10 through 35. Only those letters whose designated values are less than `base` are permitted. If the value of `base` is 16, the characters "0x" or "0X" may optionally precede the sequence of letters and digits.

The `wcstoll` function is a wide-character version of `strtoll` that operates with wide-character strings.

Returns: The `strtoll` function returns the converted value. If the correct value would cause overflow, `LLONG_MAX` or `LLONG_MIN` is returned according to the sign, and `errno` is set to `ERANGE`. If `base` is out of range, zero is returned and `errno` is set to `EDOM`.

See Also: `atoi`, `atol`, `atoll`, `itoa`, `ltoa`, `lltoa`, `sscanf`, `strtol`, `strtoul`, `strtoull`, `strtoimax`, `strtoumax`, `ultoa`, `ulltoa`, `utoa`

Example:

```c
#include <stdlib.h>

void main()
{
  long long int v;
  v = strtoll( "12345678909876", NULL, 10 );
}
```

Classification: `strtoll` is ANSI
`wcstoll` is ANSI

Systems: `strtoll` - All, Netware
`wcstoll` - All
**strtoimax, wcstomax**

**Synopsis:**

```c
#include <stdint.h>
intmax_t strtoimax( const char *ptr,
                     char **endptr,
                     int base );

#include <stdint.h>
intmax_t wcstomax( const wchar_t *ptr,
                   wchar_t **endptr,
                   int base );
```

**Description:**
The `strtoimax` function converts the string pointed to by `ptr` to an object of type `intmax_t`. The `strtoimax` function recognizes a string containing:

- optional white space,
- an optional plus or minus sign,
- a sequence of digits and letters.

The conversion ends at the first unrecognized character. A pointer to that character will be stored in the object to which `endptr` points if `endptr` is not `NULL`.

If `base` is zero, the first characters after the optional sign determine the base used for the conversion. If the first characters are "0x" or "0X" the digits are treated as hexadecimal. If the first character is '0', the digits are treated as octal. Otherwise the digits are treated as decimal.

If `base` is not zero, it must have a value between 2 and 36. The letters a-z and A-Z represent the values 10 through 35. Only those letters whose designated values are less than `base` are permitted. If the value of `base` is 16, the characters "0x" or "0X" may optionally precede the sequence of letters and digits.

The `wcstomax` function is a wide-character version of `strtoimax` that operates with wide-character strings.

**Returns:**
The `strtoimax` function returns the converted value. If the correct value would cause overflow, `INTMAX_MAX` or `INTMAX_MIN` is returned according to the sign, and `errno` is set to `ERANGE`. If `base` is out of range, zero is returned and `errno` is set to `EDOM`.

**See Also:**
`atoi`, `atol`, `atoll`, `itoa`, `ltoa`, `lltoa`, `sscanf`, `strtol`, `strtoll`, `strtoul`, `strtoull`, `strtoimax`, `ultoa`, `ulltoa`, `utoa`

**Example:**

```c
#include <stdio.h>
#include <stdlib.h>

void main()
{
    intmax_t v;
    v = strtoimax( "12345678909876", NULL, 10 );
}
```

**Classification:**
`strtoimax` is ANSI
`wcstomax` is ANSI

**Systems:**
`strtoimax` - All, Netware
`wcstomax` - All
Synopsis: #include <stdlib.h>
unsigned long int strtoul( const char *ptr,
                           char **endptr,
                           int base );

#include <wchar.h>
unsigned long int wcstoul( const wchar_t *ptr,
                           wchar_t **endptr,
                           int base );

Description: The strtoul function converts the string pointed to by ptr to an unsigned long. The function recognizes a string containing optional white space, an optional sign (+ or -), followed by a sequence of digits and letters. The conversion ends at the first unrecognized character. A pointer to that character will be stored in the object endptr points to if endptr is not NULL.

If base is zero, the first characters determine the base used for the conversion. If the first characters are "0x" or "0X" the digits are treated as hexadecimal. If the first character is '0', the digits are treated as octal. Otherwise the digits are treated as decimal.

If base is not zero, it must have a value of between 2 and 36. The letters a-z and A-Z represent the values 10 through 35. Only those letters whose designated values are less than base are permitted. If the value of base is 16, the characters "0x" or "0X" may optionally precede the sequence of letters and digits.

If there is a leading minus sign in the string, the value is negated.

The wcstoul function is a wide-character version of strtoul that operates with wide-character strings.

Returns: The strtoul function returns the converted value. If the correct value would cause overflow, ULONG_MAX is returned and errno is set to ERANGE. If base is out of range, zero is returned and errno is set to EDOM.

See Also: atoi, atol, atol, ltoa, lltoa, sscanf, strtol, strtoll, strtoull, strtoimax, strtoumax, ultoa, ulltoa, utoa

Example: #include <stdlib.h>

void main()
{
    unsigned long int v;
    v = strtoul( "12345678", NULL, 10 );
}

Classification: strtoul is ANSI
wcstoul is ANSI

Systems: strtoul - All, Netware
wcstoul - All
strtolll, wcstolll

Synopsis:

```
#include <stdlib.h>
unsigned long long int strtolll(const char *ptr,
                                char **endptr,
                                int base);

#include <wchar.h>
unsigned long long int wcstolll(const wchar_t *ptr,
                                wchar_t **endptr,
                                int base);
```

Description: The `strtolll` function converts the string pointed to by `ptr` to an `unsigned long long`. The function recognizes a string containing optional white space, an optional sign (+ or -), followed by a sequence of digits and letters. The conversion ends at the first unrecognized character. A pointer to that character will be stored in the object `endptr` points to if `endptr` is not `NULL`.

If `base` is zero, the first characters determine the base used for the conversion. If the first characters are "0x" or "0X" the digits are treated as hexadecimal. If the first character is '0', the digits are treated as octal. Otherwise the digits are treated as decimal.

If `base` is not zero, it must have a value of between 2 and 36. The letters a-z and A-Z represent the values 10 through 35. Only those letters whose designated values are less than `base` are permitted. If the value of `base` is 16, the characters "0x" or "0X" may optionally precede the sequence of letters and digits.

If there is a leading minus sign in the string, the value is negated.

The `wcstolll` function is a wide-character version of `strtolll` that operates with wide-character strings.

Returns: The `strtolll` function returns the converted value. If the correct value would cause overflow, `ULLONG_MAX` is returned and `errno` is set to `ERANGE`. If `base` is out of range, zero is returned and `errno` is set to `EDOM`.

See Also: `atoi`, `atol`, `atoll`, `itoa`, `ltoa`, `lltoa`, `sscanf`, `strotl`, `strtol`, `strtoll`, `strtol`, `strtoimax`, `strtoumax`, `ultoa`, `ulltoa`, `utoa`

Example:

```
#include <stdlib.h>

void main()
{
    unsigned long long int v;
    v = strtoll( "12345678909876", NULL, 10 );
}
```

Classification: `strtolll` is ANSI
`wcstolll` is ANSI

Systems: `strtolll` - All, Netware
`wcstolll` - All
Synopsis:  
#include <inttypes.h>

uintmax_t strtoumax( const char *ptr,
                     char **endptr,
                     int base );

#include <inttypes.h>
uintmax_t wcstoumax( const wchar_t *ptr,
                     wchar_t **endptr,
                     int base );

Description:  The **strtoumax** function converts the string pointed to by *ptr* to an uintmax_t. The function recognizes a string containing optional white space, an optional sign (+ or -), followed by a sequence of digits and letters. The conversion ends at the first unrecognized character. A pointer to that character will be stored in the object *endptr* points to if *endptr* is not NULL.

If *base* is zero, the first characters determine the base used for the conversion. If the first characters are "0x" or "0X" the digits are treated as hexadecimal. If the first character is '0', the digits are treated as octal. Otherwise the digits are treated as decimal.

If *base* is not zero, it must have a value of between 2 and 36. The letters a-z and A-Z represent the values 10 through 35. Only those letters whose designated values are less than *base* are permitted. If the value of *base* is 16, the characters "0x" or "0X" may optionally precede the sequence of letters and digits.

If there is a leading minus sign in the string, the value is negated.

The **wcstoumax** function is a wide-character version of **strtoumax** that operates with wide-character strings.

Returns:  The **strtoumax** function returns the converted value. If the correct value would cause overflow, UINTMAX_MAX is returned and **errno** is set to ERANGE. If *base* is out of range, zero is returned and **errno** is set to EDOM.

See Also:  atoi, atol, atoll, itoa, ltoa, lltoa, sscanf, strtol, strtoll, strtoul, strtoull, strtoimax, ultoa, ulltoa, utoa

Example:  
#include <inttypes.h>
#include <stdlib.h>

void main()
{
    uintmax_t v;

    v = strtoumax( "12345678909876", NULL, 10 );
}

Classification:  **strtoumax** is ANSI
               **wcstoumax** is ANSI

Systems:  **strtoumax** - All, Netware
          **wcstoumax** - All

Library Functions and Macros 1007
Synopsis:  
#include <string.h>
char *strupr( char *s );
char *__strupr( char *s );
char __far *__fstrupr( char __far *s );
#include <wchar.h>
wchar_t *__wcsupr( wchar_t *s );
#include <mbsstring.h>
unsigned char *__mbsupr( unsigned char *s );
unsigned char __far *__fmbsupr( unsigned char __far *s );

Description:  The \texttt{strupr} function replaces the string \texttt{s} with uppercase characters by invoking the \texttt{toupper} function for each character in the string.

The \texttt{__strupr} function is identical to \texttt{strupr}. Use \texttt{__strupr} for ANSI/ISO naming conventions.

The \texttt{__fstrupr} function is a data model independent form of the \texttt{strupr} function. It accepts far pointer arguments and returns a far pointer. It is most useful in mixed memory model applications.

The \texttt{__wcsupr} function is a wide-character version of \texttt{strupr} that operates with wide-character strings.

The \texttt{__mbsupr} function is a multibyte character version of \texttt{strupr} that operates with multibyte character strings.

Returns:  The address of the original string \texttt{s} is returned.

See Also:  \texttt{strlwr}

Example:  
#include <stdio.h>
#include <string.h>

char source[] = { "A mixed-case STRING" };

void main()
{
    printf( "%s\n", source );
    printf( "%s\n", strupr( source ) );
    printf( "%s\n", source );
}

produces the following:

A mixed-case STRING
A MIXED-CASE STRING
A MIXED-CASE STRING

Classification:  WATCOM
斯特普尔 conforms to ANSI/ISO naming conventions

Systems:  
strупr - All, Netware
__strупr - All, Netware
__fstrupr - All
__wcsupr - All
__mbsupr - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
strupr, _strupr, _fstrupr, _wcsupr, _mbsupr, _fmbsupr

_fmbsupr - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:
```
#include <string.h>
size_t strxfrm( char *dst,
    const char *src,
    size_t n );
#include <wchar.h>
size_t wcsxfrm( wchar_t *dst,
    const wchar_t *src,
    size_t n );
```

Description:
The `strxfrm` function transforms, for no more than `n` characters, the string pointed to by `src` to the buffer pointed to by `dst`. The transformation uses the collating sequence selected by the `setlocale` function so that two transformed strings will compare identically (using the `strncmp` function) to a comparison of the original two strings using the `strcoll` function. The function will be equivalent to the `strncpy` function (except there is no padding of the `dst` argument with null characters when the argument `src` is shorter than `n` characters) when the collating sequence is selected from the "C" locale.

The `wcsxfrm` function is a wide-character version of `strxfrm` that operates with wide-character strings. For `wcsxfrm`, after the string transformation, a call to `wcscmp` with the two transformed strings yields results identical to those of a call to `wcscoll` applied to the original two strings. `wcsxfrm` and `strxfrm` behave identically otherwise.

Returns:
The `strxfrm` function returns the length of the transformed string. If this length is more than `n`, the contents of the array pointed to by `dst` are indeterminate.

See Also: `setlocale`, `strcoll`

Example:
```
#include <stdio.h>
#include <string.h>
#include <locale.h>

char src[] = { "A sample STRING" };  
char dst[20];

void main()
{
    size_t len;

    setlocale( LC_ALL, "C" );
    printf( "%s\n", src );
    len = strxfrm( dst, src, 20 );
    printf( "%s (%lu)\n", dst, len );
}
```

produces the following:

A sample STRING
A sample STRING (15)

Classification: `strxfrm` is ANSI
`wcsxfrm` is ANSI

Systems: `strxfrm` - All, Netware
`wcsxfrm` - All
Synopsis:  
#include <stdlib.h>
void swab( char *src, char *dest, int num );

Description:  The swab function copies num bytes (which should be even) from src to dest swapping every pair of characters. This is useful for preparing binary data to be transferred to another machine that has a different byte ordering.

Returns:  The swab function has no return value.

Example:  
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

char *msg = "hTsim seasegi  swspaep.d";
#define NBYTES 24

void main()
{
    auto char buffer[80];
    printf( "%s\n", msg );
    memset( buffer, '\0', 80 );
    swab( msg, buffer, NBYTES );
    printf( "%s\n", buffer );
}

produces the following:

hTsim seasegi  swspaep.d
This message is swapped.

Classification:  WATCOM

Systems:  All, Netware
system, _wsystem

Synopsis:
#include <stdlib.h>
int system( const char *command );
ext _wsystem( const wchar_t *command );

Description: If the value of command is NULL, then the system function determines whether or not a command processor is present ("COMMAND.COM" in DOS and Windows 95/98 or "CMD.EXE" in OS/2 and Windows NT/2000).

Otherwise, the system function invokes a copy of the command processor, and passes the string command to it for processing. This function uses spawnl to load a copy of the command processor identified by the COMSPEC environment variable.

This means that any command that can be entered to DOS can be executed, including programs, DOS commands and batch files. The exec... and spawn... functions can only cause programs to be executed.

The _wsystem function is identical to system except that it accepts a wide-character string argument.

Returns: If the value of command is NULL, then the system function returns zero if the command processor is not present, a non-zero value if the command processor is present. Note that Microsoft Windows 3.x does not support a command shell and so the system function always returns zero when command is NULL.

Otherwise, the system function returns the result of invoking a copy of the command processor. A non-zero value is returned if the command processor could not be loaded; otherwise, zero is returned. When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also: abort, atexit, _bgetcmd, exec..., exit, _Exit, _exit, getcmd, getenv, main, onexit, putenv, spawn...

Example:
#include <stdlib.h>
#include <stdio.h>

void main()
{
    int rc;

    rc = system( "dir" );
    if( rc != 0 ) {
        printf( "shell could not be run\n" );
    }
}

Classification: system is ANSI, POSIX 1003.2
_wsystem is not ANSI

Systems: system - All, Netware
_wsystem - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

1012 Library Functions and Macros
Synopsis:  
#include <math.h>  
double tan( double x );

Description:  The tan function computes the tangent of \( x \) (measured in radians). A large magnitude argument may yield a result with little or no significance.

Returns:  The tan function returns the tangent value. When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also:  atan, atan2, cos, sin, tanh

Example:  
#include <stdio.h>  
#include <math.h>

t void main()
{
   printf( "%f\n", tan(.5) );
}

produces the following:

0.546302

Classification:  ANSI

Systems:  Math
Synopsis:  
#include <math.h>
double tanh( double x );

Description:  The \texttt{tanh} function computes the hyperbolic tangent of \texttt{x}.

When the \texttt{x} argument is large, partial or total loss of significance may occur. The \texttt{matherr} function will be invoked in this case.

Returns:  The \texttt{tanh} function returns the hyperbolic tangent value. When an error has occurred, \texttt{errno} contains a value indicating the type of error that has been detected.

See Also:  \texttt{cosh}, \texttt{sinh}, \texttt{matherr}

Example:  
#include <stdio.h>
#include <math.h>

void main()
{
    printf( "%f\n", tanh(.5) );
}

produces the following:

0.462117

Classification:  ANSI

Systems:  Math
Synopsis:  
#include <io.h>
off_t tell( int handle );
off_t __tell( int handle );
__int64 _telli64( int handle );

Description:  The tell function reports the current file position at the operating system level. The handle value is the file handle returned by a successful execution of the open function.

The returned value may be used in conjunction with the lseek function to reset the current file position.

The _tell function is identical to tell. Use _tell for ANSI/ISO naming conventions.

The _telli64 function is similar to the tell function but returns a 64-bit file position. This value may be used in conjunction with the _lseeki64 function to reset the current file position.

Returns:  If an error occurs in tell, (-1L) is returned.

If an error occurs in _telli64, (-1I64) is returned.

When an error has occurred, errno contains a value indicating the type of error that has been detected.

Otherwise, the current file position is returned in a system-dependent manner. A value of 0 indicates the start of the file.

See Also:  chsize, close, creat, dup, dup2, eof, exec..., fdopen, filelength, fileno, fstat, _grow_handles, isatty, lseek, open, read, setmode, sopen, stat, write, umask

Example:  
#include <stdio.h>
#include <sys/stat.h>
#include <io.h>
#include <fcntl.h>

char buffer[]
   = { "A text record to be written" };

void main( void )
{
   int handle;
   int size_written;

   /* open a file for output */
   /* replace existing file if it exists */
   handle = open( "file",
                  O_WRONLY | O_CREAT | O_TRUNC | O_TEXT,
                  S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );
if( handle != -1 ) {

    /* print file position */
    printf( "%ld\n", tell( handle ) );

    /* write the text */
    size_written = write( handle, buffer,
                          sizeof( buffer ) );

    /* print file position */
    printf( "%ld\n", tell( handle ) );

    /* close the file */
    close( handle );
}

produces the following:
0
28

Classification: WATCOM

Systems: tell - All, Netware
        _tell - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
        _telli64 - All
Synopsis:  
#include <stdio.h>  
char *_tempnam( char *dir, char *prefix );  
wchar_t *_wtempnam( wchar_t *dir, wchar_t *prefix );

Description:  _tempnam creates a temporary filename for use in another directory. This filename is different from that of any existing file. The prefix argument is the prefix to the filename. _tempnam uses malloc to allocate space for the filename; the program is responsible for freeing this space when it is no longer needed. _tempnam looks for the file with the given name in the following directories, listed in order of precedence.

Directory Used Conditions

Directory specified by TMP The TMP environment variable must be set and the directory specified by TMP must exist.

dir (function argument) The TMP environment variable must not be set or the directory specified by TMP does not exist.

_P_tmpdir (_wP_tmpdir) in STDIO.H The dir argument is NULL or dir is the name of a nonexistent directory. The _wP_tmpdir string is used by _wtempnam.

Current working directory _tempnam uses the current working directory when _P_tmpdir does not exist. _wtempnam uses the current working directory when _wP_tmpdir does not exist.

_tempnam automatically handles multibyte-character string arguments as appropriate, recognizing multibyte-character sequences according to the OEM code page obtained from the operating system. _wtempnam is a wide-character version of _tempnam the arguments and return value of _wtempnam are wide-character strings. _wtempnam and _tempnam behave identically except that _wtempnam does not handle multibyte-character strings.

The function generates unique filenames for up to TMP_MAX calls.

Returns:  The _tempnam function returns a pointer to the name generated, unless it is impossible to create this name or the name is not unique. If the name cannot be created or if a file with that name already exists, _tempnam returns NULL.

See Also: fopen, freopen, mkstemp, _mktemp, tmpfile, tmpnam

Example:
```c
#include <stdio.h>
#include <stdlib.h>

/*
Environment variable TMP=C:\WINDOWS\TEMP
*/
void main()
{
    char *filename;

    FILE *fp;

    filename = _tempnam( "D:\TEMP", "_T" );
    if( filename == NULL )
        printf( "Can’t obtain temp file name\n" );
    else {
        printf( "Temp file name is %s\n", filename );
        fp = fopen( filename, "w+b" );
        /* . */
        /* . */
        /* . */
        fclose( fp );
        remove( filename );
        free( filename );
    }
}
```

produces the following:

Temp file name is C:\WINDOWS\TEMP\_T1

Classification: WATCOM

Systems:  _tempnam - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
         _wtempnam - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:    #include <time.h>
            time_t time( time_t *tloc );

Description: The time function determines the current calendar time and encodes it into the type time_t.

            The time represents the time since January 1, 1970 Coordinated Universal Time (UTC) (formerly known as Greenwich Mean Time (GMT)).

            The time set on the computer with the DOS time command and the DOS date command reflects the local time. The environment variable TZ is used to establish the time zone to which this local time applies. See the section The TZ Environment Variable for a discussion of how to set the time zone.

Returns:    The time function returns the current calendar time. If tloc is not NULL, the current calendar time is also stored in the object pointed to by tloc.

See Also:    asctime Functions, asctime_s, clock, ctime Functions, ctime_s, difftime, gmtime, gmtime_s, localtime, localtime_s, mktime, strftime, tzset

Example:    #include <stdio.h>
            #include <time.h>

            void main()
            {
                time_t time_of_day;

                time_of_day = time( NULL );
                printf( "It is now: %s", ctime( &time_of_day ) );
            }

            produces the following:

            It is now: Fri Dec 25 15:58:42 1987

Classification: ANSI, POSIX 1003.1

Systems:    All, Netware
**Synopsis:**

```
#include <stdio.h>
FILE *tmpfile( void );
```

**Safer C:** The Safer C Library extension provides the `tmpfile_s` function which is a safer alternative to `tmpfile`. This newer `tmpfile_s` function is recommended to be used instead of the traditional "unsafe" `tmpfile` function.

**Description:** The `tmpfile` function creates a temporary binary file that will automatically be removed when it is closed or at program termination. The file is opened for update. For all systems except NetWare, the temporary file is located in the path specified by one of the following environment variables, if one is defined. Otherwise, the current working directory is used. They are listed in the order examined: TMP, TEMP, TMPDIR, and TEMPDIR.

**Returns:** The `tmpfile` function returns a pointer to the stream of the file that it created. If the file cannot be created, the `tmpfile` function returns `NULL`. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

**See Also:** `fopen`, `fopen_s`, `freopen`, `freopen_s`, `mkstemp`, `_mktemp`, `_tempnam`, `tmpfile_s`, `tmpnam`, `tmpnam_s`

**Example:**

```
#include <stdio.h>

static FILE *TempFile;

void main()
{
    TempFile = tmpfile();
    /* . */
    /* . */
    /* . */
    fclose( TempFile );
}
```

**Classification:** ANSI

**Systems:** All, Netware
Synopsis:
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
errno_t tmpfile_s( FILE * restrict * restrict streamptr);

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and tmpfile_s will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

streamptr shall not be a null pointer. If there is a runtime-constraint violation, tmpfile_s does not attempt to create a file.

Description: The tmpfile_s function creates a temporary binary file that is different from any other existing file and that will automatically be removed when it is closed or at program termination. If the program terminates abnormally, whether an open temporary file is removed is implementation-defined. The file is opened for update with "wb+" mode with the meaning that mode has in the fopen_s function (including the mode’s effect on exclusive access and file permissions). If the file was created successfully, then the pointer to FILE pointed to by streamptr will be set to the pointer to the object controlling the opened file. Otherwise, the pointer to FILE pointed to by streamptr will be set to a null pointer. For all systems except NetWare, the temporary file is located in the path specified by one of the following environment variables, if one is defined. Otherwise, the current working directory is used. They are listed in the order examined: TMP, TEMP, TMPDIR, and TEMPDIR.

Returns: The tmpfile_s function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

See Also: fopen, fopen_s, freopen, freopen_s, mkstemp, _mktemp, _tempnam, tmpfile, tmpnam, tmpnam_s

Example: #define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>

void main()
{
   errno_t rc;
   FILE    *TempFile;

   rc = tmpfile_s( &TempFile );
   if( rc == 0 ) {
      /* . */
      /* . */
      /* . */
      fclose( TempFile );
   }
}

Classification: TR 24731

Systems: All, Netware
### Synopsis:

```c
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
errno_t tmpnam_s( char * s, rsize_t maxsize );
#include <wchar.h>
errno_t _wtmpnam_s( wchar_t * s, rsize_t maxsize );
```

### Constraints:

If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `tmpnam_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

- `s` shall not be a null pointer.
- `maxsize` shall be less than or equal to `RSIZE_MAX`.
- `maxsize` shall be greater than the length of the generated file name string.

### Description:

The `tmpnam_s` function generates a string that is a valid file name and that is not the same as the name of an existing file. The function is potentially capable of generating `TMP_MAX_S` different strings, but any or all of them may already be in use by existing files and thus not be suitable return values. The lengths of these strings shall be less than the value of the `L_tmpnam_s` macro. The `tmpnam_s` function generates a different string each time it is called.

The `_wtmpnam_s` function is identical to `tmpnam_s` except that it generates a unique wide-character string for the file name.

### Returns:

If no suitable string can be generated, or if there is a runtime-constraint violation, the `tmpnam_s` function writes a null character to `s[0]` (only if `s` is not null and `maxsize` is greater than zero) and returns a non-zero value. Otherwise, the `tmpnam_s` function writes the string in the array pointed to by `s` and returns zero.

### See Also:

`fopen`, `fopen_s`, `freopen`, `freopen_s`, `mkstemp`, `_mktemp`, `_tempnam`, `tmpfile`, `tmpfile_s`, `tmpnam`.

### Example:

```c
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>

void main()
{
    char   filename[ L_tmpnam_s ];
    FILE   *fp;
    errno_t rc;

    rc = tmpnam( filename, sizeof( filename ) );
    if( rc == 0 ) {
        fp = fopen( filename, "w+b" );
        /* . */
        fclose( fp );
        /* . */
        remove( filename );
    }
}
```

### Classification:

- `tmpnam_s` is TR 24371
- `_wtmpnam_s` is WATCOM

### Systems:

- `tmpnam_s` - All, Netware
**Synopsis:**

```c
#include <stdio.h>
char *tmpnam( char *buffer );
wchar_t * _wtmpnam( wchar_t *buffer );
```

**Safer C:**

The Safer C Library extension provides the `tmpnam_s` function which is a safer alternative to `tmpnam`. This newer `tmpnam_s` function is recommended to be used instead of the traditional "unsafe" `tmpnam` function.

**Description:**

The `tmpnam` function generates a unique string for use as a valid file name. The `_wtmpnam` function is identical to `tmpnam` except that it generates a unique wide-character string for the file name. An internal static buffer is used to construct the filename. Subsequent calls to `tmpnam` reuse the internal buffer.

The function generates unique filenames for up to `TMP_MAX` calls.

**Returns:**

If the argument `buffer` is a NULL pointer, `tmpnam` returns a pointer to an internal buffer containing the temporary file name. If the argument `buffer` is not a NULL pointer, `tmpnam` copies the temporary file name from the internal buffer to the specified buffer and returns a pointer to the specified buffer. It is assumed that the specified buffer is an array of at least `L_tmpnam` characters.

If the argument `buffer` is a NULL pointer, you may wish to duplicate the resulting string since subsequent calls to `tmpnam` reuse the internal buffer.

```c
char *name1, *name2;
name1 = strdup( tmpnam( NULL ) );
name2 = strdup( tmpnam( NULL ) );
```

**See Also:**

`fopen`, `fopen_s`, `freopen`, `freopen_s`, `mkstemp`, `_mktemp`, `_tempnam`, `tmpfile`, `tmpfile_s`, `tmpnam_s`

**Example:**

```c
#include <stdio.h>

void main()
{
    char filename[ L_tmpnam ];
    FILE *fp;

    tmpnam( filename );
    fp = fopen( filename, "w+b" );
    /* . */
    /* . */
    /* . */
    fclose( fp );
    remove( filename );
}
```

**Classification:**

`tmpnam` is ANSI

`_wtmpnam` is not ANSI

**Systems:**

`tmpnam` - All, Netware

`_wtmpnam` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

```c
#include <ctype.h>
int tolower( int c );
int _tolower( int c );
#include <wctype.h>
wint_t towlower( wint_t c );
```

Description:

The `tolower` function converts `c` to a lowercase letter if `c` represents an uppercase letter.

The `_tolower` function is a version of `tolower` to be used only when `c` is known to be uppercase.

The `towlower` function is similar to `tolower` except that it accepts a wide-character argument.

Returns:

The `tolower` function returns the corresponding lowercase letter when the argument is an uppercase letter; otherwise, the original character is returned. The `towlower` function returns the corresponding wide-character lowercase letter when the argument is a wide-character uppercase letter; otherwise, the original wide character is returned.

The result of `_tolower` is undefined if `c` is not an uppercase letter.

See Also:

`isalnum`, `isalpha`, `isblank`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `toupper`, `towctrans`, `strlwr`, `strupr`, `toupper`

Example:

```c
#include <stdio.h>
#include <ctype.h>

char chars[] = {
    'A',
    '5',
    '$',
    'Z'
};

#define SIZE sizeof( chars ) / sizeof( char )

void main()
{
    int i;
    for( i = 0; i < SIZE; i++ ) {
        printf( "%c ", tolower( chars[ i ] ) );
        printf( "\n" );
    }
}

produces the following:

a 5 $ z
```

Classification:

- `tolower` is ANSI
- `_tolower` is not ANSI
- `towlower` is ANSI

Systems:

- `tolower` - All, Netware
tolower, _tolower, towlower

_totolower - All, Netware
towlower - All, Netware
Synopsis:

```c
#include <ctype.h>
int toupper( int c );
int _toupper( int c );
#include <wctype.h>
wint_t towupper( wint_t c );
```

Description:

The `toupper` function converts `c` to an uppercase letter if `c` represents a lowercase letter.

The `_toupper` function is a version of `toupper` to be used only when `c` is known to be lowercase.

The `towupper` function is similar to `toupper` except that it accepts a wide-character argument.

Returns:

The `toupper` function returns the corresponding uppercase letter when the argument is a lowercase letter; otherwise, the original character is returned. The `towupper` function returns the corresponding wide-character uppercase letter when the argument is a wide-character lowercase letter; otherwise, the original wide character is returned.

The result of `_toupper` is undefined if `c` is not a lowercase letter.

See Also:
`isalnum`, `isalpha`, `isblank`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `ispunct`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `towctrans`, `strlwr`, `strupr`, `tolower`

Example:

```c
#include <stdio.h>
#include <ctype.h>

char chars[] = {
    'a',
    '5',
    '$',
    'z'
};

#define SIZE sizeof( chars ) / sizeof( char )

void main()
{
    int   i;
    for( i = 0; i < SIZE; i++ ) {
        printf( "%c ", toupper( chars[ i ] ) );
    }
    printf( "\n" );
}
```

produces the following:

```
A 5 $ Z
```

Classification:

- `toupper` is ANSI
- `_toupper` is not ANSI
- `towupper` is ANSI

Systems:

- `toupper` - All, Netware
toupper, _toupper, towupper

_toupper - All, Netware
towupper - All, Netware
Synopsis:    #include <wctype.h>
    wint_t towctrans( wint_t wc, wctrans_t desc );

Description: The towctrans function maps the wide character wc using the mapping described by desc. Valid values of desc are defined by the use of the wctrans function.

