Global Variables

Borland C++Builder provides you with predefined global variables for many common needs, such as dates, times, command-line arguments, and so on.

_8087
_argc
_argv
_ctype
_daylight
_doserrno
_environ
_errno
_floatconvert
_fmode
_new_handler
_osmajor

_osminor
_osversion
_sys_errlist
_sys_nerr
_threadid
__throwExceptionName
__throwFileName
__throwLineNumber
_timezone
_tzname
_wtzname
_version

_8087

Portability Global Variables

Syntax

```
extern int _8087;
```

Header File

dos.h

Description

The *_8087* variable is set to a nonzero value if the startup code autodetection logic detects a floating-point coprocessor.

_8087 Value	Math Coprocessor
1	*8087*
2	80287
3	80387
0	(none detected)

The autodetection logic can be overridden by setting the 87 environment variable to YES or NO. (The commands are `SET 87=YES` and `SET 87=NO`; it is essential that there be no spaces before or after the equal sign.) In this case, the *_8087* variable will reflect the override.

_argc

Portability   Example   Global Variables

Syntax

```
extern int _argc;
```

Header File

dos.h

Description

_*argc* has the value of *argc* passed to *main* when the program starts.

/* _argc and _argv example */

```
#include <iostream.h>
#include <dos.h>     // TO GET THE GLOBAL _arg VALUES

void func() {
   cout << "argc= " << _argc << endl;

   for (int i = 0; i < _argc; ++i)
      cout << _argv[i] << endl;
   }

void main(int argc, char ** argv) {
   func(); // THIS FUNCTION KNOWS ALL THE main() ARGUMENTS
   }
```

_argv, _wargv

Portability Example Global Variables

Syntax

```
extern char **_argv;
extern wchar_t ** _wargv
```

Header File

dos.h

Description

_*argv* points to an array containing the original command-line arguments (the elements of *argv[]*) passed to *main* when the program starts.

_*wargv* is the Unicode version of _*argv*.

_ctype

Portability   Global Variables

Syntax

```
extern char _ctype[];
```

Header File

ctype.h

Description

_*ctype* is an array of character attribute information indexed by ASCII value + 1. Each entry is a set of bits describing the character. This array is used by isdigit, isprint, and so on.

_daylight

See also Portability Global Variables

Syntax

```
extern int _daylight;
```

Header File

time.h

Description

_*daylight* is used by the time and date functions. It is set by the tzset, ftime, and localtime functions to 1 for daylight saving time, 0 for standard time.

On Win32, the value of _*daylight* is obtained from the operating system.

_environ, _wenviron

See also Portability Global Variables

Syntax

```
extern char ** _environ;
extern wchar_t ** _wenviron
```

Header File

dos.h

Description

_environ is an array of pointers to strings; it is used to access and alter the operating system environment variables. Each string is of the form:

```
envvar = varvalue
```

where *envvar* is the name of an environment variable (such as PATH), and *varvalue* is the string value to which *envvar* is set (such as C:\BIN;C:\DOS). The string *varvalue* can be empty.

When a program begins execution, the operating system environment settings are passed directly to the program. Note that *env*, the third argument to *main*, is equal to the initial setting of *_environ*.

The *_environ* array can be accessed by getenv; however, the putenv function is the only routine that should be used to add, change or delete the *_environ* array entries. This is because modification can resize and relocate the process environment array, but *_environ* is automatically adjusted so that it always points to the array.

errno

Portability   Example   Global Variables

Syntax

```
extern int errno;
```

Header File

errno.h

Description

errno is used by perror to print error messages when certain library routines fail to accomplish their appointed tasks.

When an error in a math or system call occurs, *errno* is set to indicate the type of error. Sometimes *errno* and _doserrno are equivalent. At other times, *errno* does not contain the actual operating system error code, which is contained in *_doserrno*. Still other errors might occur that set only *errno*, not *_doserrno*.

/* errno, _doserrno, _sys_errlist, and _sys_nerr example */

```
/* DISPLAY THE SYSTEM ERRORS. */
#include <errno.h>
#include <stdio.h>

extern char *_sys_errlist[];

main()
{
   int i = 0;

   while(_sys_errlist[i++]) printf("%s\n", _sys_errlist[i]);
   return 0;
}
```

_doserrno

Portability   Example   Global Variables

Syntax

```
extern int _doserrno;
```

Header File

errno.h

Description

_doserrno is a variable that maps many operating system error codes to errno; however, perror does not use _doserrno directly.

When an operating system call results in an error, *_doserrno* is set to the actual operating system error code. *errno* is a parallel error variable inherited from UNIX.

The following list gives mnemonics for the actual DOS error codes to which *_doserrno* can be set. (This value of *_doserrno* may or may not be mapped (through *errno*) to an equivalent error message string in _sys_errlist.

Mnemonic	DOS error code
E2BIG	Bad environ
EACCES	Access denied
EACCES	Bad access
EACCES	Is current dir
EBADF	Bad handle
EFAULT	Reserved
EINVAL	Bad data
EINVAL	Bad function
EMFILE	Too many open
ENOENT	No such file or directory
ENOEXEC	Bad format
ENOMEM	Mcb destroyed
ENOMEM	Out of memory
ENOMEM	Bad block
EXDEV	Bad drive
EXDEV	Not same device

Refer to your DOS reference manual for more information about DOS error return codes.

_sys_errlist

Portability Example Global Variables

Syntax

```
extern char * _sys_errlist[ ];
```

Header File

errno.h

Description

_sys_errlist is used by perror to print error messages when certain library routines fail to accomplish their appointed tasks.

To provide more control over message formatting, the array of message strings is provided in *_sys_errlist*. You can use *errno* as an index into the array to find the string corresponding to the error number. The string does not include any newline character.

The following table gives mnemonics and their meanings for the values stored in *_sys_errlist*. The list is alphabetically ordered for ease your reading convenience. For the numerical ordering, see the header file errno.h.

Mnemonic	16-bit Description	32-bit Description
E2BIG	Arg list too long	Arg list too long
EACCES	Permission denied	Permission denied
EBADF	Bad file number	Bad file number
ECHILD		No child process
ECONTR	Memory blocks destroyed	Memory blocks destroyed
ECURDIR	Attempt to remove CurDir	Attempt to remove CurDir
EDEADLOCK		Locking violation
EDOM	Domain error	Math argument
EEXIST	File already exists	File already exists
EFAULT	Unknown error	Unknown error
EINTR		Interrupted function call
EINVACC	Invalid access code	Invalid access code
EINVAL	Invalid argument	Invalid argument
EINVDAT	Invalid data	Invalid data
EINVDRV	Invalid drive specified	Invalid drive specified
EINVENV	Invalid environment	Invalid environment
EINVFMT	Invalid format	Invalid format
EINVFNC	Invalid function number	Invalid function number
EINVMEM	Invalid memory block address	Invalid memory block address
EIO		input/output error
EMFILE	Too many open files	Too many open files
ENAMETOOLONG		File name too long
ENFILE		Too many open files
ENMFILE	No more files	No more files
ENODEV	No such device	No such device
ENOENT	No such file or directory	No such file or directory
ENOEXEC	Exec format error	Exec format error
ENOFILE	No such file or directory	File not found
ENOMEM	Not enough memory	Not enough core
ENOPATH	Path not found	Path not found
ENOSPC		No space left on device
ENOTSAM	Not same device	Not same device
ENXIO		No such device or address
EPERM		Operation not permitted
EPIPE		Broken pipe
ERANGE	Result out of range	Result too large
EROFS		Read-only file system

ESPIPE		Illegal seek
EXDEV	Cross-device link	Cross-device link
EZERO	Error 0	Error 0

Refer to your DOS reference manual for more information about DOS error return codes.

_sys_nerr

Portability   Example   Global Variables

Syntax

```
extern int _sys_nerr;
```

Header File

errno.h

Description

_sys_nerr is used by perror to print error messages when certain library routines fail to accomplish their appointed tasks.

This variable is defined as the number of error message strings in _sys_errlist.

_floatconvert

Portability   Example   Global Variables

Syntax

```
extern int _floatconvert;
```

Header File

stdio.h

Description

Floating-point output requires linking of conversion routines used by *printf*, *scanf*, and any variants of these functions. In order to reduce executable size, the floating-point formats are not automatically linked. However, this linkage is done automatically whenever your program uses a mathematical routine or the address is taken of some floating-point number. If neither of these actions occur, the missing floating-point formats can result in a run-time error.

/* _floatconvert example */

```
/* PREPARE TO OUTPUT FLOATING-POINT NUMBERS. */
#include <stdio.h>
#pragma extref _floatconvert

void main() {
  printf("d = %lf\n", 1);
}
```

_fmode

Portability   Global Variables

Syntax

```
extern int _fmode;
```

Header File

fcntl.h

Description

_fmode determines in which mode (text or binary) files will be opened and translated. The value of *_fmode* is O_TEXT by default, which specifies that files will be read in text mode. If *_fmode* is set to O_BINARY, the files are opened and read in binary mode. (O_TEXT and O_BINARY are defined in fcntl.h.)

In text mode, carriage-return/linefeed (CR/LF) combinations are translated to a single linefeed character (LF) on input. On output, the reverse is true: LF characters are translated to CR/LF combinations.

In binary mode, no such translation occurs.

You can override the default mode as set by *_fmode* by specifying a *t* (for text mode) or *b* (for binary mode) in the argument *type* in the library functions *fopen, fdopen*, and *freopen*. Also, in the function *open*, the argument access can include either O_BINARY or O_TEXT, which will explicitly define the file being opened (given by the *open pathname* argument) to be in either binary or text mode.

_new_handler

[Portability](#) [Global Variables](#)

Syntax

```
typedef void (*pvf)();
pvf _new_handler;
```

Header File

new.h

Description

_new_handler contains a pointer to a function that takes no arguments and returns **void**. If **operator new()** is unable to allocate the space required, it will call the function pointed to by *_new_handler*; if that function returns it will try the allocation again. By default, the function pointed to by *_new_handler* simply terminates the application. The application can replace this handler, however, with a function that can try to free up some space. This is done by assigning directly to *_new_handler* or by calling the function set_new_handler,, which returns a pointer to the former handler.

As an alternative, you can set using the function *set_new_handler*, like this:

```
pvf set_new_handler(pvf p);
```

_new_handler is provided primarily for compatibility with C++ version 1.2. In most cases this functionality can be better provided by overloading **operator new()**.

_osmajor

See also   Portability   Global Variables

Syntax

```
extern unsigned char _osmajor;
```

Header File

dos.h

Description

The major version number of the operating system is available individually through *_osmajor*. For example, if you are running DOS version 3.2, *_osmajor* will be 3.

This variable can be useful when you want to write modules that will run on DOS versions 2.x and 3.x. Some library routines behave differently depending on the DOS version number, while others only work under DOS 3.x and higher. For example, refer to creatnew and _rtl_open.

_osminor

See also   Portability   Global Variables

Syntax

```
extern unsigned char _osminor;
```

Header File

dos.h

Description

The minor version number of the operating system is available individually through *_osminor*. For example, if you are running DOS version 3.2, *_osminor* will be 20.

This variables can be useful when you want to write modules that will run on DOS versions 2.x and 3.x. Some library routines behave differently depending on the DOS version number, while others only work under DOS 3.x and higher. For example, refer to creatnew and _rtl_open.

_osversion

See also Portability Global Variables

Syntax

```
extern unsigned _osversion;
```

Header File

dos.h

Description

_*osversion* contains the operating system version number, with the major version number in the low byte and the minor version number in the high byte. (For DOS version x.y, the *x* is the major version number, and *y* is the minor version number.)

_*osversion* is functionally identical to _version.

_ _throwExceptionName

See also   Portability   Global Variables

Syntax

```
extern char * _ _throwExceptionName;
```

Header File

except.h

Description

Use this global variable to get the name of a thrown exception. The output for this variable is a printable character string.

_ _throwFileName

See also Portability Global Variables

Syntax

```
extern char * _ _throwFileName;
```

Header File

except.h

Description

Use this global variable to get the name of a thrown exception. The output for this variables is a printable character string.

To get the file name for a thrown exception with *__throwFileName*, you must compile the module with the `-xp` compiler option.

_ _throwLineNumber

See also   Portability   Global Variables

Syntax

```
extern char * _ _throwLineNumber;
```

Header File

except.h

Description

Use this global variable to get the name of a thrown exception. The output for this variables is a printable character string.

To get the line number for a thrown exception with *__throwLineNumber*, you must compile the module with the `-xp` compiler option.

_threadid

Portability Global Variables

Syntax

```
extern long _threadid;
```

Header File

stddef.h

Description

_threadid is a long integer that contains the ID of the currently executing thread. It is implemented as a macro, and should be declared only by including stddef.h.

_timezone

See also Portability Global Variables

Syntax

```
extern long _timezone;
```

Header File

time.h

Description

_*timezone* is used by the time-and-date functions. It is calculated by the tzset function; it is assigned a long value that is the difference, in seconds, between the current local time and Greenwich mean time.

On Win32, the value of _*timezone* is obtained from the operating system.

_tzname, _wtzname

See also   Portability   Global Variables

Syntax

```
extern char * _tzname[2]
extern wchar_t *const _wtzname[2]
```

Header File

time.h

Description

The global variable *_tzname* is an array of pointers to strings containing abbreviations for time zone names. *_tzname*[0] points to a three-character string with the value of the time zone name from the *TZ* environment string. The global variable *_tzname*[1] points to a three-character string with the value of the daylight saving time zone name from the *TZ* environment string. If no daylight saving name is present, *_tzname*[1] points to a null string.

On Win32, the value of *_tzname* is obtained from the operating system.

_version

[See also](#)   [Portability](#)   [Global Variables](#)

Syntax

```
extern unsigned _version;
```

Header File

dos.h

Description

_version contains the operating system version number, with the major version number in the low byte and the minor version number in the high byte. (For DOS version *x.y*, the *x* is the major version number, and *y* is the minor.)

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+			+			+

The main() Function

See also

Every C and C++ program must have a program-startup function.

- Console-based programs call the *main* function at startup.
- Windows GUI programs call the WinMain function at startup.

Where you place the startup function is a matter of preference. Some programmers place *main* at the beginning of the file, others at the end. Regardless of its location, the following points about *main* always apply.

- Arguments to main
- Wildcard Arguments
- Using -p (Pascal Calling Conventions)
- Value main( ) Returns

Arguments to main ()

The main() Function Example

Three parameters (arguments) are passed to *main* by the Borland C++ startup routine: *argc, argv*, and *env*.

- *argc*, an integer, is the number of command-line arguments passed to *main*, including the name of the executable itself.
- *argv* is an array of pointers to strings (**char** *[]).
  - *argv*[0] is the full path name of the program being run.
  - *argv*[1] points to the first string typed on the operating system command line after the program name.
  - *argv*[2] points to the second string typed after the program name.
  - *argv*[*argc*-1] points to the last argument passed to *main*.
  - *argv*[*argc*] contains NULL.
- *env* is also an array of pointers to strings. Each element of *env*[] holds a string of the form ENVVAR=value.
  - ENVVAR is the name of an environment variable, such as PATH or COMSPEC.
  - *value* is the value to which ENVVAR is set, such as C:\APPS;C:\TOOLS; (for PATH) or C:\DOS\COMMAND.COM (for COMSPEC).

If you declare any of these parameters, you *must* declare them exactly in the order given: *argc, argv, env*. For example, the following are all valid declarations of arguments to *main*:

```
int main()
int main(int argc)                  /* legal but very unlikely */
int main(int argc, char * argv[])
int main(int argc, char * argv[], char * env[])]
```

The declaration int main(int argc) is legal, but it is very unlikely that you would use *argc* in your program without also using the elements of *argv*.

The argument *env* is also available through the global variable _environ..

For all platforms, *argc* and *argv* are also available via the global variables _argc and _argv.

Example of how Arguments are Passed to main( )

Here is an example that demonstrates a simple way of using these arguments passed to main:

```
/* Program ARGS.C */
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[], char *env[]) {
   int i;

   printf("The value of argc is %d \n\n", argc);
   printf("These are the %d command-line arguments passed to"
          " main:\n\n", argc);

   for (i = 0; i < argc; i++)
      printf("   argv[%d]: %s\n", i, argv[i]);

   printf("\nThe environment string(s) on this system are:\n\n");

   for (i = 0; env[i] != NULL; i++)
      printf("   env[%d]: %s\n", i, env[i]);
   return 0;
   }
```

Suppose you run ARGS.EXE at the command prompt with the following command line:

```
   C:> args first_arg "arg with blanks" 3  4 "last but one" stop!
```

Notice that you can pass arguments with embedded blanks by surrounding them with quotes, as shown by "argument with blanks" and "last but one" in this example command line.

The output of ARGS.EXE (assuming that the environment variables are set as shown here) would then be like this:

```
The value of argc is 7

These are the 7 command-line arguments passed to main:

   argv[0]: C:\BC5\ARGS.EXE
   argv[1]: first_arg
   argv[2]: arg with blanks
   argv[3]: 3
   argv[4]: 4
   argv[5]: last but one
   argv[6]: stop!

The environment string(s) on this system are

   env[0]: COMSPEC=C:\COMMAND.COM
   env[1]: PROMPT=$p $g
   env[2]: PATH=C:\SPRINT;C:\DOS;C:\BC5
```

The maximum combined length of the command-line arguments passed to *main* (including the space between adjacent arguments and the program name itself) is

- 128 for DOS
- 260 for Win16
- 255 for Win32

Wildcard Arguments

The main() Function Example

Command-line arguments containing wildcard characters can be expanded to all the matching file names, much the same way DOS expands wildcards when used with commands like COPY. All you have to do to get wildcard expansion is to link your program with the WILDARGS.OBJ object file, which is included with Borland C++.

Note: Wildcard arguments are used only in console-mode applications.

Once WILDARGS.OBJ is linked into your program code, you can send wildcard arguments (such as *.*) to your *main* function. The argument will be expanded (in the *argv* array) to all files matching the wildcard mask. The maximum size of the *argv* array varies, depending on the amount of memory available in your heap.

If no matching files are found, the argument is passed unchanged. (That is, a string consisting of the wildcard mask is passed to *main*.)

Arguments enclosed in quotes ("...") are not expanded.

Example of using Wildcard Arguments with main( )

The following commands compile the file ARGS.C and link it with the wildcard expansion module WILDARGS.OBJ, then run the resulting executable file ARGS.EXE:

```
BCC  ARGS.C  WILDARGS.OBJ
ARGS  C:\BC5\INCLUDE\*.H  "*.C"
```

When you run ARGS.EXE, the first argument is expanded to the names of all the *.H files in the INCLUDE directory. Note that the expanded argument strings include the entire path. The argument *.C is not expanded because it is enclosed in quotes.

If you prefer the wildcard expansion to be the default, modify your standard CW32?.LIB library files to have WILDARGS.OBJ linked automatically. To do so, remove SETARGV and INITARGS from the libraries and add WILDARGS. The following commands invoke the Turbo librarian (TLIB) to modify all the standard library files (assuming the current directory contains the standard C and C++ libraries and WILDARGS.OBJ):

Window Users

```
tlib  CW32          -setargv  +wildargs
tlib  CW32MT        -setargv  +wildargs
tlib  -setargv      +wildargs
```

Using --p (Pascal Calling Conventions)

The main() Function

If you compile your program using Pascal calling conventions, you must remember to explicitly declare *main* as a C type. Do this with the **_ _cdecl** keyword, like this:

```
int  _ _cdecl main(int argc, char* argv[], char* envp[])
```

The Value main() Returns

The main() Function

The value returned by *main* is the status code of the program: an **int**. If, however, your program uses the routine exit (or _exit) to terminate, the value returned by *main* is the argument passed to the call to *exit* (or to _*exit*).

For example, if your program contains the call

```
exit(1)
```

the status is 1.

Passing File Information to Child Processes

The main() Function

If your program uses the exec or spawn functions to create a new process, the new process will normally inherit all of the open file handles created by the original process. Some information, however, about these handles will be lost, including the access mode used to open the file. For example, if your program opens a file for read-only access in binary mode, and then spawns a child process, the child process might corrupt the file by writing to it, or by reading from it in text mode.

To allow child processes to inherit such information about open files, you must link your program with the object file FILEINFO.OBJ.

For example:

```
BCC32 TEST.C \BCB\LIB\FILEINFO.OBJ
```

The file information is passed in the environment variable _C_FILE_INFO. This variable contains encoded binary information. Your program should not attempt to read or modify its value. The child program must have been built with the C++ run-time library to inherit this information correctly.

Other programs can ignore _C_FILE_INFO, and will not inherit file information.

Multithread Programs

See also

32-bit programs can create more than one thread of execution. If your program creates multiple threads, and these threads also use the C++ run-time library, you must use the CW32MT.LIB or CW32MTI library instead.

The multithread libraries provide the following functions which you use to create threads:

_beginthread

_beginthreadNT

The multithread libraries also provide

_endthread   a function that terminates threads

_threadid   a global variable that contains the current identification number of the thread also known as the *thread ID*).

The header file stddef.h contains the declaration of _*threadid*.

When you compile or link a program that uses multiple threads, you must use the **-tWM** compiler switch. For example:

```
BCC32 -tWM THREAD.C
```

Note: Take special care when using the signal function in a multithread program. The SIGINT, SIGTERM, and SIGBREAK signals can be used only by the main thread (thread one) in a non-Win32 application. When one of these signals occurs, the currently executing thread is suspended, and control transfers to the signal handler (if any) set up by thread one. Other signals can be handled by any thread.

A signal handler should not use C++ run-time library functions, because a semaphore deadlock might occur. Instead, the handler should simply set a flag or post a semaphore, and return immediately.

WinMain

See also

Syntax

```
int PASCAL WinMain(HINSTANCE hCurInstance, HINSTANCE hPrevInstance, LPSTR
  lpCmdLine, int nCmdShow)
```

Description

This function is the main entry point for a Windows application. It must be supplied by the user.

Type	Parameter	Description
HINSTANCE	*hCurInstance*	The instance handle of the application. Each instance of an application has a unique instance handle. It is used as an argument to several Windows functions and can be used to distinguish between multiple instances of a given application.
HINSTANCE	*hPrevInstance*	The handle of the previous instance of this application. This value is NULL if this is the first instance.
LPSTR	*lpCmdLine*	A far pointer to a null-terminated command-line. Specify this value when invoking the application from the program manager or from a call to **WinExec**.
int	*nCmdShow*	An integer that specifies the application's window display. Pass this value to **ShowWindow**.

Under Win32, there are two differences in the values passed through these parameters:

- *hPrevInstance* always returns NULL.
- *lpCmdLine* points to a string containing the entire command line, not just the parameters.

Return Value

The return value from *WinMain* is not currently used by Windows. It is useful during debugging because you can display this value upon termination of your program.

Library Routines, by Category

See also Overview

If you know the name of the function you want Help on, see:

Library Routines, by Name

Otherwise, if you do not know the name of a particular function, but you know what type of action it performs, choose one of the following categories:

Classification Routines
Console I/O Routines
Conversion Routines
Diagnostic Routines
Directory Control Routines
Inline Routines
Input/output Routines
Manipulation Routines
Math Routines
Memory Routines
Miscellaneous Routines
Obsolete Functions
Process Control Routines
Time and Date Routines
Variable Argument List Routines

C++Builder has several hundred classes, functions, and macros that you call from within your C and C++ programs to perform a wide variety of tasks, including low- and high-level I/O, string and file manipulation, memory allocation, process control, data conversion, mathematical calculations, and much more. These classes, functions, and macros are collectively referred to as library routines.

Run-time Libraries Overview

C++Builder has several hundred classes, functions, and macros that you call from within your C and C++ programs to perform a wide variety of tasks, including low- and high-level I/O, string and file manipulation, memory allocation, process control, data conversion, mathematical calculations, and much more. These classes, functions, and macros are collectively referred to as library routines.

The C++Builder run-time libraries are divided into static (.OBJ and .LIB) and dynamic-link (.DLL) versions.

- Static libraries are located in the LIB subdirectory of your installation.
- Dynamic-link libraries are located in the BIN subdirectory of your installation.

Several versions of the run-time libraries are available. For example, there are specific versions for, debugging and versions that either include Delph and VCL support, or contain only C and C++ routines

Static Run-time Libraries

See also Overview

Listed below are each of the C++Builder static library names, the operating environment in which it is available, and its use.

File name	Use
Directory of BCB\LIB	
BWCC32.LIB	32-bit import library for BWCC32.DLL
C0D32.OBJ	32-bit DLL startup module
C0W32.OBJ	32-bit GUI EXE startup module
C0X32.OBJ	32-bit console-mode EXE startup module
CP32MT.LIB	32-bit GUI multithread library with VCL and support for Delphi exception handling
CW32MT.LIB	32-bit GUI multithread library
CW32MTI.LIB	32-bit multithread, GUI, dynamic RTL import library for W3230MT.DLL
IMPORT32.LIB	32-bit import library; includes Winsock 1.x
INET.LIB	Import library for the Internet API (URLMON, WININET, HLINK, MSCONF, WEBPOST).
MSEXTRA.LIB	Import library for some APIs who's module names differ between NT and Win95.
MSWSOCK.LIB	Import library for MSWSOCK.DLL.
OBSOLETE.LIB	Provides obsolete global variables.
OLE2W32.LIB	Import library for the 32-bit OLE 2.0 API.
RPCEXTRA.LIB	Import library for some RPC APIs who's module names differ between NT and Win95.
TH32.LIB	Import library for the 32-bit ToolHelp API under Win95.
VCL.LIB	Visual Component Library
VCLD.LIB	Visual Component Library , debug version
WS2_32.LIB	Import library for the 32-bit WinSock 2.0 API.
W32SUT32.LIB	32-bit universal thunking library
Directory of BCB\LIB\RTL	
FILES.C	Increases the number of file handles
FILES2.C	Increases the number of file handles
FILEINFO.OBJ	Passes open file-handle information to child processes
GP.OBJ	Prints register-dump information when an exception occurs
MATHERR.C	Sample of a user-defined floating-point math exception handler for **float** and **double** types
MATHERRL.C	Sample of a user-defined floating-point math exception handler for **long double** type
WILDARGS.OBJ	Transforms wild-card arguments into an array of arguments to *main()* in console-mode applications

Dynamic-link Libraries

See also Overview

The dynamic-link library (DLL) version of the run-time library is contained in the BIN subdirectory of your installation. The dynamic-link versions of the static libraries either contain VCL support and Delphi style exception handling.

Listed below are each of the C++Builder DLL names, the operating environment in which it is available, and its use.

Directory: BCB\BIN

File Name	
CP3230MT.DLL	32-bit, multithread, GUI mode, Delphi exception handling and VCL support
CW3230MT.DLL	32-bit, multithread, GUI mode; no Delphi or VCL support

Default Run-Time Libraries

See also Overview

The following table identifies the default run-time libraries used with each compiler.

Compiler	Environment	Default Libraries
BCC32.EXE	Win32	C0W32.OBJ, CW32MT.LIB, IMPORTW32.LIB

C++ Prototyped Routines

See also

Certain routines described in this book have multiple declarations. You must choose the prototype appropriate for your program. In general, the multiple prototypes are required to support the original C implementation and the stricter and sometimes different C++ function declaration syntax. For example, some string-handling routines have multiple prototypes because in addition to the ANSI-C specified prototype, C++Builder provides prototypes consistent with the ANSI C++ draft.

Function	Header
max	stdlib.h
memchr	string.h
min	stdlib.h
strchr	string.h
strpbrk	string.h
strrchr	string.h
strstr	string.h

Classification Routines

See also

The following routines classify ASCII characters as letters, control characters, punctuation, uppercase, and so.

These routines are all declared in ctype.h.

isalnum	islower
isalpha	isprint
isascii	ispunct
iscntrl	isspace
isdigit	isupper
isgraph	isxdigit

Console I/O Routines

See also

The following routines output text to the screen or read from the keyboard. They cannot be used in a GUI application.

Function	Header	Function	Header
cgets	conio.h	movetext	conio.h
clreol	conio.h	normvideo	conio.h
clrscr	conio.h	putch	conio.h
cprintf	conio.h	puttext	conio.h
cputs	conio.h	_setcursortype	conio.h
delline	conio.h	textattr	conio.h
getpass	conio.h	textbackground	conio.h
gettext	conio.h	textcolor	conio.h
gettextinfo	conio.h	textmode	conio.h
gotoxy	conio.h	ungetc	stdio.h
highvideo	conio.h	wherex	conio.h
insline	conio.h	wherey	conio.h
lowvideo	conio.h	window	conio.h

Conversion Routines

See also

The following routines convert characters and strings from

- alpha to different numeric representations (floating-point, integers, longs)
- numeric to alpha representations
- uppercase to lowercase (and vice versa).

Function	Header	Function	Header
atof	stdlib.h	strtol	stdlib.h
atoi	stdlib.h	_strtold	stdlib.h
atol	stdlib.h	strtoul	stdlib.h
ecvt	stdlib.h	toascii	ctype.h
fcvt	stdlib.h	_tolower	ctype.h
gcvt	stdlib.h	tolower	ctype.h
itoa	stdlib.h	_toupper	ctype.h
_ltoa	stdlib.h	toupper	ctype.h
strtod	stdlib.h	ultoa	stdlib.h

Diagnostic Routines

See also

The following routines provide built-in troubleshooting capability.

Function	Header
assert	assert.h
_matherr	math.h
_matherrl	math.h
perror	errno.h

Directory Control Routines

See also

The following routines manipulate directories and path names.

Function	Header	Function	Header
chdir	dir.h	_getdcwd	direct.h
_chdrive	direct.h	getdisk	dir.h
closedir	dirent.h	_makepath	stdlib.h
		mkdir	dir.h
		_mktemp	dir.h
		opendir	direct.h
		readdir	dirent.h
		rewinddir	dirent.h
findfirst	dir.h	_rmdir	dir.h
findnext	dir.h	_searchenv	stdlib.h
fnmerge	dir.h	searchpath	dir.h
fnsplit	dir.h	_searchstr	stdlib.h
_fullpath	stdlib.h	setdisk	dir.h
getcurdir	dir.h	_splitpath	stdlib.h
getcwd	dir.h		

Inline Routines

See also

The following routines have inline versions. The compiler will generate code for the inline versions when you use `#pragma intrinsic` or if you specify program optimization.

Function	Header	Function	Header
abs	math.h	stpcpy	string.h
alloca	malloc.h	strcat	string.h
_crotl	stdlib.h	strchr	string.h
_crotr	stdlib.h	strcmp	string.h
_lrotl	stdlib.h	strcpy	string.h
_lrotr	stdlib.h	strlen	string.h
memchr	mem.h	strncat	string.h
memcmp	mem.h	strncmp	string.h
memcpy	mem.h	strncpy	string.h
memset	mem.h	strnset	string.h
_rotl	stdlib.h	strrchr	string.h
_rotr	stdlib.h	strset	string.h

Input/output Routines

See also

The following routines provide stream- and operating-system level I/O capability.

Function	Header	Function	Header
access	io.h	getftime	io.h
chmod	io.h	gets	stdio.h
chsize	io.h	_getw	stdio.h
clearerr	stdio.h		
close	io.h	isatty	io.h
creat	io.h	kbhit	conio.h
creatnew	io.h	lock	io.h
creattemp	io.h	locking	io.h
cscanf	conio.h	lseek	io.h
		open	io.h
_pclose	stdio.h	_open_osfhandle	io.h
perror	stdio.h		
	_pipe	io.h	
	_popen	stdio.h	
	printf	stdio.h	
putc	stdio.h		
	putchar	stdio.h	
	puts	stdio.h	
	_putw	stdio.h	
	read	io.h	
dup	io.h	remove	stdio.h
dup2	io.h	rename	stdio.h
eof	io.h	rewind	stdio.h
fclose	stdio.h	_rmtmp	stdio.h
_fcloseall	stdio.h	_rtl_chmod	io.h
_fdopen	stdio.h	_rtl_close	io.h
feof	stdio.h	_rtl_creat	io.h
ferror	stdio.h	_rtl_open	io.h
fflush	stdio.h	_rtl_read	io.h
fgetc	stdio.h	_rtl_write	io.h
_fgetchar	stdio.h	scanf	stdio.h
fgetpos	stdio.h	setbuf	stdio.h
fgets	stdio.h		
filelength	io.h	setmode	io.h
_fileno	stdio.h	setvbuf	stdio.h
_flushall	stdio.h	_sopen	io.h

fopen	stdio.h	sprintf	stdio.h
fprintf	stdio.h	sscanf	stdio.h
fputc	stdio.h	strerror	stdio.h
_fputchar	stdio.h	_strerror	string.h, stdio.h
fputs	stdio.h	tell	io.h
fread	stdio.h	_tempnam	stdio.h
freopen	stdio.h		
fscanf	stdio.h	tmpfile	stdio.h
fseek	stdio.h	tmpnam	stdio.h
fsetpos	stdio.h	umask	io.h
_fsopen	stdio.h	_unlink	dos.h
fstat	sys\stat.h	unlock	io.h
ftell	stdio.h	_utime	utime.h
fwrite	stdio.h	vfprintf	stdio.h
get_osfhandle	io.h	vfscanf	stdio.h
getc	stdio.h	vprintf	stdio.h
getch	conio.h	vscanf	stdio.h
getchar	stdio.h	vsprintf	stdio.h
getche	conio.h	vsscanf	io.h

Manipulation Routines

See also

The following routines handle strings and blocks of memory: copying, comparing, converting, and searching.

Function	Header	Function	Header
mblen	stdlib.h	strerror	string.h
mbstowcs	stdlib.h	stricmp	string.h
mbtowc	stdlib.h	strlen	string.h
memccpy	mem.h, string.h	strlwr	string.h
memchr	mem.h, string.h	strncat	string.h
memcmp	mem.h, string.h	strncmpi	string.h
memcpy	mem.h, string.h	strncmp	string.h
memicmp	mem.h, string.h	strncpy	string.h
memmove	mem.h, string.h	strnicmp	string.h
memset	mem.h, string.h	strnset	string.h
		strpbrk	string.h
		strrchr	string.h
setmem	mem.h	strrev	string.h
stpcpy	string.h	strset	string.h
strcat	string.h	strspn	string.h
strchr	string.h	strstr	string.h
strcmpi	string.h	strtok	string.h
strcmp	string.h	strupr	string.h
strcoll	string.h	strxfrm	string.h
strcpy	string.h	wcstombs	stdlib.h
strcspn	string.h	wctomb	stdlib.h
strdup	string.h		

Math Routines

See also

The folowing routines perform mathematical calculations and conversions.

Function	Header	Function	Header
abs	stdlib.h	labs	stdlib.h
acos	math.h	ldexp	math.h
acosl	math.h	ldexpl	math.h
		ldiv	math.h
asin	math.h	log	math.h
asinl	math.h	logl	math.h
atan	math.h	log10	math.h
atan2	math.h	log10l	math.h
atan2l	math.h	_lrotl	stdlib.h
atanl	math.h	_lrotr	stdlib.h
atof	stdlib.h, math.h	_ltoa	stdlib.h
atoi	stdlib.h	_matherr	math.h
atol	stdlib.h	_matherrl	math.h
_atold	math.h	modf	math.h
	modfl	math.h	
cabs	math.h		
cabsl	math.h		
ceil	math.h	poly	math.h
ceill	math.h	polyl	math.h
_clear87	float.h	pow	math.h
_control87	float.h	powl	math.h
cos	math.h	rand	stdlib.h
cosh	math.h	random	stdlib.h
coshl	math.h	randomize	stdlib.h
cosl	math.h		
div	math.h	_rotl	stdlib.h
ecvt	stdlib.h	_rotr	stdlib.h
exp	math.h	sin	math.h
expl	math.h	sinh	math.h
fabs	math.h	sinhl	math.h
fabsl	math.h	sinl	math.h
fcvt	stdlib.h	sqrt	math.h
floor	math.h	sqrtl	math.h
floorl	math.h	srand	stdlib.h
fmod	math.h	_status87	float.h
fmodl	math.h	strtod	stdlib.h

_fpreset	float.h	strtol	stdlib.h
frexp	math.h	_strtold	stdlib.h
frexpl	math.h	strtoul	stdlib.h
gcvt	stdlib.h	tan	math.h
hypot	math.h	tanh	math.h
hypotl	math.h	tanhl	math.h
		tanl	math.h
itoa	stdlib.h	ultoa	stdlib.h

Memory Routines

See also

The following routines provide dynamic memory allocation in the small-data and large-data models.

Function	Header	Function	Header
alloca	malloc.h	heapcheckfree	alloc.h
	heapchecknode	alloc.h	
calloc	alloc.h, stdlib.h	heapwalk	alloc.h
	malloc	alloc.h, stdlib.h	
	realloc	alloc.h, stdlib.h	
	set_new_handler	new.h	
free	alloc.h, stdlib.h	stackavail	malloc.h
heapcheck	alloc.h		

Miscellaneous Routines

See also

The following routines provide non-local goto capabilities and locale.

Function	Header
localeconv	locale.h
longjmp	setjmp.h
setjmp	setjmp.h
setlocale	locale.h

Obsolete Functions

See also

The old names of the following functions are available, but the compiler will generate a warning that you are using an obsolete name. Future versions of C++Builder might not provide support for the old function names.

The following function names have been changed:

Old name	New name	Header file
_chmod	_rtl_chmod	io.h
_close	_rtl_close	io.h
_creat	_rtl_creat	io.h
_heapwalk	_rtl_heapwalk	malloc.h
_open	_rtl_open	io.h
_read	_rtl_read	io.h
_write	_rtl_write	io.h

Process Control Routines

See also

The following routines invoke and terminate new processes from within another routine.

Function	Header	Function	Header
abort	(process.h)	exit	(process.h)
_beginthread		_expand	(process.h)
_beginthreadNT	(process.h)	getpid	(process.h)
_c_exit	(process.h)	_pclose	(stdio.h)
_cexit	(process.h)	_popen	(stdio.h)
cwait	(process.h)	raise	(signal.h)
_endthread	(process.h)	signal	(signal.h)
execle	(process.h)	spawnle	(process.h)
execl	(process.h)	spawnlpe	(process.h)
execlpe	(process.h)	spawnlp	(process.h)
execlp	(process.h)	spawnl	(process.h)
execve	(process.h)	spawnve	(process.h)
execv	(process.h)	spawnvpe	(process.h)
execvpe	(process.h)	spawnvp	(process.h)
execvp	(process.h)	spawnv	(process.h)
_exit	(process.h)	wait	(process.h)

Time and Date Routines

See also

The following following functions are time conversion and time manipulation routines.

Function	Header	Function	Header
asctime	time.h	gmtime	time.h
	bios.h	localtime	time.h
ctime	time.h	mktime	time.h
difftime	time.h	stime	time.h
		_strdate	time.h
		strftime	time.h
		_strtime	time.h
		time	time.h
ftime	sys\timeb.h		
		_tzset	time.h
gettime	dos.h	unixtodos	dos.h

Variable Argument List Routines

See also

The following routines are for use when accessing variable argument lists (such as with printf, vscanf, and so on).

Function	Header
va_start	stdarg.h
va_arg	stdarg.h
va_end	stdarg.h

Library Routines, by Name

See also Overview
{button A,JI(`',`libxref_a')} {button B,JI(`',`libxref_b')} {button C,JI(`',`libxref_c')} {button D,JI(`',`libxref_d')} {button E,JI(`',`libxref_e')} {button F,JI(`',`libxref_f')} {button G,JI(`',`libxref_g')} {button H,JI(`',`libxref_h')} {button I,JI(`',`libxref_i')} {button K,JI(`',`libxref_k')} {button L,JI(`',`libxref_l')} {button M,JI(`',`libxref_m')} {button N,JI(`',`libxref_n')} {button O,JI(`',`libxref_o')} {button P,JI(`',`libxref_p')} {button Q,JI(`',`libxref_q')} {button R,JI(`',`libxref_r')} {button S,JI(`',`libxref_s')} {button T,JI(`',`libxref_t')} {button U,JI(`',`libxref_u')} {button V,JI(`',`libxref_v')} {button W,JI(`',`libxref_w')}

If you do not know the name of a particular function, but you know what type of action it performs, see:

Library Routines, by Category

Otherwise, if you know which function you want Help on, choose one of the following topics:

A

abort
abs
access
acos
acosl
alloca
asctime
asin
asinl
assert
atan
atan2
atan2l
atanl
atexit
atof
atoi
atol
_atold

B

_beginthread
_beginthreadNT
bsearch

C

_c_exit
cabs
cabsl
calloc
ceil
ceill
_cexit
cgets
chdir
_chdrive
chmod
chsize
_clear87
clearerr

clock
close
closedir
clreol
clrscr
_control87
cos
cosh
coshl
cosl
creat
creatnew
creattemp
_crotl
_crotr
cscanf
ctime
cwait

D
delline
difftime
disable
_disable
div
dup

E
ecvt
__emit__
enable
_enable
_endthread
eof
execl
execle
execlp
execlpe
execv
execve
execvp
execvpe
exit
_exit
exp
_expand
expl

F
fabs
fabsl

fclose
_fcloseall
fcvt
_fdopen
feof
ferror
fflush
fgetc
_fgetchar
fgetpos
fgets
filelength
_fileno
findfirst
findnext
floor
floorl
_flushall
fmod
fmodl
fnmerge
fnsplit
fopen
_fpreset
fprintf
fputc
_fputchar
fputs
fread
free
freopen
frexp
frexpl
fscanf
fseek
fsetpos
_fsopen
fstat
ftell
ftime
_fullpath
fwrite

G

gcvt
geninterrupt
_get_osfhandle
getc
getch
getchar

getche
getcurdir
getcwd
_getdcwd
getdfree
getdisk
getenv
getftime
getpass
getpid
gets
gettext
gettextinfo
gettime
_getw
gmtime
gotoxy

H
heapcheck
heapcheckfree
heapchecknode
_heapchk
heapfillfree
_heapmin
_heapset
heapwalk
highvideo
hypot
hypotl

I
insline
isalnum
isalpha
isascii
isatty
iscntrl
isdigit
isgraph
islower
isprint
ispunct
isspace
isupper
isxdigit
itoa

K
kbhit

L

labs
ldexp
ldexpl
ldiv
lfind
localeconv
localtime
lock
locking
log
log10
log10l
logl
longjmp
lowvideo
_lrotl
_lrotr
lsearch
lseek
_ltoa

M

_makepath
malloc
_matherr
_matherrl
max
mblen
mbstowcs
mbtowc
memccpy
memchr
memcmp
memcpy
memicmp
memmove
memset
min
mkdir
_mktemp
mktime
modf
modfl
movetext
_msize

N

normvideo

O

offsetof
open
_open_osfhandle
opendir

P
_pclose
perror
_pipe
poly
polyl
_popen
pow
pow10
pow10l
powl
printf
putc
putch
putchar
putenv
puts
puttext
_putw

Q
qsort

R
raise
rand
random
randomize
read
readdir
realloc
remove
rename
rewind
rewinddir
_rmdir
_rmtmp
_rotl
_rotr
_rtl_chmod
_rtl_close
_rtl_creat
_rtl_heapwalk
_rtl_open
_rtl_read
_rtl_write

S

scanf
_searchenv
searchpath
_searchstr
set_new_handler
setbuf
_setcursortype
setdate
setdisk
setjmp
setlocale
setmem
setmode
settime
setvbuf
signal
sin
sinh
sinhl
sinl
sleep
_sopen
spawnl
spawnle
spawnlp
spawnlpe
spawnv
spawnve
spawnvp
spawnvpe
_splitpath
sprintf
sqrt
sqrtl
srand
sscanf
stackavail
stat
_status87
stime
stpcpy
strcat
strchr
strcmp
strcmpi
strcoll
strcpy
strcspn

_strdate
strdup
strerror
_strerror
strftime
stricmp
strlen
strlwr
strncat
strncmp
strncmpi
strncpy
strnicmp
strnset
strpbrk
strrchr
strrev
strset
strspn
strstr
_strtime
strtod
strtok
strtol
_strtold
strtoul
strupr
strxfrm
swab
system

T
tan
tanh
tanhl
tanl
tell
_tempnam
textattr
textbackground
textcolor
textmode
time
tmpfile
tmpnam
toascii
tolower
_tolower
toupper
_toupper

_tzset

U
ultoa
umask
ungetc
ungetch
unixtodos
_unlink
unlock
_utime

V
va_arg
va_end
va_start
vfprintf
vfscanf
vprintf
vscanf
vsprintf
vsscanf

W
wait
wcstombs
wctomb
wherex
wherey
window
write

abort

See also Example Portability

Syntax

```
#include <stdlib.h>
void abort(void);
```

Description

Abnormally terminates a program.

abort causes an abnormal program termination by calling *raise*(SIGABRT). If there is no signal handler for SIGABRT, then *abort* writes a termination message (Abnormal program termination) on stderr, then aborts the program by a call to _*exit* with exit code 3.

Return Value

abort returns the exit code 3 to the parent process or to the operating system command processor.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

abs

See also Example Portability

Syntax

```
#include <stdlib.h>
int abs(int x);
```

Description

Returns the absolute value of an integer.

abs returns the absolute value of the integer argument *x*. If *abs* is called when stdlib.h has been included, it's treated as a macro that expands to inline code.

If you want to use the *abs* function instead of the macro, include

```
#undef abs
```

in your program, after the #include <stdlib.h>.

Return Value

The *abs* function returns an integer in the range of 0 to INT_MAX, with the exception that an argument with the value INT_MIN is returned as INT_MIN. The values for INT_MAX and INT_MIN are defined in header file limit.h.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

access, _waccess

See also   Example   Portability

Syntax

```
#include <io.h>
int access(const char *filename, int amode);
int _waccess(const wchar_t *filename, int amode);
```

Description

Determines accessibility of a file.

access checks the file named by *filename* to determine if it exists, and whether it can be read, written to, or executed.

The list of *amode* values is as follows:

06	Check for read and write permission
04	Check for read permission
02	Check for write permission
01	Execute (ignored)
00	Check for existence of file

Under DOS, OS/2, and Windows (16- and 32-bit) all existing files have read access (*amode* equals 04), so 00 and 04 give the same result. Similarly, *amode* values of 06 and 02 are equivalent because under DOS write access implies read access.

If *filename* refers to a directory, *access* simply determines whether the directory exists.

Return Value

If the requested access is allowed, *access* returns 0; otherwise, it returns a value of -1, and the global variable errno is set to one of the following values:

ENOENT	Path or file name not found
EACCES	Permission denied

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

acos, acosl

See also Example Portability

Syntax

```
#include <math.h>
double acos(double x);
long double acosl(long double x);
```

Description

Calculates the arc cosine.

acos returns the arc cosine of the input value.

acosl is the **long double** version; it takes a **long double** argument and returns a **long double** result.

Arguments to *acos* and *acosl* must be in the range -1 to 1, or else *acos* and *acosl* return NAN and set the global variable errno to

EDOM Domain error

Return Value

acos and *acosl* of an argument between -1 and +1 return a value in the range 0 to *pi*. Error handling for these routines can be modified through the functions _matherr_matherr and _matherrl.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
acos	+	+	+	+	+	+	+
acosl	+		+	+			+

alloca

See also Example Portability

Syntax

```
#include <malloc.h>
void *alloca(size_t size);
```

Description

Allocates temporary stack space.

alloca allocates *size* bytes on the stack; the allocated space is automatically freed up when the calling function exits.

Because *alloca* modifies the stack pointer, do not place calls to *alloca* in an expression that is an argument to a function.

The *alloca* function should not be used in the **try**-block of a C++ program. If an exception is thrown, any values placed on the stack by *alloca* will be corrupted.

If the calling function does not contain any references to local variables in the stack, the stack will not be restored correctly when the function exits, resulting in a program crash. To ensure that the stack is restored correctly, use the following code in the calling function:

```
char *p;
char dummy[5];

dummy[0] = 0;

   .
   .
   .
p = alloca(nbytes);
```

Return Value

If enough stack space is available, *alloca* returns a pointer to the allocated stack area. Otherwise, it returns NULL.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

asctime, _wasctime

See also Example Portability

Syntax

```
#include <time.h>
char *asctime(const struct tm *tblock);
wchar_t *_wasctime(const struct tm *tblock);
```

Description

asctime converts date and time to ASCII.

_*wasctime* converts date and time to a **wchar_t** string.

asctime converts a time stored as a structure in **tblock* to a 26-character string of the same form as the *ctime* string:

```
Sun Sep 16 01:03:52 1973\n\0
```

Return Value

asctime returns a pointer to the character string containing the date and time. This string is a static variable that is overwritten with each call to *asctime*.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

asin, asinl

See also Example Portability

Syntax

```
#include <math.h>
double asin(double x);
long double asinl(long double x);
```

Description

Calculates the arc sine.

asin of a real argument returns the arc sine of the input value.

asinl is the **long double** version; it takes a **long double** argument and returns a **long double** result.

Real arguments to *asin* and *asinl* must be in the range -1 to 1, or else *asin* and *asinl* return NAN and set the global variable errno to

EDOM Domain error

Return Value

asin and *asinl* of a real argument return a value in the range -*pi*/2 to *pi*/2. Error handling for these functions may be modified through the functions _*matherr* and _*matherrl*.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
asin	+	+	+	+	+	+	+
asinl	+		+	+			+

assert

See also Example Portability

Syntax

```
#include <assert.h>
void assert(int test);
```

Description

Tests a condition and possibly aborts.

assert is a macro that expands to an **if** statement; if *test* evaluates to zero, *assert* aborts the program (by calling abort) and asserts the following a message on stderr:

```
Assertion failed: test, file filename, line linenum
```

The *filename* and *linenum* listed in the message are the source file name and line number where the *assert* macro appears.

If you place the #define NDEBUG directive ("no debugging") in the source code before the #include <assert.h> directive, the effect is to comment out the assert statement.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

atan, atanl

See also Example Portability

Syntax

```
#include <math.h>
double atan(double x);
long double atanl(long double x);
```

Description

Calculates the arc tangent.

atan calculates the arc tangent of the input value.

atanl is the **long double** version; it takes a **long double** argument and returns a **long double** result.

Return Value

atan and *atanl* of a real argument return a value in the range -*pi*/2 to *pi*/2. Error handling for these functions can be modified through the functions _*matherr* and _*matherrl.*

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
atan	+	+	+	+	+	+	+
atanl	+		+	+			+

atan2, atan2l

See also Example Portability

Syntax

```
#include <math.h>
double atan2(double y, double x);
long double atan2l(long double y, long double x);
```

Description

Calculates the arc tangent of *y/x*.

atan2 returns the arc tangent of *y/x*; it produces correct results even when the resulting angle is near *pi*/2 or -*pi*/2 *(x* near 0). If both *x* and *y* are set to 0, the function sets the global variable errno to EDOM, indicating a domain error.

atan2l is the **long double** version; it takes **long double** arguments and returns a **long double** result.

Return Value

atan2 and *atan2l* return a value in the range -*pi* to *pi*. Error handling for these functions can be modified through the functions _*matherr* and _*matherrl*.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
atan2	+	+	+	+	+	+	+
atan2l	+		+	+			+

atexit

See also Example Portability

Syntax

```
#include <stdlib.h>
int atexit(void (_USERENTRY * func)(void));
```

Description

Registers termination function.

atexit registers the function pointed to by *func* as an exit function. Upon normal termination of the program, *exit* calls *func* just before returning to the operating system. fcmp must be used with the _USERENTRY calling convention.

Each call to *atexit* registers another exit function. Up to 32 functions can be registered. They are executed on a last-in, first-out basis (that is, the last function registered is the first to be executed).

Return Value

atexit returns 0 on success and nonzero on failure (no space left to register the function).

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+	+	+	+

atof, _atold, _wtof, _wtold

See also Example Portability

Syntax

```
#include <math.h>
double atof(const char *s);
double _wtof(const wchar_t *s);
long double _atold(const char *s);
long double _wtold(const wchar_t *s);
```

Description

Converts a string to a floating-point number.

atof converts a string pointed to by *s* to **double**; this function recognizes the character representation of a floating-point number, made up of the following:

- An optional string of tabs and spaces
- An optional sign
- A string of digits and an optional decimal point (the digits can be on both sides of the decimal point)
- An optional *e* or *E* followed by an optional signed integer

The characters must match this generic format:

[whitespace] [sign] [ddd] [.] [ddd] [e|E[sign]ddd]

atof also recognizes +INF and -INF for plus and minus infinity, and +NAN and -NAN for Not-a-Number.

In this function, the first unrecognized character ends the conversion.

_atold is the **long double** version; it converts the string pointed to by *s* to a **long double.**

The functions strtod and _strtold are similar to *atof* and *_atold*; they provide better error detection, and hence are preferred in some applications.

Return Value

atof and *_atold* return the converted value of the input string.

If there is an overflow, *atof* (or *_atold*) returns plus or minus HUGE_VAL (or _LHUGE_VAL), errno is set to ERANGE (Result out of range), and _matherr (or _matherrl) is not called.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
atof	+	+	+	+	+	+	+
_atold	+		+	+			+

atoi, _atoi64, _wtoi, _wtoi64

See also Example Portability

Syntax

```
#include <stdlib.h>
int atoi(const char *s);
__int64 _atoi64(const char *s);
int _wtoi(const wchar_t *s);
__int64 _wtoi64(const wchar_t *s);
```

Description

Converts a string to an integer.

atoi converts a string pointed to by *s* to **int**; *atoi* recognizes (in the following order)

- An optional string of tabs and spaces
- An optional sign
- A string of digits

The characters must match this generic format:

[ws] [sn] [ddd]

In this function, the first unrecognized character ends the conversion. There are no provisions for overflow in *atoi* (results are undefined).

Return Value

atoi returns the converted value of the input string. If the string cannot be converted to a number of the corresponding type (**int**), atoi returns 0.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

aatol, _wtol

See also Example Portability

Syntax

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

Description

Converts a string to a long.

atol converts the string pointed to by *s* to **long**. *atol* recognizes (in the following order)

- An optional string of tabs and spaces
- An optional sign
- A string of digits

The characters must match this generic format:

```
[ws] [sn] [ddd]
```

In this function, the first unrecognized character ends the conversion. There are no provisions for overflow in *atol* (results are undefined).

Return Value

atol returns the converted value of the input string. If the string cannot be converted to a number of the corresponding type (b), *atol* returns 0.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

_beginthread

See also Example Portability

Syntax

```
#include <process.h>
unsigned long _beginthread(_USERENTRY (*start_address)(void *), unsigned
  stack_size, void *arglist)
```

Description

Starts execution of a new thread.

Note: The *start_address* must be declared to be *_USERENTRY*.

The *_beginthread* function creates and starts a new thread. The thread starts execution at *start_address*.

The size of its stack in bytes is *stack_size*; the stack is allocated by the operating system after the stack size is rounded up to the next multiple of 4096. The thread is passed *arglist* as its only parameter; it can be NULL, but must be present. The thread terminates by simply returning, or by calling _endthread.

Either this function or _beginthreadNT must be used instead of the operating system thread-creation API function because *_beginthread* and *_beginthreadNT* perform initialization required for correct operation of the run-time library functions.

This function is available only in the multithread libraries.

Return Value

_beginthread returns the handle of the new thread.

On error, the function returns -1, and the global variable errno is set to one of the following values:

EAGAIN	Too many threads
EINVAL	Invalid request

Also see the Win32 description of *GetLastError*.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
			+			+

_beginthreadNT

See also Example Portability

Syntax

```
#include <process.h>
unsigned long _beginthreadNT(void (_USERENTRY *start_address)(void *),
  unsigned stack_size, void *arglist, void *security_attrib, unsigned long
  create_flags, unsigned long *thread_id);
```

Description

Starts execution of a new thread under Windows NT.

Note: The start_address must be declared to be *_USERENTRY*.

All multithread Windows NT programs must use *_beginthreadNT* or the _beginthreadfunction instead of the operating system thread-creation API function because these functions perform initialization required for correct operation of the run-time library functions. The *_beginthreadNT* function provides support for the operating system security. These functions are available only in the multithread libraries.

The *_beginthreadNT* function creates and starts a new thread. The thread starts execution at *start_address*.

The size of its stack in bytes is *stack_size*; the stack is allocated by the operating system after the stack size is rounded up to the next multiple of 4096. The thread *arglist* can be NULL, but must be present. The thread terminates by simply returning, or by calling *_endthread*.

The *_beginthreadNT* function uses the *security_attr* pointer to access the SECURITY_ATTRIBUTES structure. The structure contains the security attributes for the thread. If *security_attr* is NULL, the thread is created with default security attributes. The thread handle is not inherited if *security_attr* is NULL.

_beginthreadNT reads the *create_flags* variable for flags that provide additional information about the thread creation. This variable can be zero, specifying that the thread will run immediately upon creation. The variable can also be CREATE_SUSPENDED; in which case, the thread will not run until the *ResumeThread* function is called. *ResumeThread* is provided by the Win32 API.

_beginthreadNT initializes the *thread_id* variable with the thread identifier.

Return Value

On success, *_beginthreadNT* returns the handle of the new thread.

On error, it returns -1, and the global variable errno is set to one of the following values:

EAGAIN	Too many threads
EINVAL	Invalid request

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
			+			

biosmemory

Example Portability

Syntax

```
#include <bios.h>
int biosmemory(void);
```

Description

Returns memory size.

biosmemory returns the size of RAM memory using BIOS interrupt 0x12. This does not include display adapter memory, extended memory, or expanded memory.

Return Value

biosmemory returns the size of RAM memory in 1K blocks.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+				

bsearch

See also Example Portability

Syntax

```
#include <stdlib.h>
void *bsearch(const void *key, const void *base, size_t nelem, size_t width,
  int (_USERENTRY *fcmp)(const void *, const void *));
```

Description

Binary search of an array.

bsearch searches a table (array) of *nelem* elements in memory, and returns the address of the first entry in the table that matches the search key. The array must be in order. If no match is found, *bsearch* returns 0.

Note: Because this is a binary search, the first matching entry is not necessarily the first entry in the table.

The type *size_t* is defined in stddef.h header file.

- *nelem* gives the number of elements in the table.
- *width* specifies the number of bytes in each table entry.

The comparison routine *fcmp* must be used with the _USERENTRY calling convention.

fcmp is called with two arguments: *elem1* and *elem2*. Each argument points to an item to be compared. The comparison function compares each of the pointed-to items (**elem1* and **elem2*), and returns an integer based on the results of the comparison.

For *bsearch*, the *fcmp* return value is

- < 0 if **elem1 < *elem2*
- == 0 if **elem1 == *elem2*
- > 0 if **elem1 > *elem2*

Return Value

bsearch returns the address of the first entry in the table that matches the search key. If no match is found, *bsearch* returns 0.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

cabs, cabsl

See also   Example   Portability

Syntax

```
#include <math.h>
double cabs(struct complex z);
long double cabsl(struct _complexl z);
```

Description

cabs calculates the absolute value of a complex number. *cabs* is a macro that calculates the absolute value of *z*, a complex number. *z* is a structure with type *complex*; the structure is defined in math.h as

```
struct complex {
   double x, y;
   };
```

where *x* is the real part, and *y* is the imaginary part.

Calling *cabs* is equivalent to calling *sqrt* with the real and imaginary components of *z*, as shown here:

```
sqrt(z.x * z.x + z.y * z.y)
```

cabsl is the **long double** version; it takes a structure with *type _complexl* as an argument, and returns a **long double** result. The structure is defined in math.h as

```
struct _complexl {
   long double x, y;
};
```

Return Value

cabs (or *cabsl*) returns the absolute value of *z*, a double. On overflow, *cabs* (or *cabsl*) returns HUGE_VAL (or _LHUGE_VAL) and sets the global variable errno to

ERANGE	Result out of range

Error handling for these functions can be modified through the functions *_matherr* and *_matherrl*.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
cabs	+	+	+	+			+
cabsl	+		+	+			+

calloc

See also Example Portability

Syntax

```
#include <stdlib.h>
void *calloc(size_t nitems, size_t size);
```

Description

Allocates main memory.

calloc provides access to the C memory heap. The heap is available for dynamic allocation of variable-sized blocks of memory. Many data structures, such as trees and lists, naturally employ heap memory allocation.

calloc allocates a block of size *nitems* * *size*. The block is initialized to 0.**Return Value**

calloc returns a pointer to the newly allocated block. If not enough space exists for the new block or if *nitems* or *size* is 0, *calloc* returns NULL.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

ceil, ceill

See also Example Portability

Syntax

```
#include <math.h>
double ceil(double x);
long double ceill(long double x);
```

Description

Rounds up.

ceil finds the smallest integer not less than *x*.

ceill is the **long double** version; it takes a **long double** argument and returns a **long double** result.

Return Value

These functions return the integer found as a **double** *(ceil)* or a **long double** *(ceill)*.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
ceil	+	+	+	+	+	+	+
ceill	+		+	+			+

_c_exit

See also   Example   Portability

Syntax

```
#include <process.h>
void _c_exit(void);
```

Description

Performs _exit cleanup without terminating the program.

_c_exit performs the same cleanup as _exit, except that it does not terminate the calling process.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

_cexit

See also Example Portability

Syntax

```
#include <process.h>
void _cexit(void);
```

Description

Performs exit cleanup without terminating the program.

_cexit performs the same cleanup as exit, closing all files but without terminating the calling process. The *_cexit* function calls any registered "exit functions" (posted with *atexit*). Before *_cexit* returns, it flushes all input/output buffers and closes all streams which were open.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

cgets

See also Example Portability

Syntax

```
#include <conio.h>
char *cgets(char *str);
```

Description

Reads a string from the console.

cgets reads a string of characters from the console, storing the string (and the string length) in the location pointed to by *str*.

cgets reads characters until it encounters a carriage-return/linefeed (CR/LF) combination, or until the maximum allowable number of characters have been read. If *cgets* reads a CR/LF combination, it replaces the combination with a \0 (null terminator) before storing the string.

Before *cgets* is called, set str[0] to the maximum length of the string to be read. On return, *str*[1] is set to the number of characters actually read. The characters read start at *str*[2] and end with a null terminator. Thus, *str* must be at least *str*[0] plus 2 bytes long.

Note: Do not use this function for Win32s or Win32 GUI applications.

Return Value

On success, *cgets* returns a pointer to *str*[2].

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+			+			+

chdir, _wchdir

See also   Example   Portability

Syntax

```
#include <dir.h>
int chdir(const char *path);
int _wchdir(const wchar_t *path);
```

Description

Changes current directory.

chdir causes the directory specified by *path* to become the current working directory; *path* must specify an existing directory.

A drive can also be specified in the path argument, such as

```
chdir("a:\\BC")
```

but this method changes only the current directory on that drive; it does not change the active drive.

Under Windows, only the current process is affected.

Under DOS, the function changes the current directory of the parent process.

Return Value

Upon successful completion, *chdir* returns a value of 0. Otherwise, it returns a value of -1, and the global variable errno is set to

ENOENT	Path or file name not found

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

_chdrive

Example Portability

Syntax

```
#include <direct.h>
int _chdrive(int drive);
```

Description

Sets current disk drive.

_chdrive sets the current drive to the one associated with *drive:* 1 for A, 2 for B, 3 for C, and so on.

This function changes the current drive of the parent process.

Return Value

_chdrive returns 0 if the current drive was changed successfully; otherwise, it returns -1.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

chmod, _wchmod

See also Example Portability

Syntax

```
#include <io.h>
int chmod(const char *path, int amode);
int _wchmod(const wchar_t *path, int amode);
```

Description

Changes file access mode.

chmod sets the file-access permissions of the file given by *path* according to the mask given by *amode. path* points to a string.

amode can contain one or both of the symbolic constants S_IWRITE and S_IREAD (defined in sys\stat.h).

Value of amode	Access permission
S_IWRITE	Permission to write
S_IREAD	Permission to read
S_IREAD | S_IWRITE	Permission to read and write (write permission implies read permission)

Return Value

Upon successfully changing the file access mode, *chmod* returns 0. Otherwise, *chmod* returns a value of -1.

In the event of an error, the global variable errno is set to one of the following values:

EACCES	Permission denied
ENOENT	Path or file name not found

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

chsize

See also Example Portability

Syntax

```
#include <io.h>
int chsize(int handle, long size);
```

Description

Changes the file size.

chsize changes the size of the file associated with *handle*. It can truncate or extend the file, depending on the value of *size* compared to the file's original size.

The mode in which you open the file must allow writing.

If *chsize* extends the file, it will append null characters (\0). If it truncates the file, all data beyond the new end-of-file indicator is lost.

Return Value

On success, *chsize* returns 0. On failure, it returns -1 and the global variable errno is set to one of the following values:

EACCESS	Permission denied
EBADF	Bad file number
ENOSPC	No space left on device

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

_clear87

See also   Example   Portability

Syntax

```
#include <float.h>
unsigned int _clear87 (void);
```

Description

Clears floating-point status word.

_clear87 clears the floating-point status word, which is a combination of the 80x87 status word and other conditions detected by the 80x87 exception handler.

Return Value

The bits in the value returned indicate the floating-point status before it was cleared. For information on the status word, refer to the constants defined in float.h.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

clearerr

See also Example Portability

Syntax

```
#include <stdio.h>
void clearerr(FILE *stream);
```

Description

Resets error indication.

clearerr resets the named stream's error and end-of-file indicators to 0. Once the error indicator is set, stream operations continue to return error status until a call is made to *clearerr* or *rewind*. The end-of-file indicator is reset with each input operation.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

clock

See also   Example   Portability

Syntax

```
#include <time.h>
clock_t clock(void);
```

Description

Determines processor time.

clock can be used to determine the time interval between two events. To determine the time in seconds, the value returned by *clock* should be divided by the value of the macro *CLK_TCK.*

Return Value

On success, *clock* returns the processor time elapsed since the beginning of the program invocation.

On error (if the processor time is not available or its value cannot be represented), *clock* returns -1.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+	+	+	+

close

See also Example Portability

Syntax

```
#include <io.h>
int close(int handle);
```

Description

Closes a file.

The *close* function closes the file associated with *handle*, a file handle obtained from a call to creat, creatnew, creattemp, dup, dup2, open, _rtl_creat, or _rtl_open.

It does not write a *Ctrl-Z* character at the end of the file. If you want to terminate the file with a *Ctrl-Z*, you must explicitly output one.

Return Value

Upon successful completion, *close* returns 0.

On error (if it fails because *handle* is not the handle of a valid, open file), *close* returns a value of -1 and the global variable errno is set to

EBADF	Bad file number

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

closedir, wclosedir

See also   Example   Portability

Syntax

```
#include <dirent.h>
void closedir(DIR *dirp);
void wclosedir(wDIR *dirp);
```

Description

Closes a directory stream.

On UNIX platforms, *closedir* is available on POSIX-compliant systems.

The *closedir* function closes the directory stream *dirp*, which must have been opened by a previous call to *opendir*. After the stream is closed, *dirp* no longer points to a valid directory stream.

wclosedir is the Unicode version of *closedir*.

Return Value

If *closedir* is successful, it returns 0. Otherwise, *closedir* returns -1 and sets the global variable errno to

EBADF	The *dirp* argument does not point to a valid open directory stream

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

clreol

See also Example Portability

Syntax

```
#include <conio.h.>
void clreol(void);
```

Description

Clears to end of line in text window.

clreol clears all characters from the cursor position to the end of the line within the current text window, without moving the cursor.

Note: This function should not be used in Win32s or Win32 GUI applications.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

clrscr

See also   Example   Portability

Syntax

```
#include <conio.h>
void clrscr(void);
```

Description

Clears the text-mode window.

clrscr clears the current text window and places the cursor in the upper left corner (at position 1,1).

Note: Do not use this function for Win32s or Win32 GUI applications.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

_control87

[See also](#) [Portability](#)

Syntax

```
#include <float.h>
unsigned int _control87(unsigned int newcw, unsigned int mask);
```

Description

Manipulates the floating-point control word.

_*control87* retrieves or changes the floating-point control word.

The floating-point control word is an **unsigned int** that, bit by bit, specifies certain modes in the floating-point package; namely, the precision, infinity, and rounding modes. Changing these modes lets you mask or unmask floating-point exceptions.

_*control87* matches the bits in *mask* to the bits in *newcw*. If a *mask* bit equals 1, the corresponding bit in *newcw* contains the new value for the same bit in the floating-point control word, and _*control87* sets that bit in the control word to the new value.

Here is a simple illustration:

Original control word:		0100	0011	0110	0011
mask:	1000	0001	0100	1111	
newcw:	1110	1001	0000	0101	
Changing bits:	1*xxx*	*xxx*1	*x*0*xx*	0101	

If *mask* equals 0, _*control87* returns the floating-point control word without altering it.

Return Value

The bits in the value returned reflect the new floating-point control word. For a complete definition of the bits returned by _*control87*, see the header file float.h.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

cos, cosl

See also Example Portability

Syntax

```
#include <math.h>
double cos(double x);
long double cosl(long double x);
```

Description

Calculates the cosine of a value.

cos computes the cosine of the input value. The angle is specified in radians.

cosl is the **long double** version; it takes a **long double** argument and returns a **long double** result.

Return Value

cos of a real argument returns a value in the range -1 to 1. Error handling for these functions can be modified through *_matherr* (or *_matherrl*).

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
cos	+	+	+	+	+	+	+
cosl	+		+	+			+

cosh, coshl

See also Example Portability

Syntax

```
#include <math.h>
double cosh(double x);
long double coshl(long double x);
```

Description

Calculates the hyperbolic cosine of a value.

cosh computes the hyperbolic cosine, $(e^x + e^{-x})/2$. *coshl* is the **long double** version; it takes a **long double** argument and returns a **long double** result.

Return Value

cosh returns the hyperbolic cosine of the argument.

When the correct value would create an overflow, these functions return the value HUGE_VAL *(cosh)* or _LHUGE_VAL *(coshl)* with the appropriate sign, and the global variable errno is set to ERANGE. Error handling for these functions can be modified through the functions *_matherr* and *_matherrl.*

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
cosh	+	+	+	+	+	+	+
coshl	+		+	+			+

cprintf

See also Example Portability

Syntax

```
#include <conio.h>
int cprintf(const char *format[, argument, ...]);
```

Description

Writes formatted output to the screen.

cprintf accepts a series of arguments, applies to each a format specifier contained in the format string pointed to by *format*, and outputs the formatted data directly to the current text window on the screen. There must be the same number of format specifiers as arguments.

For details details on format specifiers, see printf Format Specifiers.

The string is written either directly to screen memory or by way of a BIOS call, depending on the value of the global variable _*directvideo*.

Unlike *fprintf* and *printf*, *cprintf* does not translate linefeed characters (\n) into carriage-return/linefeed character pairs (\r\n). Tab characters (specified by \t) are not expanded into spaces.

Note: Do not use this function for Win32s or Win32 GUI applications.

Return Value

cprintf returns the number of characters output.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+			+			+

cputs

See also Example Portability

Syntax

```
#include <conio.h>
int cputs(const char *str);
```

Description

Writes a string to the screen.

cputs writes the null-terminated string *str* to the current text window. It does not append a newline character.

The string is written either directly to screen memory or by way of a BIOS call, depending on the value of the global variable _*directvideo*. Unlike *puts*, *cputs* does not translate linefeed characters (\n) into carriage-return/linefeed character pairs (\r\n).

Note: Do not use this function for Win32s or Win32 GUI applications.

Return Value

cputs returns the last character printed.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+			+			+

_creat, _wcreat

See also Example Portability

Syntax

```
#include <io.h>
int _creat(const char *path, int amode);
int _wcreat(const wchar_t *path, int amode);
```

Description

Creates a new file or overwrites an existing one.

Note: Remember that a backslash in a path requires '\\'.

_creat creates a new file or prepares to rewrite an existing file given by *path*. *amode* applies only to newly created files.

A file created with creat is always created in the translation mode specified by the global variable *_fmode* (O_TEXT or O_BINARY).

If the file exists and the write attribute is set, *creat* truncates the file to a length of 0 bytes, leaving the file attributes unchanged. If the existing file has the read-only attribute set, the creat call fails and the file remains unchanged.

The *_creat* call examines only the S_IWRITE bit of the access-mode word *amode*. If that bit is 1, the file can be written to. If the bit is 0, the file is marked as read-only. All other operating system attributes are set to 0.

amode can be one of the following (defined in sys\stat.h):

Value of amode	Access permission
S_IWRITE	Permission to write
S_IREAD	Permission to read
S_IREAD / S_IWRITE	Permission to read and write (write permission implies read permission)

Return Value

Upon successful completion, *_creat* returns the new file handle, a nonnegative integer; otherwise, it returns -1.

In the event of error, the global variable errno is set to one of the following:

EACCES	Permission denied
ENOENT	Path or file name not found
EMFILE	Too many open files

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

creatnew

See also Example Portability

Syntax

```
#include <io.h>
int creatnew(const char *path, int mode);
```

Description

Creates a new file.

creatnew is identical to *_rtl_creat* with one exception: If the file exists, *creatnew* returns an error and leaves the file untouched.

The *mode* argument to *creatnew* can be zero or an OR-combination of any one of the following constants (defined in dos.h):

FA_HIDDEN	Hidden file
FA_RDONLY	Read-only attribute
FA_SYSTEM	System file

Return Value

Upon successful completion, *creat* returns the new file handle, a nonnegative integer; otherwise, it returns -1.

In the event of error, the global variable errno is set to one of the following values:

EACCES	Permission denied
EEXIST	File already exists
EMFILE	Too many open files
ENOENT	Path or file name not found

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

creattemp

See also   Example   Portability

Syntax

```
#include <io.h>
int creattemp(char *path, int attrib);
```

Description

Creates a unique file in the directory associated with the path name.

A file created with *creattemp* is always created in the translation mode specified by the global variable *_fmode* (O_TEXT or O_BINARY).

path is a path name ending with a *backslash* (\). A unique file name is selected in the directory given by *path*. The newly created file name is stored in the *path* string supplied. *path* should be long enough to hold the resulting file name. The file is not automatically deleted when the program terminates.

creattemp accepts *attrib*, a DOS attribute word. Upon successful file creation, the file pointer is set to the beginning of the file. The file is opened for both reading and writing.

The *attrib* argument to *creattemp* can be zero or an OR-combination of any one of the following constants (defined in dos.h):

FA_HIDDEN	Hidden file
FA_RDONLY	Read-only attribute
FA_SYSTEM	System file

Return Value

Upon successful completion, the new file handle, a nonnegative integer, is returned; otherwise, -1 is returned.

In the event of error, the global variable errno is set to one of the following values:

EACCES	Permission denied
EMFILE	Too many open files
ENOENT	Path or file name not found

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

_crotl, _crotr

See also Example Portability

Syntax

```
#include <stdlib.h>
unsigned char _crotl(unsigned char val, int count);
unsigned char _crotr(unsigned char val, int count);
```

Description

Rotates an unsigned char left or right.

_*crotl* rotates the given *val* to the left *count* bits. _*crotr* rotates the given *val* to the right *count* bits.

The argument *val* is an **unsigned char**, or its equivalent in decimal or hexadecimal form.

Return Value

The functions return the rotated word:

- _*crotl* returns the value of *val* left-rotated *count* bits.
- _*crotr* returns the value of *val* right-rotated *count* bits.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

cscanf

See also Example Portability

Syntax

```
#include <conio.h>
int cscanf(char *format[, address, ...]);
```

Description

Scans and formats input from the console.

cscanf scans a series of input fields one character at a time, reading directly from the console. Then each field is formatted according to a format specifier passed to *cscanf* in the format string pointed to by format. Finally, *cscanf* stores the formatted input at an address passed to it as an argument following *format*, and echoes the input directly to the screen. There must be the same number of format specifiers and addresses as there are input fields.

Note: For details on format specifiers, see scanf Format Specifiers.

cscanf might stop scanning a particular field before it reaches the normal end-of-field (whitespace) character, or it might terminate entirely for a number of reasons. See *scanf* for a discussion of possible causes.

Note: Do not use this function for Win32s or Win32 GUI applications.

Return Value

cscanf returns the number of input fields successfully scanned, converted, and stored; the return value does not include scanned fields that were not stored. If no fields were stored, the return value is 0.

If *cscanf* attempts to read at end-of-file , the return value is EOF.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+			+			+

actime, _wctime

See also Example Portability

Syntax

```
#include <time.h>
char *ctime(const time_t *time);
wchar_t *_wctime(const time_t *time);
```

Description

Converts date and time to a string.

ctime converts a time value pointed to by *time* (the value returned by the function *time*) into a 26-character string in the following form, terminating with a newline character and a null character:

```
Mon Nov 21 11:31:54 1983\n\0
```

All the fields have constant width.