The two expressions listed below behave the same as a call to the wide character case mapping function shown.

Expression Equivalent

\[
towctrans(wc, \text{wctrans("tolower")}) \rightarrow \text{tolower}(wc)
\]

\[
towctrans(wc, \text{wctrans("toupper")}) \rightarrow \text{toupper}(wc)
\]

Returns: The towctrans function returns the mapped value of wc using the mapping described by desc.

See Also: isalnum, isalpha, isblank, iscntrl, isdigit, isgraph, isleadbyte, islower, isprint, ispunct, isspace, isupper, iswctype, isxdigit, tolower, toupper

Example: #include <stdio.h>
#include <wctype.h>

    char *translations[2] = {
        "tolower",
        "toupper"
    };

    void main( void )
    {
        int i;
        wint_t wc = 'A';
        wint_t twc;

        for( i = 0; i < 2; i++ ) {
            twc = towctrans( wc, wctrans( translations[i] ) );
            printf( "%s(%lc): %lc\n", translations[i], wc, twc );
        }
    }

produces the following:

tolower(A): a
toupper(A): A

Classification: ANSI

Systems: All, Netware
Synopsis:    #include <time.h>
   void tzset( void );

Description: The tzset function sets the global variables daylight, timezone and tzname according to the value of the TZ environment variable. The section The TZ Environment Variable describes how to set this variable.

Under Win32, tzset also uses operating system supplied time zone information. The TZ environment variable can be used to override this information.

The global variables have the following values after tzset is executed:

   daylight     Zero indicates that daylight saving time is not supported in the locale; a non-zero value indicates that daylight saving time is supported in the locale. This variable is cleared/set after a call to the tzset function depending on whether a daylight saving time abbreviation is specified in the TZ environment variable.

   timezone    Contains the number of seconds that the local time zone is earlier than Coordinated Universal Time (UTC) (formerly known as Greenwich Mean Time (GMT)).

   tzname       Two-element array pointing to strings giving the abbreviations for the name of the time zone when standard and daylight saving time are in effect.

The time set on the computer with the DOS time command and the DOS date command reflects the local time. The environment variable TZ is used to establish the time zone to which this local time applies. See the section The TZ Environment Variable for a discussion of how to set the time zone.

Returns:    The tzset function does not return a value.

See Also:   ctime Functions, localtime, mktime, strftime

Example:    #include <stdio.h>
   #include <env.h>
   #include <time.h>

   void print_zone()
   {
      char *tz;

      printf( "TZ: %s\n", (tz = getenv( "TZ" ))
      ? tz : "default EST5EDT" );
      printf( "  daylight: %d\n", daylight );
      printf( "  timezone: %ld\n", timezone );
      printf( "  time zone names: %s %s\n",
                     tzname[0], tzname[1] );
   }
void main()
{
    print_zone();
    setenv( "TZ", "PST8PDT", 1 );
    tzset();
    print_zone();
}

produces the following:

TZ: default EST5EDT
daylight: 1
timezone: 18000
time zone names: EST EDT
TZ: PST8PDT
daylight: 1
timezone: 28800
time zone names: PST PDT

Classification: POSIX 1003.1

Systems: All, Netware
ulltoa, _ulltoa, _ulltow

Synopsis:  
#include <stdlib.h>  
char *ulltoa( unsigned long long int value,  
             char *buffer,  
             int radix );  
char *_ulltoa( unsigned long long int value,  
              char *buffer,  
              int radix );  
wchar_t *_ulltow( unsigned long long int value,  
                 wchar_t *buffer,  
                 int radix );

Description: The ulltoa function converts the unsigned binary integer value into the equivalent string in base radix notation storing the result in the character array pointed to by buffer. A null character is appended to the result. The size of buffer must be at least 65 bytes when converting values in base 2. The value of radix must satisfy the condition:

\[ 2 \leq \text{radix} \leq 36 \]

The _ulltoa function is identical to ulltoa. Use _ulltoa for ANSI/ISO naming conventions.

The _ulltow function is identical to ulltoa except that it produces a wide-character string (which is twice as long).

Returns: The ulltoa function returns the pointer to the result.

See Also: atoi, atol, atol1, itoa, ltoa, lltoa, sscanf, strtol, strtoll, strto1ul, strto1max, strto1umax, ul1oa, ul1oa

Example:  
#include <stdio.h>  
#include <stdlib.h>

void print_value( unsigned long long int value )  
{  
    int base;  
    char buffer[65];  
    
    for( base = 2; base <= 16; base = base + 2 )  
        printf( "%2d %s\n", base,  
                 ul1oa( value, buffer, base ) );  
}  

void main()  
{  
    print_value( (unsigned long long) 1234098765LL );  
}  

produces the following:
Classification: WATCOM
_ulltoa conforms to ANSI/ISO naming conventions

Systems:
ulltoa - All, Netware
_ulltoa - All, Netware
_ulltow - All
ultoa, _ultoa, _ultow

Synopsis:

#include <stdlib.h>
char *ultoa( unsigned long int value,
        char *buffer,
        int radix );
char * _ultoa( unsigned long int value,
        char *buffer,
        int radix );
wchar_t * _ultow( unsigned long int value,
        wchar_t *buffer,
        int radix );

Description: The ultoa function converts the unsigned binary integer value into the equivalent string in base radix notation storing the result in the character array pointed to by buffer. A null character is appended to the result. The size of buffer must be at least 33 bytes when converting values in base 2. The value of radix must satisfy the condition:

    2 <= radix <= 36

The _ultoa function is identical to ultoa. Use _ultoa for ANSI/ISO naming conventions.

The _ultow function is identical to ultoa except that it produces a wide-character string (which is twice as long).

Returns: The ultoa function returns the pointer to the result.

See Also: atoi, atol, atol1, itoa, ltoa, lltoa, sscanf, strtol, strtoll, strtoul, strtoull, strtoimax, strtoumax, ulltoa, uitoa

Example: #include <stdio.h>
#include <stdlib.h>

void print_value( unsigned long int value )
{
    int base;
    char buffer[33];

    for( base = 2; base <= 16; base = base + 2 )
        printf( "%2d %s\n", base,
                ultoa( value, buffer, base ) );
}

void main()
{
    print_value( (unsigned) 12765L );
}

produces the following:
_ultoa conforms to ANSI/ISO naming conventions

Systems:  
ultoa - All, Netware  
_ultoa - All, Netware  
_ultow - All
umask, _umask

Synopsis:  
#include <sys/types.h>  
#include <sys/stat.h>  
#include <fcntl.h>  
#include <io.h>  
int umask( int cmask );  
int _umask( int cmask );

Description:  The umask function sets the process’s file mode creation mask to cmask.  The process’s file mode creation mask is used during creat, open or sopen to turn off permission bits in the permission argument supplied.  In other words, if a bit in the mask is on, then the corresponding bit in the file’s requested permission value is disallowed.

The _umask function is identical to umask.  Use _umask for ANSI/ISO naming conventions.

The argument cmask is a constant expression involving the constants described below.  The access permissions for the file or directory are specified as a combination of bits (defined in the <sys/stat.h> header file).

The following bits define permissions for the owner.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_IRWXU</td>
<td>Read, write, execute/search</td>
</tr>
<tr>
<td>S_IRUSR</td>
<td>Read permission</td>
</tr>
<tr>
<td>S_IWUSR</td>
<td>Write permission</td>
</tr>
<tr>
<td>S_IXUSR</td>
<td>Execute/search permission</td>
</tr>
</tbody>
</table>

The following bits define permissions for the group.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_IRWXG</td>
<td>Read, write, execute/search</td>
</tr>
<tr>
<td>S_IRGRP</td>
<td>Read permission</td>
</tr>
<tr>
<td>S_IWGRP</td>
<td>Write permission</td>
</tr>
<tr>
<td>S_IXGRP</td>
<td>Execute/search permission</td>
</tr>
</tbody>
</table>

The following bits define permissions for others.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_IRWXO</td>
<td>Read, write, execute/search</td>
</tr>
<tr>
<td>S_IROTH</td>
<td>Read permission</td>
</tr>
<tr>
<td>S_IWOTH</td>
<td>Write permission</td>
</tr>
<tr>
<td>S_IXOTH</td>
<td>Execute/search permission</td>
</tr>
</tbody>
</table>

The following bits define miscellaneous permissions used by other implementations.

<table>
<thead>
<tr>
<th>Permission</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_IREAD</td>
<td>is equivalent to S_IRUSR (read permission)</td>
</tr>
<tr>
<td>S_IWRITE</td>
<td>is equivalent to S_IWUSR (write permission)</td>
</tr>
</tbody>
</table>
S_IEXEC is equivalent to S_IXUSR (execute/search permission)

For example, if S_IRUSR is specified, then reading is not allowed (i.e., the file is write only). If S_IWUSR is specified, then writing is not allowed (i.e., the file is read only).

Returns: The umask function returns the previous value of cmask.

See Also: chmod, creat, mkdir, open, sopen

Example:
```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <io.h>

void main( void )
{
    int old_mask;

    /* set mask to create read-only files */
    old_mask = umask( S_IWUSR | S_IWGRP | S_IWOTH |
                      S_IXUSR | S_IXGRP | S_IXOTH );
}
```

Classification: umask is POSIX 1003.1
_umask is not POSIX
_umask conforms to ANSI/ISO naming conventions

Systems: umask - All, Netware
_umask - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
ungetc, ungetwc

Synopsis:

```
#include <stdio.h>
int ungetc( int c, FILE *fp );
#include <stdio.h>
#include <wchar.h>
wint_t ungetwc( wint_t c, FILE *fp );
```

Description:  The `ungetc` function pushes the character specified by `c` back onto the input stream pointed to by `fp`. This character will be read by the next read on the stream. The pushed-back character will be discarded if a call is made to the `fflush` function or to a file positioning function (`fseek`, `fsetpos` or `rewind`) before the next read operation is performed.

Only one character (the most recent one) of pushback is remembered.

The `ungetc` function clears the end-of-file indicator, unless the value of `c` is `EOF`.

The `ungetwc` function is identical to `ungetc` except that it pushes the wide character specified by `c` back onto the input stream pointed to by `fp`.

The `ungetwc` function clears the end-of-file indicator, unless the value of `c` is `WEOF`.

Returns:  The `ungetc` function returns the character pushed back.

See Also:  `fgetc`, `fgetchar`, `fgets`, `fopen`, `getc`, `getchar`, `gets`

Example:

```
#include <stdio.h>
#include <ctype.h>

void main()
{
    FILE *fp;
    int c;
    long value;

    fp = fopen( "file", "r" );
    value = 0;
    c = fgetc( fp );
    while( isdigit(c) ) {
        value = value*10 + c - '0';
        c = fgetc( fp );
    }
    ungetc( c, fp ); /* put last character back */
    printf( "Value=%ld\n", value );
    fclose( fp );
}
```

Classification:  `ungetc` is ANSI

`ungetwc` is ANSI

Systems:  `ungetc` - All, Netware

`ungetwc` - All
Synopsis:  
#include <conio.h>
int ungetch( int c );

Description:  The ungetch function pushes the character specified by c back onto the input stream for the console. This character will be returned by the next read from the console (with getch or getche functions) and will be detected by the function kbhit. Only the last character returned in this way is remembered.

The ungetch function clears the end-of-file indicator, unless the value of c is EOF.

Returns:  The ungetch function returns the character pushed back.

See Also:  getch, getche, kbhit, putch

Example:  
#include <stdio.h>
#include <ctype.h>
#include <conio.h>

void main()
{
   int c;
   long value;

   value = 0;
   c = getche();
   while( isdigit( c ) ) {
      value = value*10 + c - '0';
      c = getche();
   }
   ungetch( c );
   printf( "Value=%ld\n", value );
}

Classification: WATCOM

Systems: All, Netware
Synopsis:  
#include <io.h>
int unlink( const char *path );
int _unlink( const char *path );
int _wunlink( const wchar_t *path );

Description:  
The `unlink` function deletes the file whose name is the string pointed to by `path`. This function is equivalent to the `remove` function.

The `_unlink` function is identical to `unlink`. Use `_unlink` for ANSI/ISO naming conventions.

The `_wunlink` function is identical to `unlink` except that it accepts a wide-character string argument.

Returns:  
The `unlink` function returns zero if the operation succeeds, non-zero if it fails.

See Also:  
`chdir`, `chmod`, `close`, `getcwd`, `mkdir`, `open`, `remove`, `rename`, `rmdir`, `stat`

Example:  
```c
#include <io.h>

void main( void )
{
    unlink( "vm.tmp" );
}
```

Classification:  
`unlink` is POSIX 1003.1
`_unlink` is not POSIX
`_wunlink` is not POSIX
`_unlink` conforms to ANSI/ISO naming conventions

Systems:  
`unlink` - All, Netware
`_unlink` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
`_wunlink` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
```c
#include <io.h>
int unlock( int handle,
            unsigned long offset,
            unsigned long nbytes );
```

Description: The `unlock` function unlocks `nbytes` amount of previously locked data in the file designated by `handle` starting at byte `offset` in the file. This allows other processes to lock this region of the file.

Multiple regions of a file can be locked, but no overlapping regions are allowed. You cannot unlock multiple regions in the same call, even if the regions are contiguous. All locked regions of a file should be unlocked before closing a file or exiting the program.

With DOS, locking is supported by version 3.0 or later. Note that `SHARE.COM` or `SHARE.EXE` must be installed.

Returns: The `unlock` function returns zero if successful, and -1 when an error occurs. When an error has occurred, `errno` contains a value indicating the type of error that has been detected.

See Also: `lock`, `locking`, `open`, `sopen`

Example:  
```c
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

void main()
{
    int handle;
    char buffer[20];

    handle = open( "file", O_RDWR | O_TEXT );
    if( handle != -1 ) {
        if( lock( handle, 0L, 20L ) ) {
            printf( "Lock failed\n" );
        } else {
            read( handle, buffer, 20 );
            /* update the buffer here */
            lseek( handle, 0L, SEEK_SET );
            write( handle, buffer, 20 );
            unlock( handle, 0L, 20L );
        }
    }
    close( handle );
}
```

Classification: WATCOM

Systems: All, Netware
**unregisterfonts**

### Synopsis:
```c
#include <graph.h>
void _FAR _unregisterfonts( void );
```

### Description:
The `_unregisterfonts` function frees the memory previously allocated by the `_registerfonts` function. The currently selected font is also unloaded.

Attempting to use the `_setfont` function after calling `_unregisterfonts` will result in an error.

### Returns:
The `_unregisterfonts` function does not return a value.

### See Also:
- `_registerfonts`, `_setfont`, `_getfontinfo`, `_outgtext`, `_getgtextextent`, `_setgtextvector`, `_getgtextvector`

### Example:
```c
#include <conio.h>
#include <stdio.h>
#include <graph.h>

main()
{
    int i, n;
    char buf[ 10 ];

    _setvideomode( _VRES16COLOR );
    n = _registerfonts( "*.fon" );
    for( i = 0; i < n; ++i )
    {
        sprintf( buf, "n%d", i );
        _setfont( buf );
        _moveto( 100, 100 );
        _outgtext( "WATCOM Graphics" );
        getch();
        _clearscreen( _GCLEARSCREEN );
    }
    _unregisterfonts();
    _setvideomode( _DEFAULTMODE );
}
```

### Classification:
PC Graphics

### Systems:
DOS, QNX
Synopsis:  
#include <sys/utime.h>
int utime( const char *path,
            const struct utimbuf *times );
int _utime( const char *path,
            const struct utimbuf *times );
int _wutime( const wchar_t *path,
            const struct utimbuf *times );

struct utimbuf {
    time_t   actime;  /* access time */
    time_t   modtime; /* modification time */
};

Description:  The utime function records the access and modification times for the file identified by path.

The _utime function is identical to utime. Use _utime for ANSI naming conventions.

If the times argument is NULL, the access and modification times of the file or directory are set to the current time. Write access to this file must be permitted for the time to be recorded.

If the times argument is not NULL, it is interpreted as a pointer to a utimbuf structure and the access and modification times of the file or directory are set to the values contained in the designated structure. The access and modification times are taken from the actime and modtime fields in this structure.

The _wutime function is identical to utime except that path points to a wide-character string.

Returns:  The utime function returns zero when the time was successfully recorded. A value of -1 indicates an error occurred.

Errors:  When an error has occurred, errno contains a value indicating the type of error that has been detected.

Constant                Meaning
EACCES                  Search permission is denied for a component of path or the times argument is NULL and the effective user ID of the process does not match the owner of the file and write access is denied.
EINVAL                 The date is before 1980 (DOS only).
EMFILE                 There are too many open files.
ENOENT                 The specified path does not exist or path is an empty string.

Example:  
#include <stdio.h>
#include <sys/utime.h>

void main( int argc, char *argv[] )
{
    if( (utime( argv[1], NULL ) != 0) && (argc > 1) ) {
        printf( "Unable to set time for %s\n", argv[1] );
    }
}

Classification:  utime is POSIX 1003.1
_utime is not POSIX
_wutime is not POSIX

**Systems:**
- utime - All, Netware
- _utime - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
- _wutime - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <stdlib.h>
char *utoa( unsigned int value,
    char *buffer,
    int radix );

char *___utoa( unsigned int value,
    char *buffer,
    int radix );

wchar_t *___utow( unsigned int value,
    wchar_t *buffer,
    int radix );

Description: The utoa function converts the unsigned binary integer value into the equivalent string in base radix notation storing the result in the character array pointed to by buffer. A null character is appended to the result. The size of buffer must be at least (8 * sizeof(int) + 1) bytes when converting values in base 2. That makes the size 17 bytes on 16-bit machines, and 33 bytes on 32-bit machines. The value of radix must satisfy the condition:

    2 <= radix <= 36

The ___utoa function is identical to utoa. Use ___utoa for ANSI/ISO naming conventions.

The ___utow function is identical to utoa except that it produces a wide-character string (which is twice as long).

Returns: The utoa function returns the pointer to the result.

See Also: atoi, atol, atoll, itoa, ltoa, lltoa, sscsnf, strtol, strtol, strtoul, strtoull, strtoimax, strtoomax, ultoa, ulltoa

Example:  
#include <stdio.h>
#include <stdlib.h>

void main() {
    int base;
    char buffer[18];

    for( base = 2; base <= 16; base = base + 2 )
        printf( "%2d %s\n", base,
            utoa( (unsigned) 12765, buffer, base ) );
}

produces the following:

    2 11000111011101
    4 3013131
    6 135033
    8 30735
   10 12765
   12 7479
   14 491b
   16 31dd

Classification: WATCOM

___utoa conforms to ANSI/ISO naming conventions
utoa, _utoa, _utow

Systems: utoa - All, Netware
         _utoa - All, Netware
         _utow - All
Synopsis:  
#include <stdarg.h>
type va_arg( va_list param, type );

Description:  va_arg is a macro that can be used to obtain the next argument in a list of variable arguments. It must be used with the associated macros va_start and va_end. A sequence such as

```c
void example( char *dst, ... )
{
    va_list curr_arg;
    int next_arg;
    va_start( curr_arg, dst );
    next_arg = va_arg( curr_arg, int );
    ...
}
```

causes next_arg to be assigned the value of the next variable argument. The argument type (which is int in the example) is the type of the argument originally passed to the function.

The macro va_start must be executed first in order to properly initialize the variable curr_arg and the macro va_end should be executed after all arguments have been obtained.

The data item curr_arg is of type va_list which contains the information to permit successive acquisitions of the arguments.

Returns:  The macro returns the value of the next variable argument, according to type passed as the second parameter.

See Also:  va_end, va_start, vfprintf, vprintf, vsprintf

Example:  
#include <stdio.h>
#include <stdarg.h>

```c
static void test_fn(
    const char *msg,   /* message to be printed */
    const char *types, /* parameter types (i,s) */
    ... )              /* variable arguments */
{
    va_list argument;
    int       arg_int;
    char      *arg_string;
    const char *types_ptr;
```
va_arg

```c
    types_ptr = types;
    printf( "\n%s -- %s\n", msg, types );
    va_start( argument, types );
    while( *types_ptr != '\0' ) {
        if (*types_ptr == 'i') {
            arg_int = va_arg( argument, int );
            printf( "integer: %d\n", arg_int );
        } else if (*types_ptr == 's') {
            arg_string = va_arg( argument, char * );
            printf( "string:  %s\n", arg_string );
        }
        ++types_ptr;
    }
    va_end( argument );
}

void main( void )
{
    printf( "VA...TEST\n" );
    test_fn( "PARAMETERS: 1, \"abc\", 546", 
             "isi", 1, \"abc\", 546 );
    test_fn( "PARAMETERS: \"def\", 789", 
             "si", \"def\", 789 );
}

produces the following:

VA...TEST

PARAMETERS: 1, "abc", 546 -- isi
integer: 1
string: abc
integer: 546

PARAMETERS: "def", 789 -- si
string: def
integer: 789

Classification: ISO C90

Systems: MACRO

1048 Library Functions and Macros
Synopsis:  
#include <stdarg.h>  
void va_end( va_list param );

Description: va_end is a macro used to complete the acquisition of arguments from a list of variable arguments. It  
must be used with the associated macros va_start and va_arg. See the description for va_arg  
for complete documentation on these macros.

Returns:  
The macro does not return a value.

See Also:  
va_arg, va_start, vfprintf, vprintf, vsprintf

Example:  
#include <stdio.h>  
#include <stdarg.h>  
#include <time.h>

#define ESCAPE 27

tprintf( int row, int col, char *fmt, ... )
{
    auto va_list ap;
    char *p1, *p2;

    va_start( ap, fmt );
    p1 = va_arg( ap, char * );
    p2 = va_arg( ap, char * );
    printf( "%c[2.2d;2.2dH", ESCAPE, row, col );
    printf( fmt, p1, p2 );
    va_end( ap );
}

main()
{
    struct tm  time_of_day;
    time_t     ltime;
    auto char  buf[26];

    time( &ltime );
    _localtime( &ltime, &time_of_day );
    tprintf( 12, 1, "Date and time is: %s\n", 
        _asctime( &time_of_day, buf ) );
}

Classification: ANSI

Systems: MACRO
Synopsis: #include <stdarg.h>
void va_start( va_list param, previous );

Description: va_start is a macro used to start the acquisition of arguments from a list of variable arguments. The param argument is used by the va_arg macro to locate the current acquired argument. The previous argument is the argument that immediately precedes the "..." notation in the original function definition. It must be used with the associated macros va_arg and va_end. See the description of va_arg for complete documentation on these macros.

Returns: The macro does not return a value.

See Also: va_arg, va_end, vfprintf, vprintf, vsprintf

Example: #include <stdio.h>
#include <stdarg.h>
#include <time.h>

#define ESCAPE 27

void tprintf( int row, int col, char *fmt, ... )
{
    auto va_list ap;
    char *p1, *p2;

    va_start( ap, fmt );
    p1 = va_arg( ap, char * );
    p2 = va_arg( ap, char * );
    printf( "%c[%2.2d;%2.2dH", ESCAPE, row, col );
    printf( fmt, p1, p2 );
    va_end( ap );
}

void main()
{
    struct tm time_of_day;
    time_t ltime;
    auto char buf[26];

    time( &ltime );
    _localtime( &ltime, &time_of_day );
    tprintf( 12, 1, "Date and time is: %s\n", _asctime( &time_of_day, buf ) );
}

Classification: ANSI

Systems: MACRO
Synopsis:  
#include <stdio.h>  
#include <stdarg.h>  
int _vbprintf( char *buf, size_t bufsize,  
const char *format, va_list arg );  
int _vbwprintf( wchar_t *buf, size_t bufsize,  
const wchar_t *format, va_list arg );

Description:  The _vbprintf function formats data under control of the format control string and writes the result to buf. The argument bufsize specifies the size of the character array buf into which the generated output is placed. The format string is described under the description of the printf function. The _vbprintf function is equivalent to the _bprintf function, with the variable argument list replaced with arg, which has been initialized by the va_start macro.

The _vbwprintf function is identical to _vbprintf except that it accepts a wide-character string argument for format and produces wide-character output.

Returns:  The _vbprintf function returns the number of characters written, or a negative value if an output error occurred.

See Also:  _bprintf, cprintf, fprintf, printf, sprintf, va_arg, va_end, va_start, vcprintf, vfprintf, vprintf, vsprintf

Example:  The following shows the use of _vbprintf in a general error message routine.

#include <stdio.h>  
#include <stdarg.h>  
#include <string.h>

char msgbuf[80];

char *fmtmsg( char *format, ... )  
{  
va_list arglist;

va_start( arglist, format );  
strcpy( msgbuf, "Error: ");  
_vbprintf( &msgbuf[7], 73, format, arglist );  
va_end( arglist );  
return( msgbuf );
}

void main()  
{  
char *msg;

msg = fmtmsg( "%s %d %s", "Failed", 100, "times" );  
printf( "%s\n", msg );
}

Classification: WATCOM

Systems:  _vbprintf - All, Netware  
          _vbwprintf - All
vcprintf

Synopsis:  
#include <conio.h>  
#include <stdarg.h>  
int vcprintf( const char *format, va_list arg );

Description:  The vcprintf function writes output directly to the console under control of the argument format. The putch function is used to output characters to the console. The format string is described under the description of the printf function. The vcprintf function is equivalent to the cprintf function, with the variable argument list replaced with arg, which has been initialized by the va_start macro.

Returns:  The vcprintf function returns the number of characters written, or a negative value if an output error occurred. When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also:  _bprintf,cprintf,fprintf,printf,sprintf,va_arg,va_end,va_start, _vbprintf,vfprintf,vprintf,vsprintf

Example:  
#include <conio.h>  
#include <stdarg.h>  
#include <time.h>  
#define ESCAPE 27  

void tprintf( int row, int col, char *format, ... )  
{  
    auto va_list arglist;  
    cprintf( "%c[%2.2d;%2.2dH", ESCAPE, row, col );  
    va_start( arglist, format );  
    vcprintf( format, arglist );  
    va_end( arglist );  
}

void main()  
{  
    struct tm  time_of_day;  
    time_t     ltime;  
    auto char  buf[26];  

    time( &ltime );  
    _localtime( &ltime, &time_of_day );  
    tprintf( 12, 1, "Date and time is: %s\n",  
             _asctime( &time_of_day, buf ) );  
}

Classification:  WATCOM

Systems:  All, Netware

1052 Library Functions and Macros
Synopsis:
#include <conio.h>
#include <stdarg.h>
int vcscanf( const char *format, va_list args )

Description:
The vcscanf function scans input from the console under control of the argument format. The
vcscanf function uses the function getche to read characters from the console. The format string is
described under the description of the scanf function.

The vcscanf function is equivalent to the cscanf function, with a variable argument list replaced
with arg, which has been initialized using the va_start macro.

Returns:
The vcscanf function returns EOF when the scanning is terminated by reaching the end of the input
stream. Otherwise, the number of input arguments for which values were successfully scanned and
stored is returned. When a file input error occurs, the errno global variable may be set.

See Also: cscanf, fscanf, scanf, sscanf, va_arg, va_end, va_start, vfscanf, vscanf,
vscanf

Example:
#include <conio.h>
#include <stdarg.h>

void cfind( char *format, ... )
{
    va_list arglist;

    va_start( arglist, format );
    vcscanf( format, arglist );
    va_end( arglist );
}

void main()
{
    int day, year;
    char weekday[10], month[10];

    cfind( "%s %s %d %d",
          weekday, month, &day, &year );
    cprintf( "\n%s, %s %d, %d\n",
             weekday, month, day, year );
}

Classification: WATCOM

Systems: All, Netware
*vfprintf*, *vfwprintf*

**Synopsis:**

```c
#include <stdarg.h>
#include <stdio.h>
int vfprintf( FILE *fp,
    const char *format,
    va_list arg );
```

```c
#include <stdio.h>
#include <wchar.h>
#include <stdlib.h>
int vfwprintf( FILE *fp,
    const wchar_t *format,
    va_list arg );
```

**Safer C:**

The Safer C Library extension provides the *vfprintf_s* function which is a safer alternative to *vfprintf*. This newer *vfprintf_s* function is recommended to be used instead of the traditional "unsafe" *vfprintf* function.

**Description:**

The *vfprintf* function writes output to the file pointed to by *fp* under control of the argument *format*. The *format* string is described under the description of the *printf* function. The *vfprintf* function is equivalent to the *fprintf* function, with the variable argument list replaced with *arg*, which has been initialized by the *va_start* macro.

The *vfwprintf* function is identical to *vfprintf* except that it accepts a wide-character string argument for *format*.

**Returns:**

The *vfprintf* function returns the number of characters written, or a negative value if an output error occurred. The *vfwprintf* function returns the number of wide characters written, or a negative value if an output error occurred. When an error has occurred, *errno* contains a value indicating the type of error that has been detected.

**See Also:**

*bprintf*, *cprintf*, *fprintf*, *printf*, *sprintf*, *va_arg*, *va_end*, *va_start*, *vbprintf*, *vcprintf*, *vprintf*, *vsprintf*

**Example:**

```c
#include <stdio.h>
#include <stdarg.h>

FILE *LogFile;

/* a general error routine */
void errmsg( char *format, ... )
{
    va_list arglist;

    fprintf( stderr, "Error: " );
    va_start( arglist, format );
    vfprintf( stderr, format, arglist );
    va_end( arglist );
    if( LogFile != NULL ) {
        fprintf( LogFile, "Error: " );
        va_start( arglist, format );
        vfprintf( LogFile, format, arglist );
        va_end( arglist );
    }
}
```

1054 Library Functions and Macros
void main( void )
{
    LogFile = fopen( "error.log", "w" );
    errmsg( "%s %d %s", "Failed", 100, "times" );
}

Classification: vfprintf is ANSI
                vwprintf is ANSI

Systems:       vfprintf - All, Netware
                vwprintf - All
**vfprintf_s, vfwprintf_s**

**Synopsis:**
```c
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdarg.h>
#include <stdio.h>
int vfprintf_s( FILE * restrict stream,
                const char * restrict format, va_list arg );