The global long variable _*timezone* contains the difference in seconds between GMT and local standard time (in PST, _*timezone* is 8*60*60). The global variable _*daylight* is nonzero *if and only if* the standard U.S. _daylight saving time conversion should be applied. These variables are set by the *tzset* function, not by the user program directly.

Return Value

ctime returns a pointer to the character string containing the date and time. The return value points to static data that is overwritten with each call to *ctime*.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

cwait

See also Example Portability

Syntax

```
#include <process.h>
int cwait(int *statloc, int pid, int action);
```

Description

Waits for child process to terminate.

The *cwait* function waits for a child process to terminate. The process ID of the child to wait for is *pid*. If *statloc* is not NULL, it points to the location where *cwait* will store the termination status. The *action* specifies whether to wait for the process alone, or for the process and all of its children.

If the child process terminated normally (by calling *exit*, or returning from *main*), the termination status word is defined as follows:

Bits 0-7	Zero
Bits 8-15	The least significant byte of the return code from the child process. This is the value that is passed to *exit*, or is returned from *main*. If the child process simply exited from *main* without returning a value, this value will be unpredictable.

If the child process terminated abnormally, the termination status word is defined as follows:

Bits 0-7 Termination information about the child:

1 Critical error abort.

2 Execution fault, protection exception.

3 External termination signal.

Bits 8-15 Zero

If *pid* is 0, *cwait* waits for any child process to terminate. Otherwise, *pid* specifies the process ID of the process to wait for; this value must have been obtained by an earlier call to an asynchronous *spawn* function.

The acceptable values for *action* are WAIT_CHILD, which waits for the specified child only, and WAIT_GRANDCHILD, which waits for the specified child *and* all of its children. These two values are defined in process.h.

Return Value

When *cwait* returns after a normal child process termination, it returns the process ID of the child.

When *cwait* returns after an abnormal child termination, it returns -1 to the parent and sets errno to EINTR (the child process terminated abnormally).

If *cwait* returns without a child process completion, it returns a -1 value and sets *errno* to one of the following values:

ECHILD	No child exists or the pid value is bad
EINVAL	A bad action value was specified

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
			+			+

delline

See also Example Portability

Syntax

```
#include <conio.h>
void delline(void);
```

Description

Deletes line in text window.

delline deletes the line containing the cursor and moves all lines below it one line up. *delline* operates within the currently active text window.

Note: Do not use this function for Win32s or Win32 GUI applications.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

difftime

See also Example Portability

Syntax

```
#include <time.h>
double difftime(time_t time2, time_t time1);
```

Description

Computes the difference between two times.

difftime calculates the elapsed time in seconds, from time1 to time2.

Return Value

difftime returns the result of its calculation as a **double**.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

disable, _disable, enable, _enable

Examples Portability

Syntax

```
#include <dos.h>
void disable(void);
void _disable(void);
void enable(void);
void _enable(void);
```

Description

Disables and enables interrupts.

These macros are designed to provide a programmer with flexible hardware interrupt control.

disable and *_disable* macros disable interrupts. Only the NMI (non-maskable interrupt) is allowed from any external device.

enable and *_enable* macros enable interrupts, allowing any device interrupts to occur.

Return Value

None.

Examples

disable

_disable

enable

_enable

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			

div

See also   Example   Portability

Syntax

```
#include <stdlib.h>
div_t  div(int numer, int denom);
```

Description

Divides two integers, returning quotient and remainder.

div divides two integers and returns both the quotient and the remainder as a *div_t* type. *numer* and *denom* are the numerator and denominator, respectively. The *div_t* type is a structure of integers defined (with **typedef**) in stdlib.h as follows:

```
typedef struct {
   int  quot;      /* quotient */
   int  rem;       /* remainder */
} div_t;
```

Return Value

div returns a structure whose elements are *quot* (the quotient) and *rem* (the remainder).

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+	+	+	+

dup

See also Example Portability

Syntax

```
#include <io.h>
int dup(int handle);
```

Description

Duplicates a file handle.

dup creates a new file handle that has the following in common with the original file handle:

- Same open file or device
- Same file pointer (that is, changing the file pointer of one changes the other)
- Same access mode (read, write, read/write)

handle is a file handle obtained from a call to *creat, open, dup, dup2, _rtl_creat*, or *_rtl_open*.

Return Value

Upon successful completion, *dup* returns the new file handle, a nonnegative integer; otherwise, *dup* returns -1.

In the event of error, the global variable errno is set to one of the following values:

EBADF	Bad file number
EMFILE	Too many open files

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

dup2

See also   Example   Portability

Syntax

```
#include <io.h>
int dup2(int oldhandle, int newhandle);
```

Description

Duplicates a file handle (oldhandle) onto an existing file handle (newhandle).

dup2 creates a new file handle that has the following in common with the original file handle:

- Same open file or device
- Same file pointer (that is, changing the file pointer of one changes the other)
- Same access mode (read, write, read/write)

dup2 creates a new handle with the value of *newhandle*. If the file associated with *newhandle* is open when *dup2* is called, the file is closed.

newhandle and *oldhandle* are file handles obtained from a *creat, open, dup,* or *dup2* call.

Return Value

dup2 returns 0 on successful completion, -1 otherwise.

In the event of error, the global variable errno is set to one of the following values:

EBADF   Bad file number

EMFILE Too many open files

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

ecvt

See also Example Portability

Syntax

```
#include <stdlib.h>
char *ecvt(double value, int ndig, int *dec, int *sign);
```

Description

Converts a floating-point number to a string.

ecvt converts *value* to a null-terminated string of *ndig* digits, starting with the leftmost significant digit, and returns a pointer to the string. The position of the decimal point relative to the beginning of the string is stored indirectly through *dec* (a negative value for *dec* means that the decimal lies to the left of the returned digits). There is no decimal point in the string itself. If the sign of *value* is negative, the word pointed to by *sign* is nonzero; otherwise, it's 0. The low-order digit is rounded.

Return Value

The return value of *ecvt* points to static data for the string of digits whose content is overwritten by each call to *ecvt* and *fcvt*.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

_ _emit_ _

Example Portability

Syntax

```
#include <dos.h>
void _ _emit_ _(argument, ...);
```

Description

Inserts literal values directly into code.

_ _*emit*_ _ is an inline function that lets you insert literal values directly into object code as it is compiling. It is used to generate machine language instructions without using inline assembly language or an assembler.

Generally the arguments of an _ _*emit*_ _ call are single-byte machine instructions. However, because of the capabilities of this function, more complex instructions, complete with references to C variables, can be constructed.

You should use this function only if you are familiar with the machine language of the 80x86 processor family. You can use this function to place arbitrary bytes in the instruction code of a function; if any of these bytes is incorrect, the program misbehaves and can easily crash your machine. Borland C++ does not attempt to analyze your calls for correctness in any way. If you encode instructions that change machine registers or memory, Borland C++ will not be aware of it and might not properly preserve registers, as it would in many cases with inline assembly language (for example, it recognizes the usage of SI and DI registers in inline instructions). You are completely on your own with this function.

You must pass at least one argument to _ _*emit*_ _; any number can be given. The arguments to this function are not treated like any other function call arguments in the language. An argument passed to _ _*emit*_ _ will not be converted in any way.

There are special restrictions on the form of the arguments to _ _*emit*_ _. Arguments must be in the form of expressions that can be used to initialize a static object. This means that integer and floating-point constants and the addresses of static objects can be used. The values of such expressions are written to the object code at the point of the call, exactly as if they were being used to initialize data. The address of a parameter or auto variable, plus or minus a constant offset, can also be used. For these arguments, the offset of the variable from BP is stored.

The number of bytes placed in the object code is determined from the type of the argument, except in the following cases:

- If a signed integer constant (that is 0x90) appears that fits within the range of 0 to 255, it is treated as if it were a character.
- If the address of an auto or parameter variable is used, a byte is written if the offset of the variable from BP is between -128 and 127; otherwise, a word is written.

Simple bytes are written as follows:

```
_ _emit_ _(0x90);
```

If you want a word written, but the value you are passing is under 255, simply cast it to **unsigned** using one of these methods:

```
_ _emit_ _(0xB8, (unsigned)17);
_ _emit_ _(0xB8, 17u);
```

Two- or four-byte address values can be forced by casting an address to **void near** * or **void far** *, respectively.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

_endthread

See also Examples Portability

Syntax

```
#include <process.h>
void _endthread(void);
```

Description

Terminates execution of a thread.

The *_endthread* function terminates the currently executing thread. The thread must have been started by an earlier call to _beginthread or _beginthreadNT.

This function is available in the multithread libraries; it is not in the single-thread libraries.

Return Value

The function does not return a value.

Examples

_beginthread (Win32s version)

_beginthreadNT (Windows NT version)

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
			+			+

eof

See also   Example   Portability

Syntax

```
#include <io.h>
int eof(int handle);
```

Description

Checks for end-of-file.

eof determines whether the file associated with *handle* has reached end-of-file.

Return Value

If the current position is end-of-file, *eof* returns the value 1; otherwise, it returns 0. A return value of -1 indicates an error; the global variable errno is set to

EBADF	Bad file number

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

execl, execle, execlp, execlpe, execv, execve, execvp, execvpe

See also Examples Portability

Syntax

```
#include <process.h>
int execl(char *path, char *arg0 *arg1, ..., *argn, NULL);
int _wexecl(wchar_t *path, wchar_t *arg0 *arg1, ..., *argn, NULL);
int execle(char *path, char *arg0, *arg1, ..., *argn, NULL, char **env);
int _wexecle(wchar_t *path, wchar_t *arg0, *arg1, ..., *argn, NULL, wchar_t
  **env);


int execlp(char *path, char *arg0,*arg1, ..., *argn, NULL);
int _wexeclp(wchar_t *path, wchar_t *arg0,*arg1, ..., *argn, NULL);
int execlpe(char *path, char *arg0, *arg1, ..., *argn, NULL, char **env);
int _wexeclpe(wchar_t *path, wchar_t *arg0, *arg1, ..., *argn, NULL, wchar_t
  **env);

int execv(char *path, char *argv[]);
int _wexecv(wchar_t *path, wchar_t *argv[]);
int execve(char *path, char *argv[], char **env);
int _wexecve(wchar_t *path, wchar_t *argv[], wchar_t **env);

int execvp(char *path, char *argv[]);
int _wexecvp(wchar_t *path, wchar_t *argv[]);
int execvpe(char *path, char *argv[], char **env);
int _wexecvpe(wchar_t *path, wchar_t *argv[], wchar_t **env);
```

Description

Loads and runs other programs.

The functions in the *exec...* family load and run (execute) other programs, known as *child processes*. When an *exec...* call succeeds, the child process overlays the *parent process*. There must be sufficient memory available for loading and executing the child process.

path is the file name of the called child process. The *exec...* functions search for *path* using the standard search algorithm:

- If no explicit extension is given, the functions search for the file as given. If the file is not found, they add .EXE and search again. If not found, they add .COM and search again. If found, the command processor, COMSPEC (Windows) or COMMAND.COM (DOS), is used to run the batch file.
- If an explicit extension or a period is given, the functions search for the file exactly as given.

The suffixes *l*, *v*, *p*, and *e* added to the *exec...* "family name" specify that the named function operates with certain capabilities.

l specifies that the argument pointers (*arg0, arg1, ..., argn*) are passed as separate arguments. Typically, the *l* suffix is used when you know in advance the number of arguments to be passed.

v specifies that the argument pointers (*argv[0] ..., arg[n]*) are passed as an array of pointers. Typically, the *v* suffix is used when a variable number of arguments is to be passed.

p specifies that the function searches for the file in those directories specified by the PATH environment variable (without the *p* suffix, the function searches only the current working directory). If the *path* parameter does not contain an explicit directory, the function searches first the current directory, then the directories set with the PATH environment variable.

e specifies that the argument *env* can be passed to the child process, letting you alter the environment for the child process. Without the *e* suffix, child processes inherit the environment of the parent process.

Each function in the *exec...* family must have one of the two argument-specifying suffixes (either *l* or *v*).

The path search and environment inheritance suffixes *(p* and *e*) are optional; for example:

- *execl* is an *exec*... function that takes separate arguments, searches only the root or current directory for the child, and passes on the parent's environment to the child.
- *execvpe* is an *exec*... function that takes an array of argument pointers, incorporates PATH in its search for the child process, and accepts the *env* argument for altering the child's environment.

The *exec...* functions must pass at least one argument to the child process (*arg0 or argv[0]*); this argument is, by convention, a copy of *path*. (Using a different value for this 0th argument won't produce an error.)

path is available for the child process.

When the *l* suffix is used, *arg0* usually points to *path*, and *arg1*, *..., argn* point to character strings that form the new list of arguments. A mandatory null following *argn* marks the end of the list.

When the *e* suffix is used, you pass a list of new environment settings through the argument *env*. This environment argument is an array of character pointers. Each element points to a null-terminated character string of the form

```
envvar = value
```

where *envvar* is the name of an environment variable, and *value* is the string value to which *envvar* is set. The last element in *env* is null. When *env* is null, the child inherits the parents' environment settings.

The combined length of *arg0* + *arg1* + *...* + *argn* (or of *argv[0]* + *argv[1]* + *...* + *argn[n]*), including space characters that separate the arguments, must be less than 128 bytes for a 16-bit application, or 260 bytes for Win32 application. Null terminators are not counted.

When an *exec...* function call is made, any open files remain open in the child process.

Return Value

If successful, the *exec...* functions do not return. On error, the *exec...* functions return -1, and the global variable errno is set to one of the following values:

EACCES	Permission denied
EMFILE	Too many open files
ENOENT	Path or file name not found
ENOEXEC	Exec format error
ENOMEM	Not enough memory

Examples

execl

execle

execlp

execlpe

execv

execve

execvp

execvpe

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+			+			+

_exit

See also Example Portability

Syntax

```
#include <stdlib.h>
void _exit(int status);
```

Description

Terminates program.

_*exit* terminates execution without closing any files, flushing any output, or calling any exit functions.

The calling process uses *status* as the exit status of the process. Typically a value of 0 is used to indicate a normal exit, and a nonzero value indicates some error.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

exit

See also   Example   Portability

Syntax

```
#include <stdlib.h>
void exit(int status);
```

Description

Terminates program.

exit terminates the calling process. Before termination, all files are closed, buffered output (waiting to be output) is written, and any registered "exit functions" (posted with *atexit*) are called.

status is provided for the calling process as the exit status of the process. Typically a value of 0 is used to indicate a normal exit, and a nonzero value indicates some error. It can be, but is not required, to be set with one of the following:

EXIT_FAILURE	Abnormal program termination; signal to operating system that program has terminated with an error
EXIT_SUCCESS	Normal program termination

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

exp, expl

See also Example Portability

Syntax

```
#include <math.h>
double exp(double x);
long double expl(long double x);
```

Description

Calculates the exponential *e* to the *x*.

expl is the **long double** version; it takes a **long double** argument and returns a **long double** result.

Return Value

exp returns *e* to the *x*.

Sometimes the arguments passed to these functions produce results that overflow or are incalculable. When the correct value overflows, *exp* returns the value HUGE_VAL and *expl* returns _LHUGE_VAL. Results of excessively large magnitude cause the global variable errno to be set to

ERANGE	Result out of range

On underflow, these functions return 0.0, and the global variable errno is not changed. Error handling for these functions can be modified through the functions *_matherr* and *_matherrl*.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
exp	+	+	+	+	+	+	+
expl	+		+	+			+

_expand

See also   Example   Portability

Syntax

```
#include <malloc.h>
void *_expand(void *block, size_t size);
```

Description

Grows or shrinks a heap block in place.

This function attempts to change the size of an allocated memory *block* without moving the block's location in the heap. The data in the *block* are not changed, up to the smaller of the old and new sizes of the block. The block must have been allocated earlier with *malloc*, *calloc*, or *realloc*, and must not have been freed.

Return Value

If *_expand* is able to resize the block without moving it, *_expand* returns a pointer to the block, whose address is unchanged. If *_expand* is unsuccessful, it returns a NULL pointer and does not modify or resize the block.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
			+			+

fabs, fabsl

See also Example Portability

Syntax

```
#include <math.h>
double fabs(double x);
long double fabsl(long double x);
```

Description

Returns the absolute value of a floating-point number.

fabs calculates the absolute value of *x*, a double. *fabsl* is the **long double** version; it takes a **long double** argument and returns a **long double** result.

Return Value

fabs and *fabsl* return the absolute value of *x*.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
fabs	+	+	+	+	+	+	+
fabsl	+		+	+			+

fclose

See also Example Portability

Syntax

```
#include <stdio.h>
int fclose(FILE *stream);
```

Description

Closes a stream.

fclose closes the named stream. All buffers associated with the stream are flushed before closing. System-allocated buffers are freed upon closing. Buffers assigned with *setbuf* or *setvbuf* are not automatically freed. (But if *setvbuf* is passed null for the buffer pointer it *will* free it upon close.)

Return Value

fclose returns 0 on success. It returns EOF if any errors were detected.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

_fcloseall

See also Example Portability

Syntax

```
#include <stdio.h>
int _fcloseall(void);
```

Description

Closes open streams.

_*fcloseall* closes all open streams except

stdin

stdout

stdprn

stderr

stdauxstdstreams

When _*fcloseall* flushes the associated buffers before closing a stream. The buffers allocated by the sytem are released.

Note: stdprn and stdaux streams are not available in OS/2 and Win32.

Return Value

_*fcloseall* returns the total number of streams it closed. The _*fcloseall* function returns EOF if any errors were detected.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

fcvt

See also Example Portability

Syntax

```
#include <stdlib.h>
char *fcvt(double value, int ndig, int *dec, int *sign);
```

Description

Converts a floating-point number to a string.

fcvt converts *value* to a null-terminated string digit starting with the leftmost significant digit with *ndig* digits to the right of the decimal point. *fcvt* then returns a pointer to the string. The position of the decimal point relative to the beginning of the string is stored indirectly through *dec* (a negative value for *dec* means to the left of the returned digits). There is no decimal point in the string itself. If the sign of *value* is negative the word pointed to by *sign* is nonzero; otherwise it is 0.

The correct digit has been rounded for the number of digits to the right of the decimal point specified by *ndig*.

Return Value

The return value of *fcvt* points to static data whose content is overwritten by each call to *fcvt* and *ecvt*.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

_fdopen, _wfdopen

See also Example Portability

Syntax

```
#include <stdio.h>
FILE *_fdopen(int handle, char *type);
FILE *_wfdopen(int handle, wchar_t *type);
```

Description

Associates a stream with a file handle.

_fdopen associates a stream with a file handle obtained from creat, dup, dup2, or open.

The type of stream must match the mode of the open *handle*.

The *type* string used in a call to *_fdopen* is one of the following values:

Value	Description
r	Open for reading only. *_fdopen* returns NULL if the file cannot be opened.
w	Create for writing. If the file already exists, its contents are overwritten.
a	Append; open for writing at end-of-file or create for writing if the file does not exist.
r+	Open an existing file for update (reading and writing). *_fdopen* returns NULL if the file cannot be opened.
w+	Create a new file for update. If the file already exists, its contents are overwritten.
a+	Open for append; open (or create if the file does not exist) for update at the end of the file.

To specify that a given file is being opened or created in text mode, append *t* to the value of the *type* string (for example, `rt` or `w+t`).

Similarly, to specify binary mode append *b* to the *type* string (for example, `rb` or `w+b`).

If *t* or *b* is not given in the type string, the mode is governed by the global variable _fmode.

If *_fmode* is set to O_BINARY, files will be opened in binary mode.

If *_fmode* is set to O_TEXT, files will be opened in text mode.

Note: The O_... constants are defined in fcntl.h.

When a file is opened for update, both input and output can be done on the resulting stream; however,

- output cannot be directly followed by input without an intervening fseekor rewind
- input cannot be directly followed by output without an intervening *fseek*, *rewind*, or an input that encounters end-offile

Return Value

On successful completion *_fdopen* returns a pointer to the newly opened stream. In the event of error it returns NULL.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

feof

See also Example Portability

Syntax

```
#include <stdio.h>
int feof(FILE *stream);
```

Description

Detects end-of-file on a stream.

feof is a macro that tests the given stream for an end-of-file indicator. Once the indicator is set read operations on the file return the indicator until *rewind* is called or the file is closed. The end-of-file indicator is reset with each input operation.

Return Value

feof returns nonzero if an end-of-file indicator was detected on the last input operation on the named stream and 0 if end-of-file has not been reached.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

ferror

See also   Example   Portability

Syntax

```
#include <stdio.h>
int ferror(FILE *stream);
```

Description

Detects errors on stream.

ferror is a macro that tests the given stream for a read or write error. If the stream's error indicator has been set it remains set until *clearerr* or *rewind* is called or until the stream is closed.

Return Value

ferror returns nonzero if an error was detected on the named stream.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

fflush

See also Example Portability

Syntax

```
#include <stdio.h>
int fflush(FILE *stream);
```

Description

Flushes a stream.

If the given stream has buffered output *fflush* writes the output for *stream* to the associated file.

The stream remains open after *fflush* has executed. *fflush* has no effect on an unbuffered stream.

Return Value

fflush returns 0 on success. It returns EOF if any errors were detected.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

fgetc, fgetwc

See also Example Portability

Syntax

```
#include <stdio.h>
int fgetc(FILE *stream);
wint_t fgetwc(FILE *stream);
```

Description

Gets character from stream.

fgetc returns the next character on the named input stream.

Return Value

On success *fgetc* returns the character read after converting it to an **int** without sign extension. On end-of-file or error it returns EOF.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

_fgetchar, _fgetwchar

See also Example Portability

Syntax

```
#include <stdio.h>
int _fgetchar(void);
wint_t _fgetwchar(void);
```

Description

Reads a character from stdin.

_fgetchar returns the next character from stdin. It is defined as fgetc(stdin).

Note: For Win32s or Win32 GUI applications, stdin must be redirected.

Return Value

On success *_fgetchar* returns the character read after converting it to an **int** without sign extension. On end-of-file or error it returns EOF.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

fgetpos

See also Example Portability

Syntax

```
#include <stdio.h>
int fgetpos(FILE *stream, fpos_t *pos);
```

Description

Gets the current file pointer.

fgetpos stores the position of the file pointer associated with the given stream in the location pointed to by *pos*. The exact value is unimportant; its value is opaque except as a parameter to subsequent *fsetpos* calls.

Return Value

On success *fgetpos* returns 0. On failure it returns a nonzero value and sets the global variable *errno* to

EBADF	Bad file number
EINVAL	Invalid number

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+	+	+	+

fgets, fgetws

See also Example Portability

Syntax

```
#include <stdio.h>
char *fgets(char *s, int n, FILE *stream);
wchar_t *fgetws(wchar_t *s, int n, FILE *stream); // Unicode version
```

Description

Gets a string from a stream.

fgets reads characters from *stream* into the string *s*. The function stops reading when it reads either *n* - 1 characters or a newline character whichever comes first. *fgets* retains the newline character at the end of *s*. A null byte is appended to s to mark the end of the string.

Return Value

On success *fgets* returns the string pointed to by *s*; it returns NULL on end-of-file or error.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

filelength

See also Example Portability

Syntax

```
#include <io.h>
long filelength(int handle);
```

Description

Gets file size in bytes.

filelength returns the length (in bytes) of the file associated with *handle*.

Return Value

On success *filelength* returns a **long** value the file length in bytes. On error it returns -1 and the global variable *errno* is set to

EBADF	Bad file number

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

_fileno

See also Example Portability

Syntax

```
#include <stdio.h>
int _fileno(FILE *stream);
```

Description

Gets the file handle.

_fileno is a macro that returns the file handle for the given stream. If *stream* has more than one handle *_fileno* returns the handle assigned to the stream when it was first opened.

Return Value

_fileno returns the integer file handle associated with *stream*.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

_findfirst, _wfindfirst

See also Example Portability

Syntax

```
#include <dir.h>
int _findfirst(const char *pathname, struct ffblk *ffblk, int attrib);
int _wfindfirst(const wchar_t *pathname, struct _wffblk *ffblk, int attrib);
```

Description

Searches a disk directory.

_findfirst begins a search of a disk directory for files specifed by attributes or wildcards.

pathname is a string with an optional drive specifier path and file name of the file to be found. Only the file name portion can contain wildcard match characters (such as ? or *). If a matching file is found the *ffblk* structure is filled with the file-directory information.

When Unicode is defined, the*_wfindfirst* function uses the following *_wffblk* structure.

```
struct  _wffblk   {
    long            ff_reserved;
    long            ff_fsize;
    unsigned long   ff_attrib;
    unsigned short  ff_ftime;
    unsigned short  ff_fdate;
    wchar_t         ff_name[256];
};
```

For Win32, the format of the structure *ffblk* is as follows:

```
struct ffblk {
   long           ff_reserved;
   long           ff_fsize;       /* file size */
   unsigned long  ff_attrib;      /* attribute found */
   unsigned short ff_ftime;       /* file time */
   unsigned short ff_fdate;       /* file date */
   char           ff_name[256];   /* found file name */
   };
```

attrib is a file-attribute byte used in selecting eligible files for the search. *attrib* should be selected from the following constants defined in dos.h:

FA_RDONLY	Read-only attribute
FA_HIDDEN	Hidden file
FA_SYSTEM	System file
FA_LABEL	Volume label
FA_DIREC	Directory
FA_ARCH	Archive

A combination of constants can be ORed together.

For more detailed information about these attributes refer to your operating system documentation.

ff_ftime and *ff_fdate* contain bit fields for referring to the current date and time. The structure of these fields was established by the operating system. Both are 16-bit structures divided into three fields.

ff_ftime:

Bits 0 to 4	The result of seconds divided by 2 (for example 10 here means 20 seconds)
Bits 5 to 10	Minutes
Bits 11 to 15	Hours

ff_fdate:

Bits 0-4	Day
Bits 5-8	Month
Bits 9-15	Years since 1980 (for example 9 here means 1989)

The structure *ftime* declared in io.h uses time and date bit fields similar in structure to *ff_ftime* and *ff_fdate*.

Return Value

_findfirst returns 0 on successfully finding a file matching the search *pathname*.

When no more files can be found, or if there is an error in the file name:

- -1 is returned
- errno is set to

  ENOENT Path or file name not found
- _doserrno is set to one of the following values:

  ENMFILE No more files

  ENOENT Path or file name not found

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

_findnext, _wfindnext

See also Example Portability

Syntax

```
#include <dir.h>
int _findnext(struct ffblk *ffblk);
int _wfindnext(struct _wffblk *ffblk);
```

Description

Continues _findfirst search.

_findnext is used to fetch subsequent files that match the *pathname* given in *findfirst. ffblk* is the same block filled in by the *findfirst* call. This block contains necessary information for continuing the search. One file name for each call to *_findnext* will be returned until no more files are found in the directory matching the *pathname*.

Return Value

_findnext returns 0 on successfully finding a file matching the search *pathname*. When no more files can be found or if there is an error in the file name

-1 is returned

errno is set to

ENOENT Path or file name not found

_doserrno is set to one of the following values:

ENMFILE No more files

ENOENT Path or file name not found

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

floor, floorl

See also   Example   Portability

Syntax

```
#include <math.h>
double floor(double x);
long double floorl(long double x);
```

Description

Rounds down.

floor finds the largest integer not greater than *x*.

floorl is the **long double** version; it takes a **long double** argument and returns a **long double** result.

Return Value

floor returns the integer found as a **double**. *floorl* returns the integer found as a **long double**.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
floor	+	+	+	+	+	+	+
floorl	+		+	+			+

_flushall

See also Example Portability

Syntax

```
#include <stdio.h>
int _flushall(void);
```

Description

Flushes all streams.

_flushall clears all buffers associated with open input streams and writes all buffers associated with open output streams to their respective files. Any read operation following *_flushall* reads new data into the buffers from the input files. Streams stay open after *_flushall* executes.

Return Value

_flushall returns an integer the number of open input and output streams.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

fmod, fmodl

See also Example Portability

Syntax

```
#include <math.h>
double fmod(double x, double y);
long double fmodl(long double x, long double y);
```

Description

Calculates *x* modulo *y*, the remainder of *x/y*.

fmod calculates *x* modulo *y* (the remainder *f*, where $x = ay + f$ for some integer *a*, and $0 \le f < y$).

fmodl is the **long double** version; it takes **long double** arguments and returns a **long double** result.

Return Value

fmod and *fmodl* return the remainder *f* where $x = ay + f$ (as described above). When y = 0, *fmod* and *fmodl* return 0.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
fmod	+	+	+	+	+	+	+
fmodl	+		+	+			+

fnmerge, _wfnmerge

See also Example Portability

Syntax

```
#include <dir.h>
void fnmerge(char *path, const char *drive, const char *dir, const char
  *name, const char *ext);
void _wfnmerge(wchar_t *path, const wchar_t *drive, const wchar_t *dir,
  const wchar_t *name, const wchar_t *ext );
```

Description

Builds a path from component parts.

fnmerge makes a path name from its components. The new path name is

```
X:\DIR\SUBDIR\NAME.EXT
```

where:

drive	=	X
dir	=	\\DIR\\SUBDIR\\
name	=	NAME
ext	=	.EXT

If *drive* is empty or NULL, no drive is inserted in the path name. If it is missing a trailing colon (:), a colon is inserted in the path name.

If *dir* is empty or NULL, no directory is inserted in the path name. If it is missing a trailing slash (\ or /), a backslash is inserted in the path name.

If *name* is empty or NULL, no file name is inserted in the path name.

If *ext* is empty or NULL, no extension is inserted in the path name. If it is missing a leading period (.), a period is inserted in the path name.

fnmerge assumes there is enough space in *path* for the constructed path name. The maximum constructed length is MAXPATH. MAXPATH is defined in dir.h.

fnmerge and *fnsplit* are invertible; if you split a given *path* with *fnsplit* then merge the resultant components with *fnmerge* you end up with *path*.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

fnsplit, _wfnsplit

See also Example Portability

Syntax

```
#include <dir.h>
int fnsplit(const char *path, char *drive, char *dir, char *name, char
  *ext);
int _wfnsplit(const wchar_t *path, wchar_t *drive, wchar_t *dir, wchar_t
  *name, wchar_t *ext );
```

Description

Splits a full path name into its components.

fnsplit takes a file's full path name *(path)* as a string in the form X:\DIR\SUBDIR\NAME.EXT and splits *path* into its four components. It then stores those components in the strings pointed to by *drive, dir, name*, and *ext*. All five components must be passed but any of them can be a null which means the corresponding component will be parsed but not stored. If any path component is null, that component corresponds to a non-NULL, empty string.

The maximum sizes for these strings are given by the constants *MAXDRIVE, MAXDIR, MAXPATH, MAXFILE*, and *MAXEXT* (defined in dir.h) and each size includes space for the null-terminator.

Constant	Max 16-bit	Max 32-bit	String
MAXPATH	80	256	path
MAXDRIVE	3	3	drive; includes colon (:)
MAXDIR	66	260	dir; includes leading and trailing backslashes (\)
MAXFILE	9	256	name
MAXEXT	5	256	ext; includes leading dot (.)

fnsplit assumes that there is enough space to store each non-null component.

When *fnsplit* splits *path* it treats the punctuation as follows:

- *drive* includes the colon (C:, A:, and so on)
- *dir* includes the leading and trailing backslashes (\BC\include\, \source\ ,and so on)
- *name* includes the file name
- *ext* includes the dot preceding the extension (.C, .EXE, and so on).

fnmerge and *fnsplit* are invertible; if you split a given *path* with *fnsplit* then merge the resultant components with *fnmerge* you end up with *path*.

Return Value

fnsplit returns an integer (composed of five flags defined in dir.h) indicating which of the full path name components were present in *path*. These flags and the components they represent are

EXTENSION	An extension
FILENAME	A file name
DIRECTORY	A directory (and possibly subdirectories)
DRIVE	A drive specification (see dir.h)
WILDCARDS	Wildcards (* or ?)

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

fopen, _wfopen

See also Example Portability

Syntax

```
#include <stdio.h>
FILE *fopen(const char *filename, const char *mode);
FILE *_wfopen(const wchar_t *filename, const wchar_t *mode);
```

Description

Opens a stream.

fopen opens the file named by *filename* and associates a stream with it. *fopen* returns a pointer to be used to identify the stream in subsequent operations.

The *mode* string used in calls to *fopen* is one of the following values:

Value	Description
r	Open for reading only.
w	Create for writing. If a file by that name already exists, it will be overwritten.
a	Append; open for writing at end-of-file or create for writing if the file does not exist.
r+	Open an existing file for update (reading and writing).
w+	Create a new file for update (reading and writing). If a file by that name already exists, it will be overwritten.
a+	Open for append; open (or create if the file does not exist) for update at the end of the file.

To specify that a given file is being opened or created in text mode append a *t* to the *mode* string (*rt w+t* and so on). Similarly to specify binary mode append a *b* to the *mode* string (*wb a+b* and so on). *fopen* also allows the *t* or *b* to be inserted between the letter and the + character in the mode string; for example *rt+* is equivalent to *r+t*.

If *a t* or *b* is not given in the *mode* string the mode is governed by the global variable *_fmode*. If *_fmode* is set to O_BINARY files are opened in binary mode. If *_fmode* is set to O_TEXT they are opened in text mode. These O_... constants are defined in fcntl.h.

When a file is opened for update, both input and output can be done on the resulting stream; however,

- output cannot be directly followed by input without an intervening fseek or rewind
- input cannot be directly followed by output without an intervening fseek, rewind, or an input that encounters end-offile

Return Value

On successful completion *fopen* returns a pointer to the newly opened stream. In the event of error it returns NULL.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

_fpreset

See also Portability

Syntax

```
#include <float.h>
void _fpreset(void);
```

Description

Reinitializes floating-point math package.

_fpreset reinitializes the floating-point math package. This function is usually used in conjunction with *system* or the *exec...* or *spawn...* functions. It is also used to recover from floating-point errors before calling *longjmp*.

Note: If an 80x87 coprocessor is used in a program a child process (executed by the system, or by an exec... or spawn... function) might alter the parent process' floating-point state.

If you use an 80x87 take the following precautions:

- Do not call *system* or an *exec...* or *spawn...* function while a floating-point expression is being evaluated.
- Call *_fpreset* to reset the floating-point state after using *system exec...* or *spawn...* if there is *any* chance that the child process performed a floating-point operation with the 80x87.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

fprintf, fwprintf

See also   Example   Portability

Syntax

```
#include <stdio.h>
int fprintf(FILE *stream, const char *format[, argument, ...]);
int fwprintf(FILE *stream, const wchar_t *format[, argument, ...]);
```

Description

Writes formatted output to a stream.

fprintf accepts a series of arguments applies to each a format specifier contained in the format string pointed to by *format* and outputs the formatted data to a stream. There must be the same number of format specifiers as arguments.

Note: For details on format specifiers, see printf Format Specifiers.

Return Value

fprintf returns the number of bytes output. In the event of error it returns EOF.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

fputc, fputwc

See also Example Portability

Syntax

```
#include <stdio.h>
int fputc(int c, FILE *stream);
wint_t fputwc(wint_t c, FILE *stream);
```

Description

Puts a character on a stream.

fputc outputs character c to the named stream.

Note: For Win32s or Win32 GUI applications, stdin must be redirected.

Return Value

On success, *fputc* returns the character *c*. On error, it returns EOF.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

_fputchar, _fputwchar

See also Example Portability

Syntax

```
#include <stdio.h>
int _fputchar(int c);
wint_t _fputwchar(wint_t c);
```

Description

Outputs a character to stdout.

_fputchar outputs character *c* to stdout. *_fputchar(c)* is the same as *fputc(cstdout)*.

For Win32s or Win32 GUI applications, stdout must be redirected.

Return Value

On success *_fputchar* returns the character *c*.

On error it returns EOF.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+		+	+		+

fputs, fputws

See also Example Portability

Syntax

```
#include <stdio.h>
int fputs(const char *s, FILE *stream);
int fputws(const wchar_t *s, FILE *stream);
```

Description

Outputs a string on a stream.

fputs copies the null-terminated string *s* to the given output stream; it does not append a newline character and the terminating null character is not copied.

Return Value

On success *fputs* returns a non-negative value.

On error it returns a value of EOF.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

fread

See also   Example   Portability

Syntax

```
#include <stdio.h>
size_t fread(void *ptr, size_t size, size_t n, FILE *stream);
```

Description

Reads data from a stream.

fread reads *n* items of data each of length *size* bytes from the given input stream into a block pointed to by *ptr*.

The total number of bytes read is *(n * size)*.

Return Value

On success *fread* returns the number of items (not bytes) actually read.

On end-of-file or error it returns a short count (possibly 0).

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

free

See also   Example   Portability

Syntax

```
#include <stdlib.h>
void free(void *block);
```

Description

Frees allocated block.

free deallocates a memory block allocated by a previous call to *calloc, malloc,* or *realloc*.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

freopen, _wfreopen

See also Example Portability

Syntax

```
#include <stdio.h>
FILE *freopen(const char *filename, const char *mode, FILE *stream);
FILE *_wfreopen(const wchar_t *filename, const wchar_t *mode, FILE *stream);
```

Description

Associates a new file with an open stream.

freopen substitutes the named file in place of the open stream. It closes *stream* regardless of whether the open succeeds. *freopen* is useful for changing the file attached to stdin, stdout, or stderr.

The *mode* string used in calls to *fopen* is one of the following values:

Value	Description
r	Open for reading only.
w	Create for writing. .
a	Append; open for writing at end-of-file or create for writing if the file does not exist.
r+	Open an existing file for update (reading and writing).
w+	Create a new file for update (reading and writing).
a+	Open for append; open (or create if the file does not exist) for update at the end of the file.

To specify that a given file is being opened or created in text mode append a *t* to the *mode* string (*rt w+t* and so on); similarly to specify binary mode append a *b* to the *mode* string (*wb a+b* and so on).

If a *t* or *b* is not given in the *mode* string the mode is governed by the global variable *_fmode*. If *_fmode* is set to O_BINARY files are opened in binary mode. If *_fmode* is set to O_TEXT they are opened in text mode. These O_... constants are defined in fcntl.h.

When a file is opened for update, both input and output can be done on the resulting stream; however,

- output cannot be directly followed by input without an intervening fseekor rewind
- input cannot be directly followed by output without an intervening *fseek, rewind,* or an input that encounters end-offile

Return Value

On successful completion *freopen* returns the argument *stream*.

On error it returns NULL.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

frexp, frexpl

See also   Example   Portability

Syntax

```
#include <math.h>
double frexp(double x, int *exponent);
long double frexpl(long double x, int *exponent);
```

Description

Splits a number into mantissa and exponent.

frexp calculates the mantissa *m* (a **double** greater than or equal to 0.5 and less than 1) and the integer value *n* such that *x* (the original **double** value) equals *m * 2n. frexp* stores *n* in the integer that *exponent* points to.

frexpl is the **long double** version; it takes a **long double** argument for *x* and returns a **long double** result.

Return Value

frexp and *frexpl* return the mantissa *m*. Error handling for these routines can be modified through the functions _matherr and _matherrl.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
frexp	+	+	+	+	+	+	+
frexpl	+		+	+			+

fscanf, fwscanf

See also Example Portability

Syntax

```
#include <stdio.h>
int fscanf(FILE *stream, const char *format[, address, ...]);
int fwscanf(FILE *stream, const wchar_t *format[, address, ...]);
```

Description

Scans and formats input from a stream.

fscanf scans a series of input fields one character at a time reading from a stream. Then each field is formatted according to a format specifier passed to *fscanf* in the format string pointed to by format. Finally *fscanf* stores the formatted input at an address passed to it as an argument following *format*. The number of format specifiers and addresses must be the same as the number of input fields.

Note: For details on format specifiers, see scanf Format Specifiers.

fscanf can stop scanning a particular field before it reaches the normal end-of-field character (whitespace) or it can terminate entirely for a number of reasons. See *scanf* for a discussion of possible causes.

Return Value

fscanf returns the number of input fields successfully scanned, converted and stored. The return value does not include scanned fields that were not stored.

If *fscanf* attempts to read at end-of-file, the return value is EOF. If no fields were stored, the return value is 0.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

fseek

See also Example Portability

Syntax

```
#include <stdio.h>
int fseek(FILE *stream, long offset, int whence);
```

Description

Repositions a file pointer on a stream.

fseek sets the file pointer associated with *stream* to a new position that is *offset* bytes from the file location given by *whence*. For text mode streams offset should be 0 or a value returned by *ftell*.

whence must be one of the values 0. 1, or 2 which represent three symbolic constants (defined in stdio.h) as follows:

Constant	whence	File location
SEEK_SET	0	File beginning
SEEK_CUR	1	Current file pointer position
SEEK_END	2	End-of-file

fseek discards any character pushed back using *ungetc*. *fseek* is used with stream I/O; for file handle I/O use *lseek*.

After *fseek* the next operation on an update file can be either input or output.

Return Value

fseek returns 0 if the pointer is successfully moved and nonzero on failure.

fseek might return a 0 indicating that the pointer has been moved successfully when in fact it has not been. This is because DOS, which actually resets the pointer, does not verify the setting. *fseek* returns an error code only on an unopened file or device.

In the event of an error return the global variable *errno* is set to one of the following values:

EBADF	Bad file pointer
EINVAL	Invalid argument
ESPIPE	Illegal seek on device

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

fsetpos

See also Example Portability

Syntax

```
#include <stdio.h>
int fsetpos(FILE *stream, const fpos_t *pos);
```

Description

Positions the file pointer of a stream.

fsetpos sets the file pointer associated with *stream* to a new position. The new position is the value obtained by a previous call to *fgetpos* on that stream. It also clears the end-of-file indicator on the file that *stream* points to and undoes any effects of *ungetc* on that file. After a call to *fsetpos* the next operation on the file can be input or output.

Return Value

On success *fsetpos* returns 0.

On failure it returns a nonzero value and also sets the global variable errno to a nonzero value.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+	+	+	+

_fsopen, _wfsopen

See also Example Portability

Syntax

```
#include <stdio.h>
#include <share.h>
FILE *_fsopen(const char *filename, const char *mode, int shflag);
FILE *_wfsopen(const wchar_t *filename, const wchar_t *mode, int shflag);
```

Description

Opens a stream with file sharing.

_*fsopen* opens the file named by *filename* and associates a stream with it. _*fsopen* returns a pointer that is used to identify the stream in subsequent operations.

The *mode* string used in calls to _*fsopen* is one of the following values:

Mode	Description
r	Open for reading only.
w	Create for writing. If a file by that name already exists, it will be overwritten.
a	Append; open for writing at end of file. or create for writing if the file does not exist.
r+	Open an existing file for update (reading and writing).
w+	Create a new file for update (reading and writing). If a file by that name already exists, it will be overwritten.
a+	Open for append; open (or create if the file does not exist) for update at the end of the file.

To specify that a given file is being opened or created in text mode append a *t* to the *mode* string (*rt w+t* and so on). Similarly to specify binary mode append a *b* to the *mode* string (*wb a+b* and so on). _*fsopen* also allows the *t* or *b* to be inserted between the letter and the + character in the mode string; for example *rt+* is equivalent to *r+t*. If a *t* or *b* is not given in the *mode* string the mode is governed by the global variable _*fmode*. If _*fmode* is set to O_BINARY files are opened in binary mode. If _*fmode* is set to O_TEXT they are opened in text mode. These O_... constants are defined in fcntl.h.

When a file is opened for update, both input and output can be done on the resulting stream, however:

- output cannot be directly followed by input without an intervening fseekor rewind
- input cannot be directly followed by output without an intervening *fseek*, *rewind*, or an input that encounters end-offile

shflag specifies the type of file-sharing allowed on the file *filename*. Symbolic constants for sh*flag* are defined in share.h.

Note: For DOS users, the file-sharing flags are ignored if SHARE is not loaded.

Value of shflag	Description
SH_COMPAT	Sets compatibility mode
SH_DENYRW	Denies read/write access
SH_DENYWR	Denies write access
SH_DENYRD	Denies read access
SH_DENYNONE	Permits read/write access
SH_DENYNO	Permits read/write access

Return Value

On successful completion _*fsopen* returns a pointer to the newly opened stream.

On error it returns NULL.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

fstat, stat, _wstat

See also   Examples   Portability

Syntax

```
#include <sys\stat.h>
int fstat(int handle, struct stat *statbuf);
int stat(const char *path, struct stat *statbuf);
int _wstat(const wchar_t *path, struct stat *statbuf);
```

Description

Gets open file information.

fstat stores information in the *stat* structure about the file or directory associated with *handle*.

stat stores information about a given file or directory in the *stat* structure. The name of the file is *path*.

statbuf points to the *stat* structure (defined in sys\stat.h). That structure contains the following fields:

st_mode	Bit mask giving information about the file's mode
st_dev	Drive number of disk containing the file or file handle if the file is on a device
st_rdev	Same as *st_dev*
st_nlink	Set to the integer constant 1
st_size	Size of the file in bytes
st_atime	Most recent access (Windows) or last time modified (DOS)
st_mtime	Same as *st_atime*
st_ctime	Same as *st_atime*

The *stat* structure contains three more fields not mentioned here. They contain values that are meaningful only in UNIX.

The *st_mode* bit mask that gives information about the mode of the open file includes the following bits:

One of the following bits will be set:

S_IFCHR	If *handle* refers to a device.
S_IFREG	If an ordinary file is referred to by *handle*.

One or both of the following bits will be set:

S_IWRITE	If user has permission to write to file.
S_IREAD	If user has permission to read to file.

The HPFS and NTFS file-management systems make the following distinctions:

st_atime	Most recent access
st_mtime	Most recent modify
st_ctime	Creation time

Return Value

fstat and *stat* return 0 if they successfully retrieved the information about the open file.

On error (failure to get the information) these functions return -1 and set the global variable *errno* to

EBADF	Bad file handle

Examples

fstat

stat

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

ftell

See also Example Portability

Syntax

```
#include <stdio.h>
long int ftell(FILE *stream);
```

Description

Returns the current file pointer.

ftell returns the current file pointer for *stream*. The offset is measured in bytes from the beginning of the file (if the file is binary). The value returned by *ftell* can be used in a subsequent call to *fseek*.

Return Value

ftell returns the current file pointer position on success. It returns -1L on error and sets the global variable *errno* to a positive value.

In the event of an error return the global variable *errno* is set to one of the following values:

EBADF	Bad file pointer
ESPIPE	Illegal seek on device

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

ftime

See also Example Portability

Syntax

```
#include <sys\timeb.h>
void ftime(struct timeb *buf)
```

Description

Stores current time in timeb structure.

On UNIX platforms *ftime* is available only on System V systems.

ftime determines the current time and fills in the fields in the *timeb* structure pointed to by *buf*. The *timeb* structure contains four fields: *time, millitm, _timezone,* and *dstflag*:

```
struct timeb {
   long time ;
   short millitm ;
   short _timezone ;
   short dstflag ;
};
```

time	provides the time in seconds since 00:00:00 Greenwich mean time (GMT) January 1 1970.
millitm	is the fractional part of a second in milliseconds.
_timezone	is the difference in minutes between GMT and the local time. This value is computed going west from GMT. *ftime* gets this field from the global variable *_timezone* which is set by *tzset*.
dstflag	is set to nonzero if daylight saving time is taken into account during time calculations.

Note: *ftime* calls *tzset*. Therefore it isn't necessary to call *tzset* explicitly when you use *ftime*.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

_fullpath, _wfullpath

See also Example Portability

Syntax

```
#include <stdlib.h>
char * _fullpath(char *buffer, const char *path, int buflen);
wchar_t * _wfullpath(wchar_t *buffer, const wchar_t *path, int buflen);
```

Description

Converts a path name from relative to absolute.

_fullpath converts the relative path name in *path* to an absolute path name that is stored in the array of characters pointed to by *buffer*. The maximum number of characters that can be stored at *buffer* is *buflen*. The function returns NULL if the buffer isn't big enough to store the absolute path name or if the path contains an invalid drive letter.

If *buffer* is NULL, *_fullpath* allocates a buffer of up to _MAX_PATH characters. This buffer should be freed using *free* when it is no longer needed. _MAX_PATH is defined in stdlib.h.

Return Value

If successful the *_fullpath* function returns a pointer to the buffer containing the absolute path name.

On error, this function returns NULL.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

fwrite

See also Example Portability

Syntax

```
#include <stdio.h>
size_t fwrite(const void *ptr, size_t size, size_t n, FILE *stream);
```

Description

Writes to a stream.

fwrite appends *n* items of data each of length *size* bytes to the given output file. The data written begins at *ptr*. The total number of bytes written is *(n x size)*. *ptr* in the declarations is a pointer to any object.

Return Value

On successful completion *fwrite* returns the number of items (not bytes) actually written.

On error it returns a short count.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

gcvt

See also Example Portability

Syntax

```
#include <stdlib.h>
char *gcvt(double value, int ndec, char *buf);
```

Description

Converts floating-point number to a string.

gcvt converts *value* to a null-terminated ASCII string and stores the string in *buf*. It produces *ndec* significant digits in FORTRAN F format, if possible; otherwise, it returns the value in the *printf E* format (ready for printing). It might suppress trailing zeros.

Return Value

gcvt returns the address of the string pointed to by *buf*.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

geninterrupt

See also Example Portability

Syntax

```
#include <dos.h>
void geninterrupt(int intr_num);
```

Description

Generates a software interrupt.

The *geninterrupt* macro triggers a software trap for the interrupt given by *intr_num.* The state of all registers after the call depends on the interrupt called.

Interrupts can leave registers in unpredictable states.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			

getc, getwc

See also   Example   Portability

Syntax

```
#include <stdio.h>
int getc(FILE *stream);
wint_t getwc(FILE *stream);
```

Description

Gets character from stream.

getc returns the next character on the given input stream and increments the stream's file pointer to point to the next character.

Note: For Win32s or Win32 GUI applications, stdin must be redirected.

Return Value

On success, *getc* returns the character read, after converting it to an **int** without sign extension.

On end-of-file or error, it returns EOF.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

getch

See also Example Portability

Syntax

```
#include <conio.h>
int getch(void);
```

Description

Gets character from keyboard, does not echo to screen.

getch reads a single character directly from the keyboard, without echoing to the screen.

Note: Do not use this function for Win32s or Win32 GUI applications.

Return Value

getch returns the character read from the keyboard.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+			+			+

getchar, getwchar

See also Example Portability

Syntax

```
#include <stdio.h>
int getchar(void);
wint_t getwchar(void);
```

Description

Gets character from stdin.

getchar is a macro that returns the next character on the named input stream stdin. It is defined to be *getc(stdin)*.

Note: Do not use this function for Win32s or Win32 GUI applications.

Return Value

On success, *getchar* returns the character read, after converting it to an **int** without sign extension.

On end-of-file or error, it returns EOF.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+		+	+	+	+

getche

See also Example Portability

Syntax

```
#include <conio.h>
int getche(void);
```

Description

Gets character from the keyboard, echoes to screen.

getche reads a single character from the keyboard and echoes it to the current text window using direct video or BIOS.

Note: Do not use this function for Win32s or Win32 GUI applications.

Return Value

getche returns the character read from the keyboard.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+			+			+

getcurdir, _wgetcurdir

See also Example Portability

Syntax

```
#include <dir.h>
int getcurdir(int drive, char *directory);
int _wgetcurdir(int drive, wchar_t *directory );
```

Description

Gets current directory for specified drive.

getcurdir gets the name of the current working directory for the drive indicated by *drive*. *drive* specifies a drive number (0 for default, 1 for A, and so on). *directory* points to an area of memory of length MAXDIR where the null-terminated directory name will be placed. The name does not contain the drive specification and does not begin with a backslash.

Return Value

getcurdir returns 0 on success or -1 in the event of error.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

getcwd, _wgetcwd

See also Example Portability

Syntax

```
#include <dir.h>
char *getcwd(char *buf, int buflen);
wchar_t *_wgetcwd(wchar_t *buf, int buflen);
```

Description

Gets current working directory.

getcwd gets the full path name (including the drive) of the current working directory, up to *buflen* bytes long and stores it in *buf*. If the full path name length (including the null terminator) is longer than *buflen* bytes, an error occurs.

If *buf* is NULL, a buffer *buflen* bytes long is allocated for you with *malloc*. You can later free the allocated buffer by passing the return value of *getcwd* to the function *free*.

Return Value

getcwd returns the following values:

- If *buf* is not NULL on input, *getcwd* returns *buf* on success, NULL on error.
- If *buf* is NULL on input, *getcwd* returns a pointer to the allocated buffer.

In the event of an error return, the global variable *errno* is set to one of the following values:

ENODEV	No such device
ENOMEM	Not enough memory to allocate a buffer (*buf* is NULL)
ERANGE	Directory name longer than *buflen* (*buf* is not NULL)

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

getdate, setdate

See Also   Examples   Portability

Syntax

```
#include <dos.h>
void getdate(struct date *datep);
void setdate(struct date *datep);
```

Description

Gets and sets system date.

getdate fills in the *date* structure (pointed to by *datep*) with the system's current date.

setdate sets the system date (month, day, and year) to that in the *date* structure pointed to by *datep*. Note that a request to set a date might fail if you do not have the privileges required by the operating system.

The *date* structure is defined as follows:

```
struct date{
   int da_year;     /* current year */
   char da_day;     /* day of the month */
   char da_mon;     /* month (1 = Jan) */
};
```

Return Value

getdate and *setdate* do not return a value.

See Also

ctime

gettime

settime

Examples

getdate

setdate

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

_getdcwd, _wgetdcwd

See also Example Portability

Syntax

```
#include <direct.h>
char * _getdcwd(int drive, char *buffer, int buflen);
wchar_t * _wgetdcwd(int drive, wchar_t *buffer, int buflen);
```

Description

Gets current directory for specified drive.

_getdcwd gets the full path name of the working directory of the specified drive (including the drive name), up to *buflen* bytes long, and stores it in *buffer*. If the full path name length (including the null-terminator) is longer than *buflen*, an error occurs. The drive is 0 for the default drive, 1=A, 2=B, and so on.

If the working directory is the root directory, the terminating character for the full path is a backslash. If the working directory is a subdirectory, there is no terminating backslash after the subdirectory name.

If *buffer* is NULL, *_getdcwd* allocates a buffer at least *buflen* bytes long. You can later free the allocated buffer by passing the *_getdcwd* return value to the *free* function.

Return Value

If successful, *_getdcwd* returns a pointer to the buffer containing the current directory for the specified drive.

Otherwise it returns NULL, and sets the global variable errno to one of the following values:

ENOMEM	Not enough memory to allocate a buffer (*buffer* is NULL)
ERANGE	Directory name longer than *buflen* (*buffer* is not NULL)

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

getdfree

Example Portability

Syntax

```
#include <dos.h>
void getdfree(unsigned char drive, struct dfree *dtable);
```

Description

Gets disk free space.

getdfree accepts a drive specifier in *drive* (0 for default, 1 for A, and so on) and fills the *dfree* structure pointed to by *dtable* with disk attributes.

The *dfree* structure is defined as follows:

```
struct dfree {
   unsigned df_avail;     /* available clusters */
   unsigned df_total;     /* total clusters */
   unsigned df_bsec;      /* bytes per sector */
   unsigned df_sclus;     /* sectors per cluster */
};
```

Return Value

getdfree returns no value. In the event of an error, *df_sclus* in the *dfree* structure is set to (**unsigned**) -1.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

getdisk, setdisk

See also Examples Portability

Syntax

```
#include <dir.h>
int getdisk(void);
int setdisk(int drive);
```

Description

Gets or sets the current drive number.

getdisk gets the current drive number. It returns an integer: 0 for A, 1 for B, 2 for C, and so on.

setdisk sets the current drive to the one associated with drive: 0 for A, 1 for B, 2 for C, and so on.

The *setdisk* function changes the current drive of the parent process.

Return Value

getdisk returns the current drive number. *setdisk* returns the total number of drives available.

Examples

getdisk

setdisk

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

getenv, _wgetenv

See also   Example   Portability

Syntax

```
#include <stdlib.h>
char *getenv(const char *name);
wchar_t *_wgetenv(const wchar_t *name);
```

Description

Find or delete an environment variable from the system environment.

The environment consists of a series of entries that are of the form `name=string\0`.

getenv returns the value of a specified variable. On DOS and OS/2, *name* must be uppercase. On other systems, *name* can be either uppercase or lowercase. *name* must not include the equal sign (**=**). If the specified environment variable does not exist, *getenv* returns a NULL pointer.

To delete the variable from the environment, use `getenv("name=")`.

Note: Environment entries must not be changed directly. If you want to change an environment value, you must use *putenv*.

Return Value

On success, *getenv* returns the value associated with *name*.

If the specified *name* is not defined in the environment, *getenv* returns a NULL pointer.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

getftime, setftime

See also Examples Portability

Syntax

```
#include <io.h>
int getftime(int handle, struct ftime *ftimep);
int setftime(int handle, struct ftime *ftimep);
```

Description

Gets and sets the file date and time.

getftime retrieves the file time and date for the disk file associated with the open *handle*. The *ftime* structure pointed to by *ftimep* is filled in with the file's time and date.

setftime sets the file date and time of the disk file associated with the open *handle* to the date and time in the *ftime* structure pointed to by *ftimep*. The file must not be written to after the *setftime* call or the changed information will be lost. The file must be open for writing; an EACCES error will occur if the file is open for read-only access.

setftime requires the file to be open for writing; an EACCES error will occur if the file is open for read-only access.

The *ftime* structure is defined as follows:

```
struct ftime {
   unsigned ft_tsec: 5;      /* two seconds */
   unsigned ft_min: 6;       /* minutes */
   unsigned ft_hour: 5;      /* hours */
   unsigned ft_day: 5;       /* days */
   unsigned ft_month: 4;     /* months */
   unsigned ft_year: 7;      /* year - 1980*/
   };
```

Return Value

getftime and *setftime* return 0 on success.

In the event of an error return -1 is returned and the global variable *errno* is set to one of the following values:

EACCES	Permission denied
EBADF	Bad file number
EINVFNC	Invalid function number

Examples

getftime

setftime

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

_get_osfhandle

See also Example Portability

Syntax

```
#include <io.h>
long _get_osfhandle(int filehandle);
```

Description

Associates file handles.

The *_get_osfhandle* function associates an operating system file handle with an existing run-time file handle. The variable *filehandle* is the file handle of your program.

Return value

On success, *_get_osfhandle* returns an operating system file handle corresponding to the variable *filehandle*.

On error, the function returns -1 and sets the global variable errno to

EBADF	an invalid file handle

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
			+			

getpass

See also Example Portability

Syntax

```
#include <conio.h>
char *getpass(const char *prompt);
```

Description

Reads a password.

getpass reads a password from the system console after prompting with the null-terminated string *prompt* and disabling the echo. A pointer is returned to a null-terminated string of up to eight characters (not counting the null-terminator).

Note: Do not use this function for Win32s or Win32 GUI applications.

Return Value

The return value is a pointer to a static string which is overwritten with each call.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+		+			+

getpid

Example   Portability

Syntax

```
#include <process.h>
unsigned getpid(void)
```

Description

Gets the process ID of a program.

This function returns the current process ID--an integer that uniquely identifies the process.

Return Value

getpid returns the current process' ID.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

gets, _getws

See also Example Portability

Syntax

```
#include <stdio.h>
char *gets(char *s);
wchar_t *_getws(wchar_t *s);  // Unicode version
```

Description

Gets a string from stdin.

gets collects a string of characters terminated by a new line from the standard input stream stdin and puts it into *s*. The new line is replaced by a null character (\0) in *s*.

gets allows input strings to contain certain whitespace characters (spaces, tabs). *gets* returns when it encounters a new line; everything up to the new line is copied into *s*.

The *gets* function is not length-terminated. If the input string is sufficiently large, data can be overwritten and corrupted. The *fgets* function provides better control of input strings.

Note: For Win32s or Win32 GUI applications, stdin must be redirected.

Return Value

On success, *gets* returns the string argument *s*.

On end-of-file or error, it returns NULL

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+		+	+	+	+

gettext

See also Example Portability

Syntax

```
#include <conio.h>
int gettext(int left, int top, int right, int bottom, void *destin);
```

Description

Copies text from text mode screen to memory.

gettext stores the contents of an onscreen text rectangle defined by *left*, *top*, *right*, and *bottom* into the area of memory pointed to by *destin*.

All coordinates are absolute screen coordinates not window-relative. The upper left corner is (1,1). *gettext* reads the contents of the rectangle into memory sequentially from left to right and top to bottom.

Each position onscreen takes 2 bytes of memory: The first byte is the character in the cell and the second is the cell's video attribute. The space required for a rectangle *w* columns wide by *h* rows high is defined as

bytes = *(h* rows) x *(w* columns) x 2

Note: Do not use this function for Win32s or Win32 GUI applications.

Return Value

gettext returns 1 if the operation succeeds.

On error, it returns 0 (for example, if it fails because you gave coordinates outside the range of the current screen mode).

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+			+			+

gettextinfo

See also Example Portability

Syntax

```
#include <conio.h>
void gettextinfo(struct text_info *r);
```

Description

Gets text mode video information.

gettextinfo fills in the *text_info* structure pointed to by *r* with the current text video information.

The *text_info* structure is defined in conio.h as follows:

```
struct text_info {
   unsigned char winleft;         /* left window coordinate */
   unsigned char wintop;          /* top window coordinate */
   unsigned char winright;        /* right window coordinate */
   unsigned char winbottom;       /* bottom window coordinate */
   unsigned char attribute;       /* text attribute */
   unsigned char normattr;        /* normal attribute */
   unsigned char currmode;        /* BW40, BW80, C40, C80, or C4350 */
   unsigned char screenheight;    /* text screen's height */
   unsigned char screenwidth;     /* text screen's width */
   unsigned char curx;            /* x-coordinate in current window */
   unsigned char cury;            /* y-coordinate in current window */
};
```

Note: Do not use this function for Win32s or Win32 GUI applications.

Return Value

None. Results are returned in the structure pointed to by *r*.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+			+			+

gettime, settime

See also   Examples   Portability

Syntax

```
#include <dos.h>
void gettime(struct time *timep);
void settime(struct time *timep);
```

Description

Gets and sets the system time.

gettime fills in the *time* structure pointed to by *timep* with the system's current time.

settime sets the system time to the values in the *time* structure pointed to by *timep*.

The *time* structure is defined as follows:

```
struct time {
   unsigned char ti_min;       /* minutes */
   unsigned char ti_hour;      /* hours */
   unsigned char ti_hund;      /* hundredths of seconds */
   unsigned char ti_sec;       /* seconds */
};
```

Return Value

None.

Examples

gettime

settime

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
gettime	+		+	+			+
settime	+		+				+

_getw

See also Example Portability

Syntax

```
#include <stdio.h>
int _getw(FILE *stream);
```

Description

Gets an integer from stream.

_getw returns the next integer in the named input stream. It assumes no special alignment in the file.

_getw should not be used when the stream is opened in text mode.

Return Value

_getw returns the next integer on the input stream.

On end-of-file or error, *_getw* returns EOF.

Note: Because EOF is a legitimate value for *_getw* to return, feof or ferror should be used to detect end-of-file or error.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

gmtime

See also Example Portability

Syntax

```
#include <time.h>
struct tm *gmtime(const time_t *timer);
```

Description

Converts date and time to Greenwich mean time (GMT).

gmtime accepts the address of a value returned by *time* and returns a pointer to the structure of type *tm* containing the time elements. *gmtime* converts directly to GMT.

The global long variable _*timezone* should be set to the difference in seconds between GMT and local standard time (in PST _*timezone* is 8 x 60 x 60). The global variable _daylight should be set to nonzero *only* if the standard U.S. daylight saving time conversion should be applied.

This is the *tm* structure declaration from the time.h header file:

```
struct tm {
   int tm_sec;              /* Seconds */
   int tm_min;              /* Minutes */
   int tm_hour;             /* Hour (0 - 23) */
   int tm_mday;             /* Day of month (1 - 31) */
   int tm_mon;              /* Month (0 - 11) */
   int tm_year;             /* Year (calendar year minus 1900) */
   int tm_wday;             /* Weekday (0 - 6; Sunday is 0) */
   int tm_yday;             /* Day of year (0 -365) */
   int tm_isdst;            /* Nonzero if daylight saving time is in effect.
  */
};
```

These quantities give the time on a 24-hour clock, day of month (1 to 31), month (0 to 11), weekday (Sunday equals 0), year - 1900, day of year (0 to 365), and a flag that is nonzero if daylight saving time is in effect.

Return Value

gmtime returns a pointer to the structure containing the time elements. This structure is a static that is overwritten with each call.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

gotoxy

See also Example Portability

Syntax

```
#include <conio.h>
void gotoxy(int x, int y);
```

Description

Positions cursor in text window.

gotoxy moves the cursor to the given position in the current text window. If the coordinates are in any way invalid the call to *gotoxy* is ignored. An example of this is a call to *gotoxy*(40,30) when (35,25) is the bottom right position in the window. Neither argument to *gotoxy* can be zero.

Note: Do not use this function for Win32s or Win32 GUI applications.

Return Value

None.

Examples

gotoxygotoxy_Ex

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

heapcheck

Example Portability

Syntax

```
#include <alloc.h>
int heapcheck(void);
```

Description

Checks and verifies the heap.

heapcheck walks through the heap and examines each block, checking its pointers, size, and other critical attributes.

Return Value

The return value is less than 0 for an error and greater than 0 for success. The return values and their meaning are as follows:

_HEAPCORRUPT	Heap has been corrupted
_HEAPEMPTY	No heap
_HEAPOK	Heap is verified

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+			+			+

heapcheckfree

Example Portability

Syntax

```
#include <alloc.h>
int heapcheckfree(unsigned int fillvalue);
```

Description

Checks the free blocks on the heap for a constant value.

Return Value

The return value is less then 0 for an error and greater than 0 for success. The return values and their meaning are as follows:

_BADVALUE	A value other than the fill value was found
_HEAPCORRUPT	Heap has been corrupted
_HEAPEMPTY	No heap
_HEAPOK	Heap is accurate

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+			+			+

heapchecknode

Example Portability

Syntax

```
#include <alloc.h>
int heapchecknode(void *node);
```

Description

Checks and verifies a single node on the heap.

If a node has been freed and *heapchecknode* is called with a pointer to the freed block, *heapchecknode* can return _BADNODE rather than the expected _FREEENTRY. This is because adjacent free blocks on the heap are merged, and the block in question no longer exists.

Return Value

One of the following values:

_BADNODE	Node could not be found
_FREEENTRY	Node is a free block
_HEAPCORRUPT	Heap has been corrupted
_HEAPEMPTY	No heap
_USEDENTRY	Node is a used block

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+			+			+

_heapchk

See also   Example   Portability

Syntax

```
#include <malloc.h>
int _heapchk(void);
```

Description

Checks and verifies the heap.

_heapchk walks through the heap and examines each block, checking its pointers, size, and other critical attributes.

Return Value

One of the following values:

_HEAPBADNODE	A corrupted heap block has been found
_HEAPEMPTY	No heap exists
_HEAPOK	The heap appears to be uncorrupted

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
			+			+

heapfillfree

Example Portability

Syntax

```
#include <alloc.h>
int heapfillfree(unsigned int fillvalue);
```

Description

Fills the free blocks on the heap with a constant value.

Return Value

One of the following values:

_HEAPCORRUPT	Heap has been corrupted
_HEAPEMPTY	No heap
_HEAPOK	Heap is accurate

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+			+			+

_heapmin

See also Portability

Syntax

```
#include <malloc.h>
int _heapmin(void);
```

Description

Release unused heap areas.

The *_heapmin* function returns unused areas of the heap to the operating system. This allows blocks that have been allocated and then freed to be used by other processes. Due to fragmentation of the heap, *_heapmin* might not always be able to return unused memory to the operating system; this is not an error.

Return Value

_heapmin returns 0 if it is successful, or -1 if an error occurs.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

_heapset

See also Example Portability

Syntax

```
#include <malloc.h>
int _heapset(unsigned int fillvalue);
```

Description

Fills the free blocks on the heap with a constant value.

_heapset checks the heap for consistency using the same methods as _heapchk. It then fills each free block in the heap with the value contained in the least significant byte of *fillvalue*. This function can be used to find heap-related problems. It does *not* guarantee that subsequently allocated blocks will be filled with the specified value.

Return Value

One of the following values:

_HEAPOK	The heap appears to be uncorrupted
_HEAPEMPTY	No heap exists
_HEAPBADNODE	A corrupted heap block has been found

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
			+			+

heapwalk

Example Portability

Syntax

```
#include <alloc.h>
int heapwalk(struct heapinfo *hi);
```

Description

heapwalk is used to "walk" through the heap, node by node.

heapwalk assumes the heap is correct. Use heapcheck to verify the heap before using *heapwalk*. _HEAPOK is returned with the last block on the heap. _HEAPEND will be returned on the next call to *heapwalk*.

heapwalk receives a pointer to a structure of type *heapinfo* (declared in alloc.h). For the first call to *heapwalk*, set the hi.ptr field to null. *heapwalk* returns with hi.ptr containing the address of the first block. hi.size holds the size of the block in bytes. hi.in_use is a flag that's set if the block is currently in use.

Return Value

One of the following values:

_HEAPEMPTY	No heap exists
_HEAPEND	The end of the heap has been reached
_HEAPOK	The *heapinfo* block contains valid information about the next heap block

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+			+			+

highvideo

See also   Example   Portability

Syntax

```
#include <conio.h>
void highvideo(void);
```

Description

Selects high-intensity characters.

highvideo selects high-intensity characters by setting the high-intensity bit of the currently selected foreground color.

This function does not affect any characters currently onscreen, but does affect those displayed by functions (such as *cprintf*) that perform direct video, text mode output after *highvideo* is called.

Note: Do not use this function for Win32s or Win32 GUI applications.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+			+			+

hypot, hypotl

Example Portability

Syntax

```
#include <math.h>
double hypot(double x, double y);
long double hypotl(long double x, long double y);
```

Description

Calculates hypotenuse of a right triangle.

hypot calculates the value *z* where

$z2 = x2 + y2$ and $z >= 0$

This is equivalent to the length of the hypotenuse of a right triangle, if the lengths of the two sides are *x* and *y*.

hypotl is the **long double** version; it takes **long double** arguments and returns a **long double** result.

Return Value

On success, these functions return *z*, a **double** (*hypot*) or a **long double**) (*hypotl*). On error (such as an overflow), they set the global variable errno to

ERANGE	Result out of range

and return the value HUGE_VAL (*hypot*) or _LHUGE_VAL) (*hypotl*). Error handling for these routines can be modified through the functions _matherr and _matherrl.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
hypot	+	+	+	+			+
hypotl	+		+	+			+

inpw

Example Portability

Syntax

```
#include <conio.h>
unsigned inpw(unsigned portid);
```

Description

Reads a word from a hardware port.

inpw is a macro that reads a 16-bit word from the inport port specified by *portid*. It reads the low byte of the word from *portid*, and the high byte from *portid* + 1.

If *inpw* is called when conio.h has been included, it will be treated as a macro that expands to inline code. If you don't include conio.h, or if you do include conio.h and **#undef** the macro *inpw*, you get the *inpw* function.

Return Value

inpw returns the value read.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+				

insline

See also   Example   Portability

Syntax

```
#include <conio.h>
void insline(void);
```

Description

Inserts a blank line in the text window.

insline inserts an empty line in the text window at the cursor position using the current text background color. All lines below the empty one move down one line, and the bottom line scrolls off the bottom of the window.

Note: Do not use this function for Win32s or Win32 GUI applications.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+			+			+

isalnum, iswalnum, _ismbcalnum

Example Portability

Syntax

```
#include <ctype.h>
int isalnum(int c);
int iswalnum(wint_t c);

#include <mbstring.h>
int _ismbcalnum(unsigned int c);
```

Description

Tests for an alphanumeric character.

isalnum is a macro that classifies ASCII-coded integer values by table lookup. The macro is affected by the current locale's LC_CTYPE category. For the default C locale, *c* is a letter (*A* to *Z* or *a* to *z*) or a digit (0 to 9).

You can make this macro available as a function by undefining (**#undef**) it.

Return Value

It is a predicate returning nonzero for true and 0 for false. *isalnum* returns nonzero if *c* is a letter or a digit.

iswalnum returns nonzero if *iswalpha* or *iswdigit* return true for *c*.

_ismbcalnum returns true if and only if the argument *c* is a single-byte ASCII English letter.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

isalpha, iswalpha, _ismbcalpha

Example Portability

Syntax

```
#include <ctype.h>
int isalpha(int c);
int iswalpha(wint_t c);

#include <mbstring.h>
int _ismbcalpha(unsigned int c);
```

Description

Classifies an alphabetical character.

isalpha is a macro that classifies ASCII-coded integer values by table lookup. The macro is affected by the current locale's LC_CTYPE category. For the default C locale, *c* is a letter (*A* to *Z* or *a* to *z*).

You can make this macro available as a function by undefining (**#undef**) it.

Return Value

isalpha returns nonzero if *c* is a letter.

iswalpha returns nonzero if *c* is a **wchar_t** in the character set defined by the implementation.

_ismbcalpha returns true if and only if the argument *c* is a single-byte ASCII English letter.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

isascii, iswascii

Example Portability

Syntax

```
#include <ctype.h>
int isascii(int c);
int iswascii(wint_t c);
```

Description

Character classification macro.

These functions depend on the LC_CTYPE

isascii is a macro that classifies ASCII-coded integer values by table lookup. It is a predicate returning nonzero for true and 0 for false.

isascii is defined on all integer values.

Return Value

isascii returns nonzero if *c* is in the range 0 to 127 (0x00-0x7F).

iswascii returns nonzero if *c* is

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

isatty

Example Portability

Syntax

```
#include <io.h>
int isatty(int handle);
```

Description

Checks for device type.

isatty determines whether *handle* is associated with any one of the following character devices:

- a terminal
- a console
- a printer
- a serial port

Return Value

If the device is one of the four character devices listed above, *isatty* returns a nonzero integer. If it is not such a device, *isatty* returns 0.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

iscntrl, iswcntrl

Example Portability

Syntax

```
#include <ctype.h>
int iscntrl(int c);
int iswcntrl(wint_t c);
```

Description

Tests for a control character.

iscntrl is a macro that classifies ASCII-coded integer values by table lookup. The macro is affected by the current locale's LC_CTYPE category. For the default C locale, *c* is a delete character or control character (0x7F or 0x00 to 0x1F).

You can make this macro available as a function by undefining (**#undef**) it.

Return Value

iscntrl returns nonzero if *c* is a delete character or ordinary control character.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

isdigit, iswdigit, _ismbcdigit

Example Portability

Syntax

```
#include <ctype.h>
int isdigit(int c);
int iswdigit(wint_t c);

#include <mbstring.h>
int _ismbcdigit(unsigned int c);
```

Description

Tests for decimal-digit character.

isdigit is a macro that classifies ASCII-coded integer values by table lookup. The macro is affected by the current locale's LC_CTYPE category. For the default C locale, *c* is a digit (0 to 9).

You can make this macro available as a function by undefining (**#undef**) it.

Return Value

isdigit returns nonzero if *c* is a digit.

_ismbcdigit returns true if and only if the argument *c* is a single-byte representation of an ASCII digit.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

isgraph, iswgraph, _ismbcgraph

Example Portability

Syntax

```
#include <ctype.h>
int isgraph(int c);
int iswgraph(wint_t c);

#include <mbstring.h>
int _ismbcgraph( unsigned int c);
```

Description

Tests for printing character.

isgraph is a macro that classifies ASCII-coded integer values by table lookup. The macro is affected by the current locale's LC_CTYPE category. For the default C locale, *c* is a printing character except blank space (' ').

You can make this macro available as a function by undefining (**#undef**) it.

Return Value

isgraph returns nonzero if *c* is a printing character.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

islower, iswlower, _ismbclower

Example Portability

Syntax

```
#include <ctype.h>
int islower(int c);
int iswlower(wint_t c);

#include <mbstring.h>
int _ismbclower(unsigned int c);
```

Description

Tests for lowercase character.

islower is a macro that classifies ASCII-coded integer values by table lookup. The macro is affected by the current locale's LC_CTYPE category. For the default C locale, *c* is a lowercase letter (*a*to *z*).

You can make this macro available as a function by undefining (**#undef**) it.

Return Value

islower returns nonzero if *c* is a lowercase letter.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

_ismbblead, _ismbbtrail

Syntax

```
#include <mbstring.h>
int _ismbblead(unsigned int c);
int _ismbbtrail(unsigned int c);
```

Description

_ismbblead and *_ismbbtrail* are used to test whether the argument *c* is the first or the second byte of a multibyte character.

_ismbblead and *_ismbbtrail* are affected by the code page in use. You can set the code page by using the *_setlocale* function.

Return Value

If *c* is in the lead byte of a multibyte character, *_ismbblead* returns true.

If *c* is in the trail byte of a multibyte character, *_ismbbtrail* returns a nonzero value.

_ismbclegal

Syntax

```
#include <mbstring.h>
int _ismbclegal(unsigned int c);
```

Description

_*ismbclegal* tests whether each byte of the *c* argument is in the code page that is currently in use.

Return Value

_*ismbclegal* returns a nonzero value if the argument *c* is a valid multibyte character on the current code page. Otherwise, the function returns zero.

_ismbslead, _ismbstrail

See also

Syntax

```
#include <mbstring.h>
int _ismbslead(const unsigned char *s1, const unsigned char *s2);
int _ismbstrail(const unsigned char *s1, const unsigned char *s2);
```

Description

The *_ismbslead* and *_ismbstrail* functions test the *s1* argument to determine whether the *s2* argument is a pointer to the lead byte or the trail byte. The test is case-sensitive.

Return Value

The *_ismbslead* and *_ismbstrail* routines return -1 if *s2* points to a lead byte or a trail byte, respectively. If the test is false, the routines return zero.

isprint, iswprint, _ismbcprint

Example   Portability

Syntax

```
#include <ctype.h>
int isprint(int c);
int iswprint(wint_t c);

#include <mbstring.h>
int _ismbcprint(unsigned int c);
```

Description

Tests for printing character.

isprint is a macro that classifies ASCII-coded integer values by table lookup. The macro is affected by the current locale's LC_CTYPE category. For the default C locale, *c* is a printing character including the blank space (' ').

You can make this macro available as a function by undefining (**#undef**) it.

Return Value

isprint returns nonzero if *c* is a printing character.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

ispunct, iswpunct, _ismbcpunct

Example Portability

Syntax

```
#include <ctype.h>
int ispunct(int c);
int iswpunct(wint_t c);

#include <mbstring.h>
int _ismbcpunct(unsigned int c);
```

Description

Tests for punctuation character.

ispunct is a macro that classifies ASCII-coded integer values by table lookup. The macro is affected by the current locale's LC_CTYPE category. For the default C locale, *c* is any printing character that is neither an alphanumeric nor a blank space (' ').

You can make this macro available as a function by undefining (**#undef**) it.

Return Value

ispunct returns nonzero if *c* is a punctuation character.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

isspace, iswspace, _ismbcspace

Example Portability

Syntax

```
#include <ctype.h>
int isspace(int c);
int iswspace(wint_t c);

#include <mbstring.h>
int _ismbcspace(unsigned int c);
```

Description

Tests for space character.

isspace is a macro that classifies ASCII-coded integer values by table lookup. The macro is affected by the current locale's LC_CTYPE category.

You can make this macro available as a function by undefining (**#undef**) it.

Return Value

isspace returns nonzero if *c* is a space, tab, carriage return, new line, vertical tab, formfeed (0x09 to 0x0D, 0x20), or any other locale-defined space character.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

isupper, iswupper, _ismbcupper

Example Portability

Syntax

```
#include <ctype.h>
int isupper(int c);
int iswupper(wint_t c);

#include <mbstring.h>
int _ismbcupper(unsigned int c);
```

Description

Tests for uppercase character.

isupper is a macro that classifies ASCII-coded integer values by table lookup. The macro is affected by the current locale's LC_CTYPE category. For the default C locale, *c* is an uppercase letter (*A* to *Z*).

You can make this macro available as a function by undefining (**#undef**) it.

Return Value

isupper returns nonzero if *c* is an uppercase letter.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

isxdigit, iswxdigit

Example Portability

Syntax

```
#include <ctype.h>
int isxdigit(int c);
int iswxdigit(wint_t c);
```

Description

Tests for hexadecimal character.

isxdigit is a macro that classifies ASCII-coded integer values by table lookup. The macro is affected by the current locale's LC_CTYPE category.

You can make this macro available as a function by undefining (*#undef*) it.

Return Value

isxdigit returns nonzero if *c* is a hexadecimal digit (0 to 9, *A* to *F*, *a* to *f*) or any other hexadecimal digit defined by the locale.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

itoa, _itow

See also Example Portability

Syntax

```
#include <stdlib.h>
char *itoa(int value, char *string, int radix);
wchar_t *_itow(int value, wchar_t *string, int radix);
```

Description

Converts an integer to a string.

itoa converts *value* to a null-terminated string and stores the result in *string*. With *itoa*, *value* is an integer.

radix specifies the base to be used in converting *value*; it must be between 2 and 36, inclusive. If *value* is negative and *radix* is 10, the first character of *string* is the minus sign (**-**).

Note: The space allocated for *string* must be large enough to hold the returned string, including the terminating null character (\0). *itoa* can return up to 17 bytes.

Return Value

itoa returns a pointer to *string*.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

kbhit

See also Example Portability

Syntax

```
#include <conio.h>
int kbhit(void);
```

Description

Checks for currently available keystrokes.

kbhit checks to see if a keystroke is currently available. Any available keystrokes can be retrieved with *getch* or *getche*.

Note: Do not use this function for Win32s or Win32 GUI applications.

Return Value

If a keystroke is available, *kbhit* returns a nonzero value. Otherwise, it returns 0.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

labs

See also Example Portability

Syntax

```
#include <math.h>
long labs(long int x);
```

Description

Gives long absolute value.

labs computes the absolute value of the parameter *x*.

Return Value

labs returns the absolute value of *x*.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

ldexp, ldexpl

See also Example Portability

Syntax

```
#include <math.h>
double ldexp(double x, int exp);
long double ldexpl(long double x, int exp);
```

Description

Calculates $x * 2^{exp}$

lexpl is the **long double** version; it takes a **long double** argument for *x* and returns a **long double** result.

Return Value

On success, *ldexp* (or *ldexpl*) returns the value it calculated, $x * 2^{exp}$. Error handling for these routines can be modified through the functions _matherr and _matherrl.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
ldexp	+	+	+	+	+	+	+
ldexpl	+		+	+			+

ldiv

See also Example Portability

Syntax

```
#include <stdlib.h>
ldiv_t ldiv(long int numer, long int denom);
```

Description

Divides two **longs**, returning quotient and remainder.

ldiv divides two **longs** and returns both the quotient and the remainder as an *ldiv_t* type. *numer* and *denom* are the numerator and denominator, respectively.

The *ldiv_t* type is a structure of **long**s defined in stdlib.h as follows:

```
typedef struct {
   long int quot;       /* quotient */
   long int rem;        /* remainder */
   } ldiv_t;
```

Return Value

ldiv returns a structure whose elements are *quot* (the quotient) and *rem* (the remainder).

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+	+	+	+

lfind

See also Example Portability

Syntax

```
#include <stdlib.h>
void  *lfind(const void *key, const void *base, size_t *num, size_t width,
  int (_USERENTRY *fcmp)(const void *, const void *));
```

Description

Performs a linear search.

lfind makes a linear search for the value of *key* in an array of sequential records. It uses a user-defined comparison routine *fcmp*. The *fcmp* function must be used with the _USERENTRY calling convention.

The array is described as having **num* records that are *width* bytes wide, and begins at the memory location pointed to by *base*.

Return Value

lfind returns the address of the first entry in the table that matches the search key. If no match is found, *lfind* returns NULL. The comparison routine must return 0 if **elem1* == **elem2*, and nonzero otherwise (*elem1* and *elem2* are its two parameters).

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

localeconv

See also Example Portability

Syntax

```
#include <locale.h>
struct lconv *localeconv(void);
```

Description

Queries the locale for numeric format.

This function provides information about the monetary and other numeric formats for the current locale. The information is stored in a **struct** *lconv* type. The structure can only be modified by the *setlocale*. Subsequent calls to *localeconv* will update the *lconv* structure.

The *lconv* structure is defined in locale.h. It contains the following fields:

Field	Application
char **decimal_point;*	Decimal point used in nonmonetary formats. This can never be an empty string.
char **thousands_sep;*	Separator used to group digits to the left of the decimal point. Not used with monetary quantities.
char **grouping;*	Size of each group of digits. Not used with monetary quantities. See the value listing table below.
char **int_curr_symbol;*	International monetary symbol in the current locale. The symbol format is specified in the ISO 4217 Codes for the Representation of Currency and Funds.
char **currency_symbol;*	Local monetary symbol for the current locale.
char **mon_decimal_point;*	Decimal point used to format monetary quantities.
char **mon_thousands_sep;*	Separator used to group digits to the left of the decimal point for monetary quantities.
char **mon_grouping;*	Size of each group of digits used in monetary quantities. See the value listing table below.
char **positive_sign;*	String indicating nonnegative monetary quantities.
char **negative_sign;*	String indicating negative monetary quantities.
char *int_frac_digits;*	Number of digits after the decimal point that are to be displayed in an internationally formatted monetary quantity.
char *frac_digits;*	Number of digits after the decimal point that are to be displayed in a formatted monetary quantity.
char *p_cs_precedes;*	Set to 1 if *currency_symbol* precedes a nonnegative formatted monetary quantity. If *currency_symbol* is after the quantity, it is set to 0.
char *p_sep_by_space;*	Set to 1 if *currency_symbol* is to be separated from the nonnegative formatted monetary quantity by a space. Set to 0 if there is no space separation.
char *n_cs_precedes;*	Set to 1 if *currency_symbol* precedes a negative formatted monetary quantity. If *currency_symbol* is after the quantity, set to 0.
char *n_sep_by_space;*	Set to 1 if *currency_symbol* is to be separated from the negative formatted monetary quantity by a space. Set to 0 if there is no space separation.
char *p_sign_posn;*	Indicate where to position the positive sign in a nonnegative formatted monetary quantity.

char *n_sign_posn;*	Indicate where to position the positive sign in a negative formatted monetary quantity.

Any of the above strings (except *decimal_point*) that is empty " " is not supported in the current locale. The nonstring **char** elements are nonnegative numbers. Any nonstring **char** element that is set to CHAR_MAX indicates that the element is not supported in the current locale.

The *grouping* and *mon_grouping* elements are set and interpreted as follows:

Value	Meaning
CHAR_MAX	No further grouping is to be performed.
0	The previous element is to be used repeatedly for the remainder of the digits.
any other integer	Indicates how many digits make up the current group. The next element is read to determine the size of the next group of digits before the current group.

The *p_sign_posn* and *n_sign_posn* elements are set and interpreted as follows:

Value	Meaning
0	Use parantheses to surround the quantity and currency_symbol.
1	Sign string precedes the quantity and currency_symbol.
2	Sign string succeeds the quatity and currency_symbol.
3	Sign string immediately precedes the quantity and currency_symbol.
4	Sign string immediately succeeds the quantity and currency_symbol.

Return Value

Returns a pointer to the the filled-in structure of type **struct** *lconv*. The values in the structure will change whenever *setlocale* modifies the LC_MONETARY or LC_NUMERIC categories.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+	+	+	+

localtime

[See also](#) [Example](#) [Portability](#)

Syntax

```
#include <time.h>
struct tm *localtime(const time_t *timer);
```

Description

Converts date and time to a structure.

localtime accepts the address of a value returned by *time* and returns a pointer to the structure of type *tm* containing the time elements. It corrects for the time zone and possible daylight saving time.

The global long variable _*timezone* contains the difference in seconds between GMT and local standard time (in PST, _*timezone* is 8 x 60 x 60). The global variable *daylight* contains nonzero *only if* the standard U.S. daylight saving time conversion should be applied. These values are set by *tzset*, not by the user program directly.

This is the **tm** structure declaration from the time.h header file:

```
struct tm {
   int tm_sec;
   int tm_min;
   int tm_hour;
   int tm_mday;
   int tm_mon;
   int tm_year;
   int tm_wday;
   int tm_yday;
   int tm_isdst;
};
```

These quantities give the time on a 24-hour clock, day of month (1 to 31), month (0 to 11), weekday (Sunday equals 0), year - 1900, day of year (0 to 365), and a flag that is nonzero if _daylight saving time is in effect.

Return Value

localtime returns a pointer to the structure containing the time elements. This structure is a static that is overwritten with each call.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

lock

See also Example Portability

Syntax

```
#include <io.h>
int lock(int handle, long offset, long length);
```

Description

Sets file-sharing locks. DOS users must be sure to load SHARE.EXE before using *lock.*

lock provides an interface to the operating system file-sharing mechanism.

A lock can be placed on arbitrary, nonoverlapping regions of any file. A program attempting to read or write into a locked region will retry the operation three times. If all three retries fail, the call fails with an error.

Return Value

lock returns 0 on success. On error, *lock* returns -1 and sets the global variable errno to

EACCES	Locking violation

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

locking

See also   Example   Portability

Syntax

```
#include <io.h>
#include <sys\locking.h>
int locking(int handle, int cmd, long length);
```

Description

Sets or resets file-sharing locks. DOS users must be sure to load SHARE.EXE before using *locking.*

locking provides an interface to the operating system file-sharing mechanism. The file to be locked or unlocked is the open file specified by *handle*. The region to be locked or unlocked starts at the current file position, and is *length* bytes long.

Locks can be placed on arbitrary, nonoverlapping regions of any file. A program attempting to read or write into a locked region will retry the operation three times. If all three retries fail, the call fails with an error.

The *cmd* specifies the action to be taken (the values are defined in sys\locking.h):

LK_LOCK	Lock the region. If the lock is unsuccessful, try once a second for 10 seconds before giving up.
LK_RLCK	Same as LK_LOCK.
LK_NBLCK	Lock the region. If the lock if unsuccessful, give up immediately.
LK_NBRLCK	Same as LK_NBLCK.
LK_UNLCK	Unlock the region, which must have been previously locked.

Return Value

On successful operations, *locking* returns 0. Otherwise, it returns -1, and the global variable errno is set to one of the following values:

EACCES	File already locked or unlocked
EBADF	Bad file number
EDEADLOCK	File cannot be locked after 10 retries (*cmd* is LK_LOCK or LK_RLCK)
EINVAL	Invalid *cmd*, or SHARE.EXE not loaded

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

log, logl

See also Example Portability

Syntax

```
#include <math.h>
double log(double x);
long double logl(long double x);
```

Description

Calculates the natural logarithm of *x*.

log calculates the natural logarithm of *x*.

logl is the **long double** version; it takes a **long double** argument and returns a **long double** result.

Return Value

On success, *log* and *logl* return the value calculated, *ln(x)*.

If the argument *x* passed to these functions is real and less than 0, the global variable errno is set to

EDOM	Domain error

If *x* is 0, the functions return the value negative HUGE_VAL (*log*) or negative _LHUGE_VAL (*logl*), and set errno to ERANGE. Error handling for these routines can be modified through the functions _matherr and _matherrl.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
log	+	+	+	+	+	+	+
logl	+		+	+			+

log10, log10l

See also Example Portability

Syntax

```
#include <math.h>
double log10(double x);
long double log10l(long double x);
```

Description

log10 calculates the base 10 logarithm of *x*.

log10l is the **long double** version; it takes a **long double** argument and returns a **long double** result.

Return Value

On success, *log10* (or *log10l*) returns the value calculated, $\log_{10}(x)$.

If the argument *x* passed to these functions is real and less than 0, the global variable errno is set to

EDOM Domain error

If *x* is 0, these functions return the value negative HUGE_VAL (*log10*) or _LHUGE_VAL (*log10l*). Error handling for these routines can be modified through the functions _matherr and _matherrl.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
log10	+	+	+	+	+	+	+
log10l	+		+	+			+

longjmp

See also Example Portability

Syntax

```
#include <setjmp.h>
void longjmp(jmp_buf jmpb, int retval);
```

Description

Performs nonlocal goto.

A call to *longjmp* restores the task state captured by the last call to setjmp with the argument *jmpb*. It then returns in such a way that *setjmp* appears to have returned with the value *retval*.

A task state includes

Win 16	Win 32
All segment registers CS, DS, ES, SS	No segment registers are saved
Register variables	Register variables
DI and SI	EBX, EDI, ESI
Stack pointer SP	Stack pointer ESP
Frame pointer BP	Frame pointer EBP
Flags	Flags are not saved

A task state is complete enough that *setjmp* and *longjmp* can be used to implement co-routines.

setjmp must be called before *longjmp*. The routine that called *setjmp* and set up *jmpb* must still be active and cannot have returned before the *longjmp* is called. If this happens, the results are unpredictable.

longjmp cannot pass the value 0; if 0 is passed in *retval*, *longjmp* will substitute 1.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

lowvideo

See also Example Portability

Syntax

```
#include <conio.h>
void lowvideo(void);
```

Description

Selects low-intensity characters.

lowvideo selects low-intensity characters by clearing the high-intensity bit of the currently selected foreground color.

This function does not affect any characters currently onscreen. It affects only those characters displayed by functions that perform text mode, direct console output *after* this function is called.

Note: Do not use this function for Win32s or Win32 GUI applications.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+			+			+

_lrotl, _lrotr

See also Example Portability

Syntax

```
#include <stdlib.h>
unsigned long _lrotl(unsigned long val, int count);
unsigned long _lrotr(unsigned long val, int count);
```

Description

Rotates an **unsigned long** integer value to the left or right.

*_lrotl*rotates the given *val* to the left *count* bits. *_lrotr* rotates the given *val* to the right *count* bits.

Return Value

The functions return the rotated integer:

- *_lrotl* returns the value of *val* left-rotated *count* bits.
- *_lrotr* returns the value of *val* right-rotated *count* bits.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

lsearch

See also Example Portability

Syntax

```
#include <stdlib.h>
void *lsearch(const void *key, void *base, size_t *num, size_t width, int
  (_USERENTRY *fcmp)(const void *, const void *));
```

Description

Performs a linear search.

lsearch searches a table for information. Because this is a linear search, the table entries do not need to be sorted before a call to *lsearch*. If the item that *key* points to is not in the table, *lsearch* appends that item to the table.

- *base* points to the base (0th element) of the search table.
- *num* points to an integer containing the number of entries in the table.
- *width* contains the number of bytes in each entry.
- *key* points to the item to be searched for (the *search key*).

The function *fcmp* must be used with the _USERENTRY calling convention.

The argument *fcmp* points to a user-written comparison routine, that compares two items and returns a value based on the comparison.

To search the table, *lsearch* makes repeated calls to the routine whose address is passed in *fcmp*.

On each call to the comparison routine, *lsearch* passes two arguments:

key	a pointer to the item being searched for
elem	pointer to the element of *base* being compared.

fcmp is free to interpret the search key and the table entries in any way.

Return Value

lsearch returns the address of the first entry in the table that matches the search key.

If the search key is not identical to **elem*, *fcmp* returns a nonzero integer. If the search key is identical to **elem*, *fcmp* returns 0.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

lseek

See also   Example   Portability

Syntax

```
#include <io.h>
long lseek(int handle, long offset, int fromwhere);
```

Description

Moves file pointer.

lseek sets the file pointer associated with *handle* to a new position *offset* bytes beyond the file location given by *fromwhere*. *fromwhere* must be one of the following symbolic constants (defined in io.h):

fromwhere	File location
SEEK_CUR	Current file pointer position
SEEK_END	End-of-file
SEEK_SET	File beginning

Return Value

lseek returns the offset of the pointer's new position measured in bytes from the file beginning. *lseek* returns -1L on error, and the global variable errno is set to one of the following values:

EBADF	Bad file handle
EINVAL	Invalid argument
ESPIPE	Illegal seek on device

On devices incapable of seeking (such as terminals and printers), the return value is undefined.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

_ltoa, _ltow, _i64toa, _ui64toa, _i64tow, _ui64tow

See also   Example   Portability

Syntax

```
#include <stdlib.h>
char *_ltoa(long value, char *string, int radix);
char *_i64toa(__int64 value, char *strP, int radix);
char *_ui64toa(unsigned __int64 value, char *strP, int radix);

// The following are Unicode versions
wchar_t *_ltow(long value, wchar_t *string, int radix);
wchar_t *_i64tow(__int64 value, wchar_t *strP, int radix);
wchar_t *_ui64tow(unsigned __int64 value, wchar_t *strP, int radix);
```

Description

Converts a **long** to a string.

_*ltoa* converts *value* to a null-terminated string and stores the result in *string*. *value* is a long integer.

radix specifies the base to be used in converting *value*; it must be between 2 and 36, inclusive. If *value* is negative and *radix* is 10, the first character of *string* is the minus sign (**-**).

Note: The space allocated for *string* must be large enough to hold the returned string, including the terminating null character (\0). _*ltoa* can return up to 33 bytes.

Return Value

_*ltoa* returns a pointer to *string*.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

_makepath, _wmakepath

See also   Example   Portability

Syntax

```
#include <stdlib.h>
void _makepath(char *path, const char *drive, const char *dir, const char
  *name, const char *ext);
void _wmakepath(wchar_t *path, const wchar_t *drive, const wchar_t *dir,
  const wchar_t *name, const wchar_t *ext);
```

Description

Builds a path from component parts.

_makepath makes a path name from its components. The new path name is

```
X:\DIR\SUBDIR\NAME.EXT
```

where

drive	=	X:
dir	=	\DIR\SUBDIR\
name	=	NAME
ext	=	.EXT

If *drive* is empty or NULL, no drive is inserted in the path name. If it is missing a trailing colon (:), a colon is inserted in the path name.

If *dir* is empty or NULL, no directory is inserted in the path name. If it is missing a trailing slash (\ or /), a backslash is inserted in the path name.

If *name* is empty or NULL, no file name is inserted in the path name.

If *ext* is empty or NULL, no extension is inserted in the path name. If it is missing a leading period (.), a period is inserted in the path name.

_makepath assumes there is enough space in *path* for the constructed path name. The maximum constructed length is _MAX_PATH. _MAX_PATH is defined in stdlib.h.

_makepath and *_splitpath* are invertible; if you split a given *path* with *_splitpath*, then merge the resultant components with *_makepath*, you end up with *path*.

Return Value

None

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

malloc

See also   Example   Portability

Syntax

```
#include <stdlib.h> or #include<alloc.h>
void *malloc(size_t size);
```

Description

malloc allocates a block of *size* bytes from the memory heap. It allows a program to allocate memory explicitly as it's needed, and in the exact amounts needed.

Allocates main memory.The heap is used for dynamic allocation of variable-sized blocks of memory. Many data structures, for example, trees and lists, naturally employ heap memory allocation.

For 16-bit programs, all the space between the end of the data segment and the top of the program stack is available for use in the small data models, except for a small margin immediately before the top of the stack. This margin is intended to allow the application some room to make the stack larger, in addition to a small amount needed by DOS.

In the large data models, all the space beyond the program stack to the end of available memory is available for the heap.

Return Value

On success, *malloc* returns a pointer to the newly allocated block of memory. If not enough space exists for the new block, it returns NULL. The contents of the block are left unchanged. If the argument *size* == 0, *malloc* returns NULL.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

_matherr, _matherrl

Example Portability

Syntax

```
#include <math.h>
int _matherr(struct _exception *e);
int _matherrl(struct _exceptionl *e);
```

Description

User-modifiable math error handler.

_matherr is called when an error is generated by the math library.

_matherrl is the **long double** version; it is called when an error is generated by the *long double* math functions.

_matherr and *_matherrl* each serve as a user hook (a function that can be customized by the user) that you can replace by writing your own math error-handling routine.

_matherr and *_matherrl* are useful for information on trapping domain and range errors caused by the math functions. They do not trap floating-point exceptions, such as division by zero. See *signal* for information on trapping such errors.

You can define your own *_matherr* or *_matherrl* routine to be a custom error handler (such as one that catches and resolves certain types of errors); this customized function overrides the default version in the C library. The customized *_matherr* or *_matherrl* should return 0 if it fails to resolve the error, or nonzero if the error is resolved. When *_matherr* or *_matherrl* return nonzero, no error message is printed and the global variable errno is not changed.

Here are the *_exception* and *_exceptionl* structures (defined in math.h):

```
struct _exception {
   int    type;
   char   *name;
   double arg1, arg2, retval;
};

struct _exceptionl {
   int    type;
   char   *name;
   long double arg1, arg2, retval;
};
```

The members of the *_exception* and *_exceptionl* structures are shown in the following table:

Member	What It Is (Or Represents)
type	The type of mathematical error that occurred; an enum type defined in the typedef *_mexcep* (see definition after this list).
name	A pointer to a null-terminated string holding the name of the math library function that resulted in an error.
arg1, arg2	The arguments (passed to the function that name points to) caused the error; if only one argument was passed to the function, it is stored in arg1.
retval	The default return value for _matherr (or _matherrl); you can modify this value.

The **typedef** *_mexcep*, also defined in math.h, enumerates the following symbolic constants representing possible mathematical errors:

Symbolic Constant	Mathematical Error
DOMAIN	Argument was not in domain of function, such as log(-1).
SING	Argument would result in a singularity, such as pow(0, -2).

OVERFLOW	Argument would produce a function result greater than DBL_MAX (or LDBL_MAX), such as exp(1000).
UNDERFLOW	Argument would produce a function result less than DBL_MIN (or LDBL_MIN), such as exp(-1000).
TLOSS	Argument would produce function result with total loss of significant digits, such as sin(10e70).

The macros DBL_MAX, DBL_MIN, LDBL_MAX, and LDBL_MIN are defined in float.h

The source code to the default *_matherr* and *_matherrl* is on the C++Builder distribution disks.

The UNIX-style *_matherr* and *_matherrl* default behavior (printing a message and terminating) is not ANSI compatible. If you want a UNIX-style version of these routines, use MATHERR.C and MATHERRL.C provided on the C++Builder distribution disks.

Disabling floating-point exceptions

By default, programs abort if a floating-point overflow or divide-by-zero error occurs. You can mask these floating-point exceptions by a call to *_control87* in *main*, before any floating-point operations are performed.

Example

```
#include <float.h>
main() {
   _control87(MCW_EM,MCW_EM);
   .
   .
   .
}
```

You can determine whether a floating-point exception occurred after the fact by calling _status87 or _clear87.

Certain math errors can also occur in library functions; for instance, if you try to take the square root of a negative number. The default behavior is to print an error message to the screen, and to return a NAN (an IEEE not-a-number). Use of the NAN is likely to cause a floating-point exception later, which will abort the program if unmasked. If you don't want the message to be printed, insert the following version of *_matherr* into your program:

```
#include <math.h>
int _matherr(struct _exception *e)
{
   return 1;             /* error has been handled */
}
```

Any other use of *_matherr* to intercept math errors is not encouraged; it is considered obsolete and might not be supported in future versions of C++Builder.

Return Value

The default return value for *_matherr* and *_matherrl* is 1 if the error is UNDERFLOW or TLOSS, 0 otherwise. *_matherr* and *_matherrl* can also modify *e* -> *retval*, which propagates back to the original caller.

When *_matherr* and *_matherrl* return 0 (indicating that they were not able to resolve the error), the global variable errno is set to 0 and an error message is printed.

When _matherr and _matherrl return nonzero (indicating that they were able to resolve the error), the global variable errno is not set and no messages are printed.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

max

See also Example Portability

Syntax

```
#include <stdlib.h> /* macro version */
(type) max(a, b);
template <class T> T max( T t1, T t2 ); // C++ only
```

Description

Returns the larger of two values.

The C macro and the C++ template function compare two values and return the larger of the two. Both arguments and the routine declaration must be of the same type.

Return Value

max returns the larger of two values.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

_mbbtype

See also

Syntax

```
#include <mbstring.h>
int _mbbtype(unsigned char ch, int mode);
```

Description

The *_mbbtype* function inspects the multibyte argument, character *ch*, to determine whether it is a single-byte character, or whether *ch* is the leadbyte or trailing byte in a multibyte character. The *_mbbtype* function can determine whether *ch* is an invalid character.

Return Value

The value that *_mbbtype* returns is one of the following manifest constants, defined in mbctype.h. The return value depends on the value of *ch* and the test which you want performed on *ch*.

Value of *mode*	Value of *ch*	Test performed	Return value
mode != 1	Single byte	Valid single or lead byte	**_MBC_SINGLE**
mode != 1	Leadbyte	Valid single or lead byte	**_MBC_LEAD**
mode = 1	Trailbyte	Valid single or trail byte	**_MBC_TRAIL**
Any value	Any value	Valid character	**_MBC_ILLEGAL**

_mbccmp

_mbccpy

Syntax

```
#include <mbstring.h>
void _mbccpy(unsigned char *dest, unsigned char *src);
```

Description

The *_mbccpy* function copies a multibyte character from *src* to *dest*. The *_mbccpy* function makes an implicit call to *_ismbblead* so that the *src* pointer references a lead byte. If *src* doesn't reference a lead byte, no copy is performed.

Return Value

None.

mblen

See also Example Portability

Syntax

```
#include <stdlib.h>
int mblen(const char *s, size_t n);
```

Description

Determines the length of a multibyte character.

If *s* is not null, *mblen* determines the number of bytes in the multibyte character pointed to by *s*. The maximum number of bytes examined is specified by *n*.

The behavior of *mblen* is affected by the setting of LC_CTYPE category of the current locale.

Return Value

If *s* is null, *mblen* returns a nonzero value if multibyte characters have state-dependent encodings. Otherwise, *mblen* returns 0.

If *s* is not null, *mblen* returns 0 if *s* points to the null character, and -1 if the next *n* bytes do not comprise a valid multibyte character; the number of bytes that comprise a valid multibyte character.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+	+	+	+

_mbsbtype

See also

Syntax

```
#include <mbstring.h>
int _mbsbtype(const unsigned char *str, size_t nbyte);
```

Description

The *nbyte* argument specifies the number of bytes from the start of the zero-based string.

The *_mbsbtype* function inspects the argument *str* to determine whether the byte at the position specified by *nbyte* is a single-byte character, or whether it is the leadbyte or trailing byte in a multibyte character. The *_mbsbtype* function can determine whether the byte pointed at is an invalid character or a NULL byte.

Any invalid bytes in *str* before *nbyte* are ignored.

Return Value

The value that *_mbsbtype* returns is one of the following manifest constants, defined in mbctype.h.

Type of byte found	Return value
Single byte	**_MBC_SINGLE**
Leadbyte	**_MBC_LEAD**
Trailbyte	**_MBC_TRAIL**
Invalid character or byte	**_MBC_ILLEGAL**

_mbsnbcmp

See also

Syntax

```
#include <mbstring.h>
int _mbsnbcmp(const unsigned char *s1, const unsigned char s2, size_t
  maxlen);
```

Description

_mbsnbcmp makes an case-sensitive comparison of *s1* and *s2* for no more than *maxlen* bytes. It starts with the first byte in each string and continues with subsequent bytes until the corresponding bytes differ or until it has examined *maxlen* bytes.

_mbsnbcmp is case sensitive.

_mbsnbcmp is not affected by locale.

_mbsnbcmp compares bytes based on the current multibyte code page.

Return Value

_mbsnbcmp returns an integer value based on the result of comparing *s1* (or part of it) to *s2* (or part of it):

- < 0 if *s1* is less than *s2*
- == 0 if *s1* is the same as *s2*
- > 0 if *s1* is greater than *s2*

_mbsnbcoll, _mbsnbicoll

See also

Syntax

```
#include <mbstring.h>
int _mbsnbcoll(const unsigned char *s1, const unsigned char *s2, maxlen);
int _mbsnbicoll(const unsigned char *s1, const unsigned char *s2, maxlen);
```

Description

_mbsnbicoll is the case-insensitive version of *_mbsnbcoll*.

These functions collate the strings specified by arguments *s1* and *s2*. The collation order is determined by lexicographic order as specified by the current multibyte code page. At most, *maxlen* number of bytes are collated.

Note: The lexicographic order is not always the same as the order of characters in the character set.

If the last byte in *s1* or *s2* is a leadbyte, it is not compared.

Return Value

Each of these functions return an integer value based on the result of comparing *s1* (or part of it) to *s2* (or part of it):

- < 0 if *s1* is less than *s2*
- == 0 if *s1* is the same as *s2*
- > 0 if *s1* is greater than *s2*

On error, each of these functions returns **_NLSCMPERROR**.

_mbsnbcpy

See also

Syntax

```
#include <mbstring.h>
unsigned char *_mbsnbcpy(unsigned char *dest, unsigned char *src, size_t
  maxlen);
```

Description

The *_mbsnbcpy* function copies at most *maxlen* number of characters from the *src* buffer to the *dest* buffer. The *dest* buffer is null terminated after the copy.

It is the user's responsibility to be sure that *dest* is large enough to allow the copy. An improper buffer size can result in memory corruption.

Return Value

The functon returns *dest*.

_mbsnbicmp

See also

Syntax

```
#include <mbstring.h>
int _mbsnbicmp(const unsigned char *s1, const unsigned char s2, size_t
  maxlen);
```

Description

_mbsnbicmp ignores case while making a comparison of *s1* and *s2* for no more than *maxlen* bytes. It starts with the first byte in each string and continues with subsequent bytes until the corresponding bytes differ or until it has examined *maxlen* bytes.

_mbsnbicmp is not case sensitive.

_mbsnbicmp is not affected by locale.

_mbsnbicmp compares bytes based on the current multibyte code page.

Return Value

_mbsnbicmp returns an integer value based on the result of comparing *s1* (or part of it) to *s2* (or part of it):

- < 0 if *s1* is less than *s2*
- == 0 if *s1* is the same as *s2*
- > 0 if *s1* is greater than *s2*

_mbsnbset

See also

Syntax

```
#include <mbstring.h>
unsigned char *_mbsnbset(unsigned char str, unsigned int ch, size_t maxlen);
```

Description

_mbsnbset sets at most *maxlen* number of bytes in the string *str* to the character *ch*. The argument *ch* can be a single or multibyte character.

The function quits if the terminating null character is found before *maxlen* is reached. If *ch* is a multibyte character that cannot be accomodated at the end of *str*, the last character in *str* is set to a blank character.

Return Value

strset returns *str*.

_mbsninc, _strninc, _wcsninc

See also

Syntax

```
#include <mbstring.h>
unsigned char *_mbsninc(const unsigned char *str, size_t num);
```

Description

These functions should be accessed throught the portable macro, *_tcsninc*, defined in tchar.h.

The functions increment the character array *str* by *num* number of characters.

Return value

The functions return a pointer to the resized character string specified by the argument *str*.

_mbsnbcnt, _mbsnccnt, _strncnt, _wcsncnt

See also

Syntax

```
#include <mbstring.h>
size_t  _mbsnbcnt(const unsigned char * str, size_t nmbc);
size_t  _mbsnccnt(const unsigned char * str, size_t nbyte);
```

Description

If _MBCS is defined:

- *_mbsnbcnt* is mapped to the portable macro *_tcsnbcnt*
- *_mbsnccnt* is mapped to the portable macro *_tcsnccnt*

If _UNICODE is defined:

- both *_mbsnbcnt* and *_mbsnccnt* are mapped to the *_wcsncnt* macro

If neither _MBCS nor _UNICODE are defined.

- *_tcsnbcnt* and *_tcsnccnt* are mapped to the *_strncnt* macro

_strncnt is the single-byte version of these functions.

_wcsncnt is the wide-character version of these functions.

_strncnt and *_wcsncnt* are available only for generic-text mappings. They should not be used directly.

_mbsnbcnt examines the first *nmbc* multibyte characters of the *str* argument. The function returns the number of bytes found in the those characters.

_mbsnccnt examines the first *nmbc* bytes of the *str* argument. The function returns the number of characters found in those bytes. If NULL is encountered in the second byte of a multibyte character, the whole character is considered NULL and will not be included in the return value.

Each of the functions ends its examination of the *str* argument if NULL is reached before the specified number of characters or bytes is examined.

If *str* has fewer than the specified number of characters or bytes, the function return the number of characters or bytes found in *str*.

Return Value

_mbsnbcnt returns the number of bytes found.

_mbsnccnt returns the number of characters found.

If *nmbc* or *nbyte* are less than zero, the functions return 0.

_mbsspnp, _strspnp, _wcsspnp

See also Example

Syntax

```
#include <mbstring.h>
unsigned char *_mbsspnp(const unsigned char *s1, const unsigned char *s2);
```

Description

Use the portable macro, *_tcsspnp*, defined in tchar.h, to access these functions.

Each of these functions search for the first character in *s1* that is not contained in *s2*.

Return Value

The functions return a pointer to the first character in *s1* that is not found in the character set for *s2*.

If every character from *s1* is found in *s2*, each of the functions return NULL.

mbstowcs

See also Example Portability

Syntax

```
#include <stdlib.h>
size_t mbstowcs(wchar_t *pwcs, const char *s, size_t n);
```

Description

Converts a multibyte string to a **wchar_t** array.

The function converts the multibyte string *s* into the array pointed to by *pwcs*. No more than *n* values are stored in the array. If an invalid multibyte sequence is encountered, *mbstowcs* returns (*size_t*) -1.

The *pwcs* array will not be terminated with a zero value if *mbstowcs* returns *n*.

Return Value

If an invalid multibyte sequence is encountered, *mbstowcs* returns (*size_t*) -1. Otherwise, the function returns the number of array elements modified, not including the terminating code, if any.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+	+	+	+

mbtowc

See also Example Portability

Syntax

```
#include <stdlib.h>
int mbtowc(wchar_t *pwc, const char *s, size_t n);
```

Description

Converts a multibyte character to **wchar_t** code.

If *s* is not null, *mbtowc* determines the number of bytes that comprise the multibyte character pointed to by *s*. Next, *mbtowc* determines the value of the type **wchar_t** that corresponds to that multibyte character. If there is a successful match between **wchar_t** and the multibyte character, and *pwc* is not null, the **wchar_t** value is stored in the array pointed to by *pwc*. At most *n* characters are examined.

Return Value

When *s* points to an invalid multibyte character, -1 is returned. When *s* points to the null character, 0 is returned. Otherwise, *mbtowc* returns the number of bytes that comprise the converted multibyte character.

The return value never exceeds MB_CUR_MAX or the value of *n*.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+	+	+	+

memccpy

See also   Example   Portability

Syntax

```
#include <mem.h>
void *memccpy(void *dest, const void *src, int c, size_t n);
```

Description

Copies a block of *n* bytes.

memccpy is available on UNIX System V systems.

memccpy copies a block of *n* bytes from *src* to *dest*. The copying stops as soon as either of the following occurs:

- The character *c* is first copied into *dest*.
- *n* bytes have been copied into *dest*.

Return Value

memccpy returns a pointer to the byte in *dest* immediately following *c*, if *c* was copied; otherwise, *memccpy* returns NULL.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
memccpy	+	+	+	+			+
_fmemccpy	+		+				

memchr, _wmemchr

Example Portability

Syntax

```
#include <mem.h>
void *memchr(const void *s, int c, size_t n);                    /* C
  only */

const void *memchr(const void *s, int c, size_t n);              // C++
  only
void *memchr(void *s, int c, size_t n);                          // C++
  only
void *memchr(const void *s, int c, size_t n);
void * _wmemchr(void *s, int c, size_t n);
```

Description

Searches *n* bytes for character *c*.

memchr is available on UNIX System V systems.

memchr searches the first *n* bytes of the block pointed to by *s* for character *c*.

Return Value

On success, *memchr* returns a pointer to the first occurrence of *c* in *s*; otherwise, it returns NULL.

Note: If you are using the intrinsic version of these functions, the case of *n* = 0 will return NULL.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
memchr	+	+	+	+	+	+	+
_fmemchr	+		+				

memcmp

See also Example Portability

Syntax

```
#include <mem.h>
int memcmp(const void *s1, const void *s2, size_t n);
```

Description

Compares two blocks for a length of exactly *n* bytes.

memcmp is available on UNIX System V systems.

memcmp compares the first *n* bytes of the blocks *s1* and *s2* as **unsigned char**s.

Return Value

Because it compares bytes as **unsigned char**s, *memcmp* returns a value that is

- < 0 if *s1* is less than *s2*
- = 0 if *s1* is the same as *s2*
- > 0 if *s1* is greater than *s2*

For example,

```
memcmp("\xFF", "\x7F", 1)
```

returns a value greater than 0.

Note: If you are using the intrinsic version of these functions, the case of *n* = 0 will return NULL.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
memcmp	+	+	+	+	+	+	+
_fmemcmp	+		+				

memcpy, _wmemcpy

See also Example Portability

Syntax

```
#include <mem.h>
void *memcpy(void *dest, const void *src, size_t n);
void *_wmemcpy(void *dest, const void *src, size_t n);
```

Description

Copies a block of *n* bytes.

memcpy is available on UNIX System V systems.

memcpy copies a block of *n* bytes from *src* to *dest*. If *src* and *dest* overlap, the behavior of *memcpy* is undefined.

Return Value

memcpy returns *dest*.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
memcpy	+	+	+	+	+	+	+
_fmemcpy	+		+				

memicmp

See also Example Portability

Syntax

```
#include <mem.h>
int memicmp(const void *s1, const void *s2, size_t n);
```

Description

Compares *n* bytes of two character arrays, ignoring case.

memicmp is available on UNIX System V systems.

memicmp compares the first *n* bytes of the blocks *s1* and *s2*, ignoring character case (upper or lower).

Return Value

memicmp returns a value that is

- < 0 if *s1* is less than *s2*
- = 0 if *s1* is the same as *s2*
- > 0 if *s1* is greater than *s2*

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
memicmp	+	+	+	+			+
_fmemicmp	+		+				

memmove

See also Example Portability

Syntax

```
#include <mem.h>
void *memmove(void *dest, const void *src, size_t n);
```

Description

Copies a block of *n* bytes.

memmove copies a block of *n* bytes from *src* to *dest*. Even when the source and destination blocks overlap, bytes in the overlapping locations are copied correctly.

Return Value

memmove returns *dest*.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
memmove	+	+	+	+	+	+	+
	+		+				

memset, _wmemset

See also Example Portability

Syntax

```
#include <mem.h>
void *memset(void *s, int c, size_t n);
void *_wmemset(void *s, int c, size_t n);
```

Description

Sets *n* bytes of a block of memory to byte *c*.

memset sets the first *n* bytes of the array *s* to the character *c*.

Return Value

memset returns *s*.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
memset	+	+	+	+	+	+	+
_fmemset	+		+				

min

See also Example Portability

Syntax

```
#include <stdlib.h> /* macro version */
(type) min(a, b);
template <class T> T min( T t1, T t2 ); // C++ only
```

Description

Returns the smaller of two values.

The C macro and the C++ template function compare two values and return the smaller of the two. Both arguments and the routine declaration must be of the same type.

Return Value

min returns the smaller of two values.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

mkdir, _wmkdir

See also Example Portability

Syntax

```
#include <dir.h>
int mkdir(const char *path);
int _wmkdir(const wchar_t *path);
```

Description

Creates a directory.

mkdir is available on UNIX, though it then takes an additional parameter.

mkdir creates a new directory from the given path name *path*.

Return Value

mkdir returns the value 0 if the new directory was created.

A return value of -1 indicates an error, and the global variable errno is set to one of the following values:

EACCES	Permission denied
ENOENT	No such file or directory

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

_mktemp, _wmktemp

Example Portability

Syntax

```
#include <dir.h>
char *_mktemp(char *template);
wchar_t *_wmktemp(wchar_t *template);
```

Description

Makes a unique file name.

_mktemp replaces the string pointed to by *template* with a unique file name and returns *template*.

template should be a null-terminated string with six trailing Xs. These Xs are replaced with a unique collection of letters plus a period, so that there are two letters, a period, and three suffix letters in the new file name.

Starting with AA.AAA, the new file name is assigned by looking up the name on the disk and avoiding pre-existing names of the same format.

Return Value

If a unique name can be created and *template* is well-formed, *_mktemp* returns the address of the *template* string. Otherwise, it returns null.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

mktime

See also Example Portability

Syntax

```
#include <time.h>
time_t mktime(struct tm *t);
```

Description

Converts time to calendar format.

Converts the time in the structure pointed to by *t* into a calendar time with the same format 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 the ranges described in the *tm* structure. If the fields are not in their proper ranges, they are adjusted. Values for fields *tm_wday* and *tm_yday* are computed after the other fields have been adjusted.

The allowable range of calendar times is Jan 1 1970 00:00:00 to Jan 19 2038 03:14:07.

Return Value

On success, *mktime* returns calendar time as described above.

On error (if the calendar time cannot be represented), *mktime* returns -1.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+	+	+	+

modf, modfl

See also   Example   Portability

Syntax

```
#include <math.h>
double modf(double x, double *ipart);
long double modfl(long double x, long double *ipart);
```

Description

Splits a **double** or **long double** into integer and fractional parts.

modf breaks the **double** *x* into two parts: the integer and the fraction. *modf* stores the integer in *ipart* and returns the fraction.

modfl is the **long double** version; it takes **long double** arguments and returns a **long double** result.

Return Value

modf and *modfl* return the fractional part of *x*.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
modf	+	+	+	+	+	+	+
modfl	+		+	+			+

movetext

See also   Example   Portability

Syntax

```
#include <conio.h>
int movetext(int left, int top, int right, int bottom, int destleft, int
  desttop);
```

Description

Copies text onscreen from one rectangle to another.

movetext copies the contents of the onscreen rectangle defined by *left, top, right*, and *bottom* to a new rectangle of the same dimensions. The new rectangle's upper left corner is position (*destleft, desttop*).

All coordinates are absolute screen coordinates. Rectangles that overlap are moved correctly.

movetext is a text mode function performing direct video output.

Note: Do not use this function for Win32s or Win32 GUI applications.

Return Value

On success, *movetext* returns nonzero.

On error (for example, if it failed because you gave coordinates outside the range of the current screen mode), *movetext* returns 0.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+			+			+

_msize

See also Example Portability

Syntax

```
#include <malloc.h>
size_t _msize(void *block);
```

Description

Returns the size of a heap block.

_msize returns the size of the allocated heap block whose address is *block*. The block must have been allocated with *malloc, calloc*, or *realloc*. The returned size can be larger than the number of bytes originally requested when the block was allocated.

Return Value

_msize returns the size of the block in bytes.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
			+			+

normvideo

See also Example Portability

Syntax

```
#include <conio.h>
void normvideo(void);
```

Description

Selects normal-intensity characters.

normvideo selects normal characters by returning the text attribute (foreground and background) to the value it had when the program started.

This function does not affect any characters currently on the screen, only those displayed by functions (such as *cprintf*) performing direct console output functions after *normvideo* is called.

Note: Do not use this function for Win32s or Win32 GUI applications.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+			+			+

offsetof

Example Portability

Syntax

```
#include <stddef.h>
size_t offsetof(struct_type, struct_member);
```

Description

Gets the byte offset to a structure member.

offsetof is available only as a macro. The argument *struct_type* is a **struct** type. *struct_member* is any element of the **struct** that can be accessed through the member selection operators or pointers.

If *struct_member* is a bit field, the result is undefined.

See also Chapter 2 in the *Programmer's Guide* for a discussion of the **sizeof** operator, memory allocation, and alignment of structures.

Return Value

offsetof returns the number of bytes from the start of the structure to the start of the named structure member.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

open, _wopen

See also   Example   Portability

Syntax

```
#include <fcntl.h>
#include<io.h>
int open(const char *path, int access [, unsigned mode]);
int _wopen(const wchar_t *path, int access [, unsigned mode]);
```

Description

Opens a file for reading or writing.

open opens the file specified by *path*, then prepares it for reading and/or writing as determined by the value of *access*.

To create a file in a particular mode, you can either assign to the global variable *_fmode* or call *open* with the O_CREAT and O_TRUNC options ORed with the translation mode desired.

For example, the call

```
open("XMP",O_CREAT|O_TRUNC|O_BINARY,S_IREAD)
```

creates a binary-mode, read-only file named XMP, truncating its length to 0 bytes if it already existed.

For *open*, *access* is constructed by bitwise ORing flags from the following lists. Only one flag from the first list can be used (and one *must* be used); the remaining flags can be used in any logical combination.

These symbolic constants are defined in fcntl.h.

Read/Write Flags

O_RDONLY	Open for reading only.
O_WRONLY	Open for writing only.
O_RDWR	Open for reading and writing.

Other Access Flags

O_NDELAY	Not used; for UNIX compatibility.
O_APPEND	If set, the file pointer will be set to the end of the file prior to each write.
O_CREAT	If the file exists, this flag has no effect. If the file does not exist, the file is created, and the bits of *mode* are used to set the file attribute bits as in *chmod*.
O_TRUNC	If the file exists, its length is truncated to 0. The file attributes remain unchanged.
O_EXCL	Used only with O_CREAT. If the file already exists, an error is returned.
O_BINARY	Can be given to explicitly open the file in binary mode.
O_TEXT	Can be given to explicitly open the file in text mode.

If neither O_BINARY nor O_TEXT is given, the file is opened in the translation mode set by the global variable *_fmode*.

If the O_CREAT flag is used in constructing access, you need to supply the *mode* argument to *open* from the following symbolic constants defined in sys\stat.h.

Value Of Mode	Access Permission
S_IWRITE	Permission to write
S_IREAD	Permission to read
S_IREAD|S_IWRITE	Permission to read and write

Return Value

On success, *open* returns a nonnegative integer (the file handle). The file pointer, which marks the

current position in the file, is set to the beginning of the file.

On error, *open* returns -1 and the global variable errno is set to one of the following values:

EACCES	Permission denied
EINVACC	Invalid access code
EMFILE	Too many open files
ENOENT	No such file or directory

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

opendir, wopendir

See also Example Portability

Syntax

```
#include <dirent.h>
DIR *opendir(char *dirname);
wDIR *wopendir(const wchar_t *dirname);
```

Description

Opens a directory stream for reading.

opendir is available on POSIX-compliant UNIX systems.

The *opendir* function opens a directory stream for reading. The name of the directory to read is *dirname*. The stream is set to read the first entry in the directory.

A directory stream is represented by the *DIR* structure, defined in dirent.h. This structure contains no user-accessible fields. Multiple directory streams can be opened and read simultaneously. Directory entries can be created or deleted while a directory stream is being read.

Use the *readdir* function to read successive entries from a directory stream. Use the *closedir* function to remove a directory stream when it is no longer needed.

Return Value

On success, *opendir* returns a pointer to a directory stream that can be used in calls to readdir, rewinddir, and closedir.

On errror (If the directory cannot be opened), it returns NULL and sets the global variable errno to

ENOENT	The directory does not exist
ENOMEM	Not enough memory to allocate a DIR object

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

_open_osfhandle

See also Example Portability

Syntax

```
#include <io.h>
int _open_osfhandle(long osfhandle, int flags);
```

Description

Associates file handles.

The *_open_osfhandle* function allocates a run-time file handle and sets it to point to the operating system file handle specified by *osfhandle*.

The value flags is a bitwise OR combination of one or more of the following manifest constants (defined in fcntl.h):

O_APPEND	Repositions the file pointer to the end of the file before every write operation.
O_RDONLY	Opens the file for reading only.
O_TEXT	Opens the file in text (translated) mode.

Return Value

On success, _open_osfhandle returns a C run-time file handle. Otherwise, it returns -1.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
			+			

_pclose

See also   Example   Portability

Syntax

```
#include <stdio.h>
int _pclose(FILE * stream);
```

Description

Waits for piped command to complete.

_pclose closes a pipe stream created by a previous call to _popen, and then waits for the associated child command to complete.

Return Value

On success, *_pclose* returns the termination status of the child command. This is the same value as the termination status returned by cwait, except that the high and low order bytes of the low word are swapped.

On error, it returns -1.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
			+			+

perror, _wperror

See also Example Portability

Syntax

```
#include <stdio.h>
void perror(const char *s);
void _wperror(const wchar_t *s);
```

Description

Prints a system error message.

perror prints to the *stderr* stream (normally the console) the system error message for the last library routine that set the global variable errno.

It prints the argument *s* followed by a colon (:) and the message corresponding to the current value of the global variable *errno* and finally a new line. The convention is to pass the file name of the program as the argument string.

The array of error message strings is accessed through the global variable _sys_errlist. The global variable *errno* can be used as an index into the array to find the string corresponding to the error number. None of the strings include a newline character.

The global variable _sys_nerr contains the number of entries in the array.

The following messages are generated by *perror*:

Win 16 and Win 32 messages

Arg list too big

Attempted to remove current directory

Bad address

Bad file number

Block device required

Broken pipe

Cross-device link

Error 0

Exec format error

Executable file in use

File already exists

File too large

Illegal seek

Inappropriate I/O control operation

Input/output error

Interrupted function call

Invalid access code

Invalid argument Resource busy

Invalid dataResource temporarily unavailable

Invalid environment

Invalid format

Invalid function number

Invalid memory block address

Is a directory

Math argument

Memory arena trashed

Name too long
No child processes
No more files
No space left on device
No such device
No such device or address
No such file or directory
No such process
Not a directory
Not enough memory
Not same device
Operation not permitted
Path not found
Permission denied
Possible deadlock
Read-only file system
Resource busy
Resource temporarily unavailable
Result too large
Too many links
Too many open files

Win 32 only messages

Note: For Win32s or Win32 GUI applications, stderr must be redirected.

Bad address
Block device required
Broken pipe
Executable file in use
File too large
Illegal seek
Inappropriate I/O control
Input/output error
Interrupted function call
Is a directory
Name too long
No child processes
No space left on device
No such device or address
No such process
Not a directory
Operation not permitted
Possible deadlock
Read-only file system
Resource busy
Resource temporarily unavailable
Too many links

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+		+	+	+	+

_pipe

See also   Example   Portability

Syntax

```
#include <fcntl.h>
#include<io.h>
int _pipe(int *handles, unsigned int size, int mode);
```

Description

Creates a read/write pipe.

The *_pipe* function creates an anonymous pipe that can be used to pass information between processes. The pipe is opened for both reading and writing. Like a disk file, a pipe can be read from and written to, but it does not have a name or permanent storage associated with it; data written to and from the pipe exist only in a memory buffer managed by the operating system.

The read handle is returned to *handles*[0], and the write handle is returned to *handles*[1]. The program can use these handles in subsequent calls to read, write, dup, dup2, or close. When all pipe handles are closed, the pipe is destroyed.

The size of the internal pipe buffer is *size*. A recommended minimum value is 512 bytes.

The translation mode is specified by *mode*, as follows:

O_BINARY	The pipe is opened in binary mode
O_TEXT	The pipe is opened in text mode

If *mode* is zero, the translation mode is determined by the external variable *_fmode*.

Return Value

On success, *_pipe* returns 0 and returns the pipe handles to *handles*[0] and *handles*[1].

On error, it returns -1 and sets errno to one of the following values:

EMFILE	Too many open files
ENOMEM	Out of memory

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
			+			+

poly, polyl

Example Portability

Syntax

```
#include <math.h>
double poly(double x, int degree, double coeffs[]);
long double polyl(long double x, int degree, long double coeffs[]);
```

Description

Generates a polynomial from arguments.

poly generates a polynomial in *x*, of degree *degree*, with coefficients *coeffs[0]*, *coeffs[1], ...*, *coeffs[degree]*. For example, if *n* = 4, the generated polynomial is:

$$coeffs[4]x^4 + coeffs[3]x^3 + coeffs[2]x^2 + coeffs[1]x + coeffs[0]$$

polyl is the **long double** version; it takes **long double** arguments and returns a **long double** result.

Return Value

poly and *polyl* return the value of the polynomial as evaluated for the given *x*.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
poly	+	+	+	+			+
polyl	+		+	+			+

_popen, _wpopen

See also Example Portability

Syntax

```
#include <stdio.h>
FILE *_popen (const char *command, const char *mode);
FILE *_wpopen (const wchar_t *command, const wchar_t *mode);
```

Description

Creates a command processor pipe.

The *_popen* function creates a pipe to the command processor. The command processor is executed asynchronously, and is passed the command line in *command*. The *mode* string specifies whether the pipe is connected to the command processor's standard input or output, and whether the pipe is to be opened in binary or text mode.

The *mode* string can take one of the following values:

Value	Description
rt	Read child command's standard output (text).
rb	Read child command's standard output (binary).
wt	Write to child command's standard input (text).
wb	Write to child command's standard input (binary).

The terminating *t* or *b* is optional; if missing, the translation mode is determined by the external variable *_fmode*.

Use the _pclose function to close the pipe and obtain the return code of the command.

Return Value

On success, *_popen* returns a FILE pointer that can be used to read the standard output of the command, or to write to the standard input of the command, depending on the *mode* string.

On error, it returns NULL.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
			+			+

pow, powl

See also Example Portability

Syntax

```
#include <math.h>
double pow(double x, double y);
long double powl(long double x, long double y);
```

Description

Calculates *x* to the power of *y*.

powl is the **long double** version; it takes **long double** arguments and returns a **long double** result.

Return Value

On success, *pow* and powl return the value calculated of *x* to the power of *y*.

Sometimes the arguments passed to these functions produce results that overflow or are incalculable. When the correct value would overflow, the functions return the value HUGE_VAL (*pow*) or _LHUGE_VAL (*powl*). Results of excessively large magnitude can cause the global variable errno to be set to

ERANGE	Result out of range

If the argument *x* passed to *pow* or *powl* is real and less than 0, and *y* is not a whole number, or you call *pow(0,0)*, the global variable *errno* is set to

EDOM	Domain error

Error handling for these functions can be modified through the functions _matherr and _matherrl.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
pow	+	+	+	+	+	+	+
powl	+		+	+			+

pow10, pow10l

See also Example Portability

Syntax

```
#include <math.h>
double pow10(int p);
long double pow10l(int p);
```

Description

Calculates 10 to the power of *p*.

pow10l is the **long double** version; it takes **long double** arguments and returns a **long double** result.

Return Value

On success, *pow10* returns the value calculated, 10 to the power of *p* and *pow10l* returns a **long double** result.

The result is actually calculated to **long double** accuracy. All arguments are valid, although some can cause an underflow or overflow.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
pow10	+	+	+	+			+
pow10l	+		+	+			+

printf, wprintf

See also   Example   Portability

Syntax

```
#include <stdio.h>
int printf(const char *format[, argument, ...]);
int wprintf(const wchar_t *format[, argument, ...]);
```

Description

Writes formatted output to stdout.

The *printf* function:

- Accepts a series of arguments
- Appllies to each argument a format specifier contained in the format string *format
- Outputs the formatted data (to the screen, a stream, *stdout*, or a string)

There must be enough arguments for the format. If there are not, the results will be unpredictable and likely disastrous. Excess arguments (more than required by the format) are merely ignored.

Note: For Win32s or Win32 GUI applications, *stdout* must be redirected.

Return Value

On success, *printf* returns the number of bytes output.

On error, *printf* returns EOF.

More About printf

Unicode output format specifiers

Format String

Format Specifiers

Format Specifier Conventions

Flag Characters

Input-size Modifiers

Precision Specifiers

Type Characters

Width Specifiers

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+		+	+	+	+

printf Format String

See also

The format string, present in each of the *printf* function calls, controls how each function will convert, format, and print its arguments.

Note: There must be enough arguments for the format; if not, the results will be unpredictable and possibly disastrous. Excess arguments (more than required by the format) are ignored.

The format string is a character string that contains two types of objects:

- Plain characters are copied verbatim to the output stream.
- Conversion specifications fetch arguments from the argument list and apply formatting to them.

Plain characters are simply copied verbatim to the output stream.

Conversion specifications fetch arguments from the argument list and apply formatting to them.

printf Format Specifiers

See also

print format specifiers have the following form

`% [flags] [width] [.prec] [F|N|h|l|L] type_char`

Each format specifier begins with the percent character (`%`).

After the `%` come the following optional specifiers, in this order:

Optional Format String Components

These are the general aspects of output formatting controlled by the optional characters, specifiers, and modifiers in the format string:

Component	Optional/Required	What it Controls or Specifies
`[flags]`	(Optional)	Flag character(s) Output justification, numeric signs, decimal points, trailing zeros, octal and hex prefixes
`[width]`	(Optional)	Width specifier Minimum number of characters to print, padding with blanks or zeros
`[prec]`	(Optional)	Precision specifier Maximum number of characters to print; for integers, minimum number of digits to print
`[F|N|h|l|L]`	(Optional)	Input size modifier Override default size of next input argument: **N** = **near** pointer **F** = **far** pointer **h** = **short int** **l** = **long** **L** = **long double**
`type_char`	(Required)	Conversion-type character

printf Flag characters

See also

They can appear in any order and combination.

Flag	What it means
-	Left-justifies the result, pads on the right with blanks. If not given, it right-justifies the result, pads on the left with zeros or blanks.
+	Signed conversion results always begin with a plus (+) or minus (-) sign.
blank	If value is nonnegative, the output begins with a blank instead of a plus; negative values still begin with a minus.
#	Specifies that *arg* is to be converted using an alternate form.

Note: Plus (+) takes precedence over blank () if both are given.

Alternate Forms for printf Conversion

See also

If you use the # flag conversion character, it has the following effect on the argument (*arg*) being converted:

Conversion character	How # affects the argument
c s d iu	No effect.
0	0 is prepended to a nonzero *arg*.
x X	0x (or 0X) is prepended to *arg*.
e E f	The result always contains a decimal point even if no digits follow the point. Normally, a decimal point appears in these results only if a digit follows it.
g G	Same as **e** and **E**, except that trailing zeros are not removed.

printf Width Specifiers

See also

The width specifier sets the minimum field width for an output value.

Width is specified in one of two ways:

- directly, through a decimal digit string
- indirectly, through an asterisk (*)

If you use an asterisk for the width specifier, the next argument in the call (which must be an **int**) specifies the minimum output field width.

Nonexistent or small field widths do *not* cause truncation of a field. If the result of a conversion is wider than the field width, the field is expanded to contain the conversion result.

Width specifier	How output width is affected
n	At least *n* characters are printed. If the output value has less than *n* characters, the output is padded with blanks (right-padded if - flag given, left-padded otherwise).
0*n*	At least *n* characters are printed. If the output value has less than *n* characters, it is filled on the left with zeros.
*	The argument list supplies the width specifier, which must precede the actual argument being formatted.

printf Precision Specifiers

See also

The *printf* precision specifiers set the maximum number of characters (or minimum number of integer digits) to print.

A *printf* precision specification always begins with a period (.) to separate it from any preceding width specifier.

Then, like `[width]`, precision is specified in one of two ways:

- directly, through a decimal digit string
- indirectly, through an asterisk (`*`)

If you use an `*` for the precision specifier, the next argument in the call (treated as an **int**) specifies the precision.

If you use asterisks for the width or the precision, or for both, the width argument must immediately follow the specifiers, followed by the precision argument, then the argument for the data to be converted.

`[.prec]`	**How Output Precision Is Affected**
(none)	Precision set to default: = 1 for *d,i,o,u,x,X* types = 6 for *e,E,f* types = All significant digits for *g,G* types = Print to first null character for *s* types = No effect on *c* types
`.0`	For *d,i,o,u,x* types, precision set to defaul for *e,E,f* types, no decimal point is printed.
`.`*n*	*n* characters or *n* decimal places are printed. If the output value has more than *n* characters, the output might be truncated or rounded. (Whether this happens depends on the type character.)
.	The argument list supplies the precision specifier, which must precede the actual argument being formatted.

No numeric characters will be output for a field (i.e., the field will be blank) if the following conditions are all met:

- you specify an explicit precision of 0
- the format specifier for the field is one of the integer formats (`d, i, o, u,` or `x`)
- the value to be printed is 0

How [.prec] Affects Conversion

Char Type	**Effect of `[.prec]` (.n) on Conversion**

`d`
`i`
`o`
`u`
`x`
`X`
	Specifies that at least *n* digits are printed.
	If input argument has less than *n* digits,
	output value is left-padded x with zeros.
	If input argument has more than *n* digits,
	the output value is not truncated.

`e` `E`	Specifies that *n* characters are printed after the decimal point, and

f	the last digit printed is rounded.
g G	Specifies that at most *n* significant digits are printed.
c	Has no effect on the output.
s	Specifies that no more than *n* characters are printed.

printf Conversion-Type Characters

See also

The information in this table is based on the assumption that no flag characters, width specifiers, precision specifiers, or input-size modifiers were included in the format specifier.

Note: Certain conventions accompany some of these format specifiers.

Type Char	Expected Input	Format of output
Numerics		
`d`	Integer	**signed** decimal **integer**
`i`	Integer	**signed** decimal **integer**
`o`	Integer	**unsigned** octal **integer**
`u`	Integer	**unsigned** decimal **integer**
`x`	Integer	**unsigned** hexadecimal **int** (with **a, b, c, d, e, f**)
`X`	Integer	**unsigned** hexadecimal **int** (with **A, B, C, D, E, F**)
`f`	Floating point	**signed** value of the form `[-]dddd.dddd`.
`e`	Floating point	**signed** value of the form `[-]d.dddd or` **e**`[+/-]ddd`
`g`	Floating point	**signed** value in either **e** or **f** form, based on given value and precision. Trailing zeros and the decimal point are printed if necessary.
`E`	Floating point	Same as **e**; with **E** for exponent.
`G`	Floating point	Same as **g;** with **E** for exponent if **e** format used
Characters		
`c`	Character	Single character
`s`	String pointer	Prints characters until a null-terminator is pressed or precision is reached
`%`	None	Prints the % character
Pointers		
`n`	Pointer to **int**	Stores (in the location pointed to by the input argument) a count of the chars written so far.
`p`	Pointer	Prints the input argument as a pointer; format depends on which memory model was used. It will be either XXXX:YYYY or YYYY (offset only).

Infinite floating-point numbers are printed as `+INF` and `-INF`.

An IEEE Not-A-Number is printed as `+NAN` or `-NAN`.

printf Input-size Modifiers

See also

These modifiers determine how printf functions interpret the next input argument, `arg[f]`.

Modifier	Type of arg	arg is interpreted as ...
F	Pointer (`p`, `s`,	A **far** pointer
N	and `n`)	A **near** pointer (**Note:** *N* can't be used with any conversion in **huge** model.)
h	*d i o u x X*	A **short int**
l	*d i o u x X*	A **long int**
	e E f g G	A **double**
L	*e E f g G*	A **long double**

These modifiers affect how all the printf functions interpret the data type of the corresponding input argument arg.

Both *F* and *N* reinterpret the input variable *arg*. Normally, the *arg* for a `%p`, `%s`, or `%n` conversion is a pointer of the default size for the memory model.

h, l, and *L* override the default size of the numeric data input arguments. Neither *h* nor *l* affects character (`c,s`) or pointer (`p,n`) types.

printf Format Specifier Conventions

See also

Certain conventions accompany some of the printf format specifiers for the following conversions:

- %e or %E

- %f

- %g or %G

- %x or %X

Note: Infinite floating point numbers are printed as +INF and -INF. An IEEE Not-a-Number is printed as +NAN or -NAN.

%e or %E Conversions

See also

The argument is converted to match the style

`[-] d.ddd...e[+/-]ddd`

where:

- one digit precedes the decimal point
- the number of digits after the decimal point is equal to the precision.
- the exponent always contains at least two digits

%f Conversions

See also

The argument is converted to decimal notation in the style

```
[-] ddd.ddd...
```

where the number of digits after the decimal point is equal to the precision (if a non-zero precision was given).

%g or %G Conversions

See also

The argument is printed in style **e, E** or **f,** with the precision specifying the number of significant digits.

Trailing zeros are removed from the result, and a decimal point appears only if necessary.

The argument is printed in style **e** or **f** (with some restraints) if **g** is the conversion character. Style **e** is used only if the exponent that results from the conversion is either greater than the precision or less than -4.

The argument is printed in style **E** if **G** is the conversion character.

%x or %X Conversions

See also

For **x** conversions, the letters **a, b, c, d, e**, and **f** appear in the output.

For **X** conversions, the letters **A, B, C, D, E**, and **F** appear in the output.

...printf functions

The ...printf functions include

fprintf	sends formatted output to a stream
printf	sends formatted output to stdout
sprintf	sends formatted output to a string
vfprintf	sends formatted output to a stream, using an argument list
vprintf	sends formatted output to *stdout*, using an argument list
vsprintf	sends formatted output to a string, using an argument list

putc, putwc

See also Portability

Syntax

```
#include <stdio.h>
int putc(int c, FILE *stream);
wint_t putwc(wint_t c, FILE *stream);
```

Description

Outputs a character to a stream.

putc is a macro that outputs the character *c* to the stream given by *stream*.

Return Value

On success, *putc* returns the character printed, *c.*

On error, *putc* returns EOF.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

putch

See also Example Portability

Syntax

```
#include <conio.h>
int putch(int c);
```

Description

Outputs character to screen.

putch outputs the character *c* to the current text window. It is a text mode function performing direct video output to the console. *putch* does not translate linefeed characters (\n) into carriage-return/linefeed pairs.

The string is written either directly to screen memory or by way of a BIOS call, depending on the value of the global variable _*directvideo*.

Note: This function should not be used in Win32s or Win32 GUI applications.

Return Value

On success, *putch* returns the character printed, *c*. On error, it returns EOF.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

putchar, putwchar

See also   Example   Portability

Syntax

```
#include <stdio.h>
int putchar(int c);
wint_t putwchar(wint_t c);
```

Description

putchar(c) is a macro defined to be *putc*(c, stdout).

Note: For Win32s or Win32 GUI applications, *stdout* must be redirected.

Return Value

On success, *putchar* returns the character *c*. On error, *putchar* returns EOF.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

putenv, _wputenv

See also   Example   Portability

Syntax

```
#include <stdlib.h>
int putenv(const char *name);
int _wputenv(const wchar_t *name);
```

Description

Adds string to current environment.

putenv accepts the string *name* and adds it to the environment of the current process. For example,

```
putenv("PATH=C:\\BC");
```

putenv can also be used to modify an existing *name*. On DOS and OS/2, *name* must be uppercase. On other systems, *name* can be either uppercase or lowercase. *name* must not include the equal sign (**=**). You can set a variable to an empty value by specifying an empty string on the right side of the '=' sign.

putenv can be used only to modify the current program's _environment. Once the program ends, the old _environment is restored. The _environment of the current process is passed to child processes, including any changes made by *putenv*.

Note that the string given to *putenv* must be static or global. Unpredictable results will occur if a local or dynamic string given to *putenv* is used after the string memory is released.

Return Value

On success, *putenv* returns 0; on failure, -1.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

puts, _putws

See also Example Portability

Syntax

```
#include <stdio.h>
int puts(const char *s);
int _putws(const wchar_t *s);
```

Description

Outputs a string to stdout.

puts copies the null-terminated string *s* to the standard output stream *stdout* and appends a newline character.

Note: For Win32s or Win32 GUI applications, *stdout* must be redirected.

Return Value

On successful completion, *puts* returns a nonnegative value. Otherwise, it returns a value of EOF.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+		+

puttext

See also Example Portability

Syntax

```
#include <conio.h>
int puttext(int left, int top, int right, int bottom, void *source);
```

Description

Copies text from memory to the text mode screen.

puttext writes the contents of the memory area pointed to by *source* out to the onscreen rectangle defined by *left, top, right*, and *bottom*.

All coordinates are absolute screen coordinates, not window-relative. The upper left corner is (1,1).

puttext places the contents of a memory area into the defined rectangle sequentially from left to right and top to bottom.

Each position onscreen takes 2 bytes of memory: The first byte is the character in the cell, and the second is the cell's video attribute. The space required for a rectangle *w* columns wide by *h* rows high is defined as

bytes = (*h* rows) x (*w* columns) x 2

puttext is a text mode function performing direct video output.

Note: This function should not be used in Win32s or Win32 GUI applications.

Return Value

puttext returns a nonzero value if the operation succeeds; it returns 0 if it fails (for example, if you gave coordinates outside the range of the current screen mode).

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+			+			+

_putw

See also Example Portability

Syntax

```
#include <stdio.h>
int _putw(int w, FILE *stream);
```

Description

Writes an integer on a stream.

_putw outputs the integer *w* to the given stream. *_putw* neither expects nor causes special alignment in the file.

Return Value

On success, *_putw* returns the integer *w*. On error, *_putw* returns EOF. Because EOF is a legitimate integer, use *ferror* to detect errors with *_putw*.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

qsort

See also   Example   Portability

Syntax

```
#include <stdlib.h>
void qsort(void *base, size_t nelem, size_t width, int (_USERENTRY *fcmp)
  (const void *, const void *));
```

Description

Sorts using the quicksort algorithm.

qsort is an implementation of the "median of three" variant of the quicksort algorithm. *qsort* sorts the entries in a table by repeatedly calling the user-defined comparison function pointed to by *fcmp*.

- *base* points to the base (0th element) of the table to be sorted.
- *nelem* is the number of entries in the table.
- *width* is the size of each entry in the table, in bytes.

fcmp, the comparison function, must be used with the _USERENTRY calling convention.

fcmp accepts two arguments, *elem1* and *elem2*, each a pointer to an entry in the table. The comparison function compares each of the pointed-to items (*elem1 and *elem2), and returns an integer based on the result of the comparison.

- *elem1 < *elem2   fcmp returns an integer < 0
- *elem1 == *elem2   fcmp returns 0
- *elem1 > *elem2   fcmp returns an integer > 0

In the comparison, the less-than symbol (<) means the left element should appear before the right element in the final, sorted sequence. Similarly, the greater-than (>) symbol means the left element should appear after the right element in the final, sorted sequence.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

raise

See also Example Portability

Syntax

```
#include <signal.h>
int raise(int sig);
```

Description

Sends a software signal to the executing program.

raise sends a signal of type *sig* to the program. If the program has installed a signal handler for the signal type specified by *sig*, that handler will be executed. If no handler has been installed, the default action for that signal type will be taken.

The signal types currently defined in signal.h are noted here:

Signal	Description
SIGABRT	Abnormal termination
SIGFPE	Bad floating-point operation
SIGILL	Illegal instruction
SIGINT	Ctrl-C interrupt
SIGSEGV	Invalid access to storage
SIGTERM	Request for program termination
SIGUSR1	User-defined signal
SIGUSR2	User-defined signal
SIGUSR3	User-defined signal
SIGBREAK	Ctrl-Break interrupt

Note: SIGABRT isn't generated by C++Builder during normal operation. It can, however, be generated by abort, raise, or unhandled exceptions.

Return Value

On succes, *raise* returns 0.

On error it returns nonzero.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

rand

See also Example Portability

Syntax

```
#include <stdlib.h>
int rand(void);
```

Description

Random number generator.

rand uses a multiplicative congruential random number generator with period 2 to the 32nd power to return successive pseudorandom numbers in the range from 0 to RAND_MAX. The symbolic constant RAND_MAX is defined in stdlib.h.

Return Value

rand returns the generated pseudorandom number.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

random

See also　　Example　　Portability

Syntax

```
#include <stdlib.h>
int random(int num);
```

Description

Random number generator.

random returns a random number between 0 and (*num*-1). *random(num)* is a macro defined in stdlib.h. Both *num* and the random number returned are integers.

Return Value

random returns a number between 0 and (*num*-1).

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

randomize

See also Example Portability

Syntax

```
#include <stdlib.h>
#include <time.h>
void randomize(void);
```

Description

Initializes random number generator.

randomize initializes the random number generator with a random value.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

read

See also   Example   Portability

Syntax

```
#include <io.h>
int read(int handle, void *buf, unsigned len);
```

Description

Reads from file.

read attempts to read *len* bytes from the file associated with *handle* into the buffer pointed to by *buf*.

For a file opened in text mode, read removes carriage returns and reports end-of-file when it reaches a Ctrl-Z.

The file handle *handle* is obtained from a *creat, open, dup*, or *dup2* call.

On disk files, *read* begins reading at the current file pointer. When the reading is complete, it increments the file pointer by the number of bytes read. On devices, the bytes are read directly from the device.

The maximum number of bytes that *read* can read is UINT_MAX -1, because UINT_MAX is the same as -1, the error return indicator. UINT_MAX is defined in limits.h.

Return Value

On successful completion, *read* returns an integer indicating the number of bytes placed in the buffer. If the file was opened in text mode, *read* does not count carriage returns or Ctrl-Z characters in the number of bytes read.