#include <stdarg.h>
#include <wchar.h>
int vfwprintf_s( FILE * restrict stream,
                const wchar_t * restrict format, va_list prg );
```

**Constraints:** If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and vfprintf_s will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither `stream` nor `format` shall be a null pointer. The `%n` specifier (modified or not by flags, field width, or precision) shall not appear in the string pointed to by `format`. Any argument to `vfprintf_s` corresponding to a `%s` specifier shall not be a null pointer.

If there is a runtime-constraint violation, the `vfprintf_s` function does not attempt to produce further output, and it is unspecified to what extent `vfprintf_s` produced output before discovering the runtime-constraint violation.

**Description:** The `vfprintf_s` function is equivalent to the `vprintf` function except for the explicit runtime-constraints listed above.

The `vfwprintf_s` function is identical to `vfprintf_s` except that it accepts a wide-character string argument for `format`.

**Returns:** The `vfprintf_s` function returns the number of characters written, or a negative value if an output error or runtime-constraint violation occurred.

The `vfwprintf_s` function returns the number of wide characters written, or a negative value if an output error or runtime-constraint violation occurred.

**See Also:** `_bprintf`, `cprintf`, `fprintf`, `printf`, `sprintf`, `_vbprintf`, `vcprintf`, `vfprintf`, `vprintf`, `vsprintf`

**Example:**
```c
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <stdarg.h>

FILE *LogFile;
/* a general error routine */

void errmsg( char *format, ... )
{
    va_list arglist;
```
vfprintf_s( stderr, "Error: " );
va_start( arglist, format );
vfprintf_s( stderr, format, arglist );
va_end( arglist );

if( LogFile != NULL ) {
    fprintf_s( LogFile, "Error: " );
    va_start( arglist, format );
vfprintf_s( LogFile, format, arglist );
    va_end( arglist );
}

void main( void )
{
    errmsg( "%s %d %s", "Failed", 100, "times" );
}

produces the following:

Error: Failed 100 times

Classification: vfprintf_s is TR 24731
vfwprintf_s is TR 24731

Systems: vfprintf_s - All, Netware
vfwprintf_s - All
**vfscanf, vfwscanf**

**Synopsis:**
```
#include <stdio.h>
#include <stdarg.h>
int vfscanf( FILE *fp, const char *format, va_list arg );
int vfwscanf( FILE *fp, const wchar_t *format, va_list arg );
```

**Safer C:** The Safer C Library extension provides the `vfscanf_s` function which is a safer alternative to `vfscanf`. This newer `vfscanf_s` function is recommended to be used instead of the traditional "unsafe" `vfscanf` function.

**Description:** The `vfscanf` function scans input from the file designated by `fp` under control of the argument `format`. The `format` string is described under the description of the `scanf` function.

The `vfscanf` function is equivalent to the `fscanf` function, with a variable argument list replaced with `arg`, which has been initialized using the `va_start` macro.

The `vfwscanf` function is identical to `vfscanf` except that it accepts a wide-character string argument for `format`.

**Returns:** The `vfscanf` function returns `EOF` if an input failure occurred before any conversion. Otherwise, the number of input arguments for which values were successfully scanned and stored is returned. When a file input error occurs, the `errno` global variable may be set.

**See Also:** `cscanf`, `fscanf`, `scanf`, `sscanf`, `va_arg`, `va_end`, `va_start`, `vscanf`, `vsscanf`

**Example:**
```
#include <stdio.h>
#include <stdarg.h>

void ffind( FILE *fp, char *format, ... )
{
    va_list arglist;

    va_start( arglist, format );
    vfscanf( fp, format, arglist );
    va_end( arglist );
}

void main( void )
{
    int day, year;
    char weekday[10], month[10];

    ffind( stdin, "%s %s %d %d", weekday, month, &day, &year );
    printf( "%s, %s %d, %d
", weekday, month, day, year );
}
```

**Classification:** `vfscanf` is ISO C99

---

1058 Library Functions and Macros
vfscanf, vfscanf

vfscanf is ISO C99

Systems:

vfscanf — All, Netware
vfscanf — All
**vfscanf_s, vfwscanf_s**

**Synopsis:**
```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdarg.h>
#include <stdio.h>
int vfscanf_s( FILE * restrict stream,
               const char * restrict format, va_list arg );
```
```
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>
int vfwscanf_s( FILE * restrict stream,
               const wchar_t * restrict format, va_list arg );
```

**Constraints:**
If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `vfscanf_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither `stream` nor `format` shall be a null pointer. Any argument indrected through in order to store converted input shall not be a null pointer.

If there is a runtime-constraint violation, the `vfscanf_s` function does not attempt to perform further input, and it is unspecified to what extent `vfscanf_s` performed input before discovering the runtime-constraint violation.

**Description:**
The `vfscanf_s` function is equivalent to `fscanf_s`, with the variable argument list replaced by `arg`, which shall have been initialized by the `va_start` macro (and possibly subsequent `va_arg` calls). The `vfscanf_s` function does not invoke the `va_end` macro.

The `vfwscanf_s` function is identical to `vfscanf_s` except that it accepts a wide-character string argument for `format`.

**Returns:**
The `vfscanf_s` function returns `EOF` if an input failure occurred before any conversion or if there was a runtime-constraint violation. Otherwise, the `vfscanf_s` function returns the number of input items successfully assigned, which can be fewer than provided for, or even zero.

When a file input error occurs, the `errno` global variable may be set.

**See Also:**
cscanf, fscanf, scanf, sscanf, va_arg, va_end, va_start, vcscanf, vfscanf, vscanf, vsscanf

**Example:**
```
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <stdarg.h>

void ffind( FILE *fp, char *format, ... )
{
    va_list arglist;

    va_start( arglist, format );
    vfscanf_s( fp, format, arglist );
    va_end( arglist );
}
```
void main( void )
{
    int day, year;
    char weekday[10], month[10];

    sscanf( stdin, "%s %s %d %d", weekday, sizeof( weekday ),
            month, sizeof( month ),
            &day, &year );
    printf_s( "\n%s, %s %d, %d\n", weekday, month, day, year );
}

Classification:  
  vfscanf_s is TR 24731  
  vfwscanf_s is TR 24731

Systems:  
  vfscanf_s - All, Netware  
  vfwscanf_s - All
Synopsis: #include <stdarg.h>
#include <stdio.h>
int vprintf( const char *format, va_list arg );
#include <stdarg.h>
#include <wchar.h>
int vwprintf( const wchar_t *format, va_list arg );

Safer C: The Safer C Library extension provides the vprintf_s function which is a safer alternative to vprintf. This newer vprintf_s function is recommended to be used instead of the traditional "unsafe" vprintf function.

Description: The vprintf function writes output to the file stdout under control of the argument format. The format string is described under the description of the printf function. The vprintf function is equivalent to the printf function, with the variable argument list replaced with arg, which has been initialized by the va_start macro.

The vwprintf function is identical to vprintf except that it accepts a wide-character string argument for format.

Returns: The vprintf function returns the number of characters written, or a negative value if an output error occurred. The vwprintf function returns the number of wide characters written, or a negative value if an output error occurred. When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also: _bprintf, cprintf, fprintf, printf, sprintf, va_arg, va_end, va_start, _vbprintf, vcprintf, vfprintf, vsprintf

Example: The following shows the use of vprintf in a general error message routine.

```c
#include <stdio.h>
#include <stdarg.h>

void errmsg( char *format, ... )
{
    va_list arglist;

    printf( "Error: " );
    va_start( arglist, format );
    vprintf( format, arglist );
    va_end( arglist );
}

void main( void )
{
    errmsg( "%s %d %s", "Failed", 100, "times" );
}
```

produces the following:

Error: Failed 100 times

Classification: vprintf is ANSI
vwprintf is ANSI

Systems: vprintf - All, Netware

1062 Library Functions and Macros
vprintf - All
Synopsis:
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdarg.h>
#include <stdio.h>
int vprintf_s( const char * restrict format,
               va_list arg );
#include <stdarg.h>
#include <wchar.h>
int vwprintf_s( const wchar_t * restrict format,
               va_list prg );

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and vprintf_s will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

The format argument shall not be a null pointer. The %n specifier (modified or not by flags, field width, or precision) shall not appear in the string pointed to by format. Any argument to vprintf_s corresponding to a %s specifier shall not be a null pointer.

If there is a runtime-constraint violation, the vprintf_s function does not attempt to produce further output, and it is unspecified to what extent vprintf_s produced output before discovering the runtime-constraint violation.

Description: The vprintf_s function is equivalent to the vprintf function except for the explicit runtime-constraints listed above.

The vwprintf_s function is identical to vprintf_s except that it accepts a wide-character string argument for format.

Returns: The vprintf_s function returns the number of characters written, or a negative value if an output error or runtime-constraint violation occurred.

The vwprintf_s function returns the number of wide characters written, or a negative value if an output error or runtime-constraint violation occurred.

See Also: _bprintf, cprintf, fprintf, printf, sprintf, _vbprintf, vcprintf, vfprintf, vprintf, vsprintf

Example:
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <stdarg.h>

void errmsg( char *format, ... )
{
    va_list arglist;

    printf_s( "Error: " );
    va_start( arglist, format );
    vprintf_s( format, arglist );
    va_end( arglist );
}
void main( void )
{
    errmsg( "%s %d %s", "Failed", 100, "times" );
}

produces the following:

Error: Failed 100 times

Classification:  vprintf_s is TR 24731
                        vwprintf_s is TR 24731

Systems:      vprintf_s - All, Netware
                        vwprintf_s - All
vscanf, vwscanf

Synopsis:
#include <stdarg.h>
#include <stdio.h>
int vscanf( const char *format,
    va_list arg );
#include <stdarg.h>
#include <wchar.h>
int vwscanf( const wchar_t *format,
    va_list arg );

Safer C:
The Safer C Library extension provides the vscanf_s function which is a safer alternative to vscanf. This newer vscanf_s function is recommended to be used instead of the traditional "unsafe" vscanf function.

Description:
The vscanf function scans input from the file designated by stdin under control of the argument format. The format string is described under the description of the scanf function.

The vscanf function is equivalent to the scanf function, with a variable argument list replaced with arg, which has been initialized using the va_start macro.

The vwscanf function is identical to vscanf except that it accepts a wide-character string argument for format.

Returns:
The vscanf function returns EOF if an input failure occurred before any conversion. values were successfully scanned and stored is returned.

See Also:
cscanf, fscanf, scanf, sscanf, va_arg, va_end, va_start, vcscanf, vfscanf, vsscanf

Example:
#include <stdio.h>
#include <stdarg.h>

void find( char *format, ... )
{
    va_list arglist;

    va_start( arglist, format );
    vscanf( format, arglist );
    va_end( arglist );
}

void main( void )
{
    int day, year;
    char weekday[10], month[10];

    find( "%s %s %d %d",
        weekday, month, &day, &year );
    printf( "%n%s, %s %d, %d
", weekday, month, day, year );
}

Classification: vscanf is ISO C99
vscanf is ISO C99

1066 Library Functions and Macros
Systems:  
  vscanf – All, Netware
  vwscanf – All
**vscanf_s, vwscanf_s**

**Synopsis:**
```c
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdarg.h>
#include <stdio.h>
int vscanf_s( const char * restrict format, va_list arg );
#include <stdarg.h>
#include <wchar.h>
int vwscanf_s( const wchar_t * restrict format, va_list arg );
```

**Constraints:**
If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `vscanf_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

The argument `format` shall not be a null pointer. Any argument indirecd through in order to store converted input shall not be a null pointer.

If there is a runtime-constraint violation, the `vscanf_s` function does not attempt to perform further input, and it is unspecified to what extent `vscanf_s` performed input before discovering the runtime-constraint violation.

**Description:**
The `vscanf_s` function is equivalent to `scanf_s`, with the variable argument list replaced by `arg`, which shall have been initialized by the `va_start` macro (and possibly subsequent `va_arg` calls). The `vscanf_s` function does not invoke the `va_end` macro.

The `vwscanf_s` function is identical to `vscanf_s` except that it accepts a wide-character string argument for `format`.

**Returns:**
The `vscanf_s` function returns `EOF` if an input failure occurred before any conversion or if there was a runtime-constraint violation. Otherwise, the `vscanf_s` function returns the number of input items successfully assigned, which can be fewer than provided for, or even zero.

When a file input error occurs, the `errno` global variable may be set.

**See Also:**
cscanf, fscanf, scanf, sscanf, va_arg, va_end, va_start, vscanf, vwscanf

**Example:**
```c
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <stdarg.h>

void find( char *format, ... )
{
    va_list arglist;
    va_start( arglist, format );
    vscanf_s( format, arglist );
    va_end( arglist );
}

void main( void )
{
    int day, year;
    char weekday[10], month[10];
```
find( "%s %s %d %d",
    weekday, sizeof( weekday ),
    month, sizeof( month ),
    &day, &year );
printf_s( "\n%s, %s %d, %d\n",
    weekday, month, day, year );
}

Classification: vscanf_s is TR 24731
               vwscanf_s is TR 24731

Systems:       vscanf_s - All, Netware
               vwscanf_s - All
Synopsis:

#include <stdarg.h>
#include <stdio.h>
int _vsnprintf( char *buf,
    size_t count,
    const char *format,
    va_list arg );

#include <stdarg.h>
#include <wchar.h>
int _vsnwprintf( wchar_t *buf,
    size_t count,
    const wchar_t *format,
    va_list arg );

Description:
The _vsnprintf function formats data under control of the format control string and stores the result in buf. The maximum number of characters to store is specified by count. A null character is placed at the end of the generated character string if fewer than count characters were stored. The format string is described under the description of the printf function. The _vsnprintf function is equivalent to the _snprintf function, with the variable argument list replaced with arg, which has been initialized by the va_start macro.

The _vsnwprintf function is identical to _vsnprintf except that the argument buf specifies an array of wide characters into which the generated output is to be written, rather than converted to multibyte characters and written to a stream. The maximum number of wide characters to write is specified by count. A null wide character is placed at the end of the generated wide character string if fewer than count wide characters were stored. The _vsnwprintf function accepts a wide-character string argument for format.

Returns:
The _vsnprintf function returns the number of characters written into the array, not counting the terminating null character, or a negative value if more than count characters were requested to be generated. An error can occur while converting a value for output. The _vsnwprintf function returns the number of wide characters written into the array, not counting the terminating null wide character, or a negative value if more than count wide characters were requested to be generated. When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also:
_bprintf, cprintf, fprintf, printf, sprintf, va_arg, va_end, va_start, _vbprintf, vcprintf, vfprintf, vprintf, vsprintf

Example:
The following shows the use of _vsnprintf in a general error message routine.

#include <stdio.h>
#include <stdarg.h>
#include <string.h>

char msgbuf[80];

char *fmtmsg( char *format, ... )
{
    va_list arglist;
    va_start( arglist, format );
    strcpy( msgbuf, "Error: " );
    _vsnprintf( &msgbuf[7], 80-7, format, arglist );
    va_end( arglist );
    return( msgbuf );
}
void main()
{
    char *msg;

    msg = fmtmsg( "%s %d %s", "Failed", 100, "times" );
    printf( "%s\n", msg );
}

Classification: WATCOM

Systems: _vsnprintf - All, Netware
         _vsnwprintf - All
vsnprintf, vsnwprintf

Synopsis:  
#include <stdarg.h>  
#include <stdio.h>  
int vsnprintf( char *buf, 
size_t count,  
const char *format,  
va_list arg );  
#include <stdarg.h>  
#include <wchar.h>  
int vsnwprintf( wchar_t *buf, 
size_t count,  
const wchar_t *format,  
va_list arg );

Safer C:  
The Safer C Library extension provides the vsnprintf_s function which is a safer alternative to vsnprintf. This newer vsnprintf_s function is recommended to be used instead of the traditional "unsafe" vsnprintf function.

Description:  
The vsnprintf function formats data under control of the format control string and stores the result in buf. The maximum number of characters to store, including a terminating null character, is specified by count. The format string is described under the description of the printf function. The vsnprintf function is equivalent to the _snprintf function, with the variable argument list replaced with arg, which has been initialized by the va_start macro.

The vsnwprintf function is identical to vsnprintf except that the argument buf specifies an array of wide characters into which the generated output is to be written, rather than converted to multibyte characters and written to a stream. The maximum number of wide characters to write, including a terminating null wide character, is specified by count. The vsnwprintf function accepts a wide-character string argument for format.

Returns:  
The vsnprintf function returns the number of characters that would have been written had count been sufficiently large, not counting the terminating null character, or a negative value if an encoding error occurred. Thus, the null-terminated output has been completely written if and only if the returned value is nonnegative and less than count. The vsnwprintf function returns the number of wide characters that would have been written had count been sufficiently large, not counting the terminating null wide character, or a negative value if an encoding error occurred. Thus, the null-terminated output has been completely written if and only if the returned value is nonnegative and less than count. When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also:  
_bprintf, cprintf, fprintf, printf, sprintf, va_arg, va_end, va_start,  
 vbprintf, wcprintf, wprintf, vfprintf, vprintf, vsprintf

Example:  
The following shows the use of vsnprintf in a general error message routine.
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>

char *fmtmsg( char *format, ... )
{
    char    *msgbuf;
    int     len;
    va_list arglist;

    va_start( arglist, format );
    len = vsnprintf( NULL, 0, format, arglist );
    va_end( arglist );
    len = len + 1 + 7;
    msgbuf = malloc( len );
    strcpy( msgbuf, "Error: " );
    va_start( arglist, format );
    vsnprintf( &msgbuf[7], len, format, arglist );
    va_end( arglist );
    return( msgbuf );
}

void main( void )
{
    char *msg;

    msg = fmtmsg( "%s %d %s", "Failed", 100, "times" );
    printf( "%s\n", msg );
    free( msg );
}

Classification: vsnprintf is ANSI
                vsnwprintf is ANSI

Systems:       vsnprintf - All, Netware
                vsnwprintf - All
vsnprintf_s, vsnprintf_s

Synopsis:  
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdarg.h>
#include <stdio.h>
int vsnprintf_s( char * restrict s, rsize_t n
    const char * restrict format, va_list arg );
#include <stdarg.h>
#include <wchar.h>
int vsnwprintf_s( char * restrict s, rsize_t n,
    const wchar_t * restrict format, va_list arg );

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and vsnprintf_s will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither s nor format shall be a null pointer. The n argument shall neither equal zero nor be greater than RSIZE_MAX. The number of characters (including the trailing null) required for the result to be written to the array pointed to by s shall not be greater than n. The %n specifier (modified or not by flags, field width, or precision) shall not appear in the string pointed to by format. Any argument to vsnprintf_s corresponding to a %s specifier shall not be a null pointer. No encoding error shall occur.

If there is a runtime-constraint violation, then if s is not a null pointer and n is greater than zero and less than RSIZE_MAX, then the vsnprintf_s function sets s[0] to the null character.

Description: The vsnprintf_s function is equivalent to the vsnprintf function except for the explicit runtime-constraints listed above.

The vsnprintf_s function, unlike vsprintf_s, will truncate the result to fit within the array pointed to by s.

The vsnprintf_s function is identical to vsnprintf_s except that it accepts a wide-character string argument for format and produces wide character output.

Returns: The vsnprintf_s function returns the number of characters that would have been written had n been sufficiently large, not counting the terminating null character, or a negative value if a runtime-constraint violation occurred. Thus, the null-terminated output has been completely written if and only if the returned value is nonnegative and less than n.

The vsnwprintf_s function returns the number of wide characters that would have been written had n been sufficiently large, not counting the terminating wide null character, or a negative value if a runtime-constraint violation occurred. Thus, the null-terminated output has been completely written if and only if the returned value is nonnegative and less than n.

See Also: _bprintf, cprintf, fprintf, printf, sprintf, _vbprintf, wcprintf, wfprintf, vprintf, vsprintf

Example: The following shows the use of vsnprintf_s in a general error message routine.
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>

char *fmtmsg( char *format, ... )
{
    char    *msgbuf;
    int     len;
    va_list arglist;

    va_start( arglist, format );
    len = vsnprintf( NULL, 0, format, arglist );
    va_end( arglist );
    len = len + 1 + 7;
    msgbuf = malloc( len );
    strcpy( msgbuf, "Error: " );
    va_start( arglist, format );
    vsnprintf_s( &msgbuf[7], len, format, arglist );
    va_end( arglist );
    return( msgbuf );
}

void main( void )
{
    char *msg;
    msg = fmtmsg( "%s %d %s", "Failed", 100, "times" );
    printf_s( "%s\n", msg );
    free( msg );
}

Classification: vsnprintf_s is TR 24731
                  vsnwprintf_s is TR 24731
Systems:         vsnprintf_s - All, Netware
                  vsnwprintf_s - All

Library Functions and Macros 1075
Includes:

```c
#include <stdarg.h>
#include <stdio.h>
#include <string.h>
```

```c
int vsprintf( char *buf,
             const char *format,
             va_list arg );

#include <stdarg.h>
#include <wchar.h>

int vswprintf( wchar_t *buf,
               size_t count,
               const wchar_t *format,
               va_list arg );
```

**Safer C:** The Safer C Library extension provides the `vsprintf_s` function which is a safer alternative to `vsprintf`. This newer `vsprintf_s` function is recommended to be used instead of the traditional "unsafe" `vsprintf` function.

**Description:** The `vsprintf` function formats data under control of the `format` control string and writes the result to `buf`. The `format` string is described under the description of the `printf` function. The `vsprintf` function is equivalent to the `sprintf` function, with the variable argument list replaced with `arg`, which has been initialized by the `va_start` macro.

The `vswprintf` function is identical to `vsprintf` except that the argument `buf` specifies an array of wide characters into which the generated output is to be written, rather than converted to multibyte characters and written to a stream. The maximum number of wide characters to write, including a terminating null wide character, is specified by `count`. The `vswprintf` function accepts a wide-character string argument for `format`.

**Returns:** The `vsprintf` function returns the number of characters written, or a negative value if an output error occurred. The `vswprintf` function returns the number of wide characters written into the array, not counting the terminating null wide character, or a negative value if `count` or more wide characters were requested to be generated.

**See Also:** `_bprintf`, `cprintf`, `fprintf`, `printf`, `sprintf`, `va_arg`, `va_end`, `va_start`, `_vbprintf`, `vcprintf`, `vfprintf`, `vprintf`

**Example:** The following shows the use of `vsprintf` in a general error message routine.

```c
#include <stdio.h>
#include <stdarg.h>
#include <string.h>

char msgbuf[80];

char *fmtmsg( char *format, ... )
{
    va_list arglist;
    va_start( arglist, format );
    strcpy( msgbuf, "Error: " );
    vsprintf( &msgbuf[7], format, arglist );
    va_end( arglist );
    return( msgbuf );
}
```
void main( void )
{
    char *msg;
    msg = fmtmsg("%s %d %s", "Failed", 100, "times");
    printf("%s\n", msg);
}

Classification: vsprintf is ANSI
    vswprintf is ANSI

Systems:    vsprintf - All, Netware
            vswprintf - All
vsprintf_s, vswprintf_s

Synopsis:
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <stdarg.h>
int vsprintf_s( char * restrict s, rsize_t n,
    const char * restrict format, va_list arg );

#include <stdio.h>
#include <wchar.h>
int vswprintf_s( char * restrict s, rsize_t n,
    const wchar_t * restrict format, va_list arg );

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler
will be invoked and vsprintf_s will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither s nor format shall be a null pointer. The n argument shall neither equal zero nor be greater than
RSIZE_MAX. The number of characters (including the trailing null) required for the result to be
written to the array pointed to by s shall not be greater than n. The %n specifier (modified or not by
flags, field width, or precision) shall not appear in the string pointed to by format. Any argument to
vsprintf_s corresponding to a %s specifier shall not be a null pointer. No encoding error shall
occur.

If there is a runtime-constraint violation, then if s is not a null pointer and n is greater than zero and less
than RSIZE_MAX, then the vsprintf_s function sets s[0] to the null character.

Description: The vsprintf_s function is equivalent to the vsprintf function except for the explicit
runtime-constraints listed above.

The vsprintf_s function, unlike vsnprintf_s, treats a result too big for the array pointed to by
s as a runtime-constraint violation.

The vswprintf_s function is identical to vsprintf_s except that it accepts a wide-character
string argument for format and produces wide character output.

Returns: If no runtime-constraint violation occurred, the vsprintf_s function returns the number of
characters written in the array, not counting the terminating null character. If an encoding error
occurred, vsprintf_s returns a negative value. If any other runtime-constraint violation occurred,
vsprintf_s returns zero.

If no runtime-constraint violation occurred, the vswprintf_s function returns the number of wide
characters written in the array, not counting the terminating null wide character. If an encoding error
occurred or if n or more wide characters are requested to be written, vswprintf_s returns a negative
value. If any other runtime-constraint violation occurred, vswprintf_s returns zero.

See Also: _bprintf, cprintf, fprintf, printf, sprintf, _vbprintf, vcprintf, vfprintf, vprintf, vsprintf

Example: The following shows the use of vsprintf_s in a general error message routine.

#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <stdarg.h>
#include <string.h>

char msgbuf[80];

1078 Library Functions and Macros
char *fmtmsg( char *format, ... )
{
    va_list arglist;

    va_start( arglist, format );
    strcpy_s( msgbuf, sizeof( buffer ), "Error: " );
    vsprintf_s( &msgbuf[7], sizeof( msgbuf ) - 7,
                format, arglist );
    va_end( arglist );
    return( msgbuf );
}

void main( void )
{
    char *msg;

    msg = fmtmsg( "%s %d %s", "Failed", 100, "times" );
    printf( "%s
", msg );
}

Classification: vsprintf_s is TR 24731
               vswprintf_s is TR 24731

Systems:      vsprintf_s - All, Netware
               vswprintf_s - All
vsscanf, vswscanf

Synopsis:  
#include <stdio.h>  
#include <stdarg.h>  
int vsscanf( const char *in_string,  
const char *format,  
va_list arg );  
int vswscanf( const wchar_t *in_string,  
const wchar_t *format,  
va_list arg );

Safer C:  
The Safer C Library extension provides the vsscanf_s function which is a safer alternative to vsscanf. This newer vsscanf_s function is recommended to be used instead of the traditional "unsafe" vsscanf function.

Description: 
The vsscanf function scans input from the string designated by in_string under control of the argument format. The format string is described under the description of the scanf function.

The vsscanf function is equivalent to the sscanf function, with a variable argument list replaced with arg, which has been initialized using the va_start macro.

The vswscanf function is identical to vsscanf except that it accepts a wide-character string argument for format.

Returns:  
The vsscanf function returns EOF if the end of the input string was reached before any conversion. Otherwise, the number of input arguments for which values were successfully scanned and stored is returned.

See Also: cscanf, fscanf, scanf, sscanf, va_arg, va_end, va_start, vscanf

Example:  
#include <stdio.h>  
#include <stdarg.h>  

    void sfind( char *string, char *format, ... )
    {
        va_list arglist;

        va_start( arglist, format );
        vsscanf( string, format, arglist );
        va_end( arglist );
    }

    void main( void )
    {
        int day, year;
        char weekday[10], month[10];

        sfind( "Saturday April 18 1987",
               "%s %s %d %d",
               weekday, month, &day, &year );
        printf( "\n%s, %s %d, %d",
                weekday, month, day, year );
    }

Classification: vsscanf is ISO C99

1080 Library Functions and Macros
vswscanf is ISO C99

Systems:  
  vsscanf - All, Netware  
  vswscanf - All
Synopsis:
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdarg.h>
#include <stdio.h>
int vsscanf_s( const char * restrict s,
            const char * restrict format,
             va_list arg );
#include <stdio.h>
#include <wchar.h>
int vswscanf_s( const wchar_t * restrict s,
            const wchar_t * restrict format,
             va_list arg );

Constraints:
If any of the following runtime-constraints is violated, the currently active runtime-constraint handler
will be invoked and vsscanf_s will return a non-zero value to indicate an error, or the
runtime-constraint handler aborts the program.

Neither s nor format shall be a null pointer. Any argument indirected through in order to store
converted input shall not be a null pointer.

If there is a runtime-constraint violation, the vsscanf_s function does not attempt to perform further
input, and it is unspecified to what extent vsscanf_s performed input before discovering the
runtime-constraint violation.

Description:
The vsscanf_s function is equivalent to sscanf_s, with the variable argument list replaced by arg,
which shall have been initialized by the va_start macro (and possibly subsequent va_arg calls).
The vsscanf_s function does not invoke the va_end macro.

The vswscanf_s function is identical to vsscanf_s except that it accepts wide-character string
arguments for s and format.

Returns:
The vsscanf_s function returns EOF if an input failure occurred before any conversion or if there
was a runtime-constraint violation. Otherwise, the vsscanf_s function returns the number of input
items successfully assigned, which can be fewer than provided for, or even zero.

When a file input error occurs, the errno global variable may be set.

See Also: cscanf, fscanf, scanf, sscanf, va_arg, va_end, va_start, vscanf, vswscanf

Example:
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <stdarg.h>

void sfind( char *string, char *format, ... )
{
    va_list arglist;
    va_start( arglist, format );
    vsscanf_s( string, format, arglist );
    va_end( arglist );
}
void main( void )
{
    int day, year;
    char weekday[10], month[10];

    sfind( "Friday August 0013 2004",
           "%s %s %d %d",
           weekday, sizeof( weekday ),
           month, sizeof( month ),
           &day, &year );
    printf_s( "\n%s, %s %d, %d",
             weekday, month, day, year );
}

produces the following:

Friday, August 13, 2004

Classification: vsscanf_s is TR 24731
               vswscanf_s is TR 24731

Systems:      vsscanf_s - All, Netware
              vswscanf_s - All
**wait**

**Synopsis:**
```c
#include <process.h>
int wait( int *status );
```

**Description:** The `wait` function suspends the calling process until any of the caller’s immediate child processes terminate.

Under Win32, there is no parent-child relationship amongst processes so the `wait` function cannot and does not wait for child processes to terminate. To wait for any process, you must specify its process id. For this reason, the `cwait` function should be used (one of its arguments is a process id).

If `status` is not `NULL`, it points to a word that will be filled in with the termination status word and return code of the terminated child process.

If the child process terminated normally, then the low order byte of the status word will be set to 0, and the high order byte will contain the low order byte of the return code that the child process passed to the `DOSEXIT` function. The `DOSEXIT` function is called whenever `main` returns, or `exit` or `_exit` are explicitly called.

If the child process did not terminate normally, then the high order byte of the status word will be set to 0, and the low order byte will contain one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hard-error abort</td>
</tr>
<tr>
<td>2</td>
<td>Trap operation</td>
</tr>
<tr>
<td>3</td>
<td>SIGTERM signal not intercepted</td>
</tr>
</tbody>
</table>

**Note:** This implementation of the status value follows the OS/2 model and differs from the Microsoft implementation. Under Microsoft, the return code is returned in the low order byte and it is not possible to determine whether a return code of 1, 2, or 3 imply that the process terminated normally. For portability to Microsoft compilers, you should ensure that the application that is waited on does not return one of these values. The following shows how to handle the status value in a portable manner.
cwait( &status, process_id, WAIT_CHILD );

#if defined(__WATCOMC__)
switch( status & 0xff ) {
    case 0:
        printf( "Normal termination exit code = %d\n", status >> 8 );
        break;
    case 1:
        printf( "Hard-error abort\n" );
        break;
    case 2:
        printf( "Trap operation\n" );
        break;
    case 3:
        printf( "SIGTERM signal not intercepted\n" );
        break;
    default:
        printf( "Bogus return status\n" );
        break;
}  
#else if defined(_MSC_VER)
switch( status & 0xff ) {
    case 1:
        printf( "Possible Hard-error abort\n" );
        break;
    case 2:
        printf( "Possible Trap operation\n" );
        break;
    case 3:
        printf( "Possible SIGTERM signal not intercepted\n" );
        break;
    default:
        printf( "Normal termination exit code = %d\n", status );
        break;
}  
#endif

Returns: The wait function returns the child’s process id if the child process terminated normally. Otherwise, wait returns -1 and sets errno to one of the following values:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECHILD</td>
<td>No child processes exist for the calling process.</td>
</tr>
<tr>
<td>EINTR</td>
<td>The child process terminated abnormally.</td>
</tr>
</tbody>
</table>

See Also: cwait, exit, _exit, spawn...

Example:

```c
#include <stdlib.h>
#include <process.h>

void main()
{
    int   process_id, status;

    process_id = spawnl( P_NOWAIT, "child.exe", "child", "parm", NULL );
    wait( &status );
}
```

Classification: WATCOM
wait

Systems: Win32, QNX, OS/2 1.x(all), OS/2-32
Synopsis:

```c
#include <wchar.h>

int wcrtomb( char *s, wchar_t wc, mbstate_t *ps );
int _fwcrtomb( char __far *s, wchar_t wc, mbstate_t __far *ps );
```

Safer C:
The Safer C Library extension provides the `wcrtomb_s` function which is a safer alternative to `wcrtomb`. This newer `wcrtomb_s` function is recommended to be used instead of the traditional "unsafe" `wcrtomb` function.

Description:
If `s` is a null pointer, the `wcrtomb` function determines the number of bytes necessary to enter the initial shift state (zero if encodings are not state-dependent or if the initial conversion state is described). The resulting state described will be the initial conversion state.

If `s` is not a null pointer, the `wcrtomb` function determines the number of bytes needed to represent the multibyte character that corresponds to the wide character given by `wc` (including any shift sequences), and stores the resulting bytes in the array whose first element is pointed to by `s`. At most `MB_CUR_MAX` bytes will be stored. If `wc` is a null wide character, the resulting state described will be the initial conversion state.

The `_fwcrtomb` function is a data model independent form of the `wcrtomb` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The restartable multibyte/wide character conversion functions differ from the corresponding internal-state multibyte character functions (`mblen`, `mbtowc`, and `wctomb`) in that they have an extra argument, `ps`, of type pointer to `mbstate_t` that points to an object that can completely describe the current conversion state of the associated multibyte character sequence. If `ps` is a null pointer, each function uses its own internal `mbstate_t` object instead. You are guaranteed that no other function in the library calls these functions with a null pointer for `ps`, thereby ensuring the stability of the state.

Also unlike their corresponding functions, the return value does not represent whether the encoding is state-dependent.

If the encoding is state-dependent, on entry each function takes the described conversion state (either internal or pointed to by `ps`) as current. The conversion state described by the pointed-to object is altered as needed to track the shift state of the associated multibyte character sequence. For encodings without state dependency, the pointer to the `mbstate_t` argument is ignored.

Returns:
If `s` is a null pointer, the `wcrtomb` function returns the number of bytes necessary to enter the initial shift state. The value returned will not be greater than that of the `MB_CUR_MAX` macro.

If `s` is not a null pointer, the `wcrtomb` function returns the number of bytes stored in the array object (including any shift sequences) when `wc` is a valid wide character; otherwise (when `wc` is not a valid wide character), an encoding error occurs, the value of the macro `EILSEQ` will be stored in `errno` and `-1` will be returned, but the conversion state will be unchanged.

See Also:
`_mbccmp, _mbccpy, _mbccicmp, _mbcjistojms, _mbcjstojis, _mbclen, _mbctohira, _mbctokata, _mbctolower, _mbctombb, _mbctoupper, mblen, mbrlen, mbttowc, mbsrtowcs, mbsrtowcs_s, mbstowcs, mbstowcs_s, mbtowc, btowc, wcrtomb_s, wcsrtombs, wcsrtombs_s, wcstombs, wcstombs_s, wctob, wctomb, wctomb_s`
Example:

```c
#include <stdio.h>
#include <wchar.h>
#include <mbctype.h>
#include <errno.h>

const wchar_t wc[] = {
    0x0020, /* double-byte space */
    0x002e,
    0x0031,
    0x0041,
    0x3000,     /* double-byte space */
    0xff21,     /* double-byte A */
    0x3048,     /* double-byte Hiragana */
    0x30a3,     /* double-byte Katakana */
    0xff61,     /* single-byte Katakana punctuation */
    0xff66,     /* single-byte Katakana alphabetic */
    0xff9f,     /* single-byte Katakana alphabetic */
    0x720d,     /* double-byte Kanji */
    0x0000
};

#define SIZE sizeof( wc ) / sizeof( wchar_t )

void main()
{
    int         i, j, k;
    char        s[2];

    _setmbcp( 932 );
    i = wcrtomb( NULL, 0, NULL );
    printf( "Number of bytes to enter " 
        "initial shift state = %d\n", i );
    j = 1;
    for( i = 0; i < SIZE; i++ ) {
        j = wcrtomb( s, wc[i], NULL );
        printf( "%d bytes in character ", j );
        if( errno == EILSEQ ) {
            printf( " - illegal wide character\n" );
        } else {
            if ( j == 0 ) {
                k = 0;
            } else if ( j == 1 ) {
                k = s[0];
            } else if( j == 2 ) {
                k = s[0] << 8 | s[1];
            }
            printf( "( %#6.4x->%#6.4x)\n", wc[i], k );
        }
    }
    printf( "\n" );
}
```

produces the following:
Number of bytes to enter initial shift state = 0
1 bytes in character (0x0020->0x0020)
1 bytes in character (0x002e->0x002e)
1 bytes in character (0x0031->0x0031)
1 bytes in character (0x0041->0x0041)
2 bytes in character (0x3000->0x8140)
2 bytes in character (0xff21->0x8260)
2 bytes in character (0x3048->0x82a6)
2 bytes in character (0x30a3->0x8342)
1 bytes in character (0xff61->0x00a1)
1 bytes in character (0xff66->0x00a6)
1 bytes in character (0xff9f->0x00df)
2 bytes in character (0x720d->0xe0a1)
1 bytes in character (0000->0x0069)

**Classification:** wcrtomb is ANSI
 fwctomb is not ANSI

**Systems:**
wcrtomb - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
fwctomb - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
# wcrtomb_s, _fwcrtomb_s

**Synopsis:**
```c
#include <wchar.h>

errno_t wcrtomb_s( size_t * restrict retval, char * restrict s, rsize_t smax, wchar_t wc, mbstate_t * restrict ps);

errno_t _wcrtomb_s( size_t __far * restrict retval, char __far * restrict s, rsize_t smax, wchar_t __far * restrict wc, mbstate_t __far * restrict ps);
```

**Constraints:**
If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and `wcrtomb_s` will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Neither `retval` nor `ps` shall be a null pointer. If `s` is not a null pointer, then `smax` shall not equal zero and shall not be greater than `RSIZE_MAX`. If `s` is a null pointer, then `smax` shall be not be less than the number of bytes to be stored in the array pointed to by `s`. If `s` is a null pointer, then `smax` shall equal zero.

If there is a runtime-constraint violation, then `wcrtomb_s` does the following. If `s` is not a null pointer and `smax` is greater than zero and not greater than `RSIZE_MAX`, then `wcrtomb_s` sets `s[0]` to the null character. If `retval` is not a null pointer, then `wcrtomb_s` sets `*retval` to `(size_t)(-1).

**Description:**
If `s` is a null pointer, the `wcrtomb_s` function is equivalent to the call `wcrtomb_s(&retval, buf, sizeof buf, L'\0', ps)` where `retval` and `buf` are internal variables of the appropriate types, and the size of `buf` is greater than `MB_CUR_MAX`.

If `s` is not a null pointer, the `wcrtomb_s` function determines the number of bytes needed to represent the multibyte character that corresponds to the wide character given by `wc` (including any shift sequences), and stores the multibyte character representation in the array whose first element is pointed to by `s`. At most `MB_CUR_MAX` bytes are stored. If `wc` is a null wide character, a null byte is stored, preceded by any shift sequence needed to restore the initial shift state; the resulting state described is the initial conversion state.

If `wc` does not correspond to a valid multibyte character, an encoding error occurs: the `wcrtomb_s` function stores the value `(size_t)(-1)` into `*retval` and the conversion state is unspecified. Otherwise, the `wcrtomb_s` function stores into `*retval` the number of bytes (including any shift sequences) stored in the array pointed to by `s`.

The `_fwcrtomb_s` function is a data model independent form of the `wcrtomb_s` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

**Returns:**
The `wcrtomb_s` function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

**See Also:**

- `_mbccmp`, `_mbccpy`, `_mbccicmp`, `_mbcJistojms`, `_mbcJmsjtojis`, `_mbc clen`, `_mbctohira`, `_mbctokata`, `_mbctolower`, `_mbctombb`, `_mbctoupper`, `mblen`, `mbrlen`, `mbtowc`, `mbsrtowcs`, `mbsrtowcs_s`, `mbstowcs`, `mbstowcs_s`, `mbtowc`, `btowc`, `wcrtomb`, `wcsrtombs`, `wcsrtombs_s`, `wcstombs`, `wcstombs_s`, `wctob`, `wctomb`, `wctomb_s`
Example:

```c
#define __STDC_WANT_LIB_EXT1__  1
#include <stdio.h>
#include <wchar.h>
#include <mbctype.h>
#include <errno.h>

const wchar_t wc[] = {
    0x0020,
    0x002e,
    0x0031,
    0x0041,
    0x3000,     /* double-byte space */
    0xff21,     /* double-byte A */
    0x3048,     /* double-byte Hiragana */
    0x30a3,     /* double-byte Katakana */
    0xff61,     /* single-byte Katakana punctuation */
    0xff66,     /* single-byte Katakana alphabetic */
    0xff9f,     /* single-byte Katakana alphabetic */
    0x720d,     /* double-byte Kanji */
    0x0000
};

#define SIZE sizeof( wc ) / sizeof( wchar_t )

int main()
{
    int         i, j, k;
    char        s[2];
    errno_t     rc;
    size_t      retval;
    mbstate_t   state;

    _setmbcp( 932 );
    j = 1;
    for( i = 0; i < SIZE; i++ ) {
        rc = wcrtomb_s( &retval, s, 2, wc[i], &state );
        if( rc != 0 ) {
            printf( " - illegal wide character
" );
        } else {
            printf( "%d bytes in character ", retval );
            if ( retval == 0 ) {
                k = 0;
            } else if ( retval == 1 ) {
                k = s[0];
            } else if ( retval == 2 ) {
                k = s[0]<8 | s[1];
            }
            printf( "(%#6.4x->%#6.4x)\n", wc[i], k );
        }
    }
    return( 0 );
}
```

produces the following:


**wcrtomb_s, _fwcrtomb_s**

1 bytes in character (0x0020->0x0020)
1 bytes in character (0x002e->0x002e)
1 bytes in character (0x0031->0x0031)
1 bytes in character (0x0041->0x0041)
2 bytes in character (0x3000->0x8140)
2 bytes in character (0xff21->0x8260)
2 bytes in character (0x3048->0x82a6)
2 bytes in character (0x30a3->0x8342)
1 bytes in character (0xff61->0x00a1)
1 bytes in character (0xff66->0x00a6)
1 bytes in character (0xff9f->0x00df)
2 bytes in character (0x720d->0xe0a1)
1 bytes in character (0x0000->0x0069)

**Classification:** wcrtomb_s is TR24731
/fwcrctomb_s is WATCOM

**Systems:**
wcrctomb_s - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
/fwcrctomb_s - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

```c
#include <wchar.h>
size_t wcsrtombs( char *dst,
    const wchar_t **src,
    size_t n, mbstate_t *ps );

#include <wchar.h>
size_t _fwcsrtombs( char __far *dst,
    const wchar_t __far *__far *src,
    size_t n, mbstate_t __far *__far *ps );
```

Safer C: The Safer C Library extension provides the `wcsrtombs_s` function which is a safer alternative to `wcsrtombs`. This newer `wcsrtombs_s` function is recommended to be used instead of the traditional "unsafe" `wcsrtombs` function.

Description: The `wcsrtombs` function converts a sequence of wide characters from the array indirectly pointed to by `src` into a sequence of corresponding multibyte characters that begins in the shift state described by `ps`, which, if `dst` is not a null pointer, are then stored into the array pointed to by `dst`. Conversion continues up to and including a terminating null wide character, but the terminating null character (byte) will not be stored. Conversion will stop earlier in two cases: when a code is reached that does not correspond to a valid multibyte character, or (if `dst` is not a null pointer) when the next multibyte character would exceed the limit of `len` total bytes to be stored into the array pointed to by `dst`. Each conversion takes place as if by a call to the `wcrtomb` function.

If `dst` is not a null pointer, the pointer object pointed to by `src` will be assigned either a null pointer (if conversion stopped due to reaching a terminating null wide character) or the address just past the last wide character converted. If conversion stopped due to reaching a terminating null wide character and if `dst` is not a null pointer, the resulting state described will be the initial conversion state.

The `_fwcsrtombs` function is a data model independent form of the `wcsrtombs` function that accepts far pointer arguments. It is most useful in mixed memory model applications.

The restartable multibyte/wide string conversion functions differ from the corresponding internal-state multibyte string functions (`mbstowcs` and `wcstombs`) in that they have an extra argument, `ps`, of type pointer to `mbstate_t` that points to an object that can completely describe the current conversion state of the associated multibyte character sequence. If `ps` is a null pointer, each function uses its own internal `mbstate_t` object instead. You are guaranteed that no other function in the library calls these functions with a null pointer for `ps`, thereby ensuring the stability of the state.

Also unlike their corresponding functions, the conversion source argument, `src`, has a pointer-to-pointer type. When the function is storing conversion results (that is, when `dst` is not a null pointer), the pointer object pointed to by this argument will be updated to reflect the amount of the source processed by that invocation.

If the encoding is state-dependent, on entry each function takes the described conversion state (either internal or pointed to by `ps`) as current and then, if the destination pointer, `dst`, is not a null pointer, the conversion state described by the pointed-to object is altered as needed to track the shift state of the associated multibyte character sequence. For encodings without state dependency, the pointer to the `mbstate_t` argument is ignored.

Returns: If the first code is not a valid wide character, an encoding error occurs: The `wcsrtombs` function stores the value of the macro `EILSEQ` in `errno` and returns `(size_t)-1`, but the conversion state will be unchanged. Otherwise, it returns the number of bytes in the resulting multibyte characters sequence, which is the same as the number of array elements modified when `dst` is not a null pointer.
**wcsrtombs, _wcsrtombs**

See Also:  
_mbccmp, _mbccpy, _mbccicmp, _mbccjistojms, _mbccjmstojis, _mbclen, _mbctohira, _mbctokata, _mbctolower, _mbctombb, _mbctoupper, mblen, mbrlen, mbrtowc, mbsrtowcs, mbsrtowcs_s, mbstowcs, mbstowcs_s, mbtowc, btowc, wcrtomb, wcrtomb_s, wcsrtombs_s, wcstombs, wcstombs_s, wctob, wctomb, wctomb_s

Example:  
```c
#include <stdio.h>
#include <wchar.h>
#include <mbctype.h>
#include <errno.h>

const wchar_t wc[] = {
  0x0020,  /* double-byte space */
  0x002e,
  0x0031,
  0x0041,
  0x3000,  /* double-byte space */
  0xff21,  /* double-byte A */
  0x3048,  /* double-byte Hiragana */
  0x30a3,  /* double-byte Katakana */
  0xff61,  /* single-byte Katakana punctuation */
  0xff66,  /* single-byte Katakana alphabetic */
  0xff9f,  /* single-byte Katakana alphabetic */
  0x720d,  /* double-byte Kanji */
  0x0000
};

void main() {

  int     i;
  size_t  elements;
  const wchar_t *src;
  char    mb[50];
  mbstate_t  pstate;

  _setmbcp( 932 );
  src = wc;
  elements = wcsrtombs( mb, &src, 50, &pstate );
  if( errno == EILSEQ ) {
    printf( "Error in wide character string\n" );
  } else {
    for( i = 0; i < elements; i++ ) {
      printf( "0x%2x\n", mb[i] );
    }
  }
}
```

produces the following:
wcsrtombs, _fwcsrtombs

0x20
0x2e
0x31
0x41
0x81
0x40
0x82
0x60
0x82
0xa6
0x83
0x42
0xa1
0xa6
0xdf
0xe0
0xa1

**Classification:** wcsrtombs is ANSI

_fwcsrtombs is not ANSI

**Systems:** wcsrtombs - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

_fwcsrtombs - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

```c
#define __STDC_WANT_LIB_EXT1__ 1
errno_t wcsrtombs_s( size_t * restrict retval,
                    char * restrict dst,
                    rsize_t dstmax,
                    const wchar_t ** restrict src,
                    rsize_t len,
                    mbstate_t * restrict ps);
errno_t _fwcsrtombs_s( size_t __far * restrict retval,
                       char __far * restrict dst,
                       rsize_t dstmax,
                       const wchar_t __far * __far * restrict src,
                       rsize_t len,
                       mbstate_t __far * restrict ps);
```

Constraints:
If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and wcsrtombs_s will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

None of `retval`, `src`, `*src`, or `ps` shall be null pointers. If `dst` is not a null pointer, then neither `len` nor `dstmax` shall be greater than `RSIZE_MAX`. If `dst` is a null pointer, then `dstmax` shall equal zero. If `dst` is not a null pointer, then `dstmax` shall not equal zero. If `dst` is not a null pointer and `len` is not less than `dstmax`, then the conversion shall have been stopped (see below) because a terminating null wide character was reached or because an encoding error occurred.

If there is a runtime-constraint violation, then `wcsrtombs_s` does the following. If `retval` is not a null pointer, then `wcsrtombs_s` sets `*retval` to `(size_t)(-1)`. If `dst` is not a null pointer and `dstmax` is greater than zero and less than `RSIZE_MAX`, then `wcsrtombs_s` sets `dst[0]` to the null character.

Description:
The `wcsrtombs_s` function converts a sequence of wide characters from the array indirectly pointed to by `src` into a sequence of corresponding multibyte characters that begins in the conversion state described by the object pointed to by `ps`. If `dst` is not a null pointer, the converted characters are then stored into the array pointed to by `dst`. Conversion continues up to and including a terminating null wide character, which is also stored.

Conversion stops earlier in two cases:

- when a wide character is reached that does not correspond to a valid multibyte character;
- (if `dst` is not a null pointer) when the next multibyte character would exceed the limit of `n` total bytes to be stored into the array pointed to by `dst`. If the wide character being converted is the null wide character, then `n` is the lesser of `len` or `dstmax`. Otherwise, `n` is the lesser of `len` or `dstmax-1`.

If the conversion stops without converting a null wide character and `dst` is not a null pointer, then a null character is stored into the array pointed to by `dst` immediately following any multibyte characters already stored. Each conversion takes place as if by a call to the `wcrtomb` function.

If `dst` is not a null pointer, the pointer object pointed to by `src` is assigned either a null pointer (if conversion stopped due to reaching a terminating null wide character) or the address just past the last wide character converted (if any). If conversion stopped due to reaching a terminating null wide character, the resulting state described is the initial conversion state.

Regardless of whether `dst` is or is not a null pointer, if the input conversion encounters a wide character that does not correspond to a valid multibyte character, an encoding error occurs: the `wcsrtombs_s` function sets the `errno` global variable to indicate an error.
The wcsrtombs_s function stores the value (size_t)(-1) into *retval and the conversion state is unspecified. Otherwise, the wcsrtombs_s function stores into *retval the number of bytes in the resulting multibyte character sequence, not including the terminating null character (if any).

All elements following the terminating null character (if any) written by wcsrtombs_s in the array of dstmax elements pointed to by dst take unspecified values when wcsrtombs_s returns.

If copying takes place between objects that overlap, the objects take on unspecified values.

The _fwcsrtombs_s function is a data model independent form of the wcsrtombs_s function that accepts far pointer arguments. It is most useful in mixed memory model applications.

**Returns:** The wcsrtombs_s function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

**See Also:** _mbccmp, _mbccpy, _mbccicmp, _mbcjistojms, _mbcjstojis, _mbclen, _mbctohira, _mbctokata, _mbctowler, _mbctombb, _mbctoupper, _mblen, _mbrlen, _mbrtowc, _mbsrtowcs, _mbsrtowcs_s, _mbsrtowcs, _mbsrtowcs, _mbtowc, _mbtowc, _wcrtomb, _wcrtomb_s, _wcsrtombs, _wcsrtombs, _wcstombs, _wcstombs_s, _wctob, _wctomb, _wctomb_s

**Example:**
```c
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <wchar.h>
#include <mbctype.h>
#include <errno.h>

const wchar_t wc[] = {
    0x0020,              /* double-byte space */
    0x002e,              /* double-byte A */
    0x0031,              /* double-byte Hiragana */
    0x0041,              /* double-byte Katakana */
    0x3000,              /* double-byte space */
    0xff21,              /* double-byte A */
    0x3048,              /* double-byte Hiragana */
    0x30a3,              /* double-byte Katakana */
    0xff61,              /* single-byte Katakana punctuation */
    0xff66,              /* single-byte Katakana alphabetic */
    0xff9f,              /* single-byte Katakana alphabetic */
    0x720d,              /* double-byte Kanji */
    0x0000
};

int main()
{
    int i;
    size_t retval;
    const wchar_t *src;
    char mb[50];
    mbstate_t pstate;
    errno_t rc;

    _setmbcp( 932 );
    src = wc;
    rc = wcsrtombs_s( &retval, mb, 50, &src, sizeof(wc), &pstate );
    if( rc != 0 ) {
        printf( "Error in wide character string\n" );
    } else {
        for( i = 0; i < retval; i++ ) {
            printf( "0x%2.2x\n", mb[i] );
        }
    }
    return( rc );
}
```

produces the following:
wcsrtombs_s, _fwcsrtombs_s

0x20
0x2e
0x31
0x41
0x81
0x40
0x82
0x60
0x82
0xa6
0x83
0x42
0xa1
0xa6
0xdf
0xe0
0xa1

Classification: wcsrtombs_s is TR 24731
/fwcsrtombs_s is WATCOM

Systems: wcsrtombs_s - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
/fwcsrtombs_s - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
**wcestombs, _fwcestombs**

**Synopsis:**
```c
#include <stdlib.h>
size_t wcestombs( char *s, const wchar_t *pwcs, size_t n );
#include <mbstring.h>
size_t _fwcestombs( char __far *s, const wchar_t __far *pwcs, size_t n );
```

**Safer C:** The Safer C Library extension provides the wcestombs_s function which is a safer alternative to wcestombs. This newer wcestombs_s function is recommended to be used instead of the traditional "unsafe" wcestombs function.

**Description:** The wcestombs function converts a sequence of wide character codes from the array pointed to by `pwcs` into a sequence of multibyte characters and stores them in the array pointed to by `s`. The wcestombs function stops if a multibyte character would exceed the limit of `n` total bytes, or if the null character is stored. At most `n` bytes of the array pointed to by `s` will be modified.

The _fwcestombs function is a data model independent form of the wcestombs function that accepts far pointer arguments. It is most useful in mixed memory model applications.

**Returns:** If an invalid multibyte character is encountered, the wcestombs function returns (size_t)-1. Otherwise, the wcestombs function returns the number of array elements modified, not including the terminating zero code if present.

**See Also:** wcestombs_s, mblen, mbtowc, mbstowcs, mbstowcs_s, wctomb, wctomb_s

**Example:**
```c
#include <stdio.h>
#include <stdlib.h>

wchar_t wbuffer[] = {
  0x0073,
  0x0074,
  0x0072,
  0x0069,
  0x006e,
  0x0067,
  0x0000
};

void main()
{
  char    mbsbuffer[50];
  int     i, len;

  len = wcestombs( mbsbuffer, wbuffer, 50 );
  if( len != -1 ) {
    for( i = 0; i < len; i++ )
      printf( "/%4.4x", wbuffer[i] );
    printf( "\n" );
    mbsbuffer[len] = '\0';
    printf( "%s(%d)\n", mbsbuffer, len );
  }
}
```

produces the following:
/0073/0074/0072/0069/006e/0067
string(6)

**Classification:**
- `wcstombs` is ANSI
- `_fwcstombs` is not ANSI

**Systems:**
- `wcstombs` - All, Netware
- `_fwcstombs` - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32

*Library Functions and Macros 1101*
Synopsis:
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
errno_t wcstombs_s( size_t * restrict retval,
char * restrict dst,
rsize_t dstmax,
const wchar_t * restrict src,
rsize_t len);

errno_t _fwcstombs_s( size_t __far * restrict retval,
char __far * restrict dst,
rsize_t dstmax,
const wchar_t __far * restrict src,
rsize_t len);

Constraints:
If any of the following runtime-constraints is violated, the currently active runtime-constraint handler
will be invoked and wcstombs_s will return a non-zero value to indicate an error, or the
runtime-constraint handler aborts the program.

Neither retval nor src shall be a null pointer. If dst is not a null pointer, then neither len nor dstmax
shall be greater than RSIZE_MAX. If dst is a null pointer, then dstmax shall equal zero. If dst is not a
null pointer, then dstmax shall not equal zero. If dst is not a null pointer and len is not less than dstmax,
then the conversion shall have been stopped (see below) because a terminating null wide character was
reached or because an encoding error occurred.

If there is a runtime-constraint violation, then wcstombs_s does the following. If retval is not a null
pointer, then wcstombs_s sets *retval to (size_t)(-1). If dst is not a null pointer and dstmax is greater
than zero and less than RSIZE_MAX, then wcstombs_s sets dst[0] to the null character.

Description:
The wcstombs_s function converts a sequence of wide characters from the array pointed to by src
into a sequence of corresponding multibyte characters that begins in the initial shift state. If dst is not a
null pointer, the converted characters are then stored into the array pointed to by dst. Conversion
continues up to and including a terminating null wide character, which is also stored.

Conversion stops earlier in two cases:
when a wide character is reached that does not correspond to a valid multibyte character;
(if dst is not a null pointer) when the next multibyte character would exceed the limit of n total bytes to
be stored into the array pointed to by dst. If the wide character being converted is the null wide
character, then n is the lesser of len or dstmax. Otherwise, n is the lesser of len or dstmax-1.

If the conversion stops without converting a null wide character and dst is not a null pointer, then a null
character is stored into the array pointed to by dst immediately following any multibyte characters
already stored. Each conversion takes place as if by a call to the wcrtomb function.

Regardless of whether dst is or is not a null pointer, if the input conversion encounters a wide character
that does not correspond to a valid multibyte character, an encoding error occurs: the wcstombs_s
function stores the value (size_t)(-1) into *retval. Otherwise, the wcstombs_s function stores into
*retval the number of bytes in the resulting multibyte character sequence, not including the terminating
null character (if any).

All elements following the terminating null character (if any) written by wcstombs_s in the array of
dstmax elements pointed to by dst take unspecified values when wcstombs_s returns.

If copying takes place between objects that overlap, the objects take on unspecified values.
The _fwcstombs_s function is a data model independent form of the wcstombs_s function that accepts far pointer arguments. It is most useful in mixed memory model applications.

**Returns:**
The wcstombs_s function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

**See Also:**
wcestombs, mblen, mbtowc, mbstowcs, mbstowcs_s, wctomb, wctomb_s

**Example:**
```c
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <stdlib.h>

wchar_t wbuffer[] = {
0x0073,
0x0074,
0x0072,
0x0069,
0x006e,
0x0067,
0x0073,
0x0074,
0x0072,
0x0069,
0x006e,
0x0067,
0x0000
};

int main()
{
    char    mbsbuffer[50];
    int     i;
    size_t  retval;
    errno_t rc;

    rc = wcstombs_s( &retval, mbsbuffer, 50, wbuffer, sizeof( wbuffer ) );
    if( rc == 0 ) {
        for( i = 0; i < retval; i++ )
            printf( "/%4.4x", wbuffer[i] );
        printf( "\n" );
        mbsbuffer[retval] = '\0';
        printf( "%s(%d)\n", mbsbuffer, retval );
    }
    return( rc );
}
```
produces the following:
```
/0073/0074/0072/0069/006e/0067
string(6)
```

**Classification:**
wcestombs_s is TR 24731
_fwcstombs_s is WATCOM

**Systems:**
wcestombs_s - All, Netware
wcstombs_s, _fwcstombs_s

_fwcstombs_s - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  
#include <wchar.h>  
int wctob( wint_t wc );

Description: The wctob function determines whether wc corresponds to a member of the extended character set whose multibyte character representation is as a single byte when in the initial shift state.

Returns: The wctob function returns EOF if wc does not correspond to a multibyte character with length one; otherwise, it returns the single byte representation.

See Also: _mbccmp, _mbccpy, _mbccicmp, _mbcjmstojms, _mbcjstojis, _mbcslen, _mbcsctohira, _mbctokata, _mbctolower, _mbctoupper, mblen, mbrlen, mbtowc, mbrtowcs, mbrtowcs_s, mbstowcs, mbstowcs_s, mbtowc, btowc, wcrtomb, wcrtomb_s, wcsrtombs, wcsrtombs_s, wcstombs, wcstombs_s, wctomb, wctomb_s

Example:  
#include <stdio.h>  
#include <wchar.h>  
#include <mbctype.h>  
const wint_t wc[] = {  
0x0020,  
0x002e,  
0x0031,  
0x0041,  
0x3000, /* double-byte space */  
0xff21, /* double-byte A */  
0x3048, /* double-byte Hiragana */  
0x30a3, /* double-byte Katakana */  
0xff61, /* single-byte Katakana punctuation */  
0xff66, /* single-byte Katakana alphabetic */  
0xff9f, /* single-byte Katakana alphabetic */  
0x720d, /* double-byte Kanji */  
0x0000  
};  
#define SIZE sizeof( wc ) / sizeof( wint_t )  
void main()  
{  
    int i, j;  
    _setmbcp( 932 );  
    for( i = 0; i < SIZE; i++ ) {  
        j = wctob( wc[i] );  
        if( j == EOF ) {  
            printf( "%#6.4x EOF\n", wc[i] );  
        } else {  
            printf( "%#6.4x->%#6.4x\n", wc[i], j );  
        }  
    }  
}  

produces the following:
wctob

0x0020 -> 0x0020
0x002e -> 0x002e
0x0031 -> 0x0031
0x0041 -> 0x0041
0x3000 EOF
0xff21 EOF
0x3048 EOF
0x30a3 EOF
0xff61 -> 0x00a1
0xff66 -> 0x00a6
0xff9f -> 0x00df
0x720d EOF
0000 -> 0x0000

**Classification:** ANSI

**Systems:** DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:

```c
#include <stdlib.h>
int wctomb( char *s, wchar_t wc );
#include <mbstring.h>
int _fwctomb( char __far *s, wchar_t wc );
```

Safer C: The Safer C Library extension provides the wctomb_s function which is a safer alternative to wctomb. This newer wctomb_s function is recommended to be used instead of the traditional "unsafe" wctomb function.

Description: The wctomb function determines the number of bytes required to represent the multibyte character corresponding to the wide character contained in wc. If s is not a NULL pointer, the multibyte character representation is stored in the array pointed to by s. At most MB_CUR_MAX characters will be stored.

The _fwctomb function is a data model independent form of the wctomb function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: If s is a NULL pointer, the wctomb function returns zero if multibyte character encodings are not state dependent, and non-zero otherwise. If s is not a NULL pointer, the wctomb function returns:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>if the value of wc does not correspond to a valid multibyte character</td>
</tr>
<tr>
<td>len</td>
<td>the number of bytes that comprise the multibyte character corresponding to the value of wc.</td>
</tr>
</tbody>
</table>

See Also: wctomb_s, mblen, mbstowcs, mbstowcs_s, mbtowc, wcstombs, wcstombs_s

Example:

```c
#include <stdio.h>
#include <stdlib.h>

wchar_t wchar = { 0x0073 };
char    mbbuffer[2];

void main()
{
    int len;
    printf( "Character encodings are %s state dependent\n",
            ( wctomb( NULL, 0 )
                      ? "" : "not " ));

    len = wctomb( mbbuffer, wchar );
    mbbuffer[len] = '\0';
    printf( "%s(%d)\n", mbbuffer, len );
}
```

produces the following:

Character encodings are not state dependent
s(1)

Classification: wctomb is ANSI
_fwctomb is not ANSI
\texttt{wctomb, \_fwctomb}

\textbf{Systems:}
\texttt{wctomb} - All, Netware
\texttt{\_fwctomb} - DOS, Windows, Win386, Win32, OS/2 1.x\texttt{all}, OS/2-32
Synopsis: #define __STDC_WANT_LIB_EXT1__ 1
#include <stdlib.h>
errno_t wctomb_s( int * restrict status,
               char * restrict s,
               rsize_t smax,
               wchar_t wc);
errno_t _fwctomb_s( int __far * restrict status,
               char __far * restrict s,
               rsize_t smax,
               wchar_t wc);

Constraints: If any of the following runtime-constraints is violated, the currently active runtime-constraint handler will be invoked and wctomb_s will return a non-zero value to indicate an error, or the runtime-constraint handler aborts the program.

Let $n$ denote the number of bytes needed to represent the multibyte character corresponding to the wide character given by $wc$ (including any shift sequences).

If $s$ is not a null pointer, then $smax$ shall not be less than $n$, and $smax$ shall not be greater than RSIZE_MAX. If $s$ is a null pointer, then $smax$ shall equal zero.

If there is a runtime-constraint violation, wctomb_s does not modify the int pointed to by $status$, and if $s$ is not a null pointer, no more than $smax$ elements in the array pointed to by $s$ will be accessed.

Description: The wctomb_s function determines $n$ and stores the multibyte character representation of $wc$ in the array whose first element is pointed to by $s$ (if $s$ is not a null pointer). The number of characters stored never exceeds MB_CUR_MAX or $smax$. If $wc$ is a null wide character, a null byte is stored, preceded by any shift sequence needed to restore the initial shift state, and the function is left in the initial conversion state.

The implementation shall behave as if no library function calls the wctomb_s function.

If $s$ is a null pointer, the wctomb_s function stores into the int pointed to by $status$ a nonzero or zero value, if multibyte character encodings, respectively, do or do not have state-dependent encodings.

If $s$ is not a null pointer, the wctomb_s function stores into the int pointed to by $status$ either $n$ or -$l$ if $wc$, respectively, does or does not correspond to a valid multibyte character.

In no case will the int pointed to by $status$ be set to a value greater than the MB_CUR_MAX macro.

The _fwctomb_s function is a data model independent form of the wctomb_s function that accepts far pointer arguments. It is most useful in mixed memory model applications.

Returns: The wctomb_s function returns zero if there was no runtime-constraint violation. Otherwise, a non-zero value is returned.

See Also: wctomb, mblen, mbstowcs, mbstowcs_s, mbtowc, wcstombs, wcstombs_s
Example:

```c
#define __STDC_WANT_LIB_EXT1__ 1
#include <stdio.h>
#include <stdlib.h>

wchar_t wchar = { 0x0073 };
char mbbuffer[3];

int main()
{
    int len;
    int status;
    errno_t rc;
    rc = wctomb_s( &status, NULL, 0, wchar );
    printf( "Character encodings are %sstate dependent\n", ( status ) ? "" : "not " );

    rc = wctomb_s( &len, mbbuffer, 2, wchar );
    if( rc != 0 ) {
        printf( "Character encoding error\n" );
    } else {
        mbbuffer[len] = '\0';
        printf( "%s(%d)\n", mbbuffer, len );
    }
    return( rc );
}
```

produces the following:

Character encodings are not state dependent
s(1)

Classification: wctomb_s is TR 24731
fwctomb_s is WATCOM

Systems: wctomb_s - All, Netware
fwctomb_s - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
Synopsis:  #include <wctype.h>
    wctrans_t wctrans( const char *property );

Description:  The `wctrans` function constructs a value with type `wctrans_t` that describes a mapping between wide characters identified by the string argument `property`. The constructed value is affected by the `LC_CTYPE` category of the current locale; the constructed value becomes indeterminate if the category’s setting is changed.

The two strings listed below are valid in all locales as `property` arguments to the `wctrans` function.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>tolower</td>
<td>uppercase characters are mapped to lowercase</td>
</tr>
<tr>
<td>toupper</td>
<td>lowercase characters are mapped to uppercase</td>
</tr>
</tbody>
</table>

Returns:  If `property` identifies a valid class of wide characters according to the `LC_CTYPE` category of the current locale, the `wctrans` function returns a non-zero value that is valid as the second argument to the `towctrans` function; otherwise, it returns zero.

See Also:  `isalnum`, `isalpha`, `isblank`, `iscntrl`, `isdigit`, `isgraph`, `isleadbyte`, `islower`, `isprint`, `ispunct`, `isspace`, `isupper`, `iswctype`, `isxdigit`, `tolower`, `toupper`, `towctrans`

Example:  
```c
#include <stdio.h>
#include <wctype.h>

char *translations[2] = {
    "tolower",
    "toupper"
};

void main( void )
{
    int  i;
    wint_t wc = 'A';
    wint_t twc;

    for( i = 0; i < 2; i++ ) {
        twc = towctrans( wc, wctrans( translations[i] ) );
        printf( "%s(%lc): %lc\n", translations[i], wc, twc );
    }
}
```
produces the following:

tolower(A): a
toupper(A): A

Classification:  ANSI

Systems:  All, Netware
**Synopsis:**

```
#include <wctype.h>

wctype_t wctype( const char *property );
```

**Description:**
The `wctype` function constructs a value with type `wctype_t` that describes a class of wide characters identified by the string argument, `property`. The constructed value is affected by the `LC_CTYPE` category of the current locale; the constructed value becomes indeterminate if the category's setting is changed.

The twelve strings listed below are valid in all locales as `property` arguments to the `wctype` function.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>alnum</td>
<td>any wide character for which one of <code>iswalpha</code> or <code>исwdigit</code> is true</td>
</tr>
<tr>
<td>alpha</td>
<td>any wide character for which <code>исwupper</code> or <code>исwlower</code> is true, that is, for any wide character that is one of an implementation-defined set for which none of <code>исwcntrl</code>, <code>исwdigit</code>, <code>исwpunct</code>, or <code>исwspace</code> is true</td>
</tr>
<tr>
<td>blank</td>
<td>any wide character corresponding to a standard blank character (space or horizontal tab) or is one of an implementation-defined set of wide characters for which <code>исwblank</code> is true</td>
</tr>
<tr>
<td>cntrl</td>
<td>any control wide character</td>
</tr>
<tr>
<td>digit</td>
<td>any wide character corresponding to a decimal-digit character</td>
</tr>
<tr>
<td>graph</td>
<td>any printable wide character except a space wide character</td>
</tr>
<tr>
<td>lower</td>
<td>any wide character corresponding to a lowercase letter, or one of an implementation-defined set of wide characters for which none of <code>исwcntrl</code>, <code>исwdigit</code>, <code>исwpunct</code>, or <code>исwspace</code> is true</td>
</tr>
<tr>
<td>print</td>
<td>any printable wide character including a space wide character</td>
</tr>
<tr>
<td>punct</td>
<td>any printable wide character that is not a space wide character or a wide character for which <code>исwalnum</code> is true</td>
</tr>
<tr>
<td>space</td>
<td>any wide character corresponding to a standard white-space character or is one of an implementation-defined set of wide characters for which <code>исwalnum</code> is false</td>
</tr>
<tr>
<td>upper</td>
<td>any wide character corresponding to a uppercase letter, or if <code>c</code> is one of an implementation-defined set of wide characters for which none of <code>исwcntrl</code>, <code>исwdigit</code>, <code>исwpunct</code>, or <code>исwspace</code> is true</td>
</tr>
<tr>
<td>xdigit</td>
<td>any wide character corresponding to a hexadecimal digit character</td>
</tr>
</tbody>
</table>

**Returns:**
If `property` identifies a valid class of wide characters according to the `LC_CTYPE` category of the current locale, the `wctype` function returns a non-zero value that is valid as the second argument to the `iswctype` function; otherwise, it returns zero.

**See Also:**
isalnum, isalpha, isblank, iscntrl,isdigit,isgraph,isleadbyte,islower,isprint,ispunct,isspace,isupper,iswctype,isxdigit,tolower,toupper, towctrans
Example:

```c
#include <stdio.h>
#include <wchar.h>

char *types[] = {
    "alnum",
    "blank",
    "alpha",
    "cntrl",
    "digit",
    "graph",
    "lower",
    "print",
    "punct",
    "space",
    "upper",
    "xdigit"
};

void main( void )
{
    int     i;
    wint_t  wc = 'A';

    for( i = 0; i < 12; i++ )
        if( iswctype( wc, wctype( types[i] ) ) )
            printf( "%s
", types[i] );
}
```

produces the following:

- alnum
- alpha
- graph
- print
- upper
- xdigit

Classification: ANSI

Systems: All
Synopsis:  
#include <graph.h>
short _FAR _wrapon( short wrap );

Description:  The _wrapon function is used to control the display of text when the text output reaches the right side of the text window. This is text displayed with the _outtext and _outmem functions. The wrap argument can take one of the following values:

_wrapon( _GWRAPON )  causes lines to wrap at the window border
_wrapon( _GWRAPOFF )  causes lines to be truncated at the window border

Returns:  The _wrapon function returns the previous setting for wrapping.

See Also:  _outtext, _outmem, _settextwindow

Example:  
#include <conio.h>
#include <graph.h>
#include <stdio.h>

main()
{
    int i;
    char buf[ 80 ];

    _setvideomode( _TEXTC80 );
    _settextwindow( 5, 20, 20, 30 );
    _wrapon( _GWRAPOFF );
    for( i = 1; i <= 3; ++i ) {
        _settextposition( 2 * i, 1 );
        sprintf( buf, "Very very long line %d", i );
        _outtext( buf );
    }
    _wrapon( _GWRAPON );
    for( i = 4; i <= 6; ++i ) {
        _settextposition( 2 * i, 1 );
        sprintf( buf, "Very very long line %d", i );
        _outtext( buf );
    }
    getch();
    _setvideomode( _DEFAULTMODE );
}

Classification: _wrapon is PC Graphics

Systems:  DOS, QNX
Synopsis:
#include <io.h>
int write( int handle, void *buffer, unsigned len );
int _write( int handle, void *buffer, unsigned len );

Description: The write function writes data at the operating system level. The number of bytes transmitted is given by len and the data to be transmitted is located at the address specified by buffer.

The _write function is identical to write. Use _write for ANSI/ISO naming conventions.

The handle value is returned by the open function. The access mode must have included either O_WRONLY or O_RDWR when the open function was invoked.

The data is written to the file at the end when the file was opened with O_APPEND included as part of the access mode; otherwise, it is written at the current file position for the file in question. This file position can be determined with the tell function and can be set with the lseek function.

When O_BINARY is included in the access mode, the data is transmitted unchanged. When O_TEXT is included in the access mode, the data is transmitted with extra carriage return characters inserted before each linefeed character encountered in the original data.

A file can be truncated under DOS and OS/2 2.0 by specifying 0 as the len argument. Note, however, that this doesn’t work under OS/2 2.1, Windows NT/2000, and other operating systems. To truncate a file in a portable manner, use the chsize function.

Returns: The write function returns the number of bytes (does not include any extra carriage-return characters transmitted) of data transmitted to the file. When there is no error, this is the number given by the len argument. In the case of an error, such as there being no space available to contain the file data, the return value will be less than the number of bytes transmitted. A value of -1 may be returned in the case of some output errors. When an error has occurred, errno contains a value indicating the type of error that has been detected.

See Also: chsize, close, creat, dup, dup2, eof, exec..., fdopen, filelength, fileno, fstat, _grow_handles, isatty, lseek, open, read, setmode, sopen, stat, tell, umask

Example:
#include <stdio.h>
#include <io.h>
#include <fcntl.h>

char buffer[] = { "A text record to be written" };

void main( void )
{
    int handle;
    int size_written;
    /* open a file for output */
    /* replace existing file if it exists */
    handle = open( "file",
                   O_WRONLY | O_CREAT | O_TRUNC | O_TEXT,
                   S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP );
    if( handle != -1 ) {

Library Functions and Macros 1115
write, _write

/* write the text */
size_written = write( handle, buffer,
               sizeof( buffer ) );

/* test for error */
if( size_written != sizeof( buffer ) ) {
    printf( "Error writing file\n" );
}

/* close the file */
close( handle );

Classification: write is POSIX 1003.1
_write is not POSIX
_write conforms to ANSI/ISO naming conventions

Systems: write - All, Netware
_write - DOS, Windows, Win386, Win32, OS/2 1.x(all), OS/2-32
5 Re-entrant Functions

The following functions in the C library are re-entrant:

```
abs atoi atol bsearch bsearch_s div fabs _fmbstowcs_s _fmbsrtowcs_s
_fmemccpy _fmemchr _fmemcmp _fmemcpy
_fmemmmove _fmemset _fstrcat
_fstrchr _fstrcmp _fstrcpy _fstrcsppn
_fstricmp _fstricpy _fstricmp _fstricset
_fstrncmp _fstrncpy _fstrnicmp _fstrnicset
_fstrpbrk _fstrrchr _fstrrev _fstrset
_fstrspn _fstrstr _fstrupr _fwctomb_s
_fwcsrtombs_s _fwcsstombs_s _fwctomb_s isalnum
isalpha isascii isblank iscntrl
isdigit isgraph islower isprint
ispunct isspace isupper isxdigit
itoa labs ldiv lfind
longjmp _lrotl _lrotl _lrotl _lrotl
ltoa _makepath mblen mbsrtowcs_s
mbstowcs mbstowcs_s mbtowc memccpy
memchr memcmp memmove memmove_s memset
memcpy memmove memmove_s memset
movedata qsort qsort_s _rotr
_rotl segread setjmp _splitpath
strcat strcat_s strchr strcmp
strcoll strcpy strcpys strcsppn
strerror_s strlen
strerrorlen_s stricmp strlen
strlen
strlwr strncat strncat_s strncat_s
strncpy strncpy_s strnicmp strnlen_s
strndup strpbrk strstr strrev
strset strspn strstr strtok_s
strupr swab tolower toupper
ultoa utoa wcrtombs_s wcscat_s
wcscpy_s wcscsppn wcscsppn wcscsppn
wcscat_s wcscat_s wcscat_s wcscat_s
wcstok_s wcstombs wcstombs wcstombs
wctomb_s wmemccpy_s wmemmove_s
```
1118 Re-entrant Functions
Appendices
This appendix describes the behavior of the 16-bit and 32-bit Watcom C libraries when the ANSI/ISO C Language standard describes the behavior as implementation-defined. The term describing each behavior is taken directly from the ANSI/ISO C Language standard. The numbers in parentheses at the end of each term refers to the section of the standard that discusses the behavior.

A.1 NULL Macro

The null pointer constant to which the macro NULL expands (7.1.6).

The macro NULL expands to 0 in small data models and to 0L in large data models.

A.2 Diagnostic Printed by the assert Function

The diagnostic printed by and the termination behavior of the assert function (7.2).

The assert function prints a diagnostic message to stderr and calls the abort routine if the expression is false. The diagnostic message has the following form:

Assertion failed: [expression], file [name], line [number]

A.3 Character Testing

The sets of characters tested for by the isalnum, isalpha, iscntrl, islower, isprint, and isupper functions (7.3.1).

<table>
<thead>
<tr>
<th>Function</th>
<th>Characters Tested For</th>
</tr>
</thead>
<tbody>
<tr>
<td>isalnum</td>
<td>Characters 0-9, A-Z, a-z</td>
</tr>
<tr>
<td>isalpha</td>
<td>Characters A-Z, a-z</td>
</tr>
<tr>
<td>iscntrl</td>
<td>ASCII 0x00-0x1f, 0x7f</td>
</tr>
<tr>
<td>islower</td>
<td>Characters a-z</td>
</tr>
<tr>
<td>isprint</td>
<td>ASCII 0x20-0x7e</td>
</tr>
<tr>
<td>isupper</td>
<td>Characters A-Z</td>
</tr>
</tbody>
</table>
A.4 Domain Errors

The values returned by the mathematics functions on domain errors (7.5.1).

When a domain error occurs, the listed values are returned by the following functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Value returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>acos</td>
<td>0.0</td>
</tr>
<tr>
<td>acosh</td>
<td>- HUGE_VAL</td>
</tr>
<tr>
<td>asin</td>
<td>0.0</td>
</tr>
<tr>
<td>atan2</td>
<td>0.0</td>
</tr>
<tr>
<td>atanh</td>
<td>- HUGE_VAL</td>
</tr>
<tr>
<td>log</td>
<td>- HUGE_VAL</td>
</tr>
<tr>
<td>log10</td>
<td>- HUGE_VAL</td>
</tr>
<tr>
<td>log2</td>
<td>- HUGE_VAL</td>
</tr>
<tr>
<td>pow(neg, frac)</td>
<td>0.0</td>
</tr>
<tr>
<td>pow(0.0, 0.0)</td>
<td>1.0</td>
</tr>
<tr>
<td>pow(0.0, neg)</td>
<td>- HUGE_VAL</td>
</tr>
<tr>
<td>sqrt</td>
<td>0.0</td>
</tr>
<tr>
<td>y0</td>
<td>- HUGE_VAL</td>
</tr>
<tr>
<td>y1</td>
<td>- HUGE_VAL</td>
</tr>
<tr>
<td>yn</td>
<td>- HUGE_VAL</td>
</tr>
</tbody>
</table>

A.5 Underflow of Floating-Point Values

Whether the mathematics functions set the integer expression errno to the value of the macro ERANGE on underflow range errors (7.5.1).

The integer expression errno is not set to ERANGE on underflow range errors in the mathematics functions.

A.6 The fmod Function

Whether a domain error occurs or zero is returned when the fmod function has a second argument of zero (7.5.6.4).

Zero is returned when the second argument to fmod is zero.

A.7 The signal Function

The set of signals for the signal function (7.7.1.1).

See the description of the signal function presented earlier in this book.

The semantics for each signal recognized by the signal function (7.7.1.1).

See the description of the signal function presented earlier in this book.
The default handling and the handling at program startup for each signal recognized by the `signal` function (7.7.1.1).

See the description of the `signal` function presented earlier in this book.

### A.8 Default Signals

If the equivalent of `signal(sig, SIG_DFL)` is not executed prior to the call of a signal handler, the blocking of the signal that is performed (7.7.1.1).

The equivalent of

```c
signal( sig, SIG_DFL );
```

is executed prior to the call of a signal handler.

### A.9 The SIGILL Signal

Whether the default handling is reset if the SIGILL signal is received by a handler specified to the `signal` function (7.7.1.1).

The equivalent of