On end-of-file, *read* returns 0. On error, *read* returns -1 and sets the global variable errno to one of the following values:

EACCES	Permission denied
EBADF	Bad file number

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

readdir, wreaddir

See also Example Portability

Syntax

```
#include <dirent.h>
struct dirent *readdir(DIR *dirp);
struct wdirent *wreaddir(wDIR *dirp)
```

Description

Reads the current entry from a directory stream.

readdir is available on POSIX-compliant UNIX systems.

The *readdir* function reads the current directory entry in the directory stream pointed to by *dirp*. The directory stream is advanced to the next entry.

The *readdir* function returns a pointer to a **dirent** structure that is overwritten by each call to the function on the same directory stream. The structure is not overwritten by a *readdir* call on a different directory stream.

The **dirent** structure corresponds to a single directory entry. It is defined in dirent.h and contains (in addition to other non-accessible members) the following member:

```
   char   d_name[];
```

where *d_name* is an array of characters containing the null-terminated file name for the current directory entry. The size of the array is indeterminate; use *strlen* to determine the length of the file name.

All valid directory entries are returned, including subdirectories, "." and ".." entries, system files, hidden files, and volume labels. Unused or deleted directory entries are skipped.

A directory entry can be created or deleted while a directory stream is being read, but *readdir* might or might not return the affected directory entry. Rewinding the directory with *rewinddir* or reopening it with *opendir* ensures that *readdir* will reflect the current state of the directory.

The *wreaddir* function is the Unicode version of *readdir*. It uses the **wdirent** structure but otherwise is similar to *readdir*.

Return Value

On success, *readdir* returns a pointer to the current directory entry for the directory stream.

If the end of the directory has been reached, or *dirp* does not refer to an open directory stream, *readdir* returns NULL.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

realloc

See also Example Portability

Syntax

```
#include <stdlib.h>
void *realloc(void *block, size_t size);
```

Description

Reallocates main memory.

realloc attempts to shrink or expand the previously allocated block to *size* bytes. If *size* is zero, the memory block is freed and NULL is returned. The *block* argument points to a memory block previously obtained by calling *malloc, calloc*, or *realloc*. If *block* is a NULL pointer, *realloc* works just like *malloc*.

realloc adjusts the size of the allocated block to *size*, copying the contents to a new location if necessary.

Return Value

realloc returns the address of the reallocated block, which can be different than the address of the original block.

If the block cannot be reallocated, *realloc* returns NULL.

If the value of *size* is 0, the memory block is freed and *realloc* returns NULL.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

remove, _wremove

See also Example Portability

Syntax

```
#include <stdio.h>
int remove(const char *filename);
int _wremove(const wchar_t *filename);
```

Description

Removes a file.

remove deletes the file specified by *filename*. It is a macro that simply translates its call to a call to *unlink*. If your file is open, be sure to close it before removing it.

The *filename* string can include a full path.

Return Value

On successful completion, *remove* returns 0. On error, it returns -1, and the global variable errno is set to one of the following values:

EACCES	Permission denied
ENOENT	No such file or directory

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

rename, _wrename

Example Portability

Syntax

```
#include <stdio.h>
int rename(const char *oldname, const char *newname);
int _wrename(const wchar_t *oldname, const wchar_t *newname);
```

Description

Renames a file.

rename changes the name of a file from *oldname* to *newname*. If a drive specifier is given in *newname*, the specifier must be the same as that given in *oldname*.

Directories in *oldname* and *newname* need not be the same, so *rename* can be used to move a file from one directory to another. Wildcards are not allowed.

This function will fail (EEXIST) if either file is currently open in any process.

Return Value

On success, *rename* returns 0.

On error (if the file cannot be renamed), it returns -1 and the global variable errno is set to one of the following values:

EEXIST	Permission denied: file already exists.
ENOENT	No such file or directory
ENOTSAM	Not same device

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+	+	+	+

rewind

See also   Example   Portability

Syntax

```
#include <stdio.h>
void rewind(FILE *stream);
```

Description

Repositions a file pointer to the beginning of a stream.

rewind(stream) is equivalent to fseek(stream, 0L, SEEK_SET), except that *rewind* clears the end-of-file and error indicators, while *fseek* clears the end-of-file indicator only.

After *rewind*, the next operation on an update file can be either input or output.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

rewinddir, wrewinddir

See also Example Portability

Syntax

```
#include <dirent.h>
void rewinddir(DIR *dirp);
void wrewinddir(wDIR *dirp);
```

Description

Resets a directory stream to the first entry.

rewinddir is available on POSIX-compliant UNIX systems.

The *rewinddir* function repositions the directory stream *dirp* at the first entry in the directory. It also ensures that the directory stream accurately reflects any directory entries that might have been created or deleted since the last *opendir* or *rewinddir* on that directory stream.

wrewinddir is the Unicode version of *rewinddir*.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

_rmdir, _wrmdir

See also Example Portability

Syntax

```
#include <dir.h>
int _rmdir(const char *path);
int _wrmdir(const wchar_t *path);
```

Description

Removes a directory.

_rmdir deletes the directory whose path is given by *path*. The directory named by *path*

- must be empty
- must not be the current working directory
- must not be the root directory

Return Value

_rmdir returns 0 if the directory is successfully deleted. A return value of -1 indicates an error, and the global variable errno is set to one of the following values:

EACCES	Permission denied
ENOENT	Path or file function not found

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

_rmtmp

See also   Example   Portability

Syntax

```
#include <stdio.h>
int _rmtmp(void);
```

Description

Removes temporary files.

The _*rmtmp* function closes and deletes all open temporary file streams which were previously created with *tmpfile*. The current directory must the same as when the files were created, or the files will not be deleted.

Return Value

_*rmtmp* returns the total number of temporary files it closed and deleted.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

_rotl, _rotr

See also Example Portability

Syntax

```
#include <stdlib.h>
unsigned short _rotl(unsigned short value, int count);
unsigned short _rotr(unsigned short value, int count);
```

Description

Bit-rotates an **unsigned** short integer value to the left or right.

_rotl rotates the given *value* to the left *count* bits.

_rotr rotates the given *value* to the right *count* bits.

Return Value

_rotl, and *_rotr* return the rotated integer:

- *_rotl* returns the value of *value* left-rotated *count* bits.
- *_rotr* returns the value of *value* right-rotated *count* bits.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

_rtl_chmod, _wrtl_chmod

See also Example Portability

Syntax

```
#include <io.h>
int _rtl_chmod(const char *path, int func [, int attrib]);
int _wrtl_chmod(const wchar_t *path, int func, ... );
```

Description

Gets or sets file attributes.

Note: The *_rtl_chmod* function replaces *_chmod* which is obsolete

_rtl_chmod can either fetch or set file attributes. If *func* is 0, *_rtl_chmod* returns the current attributes for the file. If *func* is 1, the attribute is set to *attrib*.

attrib can be one of the following symbolic constants (defined in dos.h):

FA_RDONLY	Read-only attribute
FA_HIDDEN	Hidden file
FA_SYSTEM	System file
FA_LABEL	Volume label
FA_DIREC	Directory
FA_ARCH	Archive

Return Value

On success, *_rtl_chmod* returns the file attribute word.

On error, it returns a value of -1 and sets the global variable errno to one of the following values:

ENOENT	Path or filename not found
EACCES	Permission denied

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			

_rtl_close

See also Example Portability

Syntax

```
#include <io.h>
int _rtl_close(int handle);
```

Description

Closes a file.

Note: This function replaces *_close* which is obsolete

The *_rtl_close* function closes the file associated with *handle*, a file handle obtained from a call to *creat, creatnew, creattemp, dup, dup2, open, _rtl_creat*, or *_rtl_open*.

It does not write a Ctrl-Z character at the end of the file. If you want to terminate the file with a Ctrl-Z, you must explicitly output one.

Return Value

On success, *_rtl_close* returns 0.

On error (if it fails because *handle* is not the handle of a valid, open file), *_rtl_close* returns a value of -1 and the global variable errno is set to

EBADF	Bad file number

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			

_rtl_creat, _wrtl_creat

See also Example Portability

Syntax

```
#include <io.h>
int _rtl_creat(const char *path, int attrib);
int _wrtl_creat(const wchar_t *path, int attrib);
```

Description

Creates a new file or overwrites an existing one.

Note: The *_rtl_creat* function replaces *_creat* which is obsolete

_rtl_creat opens the file specified by *path*. The file is always opened in binary mode. Upon successful file creation, the file pointer is set to the beginning of the file. The file is opened for both reading and writing.

If the file already exists its size is reset to 0. (This is essentially the same as deleting the file and creating a new file with the same name.)

The *attrib* argument is an ORed combination of one or more of the following constants (defined in dos.h):

FA_RDONLY	Read-only attribute
FA_HIDDEN	Hidden file
FA_SYSTEM	System file

Return Value

On success, *_rtl_creat* returns the new file handle (a non-negative integer).

On error, it returns -1 and sets the global variable errno to one of the following values:

EACCES	Permission denied
EMFILE	Too many open files
ENOENT	Path or file name not found

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			

_rtl_heapwalk

See also Example Portability

Syntax

```
#include <malloc.h>
int _rtl_heapwalk(_HEAPINFO *hi);
```

Description

Inspects the heap node by node.

Note: This function replaces *_heapwalk* which is obsolete.

_rtl_heapwalk assumes the heap is correct. Use _heapchk to verify the heap before using *_rtl_heapwalk.* _HEAPOK is returned with the last block on the heap. _HEAPEND will be returned on the next call to *_rtl_heapwalk*.

_rtl_heapwalk receives a pointer to a structure of type _HEAPINFO (declared in malloc.h).

For the first call to *_rtl_heapwalk*, set the *hi._pentry* field to NULL. *_rtl_heapwalk* returns with *hi._pentry* containing the address of the first block.

hi._size	holds the size of the block in bytes.
hi._useflag	is a flag that is set to _USEDENTRY if the block is currently in use. If the block is free, *hi._useflag* is set to _FREEENTRY.

Return Value

This function returns one of the following values:

_HEAPBADNODE	A corrupted heap block has been found
_HEAPBADPTR	The _pentry field does not point to a valid heap block
_HEAPEMPTY	No heap exists
_HEAPEND	The end of the heap has been reached
_HEAPOK	The _heapinfo block contains valid information about the next heap block

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
			+			+

_rtl_open, _wrtl_open

See also Example Portability

Syntax

```
#include <io.h>
int _rtl_open(const char *filename, int oflags);
int _wrtl_open(const wchar_t *path, int oflags);
```

Description

Opens a file for reading or writing.

Note: The *_rtl_open* function replaces *_open* which is obsolete.

_rtl_open opens the file specified by *filename*, then prepares it for reading or writing, as determined by the value of *oflags*. The file is always opened in binary mode.

oflags uses the flags from the following two lists. Only one flag from List 1 can be used (and one *must* be used) and the flags in List 2 can be used in any logical combination.

List 1: Read/write flags

O_RDONLY	Open for reading.
O_WRONLY	Open for writing.
O_RDWR	Open for reading and writing.

The following additional values can be included in *oflags* (using an OR operation):

List 2: Other access flags

O_NOINHERIT	The file is not passed to child programs.
SH_COMPAT	Allow other opens with SH_COMPAT. All other openings of a file with the SH_COMPAT flag must be opened using SH_COMPAT flag. You can request a file open that uses SH_COMPAT logically OR'ed with some other flag (for example, SH_COMPAT | SH_DENWR is allowed). The call will fail if the file has already been opened in any other shared mode.
SH_DENYRW	Only the current handle can have access to the file.
SH_DENWR	Allow only reads from any other open to the file.
SH_DENYRD	Allow only writes from any other open to the file.
SH_DENYNO	Allow other shared opens to the file, but not other SH_COMPAT opens.

Note: These symbolic constants are defined in fcntl.h and share.h.

Only one of the SH_DENY*xx* values can be included in a single *_rtl_open* routine. These file-sharing attributes are in addition to any locking performed on the files.

The maximum number of simultaneously open files is defined by HANDLE_MAX.

Return Value

On success:*_rtl_open* returns a non-negative integer (the file handle). The file pointer, which marks the current position in the file, is set to the beginning of the file.

On error, it returns -1 and sets the global variable errno to one of the following values:

EACCES	Permission denied
EINVACC	Invalid access code
EMFILE	Too many open files
ENOENT	Path or file not found

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			

_rtl_read

See also Example Portability

Syntax

```
#include <dos.h>
int _rtl_read(int handle, void *buf, unsigned len);
```

Description

Reads from file.

Note: This function replaces *_read* which is obsolete.

This function reads *len* bytes from the file associated with *handle* into the buffer pointed to by *buf*. When a file is opened in text mode, *_rtl_read* does not remove carriage returns.

The argument *handle* is a file handle obtained from a creat, open, dup, or dup2 call.

On disk files, *_rtl_read* begins reading at the current file pointer. When the reading is complete, it increments the file pointer by the number of bytes read. On devices, the bytes are read directly from the device.

The maximum number of bytes it can read is UINT_MAX -1 (because UINT_MAX is the same as -1, the error return indicator). UINT_MAX is defined in limits.h.

Return Value

On success, *_rtl_read* returns either

- a positive integer, indicating the number of bytes placed in the buffer
- zero, indicating end-of-file

On error, it returns -1 and sets the global variable errno to one of the following values:

EACCES	Permission denied
EBADF	Bad file number

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			

_rtl_write

See also Example Portability

Syntax

```
#include <io.h>
int _rtl_write(int handle void *buf unsigned len);
```

Description

Writes to a file.

Note: This function replaces *_write* which is obsolete.

_rtl_write attempts to write *len* bytes from the buffer pointed to by *buf* to the file associated with *handle*.

The maximum number of bytes that *_rtl_write* can write is UINT_MAX -1 (because UINT_MAX is the same as -1), which is the error return indicator for *_rtl_write*. UINT_MAX is defined in limits.h. *_rtl_write* does not translate a linefeed character (LF) to a CR/LF pair because all its files are binary files.

If the number of bytes actually written is less than that requested the condition should be considered an error and probably indicates a full disk.

For disk files, writing always proceeds from the current file pointer. On devices, bytes are directly sent to the device.

For files opened with the O_APPEND option, the file pointer is not positioned to EOF before writing the data.

Return Value

On success, *_rtl_write* returns number of bytes written.

On error, it returns -1 and sets the global variable errno to one of the following values:

EACCES	Permission denied
EBADF	Bad file number

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			

scanf, wscanf

See also Example Portability

Syntax

```
#include <stdio.h>
int scanf(const char *format[, address, ...]);
int wscanf(const wchar_t *format[, address, ...]);
```

Description

Scans and formats input from the stdin stream.

Note: For Win32s or Win32 GUI applications, stdin must be redirected.

The *scanf* function:

- scans a series of input fields one character at a time
- formats each field according to a corresponding format specifier passed in the format string **format*.
- *vsscanf* scans and formats input from a string, using an argument list

There must be one format specifier and address for each input field.

scanf might stop scanning a particular field before it reaches the normal end-of-field (whitespace) character, or it might terminate entirely. For details about why this might happen, see When ...scanf Stops Scanning.

Warning: *scanf* often leads to unexpected results if you diverge from an expected pattern. You must provide information that tells scanf how to synchronize at the end of a line.

The combination of gets or fgets followed by *sscanf* is safe and easy, and therefore recommended over *scanf*.

Return Value

On success, *scanf* returns the number of input fields successfully scanned, converted, and stored. The return value does not include scanned fields that were not stored.

On error:

- if no fields were stored, *scanf* returns 0.
- if *scanf* attempts to read at end-of-file or at end-of-string, it returns EOF.

More About scanf

Unicode input format specifiers

Argument-type Modifiers

Assignment Suppression

Format Specifiers

Format Specifier Conventions

Format String

Input Fields

Pointer-size Modifiers

Type Characters

Width Specifiers

When ...scanf Functions Stop Scanning

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+		+	+	+	+

The scanf Format String

See also

The format string controls how each ...*scanf* function scans, converts, and stores its input fields.

The format string is a character string that contains three types of objects:

- *whitespace characters*
- *non-whitespace characters*
- *format specifiers*

Whitespace Characters

The whitespace characters are blank, tab (`\t`) or newline (`\n`).

If a ...*scanf* function encounters a whitespace character in the format string, it reads, but does not store, all consecutive whitespace characters up to the next non-whitespace character in the input.

Trailing whitespace is left unread (including a newline), unless explicitly matched in the format string.

Non-whitespace Characters

The non-whitespace characters are all other ASCII characters except the percent sign (%).

If a ...*scanf* function encounters a non-whitespace character in the format string, it will read, but not store, a matching non-whitespace character.

Format Specifiers

The format specifiers direct the ...*scanf* functions to read and convert characters from the input field into specific types of values, then store them in the locations given by the address arguments.

Warning: Each format specifier must have an address argument. If there are more format specs than addresses, the results are unpredictable and likely disastrous.

Excess address arguments (more than required by the format) are ignored.

scanf Format Specifiers

See also

In ...scanf format strings, format specifiers have the following form:

`% [*] [width] [F|N] [h|l|L] type_char`

Each format specifier begins with the percent character (`%`).

After the `%` come the following, in this order:

Component	Optional/Required	What It Is/Does
`[*]`	(Optional)	Assignment-suppression character. Suppresses assignment of the next input field.
`[width]`	(Optional)	Width specifier. pecifies maximum number of characters to read; fewer characters might be read if the ...*scanf* function encounters a whitespace or unconvertible character.
`[F|N]`	(Optional)	Pointer size modifier. Overrides default size of address argument: *N* = **near** pointer *F* = **far** pointer
`[h|l|L]`	(Optional)	Argument-type modifier. Overrides default type of address argument: *h* = **short int** *l* = **long int**, if `type_char` specifies integer conversion *l* = **double**, if `type_char` specifies floating-point conversion *L* = **long double**, (valid only with floating-point conversion)
`type_char`	(Required)	Type character

Type Characters

See also

The information in this table is based on the assumption that no optional characters, specifiers, or modifiers (*, width, or size) were included in the format specifier.

Note: Certain conventions accompany some of these format specifiers.

Type	Expected input	Type of argument
Numerics		
d	Decimal integer	Pointer to **int** (**int** **arg*)
D	Decimal integer	Pointer to **long** (**long** **arg*)
e,E	Floating point	Pointer to **float** (**float** **arg*)
f	Floating point	Pointer to **float** (**float** **arg*)
g,G	Floating point	Pointer to **float** (**float** **arg*)
o	Octal integer	Pointer to **int** (**int** **arg*)
O	Octal integer	Pointer to **long** (**long** **arg*)
i	Decimal, octal, or hexadecimal integer	Pointer to **int** (**int** **arg*)
I	Decimal, octal, or hexadecimal integer	Pointer to **long** (**long** **arg*)
u	Unsigned decimal integer	Pointer to **unsigned int** (**unsigned int** **arg*)
U	Unsigned decimal integer	Pointer to **unsigned long** (**unsigned long** **arg*)
x	Hexadecimal integer	Pointer to **int** (**int** **arg*)
X	Hexadecimal integer	Pointer to **int** (**int** **arg*)
Characters		
s	Character string	Pointer to array of **chars** (**char** *arg*[])
c	Character	Pointer to **char** (**char** **arg*) if a field width is given along with the *c*-type character (such as %5c) Pointer to array of *W* chars (**char** *arg[W]*)
%	% character	No conversion done; the % is stored
Pointers		
n		Pointer to **int** (**int** **arg*). The number of characters read successfully up to %n is stored in this **int**.
p	Hexadecimal form YYYY:ZZZZ or ZZZZ	Pointer to an object (**far*** or **near***) %p conversions default to the pointer size native to the memory model

Input Fields for Scanf Functions

See also

In a ...*scanf* function, any one of the following is an input field:

- all characters up to (but not including) the next whitespace character
- all characters up to the first one that can't be converted under the current format specifier (such as an 8 or 9 under octal format)
- up to *n* characters, where *n* is the specified field width

Assignment-suppression Character

See also

The assignment-suppression character is an asterisk (*), not to be confused with the C indirection (pointer) operator.

If the asterisk follows the percent sign (%) in a format specifier, the next input field will be scanned but it won't be assigned to the next address argument.

The suppressed input data is assumed to be of the type specified by the type character that follows the asterisk character.

Width Specifiers

See also

The width specifier (*n*), a decimal integer, controls the maximum number of characters to be read from the current input field.

Up to *n* characters are read, converted, and stored in the current address argument.

If the input field contains fewer than *n* characters, the ...*scanf* function reads all the characters in the field, then proceeds with the next field and format specifier.

The success of literal matches and suppressed assignments is not directly determinable.

If the ...*scanf* function encounters a whitespace or non-convertible character before it reads "width" characters, it:

- reads, converts, and stores the characters read so far, then
- attends to the next format specifier.

A non-convertible character is one that can't be converted according to the given format (8 or 9 when the format is octal, *J* or *K* when the format is hexadecimal or decimal, etc.).

Pointer-size and Argument-type Modifiers

See also

These modifiers affect how ...*scanf* functions interpret the corresponding address argument `arg[f]`.

Pointer-size Modifiers

Pointer-size modifiers override the default or declared size of *arg*.

Modifier	arg Interpreted As...
F	**Far** pointer
N	**Near** pointer (Can't be used with any conversion in huge model)

Argument-type Modifiers

Argument-type modifiers indicate which type of the following input data is to be used (h = **short**, l = **long**, L = **long double**).

The input data is converted to the specified version, and the *arg* for that input data should point to an object of corresponding size.

Modifier	For This Type	Convert Input to...
h	*d i o u x*	**short int**; store in **short** object
	D I O U X	(No effect)
	e f c s n p	(No effect)
l	*d i o u x*	**long int**; store in **long** object
	e f g	**double**; store in **double** object
	D I O U X	(No effect)
	c s n p	(No effect)
L	*e f g*	**long double**; store in **long double** object
	(all others)	(No effect)

Format Specifier Conventions

See also

Certain conventions accompany some of the ...scanf format specifiers for the following conversions:

single character (%c)

character array (%[W]c)

string (%s)

floating-point (%e, %E, %f, %g, and %G)

unsigned (%d, %i, %o, %x, %D, %I, %O, %X, %c, %n)

search sets(%[...], %[^...])

Single Character Conversion (%c)

See also

This specification reads the next character, including a whitespace character.

To skip one whitespace character and read the next non-whitespace character, use `%1s`.

Character Array Conversion (%[W]c)

`[W]` = width specification

The address argument is a pointer to an array of characters (**`char`** *`arg[W]`*).

The array consists of *W* elements.

String Conversion (%s)

See also

The address argument is a pointer to an array of characters (**`char`** *`arg[]`*).

The array size must be *at least* (n+1) bytes, where n = the length of string *s* (in characters).

A space or newline character terminates the input field.

A null terminator is automatically appended to the string and stored as the last element in the array.

Floating-point Conversions (%e, %E, %f, %g, and %G)

See also

Floating-point numbers in the input field must conform to the following generic format:

```
[+/-] ddddddddd [.] dddd [E|e] [+/-] ddd
```

where `[item]` indicates that *item* is optional, and *ddd* represents digits (decimal, octal, or hexadecimal).

In addition, +INF, -INF, +NAN, and -NAN are recognized as floating-point numbers. The sign (+ or -) and capitalization are required.

Unsigned Conversions (%d, %i, %o, %x, %D, %I, %O, %X, %c, and %n)

See also

A pointer to **unsigned** character, **unsigned** integer, or **unsigned long** can be used in any conversion where a pointer to a character, integer, or **long** is allowed.

Search Set Conversion (%[search_set])

See also Examples

The set of characters surrounded by brackets can be substituted for the *s*-type character.

The address argument is a pointer to an array of characters (**char** *arg[]*).

These brackets surround a set of characters that define a *search set* of possible characters making up the string (the input field).

If the first character in the brackets is a caret (^), the search set is inverted to include all ASCII characters except those between the brackets.

(Normally, a caret will be included in the inverted search set unless explicitly listed somewhere after the first caret.)

The input field is a string not delimited by whitespace. *...scanf* reads the corresponding input field up to the first character it reaches that does not appear in the search set (or in the inverted search set).

Rules covering search set ranges

1. The character prior to the hyphen (-) must be lexically less than the one after it.
2. The hyphen must not be the first or last character in the set. (If it is first or last, it is considered to just be the hyphen character, not a range definer.)
3. The characters on either side of the hyphen must be the ends of the range and not part of some other range.

Examples

%[abcd]	Searches the input field for any of the characters *a, b, c*, and *d*
%[^abcd]	Searches the input field for any characters except *a, b, c*, and *d*

You can also use a range facility shortcut [<first>-<last>] to define a range of letters or numerals in the search set.

Examples

To catch all decimal digits, you could define the search set with the explicit search set: %[0123456789] or with the range shortcut: %[0-9]

To catch alphanumeric characters, you could use the following shortcuts:

%[A-Z]	Catches all uppercase letters
%[0-9A-Za-z]	Catches all decimal digits and all letters
%[A-FT-Z]	Catches all uppercase letters from *A* through *F* and from *T* through *Z*.

When ...scanf Functions Stop Scanning

See also

A ...*scanf* function might stop scanning a particular input field before reaching the normal field-end character (whitespace), or it might terminate entirely.

Stop and Skip to Next Input Field

...*scanf* functions stop scanning and storing the current input field and proceed to the next one if any of the following occurs:

- An assignment-suppression character (*) appears after the % in the format specifier. The current input field is scanned but not stored.
- width characters have been read.
- The next character read can't be converted under the current format (for example, an *A* when the format is decimal).
- The next character in the input field does not appear in the search set (or does appear in an inverted search set).

When *scanf* stops scanning the current input field for one of these reasons, it assumes that the next character is unread and is either

- the first character of the following input field, or
- the first character in a subsequent read operation on the input.

Terminate

...*scanf* functions will terminate under the following circumstances:

1. The next character in the input field conflicts with a corresponding non-whitespace character in the format string.
2. The next character in the input field is EOF.
3. The format string has been exhausted.

If a character sequence that is not part of a format specifier occurs in the format string, it must match the current sequence of characters in the input field.

...*scanf* functions will scan but not store the matched characters.

When a conflicting character occurs, it remains in the input field as if the *...scanf* function never read it.

...scanf functions

The ..*scanf* functions include

fscanf	scans and formats input from a stream
scanf	scans and formats input from stdin
sscanf	scans and formats input from a string
vfscanf	scans and formats input from a stream, using an argument list
vscanf	scans and formats input from stdin using an argument list
vsscanf	scans and formats input from a string, using an argument list

_searchenv, _wsearchenv

See also Example Portability

Syntax

```
#include <stdlib.h>
char *_searchenv(const char *file, const char *varname, char *buf);
char *_wsearchenv(const wchar_t *file, const wchar_t *varname, wchar_t
  *buf);
```

Description

Searches an environment path for a file.

_*searchenv* attempts to locate *file*, searching along the path specified by the operating system environment variable *varname*. Typical environment variables that contain paths are PATH, LIB, and INCLUDE.

_*searchenv* searches for the file in the current directory of the current drive first. If the file is not found there, the environment variable *varname* is fetched, and each directory in the path it specifies is searched in turn until the file is found, or the path is exhausted.

When the file is located, the full path name is stored in the buffer pointed to by *buf*. This string can be used in a call to access the file (for example, with *fopen* or *exec...*). The buffer is assumed to be large enough to store any possible file name. If the file cannot be successfully located, an empty string (consisting of only a null character) will be stored at *buf*.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

searchpath, wsearchpath

See also   Example   Portability

Syntax

```
#include <dir.h>
char *searchpath(const char *file);
wchar_t *wsearchpath( const wchar_t *file );
```

Description

Searches the operating system path for a file.

searchpath attempts to locate *file*, searching along the operating system path, which is the PATH=... string in the environment. A pointer to the complete path-name string is returned as the function value.

searchpath searches for the file in the current directory of the current drive first. If the file is not found there, the PATH environment variable is fetched, and each directory in the path is searched in turn until the file is found, or the path is exhausted.

When the file is located, a string is returned containing the full path name. This string can be used in a call to access the file (for example, with *fopen* or *exec...*).

The string returned is located in a static buffer and is overwritten on each subsequent call to *searchpath*.

Return Value

searchpath returns a pointer to a file name string if the file is successfully located; otherwise, *searchpath* returns null.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

_searchstr, _wsearchstr

See also Example Portability

Syntax

```
#include <stdlib.h>
void _searchstr(const char *file, const char *ipath, char *buf);
void _wsearchstr(const wchar_t *file, const wchar_t *ipath,wchar_t
  *pathname);
```

Description

Searches a list of directories for a file.

_searchstr attempts to locate *file*, searching along the path specified by the string *ipath*.

_searchstr searches for the file in the current directory of the current drive first. If the file is not found there, each directory in *ipath* is searched in turn until the file is found, or the path is exhausted. The directories in *ipath* must be separated by semicolons.

When the file is located, the full path name is stored in the buffer pointed by by *buf*. This string can be used in a call to access the file (for example, with *fopen* or *exec...*). The buffer is assumed to be large enough to store any possible file name. The constant _MAX_PATH defined in stdlib.h, is the size of the largest file name. If the file cannot be successfully located, an empty string (consisting of only a null character) will be stored at *buf*.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

setbuf

[See also](#) [Example](#) [Portability](#)

Syntax

```
#include <stdio.h>
void setbuf(FILE *stream, char *buf);
```

Description

Assigns buffering to a stream.

setbuf causes the buffer *buf* to be used for I/O buffering instead of an automatically allocated buffer. It is used after *stream* has been opened.

If *buf* is null, I/O will be unbuffered; otherwise, it will be fully buffered. The buffer must be BUFSIZ bytes long (specified in stdio.h).

stdin and *stdout* are unbuffered if they are not redirected; otherwise, they are fully buffered. *setbuf* can be used to change the buffering style used.

Unbuffered means that characters written to a stream are immediately output to the file or device, while *buffered* means that the characters are accumulated and written as a block.

setbuf produces unpredictable results unless it is called immediately after opening *stream* or after a call to *fseek*. Calling *setbuf* after *stream* has been unbuffered is legal and will not cause problems.

A common cause for error is to allocate the buffer as an automatic (local) variable and then fail to close the file before returning from the function where the buffer was declared.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

_setcursortype

Example Portability

Syntax

```
#include <conio.h>
void _setcursortype(int cur_t);
```

Description

Selects cursor appearance.

Sets the cursor type to

_NOCURSOR	Turns off the cursor
_NORMALCURSOR	Normal underscore cursor
_SOLIDCURSOR	Solid block cursor

Note: Do not use this function for Win32s or Win32 GUI applications.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+			+			+

setjmp

See also Example Portability

Syntax

```
#include <setjmp.h>
int setjmp(jmp_buf jmpb);
```

Description

Sets up for nonlocal goto.

setjmp captures the complete *task state* in *jmpb* and returns 0.

A later call to *longjmp* with *jmpb* restores the captured task state and returns in such a way that *setjmp* appears to have returned with the value *val*.

A task state includes

Win 16	Win 32
All segment registers CS, DS, ES, SS	No segment registers are saved
Register variables	Register variables
DI and SI	EBX, EDI, ESI
Stack pointer SP	Stack pointer ESP
Frame pointer BP	Frame pointer EBP
Flags	Flags are not saved

A task state is complete enough that *setjmp* can be used to implement co-routines.

setjmp must be called before *longjmp*. The routine that calls *setjmp* and sets up *jmpb* must still be active and cannot have returned before the *longjmp* is called. If it has returned, the results are unpredictable.

setjmp is useful for dealing with errors and exceptions encountered in a low-level subroutine of a program.

DOS Users

You cannot use *setjmp* and *longjmp* for implementing co-routines if your program is overlaid. Normally, *setjmp* and *longjmp* save and restore all the registers needed for co-routines, but the overlay manager needs to keep track of stack contents and assumes there is only one stack. When you implement co-routines there are usually either two stacks or two partitions of one stack, and the overlay manager will not track them properly.

You can have background tasks that run with their own stacks or sections of stack, but you must ensure that the background tasks do not invoke any overlaid code, and you must not use the overlay versions of *setjmp* or *longjmp* to switch to and from background. When you avoid using overlay code or support routines, the existence of the background stacks does not disturb the overlay manager.

Return Value

setjmp returns 0 when it is initially called. If the return is from a call to *longjmp*, *setjmp* returns a nonzero value (as in the example).

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

setlocale, _wsetlocale

See also Example Portability

Syntax

```
#include <locale.h>
char *setlocale(int category, const char *locale);
wchar_t * _wsetlocale( int category, const wchar_t *locale);
```

Description

Use the *setlocale* to select or query a locale.

Borland C++ supports all locales supported in NT 3.5x and Win95/NT 4.0 operating systems. See your system documentation for details.

The possible values for the *category* argument are as follows:

Value	Affect
LC_ALL	Affects all the following categories
LC_COLLATE	Affects *strcoll* and *strxfrm*
LC_CTYPE	Affects single-byte character handling functions. The *mbstowcs* and *mbtowc* functions are not affected.
LC_MONETARY	Affects monetary formatting by the *localeconv* function
LC_NUMERIC	Affects the decimal point of non-monetary data formatting. This includes the *printf* family of functions, and the information returned by *localeconv*.
LC_TIME	Affects *strftime*

The *locale* argument is a pointer to the name of the locale or named locale category. Passing a NULL pointer returns the current locale in effect. Passing a pointer that points to a null string requests *setlocale* to look for environment variables to determine which locale to set. The locale names are **not** case sensitive.

When *setlocale* is unable to honor a locale request, the preexisting locale in effect is unchanged and a null pointer is returned.

If the *locale* argument is a NULL pointer, the locale string for the category is returned. If *category* is LC_ALL, a complete locale string is returned. The structure of the complete locale string consists of the names of all the categories in the current locale concatenated and separated by semicolons. This string can be used as the locale parameter when calling *setlocale* with any of the LC_*xxx* values. This will reinstate all the locale categories that are named in the complete locale string, and allows saving and restoring of locale states. If the complete locale string is used with a single category, for example, LC_TIME, only that category will be restored from the locale string.

If an empty string "" is used as the locale parameter an implementation-defined locale is used. This is the ANSI C specified behavior.

To take advantage of dynamically loadable locales in your application, define _ _USELOCALES_ _ for each module. If _ _USELOCALES_ _ is not defined, all locale-sensitive functions and macros will work only with the default C locale.

If a NULL pointer is used as the argument for the *locale* parameter, *setlocale* returns a string that specifies the current locale in effect. If the *category* parameter specifies a single category, such as LC_COLLATE, the string pointed to will be the name of that category. If LC_ALL is used as the *category* parameter then the string pointed to will be a full locale string that will indicate the name of each category in effect.

```
      .
      .
      .
localenameptr = setlocale( LC_COLLATE, NULL );
```

```
if (localenameptr)
   printf( "%s\n", localenameptr );
    .
    .
    .
```

The output here will be one of the module names together with the specified code page. For example, the output could be `LC_COLLATE = English_United States.437`.

```
    .
    .
    .
localenameptr = setlocale( LC_ALL, NULL );

if (localenameptr)
   printf( "%s\n", localenameptr );
    .
    .
    .
```

An example of the output here could be the following:

```
LC_COLLATE=English_United States.437;
LC_TIME=English_United States.437;
LC_CTYPE=English_United States.437;
```

Each category in this full string is delimited by a semicolon. This string can be copied and saved by an application and then used again to restore the same locale categories at another time. Each delimited name corresponds to the locale category constants defined in locale.h. Therefore, the first name is the name of the LC_COLLATE category, the second is the LC_CTYPE category, and so on. Any other categories named in the locale.h header file are reserved for future implementation.

To set all default categories for the specified French locale:

```
setlocale( LC_ALL, "French_France.850" );
```

To find out which code page is currently being used:

```
localenameptr = setlocale( LC_ALL, NULL );
```

Return value

If selection is successful, *setlocale* returns a pointer to a string that is associated with the selected category (or possibly all categories) for the new locale.

If UNICODE is defined, *_wsetlocale* returns a **wchar_t** string.

On failure, a NULL pointer is returned and the locale is unchanged. All other possible returns are discussed in the Remarks section above.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+	+	+	+

setmem

See also   Example   Portability

Syntax

```
#include <mem.h>
void setmem(void *dest, unsigned length, char value);
```

Description

Assigns a value to a range of memory.

setmem sets a block of *length* bytes, pointed to by *dest*, to the byte *value*.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

setmode

See also Example Portability

Syntax

```
#include <io.h>
int setmode(int handle, int amode);
```

Description

Sets mode of an open file.

setmode sets the mode of the open file associated with *handle* to either binary or text. The argument *amode* must have a value of either O_BINARY or O_TEXT, never both. (These symbolic constants are defined in fcntl.h.)

Return Value

setmode returns the previous translation mode if successful. On error it returns -1 and sets the global variable errno to

EINVAL	Invalid argument

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

setvbuf

See also Example Portability

Syntax

```
#include <stdio.h>
int setvbuf(FILE *stream, char *buf, int type, size_t size);
```

Description

Assigns buffering to a stream.

setvbuf causes the buffer *buf* to be used for I/O buffering instead of an automatically allocated buffer. It is used after the given stream is opened.

If *buf* is null, a buffer will be allocated using *malloc*; the buffer will use *size* as the amount allocated. The buffer will be automatically freed on close. The *size* parameter specifies the buffer size and must be greater than zero.

The parameter size is limited by the constant UINT_MAX as defined in limits.h.

stdin and *stdout* are unbuffered if they are not redirected; otherwise, they are fully buffered. *Unbuffered* means that characters written to a stream are immediately output to the file or device, while *buffered* means that the characters are accumulated and written as a block.

The *type* parameter is one of the following:

_IOFBF	*fully buffered* file. When a buffer is empty, the next input operation will attempt to fill the entire buffer. On output, the buffer will be completely filled before any data is written to the file.
_IOLBF	*line buffered* file. When a buffer is empty, the next input operation will still attempt to fill the entire buffer. On output, however, the buffer will be flushed whenever a newline character is written to the file.
_IONBF	*unbuffered* file. The *buf* and *size* parameters are ignored. Each input operation will read directly from the file, and each output operation will immediately write the data to the file.

A common cause for error is to allocate the buffer as an automatic (local) variable and then fail to close the file before returning from the function where the buffer was declared.

Return Value

On success, *setvbuf* returns 0.

On error (if an invalid value is given for *type* or *size*, or if there is not enough space to allocate a buffer), it returns nonzero.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

signal

See also   Example   Portability

Syntax

```
#include <signal.h>
void (_USERENTRY *signal(int sig, void (_USERENTRY *func)
                                (int sig[, int subcode])))(int);
```

Description

Specifies signal-handling actions.

signal determines how receipt of signal number *sig* will subsequently be treated. You can install a user-specified handler routine (specified by the argument *func*) or use one of the two predefined handlers, SIG_DFL and SIG_IGN, in signal.h. The function *func* must be used with the _USERENTRY calling convention.

A routine that catches a signal (such as a floating point) also clears the signal. To continue to receive signals, a signal handler must be reinstalled by calling signal again.

Function Pointer	Description
SIG_DFL	Terminates the program
SIG_ERR	Indicates an error return from signal
SIG_IGN	Ignore this type signal

The following table shows signal types and their defaults:

Signal Type	Description
SIGBREAK	Keyboard must be in raw mode.
SIGABRT	Abnormal termination. Default action is equivalent to calling _*exit*(3).
SIGFPE	Arithmetic error caused by division by 0, invalid operation, and the like. Default action is equivalent to calling _*exit*(1).
SIGILL	Illegal operation. Default action is equivalent to calling _*exit*(1).
SIGINT	Ctrl-C interrupt. Default action is to do an INT 23h.
SIGSEGV	Illegal storage access. Default action is equivalent to calling _*exit*(1).
SIGTERM	Request for program termination. Default action is equivalent to calling _*exit*(1).
SIGUSR1, SIGUSR2, SIGUSR3	User-defined signals (available only in Win32) can be generated only by calling *raise*. Default action is to ignore the signal

signal.h defines a type called *sig_atomic_t*, the largest integer type the processor can load or store atomically in the presence of asynchronous interrupts (for the 8086 family, this is a 16-bit word, for 80386 and higher number processors, it is a 32-bit word -- a Borland C++ integer).

When a signal is generated by the *raise* function or by an external event, the following two things happen:

- If a user-specified handler has been installed for the signal, the action for that signal type is set to SIG_DFL.
- The user-specified function is called with the signal type as the parameter.

User-specified handler functions can terminate by a return or by a call to *abort, _exit, exit*, or *longjmp*. If your handler function is expected to continue to receive and handle more signals, you must have the handler function call *signal* again.

Borland C++ implements an extension to ANSI C when the signal type is SIGFPE, SIGSEGV, or SIGILL. The user-specified handler function is called with one or two extra parameters. If SIGFPE, SIGSEGV, or SIGILL has been raised as the result of an explicit call to the *raise* function, the user-specified handler is

called with one extra parameter, an integer specifying that the handler is being explicitly invoked. The explicit activation values for SIGFPE, SIGSEGV and SIGILL are as follows

Note: Declarations of these types are defined in float.h.

SIGSEGV signal	Meaning
SIGFPE	FPE_EXPLICITGEN
SIGSEGV	SEGV_EXPLICITGEN
SIGILL	ILL_EXPLICITGEN

If SIGFPE is raised because of a floating-point exception, the user handler is called with one extra parameter that specifies the FPE_xxx type of the signal. If SIGSEGV, SIGILL, or the integer-related variants of SIGFPE signals (FPE_INTOVFLOW or FPE_INTDIV0) are raised as the result of a processor exception, the user handler is called with two extra parameters:

1. The SIGFPE, SIGSEGV, or SIGILL exception type (see float.h for all these types). This first parameter is the usual ANSI signal type.
2. An integer pointer into the stack of the interrupt handler that called the user-specified handler. This pointer points to a list of the processor registers saved when the exception occurred. The registers are in the same order as the parameters to an interrupt function; that is, BP, DI, SI, DS, ES, DX, CX, BX, AX, IP, CS, FLAGS. To have a register value changed when the handler returns, change one of the locations in this list.

   For example, to have a new SI value on return, do something like this:

   ```
   *((int*)list_pointer + 2) = new_SI_value;
   ```

   In this way, the handler can examine and make any adjustments to the registers that you want.

The following SIGFPE-type signals can occur (or be generated). They correspond to the exceptions that the 8087 family is capable of detecting, as well as the "INTEGER DIVIDE BY ZERO" and the "INTERRUPT ON OVERFLOW" on the main CPU. (The declarations for these are in float.h.)

SIGFPE signal	Meaning
FPE_INTOVFLOW	INTO executed with OF flag set
FPE_INTDIV0	Integer divide by zero
FPE_INVALID	Invalid operation
FPE_ZERODIVIDE	Division by zero
FPE_OVERFLOW	Numeric overflow
FPE_UNDERFLOW	Numeric underflow
FPE_INEXACT	Precision
FPE_EXPLICITGEN	User program executed *raise*(SIGFPE)
FPE_STACKFAULT	Floating-point stack overflow or underflow
FPE_STACKFAULT	Stack overflow

The FPE_INTOVFLOW and FPE_INTDIV0 signals are generated by integer operations, and the others are generated by floating-point operations. Whether the floating-point exceptions are generated depends on the coprocessor control word, which can be modified with *_control87*. Denormal exceptions are handled by Borland C++ and not passed to a signal handler.

The following SIGSEGV-type signals can occur:

SEGV_BOUND	Bound constraint exception
SEGV_EXPLICITGEN	*raise*(SIGSEGV) was executed

The 8088 and 8086 processors *don't* have a bound instruction. The 186, 286, 386, and NEC V series processors *do* have this instruction. So, on the 8088 and 8086 processors, the SEGV_BOUND type of SIGSEGV signal won't occur. Borland C++ doesn't generate bound instructions, but they can be used in

inline code and separately compiled assembler routines that are linked in.

The following SIGILL-type signals can occur:

ILL_EXECUTION	Illegal operation attempted
ILL_EXPLICITGEN	*raise*(SIGILL) was executed

The 8088, 8086, NEC V20, and NEC V30 processors *do not* have an illegal operation exception. The 186, 286, 386, NEC V40, and NEC V50 processors *do* have this exception type. On 8088, 8086, NEC V20, and NEC V30 processors, the ILL_EXECUTION type of SIGILL won't occur.

When the signal type is SIGFPE, SIGSEGV, or SIGILL, a return from a signal handler is generally not advisable if the state of the 8087 is corrupt, the results of an integer division are wrong, an operation that shouldn't have overflowed did, a bound instruction failed, or an illegal operation was attempted. The only time a return is reasonable is when the handler alters the registers so that a reasonable return context exists *or* the signal type indicates that the signal was generated explicitly (for example, FPE_EXPLICITGEN, SEGV_EXPLICITGEN, or ILL_EXPLICITGEN). Generally in this case you would print an error message and terminate the program using _exit, exit, or abort. If a return is executed under any other conditions, the program's action will probably be unpredictable.

Note: Take special care when using the *signal* function in a multithread program. The SIGINT, SIGTERM, and SIGBREAK signals can be used only by the main thread (thread one) in a non-Win32 application. When one of these signals occurs, the currently executing thread is suspended, and control transfers to the signal handler (if any) set up by thread one. Other signals can be handled by any thread.

A signal handler should not use C++ run-time library functions, because a semaphore deadlock might occur. Instead, the handler should simply set a flag or post a semaphore, and return immediately.

Return Value

On success, *signal* returns a pointer to the previous handler routine for the specified signal type.

On error, *signal* returns SIG_ERR, and the external variable errno is set to EINVAL.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

sin, sinl

See also Example Portability

Syntax

```
#include <math.h>
double sin(double x);
long double sinl(long double x);
```

Description

Calculates sine.

sin computes the sine of the input value. Angles are specified in radians.

sinl is the **long double** version; it takes a **long double** argument and returns a **long double** result. Error handling for these functions can be modified through the functions *_matherr* and *_matherrl*.

Return Value

sin and *sinl* return the sine of the input value.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
sin	+	+	+	+	+		+
sinl	+		+	+			+

sinh, sinhl

See also Example Portability

Syntax

```
#include <math.h>
double sinh(double x);
long double sinhl(long double x);
```

Description

Calculates hyperbolic sine.

sinh computes the hyperbolic sine, $(e^{x} - e^{x})/2$.

sinl is the **long double** version; it takes a **long double** argument and returns a **long double** result. Error handling for *sinh* and *sinhl* can be modified through the functions _*matherr* and _*matherrl*.

Return Value

sinh and *sinhl* return the hyperbolic sine of *x*.

When the correct value overflows, these functions return the value HUGE_VAL (*sinh*) or _LHUGE_VAL (*sinhl*) of appropriate sign. Also, the global variable *errno* is set to ERANGE.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
sinh	+	+	+	+	+		+
sinhl	+		+	+			+

sleep

Example Portability

Syntax

```
#include <dos.h>
void sleep(unsigned seconds);
```

Description

Suspends execution for an interval (seconds).

With a call to *sleep*, the current program is suspended from execution for the number of seconds specified by the argument *seconds*. The interval is accurate only to the nearest hundredth of a second or to the accuracy of the operating system clock, whichever is less accurate.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+		+			+

_sopen, _wsopen

See also   Example   Portability

Syntax

```
#include <fcntl.h>
#include <sys\stat.h>
#include <share.h>
#include <io.h>
#include <stdio.h>
int _sopen(char *path, int access, int shflag[, int mode]);
int _wsopen(wchar_t *path, int access, int shflag[, int mode]);
```

Description

Opens a shared file.

_sopen opens the file given by *path* and prepares it for shared reading or writing, as determined by *access, shflag*, and *mode*.

_wsopen is the Unicode version of *_sopen*. The Unicode version accepts a filename that is a *wchar_t* character string. Otherwise, the functions perform identically.

For *_sopen*, *access* is constructed by ORing flags bitwise from the following lists:

Read/write flags

You can use only one of the following flags:

O_RDONLY	Open for reading only.
O_WRONLY	Open for writing only.
O_RDWR	Open for reading and writing.

Other access flags

You can use any logical combination of the following flags:

O_NDELAY	Not used; for UNIX compatibility.
O_APPEND	If set, the file pointer is set to the end of the file prior to each write.
O_CREA	If the file exists, this flag has no effect. If the file does not exist, the file is created, and the bits of *mode* are used to set the file attribute bits as in *chmod*.
O_TRUNC	If the file exists, its length is truncated to 0. The file attributes remain unchanged.
O_EXCL	Used only with O_CREAT. If the file already exists, an error is returned.
O_BINARY	This flag can be given to explicitly open the file in binary mode.
O_TEXT	This flag can be given to explicitly open the file in text mode.
O_NOINHERIT	The file is not passed to child programs.

Note: These O_... symbolic constants are defined in fcntl.h.

If neither O_BINARY nor O_TEXT is given, the file is opened in the translation mode set by the global variable *_fmode*.

If the O_CREAT flag is used in constructing *access*, you need to supply the *mode* argument to *_sopen* from the following symbolic constants defined in sys\stat.h.

Value of mode	Access permission
S_IWRITE	Permission to write
S_IREAD	Permission to read
S_IREAD|S_IWRITE	Permission to read/write

shflag specifies the type of file-sharing allowed on the file *path*. Symbolic constants for *shflag* are

defined in share.h.

Value of shflag	What it does
SH_COMPAT	Sets compatibility mode.
SH_DENYRW	Denies read/write access
SH_DENYWR	Denies write access
SH_DENYRD	Denies read access
SH_DENYNONE	Permits read/write access
SH_DENYNO	Permits read/write access

Return Value

On success, _*sopen* returns a nonnegative integer (the file handle), and the file pointer (that marks the current position in the file) is set to the beginning of the file.

On error, it returns -1, and the global variable errno is set to

EACCES	ermission denied
EINVACC	nvalid access code
EMFILE	oo many open files
ENOENT	Path or file function not found

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

spawnl, spawnle, spawnlp, spawnlpe, spawnv, spawnve, spawnvp, spawnvpe

See also Examples Portability

Syntax

```
#include <process.h>
#include <stdio.h>
int spawnl(int mode, char *path, char *arg0, arg1, ..., argn, NULL);
int _wspawnl(int mode, wchar_t *path, wchar_t *arg0, arg1, ..., argn, NULL);
int spawnle(int mode, char *path, char *arg0, arg1, ..., argn, NULL, char
  *envp[]);
int _wspawnle(int mode, wchar_t *path, wchar_t *arg0, arg1, ..., argn, NULL,
  wchar_t *envp[]);
int spawnlp(int mode, char *path, char *arg0, arg1, ..., argn, NULL);
int _wspawnlp(int mode, wchar_t *path, wchar_t *arg0, arg1, ..., argn,
  NULL);
int spawnlpe(int mode, char *path, char *arg0, arg1, ..., argn, NULL, char
  *envp[]);
int _wspawnlpe(int mode, wchar_t *path, wchar_t *arg0, arg1, ..., argn,
  NULL, wchar_t *envp[]);
int spawnv(int mode, char *path, char *argv[]);
int _wspawnv(int mode, wchar_t *path, wchar_t *argv[]);
int spawnve(int mode, char *path, char *argv[], char *envp[]);
int _wspawnve(int mode, wchar_t *path, wchar_t *argv[], wchar_t *envp[]);
int spawnvp(int mode, char *path, char *argv[]);
int _wspawnvp(int mode, wchar_t *path, wchar_t *argv[]);
int spawnvpe(int mode, char *path, char *argv[], char *envp[]);
int _wspawnvpe(int mode, wchar_t *path, wchar_t *argv[], wchar_t *envp[]);
```

Note: In *spawnle, spawnlpe, spawnv, spawnve, spawnvp*, and *spawnvpe*, the last string must be NULL.

Description

The functions in the *spawn*... family create and run (execute) other files, known as child processes. There must be sufficient memory available for loading and executing a child process.

The value of *mode* determines what action the calling function (the *parent process*) takes after the *spawn*... call. The possible values of *mode* are

P_WAIT	Puts parent process on hold until child process completes execution.
P_NOWAIT	Continues to run parent process while child process runs. The child process ID is returned, so that the parent can wait for completion using *cwait* or *wait*. This mode is currently not available for 16-bit Windows or 16-bit DOS; using it generates an error value.
P_NOWAITO	Identical to P_NOWAIT except that the child process ID isn't saved by the operating system, so the parent process can't wait for it using *cwait* or *wait*.
P_DETACH	Identical to P_NOWAITO, except that the child process is executed in the background with no access to the keyboard or the display.
P_OVERLAY	Overlays child process in memory location formerly occupied by parent. Same as an *exec...* call.

path is the file name of the called child process. The *spawn...* function calls search for *path* using the standard operating system search algorithm:

- If there is no extension or no period, they search for an exact file name. If the file is not found, they search for files first with the extension EXE, then COM, and finally BAT.
- If an extension is given, they search only for the exact file name.
- If only a period is given, they search only for the file name with no extension.
- If *path* does not contain an explicit directory, *spawn...* functions that have the **p** suffix search the

current directory, then the directories set with the operating system PATH environment variable.

The suffixes *p, l*, and *v*, and *e* added to the *spawn...* "family name" specify that the named function operates with certain capabilities.

p The function searches for the file in those directories specified by the PATH environment variable. Without the p suffix, the function searches only the current working directory.

l The argument pointers *arg0, arg1, ..., argn* are passed as separate arguments. Typically, the *l* suffix is used when you know in advance the number of arguments to be passed.

v The argument pointers *argv[0], ..., arg[n]* are passed as an array of pointers. Typically, the *v* suffix is used when a variable number of arguments is to be passed.

e The argument *envp* can be passed to the child process, letting you alter the environment for the child process. Without the *e* suffix, child processes inherit the environment of the parent process.

Each function in the *spawn...* family must have one of the two argument-specifying suffixes (either *l* or *v*). The path search and environment inheritance suffixes (*p* and *e*) are optional.

For example:

- *spawnl* takes separate arguments, searches only the current directory for the child, and passes on the parent's environment to the child.
- *spawnvpe* takes an array of argument pointers, incorporates PATH in its search for the child process, and accepts the *envp* argument for altering the child's environment.

The *spawn...* functions must pass at least one argument to the child process (*arg0 or argv[0]*). This argument is, by convention, a copy of *path*. (Using a different value for this 0 argument won't produce an error.) If you want to pass an empty argument list to the child process, then *arg0* or *argv[0]* must be NULL.

Under DOS 3.x, path is available for the child process; under earlier versions, the child process cannot use the passed value of the 0 argument (*arg0* or *argv[0]*).

When the *l* suffix is used, *arg0* usually points to *path*, and *arg1, ..., argn* point to character strings that form the new list of arguments. A mandatory null following *argn* marks the end of the list.

When the *e* suffix is used, you pass a list of new environment settings through the argument *envp*. This environment argument is an array of character pointers. Each element points to a null-terminated character string of the form

```
envvar = value
```

where *envvar* is the name of an environment variable, and *value* is the string value to which *envvar* is set. The last element in *envp[]* is null. When *envp* is null, the child inherits the parents' environment settings.

The combined length of *arg0* + *arg1* + ... + *argn* (or of *argv[0]* + *argv[1]* + ... + *argv[n]*), including space characters that separate the arguments, must be less than 260 bytes for Windows (128 for DOS). Null-terminators are not counted.

When a *spawn...* function call is made, any open files remain open in the child process.

Return Value

When successful, the *spawn...* functions, where *mode* is P_WAIT, return the child process' exit status (0 for a normal termination). If the child specifically calls *exit* with a nonzero argument, its exit status can be set to a nonzero value.

If *mode* is P_NOWAIT or P_NOWAITO, the spawn functions return the process ID of the child process. The ID obtained when using P_NOWAIT can be passed to cwait.

On error, the *spawn...* functions return -1, and the global variable errno is set to one of the following values:

E2BIG	Arg list too long
EINVAL	Invalid argument
ENOENT	Path or file name not found

ENOEXEC	Exec format error
ENOMEM	Not enough memory

Examples

spawnl

spawnle

spawnlp

spawnlpe

spawnv

spawnve

spawnvp

spawnvpe

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+			+			+

_splitpath, _wsplitpath

See also   Example   Portability

Syntax

```
#include <stdlib.h>
void _splitpath(const char *path, char *drive, char *dir, char *name, char
  *ext);
void _wsplitpath(const wchar_t *path, wchar_t *drive, wchar_t *dir, wchar_t
  *name, wchar_t *ext);
```

Description

Splits a full path name into its components.

_splitpath takes a file's full path name (*path*) as a string in the form

X:\DIR\SUBDIR\NAME.EXT

and splits *path* into its four components. It then stores those components in the strings pointed to by *drive*, *dir*, *name*, and *ext*. (All five components must be passed, but any of them can be a null, which means the corresponding component will be parsed but not stored.) The maximum sizes for these strings are given by the constants _MAX_DRIVE, _MAX_DIR, _MAX_PATH, _MAX_FNAME, and _MAX_EXT (defined in stdlib.h), and each size includes space for the null-terminator. These constants are defined in stdlib.h.

Constant	String
_MAX_PATH	*path*
_MAX_DRIVE	*drive*; includes colon (:)
_MAX_DIR	*dir*; includes leading and trailing backslashes (\)
_MAX_FNAME	name
_MAX_EXT	*ext*; includes leading dot (.)

_splitpath assumes that there is enough space to store each non-null component.

When *_splitpath* splits *path*, it treats the punctuation as follows:

- *drive* includes the colon (C:, A:, and so on).
- *dir* includes the leading and trailing backslashes (\BC\include\, \source\, and so on).
- *name* includes the file name.
- *ext* includes the dot preceding the extension (.C, .EXE, and so on).

_makepath and *_splitpath* are invertible; if you split a given *path* with *_splitpath*, then merge the resultant components with *_makepath*, you end up with *path*.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

sprintf, swprintf

See also   Example   Portability

Syntax

```
#include <stdio.h>
int sprintf(char *buffer, const char *format[, argument, ...]);
int swprintf(wchar_t *buffer, const wchar_t *format[, argument, ...]);
```

Description

Writes formatted output to a string.

Note: For details on format specifiers, see printf.

sprintf accepts a series of arguments, applies to each a format specifier contained in the format string pointed to by *format*, and outputs the formatted data to a string.

sprintf applies the first format specifier to the first argument, the second to the second, and so on. There must be the same number of format specifiers as arguments.

Return Value

On success, *sprintf* returns the number of bytes output. The return value does not include the terminating null byte in the count.

On error, *sprintf* returns EOF.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

sqrt, sqrtl

See also   Example   Portability

Syntax

```
#include <math.h>
double sqrt(double x);
long double sqrtl(long double x);
```

Description

Calculates the positive square root.

sqrt calculates the positive square root of the argument *x*.

sqrtl is the **long double** version; it takes a **long double** argument and returns a **long double** result. Error handling for these functions can be modified through the functions _matherr and _matherrl.

Return Value

On success, *sqrt* and *sqrtl* return the value calculated, the square root of *x*. If *x* is real and positive, the result is positive. If *x* is real and negative, the global variable *errno* is set to

EDOM	Domain error

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
sqrt	+	+	+	+	+		+
sqrtl	+		+	+			+

srand

See also   Example   Portability

Syntax

```
#include <stdlib.h>
void srand(unsigned seed);
```

Description

Initializes random number generator.

The random number generator is reinitialized by calling *srand* with an argument value of 1. It can be set to a new starting point by calling *srand* with a given *seed* number.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

sscanf, swscanf

See also Example Portability

Syntax

```
#include <stdio.h>
int sscanf(const char *buffer, const char *format[, address, ...]);
int swscanf(const wchar_t *buffer, const wchar_t *format[, address, ...]);
```

Description

Scans and formats input from a string.

Note: For details on format specifiers, see scanf.

sscanf scans a series of input fields, one character at a time, reading from a string. Then each field is formatted according to a format specifier passed to *sscanf* in the format string pointed to by *format*. Finally, *sscanf* stores the formatted input at an address passed to it as an argument following *format*. There must be the same number of format specifiers and addresses as there are input fields.

sscanf might stop scanning a particular field before it reaches the normal end-of-field (whitespace) character, or it might terminate entirely, for a number of reasons. See *scanf* for a discussion of possible causes.

Return Value

On success, *sscanf* returns the number of input fields successfully scanned, converted, and stored; the return value does not include scanned fields that were not stored.

If *sscanf* attempts to read at end-of-string, it returns EOF.

On error (If no fields were stored), it returns 0.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

stackavail

See also Example Portability

Syntax

```
#include <malloc.h>
size_t stackavail(void);
```

Description

Gets the amount of available stack memory.

stackavail returns the number of bytes available on the stack. This is the amount of dynamic memory that *alloca* can access.

Return Value

stackavail returns a *size_t* value indicating the number of bytes available.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

_status87

Example   Portability

Syntax

```
#include <float.h>
unsigned int _status87(void);
```

Description

Gets floating-point status.

_status87 gets the floating-point status word, which is a combination of the 80x87 status word and other conditions detected by the 80x87 exception handler.

Return Value

The bits in the return value give the floating-point status. See float.h for a complete definition of the bits returned by *_status87*.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

stime

See also   Example   Portability

Syntax

```
#include <time.h>
int stime(time_t *tp);
```

Description

Sets system date and time.

stime sets the system time and date. *tp* points to the value of the time as measured in seconds from 00:00:00 GMT, January 1, 1970.

Return Value

stime returns a value of 0.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+				+

_stpcpy, _wcspcpy

See also Example Portability

Syntax

```
#include <string.h>
char *stpcpy(char *dest, const char *src);
wchar * _wcspcpy(wchar *dest, const wchar *src);
```

Description

Copies one string into another.

_stpcpy copies the string *src* to *dest*, stopping after the terminating null character of *src* has been reached.

Return Value

stpcpy returns a pointer to the terminating null character of *dest*.

If UNICODE is defined, *_wcspcpy* returns a pointer to the terminating null character of the **wchar_t** *dest* string.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

strcat, _mbscat, wcscat

Example Portability

Syntax

```
#include <string.h>
char *strcat(char *dest, const char *src);
wchar_t *wcscat(wchar_t *dest, const wchar_t *src);

#include <mbstring.h>
unsigned char *_mbscat(unsigned char *dest, const unsigned char *src);
```

Description

Appends one string to another.

strcat appends a copy of *src* to the end of *dest*. The length of the resulting string is *strlen*(*dest*) + *strlen*(*src*).

Return Value

strcat returns a pointer to the concatenated strings.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
strcat	+	+	+	+	+	+	+
_fstrcat	+		+				

strchr, _mbschr, wcschr

See also Example Portability

Syntax

```
#include <string.h>
char *strchr(const char *s, int c);                          /* C only */

const char *strchr(const char *s, int c);                    // C++ only
char *strchr( char *s, int c);                               // C++ only
wchar_t *wcschr(const wchar_t *s, int c);

#include <mbstring.h>
unsigned char * _mbschr(const unsigned char *s, unsigned int c);
```

Description

Scans a string for the first occurrence of a given character.

strchr scans a string in the forward direction, looking for a specific character. *strchr* finds the *first* occurrence of the character *c* in the string *s*. The null-terminator is considered to be part of the string.

For example:

```
strchr(strs,0)
```

returns a pointer to the terminating null character of the string *strs*.

Return Value

strchr returns a pointer to the first occurrence of the character *c* in *s*; if *c* does not occur in *s*, *strchr* returns null.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
strchr	+	+	+	+	+	+	+

+

strcmp, _mbscmp, wcscmp

See also Example Portability

Syntax

```
#include <string.h>
int strcmp(const char *s1, const char *s2);
int wcscmp(const wchar_t *s1, const wchar_t *s2);

#include <mbstring.h>
int _mbscmp(const unsigned char *s1, const unsigned char *s2);
```

Description

Compares one string to another.

strcmp performs an unsigned comparison of *s1* to *s2*, starting with the first character in each string and continuing with subsequent characters until the corresponding characters differ or until the end of the strings is reached.

Return Value

If s1 is...	return value is...
less than *s2*	< 0
the same as *s2*	== 0
greater than *s2*	> 0

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

strcmpi

See also Example Portability

Syntax

```
#include <string.h>
int strcmpi(const char *s1, const char *s2);
```

Description

Compares one string to another, without case sensitivity.

strcmpi performs an unsigned comparison of *s1* to *s2*, without case sensitivity (same as *stricmp*--implemented as a macro).

It returns a value (< 0, 0, or > 0) based on the result of comparing *s1* (or part of it) to *s2* (or part of it).

The routine *strcmpi* is the same as *stricmp*. *strcmpi* is implemented through a macro in string.h and translates calls from *strcmpi* to *stricmp*. Therefore, in order to use *strcmpi*, you must include the header file string.h for the macro to be available. This macro is provided for compatibility with other C compilers.

Return Value

If s1 is...	strcmpi returns a value that is...
less than *s2*	< 0
the same as *s2*	== 0
greater than *s2*	> 0

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+		+			+

strcoll, _stricoll, _mbscoll, _mbsicoll, wcscoll, _wcsicoll

See also Example Portability

Syntax

```
#include <string.h>
int strcoll(const char *s1, const char *s2);
int wcscoll(const wchar_t *s1, const wchar_t *s2);

int _stricoll(const char *s1, const char *s2);
int _wcsicoll(const wchar_t *s1, wconst_t char *s2);

#include <mbstring.h>
int _mbscoll(const unsigned char *s1, const unsigned char *s2);
int _mbsicoll(const unsigned char *s1, const unsigned char *s2);
```

Description

Compares two strings.

strcoll compares the string pointed to by *s1* to the string pointed to by *s2*, according to the current locale's LC_COLLATE category.

_*stricoll* performs like *strcoll* but is not case sensitive.

Return Value

If s1 is...	strcoll and _stricoll return a value that is...
less than *s2*	< 0
the same as *s2*	== 0
greater than *s2*	> 0

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
strcoll	+		+	+	+	+	+
_stricoll	+			+			

strcpy, _mbscpy, wcscpy

See also Example Portability

Syntax

```
#include <string.h>
char *strcpy(char *dest, const char *src);
wchar_t *wcscpy(wchar_t *dest, const wchar_t *src);

#include <mbstring.h>
unsigned char *_mbscpy(unsigned char *dest, const unsigned char *src);
```

Description

Copies one string into another.

Copies string *src* to *dest*, stopping after the terminating null character has been moved.

Return Value

strcpy returns *dest*.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

strcspn, _mbscspn, wcscspn

See also Example Portability

Syntax

```
#include <string.h>
size_t strcspn(const char *s1, const char *s2);
size_t wcscspn(const wchar_t *s1, const wchar_t *s2);

#include <mbstring.h>
size_t _mbscspn(const unsigned char *s1, const unsigned char *s2);
```

Description

Scans a string for the initial segment not containing any subset of a given set of characters.

The *strcspn* functions search *s1* until any one of the characters contained in *s2* is found. The number of characters which were read in *s1* is the return value. The string termination character is not counted. Neither string is altered during the search.

Return Value

strcspn returns the length of the initial segment of string *s1* that consists entirely of characters *not* from string *s2*.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
strcspn	+	+	+	+	+	+	+
_fstrcspn	+		+				

_strdate, _wstrdate

See also Example Portability

Syntax

```
#include <time.h>
char *_strdate(char *buf);
wchar_t *_wstrdate(wchar_t *buf);
```

Description

Converts current date to string.

_strdate converts the current date to a string, storing the string in the buffer *buf*. The buffer must be at least 9 characters long.

The string has the form MM/DD/YY where MM, DD, and YY are all two-digit numbers representing the month, day, and year. The string is terminated by a null character.

Return Value

_strdate returns *buf*, the address of the date string.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

strdup, _mbsdup, _wcsdup

See also   Example   Portability

Syntax

```
#include <string.h>
char *strdup(const char *s);
wchar_t *_wcsdup(const wchar_t *s);

#include <mbstring.h>
unsigned char *_mbsdup(const wchar_t *s);
```

Description

Copies a string into a newly created location.

strdup makes a duplicate of string *s*, obtaining space with a call to *malloc*. The allocated space is (*strlen(s)* + 1) bytes long. The user is responsible for freeing the space allocated by *strdup* when it is no longer needed.

Return Value

strdup returns a pointer to the storage location containing the duplicated string, or returns null if space could not be allocated.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
strdup	+	+	+	+			+
_fstrdup	+		+				

_strerror

See also   Example   Portability

Syntax

```
#include <string.h>
char  *_strerror(const char *s);
```

Description

Builds a customized error message.

_*strerror* lets you generate customized error messages; it returns a pointer to a null-terminated string containing an error message.

- If *s* is null, the return value points to the most recent error message.
- If *s* is not null, the return value contains *s* (your customized error message), a colon, a space, the most-recently generated system error message, and a new line. *s* should be 94 characters or less.

Return Value

_*strerror* returns a pointer to a constructed error string. The error message string is constructed in a static buffer that is overwritten with each call to _*strerror*.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			

strerror

See also   Example   Portability

Syntax

```
#include <string.h>
char *strerror(int errnum);
```

Description

Returns a pointer to an error message string.

strerror takes an **int** parameter *errnum*, an error number, and returns a pointer to an error message string associated with *errnum*.

Return Value

strerror returns a pointer to a constructed error string. The error message string is constructed in a static buffer that is overwritten with each call to *strerror*.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+	+	+	+

strftime, wcsftime

See also Example Portability

Syntax

```
#include <time.h>
size_t strftime(char *s, size_t maxsize, const char *fmt, const struct tm
  *t);
size_t wcsftime(wchar_t *s, size_t maxsize, const wchar_t *fmt, const struct
  tm *t);
```

Description

Formats time for output.

strftime formats the time in the argument *t* into the array pointed to by the argument *s* according to the *fmt* specifications. All ordinary characters are copied unchanged. No more than *maxsize* characters are placed in *s*.

The time is formatted according to the current locale's LC_TIME category.

Return Value

On success, *strftime* returns the number of characters placed into *s.*

On error (if the number of characters required is greater than *maxsize*), *strftime* returns 0.

More about strftime

ANSI-defined format specifiers
POSIX-defined Format Specifiers
POSIX-defined Format Specifier Modifiers

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+	+	+	+

strftime Format String

Consists of zero or more directives and ordinary characters. A directive consists of the % character followed by a character that determines the substitution that is to take place.

ANSI-defined Format Specifiers for strftime

See also

The following table describes the ANSI-defined specifiers for the format string used with strftime.

Format specifier	Substitutes
%%	Character %
%a	Abbreviated weekday name
%A	Full weekday name
%b	Abbreviated month name
%B	Full month name
%c	Date and time
%d	Two-digit day of month (01 - 31)
%H	Two-digit hour (00 - 23)
%I	Two-digit hour (01 - 12)
%j	Three-digit day of year (001 - 366)
%m	Two-digit month as a decimal number (1 - 12)
%M	2-digit minute (00 - 59)
%p	AM or PM
%S	Two-digit second (00 - 59)
%U	Two-digit week number where Sunday is the first day of the week (00 - 53)
%w	Weekday where 0 is Sunday (0 - 6)
%W	Two-digit week number where Monday is the first day of week the week (00 - 53)
%x	Date
%X	Time
%y	Two-digit year without century (00 to 99)
%Y	Year with century
%Z	Time zone name, or no characters if no time zone

POSIX-defined Format Specifiers for strftime

See also

The following table describes the POSIX-defined specifiers for the format string used with strftime.

Note: You must define __USELOCALES__ in order to use these descriptors.

Format specifier	Substitution
%C	Century as a decimal number (00 - 99). For example, 1992 => 19
%D	Date in the format mm/dd/yy
%e	Day of the month as a decimal number in a two-digit field with leading space (1 -31)
%h	A synonym for %b
%n	Newline character
%r	12-hour time (01 - 12) format with am/pm string i.e. "%I:%M:%S %p"
%t	Tab character
%T	24-hour time (00 - 23) in the format "HH:MM:SS"
%u	Weekday as a decimal number (1 Monday - 7 Sunday)

Modifiers

strftime also supports POSIX-defined modifiers for certain specifiers. See POSIX-defined Format Specifier Modifiers.

POSIX-defined Format Specifier Modifiers for strftime

See also

The following table describes the POSIX-defined modifiers for the following format string specifiers used with strftime.

Note: You must define __USELOCALES__ in order to use these descriptors.

Descriptor modifier	Substitutes
%Od	Day of the month using alternate numeric symbols
%Oe	Day of the month using alternate numeric symbols
%OH	Hour (24 hour) using alternate numeric symbols
%OI	Hour (12 hour) using alternate numeric symbols
%Om	Month using alternate numeric symbols
%OM	Minutes using alternate numeric symbols
%OS	Seconds using alternate numeric symbols
%Ou	Weekday as a number using alternate numeric symbols
%OU	Week number of the year using alternate numeric symbols
%Ow	Weekday as number using alternate numeric symbols
%OW	Week number of the year using alternate numeric symbols
%Oy	Year (offset from %C) using alternate numeric symbols

%O modifier

When the %O modifier is used before any of the above supported numeric format descriptors (for example, `%Od`), the numeric value is converted to the corresponding ordinal string, if it exists. If an ordinal string does not exist, the basic format descriptor is used unmodified.

For example, on 4/20/94:

- %d produces 20
- %Od produces 20th

stricmp, _mbsicmp, _wcsicmp

See also Example Portability

Syntax

```
#include <string.h>
int stricmp(const char *s1, const char *s2);
int _wcsicmp(const wchar_t *s1, const wchar_t *s2);

#include <mbstring.h>
int _mbsicmp(const unsigned char *s1, const unsigned char *s2);
```

Description

Compares one string to another, without case sensitivity.

stricmp performs an unsigned comparison of *s1* to *s2*, starting with the first character in each string and continuing with subsequent characters until the corresponding characters differ or until the end of the strings is reached. The comparison is not case sensitive.

It returns a value (< 0, 0, or > 0) based on the result of comparing *s1* (or part of it) to *s2* (or part of it).

The routines *stricmp* and *strcmpi* are the same; *strcmpi* is implemented through a macro in string.h that translates calls from *strcmpi* to *stricmp*. Therefore, in order to use *stricmp*, you must include the header file string.h for the macro to be available.

Return Value

If s1 is...	stricmp returns a value that is...
less than *s2*	< 0
the same as *s2*	== 0
greater than *s2*	> 0

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
stricmp	+	+	+	+	+	+	+
_fstricmp	+		+				

strlen, _mbslen, wcslen, _mbstrlen

Example Portability

Syntax

```
#include <string.h>
size_t strlen(const char *s);
size_t wcslen(const wchar_t *s);

#include <mbstring.h>
size_t _mbslen(const unsigned char *s);

#include <stdlib.h>
size_t _mbstrlen(const char *s)
```

Description

Calculates the length of a string.

strlen calculates the length of *s*.

_mbslen and *_mbstrlen* test the string argument to determine the number of multibyte characters they contain.

_mbstrlen is affected by the *LC_CTYPE* category setting as determined by the *setlocale* function. The function tests to determine whether the string argument is a valid multibyte string.

_mbslen is affected by the code page that is in use. This function doesn’t test for multibyte validity.

Return Value

strlen returns the number of characters in *s*, not counting the null-terminating character.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
strlen	+	+	+	+	+	+	+
_fstrlen	+		+				

strlwr, _mbslwr, _wcslwr

See also Example Portability

Syntax

```
#include <string.h>
char *strlwr(char *s);
wchar_t *_wcslwr(wchar_t *s);

#include <mbstring.h>
unsigned char *_mbslwr(unsigned char *s);
```

Description

Converts uppercase letters in a string to lowercase.

strlwr converts uppercase letters in string *s* to lowercase according to the current locale's LC_CTYPE category. For the C locale, the conversion is from uppercase letters (*A* to *Z*) to lowercase letters (*a* to *z*). No other characters are changed.

Return Value

strlwr returns a pointer to the string *s*.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
strlwr	+	+	+	+	+	+	+
_fstrlwr	+		+				

strncat, _mbsncat, wcsncat

Example Portability

Syntax

```
#include <string.h>
char *strncat(char *dest, const char *src, size_t maxlen);
wchar_t *wcsncat(wchar_t *dest, const wchar_t *src, size_t maxlen);

#include <mbstring.h>
unsigned char *_mbsncat(unsigned char *dest, const unsigned char *src,
  size_t maxlen);
```

Description

Appends a portion of one string to another.

strncat copies at most *maxlen* characters of src to the end of *dest* and then appends a null character. The maximum length of the resulting string is *strlen(dest)* + *maxlen*.

These three functions behave identically and differ only with respect to the type of arguments and return types.

Return Value

strncat returns *dest*.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
strncat	+	+	+	+	+	+	+

strncmp, _mbsncmp, wcsncmp

See also Example Portability

Syntax

```
#include <string.h>
int strncmp(const char *s1, const char *s2, size_t  maxlen);
int wcsncmp(const wchar_t *s1, const wchar_t *s2, size_t  maxlen);

#include <mbstring.h>
int _mbsncmp(const unsigned char *s1, const unsigned char *s2, size_t
  maxlen);
```

Description

Compares a portion of one string to a portion of another.

strncmp makes the same unsigned comparison as *strcmp*, but looks at no more than *maxlen* characters. It starts with the first character in each string and continues with subsequent characters until the corresponding characters differ or until it has examined *maxlen* characters.

Return Value

strncmp returns an *int* value based on the result of comparing *s1* (or part of it) to *s2* (or part of it):

- < 0 if *s1* is less than *s2*
- == 0 if *s1* is the same as *s2*
- > 0 if *s1* is greater than *s2*

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
strncmp	+	+	+	+	+	+	+
_fstrncmp	+		+				

_strncoll, _strnicoll, _mbsncoll, _mbsnicoll, _wcsncoll, _wcsnicoll

See also Portability

Syntax

```
#include <string.h>
int _strncoll(const char *s1, const char *s2, size_t n);
int _wcsncoll(const wchar_t *s1, const wchar_t *s2, size_t n);

int _strnicoll(const char *s1, const char *s2, size_t n);
int _wcsnicoll(const wchar_t *s1, const wchar_t *s2, size_t n);

#include <mbstring.h>
int _mbsncoll(const unsigned char *s1, const unsigned char *s2, size_t n);
int _mbsnicoll(const unsigned char *s1, const unsigned char *s2, size_t n);
```

Description

_strncoll compares *n* number of elements from the string pointed to by *s1* to the string pointed to by *s2*, according to the current locale's LC_COLLATE category.

_strnicoll performs like *_strncoll* but is not case sensitive.

Return Value

If s1 is...	_strncoll and _strnicoll return a value that is...
less than *s2*	< 0
the same as *s2*	== 0
greater than *s2*	> 0

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+			+			

strncmpi, wcsncmpi

Example Portability

Syntax

```
#include <string.h>
int strncmpi(const char *s1, const char *s2, size_t n);
int wcsncmpi(const wchar_t *s1, const wchar_t *s2, size_t n);
```

Description

Compares a portion of one string to a portion of another, without case sensitivity.

strncmpi performs a signed comparison of *s1* to *s2*, for a maximum length of *n* bytes, starting with the first character in each string and continuing with subsequent characters until the corresponding characters differ or until *n* characters have been examined. The comparison is not case sensitive. (*strncmpi* is the same as *strnicmp*--implemented as a macro). It returns a value (< 0, 0, or > 0) based on the result of comparing *s1* (or part of it) to *s2* (or part of it).

The routines *strnicmp* and *strncmpi* are the same; *strncmpi* is implemented through a macro in string.h that translates calls from *strncmpi* to *strnicmp*. Therefore, in order to use *strncmpi*, you must include the header file string.h for the macro to be available. This macro is provided for compatibility with other C compilers.

Return Value

If s1 is...	strncmpi returns a value that is...
less than *s2*	< 0
the same as *s2*	== 0
greater than *s2*	> 0

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			

strncpy, _mbsncpy, wcsncpy

Example Portability

Syntax

```
#include <stdio.h>
char *strncpy(char *dest, const char *src, size_t maxlen);
wchar_t *wcsncpy(wchar_t *dest, const wchar_t *src, size_t maxlen);

#include <mbstring.h>
unsigned char *_mbsncpy(unsigned char *dest, const unsigned char *src,
  size_t maxlen);
```

Description

Copies a given number of bytes from one string into another, truncating or padding as necessary.

strncpy copies up to *maxlen* characters from *src* into *dest*, truncating or null-padding *dest*. The target string, *dest*, might not be null-terminated if the length of *src* is *maxlen* or more.

Return Value

strncpy returns *dest*.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
strncpy	+	+	+	+	+	+	+
_fstrncpy	+		+				

_strnextc, _mbsnextc, _wcsnextc

Example

Syntax

```
#include <tchar.h>
unsigned int _strnextc(const char *str);

#include <mbstring.h>
unsigned int _mbsnextc (const unsigned char *str);
```

Description

These routines should be accessed by using the portable *_tcsnextc* function. The functions inspect the current character in *str*. The pointer to *str* is not advanced.

Return Value

The functions return the integer value of the character pointed to by *str*.

strnicmp, _mbsnicmp, _wcsnicmp

Example Portability

Syntax

```
#include <string.h>
int strnicmp(const char *s1, const char *s2, size_t maxlen);
int _wcsnicmp(const wchar_t *s1, const wchar_t *s2, size_t maxlen);

#include <mbstring.h>
int _mbsnicmp(const unsigned char *s1, const unsigned char *s2, size_t
  maxlen);
```

Description

Compares a portion of one string to a portion of another, without case sensitivity.

strnicmp performs a signed comparison of *s1* to *s2*, for a maximum length of *maxlen* bytes, starting with the first character in each string and continuing with subsequent characters until the corresponding characters differ or until the end of the strings is reached. The comparison is not case sensitive.

It returns a value (< 0, 0, or > 0) based on the result of comparing *s1* (or part of it) to *s2* (or part of it).

Return Value

If s1 is...	strnicmp returns a value that is...
less than *s2*	< 0
the same as *s2*	== 0
greater than *s2*	> 0

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
strnicmp	+		+	+			+
_fstrnicmp	+		+				

strnset, _mbsnset, _wcsnset

Example Portability

Syntax

```
#include <string.h>
char *strnset(char *s, int ch, size_t n);
wchar_t *_wcsnset(wchar_t *s, wchar_t ch, size_t n);

#include <mbstring.h>
unsigned char *_mbsnset(unsigned char *s, unsigned int ch, size_t n);
```

Description

Sets a specified number of characters in a string to a given character.

strnset copies the character *ch* into the first *n* bytes of the string *s*. If *n* > *strlen(s)*, then *strlen(s)* replaces *n*. It stops when *n* characters have been set, or when a null character is found.

Return Value

Each of these functions return *s*.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
strnset	+		+	+			+
_fstrnset	+		+				

strpbrk, _mbspbrk, wcspbrk

Example   Portability

Syntax

```
#include <string.h>
char *strpbrk(const char *s1, const char *s2);          /* C only */
const char *strpbrk(const char *s1, const char *s2);    // C++ only
char *strpbrk(char *s1, const char *s2);                // C++ only
wchar_t * wcspbrk(const wchar_t *s1, const wchar_t *s2);

#include <mbstring.h>
unsigned char *_mbspbrk(const unsigned char *s1, const unsigned char *s2);
```

Description

Scans a string for the first occurrence of any character from a given set.

strpbrk scans a string, *s1*, for the first occurrence of any character appearing in *s2*.

Return Value

strpbrk returns a pointer to the first occurrence of any of the characters in *s2*. If none of the *s2* characters occur in *s1*, *strpbrk* returns null.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
strpbrk	+	+	+	+	+	+	+
_fstpbrk	+		+				

strrchr, _mbsrchr, wcsrchr

See also Example Portability

Syntax

```
char *strrchr(const char *s, int c);                    /* C only */

const char *strrchr(const char *s, int c);              // C++ only
char *strrchr(char *s, int c);                          // C++ only
wchar_t *wcsrchr(const wchar_t *s, wchar_t c);

#include <mbstring.h>
unsigned char * _mbsrchr(const unsigned char *s, unsigned int c);
```

Description

Scans a string for the last occurrence of a given character.

strrchr scans a string in the reverse direction, looking for a specific character. *strrchr* finds the *last* occurrence of the character *c* in the string *s*. The null-terminator is considered to be part of the string.

Return Value

strrchr returns a pointer to the last occurrence of the character *c*. If *c* does not occur in *s*, *strrchr* returns null.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
strrchr	+	+	+	+	+	+	+
_fstrrchr	+		+				

strrev, _mbsrev, _wcsrev

Example Portability

Syntax

```
#include <string.h>
char *strrev(char *s);
wchar_t *_wcsrev(wchar_t *s);

#include <mbstring.h>
unsigned char *_mbsrev(unsigned char *s);
```

Description

Reverses a string.

strrev changes all characters in a string to reverse order, except the terminating null character. (For example, it would change *string\0* to *gnirts\0*.)

Return Value

strrev returns a pointer to the reversed string.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
strrev	+		+	+			+
_fstrrev	+		+				

strset, _mbsset, _wcsset

See also Example Portability

Syntax

```
#include <string.h>
char *strset(char *s, int ch);
wchar_t *_wcsset(wchar_t *s, wchar_t ch);

#include <mbstring.h>
unsigned char *_mbsset(unsigned char *s, unsigned int ch);
```

Description

Sets all characters in a string to a given character.

strset sets all characters in the string *s* to the character *ch*. It quits when the terminating null character is found.

Return Value

strset returns *s*.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
strset	+		+	+			+
_fstrset	+		+				

strspn, _mbsspn, wcsspn

See also Example Portability

Syntax

```
#include <string.h>
size_t strspn(const char *s1, const char *s2);
size_t wcsspn(const wchar_t *s1, const wchar_t *s2);

#include <mbstring.h>
size_t _mbsspn(const unsigned char *s1, const unsigned char *s2);
```

Description

Scans a string for the first segment that is a subset of a given set of characters.

strspn finds the initial segment of string *s1* that consists entirely of characters from string *s2*.

Return Value

strspn returns the length of the initial segment of *s1* that consists entirely of characters from *s2*.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
strspn	+	+	+	+	+	+	+
_fstrspn	+		+				

strstr, _mbsstr, wcsstr

Example Portability

Syntax

```
#include <string.h>
char *strstr(const char *s1, const char *s2);                /* C only */
const char *strstr(const char *s1, const char *s2);          // C++ only
char *strstr(char *s1, const char *s2);                      // C++ only
wchar_t * wcsstr(const wchar_t *s1, const wchar_t *s2);

#include <mbstring.h>
unsigned char * _mbsstr(const unsigned char *s1, const unsigned char *s2);
```

Description

Scans a string for the occurrence of a given substring.

strstr scans *s1* for the first occurrence of the substring *s2*.

Return Value

strstr returns a pointer to the element in *s1*, where *s2* begins (points to *s2* in *s1*). If *s2* does not occur in *s1*, *strstr* returns null.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
strstr	+	+	+	+	+	+	+
_fstrstr	+		+				

_strtime, _wstrtime

See also Example Portability

Syntax

```
#include <time.h>
char *_strtime(char *buf);
wchar_t *_wstrtime(wchar_t *buf);
```

Description

Converts current time to string.

_*strtime* converts the current time to a string, storing the string in the buffer *buf*. The buffer must be at least 9 characters long.

The string has the following form:

```
HH:MM:SS
```

where HH, MM, and SS are all two-digit numbers representing the hour, minute, and second, respectively. The string is terminated by a null character.

Return Value

_*strtime* returns *buf*, the address of the time string.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

strtod, _strtold, wcstod, _wcstold

See also Example Portability

Syntax

```
#include <stdlib.h>
double strtod(const char *s, char **endptr);
double wcstod(const wchar_t *s, wchar_t **endptr);
long double _strtold(const char *s, char **endptr);
long double _wcstold(const wchar_t *s, wchar_t **endptr);
```

Description

Convert a string to a **double** or **long double** value.

strtod converts a character string, *s*, to a **double** value. *s* is a sequence of characters that can be interpreted as a **double** value; the characters must match this generic format:

```
[ws] [sn] [ddd] [.] [ddd] [fmt[sn]ddd]
```

where:

[ws]	=	optional whitespace
[sn]	=	optional sign (+ or -)
[ddd]	=	optional digits
[fmt]	=	optional *e* or *E*
[.]	=	optional decimal point

strtod also recognizes +INF and -INF for plus and minus infinity, and +NAN and -NAN for Not-a-Number.

For example, here are some character strings that *strtod* can convert to **double**:

```
+ 1231.1981 e-1
502.85E2
+ 2010.952
```

strtod stops reading the string at the first character that cannot be interpreted as an appropriate part of a **double** value.

If *endptr* is not null, *strtod* sets **endptr* to point to the character that stopped the scan (**endptr* = &*stopper*). *endptr* is useful for error detection.

_strtold is the **long double** version; it converts a string to a **long double** value.

Return Value

These functions return the value of *s* as a **double** (*strtod*) or a **long double** (*_strtold*). In case of overflow, they return plus or minus HUGE_VAL (*strtod*) or _LHUGE_VAL (*_strtold*).

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
strtod	+	+	+	+	+	+	+
_strtold	+		+	+			+

strtok, _mbstok, wcstok

[Example](#) [Portability](#)

Syntax

```
#include <string.h>
char *strtok(char *s1, const char *s2);
wchar_t *wcstok(wchar_t *s1, const wchar_t *s2);

#include <mbstring.h>
unsigned char *_mbstok(unsigned char *s1, const unsigned char *s2);
```

Description

Searches one string for tokens, which are separated by delimiters defined in a second string.

strtok considers the string *s1* to consist of a sequence of zero or more text tokens, separated by spans of one or more characters from the separator string *s2*.

The first call to *strtok* returns a pointer to the first character of the first token in *s1* and writes a null character into *s1* immediately following the returned token. Subsequent calls with null for the first argument will work through the string *s1* in this way, until no tokens remain.

The separator string, *s2*, can be different from call to call.

Note: Calls to *strtok* cannot be nested with a function call that also uses *strtok*. Doing so will causes an endless loop.

Return Value

strtok returns a pointer to the token found in *s1*. A NULL pointer is returned when there are no more tokens.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
strtok	+	+	+	+	+	+	+
_fstrtok	+		+				

strtol, wcstol

See also Example Portability

Syntax

```
#include <stdlib.h>
long strtol(const char *s, char **endptr, int radix);
long wcstol(const wchar_t *s, wchar_t **endptr, int radix);
```

Description

Converts a string to a **long** value.

strtol converts a character string, *s*, to a **long** integer value. *s* is a sequence of characters that can be interpreted as a **long** value; the characters must match this generic format:

```
[ws] [sn] [0] [x] [ddd]
```

where:

[ws]	=	optional whitespace
[sn]	=	optional sign (+ or -)
[0]	=	optional zero (0)
[x]	=	optional x or X
[ddd]	=	optional digits

strtol stops reading the string at the first character it doesn't recognize.

If *radix* is between 2 and 36, the long integer is expressed in base *radix*. If *radix* is 0, the first few characters of *s* determine the base of the value being converted.

First character	Second character	String interpreted as...
0	1 - 7	Octal
0	x or X	Hexadecimal
1 - 9	Decimal	

If *radix* is 1, it is considered to be an invalid value. If *radix* is less than 0 or greater than 36, it is considered to be an invalid value.

Any invalid value for *radix* causes the result to be 0 and sets the next character pointer **endptr* to the starting string pointer.

If the value in *s* is meant to be interpreted as octal, any character other than 0 to 7 will be unrecognized.

If the value in *s* is meant to be interpreted as decimal, any character other than 0 to 9 will be unrecognized.

If the value in *s* is meant to be interpreted as a number in any other base, then only the numerals and letters used to represent numbers in that base will be recognized. (For example, if *radix* equals 5, only 0 to 4 will be recognized; if *radix* equals 20, only 0 to 9 and *A* to *J* will be recognized.)

If *endptr* is not null, *strtol* sets **endptr* to point to the character that stopped the scan (**endptr* = &*stopper*).

Return Value

strtol returns the value of the converted string, or 0 on error.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+	+	+	+

strtoul, wcstoul

See also Example Portability

Syntax

```
#include <stdlib.h>
unsigned long strtoul(const char *s, char **endptr, int radix);
unsigned long wcstoul(const wchar_t *s, wchar_t **endptr, int radix);
```

Description

Converts a string to an **unsigned long** in the given radix.

strtoul operates the same as *strtol*, except that it converts a string *str* to an **unsigned long** value (where *strtol* converts to a **long**). Refer to the entry for *strtol* for more information.

Return Value

strtoul returns the converted value, an **unsigned long**, or 0 on error.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+	+	+	+

strupr, _mbsupr, _wcsupr

See also   Example   Portability

Syntax

```
#include <string.h>
char *strupr(char *s);
wchar_t *_wcsupr(wchar_t *s);

#include <mbstring.h>
unsigned char *_mbsupr(unsigned char *s);
```

Description

Converts lowercase letters in a string to uppercase.

strupr converts lowercase letters in string *s* to uppercase according to the current locale's LC_CTYPE category. For the default C locale, the conversion is from lowercase letters *(a* to *z*) to uppercase letters (*A* to *Z*). No other characters are changed.

Return Value

strupr returns *s.*

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
strupr	+		+	+			+
_fstrupr	+		+				

strxfrm, wcsxfrm

See also Example Portability

Syntax

```
#include<string.h>
size_t strxfrm(char *target, const char *source, size_t n);
size_t wcsxfrm(wchar_t *target, const wchar_t *source, size_t n);
```

Description

Transforms a portion of a string to a specified collation.

strxfrm transforms the string pointed to by *source* into the string *target* for no more than *n* characters. The transformation is such that if the *strcmp* function is applied to the resulting strings, its return corresponds with the return values of the *strcoll* function.

No more than *n* characters, including the terminating null character, are copied to *target*.

strxfrm transforms a character string into a special string according to the current locale's LC_COLLATE category. The special string that is built can be compared with another of the same type, byte for byte, to achieve a locale-correct collation result. These special strings, which can be thought of as keys or tokenized strings, are not compatible across the different locales.

The tokens in the tokenized strings are built from the collation weights used by *strcoll* from the active locale's collation tables.

Processing stops only after all levels have been processed for the character string or the length of the tokenized string is equal to the maxlen parameter.

All redundant tokens are removed from each level's set of tokens.

The tokenized string buffer must be large enough to contain the resulting tokenized string. The length of this buffer depends on the size of the character string, the number of collation levels, the rules for each level and whether there are any special characters in the character string. Certain special characters can cause extra character processing of the string resulting in more space requirements. For example, the French character "oe" will take double the space for itself because in some locales, it expands to collation weights for each level. Substrings that have substitutions will also cause extra space requirements.

There is no safe formula to determine the required string buffer size, but at least (levels * string length) are required.

Return Value

Number of characters copied not including the terminating null character. If the value returned is greater than or equal to *n*, the content of *target* is indeterminate.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+	+	+	+

swab

Example Portability

Syntax

```
#include <stdlib.h>
void swab(char *from, char *to, int nbytes);
```

Description

Swaps bytes.

swab copies *nbytes* bytes from the *from* string to the *to* string. Adjacent even- and odd-byte positions are swapped. This is useful for moving data from one machine to another machine with a different byte order. *nbytes* should be even.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

system, _wsystem

See also   Example   Portability

Syntax

```
#include <stdlib.h>
int system(const char *command);
int _wsystem(const wchar_t *command);
```

Description

Issues an operating system command.

system invokes the operating system command processor to execute an operating system command, batch file, or other program named by the string *command*, from inside an executing C program.

To be located and executed, the program must be in the current directory or in one of the directories listed in the PATH string in the environment.

The COMSPEC environment variable is used to find the command processor program file, so that file need not be in the current directory.

Return Value

If *command* is a NULL pointer, *system* returns nonzero if a command processor is available.

If *command* is not a NULL pointer, *system* returns 0 if the command processor was successfully started.

If an error occurred, a -1 is returned and errno is set to one of the following:

ENOENT	Path or file function not found
ENOEXEC	Exec format error
ENOMEM	Not enough memory

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+		+			+

tan, tanl

See also Example Portability

Syntax

```
#include <math.h>
double tan(double x);
long double tanl(long double x);
```

Description

Calculates the tangent.

tan calculates the tangent. Angles are specified in radians.

tanl is the **long double** version; it takes a **long double** argument and returns a **long double** result. Error handling for these routines can be modified through the functions _matherr and _matherrl.

Return Value

tan and *tanl* return the tangent of *x*, *sin(x)/cos(x)*.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
tan	+	+	+	+	+	+	+
tanl	+		+	+			+

tanh, tanhl

See also Example Portability

Syntax

```
#include <math.h>
double tanh(double x);
long double tanhl(long double x);
```

Description

Calculates the hyperbolic tangent.

tanh computes the hyperbolic tangent, *sinh(x)/cosh(x)*.

tanhl is the **long double** version; it takes a **long double** argument and returns a **long double** result. Error handling for these functions can be modified through the functions *_matherr* and *_matherrl*.

Return Value

tanh and *tanhl* return the hyperbolic tangent of *x*.

Portability

	DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
tanh	+	+	+	+	+	+	+
tanhl	+		+	+			+

tell

See also Example Portability

Syntax

```
#include <io.h>
long tell(int handle);
```

Description

Gets the current position of a file pointer.

tell gets the current position of the file pointer associated with *handle* and expresses it as the number of bytes from the beginning of the file.

Return Value

tell returns the current file pointer position. A return of -1 (**long**) indicates an error, and the global variable errno is set to

EBADF	Bad file number

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

_tempnam, _wtempnam

See also Example Portability

Syntax

```
#include <stdio.h>
char *_tempnam(char *dir, char *prefix)
wchar_t *_wtempnam(wchar_t *dir, wchar_t *prefix)
```

Description

Creates a unique file name in specified directory.

The *_tempnam* function accepts single-byte or multibyte string arguments.

The *_tempnam* function creates a unique file name in arbitrary directories. The unique file is not actually created; *_tempnam* only verifies that it does not currently exist. It attempts to use the following directories, in the order shown, when creating the file name:

- The directory specified by the TMP environment variable.
- The *dir* argument to *_tempnam*.
- The *P_tmpdir* definition in stdio.h. If you edit stdio.h and change this definition, *_tempnam* will *not* use the new definition.
- The current working directory.

If any of these directories is NULL, or undefined, or does not exist, it is skipped.

The *prefix* argument specifies the first part of the file name; it cannot be longer than 5 characters, and cannot contain a period (.). A unique file name is created by concatenating the directory name, the *prefix*, and 6 unique characters. Space for the resulting file name is allocated with *malloc*; when this file name is no longer needed, the caller should call *free* to free it.

If you do create a temporary file using the name constructed by *_tempnam*, it is your responsibility to delete the file name (for example, with a call to *remove*). It is not deleted automatically. (*tmpfile* does delete the file name.)

Return Value

If *_tempnam* is successful, it returns a pointer to the unique temporary file name, which the caller can pass to *free* when it is no longer needed. Otherwise, if *_tempnam* cannot create a unique file name, it returns NULL.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

textattr

See also　　Example　　Portability

Syntax

```
#include <conio.h>
void textattr(int newattr);
```

Description

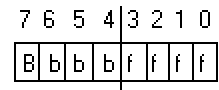
Sets text attributes.

Note: Do not use this function for Win32s or Win32 GUI applications.

textattr lets you set both the foreground and background colors in a single call. (Normally, you set the attributes with *textcolor* and *textbackground*.)

This function does not affect any characters currently onscreen; it affects only those characters displayed by functions (such as *cprintf*) performing text mode, direct video output *after* this function is called.

The color information is encoded in the *newattr* parameter as follows:



In this 8-bit *newattr* parameter:

- *ffff* is the 4-bit foreground color (0 to 15).
- *bbb* is the 3-bit background color (0 to 7).
- *B* is the blink-enable bit.

If the blink-enable bit is on, the character blinks. This can be accomplished by adding the constant BLINK to the attribute.

If you use the symbolic color constants defined in conio.h for creating text attributes with *textattr*, note the following limitations on the color you select for the background:

- You can select only one of the first eight colors for the background.
- You must shift the selected background color left by 4 bits to move it into the correct bit positions.

These symbolic constants are listed in the following table:

Symbolic constant	Numeric value	Foreground or background
BLACK	0	Both
BLUE	1	Both
GREEN	2	Both
CYAN	3	Both
RED	4	Both
MAGENTA	5	Both
BROWN	6	Both
LIGHTGRAY	7	Both
DARKGRAY	8	Foreground only
LIGHTBLUE	9	Foreground only
LIGHTGREEN	10	Foreground only
LIGHTCYAN	11	Foreground only
LIGHTRED	12	Foreground only
LIGHTMAGENTA	13	Foreground only
YELLOW	14	Foreground only

WHITE	15	Foreground only
BLINK	128	Foreground only

Return Value
None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+			+			+

textbackground

See also   Example   Portability

Syntax

```
#include <conio.h>
void textbackground(int newcolor);
```

Description

Selects new text background color.

Note: Do not use this function for Win32s or Win32 GUI applications.

textbackground selects the background color. This function works for functions that produce output in text mode directly to the screen. *newcolor* selects the new background color. You can set *newcolor* to an integer from 0 to 7, or to one of the symbolic constants defined in conio.h. If you use symbolic constants, you must include conio.h.

Once you have called *textbackground*, all subsequent functions using direct video output (such as *cprintf*) will use *newcolor*. *textbackground* does not affect any characters currently onscreen.

The following table lists the symbolic constants and the numeric values of the allowable colors:

Symbolic constant	Numeric value
BLACK	0
BLUE	1
GREEN	2
CYAN	3
RED	4
MAGENTA	5
BROWN	6
LIGHTGRAY	7

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+			+			+

textcolor

See also Example Portability

Syntax

```
#include <conio.h>
void textcolor(int newcolor);
```

Description

Selects new character color in text mode.

Note: Do not use this function for Win32s or Win32 GUI applications.

textcolor selects the foreground character color. This function works for the console output functions. *newcolor* selects the new foreground color. You can set *newcolor* to an integer as given in the table below, or to one of the symbolic constants defined in conio.h. If you use symbolic constants, you must include conio.h.

Once you have called *textcolor*, all subsequent functions using direct video output (such as *cprintf*) will use *newcolor*. *textcolor* does not affect any characters currently onscreen.

The following table lists the allowable colors (as symbolic constants) and their numeric values:

Symbolic constant	Numeric value
BLACK	0
BLUE	1
GREEN	2
CYAN	3
RED	4
MAGENTA	5
BROWN	6
LIGHTGRAY	7
DARKGRAY	8
LIGHTBLUE	9
LIGHTGREEN	10
LIGHTCYAN	11
LIGHTRED	12
LIGHTMAGENTA	13
YELLOW	14
WHITE	15
BLINK	128

You can make the characters blink by adding 128 to the foreground color. The predefined constant BLINK exists for this purpose.

For example:

```
textcolor(CYAN + BLINK);
```

Note: Some monitors do not recognize the intensity signal used to create the eight "light" colors (8-15). On such monitors, the light colors are displayed as their "dark" equivalents (0-7). Also, systems that do not display in color can treat these numbers as shades of one color, special patterns, or special attributes (such as underlined, bold, italics, and so on). Exactly what you'll see on such systems depends on your hardware.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+			+			+

textmode

See also   Example   Portability

Syntax

```
#include <conio.h>
void textmode(int newmode);
```

Description

Puts screen in text mode.

Note: Do not use this function for Win32s or Win32 GUI applications.

textmode selects a specific text mode.

You can give the text mode (the argument *newmode*) by using a symbolic constant from the enumeration type *text_modes* (defined in conio.h).

The most commonly used *text_modes* type constants and the modes they specify are given in the following table. Some additional values are defined in conio.h.

Symbolic Constant	Text Mode
LASTMODE	Previous text mode
BW40	Black and white, 40 columns
C40	Color, 40 columns
BW80	Black and white, 80 columns
C80	Color, 80 columns
MONO	Monochrome, 80 columns
C4350	EGA 43-line and VGA 50-line modes

When *textmode* is called, the current window is reset to the entire screen, and the current text attributes are reset to normal, corresponding to a call to *normvideo*.

Specifying LASTMODE to *textmode* causes the most recently selected text mode to be reselected.

textmode should be used only when the screen or window is in text mode (presumably to change to a different text mode). This is the only context in which *textmode* should be used. When the screen is in graphics mode, use *restorecrtmode* instead to escape temporarily to text mode.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+			+			+

time

See also Example Portability

Syntax

```
#include <time.h>
time_t time(time_t *timer);
```

Description

Gets time of day.

time gives the current time, in seconds, elapsed since 00:00:00 GMT, January 1, 1970, and stores that value in the location pointed to by *timer*, provided that *timer* is not a NULL pointer.

Return Value

time returns the elapsed time in seconds.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

tmpfile

See also　　Example　　Portability

Syntax

```
#include <stdio.h>
FILE *tmpfile(void);
```

Description

Opens a "scratch" file in binary mode.

tmpfile creates a temporary binary file and opens it for update (*w* + *b*). If you do not change the directory after creating the temporary file, the file is automatically removed when it's closed or when your program terminates.

Return Value

tmpfile returns a pointer to the stream of the temporary file created. If the file can't be created, *tmpfile* returns NULL.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

tmpnam, _wtmpnam

See also Example Portability

Syntax

```
#include <stdio.h>
char *tmpnam(char *s);
wchar_t *_wtmpnam(wchar_t *s);
```

Description

Creates a unique file name.

tmpnam creates a unique file name, which can safely be used as the name of a temporary file. *tmpnam* generates a different string each time you call it, up to TMP_MAX times. TMP_MAX is defined in stdio.h as 65,535.

The parameter to *tmpnam*, *s*, is either null or a pointer to an array of at least *L_tmpnam* characters. *L_tmpnam* is defined in stdio.h. If *s* is NULL, *tmpnam* leaves the generated temporary file name in an internal static object and returns a pointer to that object. If *s* is not NULL, *tmpnam* overwrites the internal static object and places its result in the pointed-to array, which must be at least *L_tmpnam* characters long, and returns *s*.

If you do create such a temporary file with *tmpnam*, it is your responsibility to delete the file name (for example, with a call to *remove*). It is not deleted automatically. (*tmpfile* does delete the file name.)

Return Value

If *s* is null, *tmpnam* returns a pointer to an internal static object. Otherwise, *tmpnam* returns *s*.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

toascii

Example Portability

Syntax

```
#include <ctype.h>
int toascii(int c);
```

Description

Translates characters to ASCII format.

toascii is a macro that converts the integer *c* to ASCII by clearing all but the lower 7 bits; this gives a value in the range 0 to 127.

Return Value

toascii returns the converted value of *c*.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

_tolower

Example Portability

Syntax

```
#include <ctype.h>
int _tolower(int ch);
```

Description

_*tolower* is a macro that does the same conversion as *tolower*, except that it should be used only when *ch* is known to be uppercase (*AZ*).

To use _*tolower*, you must include ctype.h.

Return Value

_*tolower* returns the converted value of *ch* if it is uppercase; otherwise, the result is undefined.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

tolower, _mbctolower, towlower

Example Portability

Syntax

```
#include <ctype.h>
int tolower(int ch);
int towlower(wint_t ch);  // Unicode version

#include <mbstring.h>
unsigned int _mbctolower(unsigned int c);
```

Description

Translates characters to lowercase.

tolower is a function that converts an integer *ch* (in the range EOF to 255) to its lowercase value (*a* to *z*; if it was uppercase, *A* to *Z*). All others are left unchanged.

Return Value

tolower returns the converted value of *ch* if it is uppercase; it returns all others unchanged.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

_toupper

Example Portability

Syntax

```
#include <ctype.h>
int _toupper(int ch);
```

Description

Translates characters to uppercase.

_toupper is a macro that does the same conversion as toupper, except that it should be used only when *ch* is known to be lowercase (*a* to *z*).

To use *_toupper*, you must include ctype.h.

Return Value

_toupper returns the converted value of *ch* if it is lowercase; otherwise, the result is undefined.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

toupper, _mbctoupper, towupper

Example Portability

Syntax

```
#include <ctype.h>
int toupper(int ch);
int towupper(wint_t ch);  // Unicode version

#include <mbstring.h>
unsigned int _mbctoupper(unsigned int c);
```

Description

Translates characters to uppercase.

toupper is a function that converts an integer *ch* (in the range EOF to 255) to its uppercase value (*A* to *Z*; if it was lowercase, *a* to *z*). All others are left unchanged.

towupper is the Unicode version of *toupper*. It is available when Unicode is defined.

Return Value

toupper returns the converted value of *ch* if it is lowercase; it returns all others unchanged.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

_tzset, _wtzset

See also Example Portability

Syntax

```
#include <time.h>
void _tzset(void)
void _wtzset(void)
```

Description

Sets value of global variables _daylight, _timezone, and _tzname.

_tzset is available on XENIX systems.

_tzset sets the *_daylight, _timezone*, and *_tzname* global variables based on the environment variable *TZ*. *_wtzset* sets the *_daylight, _timezone*, and *_wtzname* global variables. The library functions *ftime* and *localtime* use these global variables to adjust Greenwich Mean Time (GMT) to the local time zone. The format of the *TZ* environment string is:

```
TZ = zzz[+/-]d[d][lll]
```

where *zzz* is a three-character string representing the name of the current time zone. All three characters are required. For example, the string "PST" could be used to represent Pacific standard time.

[+/-]*d*[*d*] is a required field containing an optionally signed number with 1 or more digits. This number is the local time zone's difference from GMT in hours. Positive numbers adjust westward from GMT. Negative numbers adjust eastward from GMT. For example, the number 5 = EST, +8 = PST, and -1 = continental Europe. This number is used in the calculation of the global variable *_timezone*. *_timezone* is the difference in seconds between GMT and the local time zone.

lll is an optional three-character field that represents the local time zone, daylight saving time. For example, the string "PDT" could be used to represent pacific daylight saving time. If this field is present, it causes the global variable *_daylight* to be set nonzero. If this field is absent, *_daylight* is set to zero.

If the *TZ* environment string isn't present or isn't in the preceding form, a default *TZ* = "EST5EDT" is presumed for the purposes of assigning values to the global variables *_daylight, _timezone*, and *_tzname*. On a Win32 system, none of these global variables are set if *TZ* is null.

The global variables *_tzname*[0] and *_wtzname*[1] point to a three-character string with the value of the time-zone name from the *TZ* environment string. *_tzname*[1] and *_wtzname*[1] point to a three-character string with the value of the daylight saving time-zone name from the *TZ* environment string. If no daylight saving name is present, *_tzname*[1] and *_wtzname*[1] point to a null string.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

ultoa, _ultow

See also   Example   Portability

Syntax

```
#include <stdlib.h>
char *ultoa(unsigned long value, char *string, int radix);
wchar_t *_ultow(unsigned long value, wchar_t *string, int radix);
```

Description

Converts an **unsigned long** to a string.

ultoa converts *value* to a null-terminated string and stores the result in *string. value* is an **unsigned long**.

radix specifies the base to be used in converting *value*; it must be between 2 and 36, inclusive. *ultoa* performs no overflow checking, and if *value* is negative and *radix* equals 10, it does not set the minus sign.

Note: The space allocated for *string* must be large enough to hold the returned string, including the terminating null character (\0). *ultoa* can return up to 33 bytes.

Return Value

ultoa returns *string*.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

umask

See also Example Portability

Syntax

```
#include <io.h>
unsigned umask(unsigned mode);
```

Description

Sets file read/write permission mask.

The *umask* function sets the access permission mask used by *open* and *creat*. Bits that are set in *mode* will be cleared in the access permission of files subsequently created by *open* and *creat*.

The *mode* can have one of the following values, defined in sys\stat.h:

Value of mode	Access permission
S_IWRITE	Permission to write
S_IREAD	Permission to read
S_IREAD|S_IWRITE	Permission to read and write

Return Value

The previous value of the mask. There is no error return.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

ungetc, ungetwc

See also   Example   Portability

Syntax

```
#include <stdio.h>
int ungetc(int c, FILE *stream);
wint_t ungetwc(wint_t c, FILE *stream);
```

Description

Pushes a character back into input stream.

Note: Do not use this function for Win32s or Win32 GUI applications.

ungetc pushes the character *c* back onto the named input *stream*, which must be open for reading. This character will be returned on the next call to *getc* or *fread* for that *stream*. One character can be pushed back in all situations. A second call to *ungetc* without a call to *getc* will force the previous character to be forgotten. A call to *fflush, fseek, fsetpos,* or *rewind* erases all memory of any pushed-back characters.

Return Value

On success, *ungetc* returns the character pushed back.

On eror, it returns EOF.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

ungetch

See also Example Portability

Syntax

```
#include <conio.h>
int ungetch(int ch);
```

Description

Pushes a character back to the keyboard buffer.

Note: Do not use this function for Win32s or Win32 GUI applications.

ungetch pushes the character *ch* back to the console, causing *ch* to be the next character read. The *ungetch* function fails if it is called more than once before the next read.

Return Value

On success, *ungetch* returns the character *ch*.

On error, it returns EOF.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+		+			+

unixtodos

See also   Example   Portability

Syntax

```
#include <dos.h>
void unixtodos(long time, struct date *d, struct time *t);
```

Description

Converts date and time from UNIX to DOS format.

unixtodos converts the UNIX-format time given in *time* to DOS format and fills in the *date* and *time* structures pointed to by *d* and *t*.

time must not represent a calendar time earlier than Jan. 1, 1980 00:00:00.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

_unlink, _wunlink

See also Example Portability

Syntax

```
#include <io.h>
int _unlink(const char *filename);
int _wunlink(const wchar_t *filename);
```

Description

Deletes a file.

_unlink deletes a file specified by *filename*. Any drive, path, and file name can be used as a *filename*. Wildcards are not allowed.

Read-only files cannot be deleted by this call. To remove read-only files, first use chmod or _rtl_chmod to change the read-only attribute.

Note: If the file is open, it must be closed before unlinking it.

_wunlink is the Unicode version of *_wunlink*. The Unicode version accepts a filename that is a *wchar_t* character string. Otherwise, the functions perform identically.

Return Value

On success, *_unlink* returns 0.

On error, it returns -1 and sets the global variable errno to one of the following values:

EACCES	Permission denied
ENOENT	Path or file name not found

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

unlock

See also Example Portability

Syntax

```
#include <io.h>
int unlock(int handle, long offset, long length);
```

Description

Releases file-sharing locks.

unlock provides an interface to the operating system file-sharing mechanism. *unlock* removes a lock previously placed with a call to *lock*. To avoid error, all locks must be removed before a file is closed. A program must release all locks before completing.

Return Value

On success, *unlock* returns 0

O error, it returns -1.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

_utime, _wutime

See also   Example   Portability

Syntax

```
#include <utime.h>
int _utime(char *path, struct utimbuf *times);
int _wutime(wchar_t *path, struct _utimbuf *times);
```

Description

Sets file time and date.

_utime sets the modification time for the file *path*. The modification time is contained in the *utimbuf* structure pointed to by *times*. This structure is defined in utime.h, and has the following format:

```
struct utimbuf {
   time_t   actime;     /* access time */
   time_t   modtime;    /* modification time */
   };
```

The FAT (file allocation table) file system supports only a modification time; therefore, on FAT file systems *_utime* ignores *actime* and uses only *modtime* to set the file's modification time.

If *times* is NULL, the file's modification time is set to the current time.

_wutime is the Unicode version of *_utime*. The Unicode version accepts a filename that is a *wchar_t* character string. Otherwise, the functions perform identically.

Return Value

On sucess, *_utime* returns 0.

On error, it returns -1, and sets the global variable errno to one of the following values:

EACCES	Permission denied
EMFILE	Too many open files
ENOENT	Path or file name not found

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

va_arg, va_end, va_start

See also   Example   Portability

Syntax

```
#include <stdarg.h>
void va_start(va_list ap, lastfix);
type va_arg(va_list ap, type);
void va_end(va_list ap);
```

Description

Implement a variable argument list.

Some C functions, such as *vfprintf* and *vprintf*, take variable argument lists in addition to taking a number of fixed (known) parameters. The *va_arg, va_end*, and *va_start* macros provide a portable way to access these argument lists. They are used for stepping through a list of arguments when the called function does not know the number and types of the arguments being passed.

The header file stdarg.h declares one type (**va_list**) and three macros (*va_start, va_arg*, and *va_end*).

- **va_list**: This array holds information needed by *va_arg* and *va_end*. When a called function takes a variable argument list, it declares a variable *ap* of type **va_list**.
- *va_start*: This routine (implemented as a macro) sets *ap* to point to the first of the variable arguments being passed to the function. *va_start* must be used before the first call to *va_arg* or *va_end*.
- *va_start* takes two parameters: *ap* and *lastfix*. (*ap* is explained under *va_list* in the preceding paragraph; *lastfix* is the name of the last fixed parameter being passed to the called function.)
- *va_arg*: This routine (also implemented as a macro) expands to an expression that has the same type and value as the next argument being passed (one of the variable arguments). The variable *ap* to *va_arg* should be the same *ap* that *va_start* initialized.

Note: Because of default promotions, you cannot use **char, unsigned char**, or **float** types with *va_arg*.

The first time *va_arg* is used, it returns the first argument in the list. Each successive time *va_arg* is used, it returns the next argument in the list. It does this by first dereferencing *ap*, and then incrementing *ap* to point to the following item. *va_arg* uses the *type* to both perform the dereference and to locate the following item. Each successive time *va_arg* is invoked, it modifies *ap* to point to the next argument in the list.

- *va_end*: This macro helps the called function perform a normal return. *va_end* might modify *ap* in such a way that it cannot be used unless *va_start* is recalled. *va_end* should be called after *va_arg* has read all the arguments; failure to do so might cause strange, undefined behavior in your program.

Return Value

va_start and *va_end* return no values; *va_arg* returns the current argument in the list (the one that *ap* is pointing to).

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

vfprintf, vfwprintf

See also   Example   Portability

Syntax

```
#include <stdio.h>
int vfprintf(FILE *stream, const char *format, va_list arglist);
int vfwprintf(FILE *stream, const wchar_t *format, va_list arglist);
```

Description

Writes formatted output to a stream.

The *v...printf* functions are known as *alternate entry points* for the *...printf* functions. They behave exactly like their *...printf* counterparts, but they accept a pointer to a list of arguments instead of an argument list.

For details on format specifiers, see Printf Format Specifiers.

vfprintf accepts a pointer to a series of arguments, applies to each argument a format specifier contained in the format string pointed to by *format*, and outputs the formatted data to a stream. There must be the same number of format specifiers as arguments.

Return Value

On success, *vfprintf* returns the number of bytes output.

On error, it returns EOF.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

vfscanf

See also Example Portability

Syntax

```
#include <stdio.h>
int vfscanf(FILE *stream, const char *format,va_list arglist);
```

Description

Scans and formats input from a stream.

The *v...scanf* functions are known as *alternate entry points* for the *...scanf* functions. They behave exactly like their *...scanf* counterparts but they accept a pointer to a list of arguments instead of an argument list.

For details on format specifiers, see Scanf Format Specifiers.

vfscanf scans a series of input fields one character at a time reading from a stream. Then each field is formatted according to a format specifier passed to *vfscanf* in the format string pointed to by *format*. Finally *vfscanf* stores the formatted input at an address passed to it as an argument following *format*. There must be the same number of format specifiers and addresses as there are input fields. *vfscanf* might stop scanning a particular field before it reaches the normal end-of-field (whitespace) character or it might terminate entirely for a number of reasons. See *scanf* for a discussion of possible causes.

Return Value

vfscanf returns the number of input fields successfully scanned converted and stored; the return value does not include scanned fields that were not stored. If no fields were stored the return value is 0.

If *vfscanf* attempts to read at end-of-file the return value is EOF.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

vprintf, vfwprintf

See also Example Portability

Syntax

```
#include <stdarg.h>
int vprintf(const char *format, va_list arglist);
int vwprintf(const wchar_t *format, va_list arglist);
```

Description

Writes formatted output to stdout.

Note: Do not use this function for Win32s or Win32 GUI applications.

The *v...printf* functions are known as *alternate entry points* for the *...printf* functions. They behave exactly like their *...printf* counterparts, but they accept a pointer to a list of arguments instead of an argument list.

For details on format specifiers, see Printf Format Specifiers.

vprintf accepts a pointer to a series of arguments, applies to each a format specifier contained in the format string pointed to by *format*, and outputs the formatted data to stdout. There must be the same number of format specifiers as arguments.

Note: When you use the SS!=DS flag in 16-bit applications, *vprintf* assumes that the address being passed is in the SS segment.

Return Value

vprint returns the number of bytes output. In the event of error, *vprint* returns EOF.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+		+	+		+

vscanf

See also   Example   Portability

Syntax

```
#include <stdarg.h>
int vscanf(const char *format, va_list arglist);
```

Description

Scans and formats input from stdin.

Note: Do not use this function for Win32s or Win32 GUI applications.

The *v...scanf* functions are known as *alternate entry points* for the *...scanf* functions. They behave exactly like their *...scanf* counterparts, but they accept a pointer to a list of arguments instead of an argument list.

For details on format specifiers, see Scanf Format Specifiers.

vscanf scans a series of input fields, one character at a time, reading from stdin. Then each field is formatted according to a format specifier passed to *vscanf* in the format string pointed to by *format*. Finally, *vscanf* stores the formatted input at an address passed to it as an argument following *format*. There must be the same number of format specifiers and addresses as there are input fields.

vscanf might stop scanning a particular field before it reaches the normal end-of-field (whitespace) character, or it might terminate entirely, for a number of reasons. See *scanf* for a discussion of possible causes.

Return Value

vscanf returns the number of input fields successfully scanned, converted, and stored; the return value does not include scanned fields that were not stored. If no fields were stored, the return value is 0.

If *vscanf* attempts to read at end-of-file, the return value is EOF.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

vsprintf, vswprintf

See also Example Portability

Syntax

```
#include <stdarg.h>
int vsprintf(char *buffer, const char *format, va_list arglist);
int vswprintf(wchar_t *buffer, const wchar_t *format, va_list arglist);
```

Description

Writes formatted output to a string.

The *v...printf* functions are known as *alternate entry points* for the *...printf* functions. They behave exactly like their *...printf* counterparts, but they accept a pointer to a list of arguments instead of an argument list.

For details on format specifiers, see Printf Format Specifiers.

vsprintf accepts a pointer to a series of arguments, applies to each a format specifier contained in the format string pointed to by *format*, and outputs the formatted data to a string. There must be the same number of format specifiers as arguments.

Return Value

vsprintf returns the number of bytes output. In the event of error, *vsprintf* returns EOF.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

vsscanf

See also   Example   Portability

Syntax

```
#include <stdarg.h>
int vsscanf(const char *buffer, const char *format, va_list arglist);
```

Description

Scans and formats input from a stream.

The *v...scanf* functions are known as *alternate entry points* for the *...scanf* functions. They behave exactly like their *...scanf* counterparts, but they accept a pointer to a list of arguments instead of an argument list.

For details on format specifiers, see Scanf Format Specifiers.

vsscanf scans a series of input fields, one character at a time, reading from a stream. Then each field is formatted according to a format specifier passed to *vsscanf* in the format string pointed to by *format*. Finally, *vsscanf* stores the formatted input at an address passed to it as an argument following *format*. There must be the same number of format specifiers and addresses as there are input fields.

vsscanf might stop scanning a particular field before it reaches the normal end-of-field (whitespace) character, or it might terminate entirely, for a number of reasons. See scanf for a discussion of possible causes.

Return Value

vsscanf returns the number of input fields successfully scanned, converted, and stored; the return value does not include scanned fields that were not stored. If no fields were stored, the return value is 0.

If *vsscanf* attempts to read at end-of-string, the return value is EOF.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

wait

See also   Portability

Syntax

```
#include <process.h>
int wait(int *statloc);
```

Description

Waits for one or more child processes to terminate.

The *wait* function waits for one or more child processes to terminate. The child processes must be those created by the calling program; *wait* cannot wait for grandchildren (processes spawned by child processes). If *statloc* is not NULL, it points to location where *wait* will store the termination status.

If the child process terminated normally (by calling *exit*, or returning from *main*), the termination status word is defined as follows:

Bits 0-7	Zero.
Bits 8-15	The least significant byte of the return code from the child process. This is the value that is passed to *exit*, or is returned from *main*. If the child process simply exited from *main* without returning a value, this value will be unpredictable. If the child process terminated abnormally, the termination status word is defined as follows:
Bits 0-7	Termination information about the child: 1 Critical error abort. 2 Execution fault, protection exception. 3 External termination signal.
Bits 8-15	Zero.

Return Value

When *wait* returns after a normal child process termination it returns the process ID of the child.

When *wait* returns after an abnormal child termination it returns -1 to the parent and sets *errno* to EINTR.

If *wait* returns without a child process completion it returns a -1 value and sets errno to

ECHILD	No child process exists

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
			+			+

wcstombs

Example Portability

Syntax

```
#include <stdlib.h>
size_t wcstombs(char *s, const wchar_t *pwcs, size_t n);
```

Description

Converts a **wchar_t** array into a multibyte string.

wcstombs converts the type **wchar_t** elements contained in *pwcs* into a multibyte character string *s*. The process terminates if either a null character or an invalid multibyte character is encountered.

No more than *n* bytes are modified. If *n* number of bytes are processed before a null character is reached, the array *s* is not null terminated.

The behavior of *wcstombs* is affected by the setting of LC_CTYPE category of the current locale.

Return Value

If an invalid multibyte character is encountered, *wcstombs* returns (size_t) -1. Otherwise, the function returns the number of bytes modified, not including the terminating code, if any.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

wctomb

[Example](#) [Portability](#)

Syntax

```
#include <stdlib.h>
int wctomb(char *s, wchar_t wc);
```

Description

Converts **wchar_t** code to a multibyte character.

If *s* is not null, *wctomb* determines the number of bytes needed to represent the multibyte character corresponding to *wc* (including any change in shift state). The multibyte character is stored in *s*. At most MB_CUR_MAX characters are stored. If the value of *wc* is zero, *wctomb* is left in the initial state.

The behavior of *wctomb* is affected by the setting of LC_CTYPE category of the current locale.

Return Value

If *s* is a NULL pointer, *wctomb* returns a nonzero value if multibyte character encodings do have state-dependent encodings, and a zero value if they do not.

If *s* is not a NULL pointer, *wctomb* returns -1 if the *wc* value does not represent a valid multibyte character. Otherwise, *wctomb* returns the number of bytes that are contained in the multibyte character corresponding to *wc*. In no case will the return value be greater than the value of MB_CUR_MAX macro.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+	+	+	+

wherex

See also   Example   Portability

Syntax

```
#include <conio.h>
int wherex(void);
```

Description

Gives horizontal cursor position within window.

Note: Do not use this function for Win32s or Win32 GUI applications.

wherex returns the x-coordinate of the current cursor position (within the current text window).

Return Value

wherex returns an integer in the range 1 to the number of columns in the current video mode.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

wherey

See also Example Portability

Syntax

```
#include <conio.h>
int wherey(void);
```

Description

Gives vertical cursor position within window.

Note: Do not use this function for Win32s or Win32 GUI applications.

wherey returns the y-coordinate of the current cursor position (within the current text window).

Return Value

wherey returns an integer in the range 1 to the number of rows in the current video mode.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

window

See also   Example   Portability

Syntax

```
#include <conio.h>
void window(int left, int top, int right, int bottom);
```

Description

Defines active text mode window.

Note: Do not use this function for Win32s or Win32 GUI applications.

window defines a text window onscreen. If the coordinates are in any way invalid, the call to *window* is ignored.

left and *top* are the screen coordinates of the upper left corner of the window.

right and *bottom* are the screen coordinates of the lower right corner.

The minimum size of the text window is one column by one line. The default window is full screen, with the coordinates:

```
1,1,C,R
```

where C is the number of columns in the current video mode, and R is the number of rows.

Return Value

None.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+			+			+

See Also

lseek

_rtl_read

write

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+			+

write

See also Example Portability

Syntax

```
#include <io.h>
int write(int handle, void *buf, unsigned len);
```

Description

Writes to a file.

write writes a buffer of data to the file or device named by the given *handle*. *handle* is a file handle obtained from a *creat, open, dup,* or *dup2* call.

This function attempts to write *len* bytes from the buffer pointed to by *buf* to the file associated with *handle*. Except when *write* is used to write to a text file, the number of bytes written to the file will be no more than the number requested. The maximum number of bytes that *write* can write is UINT_MAX -1, because UINT_MAX is the same as -1, which is the error return indicator for *write*. On text files, when *write* sees a linefeed (LF) character, it outputs a CR/LF pair. UINT_MAX is defined in limits.h.

If the number of bytes actually written is less than that requested, the condition should be considered an error and probably indicates a full disk. For disks or disk files, writing always proceeds from the current file pointer. For devices, bytes are sent directly to the device. For files opened with the O_APPEND option, the file pointer is positioned to EOF by *write* before writing the data.

Return Value

write returns the number of bytes written. A *write* to a text file does not count generated carriage returns. In case of error, *write* returns -1 and sets the global variable errno to one of the following values:

EACCES	Permission denied
EBADF	Bad file number

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+	+	+	+			+

/* abs example */

```
#include <stdio.h>
#include <math.h>

int main(void)
{
  int number = -1234;

  printf("number: %d  absolute value: %d\n", number, abs(number));
  return 0;
}
```

/* cabs example */

```
#include <stdio.h>
#include <math.h>

#ifdef __cplusplus
  #include <complex.h>
#endif

#ifdef __cplusplus /* if C++, use class complex */

  void print_abs(void)
  {
    complex z(1.0, 2.0);
    double  absval;

    absval = abs(z);
    printf("The absolute value of %.2lfi %.2lfj is %.2lf",
      real(z), imag(z), absval);
  }

#else  /* below function is for C (and not C++) */

  void print_abs(void)
  {
    struct complex z;
    double absval;

    z.x = 2.0;
    z.y = 1.0;
    absval = cabs(z);

    printf("The absolute value of %.2lfi %.2lfj is %.2lf",
      z.x, z.y, absval);
  }

#endif

int main(void)
{
  print_abs();
  return 0;
}
```

/* fabs example */

```
#include <stdio.h>
#include <math.h>

int main(void)
{
   float  number = -1234.0;

   printf("number: %f  absolute value: %f\n", number, fabs(number));
   return 0;
}
```

/* labs example */

```
#include <stdio.h>
#include <math.h>

int main(void)
{
   long result;
   long x = -12345678L;

   result= labs(x);
   printf("number: %ld abs value: %ld\n", x, result);

   return 0;
}
```

/* acos example */

```
#include <stdio.h>
#include <math.h>

int main(void)
{
  double result;
  double x = 0.5;

  result = acos(x);
  printf("The arc cosine of %lf is %lf\n", x, result);
  return 0;
}
```

/* asin example */

```
#include <stdio.h>
#include <math.h>

int main(void)
{
   double result;
   double x = 0.5;
   result = asin(x);
   printf("The arc sin of %lf is %lf\n", x, result);
  return(0);
}
```

/* atan example */

```
#include <stdio.h>
#include <math.h>

int main(void)
{
   double result;
   double x = 0.5;

   result = atan(x);
   printf("The arc tangent of %lf is %lf\n", x, result);
  return(0);
}
```

/* atan2 example */

```
#include <stdio.h>
#include <math.h>

int main(void)
{
   double result;
   double x = 90.0, y = 45.0;

   result = atan2(y, x);
   printf("The arc tangent ratio of %lf is %lf\n", (y / x), result);
   return 0;
}
```

/* alloca example */

```
#include <malloc.h>
#include <stdio.h>
#include <stdlib.h>

void test(int a)
{
   char *newstack;
   int len = a;
   char dummy[1];

  dummy[0] = 0;          /* force good stack frame */
   printf("SP before calling alloca(0x%X) = 0x%X\n",len,_SP);
   newstack = (char *) alloca(len);
  printf("SP after calling alloca = 0x%X\n",_SP);
   if (newstack)
      printf("Alloca(0x%X) returned %p\n",len,newstack);
   else
      printf("Alloca(0x%X) failed\n",len);
}
void main()
{
   test(256);
   test(16384);
}
```

/* asctime example */

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

int main(void)
{
  struct tm t;
  char str[80];

  /* sample loading of tm structure  */

  t.tm_sec    = 1;  /* Seconds */
  t.tm_min    = 30; /* Minutes */
  t.tm_hour   = 9;  /* Hour */
  t.tm_mday   = 22; /* Day of the Month  */
  t.tm_mon    = 11; /* Month */
  t.tm_year   = 56; /* Year - does not include century */
  t.tm_wday   = 4;  /* Day of the week  */
  t.tm_yday   = 0;  /* Does not show in asctime */
  t.tm_isdst  = 0;  /* Is Daylight SavTime; does not show in asctime */

  /* converts structure to null terminated string */

  strcpy(str, asctime(&t));
  printf("%s\n", str);

  return 0;
}
```

/* ctime example */

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

int main(void)
{
  time_t t;

  time(&t);
  printf("Today's date and time: %s\n", ctime(&t));
  return 0;
}
```

/* _beginthread example */

```
#include <stdio.h>
#include <errno.h>
#include <stddef.h>     /* _threadid variable */
#include <process.h>    /* _beginthread, _endthread */
#include <time.h>       /* time, _ctime */

void thread_code(void *threadno)
{
    time_t t;

    time(&t);
    printf("Executing thread number %d, ID = %d, time = %s\n",
        (int)threadno, _threadid, ctime(&t));
    _endthread();
}

void start_thread(int i)
{
    int thread_id;

#if     defined(__WIN32__)
    if ((thread_id = _beginthread(thread_code,4096,(void *)i)) == (unsigned
  long)-1)
#else
    if ((thread_id = _beginthread(thread_code,4096,(void *)i)) == -1)
#endif
    {
        printf("Unable to create thread %d, errno = %d\n",i,errno);
        return;
    }
    printf("Created thread %d, ID = %ld\n",i,thread_id);
}

int main(void)
{
    int i;

    for (i = 1; i < 20; i++)
        start_thread(i);
    printf("Hit ENTER to exit main thread.\n");
    getchar();
    return 0;
}
```

/* beginthreadNT example */

```
#include        <windows.h>
#include        <process.h>
#include        <stdio.h>
#include        <conio.h>


/*      This function acts as the 'main' function for each new thread.
static  void    threadMain(void *arg) */
{
        printf("Thread %2d has an ID of %u\n", (int)arg,
  GetCurrentThreadId());

        _endthread();
}


int     main(void)
{
        #define NTHREADS        25

        HANDLE  hThreads[NTHREADS];
        int     i;

        //      Create NTHREADS inheritable threads that are initially
        //      suspended and that will run starting at threadMain().
        //      at threadMain().
        for     (i = 0; i < NTHREADS; i++)
        {
                SECURITY_ATTRIBUTES     sa =
                {
                        sizeof(SECURITY_ATTRIBUTES),    // structure size
                        0,                              // No security
  descriptor
                        TRUE,                           // Thread handle is
  inheritable
                };

                DWORD   threadId;

                hThreads[i] = (HANDLE)_beginthreadNT(
                                        threadMain,             // Thread
  starting address
                                        4096,                   // Thread
  stack size
                                        (void *)i,              // Thread
  start argument
                                        &sa,                    // Thread
  security
                                        CREATE_SUSPENDED,       // Create
  in suspended state
                                        &threadId);             // Thread
  ID.

                if      (hThreads[i] == INVALID_HANDLE_VALUE)
                {
```

```
                        MessageBox(0, "Thread Creation Failed", "Error",
  MB_OK);
                        return  1;
                }

                printf("Created thread %2d with an ID of %u\n", i,
  threadId);
        }

        printf("\nPress a key to thaw all threads\n\n");
        getch();

        //      Resume the suspended threads.
        for     (i = 0; i < NTHREADS; i++)
                ResumeThread(hThreads[i]);

        //      Wait for all threads to finish execution.
        WaitForMultipleObjects(NTHREADS,        // Number of objects to
  wait for
                               hThreads,        // The objects to wait for
                               TRUE,            // Wait for all objects
                               INFINITE);       // No timeout

        //      Close all of the thread handles.
        for     (i = 0; i < NTHREADS; i++)
                CloseHandle(hThreads[i]);

        return  0;
}
```

/* biosequip example */

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

#define CO_PROCESSOR_MASK 0x0002

int main(void)
{
  int equip_check;

   /* get the current equipment configuration */
   equip_check = biosequip();

   /* check to see if there is a coprocessor installed */
   if (equip_check & CO_PROCESSOR_MASK)
      printf("There is a math coprocessor installed.\n");
   else
      printf("No math coprocessor installed.\n");
   return 0;
}
```

/* _bios_equiplist example */

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

#define CO_PROCESSOR_MASK 0x0002

int main(void)
{
  unsigned equip_check;

  /* get the current equipment configuration */
  equip_check = _bios_equiplist();

  /* check to see if there is a coprocessor installed */
  if (equip_check & CO_PROCESSOR_MASK)
    printf("There is a math coprocessor installed.\n");
   else
    printf("No math coprocessor installed.\n");
  return 0;
}
```

/* biosmemory example */

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

int main(void)
{
   int memory_size;

   memory_size = biosmemory();  /* returns value up to 640K */
   printf("RAM size = %dK\n",memory_size);
   return 0;
}
```

/* _bios_memsize example */

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

int main(void)
{
  unsigned   memory_size;

  memory_size = _bios_memsize();   /* returns value up to 640K */

  printf("RAM size = %dK\n", memory_size);

  return 0;
}
```

/* biostime example */

```
#include <stdio.h>
#include <bios.h>
#include <time.h>
#include <conio.h>

int main(void)
{
  long bios_time;
   clrscr();
   printf("The number of clock ticks since midnight is:\n");
   printf("The number of seconds since midnight is:\n");
   printf("The number of minutes since midnight is:\n");
   printf("The number of hours since midnight is:\n");
   printf("\nPress any key to stop:");
   while(!kbhit())
   {
      bios_time = biostime(0, 0L);

      gotoxy(50, 1);
      printf("%lu", bios_time);

      gotoxy(50, 2);
      printf("%.4f", bios_time / _BIOS_CLK_TCK);

      gotoxy(50, 3);
      printf("%.4f", bios_time / _BIOS_CLK_TCK / 60);

      gotoxy(50, 4);
      printf("%.4f", bios_time / _BIOS_CLK_TCK / 3600);
   }
   return 0;
}
```

/* _bios_timeofday example */

```
#include <bios.h>
#include <time.h>
#include <conio.h>
#include <stdio.h>

int main(void)
{
  long bios_time;
  clrscr();
  printf("The number of clock ticks since midnight is:\n");
  printf("The number of seconds since midnight is:\n");
  printf("The number of minutes since midnight is:\n");
  printf("The number of hours since midnight is:\n");
  printf("\nPress any key to stop:");
  while(!kbhit())
  {
    _bios_timeofday(_TIME_GETCLOCK, &bios_time);
    gotoxy(50, 1);
    printf("%lu", bios_time);
    gotoxy(50, 2);
    printf("%.4f", bios_time / CLK_TCK);
    gotoxy(50, 3);
    printf("%.4f", bios_time / CLK_TCK / 60);
    gotoxy(50, 4);
    printf("%.4f", bios_time / CLK_TCK / 3600);
  }
  return 0;
}
```

/* bsearch example */

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

typedef int (*fptr)(const void*, const void*);

#define NELEMS(arr) (sizeof(arr) / sizeof(arr[0]))

int numarray[] = {123, 145, 512, 627, 800, 933};

int numeric (const int *p1, const int *p2)
{
   return(*p1 - *p2);
}

#pragma argsused
int lookup(int key)
{
   int  *itemptr;

   /* The cast of (int(*)(const void *,const void*))
      is needed to avoid a type mismatch error at
      compile time */
   itemptr = (int *) bsearch (&key, numarray, NELEMS(numarray),
      sizeof(int), (fptr)numeric);
   return (itemptr != NULL);
}

int main(void)
{
   if (lookup(512))
      printf("512 is in the table.\n");
   else
      printf("512 isn't in the table.\n");

   return 0;
}
```

/* lfind example */

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

int compare(int *x, int *y)
{
   return( *x - *y );
}

int main(void)
{
   int array[5] = {35, 87, 46, 99, 12};
   size_t nelem = 5;
   int key;
   int *result;

   key = 99;
   result = (int *) lfind(&key, array, &nelem,
      sizeof(int), (int(*)(const void *,const void *))compare);
   if (result)
      printf("Number %d found\n",key);
   else
      printf("Number %d not found\n",key);

   return 0;
}
```

/* lsearch example */

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>     /* for strcmp declaration */

/* initialize number of colors */
char *colors[10] = { "Red", "Blue", "Green" };
int ncolors = 3;

int colorscmp(char **arg1, char **arg2)
{
   return(strcmp(*arg1, *arg2));
}

int addelem(char **key)
{
   int oldn = ncolors;
   lsearch(key, colors, (size_t *)&ncolors, sizeof(char *),
     (int(*)(const void *,const void *))colorscmp);
   return(ncolors == oldn);
}

int main(void)
{
   int i;
   char *key = "Purple";

   if (addelem(&key))
      printf("%s already in colors table\n", key);
   else
   {
      printf("%s added to colors table\n", key);
   }

   printf("The colors:\n");
   for (i = 0; i < ncolors; i++)
      printf("%s\n", colors[i]);
   return 0;
}
```

/* qsort example */

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

int sort_function( const void *a, const void *b);
char list[5][4] = { "cat", "car", "cab", "cap", "can" };

int main(void)
{
   int  x;

   qsort((void *)list, 5, sizeof(list[0]), sort_function);
   for (x = 0; x < 5; x++)
      printf("%s\n", list[x]);
   return 0;
}

int sort_function( const void *a, const void *b)
{
   return( strcmp((char *)a,(char *)b) );
}
```

/* _rtl_chmod example */

```
#include <errno.h>
#include <stdio.h>
#include <dos.h>
#include <io.h>

int get_file_attrib(char *filename);

int main(void)
{
  char filename[128];
  int attrib;
  printf("Enter a filename:");
  scanf("%s", filename);
  attrib = get_file_attrib(filename);
  if (attrib == -1)
     switch(errno)
     {
        case ENOENT : printf("Path or file not found.\n");
                      break;
        case EACCES : printf("Permission denied.\n");
                      break;
        default:      printf("Error number: %d", errno);
                      break;
     }
  else
     {
        if (attrib & FA_RDONLY)
           printf("%s is read-only.\n", filename);

        if (attrib & FA_HIDDEN)
           printf("%s is hidden.\n", filename);

        if (attrib & FA_SYSTEM)
           printf("%s is a system file.\n", filename);

        if (attrib & FA_DIREC)
           printf("%s is a directory.\n", filename);

        if (attrib & FA_ARCH)
           printf("%s is an archive file.\n", filename);
     }
  return 0;
}

/* returns the attributes of a DOS file */
int get_file_attrib(char *filename)
{
   return(_rtl_chmod(filename, 0));
}
```

/* _dos_getfileattr example */

```
#include <stdio.h>
#include <dos.h>

int main(void)
{
  char filename[128];
  unsigned attrib;
  printf("Enter a file name:");
  scanf("%s", filename);
  if (_dos_getfileattr(filename,&attrib) != 0)
  {
    perror("Unable to obtain file attributes");
    return 1;
  }
  if (attrib & _A_RDONLY)
    printf("%s is read-only.\n", filename);

  if (attrib & _A_HIDDEN)
    printf("%s is hidden.\n", filename);

  if (attrib & _A_SYSTEM)
    printf("%s is a system file.\n", filename);

  if (attrib & _A_VOLID)
    printf("%s is a volume label.\n", filename);

  if (attrib & _A_SUBDIR)
    printf("%s is a directory.\n", filename);

  if (attrib & _A_ARCH)
    printf("%s is an archive file.\n", filename);
  return 0;
}
```

/* _dos_setfileattr example */

```
#include <stdio.h>
#include <dos.h>

int main(void)
{
  char filename[128];
  unsigned attrib;
  printf("Enter a file name:");
  scanf("%s", filename);
  if (_dos_getfileattr(filename,&attrib) != 0)
  {
    perror("Unable to obtain file attributes");
    return 1;
  }
  if (attrib & _A_RDONLY)
  {
    printf("%s currently read-only, making it read-write.\n", filename);
    attrib &= ~_A_RDONLY;
  }
  else
  {
    printf("%s currently read-write, making it read-only.\n", filename);
    attrib |= _A_RDONLY;
  }
  if (_dos_setfileattr(filename,attrib) != 0)
    perror("Unable to set file attributes");
  return 0;
}
```

/* close example */

```
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

main()
{
  int handle;
  char buf[11] = "0123456789";

  /* create a file containing 10 bytes */
  handle = open("NEW.FIL", O_CREAT);
  if (handle > -1)
  {
    write(handle, buf, strlen(buf));

    close(handle);                    /* close the file */
  }
  else
  {
    printf("Error opening file\n");
  }
  return 0;
}
```

/* _rtl_close example */

```
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{
   int handle;
   char msg[] = "Hello world";

   if ((handle = _rtl_open("TEST.$$$", O_RDWR)) == -1)
   {
      perror("Error:");
      return 1;
   }
   _rtl_write(handle, msg, strlen(msg));
   _rtl_close(handle);
   return 0;
}
```

/* _dos_close example */

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

int main(void)
{
  unsigned count;
  int handle;
  char buf[11] = "0123456789";

  /* create a file containing 10 bytes */
  if (_dos_creat("DUMMY.FIL", _A_NORMAL, &handle) != 0)
  {
    perror("Unable to create DUMMY.FIL");
    return 1;
  }
  if (_dos_write(handle, buf, strlen(buf), &count) != 0)
  {
    perror("Unable to write to DUMMY.FIL");
    return 1;
  }
  /* close the file */
  _dos_close(handle);
  return 0;
}
```

/* cos example */

```
#include <stdio.h>
#include <math.h>

int main(void)
{
   double result;
   double x = 0.5;

   result = cos(x);
   printf("The cosine of %lf is %lf\n", x, result);
   return 0;
}
```

/* sin example */

```
#include <stdio.h>
#include <math.h>

int main(void)
{
   double result, x = 0.5;

   result = sin(x);
   printf("The sin of %lf is %lf\n", x, result);
   return 0;
}
```

/* tan example */

```
#include <stdio.h>
#include <math.h>

int main(void)
{
   double result, x;

   x = 0.5;
   result = tan(x);
   printf("The tan of %lf is %lf\n", x, result);
   return 0;
}
```

/* cosh example */

```
#include <stdio.h>
#include <math.h>

int main(void)
{
   double result;
   double x = 0.5;

   result = cosh(x);
   printf("The hyperbolic cosine of %lf is %lf\n", x, result);
   return 0;
}
```

/* sinh example */

```
#include <stdio.h>
#include <math.h>

int main(void)
{
   double result, x = 0.5;

   result = sinh(x);
   printf("The hyperbolic sin of %lf is %lf\n", x, result);
   return 0;
}
```

/* tanh example */

```
#include <stdio.h>
#include <math.h>

int main(void)
{
   double result, x;

   x = 0.5;
   result = tanh(x);
   printf("The hyperbolic tangent of %lf is %lf\n", x, result);
   return 0;
}
```

/* creat example */

```
#include <sys\stat.h>
#include <string.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{
   int handle;
   char buf[11] = "0123456789";

   /* change the default file mode from text to binary */
   _fmode = O_BINARY;

   /* create a binary file for reading and writing */
   handle = creat("DUMMY.FIL", S_IREAD |S_IWRITE);

   /* write 10 bytes to the file */
   write(handle, buf, strlen(buf));

   /* close the file */
   close(handle);
   return 0;
}
```

/* _rtl_creat example */

```
#include <dos.h>
#include <string.h>
#include <stdio.h>
#include <io.h>

int main() {
  unsigned count;
  int handle;
  char buf[11] = "0123456789";

  /* Create a 10-byte file using _dos_creat. */
  if (_dos_creat("DUMMY.FIL", _A_NORMAL, &handle) != 0) {
    perror("Unable to _dos_creat DUMMY.FIL");
    return 1;
  }
  if (_dos_write(handle, buf, strlen(buf), &count) != 0) {
    perror("Unable to _dos_write to DUMMY.FIL");
    return 1;
  }
  _dos_close(handle);

  /* Create another 10-byte file using _rtl_creat. */
  if ((handle = _rtl_creat("DUMMY2.FIL", 0)) < 0) {
    perror("Unable to _rtl_create DUMMY2.FIL");
    return 1;
  }
  if (_rtl_write(handle, buf, strlen(buf)) < 0) {
    perror("Unable to _rtl_write to DUMMY2.FIL");
    return 1;
  }
  _rtl_close(handle);
  return 0;
}
```

/* _dos_creat example */

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

int main(void)
{
  unsigned count;
  int handle;
  char buf[11] = "0123456789";

  /* create a file containing 10 bytes */
  if (_dos_creat("DUMMY.FIL", _A_NORMAL, &handle) != 0)
  {
    perror("Unable to create DUMMY.FIL");
    return 1;
  }
  if (_dos_write(handle, buf, strlen(buf), &count) != 0)
  {
    perror("Unable to write to DUMMY.FIL");
    return 1;
  }
  /* close the file */
  _dos_close(handle);
  return 0;
}
```

/* _dos_creatnew example */

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

int main(void)
{
  unsigned count;
  int handle;
  char buf[11] = "0123456789";

  /* create a file containing 10 bytes */
  if (_dos_creatnew("DUMMY.FIL", _A_NORMAL, &handle) != 0)
  {
    perror("Unable to create DUMMY.FIL");
    return 1;
  }
  if (_dos_write(handle, buf, strlen(buf), &count) != 0)
  {
    perror("Unable to write to DUMMY.FIL");
    return 1;
  }
  /* close the file */
  _dos_close(handle);
  return 0;
}
```

/* creatnew example */

```
#include <string.h>
#include <stdio.h>
#include <errno.h>
#include <dos.h>
#include <io.h>

int main(void)
{
   int handle;
   char buf[11] = "0123456789";

   /* attempt to create a file that doesn't already exist */
   handle = creatnew("DUMMY.FIL", 0);

  if (handle == -1)
      printf("DUMMY.FIL already exists.\n");
   else
   {
      printf("DUMMY.FIL successfully created.\n");
      write(handle, buf, strlen(buf));
      close(handle);
   }
   return 0;
}
```

/* disable example */

```
/* * * * * * * * * * *
NOTE: This is an interrupt service routine. You cannot compile this program
  with Test Stack Overflow turned on and get an executable file that
  operates correctly.
 * * * * * * * * * * */

#include <stdio.h>
#include <dos.h>
#include <conio.h>

#define INTR 0X1C    /* The clock tick interrupt */

#ifdef __cplusplus
    #define __CPPARGS ...
#else
    #define __CPPARGS
#endif

void interrupt (*oldhandler)(__CPPARGS);

int count=0;

void interrupt handler(__CPPARGS) /* if C++, need the the ellipsis */
{
/* disable interrupts during the handling of the interrupt */
   disable();
/* increase the global counter */
   count++;
/* reenable interrupts at the end of the handler */
  enable();
/* call the old routine */
   oldhandler();
}

int main(void)
{
/* save the old interrupt vector */
   oldhandler = getvect(INTR);

/* install the new interrupt handler */
   setvect(INTR, handler);

/* loop until the counter exceeds 20 */
   while (count < 20)
      printf("count is %d\n",count);

/* reset the old interrupt handler */
   setvect(INTR, oldhandler);

  return 0;
}
```

/* _disable example */

```
/* * * * * * * * * * *
NOTE: This is an interrupt service routine. You cannot compile this program
  with Test Stack Overflow turned on and get an executable file that
  operates correctly.
 * * * * * * * * * * */

#include <stdio.h>
#include <dos.h>
#include <conio.h>

#define INTR 0X1C    /* The clock tick interrupt */

#ifdef __cplusplus
    #define __CPPARGS ...
#else
    #define __CPPARGS
#endif

void interrupt (*oldhandler)(__CPPARGS);

int count=0;

void interrupt handler(__CPPARGS) /* if C++, need the the ellipsis */
{
/* disable interrupts during the handling of the interrupt */
   _disable();
/* increase the global counter */
   count++;
/* reenable interrupts at the end of the handler */
   enable();
/* call the old routine */
   oldhandler();
}

int main(void)
{
/* save the old interrupt vector */
  oldhandler = _dos_getvect(INTR);

/* install the new interrupt handler */
   _dos_setvect(INTR, handler);

/* loop until the counter exceeds 20 */
   while (count < 20)
      printf("count is %d\n",count);

/* reset the old interrupt handler */
   _dos_setvect(INTR, oldhandler);

   return 0;
}
```

/* enable example */

```
/* * * * * * * * * * *
NOTE: This is an interrupt service routine. You cannot compile this program
  with Test Stack Overflow turned on and get an executable file that
  operates correctly.
 * * * * * * * * * * */