```c
signal( SIGILL, SIG_DFL );
```

is executed prior to the call of the signal handler.

### A.10 Terminating Newline Characters

Whether the last line of a text stream requires a terminating new-line character (7.9.2).

The last line of a text stream does not require a terminating new-line character.

### A.11 Space Characters

Whether space characters that are written out to a text stream immediately before a new-line character appear when read in (7.9.2).

All characters written out to a text stream will appear when read in.
Appendices

A.12 Null Characters

The number of null characters that may be appended to data written to a binary stream (7.9.2).

No null characters are appended to data written to a binary stream.

A.13 File Position in Append Mode

Whether the file position indicator of an append mode stream is initially positioned at the beginning or end of the file (7.9.3).

When a file is open in append mode, the file position indicator initially points to the end of the file.

A.14 Truncation of Text Files

Whether a write on a text stream causes the associated file to be truncated beyond that point (7.9.3).

Writing to a text stream does not truncate the file beyond that point.

A.15 File Buffering

The characteristics of file buffering (7.9.3).

Disk files accessed through the standard I/O functions are fully buffered. The default buffer size is 512 bytes for 16-bit systems, and 4096 bytes for 32-bit systems.

A.16 Zero-Length Files

Whether a zero-length file actually exists (7.9.3).

A file with length zero can exist.

A.17 File Names

The rules of composing valid file names (7.9.3).

A valid file specification consists of an optional drive letter (which is always followed by a colon), a series of optional directory names separated by backslashes, and a file name.

FAT File System: Directory names and file names can contain up to eight characters followed optionally by a period and a three letter extension. The complete path (including drive, directories and file name) cannot exceed 143 characters. Case is ignored (lowercase letters are converted to uppercase letters).

HPFS File System: Directory names and file names can contain up to 254 characters in the OS/2 High Performance File System (HPFS). However, the complete path (including drive, directories and file name)
Implementation-Defined Behavior of the C Library

cannot exceed 259 characters. The period is a valid file name character and can appear in a file name or
directory name as many times as required; HPFS file names do not require file extensions as in the FAT file
system. The HPFS preserves case in file names only in directory listings but ignores case in file searches
and other system operations (i.e., a directory cannot have more than one file whose names differ only in
case).

A.18 File Access Limits

Whether the same file can be open multiple times (7.9.3).

It is possible to open a file multiple times.

A.19 Deleting Open Files

The effect of the remove function on an open file (7.9.4.1).

The remove function deletes a file, even if the file is open.

A.20 Renaming with a Name that Exists

The effect if a file with the new name exists prior to a call to the rename function (7.9.4.2).

The rename function will fail if you attempt to rename a file using a name that exists.

A.21 Printing Pointer Values

The output for %p conversion in the fprintf function (7.9.6.1).

Two types of pointers are supported: near pointers (%hp), and far pointers (%lp). The output for %p
depends on the memory model being used.

In 16-bit mode, the fprintf function produces hexadecimal values of the form XXXX for 16-bit near
pointers, and XXXX:XXXX (segment and offset separated by a colon) for 32-bit far pointers.

In 32-bit mode, the fprintf function produces hexadecimal values of the form XXXXXXXX for 32-bit
near pointers, and XXXX:XXXXXXXX (segment and offset separated by a colon) for 48-bit far pointers.

A.22 Reading Pointer Values

The input for %p conversion in the fscanf function (7.9.6.2).

The fscanf function converts hexadecimal values into the correct address when the %p format specifier
is used.
Appendices

A.23 Reading Ranges

The interpretation of a – character that is neither the first nor the last character in the scanlist for %conversion in the fscanf function (7.9.6.2).

The "-" character indicates a character range. The character prior to the "-" is the first character in the range. The character following the "-" is the last character in the range.

A.24 File Position Errors

The value to which the macro errno is set by the fgetpos or ftell function on failure (7.9.9.1, 7.9.9.4).

When the function fgetpos or ftell fails, they set errno to EBADF if the file number is bad. The constants are defined in the <errno.h> header file.

A.25 Messages Generated by the perror Function

The messages generated by the perror function (7.9.10.4).

The perror function generates the following messages.

<table>
<thead>
<tr>
<th>Error</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&quot;Error 0&quot;</td>
</tr>
<tr>
<td>1</td>
<td>&quot;No such file or directory&quot;</td>
</tr>
<tr>
<td>2</td>
<td>&quot;Argument list too big&quot;</td>
</tr>
<tr>
<td>3</td>
<td>&quot;Exec format error&quot;</td>
</tr>
<tr>
<td>4</td>
<td>&quot;Bad file number&quot;</td>
</tr>
<tr>
<td>5</td>
<td>&quot;Not enough memory&quot;</td>
</tr>
<tr>
<td>6</td>
<td>&quot;Permission denied&quot;</td>
</tr>
<tr>
<td>7</td>
<td>&quot;File exists&quot;</td>
</tr>
<tr>
<td>8</td>
<td>&quot;Cross-device link&quot;</td>
</tr>
<tr>
<td>9</td>
<td>&quot;Invalid argument&quot;</td>
</tr>
<tr>
<td>10</td>
<td>&quot;File table overflow&quot;</td>
</tr>
<tr>
<td>11</td>
<td>&quot;Too many open files&quot;</td>
</tr>
<tr>
<td>12</td>
<td>&quot;No space left on device&quot;</td>
</tr>
<tr>
<td>13</td>
<td>&quot;Argument too large&quot;</td>
</tr>
<tr>
<td>14</td>
<td>&quot;Result too large&quot;</td>
</tr>
<tr>
<td>15</td>
<td>&quot;Resource deadlock would occur&quot;</td>
</tr>
</tbody>
</table>

A.26 Allocating Zero Memory

The behavior of the calloc, malloc, or realloc function if the size requested is zero (7.10.3).

The value returned will be NULL. No actual memory is allocated.

1126 Allocating Zero Memory
A.27 The abort Function

The behavior of the `abort` function with regard to open and temporary files (7.10.4.1).

The `abort` function does not close any files that are open or temporary, nor does it flush any output buffers.

A.28 The atexit Function

The status returned by the `exit` function if the value of the argument is other than zero, `EXIT_SUCCESS`, or `EXIT_FAILURE` (7.10.4.3).

The `exit` function returns the value of its argument to the operating system regardless of its value.

A.29 Environment Names

The set of environment names and the method for altering the environment list used by the `getenv` function (7.10.4.4).

The set of environment names is unlimited. Environment variables can be set from the DOS command line using the SET command. A program can modify its environment variables with the `putenv` function. Such modifications last only until the program terminates.

A.30 The system Function

The contents and mode of execution of the string by the `system` function (7.10.4.5).

The `system` function executes an internal DOS, Windows, or OS/2 command, or an EXE, COM, BAT or CMD file from within a C program rather than from the command line. The `system` function examines the `COMSPEC` environment variable to find the command interpreter and passes the argument string to the command interpreter.

A.31 The strerror Function

The contents of the error message strings returned by the `strerror` function (7.11.6.2).

The `strerror` function generates the following messages.

<table>
<thead>
<tr>
<th>Error</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&quot;Error 0&quot;</td>
</tr>
<tr>
<td>1</td>
<td>&quot;No such file or directory&quot;</td>
</tr>
<tr>
<td>2</td>
<td>&quot;Argument list too big&quot;</td>
</tr>
<tr>
<td>3</td>
<td>&quot;Exec format error&quot;</td>
</tr>
</tbody>
</table>
A.32 The Time Zone

The local time zone and Daylight Saving Time (7.12.1).

The default time zone is "Eastern Standard Time" (EST), and the corresponding daylight saving time zone is "Eastern Daylight Saving Time" (EDT).

A.33 The clock Function

The era for the clock function (7.12.2.1).

The clock function’s era begins with a value of 0 when the program starts to execute.
abs 56
access 57
__access 57
acos 59, 1122
acosh 60, 1122
actime 1043
alloca 61
__amblksiz 32, 810
ANALOGCOLOR 362
ANALOGMONO 362
ANSI classification 51
arc 42
__arc 62, 42, 315, 325, 756
__arc_w 62
__arc_wxy 62
__argc 32
__argv 32
__dos_getvect 180
__dos_setvect 196
int386 402
int386x 403
int86 405
int86x 406
intr 410

8086 Interrupts
__chain_intr 118
__dos_getvect 180
__dos_setvect 196
int386 402
int386x 403
int86 405
int86x 406
intr 410

BSWITCH...

BASE 854
basename 81
__bcalloc 114, 14, 93, 114
bcmp 88
bcopy 89
bdos 82, 50
__beginthread 83, 213
__beginthreaddx 83, 83-84, 213
bessel 87
__bexpand 223, 93, 223
__bfree 285, 14, 285
__bfreeseg 90
__bgetcmd 92
__bheapchk 382, 382
__bhealchk 382, 382

abort_handler_s 55
Index

_bheapmin 386, 386
_bheapseg 93, 90
_bheapset 387, 387
_bheapshrink 389, 389
_bheapwalk 390, 390
binary files 33
BINMODE.OBJ 33, 269, 272
BIOS classification 51
BIOS Functions 24
_bios_disk 95
_bios_equiplist 97
_bios_keybrd 98
_bios_memsize 100
_bios_printer 101
_bios_serialcom 102
_bios_timeofday 104
bios.h 28
_bios_disk 95
_bios_equiplist 97
_bios_keybrd 98
_bios_memsize 100
_bios_printer 101
_bios_serialcom 102
_bios_timeofday 104
_bmalloc 544, 14, 93, 544
_bmsize 697, 697
bool 30
BOTTOM 854
_bprintf 105, 1051
BREAK 106, 875
break_off 106, 106
break_on 106, 106
_brealloc 795, 14, 93, 795-796
bsearch 107
bsearch_s 109
btom 594
btowc 111
BUFSIZE 827
_bwprintf 105
bzero 112
c 293
cabs 113
calloc 114, 14, 114, 285, 697, 1126
CAP 854
ceil 116
CENTER 854
CGA 361, 865
cgets 117
_chain_intr 118
CHAR_MAX 514
Character Manipulation Functions 5-6
isalnum 411
isalpha 412
iscsii 413
_isascsi 413
isblank 416
iscntrl 418
iscsym 419
_iscsym 419
iscsymf 421
isdigit 423
isgraph 425
isleadbyte 427
islower 429
isprint 492
ispunct 493
isspace 495
isupper 497
iswalnum 411
iswalpha 412
iswascii 413
iswblank 416
iswcntrl 418
_iswcsym 419
_iswcsymf 421
iswdigit 423
iswgraph 425
iswlower 429
iswprint 492
iswpunct 493
iswspace 495
iswupper 497
iswxdigit 500
iswxdigitf 500
_mbctohira 566
_mbctokata 568
_mbctolower 562
_mbctoupper 564
tolower 1025
tolower 1025
toupper 1027
toupper 1027
towlower 1025
towupper 1027
wctype 1112
_chdir 119
_chdir 119
_chdrive 121
chkctype 550
chmod 122
_chmod 122

1130
Index

chsize 124, 531, 1115
  _chsize 124
classes of functions 40
  _clear87 125
clearenv 126
clearerr 127
  _clearscreen 128, 819
clock 129, 1128
  CLOCKS_PER_SEC 129
close 130, 346, 709
  _close 130
closedir 131, 705-706, 792
  _cmdname 133
  COLOR 362
  _COM_INIT 102
  _COM_RECEIVE 102
  _COM_SEND 102
  _COM_STATUS 102
  COMMODE.OBJ 270, 272
Comparison Functions
  bcmp 88
  _fmbscmp 926
  _fmbsicmcp 948
  _fmbsnbcmp 587
  _fmbsnbicmcp 592
  _fmbsncmp 966
  _fmbsnicmcp 973
  _fmemcmp 616
  _fmemicmp 619
  fstrcmp 926
  _fstricmp 948
  _fstrncmp 966
  _fstrnicmp 973
  _mbscmp 926
  _mbscoll 929
  _mbsicmcp 948
  _mbsicoll 950
  _mbsnbcmp 587
  _mbsnbicmcp 592
  _mbsncmp 966
  _mbsncoll 973
  _mbsnicmcp 973
  _mbsnicoll 975
  memcmp 616
  memicmp 619
  _memicmp 619
  strcasecmp 920
  strcmp 926
  strcmpl 928
  strcoll 929
  stricmp 948
  _stricmp 948
  _stricoll 950
  strncasecmp 961
  strcmp 926
  strcmpi 928
  strncmp 966
  _strncoll 968
  strnicmp 973
  _strnicmp 973
  _strncoll 975
  strxfrm 1010
  wcescmp 926
  wcescmpi 928
  wcscoll 929
  _wcesicmcp 948
  _wcesicoll 950
  wcsncmp 966
  _wcsvecoll 968
  _wcsnicmcp 973
  _wcsnicoll 975
  wmemcmp 616
  complex 30
  COMSPEC 126, 1012, 1127
  CON 48
Concatenation Functions
  _fmbscat 921
  _fmbsncat 962
  _fstrcat 921
  _fstrncat 962
  _mbscat 921
  _mbsncmp 966
  _mbsncat 962
  _mbsncat_s 964
  _mbsstrcat 921
  _mbsstrcat_s 923
  wcsstrcat 921
  wcsstrcat_s 923
  wcsstrcat_s 964
  wcstrcat 921
  wcstrcat_s 923
  wcstrcat_s 964
  cget 117
  cprintf 140
  cputc 141
  cscanf 144
  getch 319
  getche 321
  kbhit 503
  _kbhit 503
  putch 776
  stdin 18
  stdout 18
  ungetc 1039
  vscanf 1052
  wcscat 921
  wcscat_s 923
  wcscoll 929
  wcsicoll 950
  wcsnicoll 975
  wcsnicoll 975
  wcswcat 921
Index

const 51
__control87 134
__control8f 136
Conversion Functions 13
    atof 76
    atoi 77
    atol 78
    atoll 79
    ecvt 208
    __ecvt 208
    fcvt 228
    __fcvt 228
    gcvt 313
    __gcvt 313
    itoa 501
    __itoa 501
    __itow 501
    lltoa 534
    __lltoa 534
    __lltow 534
    ltod 536
    __ltod 536
    strdate 936
    __strdate 936
    strftime 995
    __strftime 995
    strtod 996
    __strtod 996
    strtoimax 1004
    __strtoimax 1004
    strtol 1002
    __strtol 1002
    strtol 1003
    __strtol 1003
    strtoul 1005
    __strtoul 1005
    strtoull 1006
    __strtoull 1006
    strtoumax 1007
    __strtoumax 1007
    wcstoumax 1007
    wctrans 1111
    __wecvt 208
    __wfenvt 228
    __wgcvt 313
    __wstrdate 936
    __wstrtime 995
    __wtof 76
    __wtoi 77
    __wtol 78
    __wtoll 79
Coordinate systems 41
Coordinated Universal Time 36-37
Copying Functions
    bcopy 89
    __fmbscopy 930
    __fmbscspn 934
    __fmbsdup 939
    __fmbssnbcpy 590
    __fmbssncpy 969
    __fnmempcy 617
    __fnmemmove 621
    __fnstrcp 930
    __fnstrdup 939
    __fnstrncpy 969
    __fnmbscp 930
    __fnmbscpn 934
    __fnmbsdup 939
    __fnmbsnbcpy 590
    __fnmbsncpy 969
    memempcy 617
    memempcy_s 618
    memmove 621
    memmove_s 622
    movedata 638
    strcp 930
    __strncpy 932
    __strdup 939
    __strdup 939
    __strcpcpy 955
    __strncpy 969
    __strncpy_s 971
    wcsncpy 930
    wcsncpy_s 932
    __wcscdup 939
    wcslepy 955
    __wcsncp 969
    __wcsncpy 971
    wmempcy 617
    __wmempcy_s 618
    wmempmove 621
    __wmempmove_s 622
    cos 138
    __cos 138
    __cos 138
    __cosh 139
    __cosh 139
cprintf 140, 1052
CPUID 624
cputs 141
creat 142, 33, 130, 230, 531, 1036
_create 142
CREATE_SUSPENDED 84
cscanf 144, 1053
cftime 145, 38, 64
_ctime 145, 38, 145
cftime_s 147
ctype.h 28
currency_symbol 513-514
current directory 119
current drive 119
current working directory 119
cwait 149, 896, 898, 1084

math.h 29
__MaxThreads 33
__minreal 33
mmintrin.h 30
optarg 33
opterr.h 30
optind 33
optopt 33
_osbuild 34
_osmajor 34
_osminor 34
_osmode 34
_osver 34
process.h 30
_psp 34
search.h 30
setjmp.h 30
share.h 30
signal.h 30
_stacksize 34
stdarg.h 30
stddef.h 30
stdio.h 30
stdlib.h 31
stdout 35
stdlib.h 31
stdout 35
string.h 31
sys\locking.h 31
sys\stat.h 31
sys\timeb.h 31
sys\types.h 31
sys\utime.h 32
sys\errno.h 31
sys\errlist 35
sys\nerr 35
_threads 35
time.h 31
timezone 35
tzname 35
varargs.h 31
__wargc 35
__wargv 35
wchar.h 31
wctype.h 31
_win_alloc_flags 36
_win_realloc_flags 36
_winmajor 36
_winminor 36
_winver 36

D

d_attr 705, 792
d_date 705, 792
d_time 705, 792
data
__amblksiz 32
__argc 32
__argv 32
assert.h 28
bios.h 28
conio.h 28
ctype.h 28
daylight 32
direct.h 28
dos.h 28
_doserrno 32
env.h 29
environ 32
errno 32
erro.h 29
fcntl.h 29
fenv.h 29
float.h 29
fused_ 32
_fmode 32
fnmatch.h 29
graph.h 29
inttypes.h 29
io.h 29
limits.h 29
locale.h 29
malloc.h 29

1133
Index

daylight 32, 37, 1030
default drive 119
Default Windowing Functions 24
DEFAULTMODE 865
delay 152
devices 47
__diseetomsbin 153
dflftime 154
DIR 28
direct.h 28
directory 47
Directory Functions 22
__bgetcmd 92
__chdir 119
__chdir 119
closedir 131
getcmd 323
getcwd 326
__getcwd 328
mkdir 629
__mkdir 629
opendir 705
__open 764
readdir 792
rewinddir 805
rmdir 807
__rmdir 807
__wchdir 119
__wclosedir 131
__wgetcwd 326
__wgetcwd 328
__wmkdir 629
__wmkdir 629
open 705
__popen 764
readdir 792
rewinddir 805
__wrmidir 807
direct 705, 792
dirname 155
__disable 156, 212
__DISK_FORMAT 95
__DISK_READ 95
__DISK_RESET 95
__DISK_STATUS 95
__DISK_VERIFY 95
__DISK_WRITE 95
diskfree_t 173, 330
diskinfo_t 95
__displaycursor 157
div 158
div_t 158
__dmsbintoieee 159
DOMAIN 546, 846
DOS
Program Segment Prefix 34
PSP 34
DOS classification 51
DOS command
CHDIR (CD) 48
date 36, 145, 368, 516, 1019, 1030
PATH 217, 897
SET 36, 217, 332-333, 778, 834, 897
time 36, 145, 368, 516, 1019, 1030
DOS commands 50
DOS considerations 47
DOS devices 47
DOS directory 47
DOS file 49
DOS Functions 25
__chdrive 121
chsize 124
__chsize 124
delay 152
__dos_allocmem 160
__dos_close 161
__dos_creat 163
__dos_creatnew 164
__dos_findclose 168
__dos_findfirst 168
__dos_findnext 168
__dos_freemem 171
__dos_getdate 172
__dos_getdiskfree 173
__dos_getdrive 174
__dos_getfileattr 175
__dos_gettime 177
__dos_gettime 179
__dos_open 182
__dos_read 184
__dos_setblock 185
__dos_setdate 187
__dos_setdrive 189
__dos_setfileattr 190
__dos_setftime 192
__dos_settime 194
__dos_write 197
__findclose 256
__findfirst 257
__findfirsti64 257
__findnext 259
__findnexti64 259
__getdiskfree 330
__getdrive 331
nosound 699
sleep 883
sound 894
swab 1011
__wdos_findclose 168
Index

_dos_findfirst 168
_dos_findnext 168
_wfindfirst 257
_wfindfirsti64 257
_wfindnext 259
_wfindnexti64 259
DOS I/O Functions 23
close 130
_close 130
creat 142
__creat 142
_dos_close 161
_dos_creat 163
_dos_creatnew 164
_dos_open 182
_dos_read 184
_dos_write 197
dup 198
dup2 200
_dup2 200
dup 198
eof 215
__eof 215
_fdopen 230
_filelength 253
__filelength 253
__filelengthi64 253
_fileno 255
_fstat 301
__fstat 301
__fstati64 301
_grow_handles 372
_isatty 415
__isatty 415
lock 520
__locking 521
_lseek 531
__lseek 531
__lseeki64 531
_open 702
__open 702
_read 790
__read 790
_setmode 849
__setmode 849
_sopen 890
__sopen 890
tell 1015
__tell 1015
__tell64 1015
_umask 1036
__umask 1036
unlock 1041
_utime 1043
__utime 1043
__wcreat 142
__wstat 301
__wstat64 301
__wopen 702
write 1115
__write 1115
__wsopen 890
__wutime 1043
DOS Interrupts 50
bdos 82
intdos 407
intdosx 408
DOS path 47
dos.h 28
__dos_allocmem 160, 171, 185
__dos_close 161
__dos_commit 162
__dos_creat 163
__dos_creatnew 164
__dos_find 168, 169
__dos_findclose 168, 169
__dos_findfirst 168, 168-169
__dos_findnext 168, 169
__dos_freemem 171
__dos_getdate 172
__dos_getdiskfree 173
__dos_getdrive 174, 189
__dos_getfileattr 175
__dos_gettime 177
__dos_gettime 179
__dos_getvext 180
__dos_keep 181
__dos_open 182
__dos_read 184
__dos_setblock 185
__dos_setdate 187
__dos_setdrive 189, 119
__dos_setfileattr 190
__dos_settime 192
__dos_settime 194
__dos_setopt 196
__dos_write 197
dosdate_t 172, 187
_doserrno 32
DOSErrOr 29
DOSEXIt 149, 898, 1084
dosexterr 166
dosetime_t 179, 194
dup 198, 130, 230, 531
dup2 200, 130, 230, 346, 531, 708
dup2 200
dup 198
1135
Index

_dwDeleteOnClose 202
_dwSetAboutDlg 203
_dwSetAppTitle 204
_dwSetConTitle 205
_dwShutDown 206
_dwYield 207

_E 215
EOF 215

E2BIG 217, 899
EACCES 57, 123-124, 143, 164, 182, 217, 521, 629, 704, 706, 892, 918, 1043
EAGAIN 84
EBADF 124, 130-131, 198, 200, 215, 303, 305, 346, 521, 532, 793, 1126
ECHILD 151, 719, 1085
ecvt 208, 228
_ecvt 208
EDEADLOCK 521
EDOM 59-60, 69, 73-74, 87, 523-525, 766, 910, 1002-1007
EEXIST 164, 629
EGA 361, 865
EILSEQ 248-249, 281, 318, 320, 576, 580, 598, 1087, 1093
EINTR 151, 719, 1085
EINVAL 84, 151, 173, 182, 194, 258, 305, 326, 330, 521, 532, 764, 875, 899, 1043
EIO 305
ellipse 42
_ellipse 210, 42, 335
_ellipse_w 210
_ellipse_wxy 210
EMFILE 143, 164, 182, 198, 200, 217, 704, 759, 892, 1043
_enable 212, 156
_endthread 213, 84
_endthreadex 213, 84, 213
ENFILE 759
ENHANCED 362
ENODEV 328
ENOENT 57, 119, 123, 143, 164, 182, 218, 256, 258, 260, 309, 629, 704, 706, 892, 899, 1043
ENOMEM 84, 126, 185, 218, 309, 326, 328, 779, 835, 899
ENOSPC 124, 759
ENOSYS 305
env.h 29
environ 32, 126, 834
environment 332-333, 778, 834
environment variable
tmpfile 1020
tmpfile_s 1021
_eof 215
ERANGE 139, 222, 309, 326, 328, 879, 997, 1002-1007, 1122
ERESCOLOR 865
ERESNOCOLOR 865
EROFS 759
erro.h 29
erro_t 823
Error Handling 29, 32
_clear87 125
clearerr 127
_control87 134
_controlfp 136
dosext 166
feof 238
ferror 240
_fpreset 277
matherr 546
perror 720
raise 787
_set_matherr 846
signal 874
_status87 919
stderr 18
strerror 940
strerror_s 941
wcserror 940
Index

wcerror_s 941
_wperror 720
exception 30
exec 216, 20-21, 30, 50, 810, 896, 1012
excl 216, 217
execl 216, 217
execlp 216, 216-217
execlpe 216, 216-217
exec 216, 217
execv 216, 217
execvp 216, 216-217
execvpe 216, 216-217
exit 221, 84, 149, 538, 874, 1084, 1127
_exit 220, 220, 84, 149, 1084
EXIT_FAILURE 54, 1127
EXIT_SUCCESS 1127

exp 222
__exp 223, 223
extern 29

F

F_OK 57
fabs 225
false 30
__fcallloc 114, 14, 114
fclose 226
fcloseall 227
fcntl 759
fcntl.h 29
fcvt 228, 208
__fcvt 228
fdopen 230, 227
__fdopen 230, 270
FE_ALL_EXCEPT 235
FE_DENORMAL 235
FE_DFL_ENV 241
FE_DIVBYZERO 235
FE_DOWNWARD 236
FE_INEXACT 235
FE_INVALID 235
FE_OVERFLOW 235
FE_TONEAREST 236
FE_TOWARDZERO 236
FE_UNDERFLOW 235
FE_UPWARD 236
fclearexcept 231
__fedisableexcept 232
__fenableexcept 233
fegetenv 234, 241
fegetexceptflag 235, 242
fegetround 236
feholdexcept 237, 241
fenv.h 29
feof 238, 284
feraiseexcept 239
ferror 240, 284, 366, 783
fesetenv 241
fesetexceptflag 242
fesetround 243
fetestexcept 244
feupdateenv 245
_fexp 223, 223
fflush 246, 264, 269-270, 272, 299, 1038
_ffree 285, 14, 285, 939
ffs 247
fget 248, 249, 318
fgetchar 249
__fgetchar 249
fgetpos 250, 297, 1126
fgets 251, 352
fgetwc 248
__fgetwchar 249
fgetws 251
__fheapchk 382, 382
__fheapgrow 385, 385
__fheapmin 386, 386
__fheapset 387, 387
__fheapshrink 389, 389
__fheapwalk 390, 390
__fieeetomsbin 252
FILE 18, 31
file open limits 372
File Operations 23
  access 57
  _access 57
  chmod 122
  _chmod 122
  lstat 916
  mkstemp 631
  _mktemp 633
  _remove 802
  rename 803
  stat 916
  _stat 916
  _stati64 916
  __tempnam 1017
  tmpnam 1024
  tmpnam_s 1022
  unlink 1040
  __unlink 1040
  _waccess 57
  _wchmod 122
  _wmktemp 633

1137
Index

_wremove 802
_wrename 803
_wsstat 916
_wsstati64 916
_wtmpnam 1017
_wtmpnam 1024
_wtmpnam_s 1022
_wunlink 1040
__FILE__ 71
filelength 253
_filelength 253
_filelength64 253
Filename Parsing Functions
    _fullpath 309
    _makepath 542
    _splitpath2 904
    _splitpath 902
    _wfullpath 309
    _wmakepath 542
    _wsplitpath2 904
    _wsplitpath 902
FILENAME_MAX 254
fileno 255, 18
FILES= 372
find_t 168
    _findclose 256, 258
    _finddata_t 257, 259
    _finddatai64_t 257, 259
    _findfirst 257, 256, 259
    _findfirsti64 257, 259
    _findnext 259, 258
    _findnexti64 259
    _finite 261, 772
fixed-point 769, 814
float.h 29
Floating Point Environment 29
    feclearexcept 231
    _fedisableexcept 232
    _feenableexcept 233
    fegetenv 234
    fegetexceptflag 235
    fegetround 236
    fefholdexcept 237
    feraiseexcept 239
    fesetenv 241
    fesetexceptflag 242
    fesetround 243