#include <stdio.h>
#include <dos.h>
#include <conio.h>

#define INTR 0X1C    /* The clock tick interrupt */

#ifdef __cplusplus
    #define __CPPARGS ...
#else
    #define __CPPARGS
#endif

void interrupt (*oldhandler)(__CPPARGS);

int count=0;

void interrupt handler(__CPPARGS) /* if C++, need the the ellipsis */
{
/* disable interrupts during the handling of the interrupt */
   disable();
/* increase the global counter */
   count++;
/* reenable interrupts at the end of the handler */
   enable();
/* call the old routine */
   oldhandler();
}

int main(void)
{
/* save the old interrupt vector */
   oldhandler = getvect(INTR);

/* install the new interrupt handler */
   setvect(INTR, handler);

/* loop until the counter exceeds 20 */
   while (count < 20)
      printf("count is %d\n",count);

/* reset the old interrupt handler */
   setvect(INTR, oldhandler);

   return 0;
}
```

/* _enable example */

```
/* * * * * * * * * * *
NOTE: This is an interrupt service routine. You cannot compile this program
  with Test Stack Overflow turned on and get an executable file that
  operates correctly.
 * * * * * * * * * * */

#include <stdio.h>
#include <dos.h>
#include <conio.h>

#define INTR 0X1C    /* The clock tick interrupt */

#ifdef __cplusplus
    #define __CPPARGS ...
#else
    #define __CPPARGS
#endif

void interrupt (*oldhandler)(__CPPARGS);

int count=0;

void interrupt handler(__CPPARGS) /* if C++, need the the ellipsis */
{
/* disable interrupts during the handling of the interrupt */
   disable();
/* increase the global counter */
   count++;
/* reenable interrupts at the end of the handler */
   _enable();
/* call the old routine */
   oldhandler();
}

int main(void)
{
/* save the old interrupt vector */
   oldhandler = _dos_getvect(INTR);

/* install the new interrupt handler */
   _dos_setvect(INTR, handler);

/* loop until the counter exceeds 20 */
   while (count < 20)
      printf("count is %d\n",count);

/* reset the old interrupt handler */
   _dos_setvect(INTR, oldhandler);

   return 0;
}
```

/* div example */

```
/* div example */

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

div_t x;

int main(void)
{
   x = div(10,3);
   printf("10 div 3 = %d remainder %d\n",
   x.quot, x.rem);

   return 0;
}
```

/* ldiv example */

```
/* ldiv example */

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

int main(void)
{
   ldiv_t lx;

   lx = ldiv(100000L, 30000L);
  printf("100000 div 30000 = %ld remainder %ld\n", lx.quot, lx.rem);
   return 0;
}
```

/* dup example */

```
#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <io.h>

void flush(FILE *stream);

int main(void)
{
  FILE *fp;
  char msg[] = "This is a test";

  /* create a file */
  fp = fopen("DUMMY.FIL", "w");

  /* write some data to the file */
  fwrite(msg, strlen(msg), 1, fp);

  clrscr();
  printf("Press any key to flush DUMMY.FIL:");
  getch();

  /* flush the data to DUMMY.FIL without closing it */
  flush(fp);

  printf("\nFile was flushed, Press any key to quit:");
  getch();
  return 0;
}

void flush(FILE *stream)
{
  int duphandle;

  /* flush TC's internal buffer */
  fflush(stream);

  /* make a duplicate file handle */
  duphandle = dup(fileno(stream));

  /* close the duplicate handle to flush the DOS buffer */
  close(duphandle);
}
```

/* dup2 example */

```
#include <sys\stat.h>
#include <string.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{
   #define STDOUT 1

   int nul, oldstdout;
   char msg[] = "This is a test";

   /* create a file */
   nul = open("DUMMY.FIL", O_CREAT | O_RDWR,
      S_IREAD | S_IWRITE);

   /* create a duplicate handle for standard output */
   oldstdout = dup(STDOUT);
   /*
      redirect standard output to DUMMY.FIL
      by duplicating the file handle onto
      the file handle for standard output.
   */
   dup2(nul, STDOUT);

   /* close the handle for DUMMY.FIL */
   close(nul);

   /* will be redirected into DUMMY.FIL */
   write(STDOUT, msg, strlen(msg));

   /* restore original standard output handle */
   dup2(oldstdout, STDOUT);

   /* close duplicate handle for STDOUT */
   close(oldstdout);

   return 0;
}
```

/* ecvt example */

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int main(void)
{
   char *string;
   double value;
   int dec, sign;
   int ndig = 10;

   clrscr();
   value = 9.876;
   string = ecvt(value, ndig, &dec, &sign);
   printf("string = %s      dec = %d sign = %d\n", string, dec, sign);

   value = -123.45;
   ndig= 15;
   string = ecvt(value,ndig,&dec,&sign);
   printf("string = %s dec = %d sign = %d\n", string, dec, sign);

   value = 0.6789e5; /* scientific notation */
   ndig = 5;
   string = ecvt(value,ndig,&dec,&sign);
   printf("string = %s         dec = %d sign = %d\n", string, dec, sign);

   return 0;
}
```

/* fcvt example */

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

int main(void)
{
  char *str;
  double num;
  int dec, sign, ndig = 5;

  /* a regular number */
  num = 9.876;
  str = fcvt(num, ndig, &dec, &sign);
  printf("string = %10s decimal place = %d sign = %d\n", str, dec, sign);

  /* a negative number */
  num = -123.45;
  str = fcvt(num, ndig, &dec, &sign);
  printf("string = %10s decimal place = %d sign = %d\n", str, dec, sign);

  /* scientific notation */
  num = 0.678e5;
  str = fcvt(num, ndig, &dec, &sign);
  printf("string = %10s  decimal place= %d  sign = %d\n", str, dec, sign);
  return 0;
}
```

/* execl example */

```
/* execl() example */
#include <stdio.h>
#include <process.h>

int main(int argc, char *argv[])
{
  int loop;

  printf("%s running...\n\n", argv[0]);

  if (argc == 1) {    /* check for only one command-line parameter */
    printf("%s calling itself again...\n", argv[0]);
    execl(argv[0], argv[0], "ONE", "TWO", "THREE", NULL);
    perror("EXEC:");
    exit(1);
  }

  printf("%s called with arguments:\n", argv[0]);

  for (loop = 1; loop <= argc; loop++)
    puts(argv[loop]);       /* Display all command-line parameters */
  return 0;
}
```

/* execlp example */

```
/* execlp example */
#include <process.h>
#include <stdio.h>
#include <errno.h>

int main( int argc, char *argv[])
{
   int i;

   printf("Command line arguments:\n");

   for (i=0; i < argc; ++i)
      printf("[%2d] %s\n", i, argv[i]);

   printf("About to exec child with arg1 arg2 ...\n");
   execlp("CHILD.EXE", "CHILD.EXE", "arg1", "arg2", NULL);

   perror("exec error");
   exit(1);

   return 0;
}
```

/* execle example */

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

int main(int argc, char *argv[], char *env[])
{
  int loop;
  char *new_env[] = { "TESTING", NULL };
  printf("%s running...\n\n", argv[0]);
  if (argc == 1) {   /* check for only one command-line parameter */
    printf("%s calling itself again...\n", argv[0]);
    execle(argv[0], argv[0], "ONE", "TWO", "THREE", NULL, new_env);
    perror("EXEC:");
    exit(1);
  }
  printf("%s called with arguments:\n", argv[0]);
  for (loop = 1; loop <= argc; loop++)
    puts(argv[loop]);  /* display all command-line parameters */

  /* display the first environment parameter */
  printf("value of env[0]: %s\n",env[0]);
  return 0;
}
```

/* execlpe example */

```
#include <process.h>
#include <stdio.h>
#include <errno.h>

int main(int argc, char *argv[], char **envp )
{
  int i;

  printf("Command line arguments:\n");
  for (i=0; i < argc; ++i)
    printf("[%2d] : %s\n", i, argv[i]);

  printf("About to exec child with arg1 arg2 ...\n");
  execlpe("CHILD.EXE", "CHILD.EXE", "arg1", "arg2", NULL, envp);

  perror("exec error");
  exit(1);
  return 0;
}
```

/* execv example */

```
#include <process.h>
#include <stdio.h>
#include <errno.h>

int main(int argc, char *argv[])
{
  int i;

  printf("Command line arguments:\n");
  for (i=0; i < argc; i++)
    printf("[%2d] : %s\n", i, argv[i]);

  printf("About to exec child with arg1 arg2 ...\n");
  execv("CHILD.EXE", argv);

  perror("exec error");
  exit(1);
  return 0;
}
```

/* execve example */

```
#include <process.h>
#include <stdio.h>
#include <errno.h>

int main(int argc, char *argv[], char **envp)
{
  int i;

  printf("Command line arguments:\n");
  for (i=0; i < argc; ++i)
    printf("[%2d] : %s\n", i, argv[i]);

  printf("About to exec child with arg1 arg2 ...\n");
  execve("CHILD.EXE", argv, envp);

  perror("exec error");
  exit(1);
  return 0;
}
```

/* execvp example */

```
#include <process.h>
#include <stdio.h>
#include <errno.h>

int main(int argc, char *argv[])
{
  int i;

  printf("Command line arguments:\n");
  for (i=0; i < argc; ++i)
    printf("[%2d] : %s\n", i, argv[i]);

  printf("About to exec child with arg1 arg2 ...\n");
  execvp("CHILD.EXE", argv);

  perror("exec error");
  exit(1);
  return 0;
}
```

/* execvpe example */

```
#include <process.h>
#include <stdio.h>
#include <errno.h>

int main(int argc, char *argv[], char **envp)
{
  int i;

  printf("Command line arguments:\n");
  for (i=0; i < argc; ++i)
    printf("[%2d] : %s\n", i, argv[i]);

  printf("About to exec child with arg1 arg2 ...\n");
  execvpe("CHILD.EXE", argv, envp);

  perror("exec error");
  exit(1);
  return 0;
}
```

/* _exit example */

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

void done(void);

int main(void)
{

   atexit(done);
   _exit(0);
   return 0;
}

void done()
{
   printf("hello\n");
}
```

/* _c_exit example */

```
#include <process.h>
#include <io.h>
#include <fcntl.h>
#include <stdio.h>
#include <dos.h>

main()
{
  int fd;
  char c;

  if ((fd = open("_c_exit.c",O_RDONLY)) < 0)
  {
    printf("Unable to open _c_exit.c for reading\n");
    return 1;
  }
  if (read(fd,&c,1) != 1)
    printf("Unable to read from open file handle %d before _c_exit\n",fd);
  else
    printf("Successfully read from open file handle %d before _c_exit\
 n",fd);
  printf("Interrupt zero vector before _c_exit = %Fp\n",_dos_getvect(0));
  _c_exit();
  if (read(fd,&c,1) != 1)
    printf("Unable to read from open file handle %d after _c_exit\n",fd);
  else
    printf("Successfully read from open file handle %d after _c_exit\
 n",fd);
  printf("Interrupt zero vector after _c_exit = %Fp\n",_dos_getvect(0));
 return 0;
}
```

/* exit */

```
#include <stdlib.h>
#include <conio.h>
#include <stdio.h>

int main(void)
{
   int status;

   printf("Enter either 1 or 2\n");
   status = getch();
   /* Sets DOS errorlevel  */
   exit(status - '0');

/* Note: this line is never reached */
   return 0;
}
```

/* _cexit example */

```
#include <windows.h>
#include <process.h>
#include <io.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>

void exit_func(void)
{
  printf("Exit function called\n\n");
  printf("Close Window to return to program... It will beep if able to read
  from file");
}

int main(void)
{
  int fd;
  char c;

  if ((fd = open("_cexit.c",O_RDONLY)) < 0)
  {
  printf("Unable to open _cexit.c for reading\n");
  return 1;
  }
  atexit(exit_func);
  if (read(fd,&c,1) != 1)
  printf("Unable to read from open file handle %d before _cexit\n",fd);
  else
  printf("Successfully read from open file handle %d before _cexit\n",fd);
  _cexit();
  if (read(fd,&c,1) == 1)
    MessageBeep(0);
  return 0;
}
```

/* farfree example */

```
#include <stdio.h>
#include <alloc.h>
#include <string.h>
#include <dos.h>

int main(void)
{
   char far *fptr;
   char *str = "Hello";

   /* allocate memory for the far pointer */
   fptr = (char far *) farcalloc(10, sizeof(char));

   /* copy "Hello" into allocated memory */
/*
Note: movedata is used because you might be in a small data model, in which
  case a normal string copy routine can't be used since it assumes the
  pointer size is near.
 */
   movedata(FP_SEG(str), FP_OFF(str),
            FP_SEG(fptr), FP_OFF(fptr),
            strlen(str));

   /* display string (note the F modifier) */
   printf("Far string is: %Fs\n", fptr);

   /* free the memory */
   farfree(fptr);

   return 0;
}
```

/* free example */

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

int main(void)
{
   char *str;

   /* allocate memory for string */
   str = (char *) malloc(10);

   /* copy "Hello" to string */
   strcpy(str, "Hello");

   /* display string */
   printf("String is %s\n", str);

   /* free memory */
   free(str);

   return 0;
}
```

/* fgetc example */

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

int main(void)
{
   FILE *stream;
   char string[] = "This is a test";
   char ch;

   /* open a file for update */
   stream = fopen("DUMMY.FIL", "w+");

   /* write a string into the file */
   fwrite(string, strlen(string), 1, stream);

   /* seek to the beginning of the file */
   fseek(stream, 0, SEEK_SET);

   do
   {
      /* read a char from the file */
      ch = fgetc(stream);

      /* display the character */
      putch(ch);
   } while (ch != EOF);

   fclose(stream);
   return 0;
}
```

/* fputc example */

```
#include <stdio.h>

int main(void)
{
   char msg[] = "Hello world";
   int i = 0;

   while (msg[i])
   {
      fputc(msg[i], stdout);
      i++;
   }
   return 0;
}
```

/* fgetchar example */

```
#include <stdio.h>

int main(void)
{
   char ch;

   /* prompt the user for input */
   printf("Enter a character followed by <Enter>: ");

   /* read the character from stdin */
   ch = fgetchar();

   /* display what was read */
   printf("The character read is: '%c'\n", ch);
   return 0;
}
```

/* fputchar example */

```
#include <stdio.h>

int main(void)
{
   char msg[] = "This is a test";
   int i = 0;

   while (msg[i])
   {
      fputchar(msg[i]);
      i++;
   }
   return 0;
}
```

/* fgets example */

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

int main(void)
{
   FILE *stream;
   char string[] = "This is a test";
   char msg[20];

   /* open a file for update */
   stream = fopen("DUMMY.FIL", "w+");

   /* write a string into the file */
   fwrite(string, strlen(string), 1, stream);

   /* seek to the start of the file */
   fseek(stream, 0, SEEK_SET);

   /* read a string from the file */
   fgets(msg, strlen(string)+1, stream);

   /* display the string */
   printf("%s", msg);

   fclose(stream);
   return 0;
}
```

/* fputs example */

```
#include <stdio.h>

int main(void)
{
   /* write a string to standard output */
   fputs("Hello world\n", stdout);

   return 0;
}
```

/* _dos_findfirst and _dos_findnext example */

```
#include <stdio.h>
#include <dos.h>

int main(void)
{
  struct find_t ffblk;
  int done;
  printf("Directory listing of *.*\n");
  done = _dos_findfirst("*.*",_A_NORMAL,&ffblk);
  while (!done) {
    printf("  %s\n", ffblk.name);
    done = _dos_findnext(&ffblk);
  }
  return 0;
}

/* Program output

Directory listing of *.*
  FINDFRST.C
  FINDFRST.OBJ
  FINDFRST.MAP
  FINDFRST.EXE  */
```

/* findfirst and findnext example */

```
/* findfirst and findnext example */

#include <stdio.h>
#include <dir.h>

int main(void)
{
   struct ffblk ffblk;
   int done;
   printf("Directory listing of *.*\n");
   done = findfirst("*.*",&ffblk,0);
   while (!done)
   {
      printf("  %s\n", ffblk.ff_name);
      done = findnext(&ffblk);
   }

   return 0;
}
```

/* _fsopen example */

```
#include <io.h>
#include <process.h>
#include <share.h>
#include <stdio.h>

int main(void)
{
  FILE *f;
  int status;
  f = _fsopen("c:\\autoexec.bat", "r", SH_DENYNO);
  if (f == NULL)
  {
    printf("_fsopen failed\n");
    exit(1);
  }
  status = access("c:\\autoexec.bat", 6);
  if (status == 0)
    printf("read/write access allowed\n");
  else
    printf("read/write access not allowed\n");
  fclose(f);
  return 0;
}
```

/* fdopen example */

```
#include <sys\stat.h>
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{
   int handle;
   FILE *stream;

   /* open a file */
   handle = open("DUMMY.FIL", O_CREAT,
                 S_IREAD | S_IWRITE);

   /* now turn the handle into a stream */
   stream = fdopen(handle, "w");

   if (stream == NULL)
      printf("fdopen failed\n");
   else
   {
      fprintf(stream, "Hello world\n");
      fclose(stream);
   }
   return 0;
}
```

/* fopen example */

```
/* Program to create backup of the AUTOEXEC.BAT file */

#include <stdio.h>

int main(void)
{
   FILE *in, *out;

   if ((in = fopen("\\AUTOEXEC.BAT", "rt"))
       == NULL)
   {
      fprintf(stderr, "Cannot open input file.\n");
      return 1;
   }

   if ((out = fopen("\\AUTOEXEC.BAK", "wt"))
       == NULL)
   {
      fprintf(stderr, "Cannot open output file.\n");
      return 1;
   }

   while (!feof(in))
      fputc(fgetc(in), out);

   fclose(in);
   fclose(out);
   return 0;
}
```

/* freopen example */

```
#include <stdio.h>

int main(void)
{
   /* redirect standard output to a file */
   if (freopen("OUTPUT.FIL", "w", stdout)
       == NULL)
      fprintf(stderr, "error redirecting stdout\n");

   /* this output will go to a file */
   printf("This will go into a file.");

   /* close the standard output stream */
   fclose(stdout);

   return 0;
}
```

/* freemem example */

```
#include <dos.h>
#include <alloc.h>
#include <stdio.h>

int main(void)
{
   unsigned int size, segp;
   int stat;

   size = 64; /* (64 x 16) = 1024 bytes */
   stat = allocmem(size, &segp);
   if (stat < 0)
      printf("Allocated memory at segment: %x\n", segp);
   else
      printf("Failed: maximum number of\
              paragraphs available is %u\n", stat);
   freemem(segp);

   return 0;
}
```

/* fstat example */

```
#include <sys\stat.h>
#include <stdio.h>
#include <time.h>

int main(void)
{
   struct stat statbuf;
   FILE *stream;

   /* open a file for update */
   if ((stream = fopen("DUMMY.FIL", "w+"))
       == NULL)
   {
      fprintf(stderr, "Cannot open output file.\n");
      return(1);
   }
   fprintf(stream, "This is a test");
   fflush(stream);

   /* get information about the file */
   fstat(fileno(stream), &statbuf);
   fclose(stream);

   /* display the information returned */
   if (statbuf.st_mode & S_IFCHR)
      printf("Handle refers to a device.\n");
   if (statbuf.st_mode & S_IFREG)
      printf("Handle refers to an ordinary file.\n");
   if (statbuf.st_mode & S_IREAD)
      printf("User has read permission on file.\n");
   if (statbuf.st_mode & S_IWRITE)
      printf("User has write permission on file.\n");

   printf("Drive letter of file: %c\n", 'A'+statbuf.st_dev);
   printf("Size of file in bytes: %ld\n", statbuf.st_size);
   printf("Time file last opened: %s\n", ctime(&statbuf.st_ctime));
   return 0;
}
```

/* stat example */

```
#include <sys\stat.h>
#include <stdio.h>
#include <time.h>

#define FILENAME "TEST.$$$"

int main(void)
{
   struct stat statbuf;
   FILE *stream;

   /* open a file for update */
   if ((stream = fopen(FILENAME, "w+")) == NULL)
   {
      fprintf(stderr, "Cannot open output file.\n");
      return(1);
   }

   /* get information about the file */
   stat(FILENAME, &statbuf);

   fclose(stream);

   /* display the information returned */
   if (statbuf.st_mode & S_IFCHR)
      printf("Handle refers to a device.\n");
   if (statbuf.st_mode & S_IFREG)
      printf("Handle refers to an ordinary file.\n");
   if (statbuf.st_mode & S_IREAD)
      printf("User has read permission on file.\n");
   if (statbuf.st_mode & S_IWRITE)
      printf("User has write permission on file.\n");

   printf("Drive letter of file: %c\n", 'A'+statbuf.st_dev);
   printf("Size of file in bytes: %ld\n", statbuf.st_size);
   printf("Time file last opened: %s\n", ctime(&statbuf.st_ctime));
   return 0;
}
```

/* getc example */

```
#include <stdio.h>

int main(void)
{
   char ch;

   printf("Input a character:");
/* read a character from the
  standard input stream */
   ch = getc(stdin);
   printf("The character input was: '%c'\n", ch);
   return 0;
}
```

/* putc example */

```
#include <stdio.h>

int main(void)
{
   char msg[] = "Hello world\n";
   int i = 0;

   while (msg[i])
      putc(msg[i++], stdout);
   return 0;
}
```

/* getch example */

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

int main(void)
{
  int c;
  int extended = 0;
  c = getch();
  if (!c)
    extended = getch();
  if (extended)
    printf("The character is extended\n");
  else
    printf("The character isn't extended\n");

  return 0;
}
```

/* getche example */

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

int main(void)
{
   char ch;

   printf("Input a character:");
   ch = getche();
   printf("\nYou input a '%c'\n", ch);
   return 0;
}
```

/* getchar example */

```
#include <stdio.h>

int main(void)
{
   int c;

/*
Note that getchar reads from stdin and is line buffered; this means it will
  not return until you press ENTER.
 */

   while ((c = getchar()) != '\n')
      printf("%c", c);

   return 0;
}
```

/* putchar example */

```
#include <stdio.h>

/* define some box-drawing characters */
#define LEFT_TOP  0xDA
#define RIGHT_TOP 0xBF
#define HORIZ     0xC4
#define VERT      0xB3
#define LEFT_BOT  0xC0
#define RIGHT_BOT 0xD9

int main(void)
{
   char i, j;

   /* draw the top of the box */
   putchar(LEFT_TOP);
   for (i=0; i<10; i++)
      putchar(HORIZ);
   putchar(RIGHT_TOP);
   putchar('\n');

   /* draw the middle */
   for (i=0; i<4; i++)
   {
      putchar(VERT);
      for (j=0; j<10; j++)
         putchar(' ');
      putchar(VERT);
      putchar('\n');
   }

   /* draw the bottom */
   putchar(LEFT_BOT);
   for (i=0; i<10; i++)
      putchar(HORIZ);
   putchar(RIGHT_BOT);
   putchar('\n');

   return 0;
}
```

/* getcwd example */

```
#include <stdio.h>
#include <dir.h>

int main(void)
{
   char buffer[MAXPATH];

   getcwd(buffer, MAXPATH);
   printf("The current directory is: %s\n", buffer);
   return 0;
}
```

/* _getdcwd example */

```
#include <direct.h>
#include <stdio.h>

char buf[65];

void main()
{
  if (_getdcwd(3, buf, sizeof(buf)) == NULL)
    perror("Unable to get current directory of drive C");
  else
    printf("Current directory of drive C is %s\n",buf);
}
```

/* _dos_getdate example */

```
#include <dos.h>
#include <stdio.h>

int main(void)
{
  struct dosdate_t d;
  _dos_getdate(&d);
  printf("The current year is: %d\n", d.year);
  printf("The current day is: %d\n", d.day);
  printf("The current month is: %d\n", d.month);
  return 0;
}
```

/* _dos_setdate example */

```
#include <dos.h>
#include <process.h>
#include <stdio.h>

int main(void)
{
  struct dosdate_t reset;
  reset.year  = 2001;
  reset.day   = 1;
  reset.month = 1;
  printf("Setting date to 1/1/2001.\n");
  _dos_setdate(&reset);
  system("date");
  return 0;
}
```

/* getdate example */

```
#include <dos.h>
#include <stdio.h>

int main(void)
{
   struct date d;

   getdate(&d);
   printf("The current year is: %d\n", d.da_year);
   printf("The current day is: %d\n", d.da_day);
   printf("The current month is: %d\n", d.da_mon);
   return 0;
}
```

/* setdate example */

```
#include <stdio.h>
#include <process.h>
#include <dos.h>

int main(void)
{
   struct date reset;
   struct date save_date;

   getdate(&save_date);
   printf("Original date:\n");
   system("date");

   reset.da_year = 2001;
   reset.da_day = 1;
   reset.da_mon = 1;
   setdate(&reset);

   printf("Date after setting:\n");
   system("date");

   setdate(&save_date);
   printf("Back to original date:\n");
   system("date");

   return 0;
}
```

/* _dos_getdiskfree example */

```
#include <stdio.h>
#include <dos.h>
#include <process.h>

int main(void)
{
  struct diskfree_t free;
  long avail;

  if (_dos_getdiskfree(0, &free) != 0) {
    printf("Error in _dos_getdiskfree() call\n");
    exit(1);
  }
  avail = (long) free.avail_clusters
        * (long) free.bytes_per_sector
        * (long) free.sectors_per_cluster;
  printf("The current drive has %ld bytes available\n", avail);
  return 0;
}
```

/* getdfree example */

```
#include <stdio.h>
#include <dos.h>
#include <process.h>

int main(void)
{
  struct diskfree_t free;
  long avail;

  if (_dos_getdiskfree(0, &free) != 0) {
    printf("Error in _dos_getdiskfree() call\n");
    exit(1);
  }
  avail = (long) free.avail_clusters
        * (long) free.bytes_per_sector
        * (long) free.sectors_per_cluster;
  printf("The current drive has %ld bytes available\n", avail);
  return 0;
}
```

/* _chdrive example */

```
 #include <stdio.h>
#include <direct.h>

int main(void)
{
  if (_chdrive(3) == 0)
    printf("Successfully changed to drive C:\n");
  else
    printf("Cannot change to drive C:\n");
  return 0;
}
```

/* _dos_getdrive example */

```
#include <stdio.h>
#include <dos.h>

int main(void)
{
  unsigned disk;
  _dos_getdrive(&disk);
  printf("The current drive is: %c\n", disk + 'A' - 1);
  return 0;
}
```

/* _dos_setdrive example */

```
#include <stdio.h>
#include <dos.h>

int main(void)
{
  unsigned maxdrives;
  _dos_setdrive(3,&maxdrives);     /* set drive to C: */
  printf("The number of logical drives is: %d\n", maxdrives);
  return 0;
}
```

/* _getdrive example */

```
#include <stdio.h>
#include <direct.h>

int main(void)
{
  int disk;
  disk = _getdrive() + 'A' - 1;
  printf("The current drive is: %c\n", disk);
  return 0;
}
```

/* getdisk example */

```
#include <stdio.h>
#include <dir.h>

int main(void)
{
  int disk, maxdrives = setdisk(2);
  disk = getdisk() + 'A';
  printf("\nThe number of logical drives is:%d\n", maxdrives);
  printf("The current drive is: %c\n", disk);
  return 0;
 }
```

/* setdisk example */

```
#include <stdio.h>
#include <dir.h>

int main(void)
{
   int save, disk, disks;

   /* save original drive */
   save = getdisk();

   /* print number of logic drives */
   disks = setdisk(save);
   printf("%d logical drives on the system\n\n", disks);

   /* print the drive letters available */
   printf("Available drives:\n");
   for (disk = 0;disk < 26;++disk)
   {
      setdisk(disk);
      if (disk == getdisk())
         printf("%c: drive is available\n", disk + 'a');
   }
   setdisk(save);

   return 0;
}
```

/* getdta example */

```
#include <dos.h>
#include <stdio.h>

int main(void)
{
   char far *dta;

   dta = getdta();
   printf("The current disk transfer address is: %Fp\n", dta);
   return 0;
}
```

/* setdta example */

```
#include <process.h>
#include <string.h>
#include <stdio.h>
#include <dos.h>

int main(void)
{
   char line[80], far *save_dta;
   char buffer[256] = "SETDTA test!";
   struct fcb blk;
   int result;

   /* get new file name from user */
   printf("Enter a file name to create:");
   gets(line);

   /* parse the new file name to the dta */
   parsfnm(line, &blk, 1);
   printf("%d %s\n", blk.fcb_drive, blk.fcb_name);

   /* request DOS services to create file */
   if (bdosptr(0x16, &blk, 0) == -1)
   {
      perror("Error creating file");
      exit(1);
   }

   /* save old dta and set new dta */
   save_dta = getdta();
   setdta(buffer);

   /* write new records */
   blk.fcb_recsize = 256;
   blk.fcb_random = 0L;
   result = randbwr(&blk, 1);
   printf("result = %d\n", result);

   if (!result)
      printf("Write OK\n");
   else
   {
      perror("Disk error");
      exit(1);
   }

   /* request DOS services to close the file */
   if (bdosptr(0x10, &blk, 0) == -1)
   {
      perror("Error closing file");
      exit(1);
   }

   /* reset the old dta */
   setdta(save_dta);
   return 0;
```

```
}
```

/* getfat example */

```
#include <stdio.h>
#include <dos.h>

int main(void)
{
   struct fatinfo diskinfo;
   int flag = 0;

   printf("Please insert disk in drive A\n");
   getchar();

   getfat(1, &diskinfo);
/* get drive information */

   printf("\nDrive A: is ");
   switch((unsigned char) diskinfo.fi_fatid)
   {
      case 0xFD:
        printf("360K low density\n");
        break;

      case 0xF9:
        printf("1.2 Meg high density\n");
        break;

      default:
        printf("unformatted\n");
        flag = 1;
   }

   if (!flag)
   {
      printf("  sectors per cluster %5d\n", diskinfo.fi_sclus);
      printf("   number of clusters %5d\n", diskinfo.fi_nclus);
      printf("     bytes per sector %5d\n", diskinfo.fi_bysec);
   }

   return 0;
}
```

/* getfatd example */

```
#include <stdio.h>
#include <dos.h>

int main()
{
   struct fatinfo diskinfo;

   /* get default drive information */
   getfatd(&diskinfo);
   printf("\nDefault Drive:\n");
   printf("sectors per cluster: %5d\n",diskinfo.fi_sclus);
   printf("FAT ID byte:         %5X\n",diskinfo.fi_fatid & 0xFF);
   printf("number of clusters   %5d\n",diskinfo.fi_nclus);
   printf("bytes per sector     %5d\n",diskinfo.fi_bysec);
   return 0;
}
```

/* getftime example */

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

int main(void)
{
   FILE *stream;
   struct ftime ft;

   if ((stream = fopen("TEST.$$$",
        "wt")) == NULL)
   {
      fprintf(stderr, "Cannot open output file.\n");
      return 1;
   }
   getftime(fileno(stream), &ft);
   printf("File time: %u:%u:%u\n",
          ft.ft_hour, ft.ft_min,
          ft.ft_tsec * 2);
   printf("File date: %u/%u/%u\n",
          ft.ft_month, ft.ft_day,
          ft.ft_year+1980);
   fclose(stream);
   return 0;
}
```

/* setftime example */

```
#include <stdio.h>
#include <process.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{
   struct ftime filet;
   FILE *fp;

   if ((fp = fopen("TEST.$$$", "w")) == NULL)
   {
      perror("Error:");
      exit(1);
   }

   fprintf(fp, "testing...\n");

   /* load ftime structure with new time and date */
   filet.ft_tsec = 1;
   filet.ft_min = 1;
   filet.ft_hour = 1;
   filet.ft_day = 1;
   filet.ft_month = 1;
   filet.ft_year = 21;

   /* show current directory for time and date */
   system("dir TEST.$$$");

   /* change the time and date stamp*/
   setftime(fileno(fp), &filet);

   /* close and remove the temporary file */
   fclose(fp);

   system("dir TEST.$$$");

   unlink("TEST.$$$");
   return 0;
}
```

/* _dos_getftime example */

```
#include <stdio.h>
#include <dos.h>

int main()
{
  FILE *stream;
  unsigned date, time;
  if ((stream = fopen("TEST.$$$", "w")) == NULL)
  {
    fprintf(stderr, "Cannot open output file.\n");
    return 1;
  }
  _dos_getftime(fileno(stream), &date, &time);
  printf("File date: 0x%x\n",date);
  printf("File time: 0x%x\n",time);
  fclose(stream);
  return 0;
}
```

/* _dos_setftime example */

```
#include <stdio.h>
#include <dos.h>

int main()
{
  FILE *stream;
  unsigned date, time;
  if ((stream = fopen("TEST.$$$", "w")) == NULL)
  {
    fprintf(stderr, "Cannot open output file.\n");
    return 1;
  }
  _dos_getftime(fileno(stream), &date, &time);
  printf("File year of TEST.$$$: %d\n",((date >> 9) & 0x7f) + 1980);
  date = (date & 0x1ff) | (21 << 9);
  _dos_setftime(fileno(stream), date, time);
  printf("Set file year to 2001.\n");
  fclose(stream);
  return 0;
}
```

/* puts example */

```
#include <stdio.h>

int main(void)
{
   char string[] = "This is an example output string\n";

   puts(string);
   return 0;
}
```

/* gets example */

```
#include <stdio.h>

int main(void)
{
   char string[80];

   printf("Input a string:");
   gets(string);
   printf("The string input was: %s\n", string);
   return 0;
}
```

/* puttext example */

```
#include <conio.h>

int main(void)
{
   char buffer[512];

   /* put some text to the console */
   clrscr();
   gotoxy(20, 12);
   cprintf("This is a test.  Press any key to continue ...");
   getch();

   /* grab screen contents */
   gettext(20, 12, 36, 21,buffer);
   clrscr();

   /* put selected characters back to the screen */
   gotoxy(20, 12);
   puttext(20, 12, 36, 21, buffer);
   getch();

   return 0;
}
```

/* gettext example */

```
#include <conio.h>

char buffer[4096];

int main(void)
{
   int i;

   clrscr();
   for (i = 0; i <= 20; i++)
      cprintf("Line #%d\r\n", i);
   gettext(1, 1, 80, 25, buffer);

   gotoxy(1, 25);
   cprintf("Press any key to clear screen...");
   getch();
   clrscr();
   gotoxy(1, 25);
   cprintf("Press any key to restore screen...");
   getch();
   puttext(1, 1, 80, 25, buffer);
   gotoxy(1, 25);
   cprintf("Press any key to quit...");
   getch();

   return 0;
}
```

/* _dos_gettime example */

```
#include   <dos.h>

int main(void)
{
  struct dostime_t t;
  _dos_gettime(&t);
  printf("The current time is: %2d:%02d:%02d.%02d\n", t.hour, t.minute,
    t.second, t.hsecond);
  return 0;
}
```

/* _dos_settime example */

```
#include <dos.h>
#include <process.h>
#include <stdio.h>

int main(void)
{
  struct dostime_t reset;
  reset.hour    = 17;
  reset.minute  = 0;
  reset.second  = 0;
  reset.hsecond = 0;
  printf("Setting time to 5 PM.\n");
  _dos_settime(&reset);
  system("time");
  return 0;
}
```

/* gettime example */

```
#include   <stdio.h>
#include   <dos.h>

int main(void)
{
   struct  time t;

   gettime(&t);
   printf("The current time is: %2d:%02d:%02d.%02d\n",
           t.ti_hour, t.ti_min, t.ti_sec, t.ti_hund);
   return 0;
}
```

/* settime example */

```
#include <stdio.h>
#include <dos.h>

int main(void)
{
   struct  time t;

   gettime(&t);
   printf("The current minute is: %d\n", t.ti_min);
   printf("The current hour is: %d\n", t.ti_hour);
   printf("The current hundredth of a second is: %d\n", t.ti_hund);
   printf("The current second is: %d\n", t.ti_sec);

   /* Add one to the minutes struct element and then call settime  */
   t.ti_min++;
   settime(&t);

   return 0;
}
```

/* _dos_getvect and _dos_setvect example */

```
#include <stdio.h>
#include <dos.h>

#ifdef __cplusplus
    #define __CPPARGS ...
#else
    #define __CPPARGS
#endif

void interrupt get_out(__CPPARGS);     /* interrupt prototype */
void interrupt (*oldfunc)(__CPPARGS); /* interrupt function pointer */


int looping = 1;

int main(void)
{
  puts("Press <Shift><PrtSc> to terminate");

  /* save the old interrupt */
  oldfunc  = _dos_getvect(5);

  /* install interrupt handler */
  _dos_setvect(5,get_out);

  /* do nothing */
  while (looping);

  /* restore to original interrupt routine */
  _dos_setvect(5,oldfunc);

  puts("Success");
  return 0;
}

void interrupt get_out(__CPPARGS) {
  looping = 0;             /* change global var to get out of oop */
}
```

/* getvect and setvect example */

```
/* * * * * * * * * * * * * * * *
NOTE: This is an interrupt service routine.
You can NOT compile this program with
Test Stack Overflow turned on and get an
executable file that will operate correctly.
 * * * * * * * * * * * * * * * */
#include <stdio.h>
#include <dos.h>
#include <conio.h>

#define INTR 0X1C    /* The clock tick interrupt */

#ifdef __cplusplus
    #define __CPPARGS ...
#else
    #define __CPPARGS
#endif

void interrupt ( *oldhandler)(__CPPARGS);

int count=0;

void interrupt handler(__CPPARGS)
{
/* increase the global counter */
   count++;

/* call the old routine */
  oldhandler();
}
  int main(void)
{
/* save the old interrupt vector */
   oldhandler = getvect(INTR);

/* install the new interrupt handler */
   setvect(INTR, handler);

/* loop until the counter exceeds 20 */
  while (count < 20)
      printf("count is %d\n",count);

/* reset the old interrupt handler */
   setvect(INTR, oldhandler);

   return 0;
}
```

/* getw example */

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

#define FNAME "test.$$$"

int main(void)
{
  FILE *fp;
  int word;

  /* place the word in a file */
  fp = fopen(FNAME, "wb");
  if (fp == NULL)
  {
     printf("Error opening file %s\n", FNAME);
      exit(1);
   }

   word = 94;
   putw(word,fp);
   if (ferror(fp))
       printf("Error writing to file\n");
   else
       printf("Successful write\n");
   fclose(fp);

   /* reopen the file */
   fp = fopen(FNAME, "rb");
   if (fp == NULL)
   {
      printf("Error opening file %s\n", FNAME);
      exit(1);
   }

   /* extract the word */
   word = getw(fp);

   if (ferror(fp))
       printf("Error reading file\n");
   else
       printf("Successful read: word = %d\n", word);

   /* clean up */
   fclose(fp);
   unlink(FNAME);

   return 0;
}
```

/* putw example */

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

#define FNAME "test.$$$"

int main(void)
{
   FILE *fp;
   int word;

   /* place the word in a file */
   fp = fopen(FNAME, "wb");
   if (fp == NULL)
   {
      printf("Error opening file %s\n", FNAME);
      exit(1);
   }

   word = 94;
   putw(word,fp);
   if (ferror(fp))
       printf("Error writing to file\n");
   else
       printf("Successful write\n");
   fclose(fp);

   /* reopen the file */
   fp = fopen(FNAME, "rb");
   if (fp == NULL)
   {
      printf("Error opening file %s\n", FNAME);
      exit(1);
   }

   /* extract the word */
   word = getw(fp);
   if (ferror(fp))
       printf("Error reading file\n");
   else
       printf("Successful read: word = %d\n", word);

   /* clean up */
   fclose(fp);
   unlink(FNAME);

   return 0;
}
```

/* gmtime example */

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <dos.h>

/* Pacific Standard Time & Daylight Savings */
char *tzstr = "TZ=PST8PDT";

int main(void)
{
   time_t t;
   struct tm *gmt, *area;

   putenv(tzstr);
   tzset();

   t = time(NULL);
   area = localtime(&t);
   printf("Local time is: %s", asctime(area));
   gmt = gmtime(&t);
   printf("GMT is:        %s", asctime(gmt));
   return 0;
}
```

/* localtime example */

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

int main(void)
{
   time_t timer;
   struct tm *tblock;

   /* gets time of day */
   timer = time(NULL);

   /* converts date/time to a structure */
   tblock = localtime(&timer);

   printf("Local time is: %s", asctime(tblock));

   return 0;
}
```

/* heapcheck and _heapchk example */

```
#include <stdio.h>
#include <alloc.h>

#define NUM_PTRS  10
#define NUM_BYTES 16

int main(void)
{
   char *array[ NUM_PTRS ];
   int i;

   for( i = 0; i < NUM_PTRS; i++ )
      array[ i ] = (char *) malloc( NUM_BYTES );

   for( i = 0; i < NUM_PTRS; i += 2 )
      free( array[ i ] );

   if( heapcheck() == _HEAPCORRUPT )
      printf( "Heap is corrupted.\n" );
   else
      printf( "Heap is OK.\n" );

   return 0;
}
```

/* farheapcheck example */

```
#include <stdio.h>
#include <alloc.h>

#define NUM_PTRS  10
#define NUM_BYTES 16

int main(void)
{
char far *array[ NUM_PTRS ];
int i;

  for( i = 0; i < NUM_PTRS; i++ )
  array[ i ] = (char far *) farmalloc( NUM_BYTES );

  for( i = 0; i < NUM_PTRS; i += 2 )
  farfree( array[ i ] );

  if( farheapcheck() == _HEAPCORRUPT )
    printf( "Heap is corrupted.\n" );
  else
    printf( "Heap is OK.\n" );

  return 0;
}
```

/* heapchecknode example */

```
#include <stdio.h>
#include <alloc.h>

#define NUM_PTRS  10
#define NUM_BYTES 16

int main(void)
{
   char *array[ NUM_PTRS ];
   int i;

   for( i = 0; i < NUM_PTRS; i++ )
      array[ i ] = (char *) malloc( NUM_BYTES );

   for( i = 0; i < NUM_PTRS; i += 2 )
      free( array[ i ] );

   for( i = 0; i < NUM_PTRS; i++ )
   {
      printf( "Node %2d ", i );
      switch( heapchecknode( array[ i ] ) )
      {
         case _HEAPEMPTY:
              printf( "No heap.\n" );
              break;
         case _HEAPCORRUPT:
              printf( "Heap corrupt.\n" );
              break;
         case _BADNODE:
              printf( "Bad node.\n" );
              break;
         case _FREEENTRY:
              printf( "Free entry.\n" );
              break;
         case _USEDENTRY:
              printf( "Used entry.\n" );
              break;
         default:
              printf( "Unknown return code.\n" );
               break;
      }
   }

   return 0;
}
```

/* heapfillfree and heapcheckfree example */

```
#include <stdio.h>
#include <alloc.h>
#include <mem.h>

#define NUM_PTRS  10
#define NUM_BYTES 16

int main(void)
{
   char *array[ NUM_PTRS ];
   int i;
   int res;

   for( i = 0; i < NUM_PTRS; i++ )
      array[ i ] = (char *) malloc( NUM_BYTES );

   for( i = 0; i < NUM_PTRS; i += 2 )
      free( array[ i ] );

   if( heapfillfree( 1 ) < 0 )
   {
      printf( "Heap corrupted.\n" );
      return 1;
   }

   for( i = 1; i < NUM_PTRS; i += 2 )
      memset( array[ i ], 0, NUM_BYTES );

   res = heapcheckfree( 1 );
   if( res < 0 )
      switch( res )
      {
         case _HEAPCORRUPT:
              printf( "Heap corrupted.\n" );
              return 1;
         case _BADVALUE:
              printf( "Bad value in free space.\n" );
              return 1;
         default:
              printf( "Unknown error.\n" );
              return 1;
      }

   printf( "Test successful.\n" );
   return 0;
}
```

/* heapwalk example*/

```
#include <stdio.h>
#include <alloc.h>

#define NUM_PTRS  10
#define NUM_BYTES 16

int main( void )
{
   struct heapinfo hi;
  char *array[ NUM_PTRS ];
   int i;

   for( i = 0; i < NUM_PTRS; i++ )
      array[ i ] = (char *) malloc( NUM_BYTES );

   for( i = 0; i < NUM_PTRS; i += 2 )
      free( array[ i ] );

   hi.ptr = NULL;
   printf( "   Size   Status\n" );
   printf( "   ----   ------\n" );
   while( heapwalk( &hi ) == _HEAPOK )
   printf( "%7u    %s\n", hi.size, hi.in_use ? "used" : "free" );

   return 0;
}
```

/* _rtl_heapwalk example*/

```
#include <stdio.h>
#include <malloc.h>
#include <alloc.h>

#define NUM_PTRS  10
#define NUM_BYTES 16
#if defined(__FLAT__)
int main( void )
{
  struct heapinfo hi;
  char *array[ NUM_PTRS ];
  int i;

  for( i = 0; i < NUM_PTRS; i++ )
     array[ i ] = (char *) malloc( NUM_BYTES );

  for( i = 0; i < NUM_PTRS; i += 2 )
     free( array[ i ] );

  hi.ptr = NULL;
  printf( "   Size   Status\n" );
  printf( "   ----   ------\n" );
  while( _rtl_heapwalk( &hi ) == _HEAPOK )
  printf( "%7u    %s\n", hi.size, hi.in_use ? "used" : "free" );

  return 0;
}
#endif
```

/* inp example */

```
#include <stdio.h>
#include <dos.h>

int main(void)
{
   int result;
   int port = 0;  /* serial port 0 */

   result = inport(port);
   printf("Word read from port %d = 0x%X\n", port, result);
   return 0;
}
```

/* inpw example */

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

int main(void)
{
  unsigned result;
  unsigned port = 0;
  result = inpw(port);
  printf("Word read from port %d = 0x%X\n", port, result);
  return 0;
}
```

/* outp example */

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

int main(void)
{
  unsigned port = 0;
  int value;
  value = outp(port, 'C');
  printf("Value %c sent to port number %d\n", value, port);
  return 0;
}
```

/* outpw example */

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

int main(void)
{
  unsigned value;
  unsigned port = 0;
  value = outpw(port, 64);
  printf("Value %d sent to port number %d\n", value, port);
  return 0;
}
```

/* inport example */

```
#include <stdio.h>
#include <dos.h>

int main(void)
{
  int result;
  int port = 0;
  result = inport(port);
  printf("Word read from port %d = 0x%X\n", port, result);
  return 0;
}
```

/* inportb example */

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

int main(void)
{
   unsigned char result;
   int port = 0;           /* serial port 1 */

   result = inportb(port);
   printf("Byte read from port %d = 0x%X\n", port, result);
   return 0;
}
```

/* outport example */

```
#include <conio.h>
#include <stdio.h>
int main(void)
{
   int port = 0;
   int value = 'C';

   outport(port, value);
   printf("Value %d sent to port number %d\n", value, port);
   return 0;
}
```

/* outportb example */

```
#include <stdio.h>
#include <dos.h>

int main(void)
{
  int port = 0;
  char value = 'C';

  outportb(port, value);
   printf("Value %c sent to port number %d\n", value, port);
  return 0;
}
```

/* int86 example */

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>

#define VIDEO 0x10

void movetoxy(int x, int y)
{
  union REGS regs;

   regs.h.ah = 2;  /* set cursor position */
  regs.h.dh = y;
   regs.h.dl = x;
   regs.h.bh = 0;  /* video page 0 */
  int86(VIDEO, &regs, &regs);
}

int main(void)
{
  clrscr();
  movetoxy(35, 10);
  printf("Hello\n");
  return 0;
}
```

/* int86x example */

```
#include <dos.h>
#include <process.h>
#include <stdio.h>

int main(void)
{
  char filename[80];
  union REGS inregs, outregs;
  struct SREGS segregs;

  printf("Enter filename: ");
  gets(filename);
  inregs.h.ah = 0x43;
  inregs.h.al = 0x21;
  inregs.x.dx = FP_OFF(filename);
  segregs.ds = FP_SEG(filename);
  int86x(0x21, &inregs, &outregs, &segregs);
   printf("File attribute: %X\n", outregs.x.cx);
   return 0;
}
```

/* intdos example */

```
#include <stdio.h>
#include <dos.h>

/* deletes file name; returns 0 on success, nonzero on failure */
int delete_file(char near *filename)
{
   union REGS regs;
   int ret;
   regs.h.ah = 0x41;
/* delete file */
   regs.x.dx = (unsigned) filename;
   ret = intdos(&regs, &regs);

   /* if carry flag is set, there was an error */
   return(regs.x.cflag ? ret : 0);
}

int main(void)
{
   int err;
   err = delete_file("NOTEXIST.$$$");
   if (!err)
      printf("Able to delete NOTEXIST.$$$\n");
   else
      printf("Not Able to delete NOTEXIST.$$$\n");
   return 0;
}
```

/* intdosx example */

```
#include <stdio.h>
#include <dos.h>

/* deletes file name; returns 0 on success,
nonzero on failure */
int delete_file(char far *filename)
{
   union REGS regs; struct SREGS sregs;
   int ret;
   regs.h.ah = 0x41;                      /* delete file */
   regs.x.dx = FP_OFF(filename);
   sregs.ds = FP_SEG(filename);
   ret = intdosx(&regs, &regs, &sregs);

   /* if carry flag is set, there was an error */
   return(regs.x.cflag ? ret : 0);
}

int main(void)

{
   int err;
   err = delete_file("NOTEXIST.$$$");
   if (!err)
      printf("Able to delete NOTEXIST.$$$\n");
   else
      printf("Not Able to delete NOTEXIST.$$$\n");
   return 0;
}
```

/* itoa example */

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

int main(void)
{
   int number = 12345;
   char string[25];

   itoa(number, string, 10);
   printf("integer = %d string = %s\n", number, string);
   return 0;
}
```

/* ltoa example */

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

int main(void)
{
   char string[25];
   long value = 123456789L;

   ltoa(value,string,10);
   printf("number = %ld  string = %s\n", value, string);

   return 0;
}
```

/* ultoa example */

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

int main( void )
{
   unsigned long lnumber = 3123456789L;
   char string[25];

   ultoa(lnumber,string,10);
   printf("string = %s  unsigned long = %lu\n",string,lnumber);

   return 0;
}
```

/* keep example */

```
/* * * * * * * * * * * * * *

NOTE: This is an interrupt service routine. You can NOT compile this
  program with Test Stack Overflow turned on and get an executable file
  which will operate correctly.

Due to the nature of this function the formula used to compute the number
  of paragraphs may not necessarily work in all cases.  Use with care!

Terminate Stay Resident (TSR) programs are complex and no other support for
  them is provided.

Refer to the MS-DOS technical documentation for more information.

 * * * * * * * * * * * * * */

#include <dos.h>
/* The clock tick interrupt */
#define INTR 0x1C
/* Screen attribute (blue on grey) */
#define ATTR 0x7900

#ifdef __cplusplus
    #define __CPPARGS ...
#else
    #define __CPPARGS
#endif

/* reduce heaplength and stacklength to make a smaller program in memory */
extern unsigned _heaplen = 1024;
extern unsigned _stklen  = 512;

void interrupt ( *oldhandler)(__CPPARGS);

typedef unsigned int (far *s_arrayptr);

void interrupt handler(__CPPARGS)
{
   s_arrayptr screen[80];
   static int count;

/* For a color screen the video memory is at B800:0000.
   For a monochrome system use B000:000 */
   screen[0] = (s_arrayptr) MK_FP(0xB800,0);

/* increase the counter and keep it within 0 to 9 */
   count++;
   count %= 10;

/* put the number on the screen */
   screen[0][79] = count + '0' + ATTR;

/* call the old interrupt handler */
   oldhandler();
}
```

```
int main(void)
{

/* get the address of the current clock
   tick interrupt */
oldhandler = getvect(INTR);

/* install the new interrupt handler */
setvect(INTR, handler);

/* * *
_psp is the starting address of the program in memory.  The top of the
  stack is the end of the program.

Using _SS and _SP together we can get the end of the stack.  You may want
  to allow a bit of safety space to insure that enough room is being
  allocated ie:

   (_SS + ((_SP + safety space)/16) - _psp)
* * */

keep(0, (_SS + (_SP/16) - _psp));
return 0;
}
```

/* localeconv example */

```
#include <locale.h>
#include <stdio.h>

int main(void)
{
   struct lconv ll;
   struct lconv *conv = &ll;

/* read the locality conversion structure */
   conv = localeconv();

/* display the structure */
  printf("Decimal Point                    : %s\n", conv-> decimal_point);
  printf("Thousands Separator              : %s\n", conv-> thousands_sep);
  printf("Grouping                         : %s\n", conv-> grouping);
  printf("International Currency symbol : %s\n", conv-> int_curr_symbol);
  printf("$ thousands separator            : %s\n", conv-> mon_thousands_sep);
  printf("$ grouping                       : %s\n", conv-> mon_grouping);
  printf("Positive sign                    : %s\n", conv-> positive_sign);
  printf("Negative sign                    : %s\n", conv-> negative_sign);
  printf("International fraction digits : %d\n", conv-> int_frac_digits);
  printf("Fraction digits                  : %d\n", conv-> frac_digits);
  printf("Positive $ symbol precedes       : %d\n", conv-> p_cs_precedes);
  printf("Positive sign space separation: %d\n", conv-> p_sep_by_space);
  printf("Negative $ symbol precedes       : %d\n", conv-> n_cs_precedes);
  printf("Negative sign space separation: %d\n", conv-> n_sep_by_space);
  printf("Positive sign position           : %d\n", conv-> p_sign_posn);
  printf("Negative sign position           : %d\n", conv-> n_sign_posn);
  return 0;
}
```

/* setlocale example */

```
#include <locale.h>
#include <stdio.h>

int main(void)
{
   char *old_locale;

   /* The only locale supported in Borland C++ is "C" */
   old_locale = setlocale(LC_ALL,"C");
   printf("Old locale was %s\n",old_locale);

   return 0;
}
```

/* locking example */

```
#include <io.h>
#include <fcntl.h>
#include <process.h>
#include <share.h>
#include <stdio.h>
#include <sys\locking.h>

int main(void)
{
  int handle, status;
  long length;

  /* must have DOS SHARE.EXE loaded for file locking to function */
  handle = sopen("c:\\autoexec.bat", O_RDONLY,SH_DENYNO);
  if (handle < 0) {
    printf("sopen failed\n");
    exit(1);
  }
  length = filelength(handle);
  status = locking(handle,LK_LOCK,length/2);
  if (status == 0)
    printf("lock succeeded\n");
  else
    perror("lock failed");
  status = locking(handle,LK_UNLCK,length/2);
  if (status == 0)
    printf("unlock succeeded\n");
  else
    perror("unlock failed");
  close(handle);
  return 0;
}
```

/* lock example */

```
#include <io.h>
#include <fcntl.h>
#include <sys\stat.h>
#include <process.h>
#include <share.h>
#include <stdio.h>

int main(void)
{
   int handle, status;
   long length;

   /* Must have DOS Share.exe loaded for */
   /* file locking to function properly */

   handle = sopen("c:\\autoexec.bat",
      O_RDONLY,SH_DENYNO,S_IREAD);

   if (handle < 0)
   {
      printf("sopen failed\n");
      exit(1);
   }

   length = filelength(handle);
   status = lock(handle,0L,length/2);

   if (status == 0)
      printf("lock succeeded\n");
   else
      printf("lock failed\n");

   status = unlock(handle,0L,length/2);

   if (status == 0)
      printf("unlock succeeded\n");
   else
      printf("unlock failed\n");

   close(handle);
   return 0;
}
```

/* unlock example */

```
#include <io.h>
#include <fcntl.h>
#include <sys\stat.h>
#include <process.h>
#include <share.h>
#include <stdio.h>

int main(void)
{
   int handle, status;
   long length;

   handle = sopen("c:\\autoexec.bat",O_RDONLY,SH_DENYNO,S_IREAD);

   if (handle < 0)
   {
       printf("sopen failed\n");
       exit(1);
   }

   length = filelength(handle);
   status = lock(handle,0L,length/2);

   if (status == 0)
      printf("lock succeeded\n");
   else
      printf("lock failed\n");

   status = unlock(handle,0L,length/2);

   if (status == 0)
      printf("unlock succeeded\n");
   else
      printf("unlock failed\n");

   close(handle);
   return 0;
}
```

/* log example */

```
#include <math.h>
#include <stdio.h>

int main(void)
{
   double result;
   double x = 8.6872;

   result = log(x);
   printf("The natural log of %lf is %lf\n", x, result);

   return 0;
}
```

/* log10 example */

```
#include <math.h>
#include <stdio.h>

int main(void)
{
   double result;
   double x = 800.6872;

   result = log10(x);
   printf("The common log of %lf is %lf\n", x, result);

   return 0;
}
```

/* _lrotl and _lrotr example */

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

/* function prototypes */

int lrotl_example(void);
int lrotr_example(void);

/* lrotl example */

int lrotl_example(void)
{
   unsigned long result;
   unsigned long value = 100;

   result = _lrotl(value,1);
   printf("The value %lu rotated left one bit is: %lu\n", value, result);

   return 0;
}

/* lrotr example */

int lrotr_example(void)
{
   unsigned long result;
   unsigned long value = 100;

   result = _lrotr(value,1);
   printf("The value %lu rotated right one bit is: %lu\n", value, result);

   return 0;
}

int main(void)
{
   lrotl_example();
   lrotr_example();
   return 0;
}
```

/* _rotl and _rotr example */

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

/* rotl example */

int rotl_example(void)
{
   unsigned value, result;

   value = 32767;
   result = _rotl(value, 1);
   printf("The value %u rotated left one bit is: %u\n", value, result);
   return 0;
}

/* rotr example */

int rotr_example(void)
{
   unsigned value, result;

   value = 32767;
   result = _rotr(value, 1);
   printf("The value %u rotated right one bit is: %u\n", value, result);
   return 0;
}

int main(void)
{
   rotl_example();
   rotr_example();
   return 0;
}
```

/* _makepath example */

```
#include <dir.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  char s[_MAX_PATH];
  char drive[_MAX_DRIVE];
  char dir[_MAX_DIR];
  char file[_MAX_FNAME];
  char ext[_MAX_EXT];

  getcwd(s,_MAX_PATH);            /* get current working directory */
 if (s[strlen(s)-1] != '\\')
     strcat(s,"\\");              /* append a trailing \ character */
  _splitpath(s,drive,dir,file,ext); /* split the string to separate
    elems */
  strcpy(file,"DATA");
  strcpy(ext,".TXT");
  _makepath(s,drive,dir,file,ext); /* merge everything into one string */
  puts(s);                        /* display resulting string */
  return 0;
}
```

/* _splitpath example */

```
#include <dir.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  char s[_MAX_PATH];
  char drive[_MAX_DRIVE];
  char dir[_MAX_DIR];
  char file[_MAX_FNAME];
  char ext[_MAX_EXT];

  /* get current working directory */
  getcwd(s,_MAX_PATH);
  if (s[strlen(s)-1] != '\\')

  /* append a trailing \ character */
    strcat(s,"\\");

  /* split the string to separate elems */
  _splitpath(s,drive,dir,file,ext);
  strcpy(file,"DATA");
  strcpy(ext,".TXT");

  /* merge everything into one string */
  _makepath(s,drive,dir,file,ext);

  /* display resulting string */
  puts(s);
  return 0;
}
```

/* fnsplit example */

```
#include <stdlib.h>
#include <stdio.h>
#include <dir.h>

int main(void)
{
    char *s;
    char drive[MAXDRIVE];
    char dir[MAXDIR];
    char file[MAXFILE];
    char ext[MAXEXT];
    int flags;

    s=getenv("COMSPEC"); /* get the comspec environment parameter */
    flags=fnsplit(s,drive,dir,file,ext);

    printf("Command processor info:\n");
    if(flags & DRIVE)
       printf("\tdrive: %s\n",drive);
    if(flags & DIRECTORY)
       printf("\tdirectory: %s\n",dir);
    if(flags & FILENAME)
       printf("\tfile: %s\n",file);
    if(flags & EXTENSION)
       printf("\textension: %s\n",ext);

    return 0;
}
```

/* fnmerge example */

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

int main(void)
{
   char s[MAXPATH];
   char drive[MAXDRIVE];
   char dir[MAXDIR];
   char file[MAXFILE];
   char ext[MAXEXT];

   getcwd(s,MAXPATH);              /* get the current working directory */
   strcat(s,"\\");                 /* append on a trailing character */
   fnsplit(s,drive,dir,file,ext);  /* split the string to separate elems */
   strcpy(file,"DATA");
   strcpy(ext,".TXT");
   fnmerge(s,drive,dir,file,ext);   /* merge everything into one string */
   puts(s);                         /* display resulting string */

   return 0;
}
```

/* memmove example */

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

int main(void)
{
  char *dest = "abcdefghijklmnopqrstuvwxyz0123456789";
  char *src = "******************************";
  printf("destination prior to memmove: %s\n", dest);
  memmove(dest, src, 26);
  printf("destination after memmove:    %s\n", dest);
  return 0;
}
```

/* memccpy example */

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

int main(void)
{
   char *src = "This is the source string";
   char dest[50];
   char *ptr;

   ptr = (char *) memccpy(dest, src, 'c', strlen(src));

   if (ptr)
   {
      *ptr = '\0';
      printf("The character was found: %s\n", dest);
   }
   else
      printf("The character wasn't found\n");
   return 0;
}
```

/* memcpy example */

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

int main(void)
{
   char src[] = "******************************";
   char dest[] = "abcdefghijlkmnopqrstuvwxyz0123456709";
   char *ptr;

   printf("destination before memcpy: %s\n", dest);
   ptr = (char *) memcpy(dest, src, strlen(src));
   if (ptr)
      printf("destination after memcpy: %s\n", dest);
   else
      printf("memcpy failed\n");
   return 0;
}
```

/* memcmp example */

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

int main(void)
{
   char *buf1 = "aaa";
   char *buf2 = "bbb";
   char *buf3 = "ccc";

   int stat;

   stat = memcmp(buf2, buf1, strlen(buf2));
   if (stat > 0)
      printf("buffer 2 is greater than buffer 1\n");
   else
      printf("buffer 2 is less than buffer 1\n");

   stat = memcmp(buf2, buf3, strlen(buf2));
   if (stat > 0)
      printf("buffer 2 is greater than buffer 3\n");
   else
      printf("buffer 2 is less than buffer 3\n");

   return 0;
}
```

/* memicmp example */

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

int main(void)
{
   char *buf1 = "ABCDE123";
   char *buf2 = "abcde456";
   int stat;
   stat = memicmp(buf1, buf2, 5);
   printf("The strings to position 5 are ");
   if (stat)
      printf("not ");
   printf("the same\n");
   return 0;
}
```

/* _dos_open example */

```
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <dos.h>

int main(void)
{
  int handle;
  unsigned nbytes;
  char msg[] = "Hello world\n";
  if (_dos_open("TEST.$$$", O_RDWR, &handle) != 0) {
    perror("Unable to open TEST.$$$");
    return 1;
  }
  if (_dos_write(handle, msg, strlen(msg),&nbytes) != 0)
    perror("Unable to write to TEST.$$$");
  printf("%u bytes written to TEST.$$$\n",nbytes);
  _dos_close(handle);
  return 0;
}
```

/* _rtl_open example */

```
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{
   int handle;
   char msg[] = "Hello world";

   if ((handle = _rtl_open("TEST.$$$", O_RDWR)) == -1)
   {
      perror("Error:");
      return 1;
   }
   _rtl_write(handle, msg, strlen(msg));
   _rtl_close(handle);
   return 0;
}
```

/* sopen example */

```
/*      Load share before running this example.
*/
 #include <io.h>
 #include <fcntl.h>
 #include <sys\stat.h>
 #include <process.h>
 #include <share.h>
 #include <stdio.h>
 #include       <stdlib.h>

 int main(void)
 {
    int handle,
        handle1;

    handle = sopen("c:\\autoexec.bat", O_RDONLY, SH_DENYWR, S_IREAD);

    if      (handle == -1)
    {
      perror (sys_errlist[errno]);
      exit (1);
    }

    if (!handle)
    {
       printf("sopen failed\n");
       exit(1);
    }

    /*      Attempt sopen for write.
    */
    handle1 = sopen("c:\\autoexec.bat", O_RDONLY, SH_DENYWR, S_IREAD);

    if      (handle1 == -1)
    {
      perror (sys_errlist[errno]);
      exit (1);
    }

    if (!handle1)
    {
       printf("sopen failed\n");
       exit(1);
    }

    close (handle);
    close (handle1);
    return 0;
}
```

/* open example */

```
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{
   int handle;
   char msg[] = "Hello world";

   if ((handle = open("TEST.$$$", O_CREAT | O_TEXT)) == -1)
   {
      perror("Error:");
      return 1;
   }
   write(handle, msg, strlen(msg));
   close(handle);
   return 0;
}
```

/* cprintf example */

```
#include <conio.h>

int main(void)
{
   /* clear the screen */
   clrscr();

   /* create a text window */
   window(10, 10, 80, 25);

   /* output some text in the window */
   cprintf("Hello world\r\n");

   /* wait for a key */
   getch();
   return 0;
}
```

/* fprintf example */

```
#include <stdio.h>

int main(void)
{
   FILE *stream;
   int i = 100;
   char c = 'C';
   float f = 1.234;

   /* open a file for update */
   stream = fopen("DUMMY.FIL", "w+");

   /* write some data to the file */
   fprintf(stream, "%d %c %f", i, c, f);

   /* close the file */
   fclose(stream);
   return 0;
}
```

/* printf example */

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

#define I 555
#define R 5.5

int main(void)
{
   int i,j,k,l;
   char buf[7];
   char *prefix = buf;
   char tp[20];
   printf("prefix  6d      6o      8x        10.2e        "
          "10.2f\n");
   strcpy(prefix,"%");
   for (i = 0; i < 2; i++)
   {
      for (j = 0; j < 2; j++)
         for (k = 0; k < 2; k++)
            for (l = 0; l < 2; l++)
            {
               if (i==0)  strcat(prefix,"-");
               if (j==0)  strcat(prefix,"+");
               if (k==0)  strcat(prefix,"#");
               if (l==0)  strcat(prefix,"0");
               printf("%5s |",prefix);
               strcpy(tp,prefix);
               strcat(tp,"6d |");
               printf(tp,I);
               strcpy(tp,"");
               strcpy(tp,prefix);
               strcat(tp,"6o |");
               printf(tp,I);
               strcpy(tp,"");
               strcpy(tp,prefix);
               strcat(tp,"8x |");
               printf(tp,I);
               strcpy(tp,"");
               strcpy(tp,prefix);
               strcat(tp,"10.2e |");
               printf(tp,R);
               strcpy(tp,prefix);
               strcat(tp,"10.2f |");
               printf(tp,R);
               printf("  \n");
               strcpy(prefix,"%");
            }
      }
   return 0;
}
```

/* sprintf example */

```
#include <stdio.h>
#include <math.h>

int main(void)
{
   char buffer[80];

   sprintf(buffer, "An approximation of Pi is %f\n", M_PI);
   puts(buffer);
   return 0;
}
```

/* vfprintf example */

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>

FILE *fp;

int vfpf(char *fmt, ...)
{
   va_list argptr;
   int cnt;

   va_start(argptr, fmt);
   cnt = vfprintf(fp, fmt, argptr);
   va_end(argptr);

   return(cnt);
}

int main(void)
{
   int inumber = 30;
   float fnumber = 90.0;
   char string[4] = "abc";

   fp = tmpfile();
   if (fp == NULL)
   {
      perror("tmpfile() call");
      exit(1);
   }

   vfpf("%d %f %s", inumber, fnumber, string);
   rewind(fp);
   fscanf(fp,"%d %f %s", &inumber, &fnumber, string);
   printf("%d %f %s\n", inumber, fnumber, string);
   fclose(fp);

   return 0;
}
```

/* vprintf example */

```
#include <stdio.h>
#include <stdarg.h>

int vpf(char *fmt, ...)
{
   va_list argptr;
   int cnt;

   va_start(argptr, fmt);
   cnt = vprintf(fmt, argptr);
   va_end(argptr);

   return(cnt);
}

int main(void)
{
   int inumber = 30;
   float fnumber = 90.0;
   char *string = "abc";

   vpf("%d %f %s\n",inumber,fnumber,string);

   return 0;
}
```

/* vsprintf example */

```
#include <stdio.h>
#include <conio.h>
#include <stdarg.h>

char buffer[80];

int vspf(char *fmt, ...)
{
   va_list argptr;
   int cnt;

   va_start(argptr, fmt);
   cnt = vsprintf(buffer, fmt, argptr);
   va_end(argptr);

   return(cnt);
}

int main(void)
{
   int inumber = 30;
   float fnumber = 90.0;
   char string[4] = "abc";

   vspf("%d %f %s", inumber, fnumber, string);
   printf("%s\n", buffer);
   return 0;
}
```

/* _dos_read example */

```
#include <stdio.h>
#include <fcntl.h>
#include <dos.h>

int main(void)
{
  int handle;
  unsigned bytes;
  char buf[10];

  /* Looks for a file in the current directory named TEST.$$$ and
    attempts to read 10 bytes from it. To use this example you
    should create the file TEST.$$$ */
  if (_dos_open("TEST.$$$", O_RDONLY, &handle) != 0) {
    perror("Unable to open TEST.$$$");
    return 1;
  }
  if (_dos_read(handle, buf, 10, &bytes) != 0) {
    perror("Unable to read from TEST.$$$");
    return 1;
  }
  else
    printf("_dos_read: %d bytes read.\n", bytes);
  return 0;
}
```

/* _rtl_read example */

```
#include <stdio.h>
#include <io.h>
#include <alloc.h>
#include <fcntl.h>
#include <process.h>
#include <sys\stat.h>

int main(void)
{
   void *buf;
   int handle, bytes;

   buf = malloc(10);

/*
Looks for a file in the current directory named TEST.$$$ and attempts to
  read 10 bytes from it. To use this example you should create the file
  TEST.$$$
 */
   if ((handle =
        open("TEST.$$$", O_RDONLY | O_BINARY, S_IWRITE | S_IREAD)) == -1)
   {
      printf("Error Opening File\n");
      exit(1);
   }

   if ((bytes = _rtl_read(handle, buf, 10)) == -1) {
      printf("Read Failed.\n");
      exit(1);
   }
   else {
      printf("_rtl_read: %d bytes read.\n", bytes);
   }
   return 0;
}
```

/* read example */

```
#include <stdio.h>
#include <io.h>
#include <alloc.h>
#include <fcntl.h>
#include <process.h>
#include <sys\stat.h>

int main(void)
{
   void *buf;
   int handle, bytes;

   buf = malloc(10);

/*
Looks for a file in the current directory named TEST.$$$ and attempts to
  read 10 bytes from it. To use this example you should create the file
  TEST.$$$.
 */
   if ((handle =
      open("TEST.$$$", O_RDONLY | O_BINARY, S_IWRITE | S_IREAD)) == -1)
   {
      printf("Error Opening File\n");
      exit(1);
   }

   if ((bytes = read(handle, buf, 10)) == -1) {
      printf("Read Failed.\n");
      exit(1);
   }
   else {
      printf("Read: %d bytes read.\n", bytes);
   }
   return 0;
}
```

/* farrealloc example */

```
#include <stdio.h>
#include <alloc.h>

int main(void)
{
   char far *fptr;
   char far *newptr;

   fptr = (char far *) farmalloc(16);
   printf("First address: %Fp\n", fptr);

/*
We use a second pointer, newptr, so that in the case of farrealloc()
  returning NULL, our original pointer is not set to NULL.
 */

   newptr = (char far *) farrealloc(fptr,64);
   printf("New address  : %Fp\n", newptr);
   if (newptr != NULL)
      farfree(newptr);
   return 0;
}
```

/* realloc example */

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

int main(void)
{
   char *str;

   /* allocate memory for string */
   str = (char *) malloc(10);

   /* copy "Hello" into string */
   strcpy(str, "Hello");

   printf("String is %s\n  Address is %p\n", str, str);
   str = (char *) realloc(str, 20);
   printf("String is %s\n  New address is %p\n", str, str);

   /* free memory */
   free(str);

   return 0;
}
```

/* cscanf example */

```
#include <conio.h>

int main(void)
{
   char string[80];

   /* clear the screen */
   clrscr();

   /* Prompt the user for input */
   cprintf("Enter a string with no spaces:");

   /* read the input */
   cscanf("%s", string);

   /* display what was read */
   cprintf("\r\nThe string entered is: %s", string);
   return 0;
}
```

/* fscanf example */

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

int main(void)
{
   int i;

   printf("Input an integer: ");

   /* read an integer from the
      standard input stream */
   if (fscanf(stdin, "%d", &i))
      printf("The integer read was: %i\n", i);
   else
   {
      fprintf(stderr, "Error reading an integer from stdin.\n");
      exit(1);
   }
   return 0;
}
```

/* scanf example */

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

int main(void)
{
   char label[20];
   char name[20];
   int entries = 0;
   int loop, age;
   double salary;

   struct Entry_struct
   {
      char  name[20];
      int   age;
      float salary;
   } entry[20];

/* Input a label as a string of characters restricting to 20 characters */
   printf("\n\nPlease enter a label for the chart: ");
   scanf("%20s", label);
   fflush(stdin);  /* flush the input stream in case of bad input */

/* Input number of entries as an integer */
  printf("How many entries will there be? (less than 20) ");
  scanf("%d", &entries);
  fflush(stdin);   /* flush the input stream in case of bad input */

/* input a name restricting input to only letters upper or lower case */
   for (loop=0;loop<entries;++loop)
   {
      printf("Entry %d\n", loop);
      printf("  Name   : ");
      scanf("%[A-Za-z]", entry[loop].name);
      fflush(stdin);  /* flush the input stream in case of bad input */

/* input an age as an integer */
      printf("  Age    : ");
      scanf("%d", &entry[loop].age);
      fflush(stdin);  /* flush the input stream in case of bad input */

/* input a salary as a float */
      printf("  Salary : ");
      scanf("%f", &entry[loop].salary);
      fflush(stdin); /* flush the input stream in case of bad input */
   }

/* Input a name, age and salary as a string, integer, and double */
   printf("\nPlease enter your name, age and salary\n");
   scanf("%20s %d %lf", name, &age, &salary);


/* Print out the data that was input */
   printf("\n\nTable %s\n",label);
   printf("Compiled by %s  age %d  $%15.2lf\n", name, age, salary);
```

```
   printf("-----------------------------------------------------\n");
   for (loop=0;loop<entries;++loop)
      printf("%4d | %-20s | %5d | %15.2lf\n",
         loop + 1,
         entry[loop].name,
         entry[loop].age,
         entry[loop].salary);
   printf("-----------------------------------------------------\n");
   return 0;
}
```

/* sscanf example */

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

char *names[4] = {"Peter", "Mike", "Shea", "Jerry"};

#define NUMITEMS 4

int main(void)
{
   int   loop;
   char  temp[4][80];

   char  name[20];
   int   age;
   long  salary;

/* clear the screen */
   clrscr();

/* create name, age and salary data */
   for (loop=0; loop < NUMITEMS; ++loop)
      sprintf(temp[loop], "%s %d %ld", names[loop], random(10) + 20,
  random(5000) + 27500L);

/* print title bar */
   printf("%4s | %-20s | %5s | %15s\n", "#", "Name", "Age", "Salary");
   printf("   --------------------------------------------------\n");

/* input a name, age and salary data */
   for (loop=0; loop < NUMITEMS; ++loop)
      {
      sscanf(temp[loop],"%s %d %ld", &name, &age, &salary);
      printf("%4d | %-20s | %5d | %15ld\n", loop + 1, name, age, salary);
      }

   return 0;
}
```

/* vfscanf example */

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>

FILE *fp;

int vfsf(char *fmt, ...)
{
   va_list  argptr;
   int cnt;

   va_start(argptr, fmt);
   cnt = vfscanf(fp, fmt, argptr);
   va_end(argptr);

   return(cnt);
}

int main(void)
{
   int inumber = 30;
   float fnumber = 90.0;
         char string[4] = "abc";

   fp = tmpfile();
   if (fp == NULL)
   {
      perror("tmpfile() call");
      exit(1);
   }
   fprintf(fp,"%d %f %s\n",inumber,fnumber,string);
   rewind(fp);

   vfsf("%d %f %s",&inumber,&fnumber,string);
   printf("%d %f %s\n",inumber,fnumber,string);
   fclose(fp);

   return 0;
}
```

/* vscanf example */

```
#include <stdio.h>
#include <conio.h>
#include <stdarg.h>

int vscnf(char *fmt, ...)
{
   va_list argptr;
   int cnt;

   printf("Enter an integer, a float, and a string (e.g. i,f,s,)\n");
   va_start(argptr, fmt);
   cnt = vscanf(fmt, argptr);
   va_end(argptr);

   return(cnt);
}

int main(void)
{
   int inumber;
   float fnumber;
   char string[80];

   vscnf("%d, %f, %s", &inumber, &fnumber, string);
   printf("%d %f %s\n", inumber, fnumber, string);

   return 0;
}
```

/* vsscanf example */

```
#include <stdio.h>
#include <conio.h>
#include <stdarg.h>

char buffer[80] = "30 90.0 abc";

int vssf(char *fmt, ...)
{
   va_list  argptr;
   int cnt;

   fflush(stdin);

   va_start(argptr, fmt);
   cnt = vsscanf(buffer, fmt, argptr);
   va_end(argptr);

   return(cnt);
}

int main(void)
{
   int inumber;
   float fnumber;
   char string[80];

   vssf("%d %f %s", &inumber, &fnumber, string);
   printf("%d %f %s\n", inumber, fnumber, string);
   return 0;
}
```

/* setbuf example */

```
#include <stdio.h>

/* BUFSIZ is defined in stdio.h */
char outbuf[BUFSIZ];

int main(void)
{
   /* attach a buffer to the standard output stream */
   setbuf(stdout, outbuf);

   /* put some characters into the buffer */
   puts("This is a test of buffered output.\n\n");
   puts("This output will go into outbuf\n");
   puts("and won't appear until the buffer\n");
   puts("fills up or we flush the stream.\n");

   /* flush the output buffer */
   fflush(stdout);

   return 0;
}
```

/* setvbuf example */

```
#include <stdio.h>

int main(void)
{
   FILE *input, *output;
   char bufr[512];

   input = fopen("file.in", "r+b");
   output = fopen("file.out", "w");

   /* set up input stream for minimal disk access,
      using our own character buffer */
  if (setvbuf(input, bufr, _IOFBF, 512) != 0)
      printf("failed to set up buffer for input file\n");
   else
      printf("buffer set up for input file\n");

   /* set up output stream for line buffering using space that
      will be obtained through an indirect call to malloc */
   if (setvbuf(output, NULL, _IOLBF, 132) != 0)
      printf("failed to set up buffer for output file\n");
   else
      printf("buffer set up for output file\n");

   /* perform file I/O here */

   /* close files */
   fclose(input);
   fclose(output);
   return 0;
}
```

/* spawnl example */

```
#include <process.h>
#include <stdio.h>
#include <conio.h>

void spawnl_example(void)
{
   int result;

   clrscr();
   result = spawnl(P_WAIT, "bcc.exe", "bcc.exe", NULL);
   if (result == -1)
   {
      perror("Error from spawnl");
      exit(1);
   }
}


int main(void)
{
  spawnl_example();
  return 0;
}
```

/* spawnle example */

```
#include <process.h>
#include <stdio.h>
#include <conio.h>

void spawnle_example(void)
 {
   int result;

   clrscr();
   result = spawnle(P_WAIT, "bcc.exe", "bcc.exe", NULL, NULL);
   if (result == -1)
   {
      perror("Error from spawnle");
      exit(1);
   }
}

int main(void)
{
  spawnle_example();
  return 0;
}
```

/* spawnlp example */

```
#include <process.h>
#include <stdio.h>
#include <errno.h>

void main(int argc, char *argv[])
{
   int i;

   printf("Command line arguments:\n");
   for (i=0; i<argc; ++i)
      printf("[%2d] : %s\n", i, argv[i]);
   printf("About to exec child with arg1 arg2 ...\n");
   spawnlp(P_WAIT, "C:\\BC5\\BIN\\BCC.EXE", "C:\\BC5\\BIN\\BCC.EXE",
  argv[1], argv[2], NULL);

   perror("exec error");
   exit(1);
}
```

/* spawnlpe example */

```
#include <process.h>
#include <stdio.h>
#include <errno.h>

int main( int argc, char *argv[], char **envp )
{
   int i;

   printf("Command line arguments:\n");

   for (i=0; i < argc; ++i)
      printf("[%2d] %s\n", i, argv[i]);

   printf("About to exec child with arg1 arg2 ...\n");
   spawnlpe(P_WAIT, "C:\\BC5\\BIN\\BCC.EXE", "C:\\BC5\\BIN\\BCC.EXE",
  argv[1], argv[2], NULL, envp);

   perror("exec error");
   exit(1);

   return 0;
}
```

/* spawnv example */

```
#include <process.h>
#include <stdio.h>
#include <errno.h>

void main(int argc, char *argv[])
{
   int i;

   printf("Command line arguments:\n");
   for (i=0; i<argc; ++i)
      printf("[%2d] : %s\n", i, argv[i]);
   printf("About to exec child with arg1 arg2 ...\n");
  spawnv(P_WAIT, "C:\\BC5\\BIN\\BCC.EXE", argv);

   perror("exec error");
   exit(1);
 }
```

/* spawnve example */

```
#include <process.h>
#include <stdio.h>
#include <errno.h>

void main(int argc, char *argv[], char **envp)
{
   int i;

   printf("Command line arguments:\n");
   for (i=0; i<argc; ++i)
      printf("[%2d] : %s\n", i, argv[i]);
   printf("About to exec child with arg1 arg2 ...\n");
  spawnve(P_WAIT, "C:\\BC5\\BIN\\TDMEM.EXE", argv, envp);

   perror("exec error");
   exit(1);
}
```

/* spawnvp example */

```
#include <process.h>
#include <stdio.h>
#include <errno.h>

void main(int argc, char *argv[])
{
   int i;

   printf("Command line arguments:\n");
   for (i=0; i<argc; ++i)
      printf("[%2d] : %s\n", i, argv[i]);
   printf("About to exec child with arg1 arg2 ...\n");
   spawnvp(P_WAIT, "C:\\BC5\\BIN\\BCC.EXE", argv);

   perror("exec error");
   exit(1);
}
```

/* spawnvpe example */

```
#include <process.h>
#include <stdio.h>
#include <errno.h>

int main( int argc, char *argv[], char **envp )
{
  int i;

  printf("Command line arguments:\n");

  for (i=0; i < argc; ++i)
     printf("[%2d] %s\n", i, argv[i]);

  printf("About to exec child with arg1 arg2 ...\n");
  spawnvpe(P_WAIT, "C:\\BC5\\BIN\\BCC.EXE", argv, envp);

  perror("exec error");
  exit(1);

   return 0;
}
```

/* strcmp example */

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

int main(void)
{
  char *buf1 = "aaa", *buf2 = "bbb", *buf3 = "ccc";
  int ptr;

  ptr = strcmp(buf2, buf1);
  if (ptr > 0)
      printf("buffer 2 is greater than buffer 1\n");
   else
      printf("buffer 2 is less than buffer 1\n");

   ptr = strcmp(buf2, buf3);
   if (ptr > 0)
      printf("buffer 2 is greater than buffer 3\n");
   else
      printf("buffer 2 is less than buffer 3\n");

   return 0;
}
```

/* strcmpi example */

```
/* strncmpi example */

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

int main(void)
{
   char *buf1 = "BBB", *buf2 = "bbb";
   int ptr;

   ptr = strcmpi(buf2, buf1);

   if (ptr > 0)
      printf("buffer 2 is greater than buffer 1\n");

   if (ptr < 0)
      printf("buffer 2 is less than buffer 1\n");

   if (ptr == 0)
      printf("buffer 2 equals buffer 1\n");

   return 0;
}
```

/* stricmp example */

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

int main(void)
{
   char *buf1 = "BBB", *buf2 = "bbb";
   int ptr;

   ptr = stricmp(buf2, buf1);

   if (ptr > 0)
      printf("buffer 2 is greater than buffer 1\n");

   if (ptr < 0)
      printf("buffer 2 is less than buffer 1\n");

   if (ptr == 0)
      printf("buffer 2 equals buffer 1\n");

   return 0;
}
```

/* _strnextc example */

```
#include <tchar.h>
#include <stdio.h>

int main()
{
   unsigned int retval = 0;
   const unsigned char *string = "ABC";

   retval = _strnextc(string);
   printf("The starting character:%c", retval);

   retval = _strnextc(++string);
   printf("\nThe next character:%c", retval);

   return 0;
}

/***
The starting character:A
The next character:B
***/
```

/* strspn example */

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

int main(void)
{
   char *string1 = "1234567890";
   char *string2 = "123DC8";
   int length;

   length = strspn(string1, string2);
   printf("Character where strings differ is at position %d\n", length);
   return 0;
}
```

/* strcspn example */

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

int main(void)
{
   char *string1 = "1234567890";
   char *string2 = "747DC8";
   int length;

   length = strcspn(string1, string2);
   printf("Character where strings intersect is at position %d\n",
          length);

   return 0;
}
```

/* _strerror example */

```
#include <stdio.h>
#include <errno.h>

int main(void)
{
   char *buffer;
   buffer = strerror(errno);
   printf("Error: %s\n", buffer);
   return 0;
}
```

/* strerror example */

```
#include <stdio.h>
#include <errno.h>

int main(void)
{
  char *buffer;
  buffer = strerror(errno);
  printf("Error: %s\n", buffer);
  return 0;
}
```

/* strlwr example */

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

int main(void)
{
   char *string = "Borland International";

   printf("string prior to strlwr: %s\n", string);
   strlwr(string);
   printf("string after strlwr:    %s\n", string);
   return 0;
}
```

/* strupr example */

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

int main(void)
{
   char *string = "abcdefghijklmnopqrstuvwxyz", *ptr;

   /* converts string to upper case characters */
   ptr = strupr(string);
   printf("%s\n", ptr);
   return 0;
}
```

/* strncmp example */

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

int  main(void)

{
   char *buf1 = "aaabbb", *buf2 = "bbbccc", *buf3 = "ccc";
   int ptr;

   ptr = strncmp(buf2,buf1,3);
   if (ptr > 0)
      printf("buffer 2 is greater than buffer 1\n");
   else
      printf("buffer 2 is less than buffer 1\n");

   ptr = strncmp(buf2,buf3,3);
   if (ptr > 0)
      printf("buffer 2 is greater than buffer 3\n");
   else
      printf("buffer 2 is less than buffer 3\n");

   return(0);
}
```

/* strncmpi Example */

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

int main(void)
{
   char *buf1 = "BBBccc", *buf2 = "bbbccc";
   int ptr;

   ptr = strncmpi(buf2,buf1,3);

   if (ptr > 0)
      printf("buffer 2 is greater than buffer 1\n");

   if (ptr < 0)
      printf("buffer 2 is less than buffer 1\n");

   if (ptr == 0)
      printf("buffer 2 equals buffer 1\n");

   return 0;
}
```

/* strnicmp Example */

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

int main(void)
{
   char *buf1 = "BBBccc", *buf2 = "bbbccc";
   int ptr;

   ptr = strnicmp(buf2, buf1, 3);

   if (ptr > 0)
      printf("buffer 2 is greater than buffer 1\n");

   if (ptr < 0)
      printf("buffer 2 is less than buffer 1\n");

   if (ptr == 0)
      printf("buffer 2 equals buffer 1\n");

   return 0;
}
```

/* strtod example */

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

int main(void)
{
   char input[80], *endptr;
   double value;

   printf("Enter a floating point number:");
   gets(input);
   value = strtod(input, &endptr);
   printf("The string is %s the number is %lf\n", input, value);
   return 0;
}
```

/* strtol example */

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

int main(void)
{
   char *string = "87654321", *endptr;
   long lnumber;

   /* strtol converts string to long integer  */
   lnumber = strtol(string, &endptr, 10);
   printf("string = %s  long = %ld\n", string, lnumber);

   return 0;
}
```

/* strtoul example */

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

int main(void)
{
   char *string = "87654321", *endptr;
   unsigned long lnumber;

   lnumber = strtoul(string, &endptr, 10);
   printf("string = %s  long = %lu\n",
   string, lnumber);

   return 0;
}
```

/* textattr example */

```
#include <conio.h>

int main(void)
{
   int i;

   clrscr();
   for (i=0; i<9; i++)
   {
       textattr(i + ((i+1) << 4));
       cprintf("This is a test\r\n");
   }

   return 0;
}
```

/* textbackground and textcolor example */

```
#include <conio.h>

int main(void)
{
   int i, j;

   clrscr();
   for (i=0; i<9; i++)
   {
       for (j=0; j<80; j++)
          cprintf("C");
       cprintf("\r\n");
       textcolor(i+1);
       textbackground(i);
   }

   return 0;
}
```

/* time example */

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

int main(void)
{
   time_t t;

   t = time(NULL);
   printf("The number of seconds since January 1, 1970 is %ld",t);
   return 0;
}
```

/* stime example */

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

int main(void)
{
  time_t t;

  t = time(NULL);

  printf("Current date is %s", ctime(&t));

  t -= 24L*60L*60L;  /* Back up to same time previous day */

  stime(&t);
  printf("\nNew date is %s", ctime(&t));

   return 0;
}
```

/* tolower example */

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

int main(void)
{
   int length, i;
   char *string = "THIS IS A STRING";

   length = strlen(string);
   for (i=0; i<length; i++)
   {
       string[i] = tolower(string[i]);
   }
   printf("%s\n",string);

   return 0;
}
```

/* _tolower example */

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

int main(void)
{
   int length, i;
   char *string = "THIS IS A STRING.";

   length = strlen(string);
   for (i = 0; i < length; i++) {
       if ((string[i] >= 'A') && (string[i] <= 'Z')){
          string[i] = _tolower(string[i]);
       }
   }

   printf("%s\n",string);
   return 0;
}
```

/* toupper example */

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

int main(void)
{
   int length, i;
   char *string = "this is a string";

   length = strlen(string);
   for (i=0; i<length; i++)
   {
      string[i] = toupper(string[i]);
   }

   printf("%s\n",string);

   return 0;
}
```

/* _toupper example */

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

int main(void)
{
   int length, i;
   char *string = "this is a string.";

   length = strlen(string);
   for (i = 0; i < length; i++) {
      if ((string[i] >= 'a') && (string[i] <= 'z')){
         string[i] = _toupper(string[i]);
         }
   }
   printf("%s\n",string);
   return 0;
}
```

/* _dos_write example */

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

int main(void)
{
  unsigned count;
  int handle;
  char buf[11] = "0123456789";

  /* create a file containing 10 bytes */
  if (_dos_creat("DUMMY.FIL", _A_NORMAL, &handle) != 0)
  {
    perror("Unable to create DUMMY.FIL");
    return 1;
  }
  if (_dos_write(handle, buf, strlen(buf), &count) != 0)
  {
    perror("Unable to write to DUMMY.FIL");
    return 1;
  }
  /* close the file */
  _dos_close(handle);
  return 0;
}
```

/* _rtl_write example */

```
#include <stdio.h>
#include <io.h>
#include <alloc.h>
#include <fcntl.h>
#include <process.h>
#include <sys\stat.h>

int main(void)
{
   void *buf;
   int handle, bytes;

   buf = malloc(200);

/*
Create a file name TEST.$$$ in the current directory and writes 200 bytes
  to it. If TEST.$$$ already exists, it's overwritten.
 */

   if ((handle = open("TEST.$$$", O_CREAT | O_WRONLY | O_BINARY,
                       S_IWRITE | S_IREAD)) == -1)
   {
      printf("Error Opening File\n");
      exit(1);
   }

   if ((bytes = _rtl_write(handle, buf, 200)) == -1) {
      printf("Write Failed.\n");
      exit(1);
   }
   printf("_rtl_write: %d bytes written.\n",bytes);

   return 0;
}
```

/* write example */

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys\stat.h>
#include <io.h>
#include <string.h>

int main(void)
{
   int handle;
   char string[40];
   int length, res;

/*
Create a file named "TEST.$$$" in the current directory and write a string
  to it.  If "TEST.$$$" already exists, it will be overwritten.
 */

   if ((handle = open("TEST.$$$", O_WRONLY | O_CREAT | O_TRUNC,
                         S_IREAD | S_IWRITE)) == -1)
   {
      printf("Error opening file.\n");
      exit(1);
   }

   strcpy(string, "Hello, world!\n");
   length = strlen(string);

   if ((res = write(handle, string, length)) != length)
   {
      printf("Error writing to the file.\n");
      exit(1);
   }
   printf("Wrote %d bytes to the file.\n", res);

   close(handle);
   return 0;
}
```

/* getcurdir example */

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

char *current_directory(char *path)
{
  strcpy(path, "X:\\");      /* fill string with form of response: X:\ */
  path[0] = 'A' + getdisk();    /* replace X with current drive letter */
  getcurdir(0, path+3);  /* fill rest of string with current directory */
  return(path);
}

int main(void)
{
  char curdir[MAXPATH];

  current_directory(curdir);
  printf("The current directory is %s\n", curdir);

  return 0;
}
```

/* getenv example */

```
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <string.h>
#include <dos.h>

int main(void)
{
   char *path, *ptr;
   int i = 0;

   /* get the current path environment */
   ptr = getenv("PATH");

   /* set up new path */
   path = (char *) malloc(strlen(ptr)+15);
   strcpy(path,"PATH=");
   strcat(path,ptr);
   strcat(path,";c:\\temp");

   /* replace the current path and display current environment */
   putenv(path);
   while (_environ[i])
       printf("%s\n",_environ[i++]);

   return 0;
}
```

/* putenv example */

```
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <string.h>
#include <dos.h>

int main(void)
{
   char *path, *ptr;
   int i = 0;

   /* get the current path environment */
   ptr = getenv("PATH");

   /* set up new path */
   path = (char *) malloc(strlen(ptr)+15);
   strcpy(path,"PATH=");
   strcat(path,ptr);
   strcat(path,";c:\\temp");

   /* replace the current path and display current environment */
   putenv(path);
   while (_environ[i])
       printf("%s\n",_environ[i++]);

   return 0;
}
```

/* getpass example */

```
#include <conio.h>

int main(void)
{
   char *password;

   password = getpass("Input a password:");
   cprintf("The password is: %s\r\n", password);
   return 0;
}
```

/* getpid example */

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

int main()
{
   printf("This program's process identification number (PID) "
          "number is %X\n", getpid());
   printf("Note: under DOS it is the PSP segment\n");
   return 0;
}
```

/* gettextinfo example */

```
#include <conio.h>

int main(void)
 {
   struct text_info ti;
   gettextinfo(&ti);
   cprintf("window left      %2d\r\n",ti.winleft);
   cprintf("window top       %2d\r\n",ti.wintop);
   cprintf("window right     %2d\r\n",ti.winright);
   cprintf("window bottom    %2d\r\n",ti.winbottom);
   cprintf("attribute        %2d\r\n",ti.attribute);
   cprintf("normal attribute %2d\r\n",ti.normattr);
   cprintf("current mode     %2d\r\n",ti.currmode);
   cprintf("screen height    %2d\r\n",ti.screenheight);
   cprintf("screen width     %2d\r\n",ti.screenwidth);
   cprintf("current x        %2d\r\n",ti.curx);
   cprintf("current y        %2d\r\n",ti.cury);
   return 0;
}
```

/* getverify example */

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>

int main(void)
{
   int verify_flag;

   printf("Enter 0 to set verify flag off\n");
   printf("Enter 1 to set verify flag on\n");

   verify_flag = getch() - 0;

   setverify(verify_flag);

   if (getverify())
      printf("DOS verify flag is on\n");
   else
      printf("DOS verify flag is off\n");

   return 0;
}
```

/* setverify example */

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>

int main(void)
{
    int verify_flag;

    printf("Enter 0 to set verify flag off\n");
    printf("Enter 1 to set verify flag on\n");

    verify_flag = getch() - 0;

    setverify(verify_flag);

    if (getverify())
       printf("DOS verify flag is on\n");
    else
       printf("DOS verify flag is off\n");

    return 0;
}
```

/* gotoxy example */

```
#include <conio.h>

int main(void)
{
    clrscr();
    gotoxy(35, 12);
    cprintf("Hello world");
    getch();
    return 0;
}
```

/* harderr example */

```
/*
This program will trap disk errors and
prompt the user for action. Try running it
with no disk in drive A: to invoke its
functions.
*/
#include <stdio.h>
#include <conio.h>
#include <dos.h>


#define IGNORE  0
#define RETRY   1
#define ABORT   2

int buf[500];

/*
define the error messages for trapping disk problems
*/
 static char *err_msg[] = {
     "write protect",
    "unknown unit",
     "drive not ready",
     "unknown command",
     "data error (CRC)",
     "bad request",
    "seek error",
     "unknown media type",
     "sector not found",
     "printer out of paper",
     "write fault",
     "read fault",
     "general failure",
    "reserved",
     "reserved",
     "invalid disk change"
};

error_win(char *msg)
{
    int retval;

    cputs(msg);

/*
prompt for user to press a key to abort, retry, ignore
*/
    while(1)
    {
  retval= getch();
  if (retval == 'a' || retval == 'A')
  {
      retval = ABORT;
      break;
```

```
  }
  if (retval == 'r' || retval == 'R')
  {
     retval = RETRY;
      break;
  }
  if (retval == 'i' || retval == 'I')
  {
      retval = IGNORE;
      break;
  }
    }

    return(retval);
}

/*
pragma warn -par reduces warnings which occur
due to the non use of the parameters
not_used1 and not_used2 to the handler.
*/
#pragma warn -par
void handler(unsigned int ax, unsigned int not_used1, unsigned int
  *not_used2)
{
  static char msg[80];
  unsigned di;
  int drive;
  int errorno;

  di= _DI;
/*
if this is not a disk error then it was
another device having trouble
*/

  if (ax < 0)
  {
     /* report the error */
     error_win("Device error");
     /* and return to the program directly requesting abort */
     _hardretn(ABORT);
  }
/* otherwise it was a disk error */
  drive = ax & 0x00FF;
  errorno = di & 0x00FF;
/* report which error it was */
  sprintf(msg, "Error: %s on drive %c\r\nA)bort, R)etry, I)gnore: ",
    err_msg[errorno], 'A' + drive);
/*
return to the program via dos interrupt 0x23 with abort, retry,
or ignore as input by the user.
*/
  _hardresume(error_win(msg));
  // return ABORT;
}
#pragma warn +par
```

```
int main(void)
{
/*
install our handler on the hardware problem interrupt
*/
  _harderr(handler);
   clrscr();
   printf("Make sure there is no disk in drive A:\n");
   printf("Press any key ....\n");
   getch();
   printf("Trying to access drive A:\n");
   printf("fopen returned %p\n",fopen("A:temp.dat", "w"));
   return 0;
}
```

/* hardresume example */

```
/*
This program will trap disk errors and
prompt the user for action. Try running it
with no disk in drive A: to invoke its
functions.
*/
#include <stdio.h>
#include <conio.h>
#include <dos.h>

#define IGNORE  0
#define RETRY   1
#define ABORT   2

int buf[500];

/*
define the error messages for trapping disk problems
*/
static char *err_msg[] = {
    "write protect",
   "unknown unit",
   "drive not ready",
    "unknown command",
    "data error (CRC)",
    "bad request",
    "seek error",
    "unknown media type",
    "sector not found",
    "printer out of paper",
    "write fault",
    "read fault",
    "general failure",
   "reserved",
   "reserved",
    "invalid disk change"
};

error_win(char *msg)
{
    int retval;

    cputs(msg);

/*
prompt for user to press a key to abort, retry, ignore
*/
    while(1)
    {
  retval= getch();
  if (retval == 'a' || retval == 'A')
  {
      retval = ABORT;
      break;
  }
```

```
  if (retval == 'r' || retval == 'R')
  {
     retval = RETRY;
     break;
  }
  if (retval == 'i' || retval == 'I')
  {
      retval = IGNORE;
      break;
  }
    }

    return(retval);
}

/*
pragma warn -par reduces warnings which occur
due to the non use of the parameters
not_used1 and not_used2 to the handler.
*/

#pragma warn -par
void handler(unsigned int ax, unsigned int not_used1, unsigned int
  *not_used2)
{
    static char msg[80];
   unsigned di;
   int drive;
   int errorno;

    di= _DI;
 /*
 if this is not a disk error then it was
 another device having trouble
 */

    if (ax < 0)
   {
      /* report the error */
      error_win("Device error");
      /* and return to the program directly requesting abort */
      _hardretn(ABORT);
    }
 /* otherwise it was a disk error */
    drive = ax & 0x00FF;
    errorno = di & 0x00FF;
 /* report which error it was */
   sprintf(msg, "Error: %s on drive %c\r\nA)bort, R)etry, I)gnore: ",
      err_msg[errorno], 'A' + drive);
 /*
 return to the program via dos interrupt 0x23 with abort, retry,
 or ignore as input by the user.
 */
   _hardresume(error_win(msg));
  // return ABORT;
 }
 #pragma warn +par
```

```
 int main(void)
 {
 /*
 install our handler on the hardware problem interrupt
 */
   _harderr(handler);
   clrscr();
   printf("Make sure there is no disk in drive A:\n");
   printf("Press any key ....\n");
    getch();
    printf("Trying to access drive A:\n");
    printf("fopen returned %p\n",fopen("A:temp.dat", "w"));
   return 0;
}
```

/* highvideo example */

```
#include <conio.h>

int main(void)
{
  clrscr();

  lowvideo();
  cprintf("Low Intensity text\r\n");
  highvideo();
  gotoxy(1,2);
  cprintf("High Intensity Text\r\n");

  return 0;
}
```

/* lowvideo example */

```
#include <conio.h>

int main(void)
{
  clrscr();

  highvideo();
   cprintf("High Intensity Text\r\n");
   lowvideo();
  gotoxy(1,2);
   cprintf("Low Intensity Text\r\n");

  return 0;
}
```

/* normvideo example */

```
#include <conio.h>

int main(void)
{
    normvideo();
    cprintf("NORMAL Intensity Text\r\n");
   return 0;
}
```

/* hypot example */

```
#include <stdio.h>
#include <math.h>

int main(void)
{
   double result;
   double x = 3.0;
    double y = 4.0;

    result = hypot(x, y);
    printf("The hypotenuse is: %lf\n", result);

    return 0;
}
```

/* imag example */

```
#include <iostream.h>
#include <complex.h>

int main(void)
{
   double x = 3.1, y = 4.2;
    complex z = complex(x,y);
    cout << "z = " << z << "\n";
   cout << "  has real part = " << real(z) << "\n";
    cout << "  and imaginary real part = " << imag(z) << "\n";
    cout << "z has complex conjugate = " << conj(z) << "\n";
   return 0;
}
```

/* insline example */

```
#include <conio.h>

int main(void)
{
    clrscr();
    cprintf("INSLINE inserts an empty line in the text window\r\n");
   cprintf("at the cursor position using the current text\r\n");
    cprintf("background color.  All lines below the empty one\r\n");
    cprintf("move down one line and the bottom line scrolls\r\n");
   cprintf("off the bottom of the window.\r\n");
    cprintf("\r\nPress any key to continue:");
   gotoxy(1, 3);
   getch();
    insline();
    getch();
   return 0;
}
```

/* intr example */

```
#include <stdio.h>
#include <string.h>
#include <dir.h>
#include <dos.h>

#define CF 1  /* Carry flag */

int main(void)
{
    char directory[80];
   struct REGPACK reg;

    printf("Enter directory to change to: ");
    gets(directory);
    reg.r_ax = 0x3B << 8;         /* shift 3Bh into  AH */
    reg.r_dx = FP_OFF(directory);
    reg.r_ds = FP_SEG(directory);
    intr(0x21, &reg);
    if (reg.r_flags & CF)
       printf("Directory change failed\n");
    getcwd(directory, 80);
    printf("The current directory is: %s\n", directory);
    return 0;
}
```

/* ioctl example */

```
#include <stdio.h>
#include <dir.h>
#include <io.h>

int main(void)
{
    int stat;

    /* use func 8 to determine if the default drive is removable */
    stat = ioctl(0, 8, 0, 0);
    if (!stat)
       printf("Drive %c is removable.\n", getdisk() + 'A');
    else
       printf("Drive %c is not removable.\n", getdisk() + 'A');
    return 0;
}
```

/* isatty example */

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

int main(void)
{
    int handle;

    handle = fileno(stdprn);
    if (isatty(handle))
       printf("Handle %d is a device type\n", handle);
    else
       printf("Handle %d isn't a device type\n", handle);
    return 0;
}
```

/* kbhit example */

```
#include <conio.h>

int main(void)
{
   cprintf("Press any key to continue:");
    while (!kbhit()) /* do nothing */ ;
    cprintf("\r\nA key was pressed...\r\n");
    return 0;
}
```

/* ldexp example */

```
#include <stdio.h>
#include <math.h>

int main(void)
{
    double value;
    double x = 2;

    /* ldexp raises 2 by a power of 3
       then multiplies the result by 2   */
    value = ldexp(x,3);
    printf("The ldexp value is: %lf\n", value);

    return 0;
}
```

/* setjmp example */

```
#include <stdio.h>
#include <setjmp.h>
#include <stdlib.h>

void subroutine(jmp_buf);

int main(void)
{

    int value;
    jmp_buf jumper;

    value = setjmp(jumper);
    if (value != 0)
    {
       printf("Longjmp with value %d\n", value);
       exit(value);
    }
    printf("About to call subroutine ... \n");
    subroutine(jumper);

    return 0;
}

void subroutine(jmp_buf jumper)
{
    longjmp(jumper,1);
}
```

/* longjmp example */

```
#include <stdio.h>
#include <setjmp.h>
#include <stdlib.h>

void subroutine(jmp_buf);

int main(void)
{

    int value;
    jmp_buf jumper;

    value = setjmp(jumper);
    if (value != 0)
    {
       printf("Longjmp with value %d\n", value);
       exit(value);
    }
    printf("About to call subroutine ... \n");
    subroutine(jumper);

    return 0;
}

void subroutine(jmp_buf jumper)
{
    longjmp(jumper,1);
}
```

/* lseek example */

```
#include <sys\stat.h>
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{
    int handle;
    char msg[] = "This is a test";
    char ch;

    /* create a file */
    handle = open("TEST.$$$", O_CREAT | O_RDWR, S_IREAD | S_IWRITE);

    /* write some data to the file */
    write(handle, msg, strlen(msg));

    /* seek to the begining of the file */
    lseek(handle, 0L, SEEK_SET);

    /* reads chars from the file until we hit EOF */
    do
    {
       read(handle, &ch, 1);
       printf("%c", ch);
    }  while (!eof(handle));

    close(handle);
    return 0;
}
```

/* malloc example */

```
#include <stdio.h>
#include <string.h>
#include <alloc.h>
#include <process.h>

int main(void)
{
    char *str;

    /* allocate memory for string */
    if ((str = (char *) malloc(10)) == NULL)
    {
       printf("Not enough memory to allocate buffer\n");
       exit(1);  /* terminate program if out of memory */
    }

    /* copy "Hello" into string */
    strcpy(str, "Hello");

    /* display string */
    printf("String is %s\n", str);

    /* free memory */
    free(str);

    return 0;
}
```

/* _matherr example */

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

int matherr (struct exception *a)
{
  if (a->type == DOMAIN)
    if (!strcmp(a->name,"sqrt")) {
      a->retval = sqrt (-(a->arg1));
    return 1;
    }
  return 0;
}

int main(void)
{
  double x = -2.0, y;
  y = sqrt(x);
  printf("Matherr corrected value: %lf\n",y);
  return 0;
}
```

/* max and min example */

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

#ifdef __cplusplus

   int max (int value1, int value2);

   int max(int value1, int value2)
   {
      return ( (value1 > value2) ? value1 : value2);
   }

#endif

int main(void)
{
    int x = 5;
    int y = 6;
    int z;
    z = max(x, y);
    printf("The larger number is %d\n", z);
    return 0;
}
```

/* memchr example */

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

int main(void)
{
    char str[17];
    char *ptr;

    strcpy(str, "This is a string");
    ptr = (char *) memchr(str, 'r', strlen(str));
    if (ptr)
       printf("The character 'r' is at position: %d\n", ptr - str);
    else
       printf("The character was not found\n");
    return 0;
}
```

/* memset example */

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

int main(void)
{
    char buffer[] = "Hello world\n";

    printf("Buffer before memset: %s\n", buffer);
    memset(buffer, '*', strlen(buffer) - 1);
    printf("Buffer after memset:  %s\n", buffer);
    return 0;
}
```

/* mkdir example */

```
#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <dir.h>

#define DIRNAME "testdir.$$$"

int main(void)
{
    int stat;

    stat = mkdir(DIRNAME);
    if (!stat)
           printf("Directory created\n");
    else
    {
       printf("Unable to create directory\n");
       exit(1);
    }

    getch();
    system("dir/p");
    getch();

    stat = rmdir(DIRNAME);
    if (!stat)
           printf("\nDirectory deleted\n");
    else
    {
           perror("\nUnable to delete directory\n");
       exit(1);
    }

    return 0;
}
```

/* rmdir example */

```
#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <dir.h>

#define DIRNAME "testdir.$$$"

int main(void)
{
    int stat;

    stat = mkdir(DIRNAME);
    if (!stat)
           printf("Directory created\n");
    else
    {
       printf("Unable to create directory\n");
       exit(1);
    }

    getch();
    system("dir/p");
    getch();

    stat = rmdir(DIRNAME);
    if (!stat)
           printf("\nDirectory deleted\n");
    else
    {
           perror("\nUnable to delete directory\n");
       exit(1);
    }

    return 0;
}
```

/* mktemp example */

```
#include <dir.h>
#include <stdio.h>

int main(void)
{
    /* fname defines the template for the
      temporary file.  */

    char *fname = "TXXXXXX", *ptr;

    ptr = mktemp(fname);
    printf("%s\n",ptr);
    return 0;
}
```

/* mktime example */

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

 char *wday[] = {"Sunday", "Monday", "Tuesday", "Wednesday",
  "Thursday", "Friday", "Saturday", "Unknown"};

int main(void)
{
    struct tm time_check;
    int year, month, day;

/*  Input a year, month and day to find the weekday for */
    printf("Year:  ");
    scanf("%d", &year);
    printf("Month: ");
    scanf("%d", &month);
    printf("Day:   ");
    scanf("%d", &day);

/*  load the time_check structure with the data */
    time_check.tm_year = year - 1900;
    time_check.tm_mon  = month - 1;
    time_check.tm_mday = day;
    time_check.tm_hour = 0;
    time_check.tm_min  = 0;
    time_check.tm_sec  = 1;
    time_check.tm_isdst = -1;

/*  call mktime to fill in the weekday field of the structure */
    if (mktime(&time_check) == -1)
       time_check.tm_wday = 7;

/*  print out the day of the week */
    printf("That day is a %s\n", wday[time_check.tm_wday]);
    return 0;
}
```

/* modf example */

```
#include <math.h>
#include <stdio.h>

int main(void)
{
    double fraction, integer;
    double number = 100000.567;

    fraction = modf(number, &integer);
    printf("The whole and fractional parts of %lf are %lf and %lf\n",
           number, integer, fraction);
    return 0;
}
```

/* movedata example */

```
#include <mem.h>

#define MONO_BASE 0xB000

char buf[80*25*2];

/* saves the contents of the monochrome screen in buffer */
void save_mono_screen(char near *buffer)
{
   movedata(MONO_BASE, 0, _DS, (unsigned)buffer, 80*25*2);
}

int main(void)
{
   save_mono_screen(buf);
   return 0;
}
```

/* movmem example */

```
#include <mem.h>
#include <alloc.h>
#include <stdio.h>
#include <string.h>

int main(void)
{
    char *source = "Borland International";
    char *destination;
    int length;

    length = strlen(source);
    destination = (char *) malloc(length + 1);
    movmem(source, destination, length);
    printf("%s\n", destination);

    return 0;
}
```

/* movetext example */

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

int main(void)
{
    char *str = "This is a test string";

    clrscr();
    cputs(str);
    getch();

    movetext(1, 1, strlen(str), 2, 10, 10);
    getch();

    return 0;
}
```

/* norm example */

```
#include <iostream.h>
#include <complex.h>

int main(void)
{
    double x = 3.1, y = 4.2;
    complex z = complex(x,y);
    cout << "z = " << z << "\n";
    cout << "  has real part = " << real(z) << "\n";
    cout << "  and imaginary real part = " << imag(z) << "\n";
    cout << "z has complex conjugate = " << conj(z) << "\n";

    double mag = sqrt(norm(z));
    double ang = arg(z);
    cout << "The polar form of z is:\n";
    cout << "   magnitude = " << mag << "\n";
    cout << "   angle (in radians) = " << ang << "\n";
    cout << "Reconstructing z from its polar form gives:\n";
    cout << "   z = " << polar(mag,ang) << "\n";
    return 0;
}
```

/* closedir and readdir example */

```
/* opendir.c - test opendir(), readdir(), closedir() */

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

void scandir(char *dirname)
{
   DIR *dir;
   struct dirent *ent;

   printf("First pass on '%s':\n",dirname);
   if ((dir = opendir(dirname)) == NULL)
   {
     perror("Unable to open directory");
     exit(1);
   }
   while ((ent = readdir(dir)) != NULL)
     printf("%s\n",ent->d_name);

   printf("Second pass on '%s':\n",dirname);
   rewinddir(dir);
   while ((ent = readdir(dir)) != NULL)
     printf("%s\n",ent->d_name);
   if (closedir(dir) != 0)
     perror("Unable to close directory");
}

void main(int argc,char *argv[])
{
   if (argc != 2)
   {
     printf("usage: opendir dirname\n");
     exit(1);
   }
   scandir(argv[1]);
   exit(0);
}
```

/* opendir example */

```
 /* opendir.c - test opendir(), readdir(), closedir() */

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

void scandir(char *dirname)
{
   DIR *dir;
   struct dirent *ent;

   printf("First pass on '%s':\n",dirname);
   if ((dir = opendir(dirname)) == NULL)
   {
     perror("Unable to open directory");
     exit(1);
   }
   while ((ent = readdir(dir)) != NULL)
     printf("%s\n",ent->d_name);

   printf("Second pass on '%s':\n",dirname);
   rewinddir(dir);
   while ((ent = readdir(dir)) != NULL)
     printf("%s\n",ent->d_name);
   if (closedir(dir) != 0)
     perror("Unable to close directory");
}

void main(int argc,char *argv[])
{
   if (argc != 2)
   {
     printf("usage: opendir dirname\n");
     exit(1);
   }
   scandir(argv[1]);
   exit(0);
}
```

/* parsfnm example */

```
#include <process.h>
#include <string.h>
#include <stdio.h>
#include <dos.h>

int main(void)
{
    char line[80];
    struct fcb blk;

    /* get file name */
    printf("Enter drive and file name (no path; e.g., a:file.dat)\n");
    gets(line);

    /* put file name in fcb */
    if (parsfnm(line, &blk, 1) == NULL)
       printf("Error in parsfm call\n");
    else
       printf("Drive #%d  Name: %11s\n", blk.fcb_drive, blk.fcb_name);

    return 0;
}
```

/* peek example */

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>

int main(void)
{
    int value = 0;

    printf("The current status of your keyboard is:\n");
    value = peek(0x0040, 0x0017);
    if (value & 1)
       printf("Right shift on\n");
    else
       printf("Right shift off\n");

    if (value & 2)
       printf("Left shift on\n");
    else
       printf("Left shift off\n");

    if (value & 4)
       printf("Control key on\n");
    else
       printf("Control key off\n");

    if (value & 8)
       printf("Alt key on\n");
    else
       printf("Alt key off\n");

    if (value & 16)
       printf("Scroll lock on\n");
    else
       printf("Scroll lock off\n");

    if (value & 32)
       printf("Num lock on\n");
    else
       printf("Num lock off\n");

    if (value & 64)
       printf("Caps lock on\n");
    else
       printf("Caps lock off\n");

    return 0;
}
```

/* perror example */

```
#include <stdio.h>

int main(void)
{
    FILE *fp;

    fp = fopen("perror.dat", "r");
    if (!fp)
       perror("Unable to open file for reading");
    return 0;
}
```

/* poke example */

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

int main(void)
{
    clrscr();
    cprintf("Make sure the scroll lock key is off and press any key\r\n");
    getch();
    poke(0x0000,0x0417,16);
    cprintf("The scroll lock is now on\r\n");
    return 0;
}
```

/* polar example */

```
#include <iostream.h>
#include <complex.h>

int main(void)
{
    double x = 3.1, y = 4.2;
    complex z = complex(x,y);
    cout << "z = " << z << "\n";
    cout << "  has real part = " << real(z) << "\n";
    cout << "  and imaginary real part = " << imag(z) << "\n";
    cout << "z has complex conjugate = " << conj(z) << "\n";

    double mag = sqrt(norm(z));
    double ang = arg(z);
    cout << "The polar form of z is:\n";
    cout << "   magnitude = " << mag << "\n";
    cout << "   angle (in radians) = " << ang << "\n";
    cout << "Reconstructing z from its polar form gives:\n";
    cout << "   z = " << polar(mag,ang) << "\n";
    return 0;
}
```

/* poly example */

```
#include <stdio.h>
#include <math.h>

/* polynomial:  x**3 - 2x**2 + 5x - 1 */

int main(void)
{
  double array[] = { -1.0, 5.0, -2.0, 1.0
};
  double result;

  result = poly(2.0, 3, array);
  printf("The polynomial: x**3 - 2.0x**2 + 5x - 1 at 2.0 is %lf\n",
  result);
  return 0;
}
```

/* pow example */

```
#include <math.h>
#include <stdio.h>

int main(void)
{
  double x = 2.0, y = 3.0;

  printf("%lf raised to %lf is %lf\n", x, y, pow(x, y));
  return 0;
}
```

/* pow10 example */

```
#include <math.h>
#include <stdio.h>

int main(void)
{
  double p = 3.0;

  printf("Ten raised to %lf is %lf\n", p, pow10(p));
  return 0;
}
```

/* putch example */

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

int main(void)
{
  char ch = 0;

  printf("Input a string:");
  while ((ch != '\r'))
  {
    ch = getch();
    putch(ch);
  }
  return 0;
}
```

/* raise example */

```
#include <signal.h>

int main(void)
{
    int a, b;

    a = 10;
    b = 0;
    if (b == 0)
    /* preempt divide by zero error */
       raise(SIGFPE);
    a = a / b;
    return 0;
}
```

/* rand example */

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

int main(void)
{
    int i;

    randomize();
    printf("Ten random numbers from 0 to 99\n\n");
    for(i=0; i<10; i++)
       printf("%d\n", rand() % 100);
    return 0;
}
```

/* random example */

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

/* prints a random number in the range 0 to 99 */
int main(void)
 {
    randomize();
    printf("Random number in the 0-99 range: %d\n", random (100));
    return 0;
}
```

/* randomize example */

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

int main(void)
{
    int i;

    randomize();
    printf("Ten random numbers from 0 to 99\n\n");
    for(i=0; i<10; i++)
        printf("%d\n", rand() % 100);
    return 0;
}
```

/* real example */

```
#include <iostream.h>
#include <complex.h>

int main(void)
{
    double x = 3.1, y = 4.2;
    complex z = complex(x,y);
    cout << "z = " << z << "\n";
    cout << "  has real part = " << real(z) << "\n";
    cout << "  and imaginary real part = " << imag(z) << "\n";
    cout << "z has complex conjugate = " << conj(z) << "\n";
    return 0;
}
```

/* remove example */

```
#include <stdio.h>

int main(void)
{
    char file[80];

    /* prompt for file name to delete */
    printf("File to delete: ");
    gets(file);

    /* delete the file */
    if (remove(file) == 0)
       printf("Removed %s.\n",file);
    else
       perror("remove");

    return 0;
}
```

/* rename example */

```
#include <stdio.h>

int main(void)
{
    char oldname[80], newname[80];

    /* prompt for file to rename and new name */
    printf("File to rename: ");
    gets(oldname);
    printf("New name: ");
    gets(newname);

    /* Rename the file */
    if (rename(oldname, newname) == 0)
       printf("Renamed %s to %s.\n", oldname, newname);
    else
       perror("rename");

    return 0;
}
```

/* rewind example */

```
#include <stdio.h>
#include <dir.h>

int main(void)
{
    FILE *fp;
    char *fname = "TXXXXXX", *newname, first;

    newname = mktemp(fname);
    fp = fopen(newname,"w+");
    fprintf(fp,"abcdefghijklmnopqrstuvwxyz");
    rewind(fp);
    fscanf(fp,"%c",&first);
    printf("The first character is: %c\n",first);
    fclose(fp);
    remove(newname);

    return 0;
}
```

/* rewinddir example */

```
/* opendir.c - test opendir(), readdir(), closedir() */

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

void scandir(char *dirname)
{
   DIR *dir;
   struct dirent *ent;

   printf("First pass on '%s':\n",dirname);
   if ((dir = opendir(dirname)) == NULL)
   {
     perror("Unable to open directory");
     exit(1);
   }
   while ((ent = readdir(dir)) != NULL)
     printf("%s\n",ent->d_name);

   printf("Second pass on '%s':\n",dirname);
   rewinddir(dir);
   while ((ent = readdir(dir)) != NULL)
     printf("%s\n",ent->d_name);
   if (closedir(dir) != 0)
     perror("Unable to close directory");
 }

 void main(int argc,char *argv[])
 {
   if (argc != 2)
   {
     printf("usage: opendir dirname\n");
     exit(1);
   }
   scandir(argv[1]);
   exit(0);
}
```

/* rmtmp example */

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

void main()
{
   FILE *stream;
   int i;

   /* Create temporary files */
   for (i = 1; i <= 10; i++)
   {
     if ((stream = tmpfile()) == NULL)
       perror("Could not open temporary file\n");
     else
       printf("Temporary file %d created\n", i);
   }
   /* Remove temporary files */
   if (stream != NULL)
     printf("%d temporary files deleted\n", rmtmp());
}
```

/* _searchenv example */

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

char buf[_MAX_PATH];

int main(void)
{
  /* looks for TLINK */
  _searchenv("TLINK.EXE","PATH",buf);
  if (buf[0] == '\0')
    printf("TLINK.EXE not found\n");
  else
    printf("TLINK.EXE found in %s\n", buf);

  /* looks for non-existent file */
  _searchenv("NOTEXIST.FIL","PATH",buf);
  if (buf[0] == '\0')
    printf("NOTEXIST.FIL not found\n");
  else
    printf("NOTEXIST.FIL found in %s\n", buf);
  return 0;
}

  /* Program output

      TLINK.EXE found in C:\BIN\TLINK.EXE
      NOTEXIST.FIL not found
  */
```

/* searchpath example */

```
#include <stdio.h>
#include <dir.h>

int main(void)
{
    char *p;

    /* Looks for TLINK and returns a pointer
       to the path  */
    p = searchpath("TLINK.EXE");
    printf("Search for TLINK.EXE : %s\n", p);

    /* Looks for non-existent file  */
    p = searchpath("NOTEXIST.FIL");
    printf("Search for NOTEXIST.FIL : %s\n", p);

    return 0;
}
```

/* abort example */

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

 int main(void)
 {
   printf("Calling abort()\n");
   abort();
   return 0; /* This is never reached */
 }
```

/* access example */

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

int file_exists(char *filename);

int main(void)
{
 printf("Does NOTEXIST.FIL exist: %s\n",
           file_exists("NOTEXISTS.FIL") ? "YES" : "NO");
 return 0;
}

int file_exists(char *filename)
{
  return (access(filename, 0) == 0);
}
```

/* arg example */

```
#include <iostream.h>
#include <complex.h>

int main(void)
{
   double x = 3.1, y = 4.2;
   complex z = complex(x,y);
   cout << "z = " << z << "\n";
   cout << "  has real part = " << real(z) << "\n";
   cout << "  and imaginary part = " << imag(z) << "\n";
   cout << "z has complex conjugate = " << conj(z) << "\n";

   double mag = sqrt(norm(z));
   double ang = arg(z);
   cout << "The polar form of z is:\n";
   cout << "   magnitude = " << mag << "\n";
   cout << "   angle (in radians) = " << ang << "\n";
   cout << "Reconstructing z from its polar form gives:\n";
   cout << "   z = " << polar(mag,ang) << "\n";
   return 0;
}
```

/* assert example */

```
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>

struct ITEM {
   int key;
   int value;
};

/* add item to list, make sure list is not  null */
void additem(struct ITEM *itemptr) {
   assert(itemptr != NULL);
   /* add item to list */
}

int main(void)
{
   additem(NULL);
   return 0;
}
```

/* atexit example */

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

void exit_fn1(void)
{
   printf("Exit function #1 called\n");
}

void exit_fn2(void)
{
   printf("Exit function #2 called\n");
}

int main(void)
{
   /* post exit function #1 */
   atexit(exit_fn1);
   /* post exit function #2 */
   atexit(exit_fn2);
   return 0;
}
```

/* atof example */

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

int main(void)
{
   float f;
   char *str = "12345.67";

   f = atof(str);
   printf("string = %s float = %f\n", str, f);
   return 0;
}
```

/* atoi example */

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

int main(void)
{
   int n;
   char *str = "12345.67";

   n = atoi(str);
   printf("string = %s integer = %d\n", str, n);
   return 0;
}
```

/* atol example */

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

int main(void)
{
   long l;
   char *lstr = "98765432";

   l = atol(lstr);
   printf("string = %s integer = %ld\n", lstr, l);
   return(0);
}
```

/* bcd example */

```
#include <iostream.h>
#include <bcd.h>

double x = 10000.0;          // ten thousand dollars
bcd a = bcd(x/3,2);          // a third, rounded to nearest penny

int main(void)
{
   cout << "share of fortune = $" << a << "\n";
   return 0;
}
```

/* bdos example */

```
#include <stdio.h>
#include <dos.h>

/* Get current drive as 'A', 'B', ... */
char current_drive(void)
{
   char curdrive;

   /* Get current disk as 0, 1, ... */
   curdrive = bdos(0x19, 0, 0);
   return('A' + curdrive);
}

int main(void)
{
   printf("The current drive is %c:\n", current_drive());
   return 0;
}
```

/* bdosptr example */

```
#include <string.h>
#include <stdio.h>
#include <dir.h>
#include <dos.h>
#include <errno.h>
#include <stdlib.h>

#define  BUFLEN  80

int main(void)
{
   char  buffer[BUFLEN];
   int   test;

   printf("Enter full pathname of a directory\n");
   gets(buffer);

   test = bdosptr(0x3B,buffer,0);
      if(test)
      {
         printf("DOS error message: %d\n", errno);
         /* See errno.h for error listings */
         exit (1);
      }

   getcwd(buffer, BUFLEN);
   printf("The current directory is: %s\n", buffer);

   return 0;
}
```

/* calloc example */

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

int main(void)
{
   char *str = NULL;

   /* allocate memory for string */
   str = (char *) calloc(10, sizeof(char));

   /* copy "Hello" into string */
   strcpy(str, "Hello");

   /* display string */
   printf("String is %s\n", str);

   /* free memory */
   free(str);

   return 0;
}
```

/* ceil and floor example */

```
#include <math.h>
#include <stdio.h>

int main(void)
{
   double number = 123.54;
   double down, up;

   down = floor(number);
   up = ceil(number);

   printf("original number     %5.2lf\n", number);
   printf("number rounded down %5.2lf\n", down);
   printf("number rounded up   %5.2lf\n", up);

   return 0;
}
```

/* cgets example */

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

int main(void)
{
   char buffer[83];
   char *p;

   /* There's space for 80 characters plus the NULL terminator */
   buffer[0] = 81;

   printf("Input some chars:");
   p = cgets(buffer);
   printf("\ncgets read %d characters: \"%s\"\n", buffer[1], p);
   printf("The returned pointer is %p, buffer[0] is at %p\n", p, &buffer);

   /* Leave room for 5 characters plus the NULL terminator */
   buffer[0] = 6;

   printf("Input some chars:");
   p = cgets(buffer);
   printf("\ncgets read %d characters: \"%s\"\n", buffer[1], p);
   printf("The returned pointer is %p, buffer[0] is at %p\n", p, &buffer);
   return 0;
}
```

/* _chain_intr example */

```
#include <dos.h>
#include <stdio.h>
#include <process.h>

#ifdef __cplusplus
    #define __CPPARGS ...
#else
    #define __CPPARGS
#endif

typedef void interrupt (*fptr)(__CPPARGS);

static void mesg(char *s)
{
  while (*s)
    bdos(2,*s++,0);
}

#pragma argsused
void interrupt handler2(unsigned bp, unsigned di)
{
  _enable();
  mesg("In handler 2.\r\n");
  if (di == 1)
    mesg("DI is 1\r\n");
  else
    mesg("DI is not 1\r\n");
  di++;
}

#pragma argsused
void interrupt handler1(unsigned bp, unsigned di)
{
  _enable();
  mesg("In handler 1.\r\n");
  if (di == 0)
    mesg("DI is 0\r\n");
  else
    mesg("DI is not 0\r\n");
  di++;
  mesg("Chaining to handler 2.\r\n");
  _chain_intr((fptr) handler2);
}

int main()
{
  _dos_setvect(128,(fptr) handler1);
  printf("About to generate interrupt 128\n");
  _DI = 0;
  geninterrupt(128);
  printf("DI was 0 before interrupt, is now 0x%x\n",_DI);
  return 0;
}
```

/* chdir example */

```
#include <stdio.h>
#include <stdlib.h>
#include <dir.h>

char old_dir[MAXDIR];
char new_dir[MAXDIR];

int main(void)
{
   if (getcurdir(0, old_dir))
   {
      perror("getcurdir()");
      exit(1);
   }
   printf("Current directory is: \\%s\n", old_dir);

   if (chdir("\\"))
   {
      perror("chdir()");
      exit(1);
   }

   if (getcurdir(0, new_dir))
   {
      perror("getcurdir()");
      exit(1);
   }
   printf("Current directory is now: \\%s\n", new_dir);

   printf("\nChanging back to original directory: \\%s\n", old_dir);
   if (chdir(old_dir))
   {
      perror("chdir()");
      exit(1);
   }

   return 0;
}
```

/* chmod example */

```
/* NEW chmod() example: */

#include <errno.h>
#include <stdio.h>
#include <io.h>
#include <process.h>
#include <sys\stat.h>

void main(void)
{
  char filename[64];
  struct stat stbuf;
  int amode;

  printf("Enter name of file: ");
  scanf("%s", filename);
  if (stat(filename, &stbuf) != 0)
  {
    perror("Unable to get file information");
    exit(1);
  }
  if (stbuf.st_mode & S_IWRITE)
  {
    printf("Changing to read-only\n");
    amode = S_IREAD;
  }
  else
  {
    printf("Changing to read-write\n");
    amode = S_IREAD|S_IWRITE;
  }
  if (chmod(filename, amode) != 0)
  {
    perror("Unable to change file mode");
    exit(1);
  }
  exit(0);
}
```

/* chsize example */

```
#include <string.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{
   int handle;
   char buf[11] = "0123456789";

   /* create text file containing 10 bytes */
   handle = open("DUMMY.FIL", O_CREAT);
   write(handle, buf, strlen(buf));

   /* truncate the file to 5 bytes in size */
   chsize(handle, 5);

   /* close the file */
   close(handle);
   return 0;
}
```

/* _clear87 and _status87 example */

```
#include <stdio.h>
#include <float.h>

int main(void)
{
   float x;
   double y = 1.5e-100;

   printf("\nStatus 87 before error: %X\n", _status87());

   x = y; /* create underflow and precision loss */
   printf("Status 87 after  error: %X\n", _status87());

   _clear87();
   printf("Status 87 after  clear: %X\n", _status87());

   y = x;

   return 0;
}
```

/* clearerr example */

```
#include <stdio.h>

int main(void)
{
   FILE *fp;
   char ch;

   /* open a file for writing */
   fp = fopen("DUMMY.FIL", "w");

   /* force an error condition by attempting to read */
   ch = fgetc(fp);
   printf("%c\n",ch);

   if (ferror(fp))
   {
      /* display an error message */
      printf("Error reading from DUMMY.FIL\n");

      /* reset the error and EOF indicators */
      clearerr(fp);
   }

   fclose(fp);
   return 0;
}
```

/* clock example */

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

int main(void)
{
   clock_t start, end;
   start = clock();

   delay(2000);

   end = clock();
   printf("The time was: %f\n", (end - start) / CLK_TCK);

   return 0;
}
```

/* clreol example */

```
#include <conio.h>

int main(void)

{
   clrscr();
   cprintf("The function CLREOL clears all characters from the\r\n");
   cprintf("cursor position to the end of the line within the\r\n");
   cprintf("current text window, without moving the cursor.\r\n");
   cprintf("Press any key to continue . . .");
   gotoxy(14, 4);
   getch();

   clreol();
   getch();

   return 0;
}
```

/* clrscr example */

```
#include <conio.h>

int main(void)
{
   int i;

   clrscr();
   for (i = 0; i < 20; i++)
      cprintf("%d\r\n", i);
   cprintf("\r\nPress any key to clear screen");
   getch();

   clrscr();
   cprintf("The screen has been cleared!");
   getch();

   return 0;
}
```

/* complex example */

```
#include <iostream.h>
#include <complex.h>

int main(void)
{
   double x = 3.1, y = 4.2;
   complex z = complex(x,y);
   cout << "z = "<< z << "\n";
   cout << " and imaginary real part = " << imag(z) << "\n";
   cout << "z has complex conjugate =  " << conj(z) << " \n";
   return 0;
}
```

/* conj example */

```
#include <iostream.h>
#include <complex.h>

int main(void)
{
   double x = 3.1, y = 4.2;
   complex z = complex(x,y);
   cout << "z = "<< z << "\n";
   cout << " and imaginary real part = " << imag(z) << "\n";
   cout << "z has complex conjugate =  " << conj(z) << " \n";
   return 0;
}
```

/* country example */

```
#include <dos.h>
#include <stdio.h>

#define USA 0

int main(void)
{
   struct COUNTRY country_info;

   country(USA, &country_info);
   printf("The currency symbol for the USA is: %s\n",
           country_info.co_curr);
   return 0;
}
```

/* cputs example */

```
#include <conio.h>

int main(void)
{
   /* clear the screen */
   clrscr();

   /* create a text window */
   window(10, 10, 80, 25);

   /* output some text in the window */
   cputs("This is within the window\r\n");

   /* wait for a key */
   getch();
   return 0;
}
```

/* creattemp example */

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

int main(void)
{
   int handle;
   char pathname[128];

   strcpy(pathname, "\\");

   /* create a unique file in the root directory */
   handle = creattemp(pathname, 0);

   printf("%s was the unique file created.\n", pathname);
   close(handle);
   return 0;
}
```

/* ctrlbrk example */

```
#include <stdio.h>
#include <dos.h>

#define ABORT 0

int c_break(void)
{
   printf("Control-Break pressed.  Program aborting ...\n");
  return (ABORT);
}

void main(void)
{
   ctrlbrk(c_break);
   for(;;)
   {
      printf("Looping... Press <Ctrl-Break> to quit:\n");
   }
}
```

/* delline example */

```
#include <conio.h>

int main(void)
{
   clrscr();
   cprintf("The function DELLINE deletes the line containing the\r\n");
   cprintf("cursor and moves all lines below it one line up.\r\n");
   cprintf("DELLINE operates within the currently active text\r\n");
   cprintf("window.  Press any key to continue . . .");
   gotoxy(1,2);  /* Move the cursor to the second line and first column */
   getch();

   delline();
   getch();

   return 0;
}
```

/* difftime example */

```
#include <time.h>
#include <stdio.h>
#include <dos.h>
#include <conio.h>

int main(void)
{
   time_t first, second;

   clrscr();
   first = time(NULL);  /* Gets system
                           time */
   delay(2000);         /* Waits 2 secs */
   second = time(NULL); /* Gets system time
                           again */

   printf("The difference is: %f seconds\n",difftime(second,first));
   getch();

   return 0;
}
```

/* dosexterr example */

```
#include <stdio.h>
#include <dos.h>

int main(void)
{
   FILE *fp;
   struct DOSERROR info;

   fp = fopen("perror.dat","r");
   if (!fp) perror("Unable to open file for reading");
   dosexterr(&info);

   printf("Extended DOS error information:\n");
   printf("   Extended error: %d\n",info.de_exterror);
   printf("            Class: %x\n",info.de_class);
   printf("           Action: %x\n",info.de_action);
   printf("      Error Locus: %x\n",info.de_locus);

   return 0;
}
```

/* dostounix example */

```
#include <time.h>
#include <stddef.h>
#include <dos.h>
#include <stdio.h>

int main(void)
{
  time_t t;
  struct time d_time;
  struct date d_date;
  struct tm *local;

  getdate(&d_date);
  gettime(&d_time);

  t = dostounix(&d_date, &d_time);
  local = localtime(&t);
  printf("Time and Date: %s\n", asctime(local));

  return 0;
}
```

/* __emit__ example */

```
#include <dos.h>

int main(void)
{
/* Emit code that will generate a print screen via int 5 */
   __emit__(0xcd,0x05);
   return 0;
}
```

/* eof example */

```
#include <sys\stat.h>
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{
   int handle;
   char msg[] = "This is a test";
   char ch;

   /* create a file */
   handle = open("DUMMY.FIL",
                 O_CREAT | O_RDWR,
                 S_IREAD | S_IWRITE);

   /* write some data to the file */
   write(handle, msg, strlen(msg));

   /* seek to the beginning of the file */
   lseek(handle, 0L, SEEK_SET);

   /* reads chars from the file until it reaches EOF */
   do
   {
      read(handle, &ch, 1);
      printf("%c", ch);
   } while (!eof(handle));

   close(handle);
   return 0;
}
```

/* exp and expl example */

```
#include <stdio.h>
#include <math.h>

int main(void)
{
   double result;
   double x = 4.0;

   result = exp(x);
   printf("'e' raised to the power \
          of %lf (e ^ %lf) = %lf\n",
          x, x, result);

   return 0;
}
```

/* farcalloc example */

```
#include <stdio.h>
#include <alloc.h>
#include <string.h>
#include <dos.h>

int main(void)
{
   char far *fptr;
   char *str = "Hello";

   /* allocate memory for the far pointer */
   fptr = (char far *) farcalloc(10, sizeof(char));

   /* copy "Hello" into allocated memory */
   /*
      Note: movedata is used because you might be in a small data model, in
      which case a normal string copy routine can not be used since it
      assumes the pointer size is near.
   */
   movedata(FP_SEG(str), FP_OFF(str),
      FP_SEG(fptr), FP_OFF(fptr),
      strlen(str));

   /* display string (note the F modifier) */
   printf("Far string is: %Fs\n", fptr);

   /* free the memory */
   farfree(fptr);

   return 0;
}
```

/* farmalloc example */

```
#include <stdio.h>
#include <alloc.h>
#include <string.h>
#include <dos.h>

int main(void)
{
   char far *fptr;
   char *str = "Hello";

   /* allocate memory for the far pointer */
   fptr = (char far *) farmalloc(10);

   /* copy "Hello" into allocated memory */
   /*
    Note: movedata is used because we might be in a small data model,
      in which case a normal string copy routine can not be used since it
      assumes the pointer size is near.
   */
   movedata(FP_SEG(str), FP_OFF(str),
            FP_SEG(fptr), FP_OFF(fptr),
            strlen(str) + 1);

   /* display string (note the F modifier)
*/
   printf("Far string is: %Fs\n", fptr);

   /* free the memory */
   farfree(fptr);

   return 0;
}
```

/* fclose example */

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

int main(void)
{
   FILE *fp;
   char buf[11] = "0123456789";

   /* create a file containing 10 bytes */
   fp = fopen("DUMMY.FIL", "w");
   fwrite(&buf, strlen(buf), 1, fp);

   /* close the file */
   fclose(fp);
   return 0;
}
```

/* fcloseall example */

```
#include <stdio.h>

int main(void)
{
   int streams_closed;

   /* open two streams */
   fopen("DUMMY.ONE", "w");
   fopen("DUMMY.TWO", "w");

   /* close the open streams */
   streams_closed = fcloseall();

   if (streams_closed == EOF)
      /* issue an error message */
      perror("Error");
   else
      /* print result of fcloseall() function */
      printf("%d streams were closed.\n", streams_closed);

   return 0;
}
```

/* feof example */

```
#include <stdio.h>

int main(void)
{
   FILE *stream;

   /* open a file for reading */
   stream = fopen("DUMMY.FIL", "r");

   /* read a character from the file */
   fgetc(stream);

   /* check for EOF */
   if (feof(stream))
      printf("We have reached end-of-file\n");

   /* close the file */
   fclose(stream);
   return 0;
}
```

/* ferror example */

```
#include <stdio.h>

int main(void)
{
   FILE *stream;

   /* open a file for writing */
   stream = fopen("DUMMY.FIL", "w");

   /* force an error condition by attempting to read */
   (void) getc(stream);

   if (ferror(stream))  /* test for an error on the stream */
   {
      /* display an error message */
      printf("Error reading from DUMMY.FIL\n");

      /* reset the error and EOF indicators */
      clearerr(stream);
   }

   fclose(stream);
   return 0;
}
```

/* fflush example */

```
#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <io.h>

void flush(FILE *stream);

int main(void)
{
   FILE *stream;
   char msg[] = "This is a test";

   /* create a file */
   stream = fopen("DUMMY.FIL", "w");

   /* write some data to the file */
   fwrite(msg, strlen(msg), 1, stream);

   clrscr();
   printf("Press any key to flush DUMMY.FIL:");
   getch();

   /* flush the data to DUMMY.FIL without closing it */
   flush(stream);

   printf("\nFile was flushed, Press any key to quit:");
   getch();
   return 0;
}

void flush(FILE *stream)
{
     int duphandle;

     /* flush the stream's internal buffer */
     fflush(stream);

     /* make a duplicate file handle */
     duphandle = dup(fileno(stream));

     /* close the duplicate handle to flush the DOS buffer */
     close(duphandle);
}
```

/* fgetpos and fsetpos example */

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

void showpos(FILE *stream);

int main(void)
{
   FILE *stream;
   fpos_t filepos;

   /* open a file for update */
   stream = fopen("DUMMY.FIL", "w+");

   /* save the file pointer position */
   fgetpos(stream, &filepos);

   /* write some data to the file */
   fprintf(stream, "This is a test");

   /* show the current file position */
   showpos(stream);

   /* set a new file position, display it */
   if (fsetpos(stream, &filepos) == 0)
     showpos(stream);
   else
   {
      fprintf(stderr, "Error setting file pointer.\n");
      exit(1);
   }

   /* close the file */
   fclose(stream);
   return 0;
}

void showpos(FILE *stream)
{
   fpos_t pos;

   /* display the current file pointer
      position of a stream */
   fgetpos(stream, &pos);
   printf("File position: %ld\n", pos);
}
```

/* filelength example */

```
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{
   int handle;
   char buf[11] = "0123456789";

   /* create a file containing 10 bytes */
   handle = open("DUMMY.FIL", O_CREAT);
   write(handle, buf, strlen(buf));

   /* display the size of the file */
   printf("file length in bytes: %ld\n", filelength(handle));

   /* close the file */
   close(handle);
   return 0;
}
```

/* fileno example */

```
#include <stdio.h>

int main(void)
{
   FILE *stream;
   int handle;

   /* create a file */
   stream = fopen("DUMMY.FIL", "w");

   /* obtain the file handle associated with the stream */
   handle = fileno(stream);

   /* display the handle number */
   printf("handle number: %d\n", handle);

   /* close the file */
   fclose(stream);
   return 0;
}
```

/* flushall example */

```
#include <stdio.h>

int main(void)
{
   FILE *stream;

   /* create a file */
   stream = fopen("DUMMY.FIL", "w");

   /* flush all open streams */
   printf("%d streams were flushed.\n", flushall());

   /* close the file */
   fclose(stream);
   return 0;
}
```

/* fmod and fmodl example */

```
#include <stdio.h>
#include <math.h>

int main(void)
{
   double x = 5.0, y = 2.0;
   double result;

   result = fmod(x,y);
   printf("The remainder of (%lf / %lf) is %lf\n", x, y, result);
   return 0;
}
```

/* FP_OFF, and FP_SEG example*/

```
#include <stdio.h>
#include <dos.h>

main()
{
  char *str = "Hello\n";

  printf("The address pointed to by str is %04X:%04X\n",
          FP_SEG(str), FP_OFF(str));
  printf("The address of str is %04X:%04X\n", FP_SEG(&str), FP_OFF(&str));
  return 0;
}
```

/* fread example */

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

int main(void)
{
   FILE *stream;
   char msg[] = "this is a test";
   char buf[20];

   if ((stream = fopen("DUMMY.FIL", "w+"))
       == NULL)
   {
      fprintf(stderr, "Cannot open output file.\n");
      return 1;
   }

   /* write some data to the file */
   fwrite(msg, strlen(msg)+1, 1, stream);

   /* seek to the beginning of the file */
   fseek(stream, SEEK_SET, 0);

   /* read the data and display it */
   fread(buf, strlen(msg)+1, 1, stream);
   printf("%s\n", buf);

   fclose(stream);
   return 0;
}
```

/* frexp and frexpl examples */

```
#include <math.h>
#include <stdio.h>

int main(void)
{
   double mantissa, number;
   int exponent;

   number = 8.0;
   mantissa = frexp(number, &exponent);

   printf("The number %lf is ", number);
   printf("%lf times two to the ", mantissa);
   printf("power of %d\n", exponent);

   return 0;
}
```

/* fseek example */

```
#include <stdio.h>

long filesize(FILE *stream);

int main(void)
{
   FILE *stream;

   stream = fopen("MYFILE.TXT", "w+");
   fprintf(stream, "This is a test");
   printf("Filesize of MYFILE.TXT is %ld bytes\n", filesize(stream));
   fclose(stream);
   return 0;
}

long filesize(FILE *stream)
{
   long curpos, length;

   curpos = ftell(stream);
   fseek(stream, 0L, SEEK_END);
   length = ftell(stream);
   fseek(stream, curpos, SEEK_SET);
   return length;
}
```

/* ftell example */

```
#include <stdio.h>
int main(void)
{
   FILE *stream;

   stream = fopen("MYFILE.TXT", "w+");
   fprintf(stream, "This is a test");
   printf("The file pointer is at byte %ld\n", ftell(stream));
   fclose(stream);
   return 0;
}
```

/* ftime example */

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sys\timeb.h>

/* pacific standard & daylight savings */
char *tzstr = "TZ=PST8PDT";

int main(void)
{
   struct timeb t;
   putenv(tzstr);
   tzset();
   ftime(&t);
   printf("Seconds since 1/1/1970 GMT: %ld\n", t.time);
   printf("Thousandths of a second: %d\n", t.millitm);
   printf("Difference between local time and GMT: %d\n", t._timezone);
   printf("Daylight savings in effect (1) not (0): %d\n", t.dstflag);
   return 0;
}
```

/* _fullpath example */

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

char buf[_MAX_PATH];

void main(int argc, char *argv[])
{
  for ( ; argc; argv++, argc--)
  {
    if (_fullpath(buf, argv[0], _MAX_PATH) == NULL)
      printf("Unable to obtain full path of %s\n",argv[0]);
    else
      printf("Full path of %s is %s\n",argv[0],buf);
  }
}
```

/* fwrite example */

```
#include <stdio.h>

struct mystruct
{
  int i;
  char ch;
};

int main(void)
{
   FILE *stream;
   struct mystruct s;

   if ((stream = fopen("TEST.$$$", "wb")) == NULL) /* open file TEST.$$$ */
   {
      fprintf(stderr, "Cannot open output file.\n");
      return 1;
   }
   s.i = 0;
   s.ch = 'A';
   fwrite(&s, sizeof(s), 1, stream); /* write struct s to file */
   fclose(stream); /* close file */
   return 0;
}
```

/* gcvt example */

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

int main(void)
{
   char str[25];
   double num;
   int sig = 5; /* significant digits */

   /* a regular number */
   num = 9.876;
   gcvt(num, sig, str);
   printf("string = %s\n", str);

   /* a negative number */
   num = -123.4567;
   gcvt(num, sig, str);
   printf("string = %s\n", str);

   /* scientific notation */
   num = 0.678e5;
   gcvt(num, sig, str);
   printf("string = %s\n", str);

   return(0);
}
```

/* geninterrupt example */

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

/* function prototype */
void writechar(char ch);

int main(void)
{
   clrscr();
   gotoxy(80,25);
   writechar('*');
   getch();
   return 0;
}

/*
   outputs a character at the current cursor
   position using the video BIOS to avoid
   the scrolling of the screen when writing
   to location (80,25).
*/

void writechar(char ch)
{
   struct text_info ti;
   /* grab current text settings */
   gettextinfo(&ti);
   /* interrupt 0x10 sub-function 9 */
   _AH = 9;
   /* character to be output */
   _AL = ch;
   _BH = 0;                  /* video page */
   _BL = ti.attribute;  /* video attribute */
   _CX = 1;           /* repetition factor */
   geninterrupt(0x10);  /* output the char */
}
```

/* getcbrk and setcbrk example */

```
#include <dos.h>
#include <conio.h>
#include <stdio.h>

int main(void)
{
   int break_flag;

   printf("Enter 0 to turn control break off\n");
   printf("Enter 1 to turn control break on\n");

   break_flag = getch() - 0;

   setcbrk(break_flag);

   if (getcbrk())
      printf("Cntrl-brk flag is on\n");
   else
      printf("Cntrl-brk flag is off\n");
   return 0;
}
```

/* segread example */

```
#include <stdio.h>
#include <dos.h>

int main(void)
{
   struct SREGS segs;

   segread(&segs);
   printf("Current segment register settings\n\n");
   printf("CS: %X   DS: %X\n", segs.cs, segs.ds);
   printf("ES: %X   SS: %X\n", segs.es, segs.ss);

   return 0;
}
```

/* setmem example */

```
#include <stdio.h>
#include <alloc.h>
#include <mem.h>

int main(void)
{
   char *dest;

   dest = (char *) calloc(21, sizeof(char));
   setmem(dest, 20, 'c');
   printf("%s\n", dest);

   return 0;
}
```

/* setmode example */

```
#include <fcntl.h>
#include <io.h>
#include <stdio.h>
int main (int argc, char ** argv )
(
  FILE *fp;
  int newmode;
  long where;
  char buf[256];

  fp = fopen( argv[1], "r+" );
  if ( !fp )
  {
    printf( "Couldn't open %s\n", argv[1] );
    return -1;
  }

  newmode = setmode( fileno( fp ), O_BINARY );
  if ( newmode == -1 )
  {
    printf( "Coudn't set mode of %s\n", argv[1] );
    return -2
  }

  fp->flags |= _F_BIN;
  where = ftell( fp );
  printf ("file position: %d\n", where );
  fread( buf, 1, 1, fp );
  where = ftell ( fp );
  printf( "read %c, file position: %ld\n", *buf, where );
  fclose ( fp );
  return 0;
}
```

/* signal example */

```
/* signal example */

/*
    This example installs a signal handler routine for SIGFPE,
    catches an integer overflow condition, makes an adjustment to AX
    register, and returns.  This example program MAY cause your computer
    to crash, and will produce runtime errors depending on which memory
    model is used.
*/

#pragma inline
#include <stdio.h>
#include <signal.h>

#ifdef __cplusplus
   typedef void (*fptr)(int);
#else
   typedef void (*fptr)();
#endif

void Catcher(int *reglist)
{
   signal(SIGFPE, (fptr)Catcher);  //  ******reinstall signal handler

   printf("Caught it!\n"); *(reglist + 8) = 3; /* make return AX = 3 */
}

int main(void)
{
   signal(SIGFPE, (fptr)Catcher);  /* cast Catcher to appropriate type */

   asm     mov     ax,07FFFH       /* AX = 32767 */
   asm     inc     ax              /* cause overflow */
   asm     into                    /* activate handler */

   /* The handler set AX to 3 on return. If that had not happened,
      there would have been another exception when the next 'into'
      executed after the 'dec' instruction. */

   asm     dec     ax              /* no overflow now */
   asm     into                    /* doesn't activate */
   return 0;
}
```

/* sleep example */

```
#include <dos.h>
#include <stdio.h>

int main(void)
{
   int i;

   for (i=1; i<5; i++)
   {
      printf("Sleeping for %d seconds\n", i);
      sleep(i);
   }
   return 0;
}
```