    fetestexcept 244
    feupdateenv 245
    _floodfill 262, 335
    _floodfill_w 262
floor 263
fltused_ 32
flushall 264, 269, 272

_fmalloc 544, 14, 385, 544, 939
_fmbccmp 553
_fmbccpy 555
_fmbccicmp 556
_fmbcstring 560
_fmbgetcode 571
_fmblen 572
_fmbputuchar 575
_fmbstrlen 576
_fmbrtowc 579
_fmbsbtype 582
_fmbscat 921
_fmbstrlen 925
_fmbstcmp 926
_fmbstcpy 930
_fmbscspn 934
_fmbssdec 937
_fmbssdup 939
_fmbssicmp 948
_fmbssinc 951
_fmbsslen 956
_fmbsslwr 959
_fmbssnbcat 985
_fmbssnbcmp 987
_fmbssnbcnt 988
_fmbssnbcpy 990
_fmbssnbicmp 992
_fmbssnbset 993
_fmbsscat 962
_fmbssncnt 954
_fmbssncmp 966
_fmbssncpy 969
_fmbssnextc 956
_fmbssnicmp 973
_fmbssnine 976
_fmbssset 979
_fmbsspbrk 981
_fmbssrch 983
_fmbssrev 985
_fmbssstowcs 988
_fmbssstowcs_s 601
_fmbssset 987
_fmbsssp 989
_fmbssspn 991
_fmbssstr 993
_fmbsstok 998
_fmbstowcs 604
_fmbstowcs_s 605
_fmbsupr 1008
_fmbsterm 607
_fmbtowc 609
_fmbvtop 611
_fmemccpy 614
_fmemchr 615
Index

_fmemcmp 616
_fmemcpy 617
_fmemicmp 619
_fmemmove 621
_fmemset 626
fmod 265, 1122
_fmode 32-33, 269, 272, 298, 702, 708, 764, 891
_fmsbintoieee 266
_fmsize 697
FNM_CASEFOLD 267
FNM_IGNORECASE 267
FNM_LEADING_DIR 267
FNM_NOESCAPE 267
FNM_PATHNAME 267
FNM_PERIOD 267
fnmatch 267, 29
fnmatch.h 29
fopen 269, 33, 227, 230, 288, 295
fopen_s 271
fp 288
FP_INFINITE 276
FP_NAN 276
FP_NORMAL 276
FP_OFF 274, 29, 630
FP_SEG 275, 29, 630
FP_SUBNORMAL 276
FP_ZERO 276
fpclassify 276
_fpreset 277
fprintf 278, 279, 720, 884, 886, 906, 1054, 1125
fprintf_s 279
fputc 281, 775
fputchar 282
_fputchar 282
fputs 283
fputwc 281
_fputwchar 282
fputws 283
fread 284
_frealloc 795, 14, 795
free 285, 14, 114, 285, 309, 326, 328, 795, 939
_frealloc 287
freopen 288, 18-19, 227, 295
freopen_s 289
frexp 291
fscanf 292, 293, 1058, 1125-1126
fscanf_s 293, 818, 913, 1060
fseek 295, 270, 272, 299, 307, 1038
fsetpos 297, 250, 270, 272, 299, 1038
_fsopen 298
fstat 301, 29, 31
_fstat 301
_fstat64 301, 303
_fstrcat 921

_fstrchr 925
_fstrcmp 926
_fstrcmp 930
_fstrcsn 934
_fstrdup 939
_fstricmp 948
_fstrlen 956
_fstrlwr 959
_fstrncat 962
_fstrncln 966
_fstrncp 969
_fstrnicm 973
_fstrmset 979
_fstrpbrk 981
_fstrrchr 983
_fstrrev 985
_fstrset 987
_fstrspn 989
_fstrspnp 991
_fstrstr 993
_fstrtok 998
_fstrupt 1008
fsync 305
ftell 307, 295, 1126
ftime 308, 31
_fullpath 309
_function 874
_function classification 3
_fwctomb 1087
_fwctomb_s 1090
_fwctombs 1093
_fwctombs_s 1096
_fwctombs 1100
_fwctombs_s 1102
_fwctomb 1107
_fwctomb_s 1109
fwprintf 278
fwprintf_s 279
fwrite 312
fwscanf 292
fwscanf_s 293

G

GAND 351, 780, 853
GBORDER 210, 757, 762, 797
GCLEARSCREEN 128
GCURSOROFF 157
GCURSORON 157

1139
Index

gcvt 313
_gcv 313
_get_osfhandle 346, 708
_getactivepage 314
_getarcinfo 315
_getbkcolor 317
getc 318, 320
getch 319, 321, 503, 1039
getchar 320
getche 321, 144, 319, 503, 1039, 1053
_getcliprgn 322
getcmd 323, 538
_getcolor 324
_getcurrentposition 325
_getcurrentposition_w 325
getcwd 326
_getdcwd 328
_getdiskfree 330
_getdrive 331
getenv 332, 37, 217, 778, 834, 898, 1127
_getenv_s 333
_getfillm 335
_getfontinfo 336
_getgtextextent 337
_getgtextvector 338
_getimage 339, 44, 396, 780
_getimage_w 339
_getimage_wxy 339
_getlinestyle 341
_getlogcoord 364
_getmbcp 342
getopt 343, 33-34
_getphyscoord 348
getpid 349
_getpixel 350
_getpixel_w 350
_getplotaction 351
gets 352, 251
gets_s 353
_gettextcolor 354
_gettextcursor 355
_gettextextent 356
_gettextposition 358, 863
_gettextsettings 359, 828
_gettextwindow 360
GetVersionEx 34, 36
_getvideoconfig 361, 40, 314, 365, 800, 825, 871
_getviewcoord 364
_getviewcoord_w 364
_getviewcoord_wxy 364
_getvisualpage 365
_getw 366
getwc 318, 320
_getwchar 320
_getwindowcoord 367
_getws 352
GFILLINTERIOR 210, 757, 762, 797
GMT 36
_gmtime 368
_gmtime 368, 368
gmtime_s 370
GOR 351, 780, 853
GPRESET 780
GPSET 351, 780, 853
_graph.h 29
_graphic page 40
_graphics adapters 39
_graphics functions 39
_graphics header files 46
_graphics library 39
_GRCLIPPED 374
Greenwich Mean Time 36
_GRERROR 374
GRFONTFILENAMEOTFOUND 374
_GRINSUFFICIENTMEMORY 374
GRINVALIDFILE 374
GRINVALIDPARAMETER 374
GRMODENOTSUPPORTED 374
GRNOOUTPUT 374
GRNOTINPROPERMODE 374
GROK 374
grouping 514
_grwHands 372
_grstatus 374
_grtext_w 375
GSCROLLDOWN 819
GSCROLLUP 819
GVIEWPORT 128
GWINDOW 128
GWRAPPOFF 1114
GWRAPON 1114
GXOR 351, 780, 853

H
HALF 854
halloc 377, 385, 393
hantozen 549
_harderr 378
_hardresum 378, 379
_hardret 378, 379
hardware port 399-401, 715-717

1140
_hdopen 381

Heap Functions 15
  _bheapchk 382
  _bheapmin 386
  _bheapse 387
  _bheaps 389
  _bheapwalk 390
  _fheapchk 382
  _fheaps 385
  _fheapmin 386
  _fheapse 387
  _fheapwalk 390
  _heaps 382
  _heapgrow 385
  _heaps 386
  _heapse 387
  _heaps 389
  _heapwalk 390
  _HEAPBADBEGIN 382, 387, 390
  _HEAPBADNODE 382, 387, 390
  _HEAPBADPTR 390
  _heapch 382, 387, 390
  _HEAPEMPTY 382, 387, 390
  _heapenable 384
  _HEAPEND 390
  _heapgrow 385
  _heainfo 390
  _heapmin 386, 386, 389
  _HEAPOK 382, 387, 390
  _heapse 387, 387, 390
  _heaps 389, 386, 389
  _heapwalk 390, 382, 387, 390

HERCUMONO 865
HERCULES 361
hfree 393
HGC 865
hInstance 538
hPrevInstance 538
HRES16COLOR 865
HRESBW 865
HUGE_VAL 997

Hyperbolic Functions
  acos 59
  acosh 60
  asinh 70
  atan 72
  atanh 74
  cosh 139
  _dos_allocmem 160
  sinh 879
tanh 1014
  hypot 394

IA MMX 30
IA MMX functions 26
  ignore_handler_s 395
  _imagesize 396, 339
  _imagesize_w 396
  _imagesize_wxy 396
  imaxabs 397
  imaxdiv 398
  imaxdiv_t 398
  INCLUDE 779, 835
  infinity 261, 772
  inp 399
  inp 400
  inpw 401
  int 850, 1047
  int386 402, 50
  int386x 403, 50
  __int64 770, 815
  int86 405, 50
  int86x 406, 50, 410
  intdos 407, 50
  intdosx 408, 50
  Intel classification 52
  Intel-Specific Functions 25
  Interrupt Functions
    __disable 156
    __enable 212
  INTMAX_MAX 1004
  INTMAX_MIN 1004
  intmax_t 770, 814
  INTPACK 29
  intr 410, 50
  inttypes.h 29
  io.h 29
  __IOFBF 864
  __IOLBF 864
  __IONBF 864
  isalnum 411, 430, 1121
  isalpha 412, 411, 432, 1121
  isasci 413
  __isasci 413
Index

isatty 415
iskey 415
isblank 416
iscntrl 418, 1121
iscsym 419
__iscsym 419
iscsymf 421
__iscsymf 421
isdigit 423, 411
isfinite 424
isgraph 425, 434, 492
isinf 426
isleadbyte 427
islower 429, 412, 1121
_ismbbalnum 430
_ismbbalpha 432
_ismbgraph 434
_ismbkalnum 436, 430
_ismbkalpha 440, 432
_ismbkana 438
_ismbkprintf 442, 434
_ismbkpunct 444
_ismbblead 446
_ismbbprintf 448
_ismbbprintf 450
_ismbbtrail 452
_ismbcalnum 454
_ismbcalpha 456, 454
_ismbccntrl 458
_ismbcdigit 460, 454
_ismbcgraph 462, 478
_ismbchira 464
_ismbckata 466
_ismbcl0 468
_ismbcl1 470
_ismbcl2 472
_ismbclegal 474
_ismbclower 476
_ismbcprint 478, 462
_ismbcprintf 480
_ismbcspace 482
_ismbcsymbol 484
_ismbcupper 486
_ismbcxdigit 488
isnan 490
isnormal 491
ISO classification 51
isprint 492, 425, 1121
ispunct 493
isspace 495
isupper 497, 412, 1121
iswalnum 411, 416, 493, 495, 1112
iswalpha 412, 411, 1112
iswascii 413
iswblank 416, 1112
iswcntrl 418, 412, 429, 497, 1112
__iswesym 419
__iswcsym 421
iswctype 498, 1112
iswdigit 423, 411-412, 429, 497, 1112
iswgraph 425
iswlower 429, 412, 1112
iswprint 492
iswpunct 493, 412, 429, 497, 1112
iswspace 495, 412, 429, 497, 1112
iswupper 497, 412, 1112
iswxdigit 500
isxdigit 500
j0 87, 87
j1 87, 87
jistojsms 558
jmpBuf 526, 841
jmstojis 559
jn 87, 87
jtohira 566
jtokata 568
jtolower 562
jtoupper 564

K

kbbhit 503, 319, 321, 1039
kbhit 503
_KEYBRD_READ 98
_KEYBRD_READY 98
_KEYBRD_SHIFTSTATUS 98

L

L_tmpnam 1024
L_tmpnam_s 1022
Index

labs 504
LC_ALL 844
LC_COLLATE 844
LC_CTYPE 844, 1111-1112
LC_MONETARY 844
LC_NUMERIC 844
LC_TIME 844
ldexp 505
ldiv 506
ldiv_t 506
LEFT 854
lfind 507, 30
limits
  file open 372
limits.h 29
  _LINE_ 71
lineto 42
  _lineto 509, 42, 325, 639
  _lineto_w 509
  _LK_LOCK, LK_LOCK 521
  _LK_LOCK 521
  _LK_NBLCK, LK_NBLCK 521
  _LK_NBLCK 521
  _LK_NBRLCK, LK_NBRLCK 521
  _LK_RBLC, LK_LRLCK 521
  _LK_UNLOCK, LK_UNLOCK 521
llabs 511
lldiv 512
lldiv_t 512
LLONG_MAX 1003
LLONG_MIN 1003
ltoa 534
  _ltoa 534
  _ltow 534
Locale Functions
  localeconv 513
    setlocale 844
      _wsetlocale 844
locale.h 29
localeconv 513
localtime 516, 38
  _localtime 516, 38, 516
localtime_s 518
lock 520
  _locking 521
log 523, 1122
log10 524, 1122
log2 525, 1122
long double 770, 815
long long 770
LONG_MAX 1002
LONG_MIN 1002
longjmp 526, 30, 787, 841, 874
lpszCmdLine 538
  _lrotl 527
  _lrot 528
lsearch 529, 30
lseek 531, 790, 1015, 1115
  _lseek 531
  _lseeki64 531, 1015
lstat 916, 918
ltoa 536
  _ltoas 536
  _ltow 536

M

  __m64 30
  __m_empty 624
  __m_from_int 627
  __m_packssdw 640
  __m_packsswb 642
  __m_packuswb 644
  __m_padd 646
  __m_padd 647
  __m_padd 648
  __m_paddsw 649
  __m_paddusb 650
  __m_paddusw 651
  __m_paddw 652
  __m_pand 653
  __m_pandn 654
  __m_pcmpeqb 655
  __m_pcmpeqd 656
  __m_pcmpeqw 657
  __m_pcmpgtb 658
  __m_pcmpgtd 659
  __m_pcmpgtw 660
  __m_maddwd 661
  __m_maddw 662
  __m_maddw 663
  __m_maddw 664
  __m_mmaddwd 665
  __m_mmaddwd 666
  __m_mmaddwd 666
  __m_mmaddwd 666
  __m_mmaddwd 667
  __m_mmaddwd 668
  __m_mmaddwd 669
  __m_mmaddwd 670
  __m_mmaddwd 671
  __m_mmaddwd 672
  __m_mmaddwd 673
  __m_mmaddwd 674
  __m_mmaddwd 675

1143
Index

_m_psrldi 676
_m_psrld 677
_m_psrlqi 678
_m_psrlq 679
_m_psrlwi 680
_m_psrlw 681
_m_psubb 682
_m_psusb 683
_m_psubsw 684
_m_psubsb 685
_m_psubusw 686
_m_psubw 687
_m_punpckhbhw 688
_m_punpckhdq 690
_m_punpckhwd 691
_m_punpckhbw 692
_m_punpckhdq 694
_m_punpckhldw 695
_m_pxor 696
_m_to_int 698
main 538, 32, 149, 343, 1084
main program 538
_makepath 542
malloc 544, 14, 83, 285, 287, 309, 326, 328, 385, 444, 697, 939, 1017, 1126
malloc.h 29
math.h 29
Mathematical Functions 15, 30
acos 59
acosh 60
asin 69
asinh 70
atan 72
atan2 73
atanh 74
bessel Functions 87
cabs 113
ceil 116
cos 138
cosh 139
_dieeetomsbin 153
_dmshintoiee 159
exp 222
fabs 225
_fieeetomsbin 252
_finite 261
floor 263
fmod 265
_fmsbintoiee 266
frexp 291
hypot 394
j0 87
j1 87
jn 87
ldexp 505
log 523
log10 524
log2 525
matherr 546
modf 637
pow 766
_set_matherr 846
sin 878
sinh 879
sqrt 910
tan 1013
tanh 1014
y0 87
y1 87
yn 87
matherr 546, 30, 59-60, 69, 73-74, 87, 113, 139, 222, 394, 523-525, 766, 846, 879, 910, 1014
_matherr 87
max 548
_MAX_DIR 542, 902
_MAX_DRIVE 542, 902
_MAX_EXT 542, 902
_MAX_FNAME 542, 902
_MAX_PATH 544, 902
_MAX_PATH2 902
_MAX_VOLUME 902
MAXCOLORMODE 865
MAXRESMODE 865
_MaxThreads 33
MB_CUR_MAX 576, 579, 609, 1087, 1090, 1107, 1109
_mbttombc 549
_mbstype 550
_MBC_ILLEGAL 550, 582
_MBC_LEAD 550, 582
_MBC_SINGLE 550, 582
_MBC_TRAIL 550, 582
_mbccmp 553
_mbccpy 555
_mbcicmp 556
_mbcjistojms 558
_mbcjnstojjs 559
_mbclen 560
_MBCS 588, 594, 596, 976
_mbctohira 566
_mbctokata 568
_mbctolower 562, 13
_mbctombb 570
_mbctoupper 564, 13
_mbgetcode 571
_mbgetchar 575
_mbstrlen 572, 576, 579, 1087
_mbputchar 575
Index

mbrlen 576
mbrtocw 579, 598, 601, 605
_mbsbtype 582
_mbscat 921
_mbschr 925
_mbscmp 926
_mbscoll 929
_mbscpy 930
_mbscspn 934
_mbsdec 937
_mbsdup 939
_mbsicmp 948
_mbsicoll 950
_mbsinc 951
_mbsinit 880
_mbslen 956
_mbslwr 959, 13
_mbsnbcat 585
_mbsnbcmp 587
_mbsnbcnt 588
_mbsnbcpy 90
_mbsnbcpy 90
_mbsnbicmp 922
_mbsnbiset 932
_mbsncat 962
_mbsncnt 954
_mbsncmp 966, 587, 592
_mbsncoll 968
_mbsncpy 969
_mbsnextc 996
_mbsnicmp 973
_mbsnicoll 975
_mbsninc 976
_mbsninc 976
_mbsnot 979, 593
_mbspbrk 981
_mbsrchr 983
_mbsrev 985
mbsrtowcs 958
mbsrtowcs_s 601
_mbsset 987
_mbsstr 993
_mbsstr 993
_mbstate_t 31, 576, 579, 598, 880, 1087, 1093
_mbstok 998
mbsrtowcs 604, 598, 1093
mbsrtowcs_s 605
_mbsupr 1008, 13
_mbtok 611
mbtowc 609, 576, 579, 1087
_mbtok 611
MCGA 361, 865
MDPA 361, 865
_memavl 613
memccpy 614
memchr 615
memcpy 616, 88
memcpy 617, 621
memcpy_s 618
memcmp 619
_memicmp 619
_memmax 620, 613
memmove 621, 89, 617, 930, 955, 969
memmove_s 622, 618

Memory Allocation 14
alloc 61
_balloc 114
_bexpand 223
_bfree 285
_bfree 285
_bfree 285
_bheapchk 382
_bheapchk 382
_bheapmin 386
_bheapseg 93
_bheapset 387
_bheapshr 392
_bheapwalk 390
_bmalloc 544
_bmsize 697
_breakalloc 795
calloc 114
_expand 223
_falloc 114
_fexpand 223
_ffree 285
_fheapchk 382
_fheapchk 382
_fheapgrow 385
_fheapmin 386
_fheapset 387
_fheapshr 389
_fheapwalk 390
_fmalloc 544
_fmsize 697
_frealloc 795
_free 285
_free 287
_halloc 377
_heapchk 382
_heapgrow 385
_heapmin 386
_heapset 387
_heapshr 389
_heapwalk 390
_hfree 393
_hmalloc 544
_memavl 613
_memmax 620
_memsize 697
_nmalloc 114
_nexpand 223
### Index

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>_nfree</td>
<td>285</td>
</tr>
<tr>
<td>_nheapchk</td>
<td>382</td>
</tr>
<tr>
<td>_nheapgrow</td>
<td>385</td>
</tr>
<tr>
<td>_nheapmin</td>
<td>386</td>
</tr>
<tr>
<td>_nheapset</td>
<td>387</td>
</tr>
<tr>
<td>_nheapshrink</td>
<td>389</td>
</tr>
<tr>
<td>_nmalloc</td>
<td>544</td>
</tr>
<tr>
<td>_nmsize</td>
<td>697</td>
</tr>
<tr>
<td>_nrealloc</td>
<td>795</td>
</tr>
<tr>
<td>realloc</td>
<td>795</td>
</tr>
<tr>
<td>sbrk</td>
<td>810</td>
</tr>
<tr>
<td>stackavail</td>
<td>915</td>
</tr>
<tr>
<td>memset</td>
<td>626, 112</td>
</tr>
<tr>
<td>min</td>
<td>628</td>
</tr>
<tr>
<td>__minreal</td>
<td>33</td>
</tr>
<tr>
<td>Miscellaneous Functions</td>
<td>27</td>
</tr>
<tr>
<td>MK_FP</td>
<td>630, 29</td>
</tr>
<tr>
<td>mkdir</td>
<td>629</td>
</tr>
<tr>
<td>_mkdir</td>
<td>629</td>
</tr>
<tr>
<td>mkstemp</td>
<td>631</td>
</tr>
<tr>
<td>_mktemp</td>
<td>633</td>
</tr>
<tr>
<td>mktime</td>
<td>635, 38</td>
</tr>
<tr>
<td>mmintrin.h</td>
<td>30</td>
</tr>
<tr>
<td>MMX</td>
<td>30</td>
</tr>
<tr>
<td>MMX detection</td>
<td>624</td>
</tr>
<tr>
<td>MMX functions</td>
<td>26</td>
</tr>
<tr>
<td>modf</td>
<td>637</td>
</tr>
<tr>
<td>modtime</td>
<td>1043</td>
</tr>
<tr>
<td>mon_grouping</td>
<td>514</td>
</tr>
<tr>
<td>MONO</td>
<td>362</td>
</tr>
<tr>
<td>movedata</td>
<td>638</td>
</tr>
<tr>
<td>_moveto</td>
<td>639, 325, 358, 712, 861</td>
</tr>
<tr>
<td>_moveto_w</td>
<td>639</td>
</tr>
<tr>
<td>MRES16COLOR</td>
<td>865</td>
</tr>
<tr>
<td>MRES256COLOR</td>
<td>865</td>
</tr>
<tr>
<td>MRES4COLOR</td>
<td>865</td>
</tr>
<tr>
<td>MRESNOCOLOR</td>
<td>865</td>
</tr>
<tr>
<td>_msize</td>
<td>697, 223, 697</td>
</tr>
<tr>
<td>mtob</td>
<td>588</td>
</tr>
</tbody>
</table>

Multibyte Character Functions 6, 10-11

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>_fnbcncmp</td>
<td>553</td>
</tr>
<tr>
<td>_fnbcncpy</td>
<td>555</td>
</tr>
<tr>
<td>_fnbcicmp</td>
<td>556</td>
</tr>
<tr>
<td>_fnbclen</td>
<td>560</td>
</tr>
<tr>
<td>_fnblen</td>
<td>572</td>
</tr>
<tr>
<td>_fnbscat</td>
<td>921</td>
</tr>
<tr>
<td>_fnbschr</td>
<td>925</td>
</tr>
<tr>
<td>_fnbscmp</td>
<td>926</td>
</tr>
<tr>
<td>_fnbscpy</td>
<td>930</td>
</tr>
<tr>
<td>_fnbscsn</td>
<td>934</td>
</tr>
<tr>
<td>_fnbsdec</td>
<td>937</td>
</tr>
<tr>
<td>_fmbsdup</td>
<td>939</td>
</tr>
<tr>
<td>_fmbsicmp</td>
<td>948</td>
</tr>
<tr>
<td>_fmbsinc</td>
<td>951</td>
</tr>
<tr>
<td>_fmbslen</td>
<td>956</td>
</tr>
<tr>
<td>_fmbslwr</td>
<td>959</td>
</tr>
<tr>
<td>_fmbsncat</td>
<td>962</td>
</tr>
<tr>
<td>_fmbsncmp</td>
<td>966</td>
</tr>
<tr>
<td>_fmbsncpy</td>
<td>969</td>
</tr>
<tr>
<td>_fmbsnicmp</td>
<td>973</td>
</tr>
<tr>
<td>_fmbsnine</td>
<td>976</td>
</tr>
<tr>
<td>_fmbsnset</td>
<td>979</td>
</tr>
<tr>
<td>_fmbsrchr</td>
<td>983</td>
</tr>
<tr>
<td>_fmbsrev</td>
<td>985</td>
</tr>
<tr>
<td>_fmbsrtowcs</td>
<td>598</td>
</tr>
<tr>
<td>_fmbsrtowcs_s</td>
<td>601</td>
</tr>
<tr>
<td>_fmsbset</td>
<td>987</td>
</tr>
<tr>
<td>_fmsbsspnp</td>
<td>989</td>
</tr>
<tr>
<td>_fmsbsspnpn</td>
<td>991</td>
</tr>
<tr>
<td>_fmsbstr</td>
<td>993</td>
</tr>
<tr>
<td>_fmbstowcs</td>
<td>604</td>
</tr>
<tr>
<td>_fmbstowcs_s</td>
<td>605</td>
</tr>
<tr>
<td>_fmsbsupr</td>
<td>1008</td>
</tr>
<tr>
<td>_fmbterm</td>
<td>607</td>
</tr>
<tr>
<td>_fmbtowc</td>
<td>609</td>
</tr>
<tr>
<td>_fmbvtop</td>
<td>611</td>
</tr>
<tr>
<td>_fwctomb</td>
<td>1087</td>
</tr>
<tr>
<td>_fwctomb_s</td>
<td>1090</td>
</tr>
<tr>
<td>_fwcsrtombs</td>
<td>1093</td>
</tr>
<tr>
<td>_fwcsrtombs_s</td>
<td>1096</td>
</tr>
<tr>
<td>_fwestombs</td>
<td>1100</td>
</tr>
<tr>
<td>_fwestombs_s</td>
<td>1102</td>
</tr>
<tr>
<td>_fwctomb</td>
<td>1107</td>
</tr>
<tr>
<td>_fwctomb_s</td>
<td>1109</td>
</tr>
<tr>
<td>_mbccmp</td>
<td>553</td>
</tr>
<tr>
<td>_mbccpy</td>
<td>555</td>
</tr>
<tr>
<td>_mbccmp</td>
<td>556</td>
</tr>
<tr>
<td>_mbcicmp</td>
<td>560</td>
</tr>
<tr>
<td>_mbcicmp</td>
<td>560</td>
</tr>
<tr>
<td>_mbscat</td>
<td>921</td>
</tr>
<tr>
<td>_mbschr</td>
<td>925</td>
</tr>
<tr>
<td>_mbscmp</td>
<td>926</td>
</tr>
<tr>
<td>_mbscoll</td>
<td>929</td>
</tr>
<tr>
<td>_mbscpy</td>
<td>930</td>
</tr>
<tr>
<td>_mbscspn</td>
<td>934</td>
</tr>
<tr>
<td>_mbscspn</td>
<td>934</td>
</tr>
<tr>
<td>_mbscpy</td>
<td>934</td>
</tr>
<tr>
<td>_mbsdup</td>
<td>934</td>
</tr>
<tr>
<td>_mbsdup</td>
<td>934</td>
</tr>
<tr>
<td>_mbdec</td>
<td>937</td>
</tr>
<tr>
<td>_mbdec</td>
<td>937</td>
</tr>
<tr>
<td>_mbsdup</td>
<td>939</td>
</tr>
<tr>
<td>_mbsdup</td>
<td>939</td>
</tr>
<tr>
<td>_mbsicmp</td>
<td>948</td>
</tr>
<tr>
<td>_mbsicmp</td>
<td>948</td>
</tr>
<tr>
<td>_mbsicoll</td>
<td>950</td>
</tr>
<tr>
<td>_mbsicoll</td>
<td>950</td>
</tr>
<tr>
<td>_mbsinc</td>
<td>951</td>
</tr>
<tr>
<td>_mbsinc</td>
<td>951</td>
</tr>
<tr>
<td>_mbsinit</td>
<td>880</td>
</tr>
<tr>
<td>_mbslen</td>
<td>956</td>
</tr>
<tr>
<td>_mbslwr</td>
<td>959</td>
</tr>
<tr>
<td>_mbslwr</td>
<td>959</td>
</tr>
<tr>
<td>_mbsncat</td>
<td>962</td>
</tr>
<tr>
<td>_mbsncat</td>
<td>962</td>
</tr>
<tr>
<td>_mbsncmp</td>
<td>966</td>
</tr>
<tr>
<td>_mbsncmp</td>
<td>966</td>
</tr>
<tr>
<td>Function</td>
<td>Page(s)</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>_mbsncoll</td>
<td>968</td>
</tr>
<tr>
<td>_mbsncpy</td>
<td>969</td>
</tr>
<tr>
<td>_mbsnicmp</td>
<td>973</td>
</tr>
<tr>
<td>_mbsnicoll</td>
<td>975</td>
</tr>
<tr>
<td>_mbsnicoll</td>
<td>976</td>
</tr>
<tr>
<td>_mbsnicoll</td>
<td>979</td>
</tr>
<tr>
<td>_mbsrchr</td>
<td>983</td>
</tr>
<tr>
<td>_mbsrev</td>
<td>985</td>
</tr>
<tr>
<td>mbstowcs</td>
<td>598</td>
</tr>
<tr>
<td>mbstowcs_s</td>
<td>601</td>
</tr>
<tr>
<td>_mbset</td>
<td>987</td>
</tr>
<tr>
<td>_mbsspn</td>
<td>989</td>
</tr>
<tr>
<td>_mbssppn</td>
<td>991</td>
</tr>
<tr>
<td>_mbssstr</td>
<td>993</td>
</tr>
<tr>
<td>mbstowcs</td>
<td>604</td>
</tr>
<tr>
<td>mbstowcs_s</td>
<td>605</td>
</tr>
<tr>
<td>_mbsupr</td>
<td>1008</td>
</tr>
<tr>
<td>_mbterm</td>
<td>607</td>
</tr>
<tr>
<td>mbtowc</td>
<td>609</td>
</tr>
<tr>
<td>_mbvtop</td>
<td>611</td>
</tr>
<tr>
<td>wcrtomb</td>
<td>1087</td>
</tr>
<tr>
<td>wcrtomb_s</td>
<td>1090</td>
</tr>
<tr>
<td>wcsrtombs</td>
<td>1093</td>
</tr>
<tr>
<td>wcsrtombs_s</td>
<td>1096</td>
</tr>
<tr>
<td>wcsstombs</td>
<td>1100</td>
</tr>
<tr>
<td>wcsstombs_s</td>
<td>1102</td>
</tr>
<tr>
<td>wctob</td>
<td>1105</td>
</tr>
<tr>
<td>wctomb</td>
<td>1107</td>
</tr>
<tr>
<td>wctomb_s</td>
<td>1109</td>
</tr>
<tr>
<td>Multimedia Extension</td>
<td>30</td>
</tr>
<tr>
<td>Multimedia Extension functions</td>
<td>26</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>_NKEYBRD_READ</td>
<td>98</td>
</tr>
<tr>
<td>_NKEYBRD_READY</td>
<td>98</td>
</tr>
<tr>
<td>_NKEYBRD_SHIFTSTATUS</td>
<td>98</td>
</tr>
<tr>
<td>_nmalloc</td>
<td>544, 14, 287, 544</td>
</tr>
<tr>
<td>_nmsize</td>
<td>697, 697</td>
</tr>
<tr>
<td>NODISPLAY</td>
<td>361</td>
</tr>
<tr>
<td>Non-local Jumps</td>
<td>30</td>
</tr>
<tr>
<td>longjmp</td>
<td>526</td>
</tr>
<tr>
<td>setjmp</td>
<td>841</td>
</tr>
<tr>
<td>NORMAL</td>
<td>854</td>
</tr>
<tr>
<td>nosound</td>
<td>699, 894</td>
</tr>
<tr>
<td>_nrealloc</td>
<td>795, 14, 795</td>
</tr>
<tr>
<td>nthctype</td>
<td>582</td>
</tr>
<tr>
<td>NULL</td>
<td>84, 976, 1121, 1126</td>
</tr>
<tr>
<td>_NULLOFF</td>
<td>114, 544, 796</td>
</tr>
<tr>
<td>_NULSEG</td>
<td>93, 382, 389</td>
</tr>
</tbody>
</table>

## O

<table>
<thead>
<tr>
<th>Function</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>O_APPEND</td>
<td>702, 708, 890, 1115</td>
</tr>
<tr>
<td>O_BINARY</td>
<td>33, 702, 708, 790, 849, 890-891, 1115</td>
</tr>
<tr>
<td>O_CREAT</td>
<td>702-703, 708, 890-891</td>
</tr>
<tr>
<td>O_EXCL</td>
<td>702, 708, 890</td>
</tr>
<tr>
<td>O_NOINHERIT</td>
<td>182, 702, 708, 890</td>
</tr>
<tr>
<td>O_NONBLOCK</td>
<td>759</td>
</tr>
<tr>
<td>O_RDONLY</td>
<td>182, 702, 708, 790, 890, 1115</td>
</tr>
<tr>
<td>O_RDWR</td>
<td>182, 702, 708, 890, 1115</td>
</tr>
<tr>
<td>O_TEXT</td>
<td>33, 702, 708, 790, 849, 890-891, 1115</td>
</tr>
<tr>
<td>O_TRUNC</td>
<td>702, 708, 890</td>
</tr>
<tr>
<td>O_WRONLY</td>
<td>182, 702, 708, 890, 1115</td>
</tr>
<tr>
<td>offsetof</td>
<td>700, 30</td>
</tr>
<tr>
<td>onexit</td>
<td>701, 220</td>
</tr>
<tr>
<td>open</td>
<td>702, 29, 33, 130, 230, 255, 531, 790, 1015, 1036, 1115</td>
</tr>
<tr>
<td>_open</td>
<td>702</td>
</tr>
<tr>
<td>_open_osfhandle</td>
<td>708, 346</td>
</tr>
<tr>
<td>opendir</td>
<td>705, 131, 706, 792, 805</td>
</tr>
<tr>
<td>optarg</td>
<td>33, 343</td>
</tr>
<tr>
<td>optarg</td>
<td>33, 343</td>
</tr>
<tr>
<td>optad</td>
<td>33, 343</td>
</tr>
<tr>
<td>optopt</td>
<td>33, 343</td>
</tr>
<tr>
<td>OS/2 classification</td>
<td>52</td>
</tr>
<tr>
<td>OS/2 Functions</td>
<td>_beginthread 83</td>
</tr>
<tr>
<td></td>
<td>_cwait 149</td>
</tr>
<tr>
<td></td>
<td>_endthread 213</td>
</tr>
<tr>
<td></td>
<td>_wait 1084</td>
</tr>
<tr>
<td>_os_handle</td>
<td>711</td>
</tr>
<tr>
<td>_osbuild</td>
<td>34</td>
</tr>
</tbody>
</table>

Index

<table>
<thead>
<tr>
<th>Function</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NaN</td>
<td>261, 772</td>
</tr>
<tr>
<td>N_calloc</td>
<td>114, 14, 114</td>
</tr>
<tr>
<td>NCmdShow</td>
<td>538</td>
</tr>
<tr>
<td>NDEBUG</td>
<td>71</td>
</tr>
<tr>
<td>new</td>
<td>850</td>
</tr>
<tr>
<td>nExitCode</td>
<td>539</td>
</tr>
<tr>
<td>_nexpand</td>
<td>223, 223</td>
</tr>
<tr>
<td>_NFPOSE</td>
<td>372</td>
</tr>
<tr>
<td>_nfree</td>
<td>285, 14, 285</td>
</tr>
<tr>
<td>_nheapchk</td>
<td>382, 382</td>
</tr>
<tr>
<td>_nheapgrow</td>
<td>385, 287, 385, 613, 620</td>
</tr>
<tr>
<td>_nheapmin</td>
<td>386, 386</td>
</tr>
<tr>
<td>_nheapset</td>
<td>387, 387</td>
</tr>
<tr>
<td>_nheapshrink</td>
<td>389, 389</td>
</tr>
<tr>
<td>_nheapwalk</td>
<td>390, 390</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NKEYBRD_READ</td>
<td>98</td>
</tr>
<tr>
<td>NKEYBRD_READY</td>
<td>98</td>
</tr>
<tr>
<td>NKEYBRD_SHIFTSTATUS</td>
<td>98</td>
</tr>
<tr>
<td>_nmalloc</td>
<td>544, 14, 287, 544</td>
</tr>
<tr>
<td>_nmsize</td>
<td>697, 697</td>
</tr>
<tr>
<td>NODISPLAY</td>
<td>361</td>
</tr>
<tr>
<td>Non-local Jumps</td>
<td>30</td>
</tr>
<tr>
<td>longjmp</td>
<td>526</td>
</tr>
<tr>
<td>setjmp</td>
<td>841</td>
</tr>
<tr>
<td>NORMAL</td>
<td>854</td>
</tr>
<tr>
<td>nosound</td>
<td>699, 894</td>
</tr>
<tr>
<td>_nrealloc</td>
<td>795, 14, 795</td>
</tr>
<tr>
<td>nthctype</td>
<td>582</td>
</tr>
<tr>
<td>NULL</td>
<td>84, 976, 1121, 1126</td>
</tr>
<tr>
<td>_NULLOFF</td>
<td>114, 544, 796</td>
</tr>
<tr>
<td>_NULSEG</td>
<td>93, 382, 389</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>O_APPEND</td>
<td>702, 708, 890, 1115</td>
</tr>
<tr>
<td>O_BINARY</td>
<td>33, 702, 708, 790, 849, 890-891, 1115</td>
</tr>
<tr>
<td>O_CREAT</td>
<td>702-703, 708, 890-891</td>
</tr>
<tr>
<td>O_EXCL</td>
<td>702, 708, 890</td>
</tr>
<tr>
<td>O_NOINHERIT</td>
<td>182, 702, 708, 890</td>
</tr>
<tr>
<td>O_NONBLOCK</td>
<td>759</td>
</tr>
<tr>
<td>O_RDONLY</td>
<td>182, 702, 708, 790, 890, 1115</td>
</tr>
<tr>
<td>O_RDWR</td>
<td>182, 702, 708, 890, 1115</td>
</tr>
<tr>
<td>O_TEXT</td>
<td>33, 702, 708, 790, 849, 890-891, 1115</td>
</tr>
<tr>
<td>O_TRUNC</td>
<td>702, 708, 890</td>
</tr>
<tr>
<td>O_WRONLY</td>
<td>182, 702, 708, 890, 1115</td>
</tr>
<tr>
<td>offsetof</td>
<td>700, 30</td>
</tr>
<tr>
<td>onexit</td>
<td>701, 220</td>
</tr>
<tr>
<td>open</td>
<td>702, 29, 33, 130, 230, 255, 531, 790, 1015, 1036, 1115</td>
</tr>
<tr>
<td>_open</td>
<td>702</td>
</tr>
<tr>
<td>_open_osfhandle</td>
<td>708, 346</td>
</tr>
<tr>
<td>opendir</td>
<td>705, 131, 706, 792, 805</td>
</tr>
<tr>
<td>optarg</td>
<td>33, 343</td>
</tr>
<tr>
<td>optarg</td>
<td>33, 343</td>
</tr>
<tr>
<td>optad</td>
<td>33, 343</td>
</tr>
<tr>
<td>optopt</td>
<td>33, 343</td>
</tr>
<tr>
<td>OS/2 classification</td>
<td>52</td>
</tr>
<tr>
<td>OS/2 Functions</td>
<td>_beginthread 83</td>
</tr>
<tr>
<td></td>
<td>_cwait 149</td>
</tr>
<tr>
<td></td>
<td>_endthread 213</td>
</tr>
<tr>
<td></td>
<td>_wait 1084</td>
</tr>
<tr>
<td>_os_handle</td>
<td>711</td>
</tr>
<tr>
<td>_osbuild</td>
<td>34</td>
</tr>
</tbody>
</table>

Index
Index

_osmajor 34
=osminor 34
=osmode 34
=osver 34
_outmem 714, 43, 354, 358, 375, 712, 714, 718, 832, 856, 861, 863, 870, 1114
_outp 715
_outpd 716
_outpw 717
_outtext 718, 43, 354, 358, 375, 712, 714, 718, 832-833, 856, 861, 863, 870, 1114
OVERFLOW 546, 846

PharLap TNT Functions

_beginthread 83
_beginthreadex 83
_physical coordinates 41
_pie 42
_pie 756, 42, 315, 335
_pie_w 756
_pie_wxy 756
_pipe 759, 764
_polygon 42
_polygon 762, 42, 335
_polygon_w 762
_polygon_wxy 762
_popen 764, 719
_port 764

PC Graphics classification 52
_pclose 764
_pclose 719
_pentry 390
 perror 720, 32, 1126
_PFU 850
_PFV 850
_pg_analyzechart 721
_pg_analyzechartms 721
_pg_analyzePie 723
_pg_analyzeScatter 725
_pg_analyzeScatterms 725
_PG_BARCHART 736
_pg_chart 727, 721
_pg_chartms 727, 721
_pg_chartsPie 730, 723
_pg_chartsScatter 733, 725
_pg_chartsScatterms 733, 725
_pg_columnchart 736
_pg_defaultchart 736
_pg_getchardef 738
_pg_getpalette 739
_pg_getstyleset 741
_pg_hlabelchart 743
_pg_initchart 744, 45
_PG_LINECHART 736
_PG_NOPERCENT 736
_PG_PERCENT 736
_PG_PIECHART 736
_PG_PLAINBARS 736
_PG_POINTANDLINE 736
_PG_POINTONLY 736
_pg_resetpalette 746
_pg_resetstyleset 748
_PG_SCATTERCHART 736
_pg_setchardef 750
_pg_setpalette 751
_pg_setstyleset 753
_PG_STACKEDBARS 736
_pg_vlabelchart 755

PharLap TNT Functions

_POSTQUITMESSAGE 539
_pow 766
_pow(0,0,0) 1122
_pow(0,neg) 1122
_pow(neg,frac) 1122
_prime Meridian 37
_PRINTERS_INIT 101
_PRINTERS_STATUS 101
Index

_PRINTER_WRITE 101
printf 767, 32, 105, 140, 278, 769-770, 773, 861, 884, 886, 906, 1051-1052, 1054, 1062, 1070, 1072, 1076
printf_s 773
PRN 48
Process Functions 20, 22, 30, 50
 abort 54
 abort_handler_s 55
 atexit 75
 _bgetcmd 92
 clearenv 126
 execl 216
 execlp 216
 execv 216
 execve 216
 execvp 216
 execvpe 216
 _exit 220
 getcmd 323
 getenv 332
 ignore_handler_s 395
 main 538
 onexit 701
 putenv 778
 _putenv 778
 set_constraint_handler_s 823
 setenv 834
 _setenv 834
 spawnl 896
 spawnle 896
 spawnlp 896
 spawnlpe 896
 spawnnv 896
 spawnnve 896
 spawnvp 896
 spawnvpe 896
 spawnvpepe 896
 system 1012
 _wexecl 216
 _wexecl 216
 _wexeclp 216
 _wexeclpe 216
 _wexecv 216
 _wexecve 216
 _wexecvp 216
 _wexecvpe 216
 _wgetenv 332
 _wputenv 778
 _wsetenv 834
 _wspawnl 896
 _wspawnle 896
 _wspawnlv 896
 _wspawnlve 896
 _wspawnlpe 896
 _wspawnvp 896
 _wspawnvpe 896
 _wspawnvpepe 896
 _wsystem 1012
 process.h 30
 Program Segment Prefix 34
 PSP 34
 _psp 34, 181
 ptrdiff_t 30, 770, 815
 putc 775
 putch 776, 140-141, 1052
 putchar 777, 282
 putenv 778, 37, 217, 332-333, 834, 897, 1127
 _putenv 778
 _putimage 780, 44, 339
 _putimage_w 780
 puts 782, 141
 _putw 783
 putwc 775
 putwchar 777
 _putws 782

Q

qsort 784
qsort_s 785
quot 158, 398, 506, 512

R

R_OK 57
raise 787, 30, 526, 875
rand 789, 911
RAND_MAX 789
Random Numbers
 rand 789
 srand 911
 read 790
 _read 790
 readdir 792, 705-706
 realloc 795, 14, 285, 697, 795, 1126
 rectangle 42
 _rectangle 797, 42, 335
 _rectangle_w 797
 _rectangle_wxy 797
Index

_registrofonts 799, 44, 838, 1042
REGPACK 29
REGS 29
rem 158, 398, 506, 512
_remapallpalette 800
_remappalette 801
remove 802, 1040, 1125
rename 803, 1125
ResumeThread 84
return 538
rewind 804, 127, 270, 272, 299, 1038
rewinddir 805
RIGHT 854
rmdir 807
˘rmdir 807
Rotate Functions
  _lr 527
  _lr 528
  _rotl 808
  _rotr 809
  _rotl 808
  _rotr 809
RSIZE_MAX 109, 333, 353, 601, 605, 618, 785, 923, 932, 941, 964, 971, 1000, 1022, 1090, 1096, 1102, 1109

S

s 293
S_IEXEC 122, 142, 302, 703, 891, 917, 1037
S_IREAD 122, 143, 302, 703, 891, 917, 1036
S_IROGR 122, 142, 302, 703, 891, 917, 1036
S_IROTH 122, 142, 302, 703, 891, 917, 1036
S_IRUSR 122, 142, 302, 703, 891, 917, 1036-1037
S_IRWXG 122, 142, 302, 703, 891, 917, 1036
S_IRWXO 122, 142, 302, 703, 891, 917, 1036
S_IRWXU 122, 142, 302, 703, 891, 917, 1036
S_ISBLK(m) 302, 917
S_ISCHR(m) 302, 917
S_ISDIR(m) 302, 917
S_ISFIFO(m) 302, 917
S_ISGID 303, 918
S_ISREG(m) 302, 917
S_ISUID 303, 918
S_IWGRP 122, 142, 302, 703, 891, 917, 1036
S_IWOTH 122, 142, 302, 703, 891, 917, 1036
S_IWRITE 122, 142, 302, 703, 891, 917, 1036
S_IWUSR 122, 142, 302, 703, 891, 917, 1036-1037
<table>
<thead>
<tr>
<th>Function</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>wcsxfrm</td>
<td>1010</td>
</tr>
<tr>
<td>wmemchr</td>
<td>615</td>
</tr>
<tr>
<td>_wsearchenv</td>
<td>820</td>
</tr>
<tr>
<td>search.h</td>
<td>30</td>
</tr>
<tr>
<td>_searchenv</td>
<td>820</td>
</tr>
<tr>
<td>Searching Functions</td>
<td>17</td>
</tr>
<tr>
<td>SECURITY_ATTRIBUTES</td>
<td>84</td>
</tr>
<tr>
<td>SEEK_CUR</td>
<td>295, 531</td>
</tr>
<tr>
<td>SEEK_END</td>
<td>295, 531</td>
</tr>
<tr>
<td>SEEK_SET</td>
<td>295, 531</td>
</tr>
<tr>
<td>segread</td>
<td>821, 403, 406, 408</td>
</tr>
<tr>
<td>_selectpalette</td>
<td>822</td>
</tr>
<tr>
<td>set_constraint_handler_s</td>
<td>823</td>
</tr>
<tr>
<td>_setmatherr</td>
<td>846</td>
</tr>
<tr>
<td>set_new_handler</td>
<td>850</td>
</tr>
<tr>
<td>_setactivepage</td>
<td>825</td>
</tr>
<tr>
<td>_setbkcolor</td>
<td>826</td>
</tr>
<tr>
<td>setbuf</td>
<td>827</td>
</tr>
<tr>
<td>setcharsize</td>
<td>43</td>
</tr>
<tr>
<td>_setcharsize</td>
<td>828, 43</td>
</tr>
<tr>
<td>_setcharsize_w</td>
<td>828</td>
</tr>
<tr>
<td>_setcharspacing</td>
<td>830</td>
</tr>
<tr>
<td>_setcharspacing_w</td>
<td>830</td>
</tr>
<tr>
<td>_setcliprgn</td>
<td>832, 322</td>
</tr>
<tr>
<td>setcolor</td>
<td>42</td>
</tr>
<tr>
<td>_setcolor</td>
<td>833, 42, 856</td>
</tr>
<tr>
<td>setenv</td>
<td>834, 37, 332-333</td>
</tr>
<tr>
<td>_setenv</td>
<td>834</td>
</tr>
<tr>
<td>setfillmask</td>
<td>42</td>
</tr>
<tr>
<td>_setfillmask</td>
<td>836, 42</td>
</tr>
<tr>
<td>_setfont</td>
<td>838, 44, 336, 712, 744, 799, 1042</td>
</tr>
<tr>
<td>_setgtextvector</td>
<td>840</td>
</tr>
<tr>
<td>setjmp</td>
<td>841, 30, 526</td>
</tr>
<tr>
<td>setjmp.h</td>
<td>30</td>
</tr>
<tr>
<td>_setlinestyle</td>
<td>42</td>
</tr>
<tr>
<td>_setlinestyle_w</td>
<td>842, 42</td>
</tr>
<tr>
<td>setlocale</td>
<td>844, 29, 929, 1010</td>
</tr>
<tr>
<td>setlogorg</td>
<td>869</td>
</tr>
<tr>
<td>_setmbcp</td>
<td>848, 950, 968, 975</td>
</tr>
<tr>
<td>setmode</td>
<td>849, 49</td>
</tr>
<tr>
<td>_setmode</td>
<td>849</td>
</tr>
<tr>
<td>_setpixel</td>
<td>852</td>
</tr>
<tr>
<td>_setpixel_w</td>
<td>852</td>
</tr>
<tr>
<td>_setplotaction</td>
<td>42</td>
</tr>
<tr>
<td>_setplotaction</td>
<td>853, 42</td>
</tr>
<tr>
<td>_settextalign</td>
<td>854, 43</td>
</tr>
<tr>
<td>_settextalign</td>
<td>854, 43</td>
</tr>
<tr>
<td>_settextcolor</td>
<td>856, 43, 714, 718, 833</td>
</tr>
<tr>
<td>_settextcursor</td>
<td>857, 355</td>
</tr>
<tr>
<td>_settextorient</td>
<td>43</td>
</tr>
<tr>
<td>_settextorient</td>
<td>858, 43</td>
</tr>
<tr>
<td>_settextpath</td>
<td>859</td>
</tr>
</tbody>
</table>

**Index**
spawn 896, 20-21, 30, 50, 150, 386, 389, 810, 1012
spawnl 896, 764, 897, 1012
spawnlp 896, 897
spawnlpe 896, 897
spawnv 896, 897
spawnvpe 896, 897
_splitpath 904
_splitpath2 904
_splitpath 902
sprintf 906, 105, 908, 1076
sprintf_s 908, 888
sqrt 910, 1122
srand 911, 789
SREGS 29
sscanf 912, 1080
sscanf_s 913, 1082
st_archivedID 301, 916
st_atime 301, 916
st_attr 301, 916
st_btime 301, 916
st_ctime 301, 916
st_dev 301, 916
st_gid 301, 916
st_inheritedRightsMask 301, 916
st_ino 301, 916
st_mode 301-302, 916-917
st_mtime 301, 916
st_nlink 301, 916
st_originatingNameSpace 301, 916
st_rdev 301, 916
st_size 301-302, 916-917
st_uid 301, 916
st_updatedID 301, 916
stackavail 915
__stacksize 34
stat 916, 31, 301-302, 916-917
__stat 916
__stat64 916, 301, 916, 918
__status87 919
stdarg.h 30
stdaux 18, 34, 227, 255, 372
STD_AUX_FILENO 255
stdbool.h 30
__STDC_CONSTANT_MACROS 30
__STDC_FORMAT_MACROS 29
__STDC_LIMIT_MACROS 30
stddef.h 30
stdout 18, 33, 35, 227, 249, 255, 320, 352-353, 372, 812
STDOUT_FILENO 255
stdint.h 30
stdio.h 30
stdlib.h 30
strcat 921
strcat_s 923, 923
strchr 925
strcmpl 926, 929
strcmpi 928
strcoll 929, 1010
strcpy 930, 778
strncpy_s 932, 932
strcspn 934
__strdate 936
__strdec 937
strdup 939, 1024
_stdstrdup 939, 778
Stream I/O Functions 18-19
__bprintf 105
__bwprintf 105
clearerr 127
fclose 226
fcloseall 227
fopen 230
feof 238
ferror 240
fflush 246
fgetc 248
fgetchar 249
__fgetchar 249
fgetpos 250
fgets 251
fgetwc 248
__fgetwchar 249
fgetws 251
flushall 264
fopen 269
fopen_s 271
fprintf 278
fprintf_s 279
fputc 281
fputchar 282
Index

_fputchar 282
fputs 283
fputwc 281
_fputwchar 282
fputws 283
fread 284
freopen 288
freopen_s 289
fscanf 292
fscanf_s 293
fseek 270, 272, 295, 299
fsetpos 297
_fsopen 298
_ftell 307
fwprintf 278
fwprintf_s 279
fwrite 312
fscanf 292
fscanf_s 293
getc 318
getchar 320
gets 352
_getw 366
getwc 318
getwchar 320
_getws 352
Multibyte Character Functions 19
perror 720
printf 767
printf_s 773
putc 775
putchar 777
puts 782
_putchar 783
putwc 775
putwchar 777
_putchar_s 782
rewind 804
scanf 812
scanf_s 818
setbuf 827
setvbuf 864
_snprintf_s 888
_snwprintf_s 888
_sprintf_s 908
_sscanf_s 913
_sscanf 913
_sscanf_s 913
_tmpfile 1020
tmpfile_s 1021
ungetc 246, 1038
ungetwc 1038
_vfprintf 1054
_vfprintf_s 1056
_vfscanf 1058
_vfscanf_s 1060
_vfwprintf 1054
_vfwprintf_s 1056
_vfwscanf 1058
_vfwscanf_s 1060
_vprintf 1062
_vprintf_s 1064
_vscanf 1066
_vscanf_s 1068
_vsnprintf_s 1074
_vsnprintf_s 1074
_vsscanf 1078
_vsscanf_s 1078
_vswprintf_s 1078
_vswscanf 1082
_vswscanf_s 1084
_vwprintf 1062
_vwprintf_s 1064
_vwscanf 1066
_vwscanf_s 1066
_setbuf 133
_ffs 247
_fmbsearch 925
_fmbscat 921
_fmbscmp 926
_fmbscpy 930
_fmbstrcat 934
_fmbstpcpy 937
_fmbstpos 939
_fmbstricmp 948

Wide Character Functions 19
_wprintf 767
_wprintf_s 773
_wscanf 812
_wscanf_s 818
_strerror 940, 32, 1127
_strerror_s 941, 943
_strerrorlen_s 943
_strftime 944, 38, 844
_stricoll 950
_stricmp 950
_stricmpi 951

String Functions 8
_bcmp 88
_bcopy 89
_bzero 112
__cmdname 133
_ffs 247
_fmbsearch 925
_fmbsearch 925
_fmbscat 921
_fmbstrcat 934
_fmbstpcpy 937
_fmbstpos 939
_fmbstricmp 948

1153
_stricmp  948
_stricoll  950
_strinc  951
strlcat  954
strlcpy  955
strlen  956
strlwr  959
_strlwr  959
strncasecmp  961
strncat  962
strncat_s  964
strncmp  966
_strncmp  588, 594
_strnccoll  968
strncpy  969
_strnccpy  971
_strnextc  596
strnicmp  973
_strnicmp  973
_strnicoll  975
_strninc  976
strnlen_s  958
strnset  979
_strnset  979
strpbrk  981
strrchr  983
strrev  985
_strrev  985
strset  987
_strset  987
strspn  989
_strspnp  991
_strspnp  991
strstr  993
strtok  998
strtok_s  1000
strupr  1008
_trupr  1008
strxfrm  1010
swprintf  906
swscanf  912
_vbprintf  1051
_vbwprintf  1051
_vsnprintf  1072
_vsnprintf  1070
_vsnwprintf  1072
_vsnwprintf  1070
vprintf  1076
vscanf  1080
_vswprintf  1076
_vswscanf  1080
wcscat  921
wcs cat_s  923
wcschr  925
wcs cmp  926
wcsncpy  928
wcs coll  929
wcscpy  930
wcscpy_s  932
wcscspn  934
wcsdec  937
wcsdup  939
wceserror  940
wceserror_s  941
wceserrorlen_s  943
wcsicmp  948
wcsicoll  950
wcsinc  951
wcslcat  954
wcslcpy  955
wcslen  956
wcs lwr  959
wcesncat  962
wcesncat_s  964
wcesncmp  966
wcsncoll  984, 588, 594
wcsncoll  968
wcesncoll  968
wcesncpy  969
wcesncpy_s  971
wcs nextrc  596
wcesnicmp  973
wcesnicoll  975
wcesninc  976
wcsnlen_s  958
wcsnsset  979
wcespbrk  981
wcesrchr  983
wcesrev  985
wcsset  987
wcspn  989
wcsppp  991
wcesstr  993
wcestok  998
wcestok_s  1000
wcesupr  1008
wcsxfrm  1010
wmemset  626
string.h  31
strlcat  954
strlcpy  955
strlen  956
str lwr  959, 13
_strlwr  959
strncasecmp  961
strncat  962
strncat_s  964
strn cmp  966, 1010
_strncnt  588, 594
Index

_strncoll 968
strncpy 969, 1010
strncpy_s 971, 971
strnecxc 956
strnicmp 973, 961
strnicmp 973
_strnicoll 975
_strnicoll 976
strnlen_s 958
strnset 979
_strnset 979
strpbrk 981
strchr 983
strstr 985
_strstr 985
_strset 987
_strset 987
strspn 989
strspnp 991
_strspnp 991
strstr 993
_strtime 995
strtoimax 1004
strtok 998
strtok_s 1000
strtol 1002
strtol 1003
strtooul 1005
strtooull 1006
strtooull 1007
struct 700
lconv 513
tm 31, 635
structure
complex 30
DOSERROR 29
exception 30
INTPACK 29
_m64 30
REGPACK 29
REGS 29
SREGS 29
stat 31
strupr 1008, 13
_trupr 1008
strxfrm 1010
SVGA 361, 866
SVRES16COLOR 865
SVRES256COLOR 865
SW_HIDE 539
SW_MINIMIZE 539
SW_RESTORE 539
SW_SHOW 539
SW_SHOWMAXIMIZED 539
SW_SHOWMINIMIZED 539
SW_SHOWMINNOACTIVE 539
SW_SHOWNA 539
SW_SHOWNOACTIVATE 539
SW_SHOWNORMAL 539
swab 1011
swprintf 906
swprintf_s 908
swscanf 912
swscanf_s 913
sys 31
sys\locking.h 31
sys\stat.h 31
sys\timeb.h 31
sys\types.h 31
sys\utime.h 31
sys\time.h 32
sys\errlist 35
sys\nerr 35
system 1012, 20, 30, 50, 386, 389, 1127

T

tan 1013
tanh 1014
_tcsnbcnt 588
_tcsnccnt 594
_tcsnextc 596
_tcsninc 976
tell 1015, 531, 790, 1115
tell 1015
_tell64 1015
TEMP 1020-1021
TEMPDIR 1020-1021
_temppnam 1017
Terminate and Stay Resident
_dos_keep 181
text files 33
TEXTBW40 865
TEXTBW80 865
TEXTC40 865
TEXTC80 865
TEXTMONO 865
_threadid 35, 84
__threadid 84
time 1019, 145, 516, 635
Time Functions 17, 31
asctime 64
__asctime 64
asctime_s 66
clock 129
time 145
_ctime 145
ctime_s 147
dftime 154
ftime 308
gmtime 368
_gmtime 368
_gmtime_s 370
localtime 516
__localtime 516
localtime_s 518
mktime 635
strftime 944
time 1019
__time 64
__wasctime 64
__wasctime_s 66
wcsftime 944
_wctime 145
__wctime 145
_wctime_s 147
__wstrftime_ms 944
time zone 36, 145, 368, 516, 1019, 1030
time.h 31
_TIME_GETCLOCK 104
_TIME_SETCLOCK 104
time_t 1019
timeb 31
timezone 35, 37, 1030
TLOSS 546, 846
tm 31, 368, 516
tm_hour 635
tm_isdst 635
tm_mday 635
tm_min 635
tm_mon 635
tm_sec 635
tm_wday 635
tm_yday 635
TMP 1017, 1020-1021
TMP_MAX 1017, 1024
TMP_MAX_S 1022
TMPDIR 1020-1021
tmpfile 1020, 220-221
tmpfile_s 1021
tmpnam 1024
tmpnam_s 1022
tolower 1025, 13, 959, 1111
__tolower 1025
TOP 854
toupper 1027, 13, 1008, 1111
_toupper 1027
towctrans 1029, 1111
towlower 1025, 13
towupper 1027, 13
TR 24731 classification 52
Trigonometric Functions 15
acos 59
acosh 60
asin 69
asinh 70
atan 72
atan2 73
atanh 74
cos 138
cosh 139
_dos_mallocmem 160
hypot 394
sin 878
sinh 879
tan 1013
tanh 1014
true 30
TZ 36-38, 126, 145, 368, 516, 1019, 1030
tzname 35, 37-38, 1030
tzset 1030, 32, 35, 37-38, 145, 516, 635, 946
U

UINTMAX_MAX 1007
uintmax_t 770, 814
ULLONG_MAX 1006
ulltoloa 1032
__ulltoloa 1032
__ulltow 1032
ULONG_MAX 1005
ultoa 1034
__ultoa 1034
__ultow 1034
umask 1036, 703, 891
__umask 1036
undefined references
fltused_ 32
UNDERFLOW 546, 846
ungetc 1038, 246, 295
ungetch 1039
ungetwc 1038
__UNICODE 588, 594, 596, 976
union 700
UNKNOWN 361
unlink 1040
__unlink 1040
unlock 1041, 520
Index

_unregisterfonts 1042
unsigned 850
URES256COLOR 865
UTC 36-37
utimbuf 32, 1043
utime 1043, 32
_utime 1043
_utoa 1045
_utoa 1045
_utow 1045

V

va_arg 1047, 1049-1050, 1060, 1068, 1082
va_end 1049, 1047, 1050, 1060, 1068, 1082
va_list 1047
va_start 1050, 1047, 1049, 1051-1054, 1058,
1060, 1062, 1066, 1068, 1070, 1072, 1076,
1080, 1082
varargs.h 31
variable arguments 17
__va_arg 1047
__va_end 1049
__va_start 1050
__vbprintf 1051
__vbwprintf 1051
vcprintf 1052
vcscnaf 1053
vfprintf 1054
vfprintf_s 1056
vfscanf 1058
vfscanf_s 1060
vfwprintf 1054
vfwprintf_s 1056
vfwsscanf 1058
vfwsscanf_s 1060
VGA 361, 865
view coordinates 41
void 538, 850
vprintf 1062, 1056, 1064
vprintf_s 1064
VRES16COLOR 865
VRES256COLOR 865
VRES2COLOR 865
vscanf 1066
vscanf_s 1068
vsnprintf 1072, 1074
_vsnprintf 1070
_vsnprintf_s 1074, 1078
vsnwprintf 1074, 1078
v_snwprintf 1072

W

W_OK 57
_waccess 57
wait 1084, 896, 898
WAIT_CHILD 150
WAIT_GRANDCHILD 150
__wargc 35
__wargv 35
__wasctime 64, 64
__wasctime 64, 64
__wasctime_s 66
WATCOM classification 52
wchar.h 31
wchar_t 31, 615-618, 621, 626, 770, 998
__wchdir 119
__wchmod 122
__wclosedir 131, 706
__wcreat 142
wcrtomb 1087, 1093, 1096
wcrtomb_s 1090, 1090
wcschr 925
wcschr 925
wcschrm 928
wcescp 930
wcescp 930
wcescp 932
wcescp 934
_wcsdec 937
_wcsdup 939
wcerror 940
wcerror_s 941
wcerrorlen_s 943
wcsftime 944
_wcsicmp 948
_wcsicoll 950
_wcsinc 951
wcsleap 955
wcslen 956
_wcslenw 959, 13
wcsncat 962
wcsncat_s 964
wcsncmp 966
_wcsnent 588, 594
_wcsncoll 968
wcsncpy 969
wcsncpy_s 971
_wcsnextr 596
_wcsnicmp 973
_wcsnicoll 975
_wcsnicnt 976
wcsnlen_s 958
_wcsnsset 979
wcsprbkr 981
wcsrchr 983
_wcsrev 985
wcsrtoombs 1093
wcsrtoombs_s 1096
_wcsset 987
wcsstr 993
wcestod 996
wcestoimax 1004
wcestok 998
wcestok_s 1000
wcestol 1002
wcestoll 1003
wcsstombs 1100, 598, 1093
wcsstombs_s 1102
wcestoul 1005
wcestoull 1006
wcestoumax 1007
_wcsupr 1008, 13
wcesxfrm 1010
_wctime 145, 145
_wctime 145, 145
_wctime_s 147
wctob 1105
wcstomb 1107, 576, 579, 1087
wcstomb_s 1109
wcstrans 1110, 1029
wcstrans_t 1111
wcctype 1112, 498
wcctype.h 31
wcctype_t 31, 1112
_wdirent 706, 793
_wdos_findclose 168, 169
_wdos_findfirst 168, 169
_wdos_findnext 168, 169
_wecvt 208
_wenviron 35
WEOF 31, 248-249, 281-283, 318, 320, 775, 777, 782, 1038
_wexecl 216, 217
_wexeclp 216, 217
_wexeclpe 216, 217
_wexecv 216, 217
_wexecve 216, 217
_wexecvp 216, 217
_wexecvpe 216, 217
_wfexist 228
_wfdopen 230
_wfindfirst 257
_wfindfirst 257
_wfindnext 259
_wfindnext 259
_wfopen 269
_wfopen_s 271
_wfreopen 288
_wfreopen_s 289
_wfsopen 298
_wfstat 301, 303
_wfstat6 301, 303
_wfutpath 309
_wgetcwd 326
_wgetdcwd 328
_wgetenv 332
Wide Character Functions 6, 10-11
_btowc 111
_bwprintf 105
_fgetwc 248
_fgetwchar 249
_fgetws 251
_fmbspbrk 981
_fmbstok 998
_fputwc 281
_fputwchar 282
_fputws 283
_fwcrtomb 1087
_fwcrtomb 1090
_fwcsrtombs 1093
_fwcstombs 1096
_fwctombs 1100
_fwcsrtombs_s 1109
_fwcsctomb 1107
_fwcstombs_s 1109
fwprintf 278
fwprintf_s 279
fwscanf 292
fwscanf_s 293
getwc 318
getwchar 320
_getws 352
iswalnum 411
iswascii 413
iswblank 416
iswcntrl 418
__iswcsym 419
__iswcsymf 421
iswdigit 423
iswgraph 425
iswlower 429
iswprint 492
iswpunct 493
iswspace 495
iswupper 497
iswxdigit 500
_itow 501
_1ltow 534
_1tow 536
_mbstrbrk 981
_mbstr 998
putwc 775
putwchar 777
_putws 782
snwprintf_s 888
swprintf_s 908
swscanf 912
swscanf_s 913
towcstr 1029
towlower 1025
towupper 1027
_u1ltow 1032
_ultow 1034
ungetwc 1038
utime 1043
_utow 1045
__ybwprintf 1051
vfwprintf 1054
vfwprintf_s 1056
vfscanf 1058
vfscanf_s 1060
vsnprintf 1072
_vsnwprintf 1070
vsnprintf_s 1074
vswprintf 1076
vswprintf_s 1078
vscanf 1080
vscanf_s 1082
vprintf 1062
vprintf_s 1064
vwprintf 1066
vwscanf 1068
_waccess 57
_wasctime 64
__wasctime 64
_wasctime_s 66
_wchdir 119
__wcmod 122
_wcscat 131
_wcreate 142
wctomb 1087
wctomb_s 1090
wcsca 921
wcsca_s 923
wcschr 925
wcscompare 926
wcscompare 928
wcest 929
wcesp 930
wcesp_s 932
wcesp 934
_wcsdec 937
wcsdup 939
wcserror 940
wcserror_s 941
wcserrorlen_s 943
wcsftime 944
__wcscmp 948
__wcscoll 950
_wcsinc 951
wcslen 954
wcssp 955
wcslen 956
__wcslen 959
wcesc 962
wcesc_s 964
wcesp 966
_wcesncnt 588, 594
_wcesncoll 968
wcesp 969
wcesp_s 971
__wcsnexit 596
__wcscmp 973
__wcscoll 975
_wcsinc 976
wcslen_s 958
_wcsnet 979
wcsbox 981
wcschr 983
__wcsrev 985
wcesrev 1093
wctomb_s 1096
_wcsset 987
wcesp 989
Index

_wcspnp 991
wcsstr 993
wcestod 996
wcestoimax 1004
wcestok 998
wcestok_s 1000
wcestol 1002
wcestoll 1003
wcestombs 1100
wcestombs_s 1102
wcestoul 1005
wcestoull 1006
wcestoumax 1007
__wcsupr 1008
wcsxfm 1010
__wctime 145
__wctime 145
__wctime_s 147
wctob 1105
wctomb 1107
wctomb_s 1109
wctrans 1111
wctype 1112
__wdos_findclose 168
__wdos_findfirst 168
__wdos_findnext 168
__wexecl 216
__wexecl 216
__wexeclp 216
__wexeclpe 216
__wexecv 216
__wexecve 216
__wexecvp 216
__wexecvpe 216
__wfopen 269
__wfopen 269
__wfreopen 288
__wfreopen_s 289
__wfsopen 298
__wfsopen 298
__wfsstat 301
__wfsat64 301
__wfullpath 309
__wgetcwd 326
__wgetcwd 328
__wgetenv 332
__wmakepath 542
__wmkdir 629
__wmktemp 633
__wopen 702
__wopendir 705
__w perror 720
__wpopen 764
wpprintf 767
wpprintf_s 773
__wputenv 778
__wreaddir 792
__w remove 802
__w rename 803
__wreaddir 805
__w remove 807
wscanf 812
wscanf_s 818
__w searchenv 820
__wsetenv 834
__wsetlocale 844
__wspawnl 896
__wspawnle 896
__wspawnlp 896
__wspawnlpe 896
__wspawnv 896
__wspawnve 896
__wspawnvpe 896
__w splitpath 2 904
__w splitpath 2 902
__w stat 916
__w stat64 916
__w strdate 936
__w strftime_ms 944
__w strftime 995
__w system 1012
__w tmpnam 1017
__w tmpnam 1024
__w tmpnam_s 1022
__wtof 76
__wtoi 77
__wtol 78
__w toll 79
__w unlink 1040
Win32 Functions
__beginthread 83
__beginthreadex 83
cwait 149
__endthread 213
__endthreadex 213
wait 1084
__win_alloc_flags 36
__win_realloc_flags 36
window coordinates 41
Windows classification 52
WinMain 538, 538-539
__winmajor 36
__winminor 36
Index

wint_t 31
_winver 36
WM_QUIT 539
wmain 538, 35
__wmakepath 542
wmemchr 615
wmemcmp 616
wmemcpy 617
wmemcpy_s 618
wmemmove 621
wmemmove_s 622
wmemset 626
__wmkdir 629
__wmktemp 633
__wopen 702
__wopendir 705, 131, 805
__wP_tmpdir 1017
wParam 539
__werror 720
__wpopen 764
wprintf 767, 769-770
wprintf_s 773
__wputenv 778
wr_date 169
wr_time 169
__wrapton 1114
__wreaddir 792, 706
__wrename 803
__wreaddir 805
write 1115
__write 1115
__wrmdir 807
wscanf 812
wscanf_s 818
__wsearchenv 820
__wsetenv 834
__wsetlocale 844
__wsopen 890
__wspawnl 896, 898
__wspawnle 896, 898
__wspawnlp 896, 898
__wspawnlpe 896, 898
__wspawnve 896, 898
__wspawnv 896, 898
__wspawnvp 896, 898
__wspawnvpe 896, 898
__wsplitpath 904
__wsplitpath2 904
__wstat 916
__wstat64 916, 918
__wstrdate 936
__wstrntime_ms 944
__wstrtime 995

__wsystem 1012
__wtempnam 1017
__wtmpnam 1024
__wtmpnam_s 1022
__wtol 76
__wtol 78
__wtol 79
__wutime 1043
wWinMain 538, 538

X

X_OK 57
XRES16COLOR 865
XRES256COLOR 865

Y

y0 87, 87, 1122
y1 87, 87, 1122
yn 87, 87, 1122

Z

zentohan 570