/* sqrt example */

```
#include <math.h>
#include <stdio.h>

int main(void)
{
   double x = 4.0, result;

   result = sqrt(x);
   printf("The square root of %lf is %lf\n", x, result);
   return 0;
}
```

/* srand example */

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

int main(void)
{
   int i;
   time_t t;

   srand((unsigned) time(&t));
   printf("Ten random numbers from 0 to 99\n\n");
   for(i=0; i<10; i++)
       printf("%d\n", rand() % 100);
   return 0;
}
```

/* stpcpy example */

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

int main(void)
{
   char string[10];
   char *str1 = "abcdefghi";

   stpcpy(string, str1);
   printf("%s\n", string);
   return 0;
}
```

/*strcat example */

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

int main(void)
{
   char destination[25];
   char *blank = " ", *c = "C++", *Borland = "Borland";

   strcpy(destination, Borland);
   strcat(destination, blank);
   strcat(destination, c);

   printf("%s\n", destination);
   return 0;
}
```

/* strchr example */

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

int main(void)
{
   char string[15];
   char *ptr, c = 'r';

   strcpy(string, "This is a string");
   ptr = strchr(string, c);
   if (ptr)
      printf("The character %c is at position: %d\n", c, ptr-string);
   else
      printf("The character was not found\n");
   return 0;
}
```

/* strcoll example */

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

int main(void)
{
   char *two = "International";
   char *one = "Borland";
   int check;

   check = strcoll(one, two);
   if (check == 0)
      printf("The strings are equal\n");
   if (check <  0)
      printf("%s comes before %s\n", one, two);
   if (check >  0)
      printf("%s comes before %s\n", two, one);
   return 0;
}
```

/* strcpy example */

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

int main(void)
{
   char string[10];
   char *str1 = "abcdefghi";

   strcpy(string, str1);
   printf("%s\n", string);
   return 0;
}
```

/* _strdate example */

```
#include <time.h>
#include <stdio.h>
void main(void)
{
  char datebuf[9];
  char timebuf[9];

  _strdate(datebuf);
  _strtime(timebuf);
  printf("Date: %s  Time: %s\n",datebuf,timebuf);
}
```

/* strdup example */

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

int main(void)
{
   char *dup_str, *string = "abcde";

   dup_str = strdup(string);
   printf("%s\n", dup_str);
   free(dup_str);

   return 0;
}
```

/* strftime example */

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

int main(void)
{
   struct tm *time_now;
   time_t secs_now;
   char str[80];

   tzset();
   time(&secs_now);
   time_now = localtime(&secs_now);
   strftime(str, 80,
            "It is %M minutes after %I o'clock (%Z)  %A, %B %d 19%y",
            time_now);
   printf("%s\n",str);
   return 0;
}
```

/*strlen example */

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

int main(void)
{
   char *string = "Borland International";

   printf("%d\n", strlen(string));
   return 0;
}
```

/*strncat example */

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

int main(void)
{
   char destination[25];
   char *source = " States";

   strcpy(destination, "United");
   strncat(destination, source, 7);
   printf("%s\n", destination);
   return 0;
}
```

/* strncpy example */

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

int main(void)
{
   char string[10];
   char *str1 = "abcdefghi";

   strncpy(string, str1, 3);
   string[3] = '\0';
   printf("%s\n", string);
   return 0;
}
```

/* strnset example */

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

int main(void)
{
   char *string = "abcdefghijklmnopqrstuvwxyz";
   char letter = 'x';

   printf("string before strnset: %s\n", string);
   strnset(string, letter, 13);
   printf("string after  strnset: %s\n", string);

   return 0;
}
```

/* strpbrk example */

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

int main(void)
{
   char *string1 = "abcdefghijklmnopqrstuvwxyz";
   char *string2 = "onm";
   char *ptr;

   ptr = strpbrk(string1, string2);

   if (ptr)
      printf("strpbrk found first character: %c\n", *ptr);
   else
      printf("strpbrk didn't find character in set\n");

   return 0;
}
```

/* strrchr example */

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

int main(void)
{
   char string[15];
   char *ptr, c = 'r';

   strcpy(string, "This is a string");
   ptr = strrchr(string, c);
   if (ptr)
      printf("The character %c is at position: %d\n", c, ptr-string);
   else
      printf("The character was not found\n");
   return 0;
}
```

/* strrev example */

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

int main(void)
{
   char *forward = "string";

   printf("Before strrev(): %s\n", forward);
   strrev(forward);
   printf("After strrev():  %s\n", forward);
   return 0;
}
```

/* strset example */

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

int main(void)
{
   char string[10] = "123456789";
   char symbol = 'c';

   printf("Before strset(): %s\n", string);
   strset(string, symbol);
   printf("After strset():  %s\n", string);
   return 0;
}
```

/* strstr example */

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

int main(void)
{
   char *str1 = "Borland International", *str2 = "nation", *ptr;

   ptr = strstr(str1, str2);
   printf("The substring is: %s\n", ptr);
   return 0;
}
```

/* _strtime example */

```
#include <time.h>
#include <stdio.h>
void main(void)
{
  char datebuf[9];
  char timebuf[9];

  _strdate(datebuf);
  _strtime(timebuf);
  printf("Date: %s  Time: %s\n",datebuf,timebuf);
}
```

/* strtok example */

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

int main(void)
{
   char input[16] = "abc,d";
   char *p;

   /* strtok places a NULL terminator
   in front of the token, if found */
   p = strtok(input, ",");
   if (p)   printf("%s\n", p);

   /* A second call to strtok using a NULL
   as the first parameter returns a pointer
   to the character following the token  */
   p = strtok(NULL, ",");
   if (p)   printf("%s\n", p);
   return 0;
}
```

/* strxfrm example */

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

int main(void)
{
   char *target;
   char *source = "Frank Borland";
   int length;

   /* allocate space for the target string */
   target = (char *) calloc(80, sizeof(char));

   /* copy the source over to the target and get the length */
   length = strxfrm(target, source, 80);

   /* print out the results */
   printf("%s has the length %d\n", target, length);
   return 0;
}
```

/* swab example */

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

char source[15] = "rFna koBlrna d";
char target[15];

int main(void)
{
   swab(source, target, strlen(source));
   printf("This is target: %s\n", target);
   return 0;
}
```

/* system example */

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

int main(void)
{
   printf("About to spawn command.com and run a DOS command\n");
   system("dir");
   return 0;
}
```

/* tell example */

```
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include <io.h>

int main(void)
{
   int handle;
   char msg[] = "Hello world";

   if ((handle = open("TEST.$$$", O_CREAT | O_TEXT | O_APPEND)) == -1)
   {
      perror("Error:");
      return 1;
   }
   write(handle, msg, strlen(msg));
   printf("The file pointer is at byte %ld\n", tell(handle));
   close(handle);
   return 0;
}
```

/* tempnam example */

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

void main(void)
{
  FILE *stream;
  int i;
  char *name;

  for (i = 1; i <= 10; i++) {
    if ((name = tempnam("\\tmp","wow")) == NULL)
      perror("tempnam couldn't create name");
    else {
      printf("Creating %s\n",name);
      if ((stream = fopen(name,"wb")) == NULL)
        perror("Could not open temporary file\n");
      else
        fclose(stream);
    }
    free(name);
  }
  printf("Warning: temp files not deleted.\n");
}
```

/* textmode example */

```
#include <conio.h>

int main(void)
{
   textmode(BW40);
   cprintf("ABC");
   getch();

   textmode(C40);
   cprintf("ABC");
   getch();

   textmode(BW80);
   cprintf("ABC");
   getch();

   textmode(C80);
   cprintf("ABC");
   getch();

   textmode(MONO);
   cprintf("ABC");
   getch();

   return 0;
}
```

/* tmpfile example */

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

int main(void)
{
   FILE *tempfp;

   tempfp = tmpfile();
   if (tempfp)
      printf("Temporary file created\n");
   else
   {
      printf("Unable to create temporary file\n");
      exit(1);
   }

   return 0;
}
```

/* tmpnam example */

```
#include <stdio.h>

int main(void)
{
   char name[13];

   tmpnam(name);
   printf("Temporary name: %s\n", name);
   return 0;
}
```

/* toascii example */

```
#include <stdio.h>
#include <ctype.h>

int main(void)
{
   int number, result;
   number = 511;
   result = toascii(number);
   printf("%d %d\n", number, result);
   return 0;
}
```

/* tzset example */

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

int main(void)
{
   time_t td;

   putenv("TZ=PST8PDT");
   tzset();
   time(&td);
   printf("Current time = %s\n", asctime(localtime(&td)));
   return 0;
}
```

/* ungetc example */

```
#include <stdio.h>
#include <ctype.h>

int main( void )
{
   int i=0;
   char ch;

   puts("Input an integer followed by a char:");

   /* read chars until non digit or EOF */
   while((ch = getchar()) != EOF && isdigit(ch))
      i = 10 * i + ch - 48; /* convert ASCII into int value */

   /* if non digit char was read, push it back into input buffer */
   if (ch != EOF)
      ungetc(ch, stdin);

   printf("i = %d, next char in buffer = %c\n", i, getchar());
   return 0;
}
```

/* ungetch example */

```
#include <stdio.h>
#include <ctype.h>
#include <conio.h>

int main( void )
{
   int i=0;
   char ch;

   puts("Input an integer followed by a char:");

   /* read chars until non digit or EOF */
   while((ch = getche()) != EOF && isdigit(ch))
      i = 10 * i + ch - 48; /* convert ASCII into int value */

   /* if non digit char was read, push it back into input buffer */
   if (ch != EOF)
      ungetch(ch);

   printf("\n\ni = %d, next char in buffer = %c\n", i, getch());
   return 0;
}
```

/* unixtodos example */

```
#include <stdio.h>
#include <dos.h>

char *month[] = {"---", "Jan", "Feb", "Mar", "Apr", "May", "Jun",
                        "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};

#define SECONDS_PER_DAY 86400L  /* the number of seconds in one day */

struct date dt;
struct time tm;

int main(void)
{
   unsigned long val;

/* get today's date and time */
   getdate(&dt);
   gettime(&tm);
   printf("today is %d %s %d\n", dt.da_day, month[dt.da_mon], dt.da_year);

/*convert date and time to unix format (num of seconds since Jan 1, 1970*/
   val = dostounix(&dt, &tm);
/* subtract 42 days worth of seconds */
   val -= (SECONDS_PER_DAY * 42);

/* convert back to dos time and date */
   unixtodos(val, &dt, &tm);
   printf("42 days ago it was %d %s %d\n",
          dt.da_day, month[dt.da_mon], dt.da_year);
   return 0;
}
```

/* unlink example */

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

int main(void)
{
   FILE *fp = fopen("junk.jnk","w");
   int status;

   fprintf(fp,"junk");

   status = access("junk.jnk",0);
   if (status == 0)
      printf("File exists\n");
   else
      printf("File doesn't exist\n");

   fclose(fp);
   unlink("junk.jnk");
   status = access("junk.jnk",0);
   if (status == 0)
      printf("File exists\n");
   else
      printf("File doesn't exist\n");

   return 0;
}
```

/* umask example */

```
#include <io.h>
#include <stdio.h>
#include <sys\stat.h>

#define FILENAME "TEST.$$$"

int main(void)
{
    unsigned oldmask;

    FILE *f;
    struct stat statbuf;

    /* Cause subsequent files to be created as read-only */
    oldmask = umask(S_IWRITE);
    printf("Old mask = 0x%x\n",oldmask);

    /* Create a zero-length file */
    if ((f = fopen(FILENAME,"w+")) == NULL)
    {
        perror("Unable to create output file");
        return (1);
    }
    fclose(f);

    /* Verify that the file is read-only */
    if (stat(FILENAME,&statbuf) != 0)
    {
        perror("Unable to get information about output file");
        return (1);
    }
    if (statbuf.st_mode & S_IWRITE)
        printf("Error! %s is writable!\n",FILENAME);
    else
        printf("Success! %s is not writable.\n",FILENAME);
    return (0);
}
```

/* utime example */

```
/* Copy timestamp from one file to another */

#include <sys\stat.h>
#include <utime.h>
#include <stdio.h>

int main( int argc, char *argv[] )
{
  struct stat src_stat;
  struct utimbuf times;
  if(argc != 3) {
    printf( "Usage: copytime <source file> <dest file>\n" );
    return 1;
  }

  if (stat(argv[1],&src_stat) != 0) {
    perror("Unable to get status of source file");
    return 1;
  }

  times.modtime = times.actime = src_stat.st_mtime;
  if (utime(argv[2],&times) != 0) {
    perror("Unable to set time of destination file");
    return 1;
  }
  return 0;
}
```

/* va_arg example */

```
#include <stdio.h>
#include <stdarg.h>

/* calculate sum of a 0 terminated list */
void sum(char *msg, ...)
{
   int total = 0;
   va_list ap;
   int arg;
   va_start(ap, msg);
   while ((arg = va_arg(ap,int)) != 0) {
      total += arg;
   }
   printf(msg, total);
   va_end(ap);
}

int main(void) {
   sum("The total of 1+2+3+4 is %d\n", 1,2,3,4,0);
   return 0;
}
```

/* wherex and wherey example */

```
#include <conio.h>

int main(void)
{
   clrscr();
   gotoxy(10,10);
   cprintf("Current location is X: %d  Y: %d\r\n", wherex(), wherey());
   getch();

   return 0;
}
```

/* window example */

```
#include <conio.h>

int main(void)
{

   window(10,10,40,11);
   textcolor(BLACK);
   textbackground(WHITE);
   cprintf("This is a test\r\n");

   return 0;
}
```

/* getpsp example */

```
#include <stdio.h>
#include <dos.h>

int main(void)
{
   static char command[128];
   char far *cp;
   int len, i;

   printf("The program segment prefix is: %u\n", getpsp());

/*
_psp is preset to segment of the PSP. Command line is located at offset
  0x81 from start of PSP
*/
   cp = (char *) MK_FP(_psp, 0x80);
   len = *cp;

   for (i = 0; i < len; i++)
      command[i] = cp[i+1];

   printf("Command line: %s\n", command);

   return 0;
}
```

/* stackavail example */

```
#include <malloc.h>
#include <stdio.h>

int main(void)
{
  char *buf;

  printf("\nThe stack: %u\tstack pointer: %u", stackavail(), _SP);
  buf = (char *) alloca(100 * sizeof(char));
  printf("\nNow, the stack: %u\tstack pointer: %u", stackavail(), _SP);
  return 0;
}

/* **********************************************

     Program output

     The stack: 64046       stack pointer: 65524
     Now, the stack: 63946  stack pointer: 65424

   ********************************************** */
```

/* set_new_handler example */

```
#include <iostream.h>
#include <new.h>
#include <stdlib.h>

void mem_warn() {
  cerr << "\nCan't allocate!";
  exit(1);
  }

void main(void) {
  set_new_handler(mem_warn);

  char *ptr = new char[100];
  cout << "\nFirst allocation: ptr = " << hex << long(ptr);
  ptr = new char[64000U];
  cout << "\nFinal allocation: ptr = " << hex << long(ptr);
  set_new_handler(0);  // Reset to default.
}
```

/* isalpha example */

```
#include <stdio.h>
#include <ctype.h>

int main(void)
{
  char c = 'C';

  if (isalpha(c))
    printf("%c is alphabetical\n",c);
  else printf("%c is not alphabetical\n",c);

  return 0;
}
```

/* isalnum example */

```
#include <stdio.h>
#include <ctype.h>

int main(void)
{
  char c = 'C';

  if (isalnum(c))
    printf("%c is alphanumeric\n",c);
  else printf("%c is not alphanumeric\n",c);

  return 0;
}
```

/* isascii example */

```
#include <stdio.h>
#include <ctype.h>
#include <stdio.h>
int main(void)
{
  char c = 'C';

  if (isascii(c))
    printf("%c is ascii\n",c);
  else printf("%c is not ascii\n",c);
  return 0;
}
```

/* iscntrl example */

```
#include <stdio.h>
#include <ctype.h>

int main(void)
{
  char c = 'C';
  if (iscntrl(c))
    printf("%c is a control character\n",c);
  else printf("%c is not a control character\n",c);

  return 0;
}
```

/* isdigit example */

```
#include <stdio.h>
#include <ctype.h>

int main(void)
{
  char c = 'C';

  if (isdigit(c))
    printf("%c is a digit\n",c);
  else printf("%c is not a digit\n",c);

  return 0;
}
```

/* isgraph example */

```
#include <stdio.h>
#include <ctype.h>

int main(void)
{
  char c = 'C';

  if (isgraph(c))
    printf("%c is a graphic character\n",c);
  else printf("%c is not a graphic character\n",c);

  return 0;
}
```

/* islower example */

```
#include <stdio.h>
#include <ctype.h>

int main(void)
{
  char c = 'C';

  if (islower(c))
    printf("%c is a lowercase character\n",c);
  else printf("%c is not a lowercase character\n",c);

  return 0;
}
```

/* isprint example */

```
#include <stdio.h>
#include <ctype.h>

int main(void)
{
  char c = 'C';

  if (isprint(c))
    printf("%c is a printable character\n",c);
  else printf("%c is not a printable character\n",c);

  return 0;
}
```

/* ispunct example */

```
#include <stdio.h>
#include <ctype.h>

int main(void)
{
  char c = 'C';

  if (ispunct(c))
    printf("%c is a punctuation character\n",c);
  else printf("%c is not a punctuation character\n",c);

  return 0;
}
```

/* isspace example */

```
#include <stdio.h>
#include <ctype.h>

int main(void)
{
  char c = 'C';

  if (isspace(c))
    printf("%c is white space\n",c);
  else printf("%c is not white space\n",c);

  return 0;
}
```

/* isupper example */

```
#include <stdio.h>
#include <ctype.h>

int main(void)
{
  char c = 'C';

  if (isupper(c))
    printf("%c is an uppercase character\n",c);
  else printf("%c is not an uppercase character\n",c);

  return 0;
}
```

/* isxdigit example */

```
#include <stdio.h>
#include <ctype.h>

int main(void)
{
  char c = 'C';

  if (isxdigit(c))
    printf("%c is a hexidecimal digit\n",c);
  else printf("%c is not a hexidecimal digit\n",c);

  return 0;
}
```

/* mblen example */

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

void main (void)
{
  int i ;
  char *mulbc = (char *)malloc( sizeof( char) );
  wchar_t widec = L'a';
  printf (" convert a wide character to multibyte character:\n" );
  i = wctomb (mulbc, widec);
  printf( "\tCharacters converted: %u\n", i);
  printf( "\tMultibyte character: %x\n\n", mulbc);

  printf( " Find length--in byte-- of multibyte character:\n");
  i = mblen( mulbc, MB_CUR_MAX);
  printf("\tLenght--in bytes--if multiple character: %u\n",i);
  printf("\tWide character: %x\n\n", mulbc);


  printf( " Attempt to find length of a Wide character Null:\n");
  widec = L'\0';
  wctomb(mulbc, widec);
  i = mblen( mulbc, MB_CUR_MAX);
  printf("\tLenght--in bytes--if multiple character: %u\n",i);
  printf("\tWide character: %x\n\n", mulbc);

}
```

/* mbstowcs example */

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

void main(void)
{
  int x;
  char    *mbst = (char *)malloc(MB_CUR_MAX);
  wchar_t *pwst = L"Hi";
  wchar_t *pwc      = (wchar_t *)malloc(sizeof( wchar_t));

  printf ("Convert to multibyte string:\n");
  x = wcstombs (mbst, pwst, MB_CUR_MAX);
  printf ("\tCharacters converted %u\n",x);
  printf ("\tHEx value of first");
  printf (" multibyte character: %#.4x\n\n", mbst);

  printf ("Convert back to wide character string:\n");
  x = mbstowcs(pwc, mbst, MB_CUR_MAX);
  printf( "\tCharacters converted: %u\n",x);
  printf( "\tHex value of first");
  printf( "wide character: %#.4x\n\n", pwc);
}
```

/* mbtowc example */

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

void main(void)
{
  int     x;
  char     *mbchar    = (char *)calloc(1, sizeof( char));
  wchar_t  wchar      = L'a';
  wchar_t   *pwcnull = NULL;
  wchar_t  *pwchar    = (wchar_t *)calloc(1,  sizeof( wchar_t ));

  printf ("Convert a wide character to multibyte character:\n");
  x = wctomb( mbchar, wchar);
  printf( "\tCharacters converted: %u\n", x);
  printf( "\tMultibyte character: %x\n\n", mbchar);

  printf ("Convert multibyte character back to a wide character:\n");
  x = mbtowc( pwchar, mbchar, MB_CUR_MAX );
  printf( "\tBytes converted: %u\n", x);
  printf( "\tWide character: %x\n\n", pwchar);

  printf ("Atempt to convert when target is NULL\n" );
  printf (" returns the length of the multibyte character:\n" );
  x = mbtowc (pwcnull, mbchar, MB_CUR_MAX );
  printf ( "\tlength of multibyte character:%u\n\n", x );

  printf ("Attempt to convert a NULL pointer to a" );
  printf (" wide character:\n" );
  mbchar = NULL;
  x = mbtowc (pwchar, mbchar, MB_CUR_MAX);
  printf( "\tBytes converted: %u\n", x );
  }
```

/* MK_FP example */

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <dos.h>
#include <malloc.h>

main()
{

  char *str = "hello\n";
  char far *farstr;

  printf ("the address pointed to by str is %04X:%04X\n",
    FP_SEG(str), FP_OFF(str));
  farstr = (char far *)MK_FP( FP_SEG(str),  FP_OFF(str));

  printf ("the string pointed by far pointer is %s\n", farstr);
  return 0;
}
```

/* _msize example */

```
/* _msize works as a 32-bit command, not as a 16-bit command */
#include <malloc.h>        /* malloc() _msize() */
#include <stdio.h>            /* printf() */

int main( )
{
  int size;
  int *buffer;

  buffer = malloc(100 * sizeof(int));
  size = _msize(buffer);
  printf("Allocated %d bytes for 100 integers\n", size);

  return(0);
}
```

/* offsetof example */

```
/*
This program uses the offsetof command to show the effect
of changing alignment boundaries within a structure.
It produces this output:

  In STRUCT1, two_bytes begins at byte 1.
  In STRUCT2, two_bytes begins at byte 2.

By default, the 16-bit compiler aligns structure members
at 1-byte boundaries.  With the -a2 flag set, the compiler
aligns fields on even boundaries.

The CPU often processes structure elements more quickly
when they align on even boundaries.
*/

#include <stddef.h>       // offsetof()
#include <stdio.h>       // printf()

#pragma option -a1       // align on bytes (default)

typedef struct {
  char one_byte;
  int two_bytes;
} STRUCT1;


#pragma option -a2       // align on even bytes

typedef struct {
  char one_byte;
  int two_bytes;
} STRUCT2;

#pragma option -a.       // restore command-line option


void main()
{
  printf( "In STRUCT1, two_bytes begins at byte %d.\n",
    offsetof(STRUCT1, two_bytes) );

  printf( "In STRUCT2, two_bytes begins at byte %d.\n",
    offsetof(STRUCT2, two_bytes) );
}
```

/* _pipe example */

```
/* _pipe example */
#include <windows.h>   //for SECURITY_ATTRIBUTES
#include <stdlib.h>
#include <stdio.h>
#include <io.h>
#include <fcntl.h>
#include <process.h>

void main(int argc, char *argv[])
{
    int retcode, stat, pid;
    char asc_handle[10];
    SECURITY_ATTRIBUTES sa;
    HANDLE s_hFileMap;

    if (argc > 1)   /* this is the child */
    {
        /* Get the read pipe handle from command line,
         * and set the handle to binary.
         */
        printf("Child: The handle passed as 2nd argument is: %s\n",
  argv[1]);
        s_hFileMap = (HANDLE) atoi(argv[1]);

        LPVOID lpView = MapViewOfFile(s_hFileMap,
                              FILE_MAP_READ|FILE_MAP_WRITE,
                              0,0,0);
        if (lpView == NULL)
        {
            perror("Child: unable to read pipe");
            retcode = 255;
        }

        printf("Child: returning %s to parent\n",lpView);

        retcode = atoi((char*) lpView);
        UnmapViewOfFile(lpView);
        CloseHandle(s_hFileMap);       //Child is responsible for this
        exit(retcode);
    }
    else            /* this is the parent */
    {
        //Here we set up the security attributes of the file mapping object
        //so that we can inherit it from the child process.  Alternatively
        //and more cheaply, we could use just the name of the mapping
  object
        //once inside the child process and call OpenFileMapping with that
        //name.
        sa.nLength = sizeof(sa);
        sa.lpSecurityDescriptor = NULL;
        sa.bInheritHandle = TRUE;

        s_hFileMap = CreateFileMapping((HANDLE) 0xFFFFFFFF,//in memory
                               &sa,                //security
  attrib
```

```
                                            PAGE_READWRITE,
                                            0,                  //min. size
                                            256,                //size
                                            NULL);              //give mapping
  object no name

        LPVOID lpView = MapViewOfFile(s_hFileMap,
                                      FILE_MAP_READ|FILE_MAP_WRITE,
                                      0,0,0);
        if (lpView == NULL)
        {
            perror("Parent: unable to create file mapping");
            exit(1);
        }

        sprintf(asc_handle,"%d",s_hFileMap);
        retcode = 10;
        if (sprintf((char*)lpView,"%d",retcode) == EOF)
        {
            perror("Parent: unable to write to pipe");
            exit(1);
        }

        /* Call ourself with read handle as argument.
         */
        if ((pid = spawnl(P_NOWAIT, argv[0], argv[0],
                          asc_handle, NULL)) == -1)
            perror("Parent: spawnl failed");
        else
        {
            printf("Parent: spawned child process %d\n",pid);
            if (wait(&stat) != pid)
                perror("Parent: wait failure");
            else
            {
                if ((stat & 0xff) == 0)
                    printf("Parent: child returned %d\n", stat >> 8);
                else
                    printf("Parent: child terminated abnormally\n");

            }
        }
        UnmapViewOfFile(lpView);
        CloseHandle(s_hFileMap);
        exit(0);
    }
}
```

/* send example */

```
/*
 There are two short programs here.  SEND spawns a child
 process, RECEIVE.  Each process holds one end of a
 pipe.  The parent transmits its command-line argument
 to the child, which prints the string and exits.

 IMPORTANT:  The parent process must be linked with
 the \32bit\fileinfo.obj file.  The code in fileinfo
 enables a parent to share handles with a child.
 Without this extra information, the child cannot use
 the handle it receives.
*/

/* SEND */

#include <fcntl.h>              // _pipe()
#include <io.h>              // write()
#include <process.h>            // spawnl() cwait()
#include <stdio.h>            // puts() perror()
#include <stdlib.h>            // itoa()
#include <string.h>            // strlen()

#define DECIMAL_RADIX 10        // for atoi()
enum PIPE_HANDLES { IN, OUT };  // to index the array of handles

int main(int argc, char *argv[])
{
  int handles[2];            // in- and
//outbound pipe handles
  char handleStr[10];         // a handle
//stored as a string
  int pid;
  // system's ID for child process

  if (argc <= 1)
  {
    puts("No message to send.");
    return(1);
  }

  if (_pipe(handles, 256, O_TEXT) != 0)
  {
    perror("Cannot create the pipe");
    return(1);
  }

  // store handle as a string for passing on the command line
  itoa(handles[IN], handleStr, DECIMAL_RADIX);

  // create the child process, passing it the inbound pipe handle
     spawnl(P_NOWAIT, "receive.exe", "receive.exe", handleStr, NULL);

  // transmit the message
  write(handles[OUT], argv[1], strlen(argv[1])+1);
```

```
  // when done with the pipe, close both handles
  close(handles[IN]);
  close(handles[OUT]);

  // wait for the child to finish
  wait(NULL);
  return(0);
}
```

/* _setcursortype example */

```
#include <conio.h>

int main( )
{
  // tell the user what to do
  clrscr();
  cputs("Press any key three times.\n\r");
  cputs("Each time the cursor will change shape.\n\r");

  gotoxy(1,5);           // show a solid cursor
  cputs("Now the cursor is solid.\n\r");
  _setcursortype(_SOLIDCURSOR);

  while(!kbhit()) {};        // wait to proceed
  getch();


  gotoxy(1,5);           // remove the cursor
  cputs("Now the cursor is gone.");
  clreol();
  gotoxy(1,6);
  _setcursortype(_NOCURSOR);

  while(!kbhit()) {};        // wait to proceed
  getch();


  gotoxy(1,5);                 // show a normal cursor
  cputs("Now the cursor is normal.");
  clreol();
  gotoxy(1,6);
  _setcursortype(_NORMALCURSOR);

  while(!kbhit()) {};        // wait to proceed
  getch();

  clrscr();
  return(0);
}
```

/* _dos_commit example */

```
#include <dos.h>
#include <errno.h>
#include <conio.h>

void main(void)
{
  char save[]  = "to disk.",
  prompt[]     = " File exist,overwrite?[y/n]",
  err[]       = "Error occured. ",
  newline[]   = "\n\r";

int handle, ch;
unsigned count;

  /* Open file and create and overwrite it */

  if (_dos_createnew( "DUMMY.FIL",_A_NORMAL, &handle) !=0)
  {
    if (errno == EEXIST)
    {
    /* Use _dos_write to display prompts*/
    _dos_write (1, prompt, sizeof( prompt ) -1, &count );
    ch = bdos( 1, 0, 0) & 0x00ff;
    if ( (ch == 'y') || (ch == 'Y') )
      _dos_creat( "DUMMY.FIL", _A_NORMAL, &handle);
    _dos_write( 1,newline, sizeof( newline) -1, &count);
    }
  }

  /* Write to file; output passes through operating system's buffer*/

  if ( _dos_write( handle, save, sizeof( save ),&count) != 0 )

  {
  _dos_write( 1, err, sizeof( err) - 1, &count );
  _dos_write( 1, newline, sizeof( newline ) -1, &count );
  }

  /* Write directly to file with no intermediate buffering */
  if ( _dos_commit( handle ) != 0 )
  {
  _dos_write( 1, err, sizeof(err) -1, &count);
  _dos_write( 1, newline, sizeof( newline ) - 1 , &count);
  }

  /* Close file */
  if (_dos_close( handle ) != 0)
  {
  _dos_write( 1, err, sizeof(err) -1, &count );
  _dos_write( 1, newline, sizeof(newline) -1, &count );
  }

}
```

/* _expand example */

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>

void main(void)
{
  char *bufchar;

  printf( "Allocate a 512 element buffer\n" );
  if( (bufchar = (char *) calloc(512, sizeof( char ) )) == NULL)
    exit( 1 );
  printf( "Allocated %d bytes at %Fp\n",
    _msize ( bufchar ), (void __far *)bufchar );

  if ((bufchar = (char *) _expand (bufchar, 1024)) == NULL)
    printf ("can not expand");
  else
    printf (" Expanded block to %d bytes at %Fp\n",
      _msize( bufchar ) , (void __far *)bufchar );
/* free memory  */
  free( bufchar );
  exit (0);
}
```

/* _get_osfhandle and _open_osfhandle example */

```
#include <windows.h>
#include <fcntl.h>
#include <stdio.h>
#include <io.h>
#ifndef __FLAT__
#error This Example must be compiled using 32 bit compiler
#endif

//Example for _get_osfhandle() and _open_osfhandle()

BOOL InitApplication(HINSTANCE hInstance);
HWND InitInstance(HINSTANCE hInstance, int nCmdShow);
LRESULT FAR PASCAL _export MainWndProc(HWND hWnd, UINT message,
            WPARAM wParam, LPARAM lParam);
Example_get_osfhandle(HWND hWnd);

#pragma argsused
int PASCAL WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
        LPSTR lpCmdLine, int nCmdShow)

{
MSG msg;      // message

  if (!InitApplication(hInstance))  // Initialize shared things
   return (FALSE); // Exits if unable to initialize

  /* Perform initializations that apply to a specific instance */

  if (!(InitInstance(hInstance, nCmdShow)))
   return (FALSE);


  /* Acquire and dispatch messages until a WM_QUIT message is received. */

  while (GetMessage(&msg, // message structure
   NULL, // handle of window receiving the message
   NULL, // lowest message to examine
   NULL))  // highest message to examine
  {
  TranslateMessage(&msg); // Translates virtual key codes
  DispatchMessage(&msg);  // Dispatches message to window
  }

  return (msg.wParam);  // Returns the value from PostQuitMessage


}
BOOL InitApplication(HINSTANCE hInstance)
{
  WNDCLASS  wc;

  // Fill in window class structure with parameters that describe the
  // main window.

  wc.style = CS_HREDRAW | CS_VREDRAW; // Class style(s).
```

```
  wc.lpfnWndProc = (long (FAR PASCAL*)(void *,unsigned int,unsigned int,
  long ))MainWndProc; // Function to retrieve messages for
                // windows of this class.
  wc.cbClsExtra = 0;  // No per-class extra data.
  wc.cbWndExtra = 0;  // No per-window extra data.
  wc.hInstance = hInstance; // Application that owns the class.
  wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
  wc.hCursor = LoadCursor(NULL, IDC_ARROW);
  wc.hbrBackground = GetStockObject(WHITE_BRUSH);
  wc.lpszMenuName = NULL;  // Name of menu resource in .RC file.
  wc.lpszClassName = "Example"; // Name used in call to CreateWindow.

  /* Register the window class and return success/failure code. */

  return (RegisterClass(&wc));

}
HWND InitInstance(HINSTANCE hInstance, int nCmdShow)
{
  HWND  hWnd; // Main window handle.

  /* Create a main window for this application instance.  */

  hWnd = CreateWindow(
   "Example",                 // See RegisterClass() call.
   "Example _get_osfhandle _open_osfhandle (32 bit)",  // Text for window
  title bar.
   WS_OVERLAPPEDWINDOW,        // Window style.
   CW_USEDEFAULT,          // Default horizontal position.
   CW_USEDEFAULT,           // Default vertical position.
   CW_USEDEFAULT,             // Default width.
   CW_USEDEFAULT,         // Default height.
   NULL,                  // Overlapped windows have no parent.
   NULL,                  // Use the window class menu.
   hInstance,              // This instance owns this window.
   NULL                   // Pointer not needed.
  );

  /* If window could not be created, return "failure" */

  if (!hWnd)
   return (FALSE);

  /* Make the window visible; update its client area; and return "success"
  */

  ShowWindow(hWnd, nCmdShow); // Show the window
  UpdateWindow(hWnd);     // Sends WM_PAINT message
  return (hWnd);        // Returns the value from PostQuitMessage

}
#pragma argsused
LRESULT FAR PASCAL _export MainWndProc(HWND hWnd, UINT message,
            WPARAM wParam, LPARAM lParam)
{

  switch (message)
```

```
  {
    case WM_CREATE:
    {
      Example_get_osfhandle(hWnd);
      return NULL;
    }
    case WM_QUIT:
    case WM_DESTROY:  // message: window being destroyed
      PostQuitMessage(0);
    break;

    default:      // Passes it on if unproccessed
      return (DefWindowProc(hWnd, message, wParam, lParam));
  }
}

Example_get_osfhandle(HWND hWnd)
{
    long osfHandle;
    char str[128];
    int fHandle = open("file1.c", O_CREAT|O_TEXT);
    if(fHandle != -1)
    {
      osfHandle = _get_osfhandle(fHandle);
      sprintf(str, "file handle = %lx OS file handle = %lx", fHandle,
  osfHandle);
      MessageBox(hWnd,str,"_get_osfhandle",MB_OK|MB_ICONINFORMATION);
      close(fHandle);

      fHandle = _open_osfhandle(osfHandle, O_TEXT );
      sprintf(str, "file handle = %lx OS file handle = %lx", fHandle,
  osfHandle);
      MessageBox(hWnd,str,"_open_osfhandle",MB_OK|MB_ICONINFORMATION);
         close(fHandle);

    }
    else
      MessageBox(hWnd,"File Open Error","WARNING",MB_OK|MB_ICONSTOP);
return 0;
}
```

/* _heapset example */

```
#include <windowsx.h>
#include <malloc.h>
#include <stdio.h>
#include <stdlib.h>

#ifndef __FLAT__
#error This Example must be compiled using 32 bit compiler
#endif
BOOL InitApplication(HINSTANCE hInstance);
HWND InitInstance(HINSTANCE hInstance, int nCmdShow);
LRESULT FAR PASCAL _export MainWndProc(HWND hWnd, UINT message,
            WPARAM wParam, LPARAM lParam);
void ExampleHeapSet(HWND hWnd);
#pragma argsused
int PASCAL WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
        LPSTR lpCmdLine, int nCmdShow)

{
MSG msg;      // message

  if (!InitApplication(hInstance))  // Initialize shared things
   return (FALSE); // Exits if unable to initialize

  /* Perform initializations that apply to a specific instance */

  if (!(InitInstance(hInstance, nCmdShow)))
   return (FALSE);


  /* Acquire and dispatch messages until a WM_QUIT message is received. */

  while (GetMessage(&msg, // message structure
   NULL, // handle of window receiving the message
   NULL, // lowest message to examine
   NULL))  // highest message to examine
  {
  TranslateMessage(&msg); // Translates virtual key codes
  DispatchMessage(&msg);  // Dispatches message to window
  }

  return (msg.wParam);  // Returns the value from PostQuitMessage


}
BOOL InitApplication(HINSTANCE hInstance)
{
  WNDCLASS  wc;

  // Fill in window class structure with parameters that describe the
  // main window.

  wc.style = CS_HREDRAW | CS_VREDRAW; // Class style(s).
  wc.lpfnWndProc = (long (FAR PASCAL*)(void *,unsigned int,unsigned int,
  long ))MainWndProc; // Function to retrieve messages for
              // windows of this class.
```

```
  wc.cbClsExtra = 0;  // No per-class extra data.
  wc.cbWndExtra = 0;  // No per-window extra data.
  wc.hInstance = hInstance; // Application that owns the class.
  wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
  wc.hCursor = LoadCursor(NULL, IDC_ARROW);
  wc.hbrBackground = GetStockObject(WHITE_BRUSH);
  wc.lpszMenuName = NULL;  // Name of menu resource in .RC file.
  wc.lpszClassName = "Example"; // Name used in call to CreateWindow.

  /* Register the window class and return success/failure code. */

  return (RegisterClass(&wc));

}
HWND InitInstance(HINSTANCE hInstance, int nCmdShow)
{
  HWND  hWnd; // Main window handle.

  /* Create a main window for this application instance.  */

  hWnd = CreateWindow(
   "Example",                  // See RegisterClass() call.
   "Example _heapset 32 bit only",  // Text for window title bar.
   WS_OVERLAPPEDWINDOW,        // Window style.
   CW_USEDEFAULT,          // Default horizontal position.
   CW_USEDEFAULT,           // Default vertical position.
   CW_USEDEFAULT,             // Default width.
   CW_USEDEFAULT,         // Default height.
   NULL,                  // Overlapped windows have no parent.
   NULL,                  // Use the window class menu.
   hInstance,              // This instance owns this window.
   NULL                   // Pointer not needed.
  );

  /* If window could not be created, return "failure" */

  if (!hWnd)
   return (FALSE);

  /* Make the window visible; update its client area; and return "success"
  */

  ShowWindow(hWnd, nCmdShow); // Show the window
  UpdateWindow(hWnd);     // Sends WM_PAINT message
  return (hWnd);        // Returns the value from PostQuitMessage

}

void ExampleHeapSet(HWND hWnd)
{
  int hsts;
  char *buffer;

  if ( (buffer = (char *)malloc( 1 )) == NULL )
    exit(0);
  hsts = _heapset( 'Z' );
```

```
  switch (hsts)
  {
    case _HEAPOK:
      MessageBox(hWnd,"Heap is OK","Heap",MB_OK|MB_ICONINFORMATION);
      break;
    case _HEAPEMPTY:
      MessageBox(hWnd,"Heap is empty","Heap",MB_OK|MB_ICONINFORMATION);
      break;
    case _HEAPBADNODE:
      MessageBox(hWnd,"Bad node in heap","Heap",MB_OK|MB_ICONINFORMATION);
      break;
    default:
      break;

  }

  free (buffer);
}
#pragma argsused
LRESULT FAR PASCAL _export MainWndProc(HWND hWnd, UINT message,
            WPARAM wParam, LPARAM lParam)
{

  switch (message) {
    case WM_CREATE:
    {
      //Example _heapset
      ExampleHeapSet(hWnd);
      return NULL;
    }
    case WM_QUIT:
    case WM_DESTROY:  // message: window being destroyed
      PostQuitMessage(0);
    break;

    default:      // Passes it on if unproccessed
      return (DefWindowProc(hWnd, message, wParam, lParam));
  }
}
```

/* _searchstr example */

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

char buf[_MAX_PATH];

int main(void)
{
  /* look for TLINK.EXE */
  _searchstr("TLINK.EXE", "PATH", buf);
  if (buf[0] == '\0')
  printf ("TLINK.EXE not found\n");
  else
  printf ("TLINK.EXE found in %s\n", buf);

  return 0;
}
```

/* _popen and _pclose example */

```
/* this program initiates a child process to run the dir command
   and pipes the directory listing from the child to the parent.
*/

#include <stdio.h>     // popen() pclose() feof() fgets() puts()
#include <string.h>    // strlen()

int main( )
{
  FILE* handle;         // handle to one end of pipe
  char message[256];    // buffer for text passed through pipe
  int status;           // function return value

  // open a pipe to receive text from a process running "DIR"
  handle = _popen("dir /b", "rt");
  if (handle == NULL)
  {
    perror("_popen error");
  }

  // read and display input received from the child process
  while (fgets(message, sizeof(message), handle))
  {
    fprintf(stdout, message);
  }

  // close the pipe and check the return status
    status = _pclose(handle);
  if (status == -1)
  {
    perror("_pclose error");
  }

  return(0);
}
```

/* wctomb example */

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

void main(void)
{
  int x;
  wchar_t wc = L'a';
  char *pmbNULL = NULL;
  char *pmb = (char *)malloc(sizeof( char ));

  printf (" Convert a wchar_t array into a multibyte string:\n");
  x = wctomb( pmb, wc);
  printf ("Character converted: %u\n", x);
  printf ("Multibyte string: %1s\n\n",pmb);

  printf (" Convert when target is NULL\n");
  x = wctomb( pmbNULL, wc);
  printf ("Character converted: %u\n",x);
  printf ("Multibyte stri ng: %1s\n\n",pmbNULL);

}
```

/* wcstombs example */

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

void main(void)
{
  int x;
  char *pbuf = (char*)malloc( MB_CUR_MAX);
  wchar_t *pwcsEOL = L'\0';
  char *pwchi= L"Hi there!";

  printf (" Convert entire wchar string into a multibyte string:\n");
  x = wcstombs( pbuf, pwchi,MB_CUR_MAX);
  printf ("Character converted: %u\n", x);
  printf ("Multibyte string character: %1s\n\n",pbuf);

  printf (" Convert when target is NULL\n");
  x = wcstombs( pbuf, pwcsEOL, MB_CUR_MAX);
  printf ("Character converted: %u\n",x);
  printf ("Multibyte string: %1s\n\n",pbuf);

}
```

Run-Time Support

See also

These topics provide a detailed description of the functions and classes that provide run-time support. Any class operators or member functions are listed immediatedly after the class constructor.

Classes

Bad_cast class
Bad_typeid class
typeinfo class
xalloc class
xmsg class

Functions

set_new_handler
set_terminate
set_unexpected
terminate
unexpected

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+		+	

Bad_cast class

See also Portability Example

Header File

typeinfo.h

Description

When dynamic_cast fails to make a cast to reference, the expression can throw *Bad_cast*. Note that when *dynamic_cast* fails to make a cast to pointer type, the result is the null pointer.

Bad_typeid class

See also Portability Example

Header File

typeinfo.h

Description

When the operand of typeid is a dereferenced null pointer, the **typeid** operator can throw *Bad_typeid*.

Portability

DOS	UNIX	Win 16	Win 32	ANSI C	ANSI C++	OS/2
+		+	+		+	+

set_new_handler

See Also Portability

Header File
new.h

Syntax
```
typedef void (new * new_handler)();
new_handler set_new_handler(new_handler my_handler);
```

Description

set_new_handler installs the function to be called when the global **operator new** or **operator new[]()** cannot allocate the requested memory. By default the *new* operators throw an xalloc exception if memory cannot be allocated. You can change this default behavior by calling *set_new_handler* to set a new handler. To retain the traditional version of *new*, which does not throw exceptions, you can use *set_new_handler*(0).

If new cannot allocate the requested memory, it calls the handler that was set by a previous call to *set_new_handler*. If there is no handler installed by *set_new_handler*, *new* returns 0. *my_handler* should specify the actions to be taken when new cannot satisfy a request for memory allocation. The new_handler type, defined in new.h, is a function that takes no arguments and returns **void**. A *new_handler* can throw an *xalloc* exception.

The user-defined *my_handler* should do one of the following:

- return after freeing memory
- throw an *xalloc* exception or an exception derived from *xalloc*
- call abort or exit functions

If *my_handler* returns, then *new* will again attempt to satisfy the request.

Ideally, *my_handler* would free up memory and return. *new* would then be able to satisfy the request and the program would continue. However, if *my_handler* cannot provide memory for *new*, *my_handler* must throw an exception or terminate the program. Otherwise, an infinite loop will be created.

Preferably, you should overload **operator new()** and **operator new[]()** to take appropriate actions for your applications.

Return Value

set_new_handler returns the old handler, if one has been registered.

The user-defined argument function, *my_handler*, should not return a value.

set_terminate

See also Portability

Header File

except.h

Syntax

```
typedef void (*terminate_function)();
terminate_function set_terminate(terminate_function t_func);
```

Description

set_terminate lets you install a function that defines the program's termination behavior when a handler for the exception cannot be found. The actions are defined in *t_func*, which is declared to be a function of type *terminate_function*. A *terminate_function* type, defined in except.h, is a function that takes no arguments, and returns void.

By default, an exception for which no handler can be found results in the program calling the terminate function. This will normally result in a call to abort. The program then ends with the message `Abnormal program termination`. If you want some function other than *abort* to be called by the *terminate* function, you should define your own *t_func* function. Your *t_func* function is installed by *set_terminate* as the termination function. The installation of *t_func* lets you implement any actions that are not taken by *abort*.

Return Value

The previous function given to *set_terminate* will be the return value.

The definition of *t_func* must terminate the program. Such a user-defined function must not return to its caller, the *terminate* function. An attempt to return to the caller results in undefined program behavior. It is also an error for *t_func* to throw an exception.

set_unexpected

See also Portability

Header File

except.h

Syntax

```
typedef void ( * unexpected_function )();
unexpected_function set_unexpected(unexpected_function unexpected_func);
```

Description

set_unexpected lets you install a function that defines the program's behavior when a function throws an exception not listed in its exception specification. The actions are defined in *unexpected_func*, which is declared to be a function of type *unexpected_function*. An *unexpected_function* type, defined in except.h, is a function that takes no arguments, and returns void.

By default, an unexpected exception causes unexpected to be called. If   is defined, it is subsequently called by *unexpected*. Program control is then turned over to the user-defined *unexpected_func*. Otherwise, terminate is called.

Return Value

The previous function given to *set_unexpected* will be the return value.

The definition of *unexpected_func* must not return to its caller, the *unexpected* function. An attempt to return to the caller results in undefined program behavior.

unexpected_func can also call abort, exit, or *terminate*.

terminate

See also Portability

Header File

except.h

Syntax

```
void terminate();
```

Description

The function *terminate* can be called by unexpected or by the program when a handler for an exception cannot be found. The default action by *terminate* is to call abort. Such a default action causes immediate program termination.

You can modify the way that your program will terminate when an exception is generated that is not listed in the exception specification. If you do not want the program to terminate with a call to *abort*, you can instead define a function to be called. Such a function (called a *terminate_function*) will be called by *terminate* if it is registered with set_terminate.

Return Value

None.

typeinfo class

See also Portability Example

Header File

typeinfo.h

Description

Provides information about a type.

Constructor

Only a private constructor is provided. You cannot create *typeinfo* objects. By declaring your objects to be __rtti types, or by using the -RT compiler switch, the compiler provides your objects with the elements of *typeinfo*. *typeinfo* references are generated by the typeid operator.

Public Member Functions

name

before

Operators

==

!=

typeinfo::name

typeinfo class

Syntax

```
const char* name() const;
```

Description

This function returns a printable string that identifies the type *name* of the operand to typeid. The space for the character string is overwritten on each call.

typeinfo::before

typeinfo class

Syntax

```
int before(const typeinfo&);
```

Description

Use this function to compare the lexical order of types. For example, to compare two types, *T1* and *T2*, use the following syntax:

```
  typeid ( T1 ).before(typeid( T2 ));
```

The *before* function returns 0 or 1.

typeinfo::operator ==

typeinfo class

Syntax

```
int operator==(const typeinfo &) const;
```

Description

Provides comparison of *typeinfos*.

typeinfo::operator !=

typeinfo class

Syntax

```
int operator!=(const typeinfo &) const;
```

Description

Provides comparison of *typeinfos*.

// Example

```
// HOW TO GET RUNTIME TYPE INFORMATION.
#include <iostream.h>
#include <typeinfo.h>

class __rtti Alpha {
   virtual void func() {};  // This makes Alpha a polymorphic class type.
};

class B : public Alpha {};

int main(void) {
   B Binst;           // Instantiate class B
   B *Bptr;           // Declare a B-type pointer
   Bptr = &Binst;     // Initialize the pointer

   // THESE TESTS ARE DONE AT RUNTIME

   if (typeid( *Bptr ) == typeid( B ) )
       // Ask "WHAT IS THE TYPE FOR *Bptr?"
       cout << "Name is " << typeid( *Bptr).name();

   if (typeid( *Bptr ) != typeid( Alpha ) )
       cout << "\nPointer is not an Alpha-type.";

   return 0;
   }
```

// Program Output

```
// Name is B
// Pointer is not an Alpha-type.
```

unexpected

See also Portability

Header File

except.h

Syntax

```
void unexpected();
```

Description

The *unexpected* function is called when a function throws an exception not listed in its exception specification. The program calls *unexpected*, which by default calls any user-defined function registered by set_unexpected. If no function is registered with *set_unexpected*, the *unexpected* function then calls terminate.

Return Value

None, although *unexpected* may throw an exception.

xalloc class

See also Portability

Header File

except.h

Description

Reports an error on allocation request.

Constructor

xalloc::xalloc

Public Member Functions

raise

requested

xalloc::xalloc

xalloc class

Syntax

```
xalloc(const string &msg, size_t size);
```

Description

The *xalloc* class has no default constructor. Every use of *xalloc* must define the message to be reported when a size allocation cannot be fulfilled. The *string* type is defined in cstring.h header file.

xalloc::raise

xalloc class

Syntax

```
void raise() throw(xalloc);
```

Description

Calling *raise* causes an *xalloc* to be thrown. In particular, it throws *this.

xalloc::requested

xalloc class

Syntax

```
size_t requested() const;
```

Description

Returns the number of bytes that were requested for allocation.

xmsg class

See also   Portability

Header File

except.h

Description

Reports a message related to an exception.

Constructor

xmsg::xmsg

Public Member Functions

raise

why

xmsg::xmsg

xmsg class

Syntax

```
xmsg(string msg);
```

Description

There is no default constructor for *xmsg*. Every *xmsg* object must have a string message explicitly defined. The string type is defined in cstring.h header file.

xmsg::raise

xmsg class

Syntax

```
void raise() throw(xmsg);
```

Description

Calling *raise* causes an *xmsg* to be thrown. In particular, it throws *this.

xmsg::why

xmsg class

Syntax

```
const string _FAR & why() const;
```

Description

Reports the string used to construct an *xmsg*. Because every *xmsg* must have its message explicitly defined, every instance should have a unique message.

except.h

The except.h header file contains the declarations and prototypes for exception-handling functions and classes, their data members, and member functions.

Classes

xalloc class

xmsg class

Functions

set_terminate

set_unexpected

terminate

unexpected

typeinfo.h

The typeinfo.h header file contains the declarations and prototypes for the run-time type information classes, their data members, and member functions.

Classes

Bad_cast class

Bad_typeid class

typeinfo class

O_xxxx #defines

Header File
fcntl.h

Description
These #defines are bit definitions for a file-access argument.

These RTL file-open functions use some (not all) of these definitions:

- fdopen
- fopen
- freopen
- _fsopen
- open
- _rtl_open
- sopen

sopen also uses file-sharing symbolic constants in the file-access argument.

Category	
Constant	**Description**
Read/Write flag (Used by _rtl_open, and sopen)	
O_RDONLY	Open for reading only
O_WRONLY	Open for writing only
O_RDWR	Open for reading and writing
Other access flags (Used by open and sopen)	
O_NDELAY	Not used; for UNIX compatibility.
O_APPEND	Append to end of file
	If set, the file pointer is set to the end of the file prior to each write.
O_CREAT	Create and open file
	If the file already exists, has no effect.
	If the file does not exist, the file is created.
O_EXCL	Exclusive open: Used only with O_CREAT.
	If the file already exists, an error is returned.
O_TRUNC	Open with truncation
	If the file already exists, its length is truncated to 0. The file attributes remain unchanged.
Binary-mode/Text-mode flags(Used by fdopen, fopen, freopen, _fsopen, open and sopen)	
O_BINARY	No translation: Explicitly opens the file in binary mode
O_TEXT	CR-LF translation: Explicitly opens the file in text mode
Additional values available under DOS 3.x (Used by _rtl_open)	
O_NOINHERIT	Child processes inherit file
O_DENYALL	Error if opened for read/write
O_DENYWRITE	Error if opened for write
O_DENYREAD	Error if opened for read
O_DENYNONE	Allow concurrent access

Note: Only one of the O_DENYxxx options can be included in a single open. These file-sharing

attributes are in addition to any locking performed on the files.

DO NOT MODIFY these special read-only bits described in DOS documentation!

O_CHANGED	Special DOS read-only bit
O_DEVICE	Special DOS read-only bit

SEEK_xxx

See also

Header File

io.h

stdio.h

Description

#defines that set seek starting points

Constant	Value	File Location
SEEK_SET	0	Seeks from beginning of file
SEEK_CUR	1	Seeks from current position
SEEK_END	2	Seeks from end of file

SH_xxxx

Header File
share.h

Description
File-sharing mode for use with sopen (under DOS 3.0 or later).

Constant	Meaning
SH_COMPAT	Sets compatibility mode:
	Allows other opens with SH_COMPAT. The call will fail if the file has already been opened in any other shared mode.
SH_DENYNONE	Permits read/write access
	Allows other shared opens to the file, but not other SH_COMPAT opens
SH_DENYNO	Permits read/write access (provided for compatibility)
SH_DENYRD	Denies read access. Allows only writes from any other open to the file
SH_DENYRW	Denies read/write access. Only the current handle may have access to the file
SH_DENYWR	Denies write access. Allows only reads from any other open to the file
O_NOINHERIT	The file is not passed to child programs

These file-sharing attributes are in addition to any locking performed on the files.

P_xxxx

Header File
process.h

Description
Modes used by the spawn... functions.

Constant	Meaning
P_WAIT	Child runs separately, parent waits until exit
P_DETACH	Child and parent run concurrently with child process in background mode
P_NOWAIT	Child and parent run concurrently (Not implemented)
P_NOWAITO	Child and parent run concurrently, but the child process is not saved
P_OVERLAY	Child replaces parent so that parent no longer exists

SIG_xxx

See also

Header File

signal.h

Description

Predefined functions for handling signals generated by raise or by external events.

Name	Meaning
SIG_DFL	Terminate the program
SIG_IGN	No action, ignore signal
SIG_ERR	Return error code

SIGxxxx

Header File

signal.h

Description

Signal types used by raise and signal.

Signal	Note	Meaning	Default Action
SIGABRT	(*)	Abnormal termination	= to calling `_exit(3)`
SIGFPE		Bad floating-point operation Arithmetic error caused by division by 0, invalid operation, etc.	= to calling `_exit(1)`
SIGILL	(#)	Illegal operation = to calling `_exit(1)`	
SIGINT		Control-C interrupt	Is to do an INT 23h
SIGSEGV	(#)	Invalid access to storage	= to calling `_exit(1)`
SIGTERM	(*)	Request for program termination	= to calling _exit(1)

(*) Signal types marked with a (*) aren't generated by DOS or Borland C++ during normal operation. However, they can be generated with raise.

(#) Signals marked by (#) can't be generated asynchronously on 8088 or 8086 processors but can be generated on some other processors (see signal for details).

stdaux, stderr, stdin, stdout, and stdprn

Header File
stdio.h

Description
Predefined streams automatically opened when the program is started.

Name	Meaning
stdin	Standard input device
stdout	Standard output device
stderr	Standard error output device
stdaux	Standard auxiliary device
stdprn	Standard printer

S_Ixxxx

Header File
sys\stat.h

Description
Definitions used for file status and directory functions.

Name	Meaning
S_IFMT	File type mask
S_IFDIR	Directory
S_IFIFO	FIFO special
S_IFCHR	Character special
S_IFBLK	Block special
S_IFREG	Regular file
S_IREAD	Owner can read
S_IWRITE	Owner can write
S_IEXEC	Owner can execute

NULL #define

Header File
stddef.h

Description
Null pointer constant that is compatible with any data object pointer. It is not compatible with function pointers. When a pointer is equivalent to NULL it is guaranteed not to point to any data object defined within the program.

Bit Definitions for fnsplit

Header File
dir.h

Description
Bit definitions returned from fnsplit to identify which pieces of a file name were found during the split.

Flag	Component
DIRECTORY	Path includes a directory (and possibly subdirectories)
DRIVE	Path includes a drive specification (see DIR.H)
EXTENSION	Path includes an extension
FILENAME	Path includes a file name
WILDCARDS	Path contains wildcards (* or ?)

MAXxxxx

Header File
dir.h

Description
These symbols define the maximum number of characters in a file specification for fnsplit (including room for a terminating NULL).

Name	Meaning
MAXPATH	Complete file name with path
MAXDRIVE	Disk drive (e.g., "A:")
MAXDIR	File subdirectory specification
MAXFILE	File name without extension
MAXEXT	File extension

_F_xxxx

Header File
stdio.h

Description
File status flags of streams

Name	Meaning
_F_RDWR	Read and write
_F_READ	Read-only file
_F_WRIT	Write-only file
_F_BUF	Malloc'ed buffer data
_F_LBUF	Line-buffered file
_F_ERR	Error indicator
_F_EOF	EOF indicator
_F_BIN	Binary file indicator
_F_IN	Data is incoming
_F_OUT	Data is outgoing
_F_TERM	File is a terminal

FA_xxxx

Header File
dos.h

Description
DOS file attributes

Constant	Description
FA_RDONLY	Read-only attribute
FA_HIDDEN	Hidden file
FA_SYSTEM	System file
FA_LABEL	Volume label
FA_DIREC	Directory
FA_ARCH	Archive

For more detailed information about these attributes, refer to your DOS reference manuals.

EXIT_xxxx

Header File
stdlib.h

Description
Constants defining exit conditions for calls to the exit function.

Name	Meaning
EXIT_SUCCESS	Normal program termination
EXIT_FAILURE	Abnormal program termination

_IOxxx

See also

Header File

stdio.h

Description

Constants for defining buffering style to be used with a file.

Name	Meaning
_IOFBF	The file is fully buffered. When a buffer is empty, the next input operation will attempt to fill the entire buffer. On output, the buffer will be completely filled before any data is written to the file.
_IOLBF	The file is line buffered. When a buffer is empty, the next input operation will still attempt to fill the entire buffer. On output, however, the buffer will be flushed whenever a newline character is written to the file.
_IONBF	The file is unbuffered. The buf and size parameters are ignored. Each input operation will read directly from the file, and each output operation will immediately write the data to the file.

BUFSIZ

Header File
stdio.h

Description
Default buffer size used by setbuf function.

EOF

Header File
stdio.h

Description
A constant indicating that end-of-file has been reached on a file.

_IS_xxx

Header File
ctype.h

Description
Bit settings in the _ctype[] used by the *is*... character macros.

Name	Meaning
_IS_SP	Is space
_IS_DIG	Is digit
_IS_UPP	Is uppercase
_IS_LOW	Is lowercase
_IS_HEX	[A-F] or [a-f]
_IS_CTL	Control
_IS_PUN	Punctuation

CHAR_xxx

Header File

limits.h

Description

Name	Meaning
CHAR_BIT	Type char, number of bits
CHAR_MAX	Type char, minimum value
CHAR_MIN	Type char, maximum value

These values are independent of whether type char is defined as signed or unsigned by default.

SCHAR_xxx

Header File
limits.h

Description

Name	Meaning
SCHAR_MAX	Type char, maximum value
SCHAR_MIN	Type char, minimum value

Uxxxx_MAX

Header File
limits.h

Description

Name	Maximum value for type xxx
UCHAR_MAX	unsigned char
USHRT_MAX	unsigned short
UINT_MAX	unsigned integer
ULONG_MAX	unsigned long

SHRT_xxx

Header File
limits.h

Description

Name	Meaning
SHRT_MAX	Type short, maximum value
SHRT_MIN	Type short, minimum value

INT_xxx

Header File
limits.h

Description
Maximum and minimum value for type int.

Name	Meaning
INT_MAX	Type int, maximum value
INT_MIN	Type int, minimum value

LONG_xxx

Header File
limits.h

Description
Maximum and minimum value for type long.

Name	Meaning
LONG_MAX	Type long, maximum value
LONG_MIN	Type long, minimum value

CW_DEFAULT

Header File
float.h

Description
Default control word for 8087/80287 math coprocessor.

EDOM, ERANGE, HUGE_VAL

Header File

errno.h

math.h

Description

Name	Meaning
EDOM	Error code for math domain error
ERANGE	Error code for result out of range
HUGE_VAL	Overflow value for math functions

NDEBUG

Header File
assert.h

Description
NDEBUG means "Use #define to treat assert as a macro or a true function".

Can be defined in a user program. If defined, assert is a true function; otherwise assert is a macro.

NFDS

Header File
dos.h

Description
Maximum number of file descriptors.

MAXxxxx

Header File
values.h

Description
Maximum values for integer data types

Name	Meaning
MAXSHORT	Largest short
MAXINT	Largest int
MAXLONG	Largest long

M_E, M_LOGxxx, M_LNxx

Header File
math.h

Description
The constant values for logarithm functions.

Name	Meaning
M_E	The value of e
M_LOG2E	The value of log(e)
M_LOG10E	The value of log10(e)
M_LN2	The value of ln(2)
M_LN10	The value of ln(10)

PI constants

Header File
math.h

Description
Common constants of pi

Name	Meaning
M_PI	pi
M_PI_2	One-half pi
M_PI_4	One-fourth pi
M_1_PI	One divided by pi
M_2_PI	Two divided by pi
M_1_SQRTPI	One divided by the square root of pi
M_2_SQRTPI	Two divided by the square root of pi

M_SQRTxxx

Header File
math.h

Description
Constant values for square roots of 2.

Name	Meaning
M_SQRT2	Square root of 2
M_SQRT_2	1/2 the square root of 2

L_ctermid

Header File
stdio.h

Description
The length of a device id string.

L_tmpnam

Header File
stdio.h

Description
Size of an array large enough to hold a temporary file name string.

TMP_MAX

Header File
stdio.h

Description
Maximum number of unique file names.

OPEN

Header File
stdio.h

Description
Number of files that can be open simultaneously.

Name	Meaning
FOPEN_MAX	Maximum files per process
SYS_OPEN	Maximum files for system

HANDLE_MAX

Header File
io.h

Description
Maximum number of handles.

RAND_MAX

Header File
stdlib.h

Syntax

Description
Maximum value returned by rand function.

BITSPERBYTE

Header File
values.h

Description
Number of bits in a byte.

Float and Double Limits

Header File
values.h

Description

UNIX System V compatible:

_LENBASE	Base to which exponent applies

Limits for double float values

_DEXPLEN	Number of bits in exponent
DMAXEXP	Maximum exponent allowed
DMAXPOWTWO	Largest power of two allowed
DMINEXP	Minimum exponent allowed
DSIGNIF	Number of significant bits
MAXDOUBLE	Largest magnitude double value
MINDOUBLE	Smallest magnitude double value

Limits for float values

_FEXPLEN	Number of bits in exponent
FMAXEXP	Maximum exponent allowed
FMAXPOWTWO	Largest power of two allowed
FMINEXP	Minimum exponent allowed
FSIGNIF	Number of significant bits
MAXFLOAT	Largest magnitude float value
MINFLOAT	Smallest magnitude float value

HIBITxxx

Header File
values.h

Description
Bit mask for the high (sign) bit of standard integer types.

Name	Meaning
HIBITS	For type short
HIBITI	For type int
HIBITL	For type long

Error Numbers in errno

Header File
errno.h

Description
These are the mnemonics and meanings for the error numbers found in errno.

Each value listed can be used to index into the sys_errlist array for displaying messages.

Also, perror will display messages.

Mnemonic	Meaning
EZERO	Error 0
EINVFNC	Invalid function number
ENOFILE	File not found
ENOPATH	Path not found
ECONTR	Memory blocks destroyed
EINVMEM	Invalid memory block address
EINVENV	Invalid environment
EINVFMT	Invalid format
EINVACC	Invalid access code
EINVDAT	Invalid data
EINVDRV	Invalid drive specified
ECURDIR	Attempt to remove CurDir
ENOTSAM	Not same device
ENMFILE	No more files
ENOENT	No such file or directory
EMFILE	Too many open files
EACCES	Permission denied
EBADF	Bad file number
ENOMEM	Not enough memory
ENODEV	No such device
EINVAL	Invalid argument
E2BIG	Arg list too long
ENOEXEC	Exec format error
EXDEV	Cross-device link
EDOM	Math argument
ERANGE	Result too large
EFAULT	Unknown error
EEXIST	File already exists

International API overview

See also

The C++Builder provides support for developing international applications. The C++Builder runtime library now includes extensions to many of the single-byte routines. These extensions allow you to write applications that can process multibyte or Unicode types.

International API routines

See also

To allow maximum portability, C++Builder provides a portable macro for that expands to a multibyte or a Unicode routine without having to rewrite the source code. When you use the portable macros, you can recompile and define one of the following macros.

_MBCS	enables multibyte routines
_UNICODE	enables wide-character routines

If neither macro is defined, the single-byte routines are used.

The following table provides a list of the routines that are available for international applications. The column **Unicode platform support** provides a list of the functions that are not supported on Windows NT. Some Unicode functions are available as macros. When a routine is available as a macro, the macro version is used by default. To get the function version of a routine, you must undefine the macro.

Single byte	Portable macro	Multibyte	Unicode	Unicode platform support
	_istlegal	_ismbclegal	-	Win 95, NT
	_istlead	_ismbblead	-	Win 95, NT
	_isleadbyte	_ismbblead	-	Win 95, NT
_argv	_targv		_wargv	Win NT
_atoi64	_ttoi64		_wtoi64	Win 95, NT
_atold	_ttold		_wtold	Win 95, NT
closedir	_tclosedir		wclosedir	WIN NT
_environ	_tenviron		_wenviron	Win NT
_fdopen	_tfdopen		_wfdopen	Win 95, NT
_fsopen	_tfsopen		_wfsopen	Win NT
_fullpath	_tfullpath		_wfullpath	Win NT
_getdcwd	_tgetdcwd		_wgetdcwd	Win NT
_i64toa	_i64tot		_i64tow	Win 95, NT
_makepath	_tmakepath		_wmakepath	Win 95, NT
_popen	_tpopen		_wpopen	Win NT
readdir	_treaddir		wreaddir	WIN NT
_rtl_chmod	_trtl_chmod		_wrtl_chmod	Win NT
_rtl_creat	_trtl_creat		_wrtl_creat	Win NT
_rtl_open	_trtl_open		_wrtl_open	Win NT
_searchenv	_tsearchenv		_wsearchenv	Win NT
_searchstr	_tsearchstr		_wsearchstr	Win NT
_snprintf	_sntprintf		_snwprintf	Win 95, NT
_splitpath	_tsplitpath		_wsplitpath	Win 95, NT

_strdate	_tstrdate		_wstrdate	Win 95, NT
_strdec	_tcsdec	_mbsdec	_wcsdec	Win 95, NT
_stricoll	_tcsicoll	_mbsicoll	_wcsicoll	Win 95, NT
_strinc	_tcsinc	_mbsinc	_wcsinc	Win 95, NT
_strncnt	_tcsnbcnt	_mbsnbcnt	_wcsncnt	Win 95, NT
_strncoll	_tcsnccoll	_mbsncoll	_wcsncoll	Win 95, NT
_strncoll	_tcsncoll	_mbsnbcoll	_wcsncoll	Win 95, NT
_strnextc	_tcsnextc	_mbsnextc	_wcsnextc	Win 95, NT
_strnicoll	_tcsncicoll	_mbsnbicoll	_wcsnicoll	Win 95, NT
_strnicoll	_tcsnicoll	_mbsnbicoll	_wcsnicoll	Win 95, NT
_strninc	_tcsninc	_mbsninc	_wcsninc	Win 95, NT
_strspnp	_tcsspnp	_mbsspnp	_wcsspnp	Win 95, NT
_strtime	_tstrtime		_wstrtime	Win 95, NT
_strtold	_tcstold		_wcstold	Win 95, NT
_tzname	_ttzname		_wtzname	Win NT
_ui64toa	_ui64tot		_ui64tow	Win 95, NT
access	_taccess		_waccess	Win NT
asctime	_tasctime		_wasctime	Win 95, NT
atof	_ttof		_wtof	Win 95, NT
atoi	_ttoi		_wtoi	Win 95, NT
atol	_ttol		_wtol	Win 95, NT
chdir	_tchdir		_wchdir	Win NT
chmod	_tchmod		_wchmod	Win NT
creat	_tcreat		_wcreat	Win NT
ctime	_tctime		_wctime	Win 95, NT
execl	_texecl		_wexecl	Win NT
execle	_texecle		_wexecle	Win NT
execlp	_texeclp		_wexeclp	Win NT
execlpe	_texeclpe		_wexeclpe	Win NT
execv	_texecv		_wexecv	Win NT
execve	_texecve		_wexecve	Win NT
execvp	_texecvp		_wexecvp	Win NT
execvpe	_texecvpe		_wexecvpe	Win NT
fgetc	_fgettc		fgetwc	Win 95, NT
_fgetchar	_fgettchar		_fgetwchar	Win 95, NT

fgets	_fgetts		fgetws	Win 95, NT
findfirst	_tfindfirst		_wfindfirst	Win NT
findnext	_tfindnext		_wfindnext	Win NT
fnmerge	_tfnmerge		_wfnmerge	Win NT
fnsplit	_tfnsplit		_wfnsplit	Win NT
fopen	_tfopen		_wfopen	Win NT
fprintf	_ftprintf		fwprintf	Win 95, NT
fputc	_fputtc		fputwc	Win 95, NT
_fputchar	_fputtchar		_fputwchar	Win 95, NT
fputs	_fputts		fputws	Win 95, NT
freopen	_tfreopen		_wfreopen	Win NT
getc	_gettc		getwc	Win 95, NT
getchar	_gettchar		getwchar	Win 95, NT
getcurdir	_tgetcurdir		_wgetcurdir	Win NT
getcwd	_tgetcwd		_wgetcwd	Win NT
getenv	_tgetenv		_wgetenv	Win 95, NT
gets	_getts		_getws	Win 95, NT
isalnum	_istalnum	_ismbcalnum	iswalnum	Win 95, NT
isalpha	_istalpha	_ismbcalpha	iswalpha	Win 95, NT
isascii	_istascii		iswascii	Win 95, NT
iscntrl	_istntrl		iswcntrl	Win 95, NT
isdigit	_istdigit	_ismbcdigit	iswdigit	Win 95, NT
isgraph	_istgraph	_ismbcgraph	iswgraph	Win 95, NT
islower	_istlower	_ismbclower	iswlower	Win 95, NT
isprint	_istprint	_ismbcprint	iswprint	Win 95, NT
ispunct	_istpunct	_ismbcpunct	iswpunct	Win 95, NT
isspace	_istspace	_ismbcspace	iswspace	Win 95, NT
isupper	_istupper	_ismbcupper	iswupper	Win 95, NT
isxdigit	_istxdigit		iswxdigit	Win 95, NT
ltoa	_ltot		_ltow	Win 95, NT
main	_tmain		wmain	Win NT
memchr	_tmemchr		_wmemchr	Win 95, NT
memcpy	_tmemcpy		_wmemcpy	Win 95, NT
memset	_tmemset		_wmemset	Win 95, NT
mkdir	_tmkdir		_wmkdir	Win NT

_mktemp	_tmktemp		_wmktemp	Win 95, NT
open	_topen		_wopen	Win NT
opendir	_topendir		wopendir	WIN NT
perror	_tperror		_wperror	Win 95, NT
printf	_tprintf		wprintf	Win 95, NT
putc	_puttc		putwc	Win 95, NT
putchar	_puttchar		putwchar	Win 95, NT
putenv	_tputenv		_wputenv	Win NT
puts	_putts		_putws	Win 95, NT
remove	_tremove		wremove	Win NT
rename	_trename		_wrename	Win NT
rewinddir	_trewinddir		wrewinddir	WIN NT
_rmdir	_trmdir		_wrmdir	Win NT
scanf	_tscanf		wscanf	Win 95, NT
searchpath	_tsearchpath		wsearchpath	Win NT
setlocale	_tsetlocale		_wsetlocale	Win 95, NT
_sopen	_tsopen		_wsopen	Win 95, NT
spawnl	_tspawnl		_wspawnl	Win NT
spawnle	_tspawnle		_wspawnle	Win NT
spawnlp	_tspawnlp		_wspawnlp	Win NT
spawnlpe	_tspawnlpe		_wspawnlpe	Win NT
spawnv	_tspawnv		_wspawnv	Win NT
spawnve	_spawnve		_wspawnve	Win NT
spawnvp	_tspawnvp		_wspawnvp	Win NT
spawnvpe	_tspawnvpe		_wspawnvpe	Win NT
sprintf	_stprintf		swprintf	Win 95, NT
sscanf	_stsscanf		swscanf	Win 95, NT
stat	_tstat		_wstat	Win NT
_stpcpy	_tcspcpy		_wcspcpy	Win 95, NT
strcat	_tcscat	_mbscat	wcscat	Win 95, NT
strchr	_tcschr	_mbschr	wcschr	Win 95, NT
strcmp	_tcscmp	_mbscmp	wcscmp	Win 95, NT
strcmpi	_tcscmpi	_mbsicmp	_wcscmpi	Win 95, NT
strcoll	_tcscoll	_mbscoll	wcscoll	Win 95, NT
strcpy	_tcscpy	_mbscpy	wcscpy	Win 95, NT

strcspn	_tcscspn	_mbscspn	wcscspn	Win 95, NT
strdup	_tcsdup	_mbsdup	_wcsdup	Win 95, NT
strftime	_tcsftime		wcsftime	Win 95, NT
_stricmp	_stricmp	_mbsicmp	_wcsicmp	Win 95, NT
strlen	_tcslen	_mbslen	wcslen	Win 95, NT
strlen	_tcsclen	_mbslen	wcslen	Win 95, NT
strlwr	_tcslwr	_mbslwr	_wcslwr	Win 95, NT
strncat	_tcsncat	_mbsnbcat	wcsncat	Win 95, NT
strncat	_tcsnccat	_mbsncat	wcsncat	Win 95, NT
strncmp	_tcsnccmp	_mbsncmp	wcsncmp	Win 95, NT
strncmp	_tcsncmp	_mbsnbcmp	wcsncmp	Win 95, NT
strncmpi	_tcsncmpi		wcsncmpi	Win 95, NT
strncnt	_tcsnccnt	__mbsncnt	_wcsncnt	Win 95, NT
strncnt	_tcsnbcnt	_mbsnbcnt	_wcsncnt	Win 95, NT
strncpy	_tcsncpy	_mbsnbcpy	wcsncpy	Win 95, NT
strncpy	_tcsnccpy	_mbsncpy	wcsncpy	Win 95, NT
strnicmp	_tcsncicmp	_mbsnicmp	_wcsnicmp	Win 95, NT
strnicmp	_tcsnicmp	_mbsnbicmp	wcsnicmp	Win 95, NT
strnset	_tcsnset	_mbsnbset	_wcsnset	Win 95, NT
strnset	_tcsncset	_mbsnset	_wcsnset	Win 95, NT
strpbrk	_tcspbrk	_mbspbr	wcspbrk	Win 95, NT
strrchr	_tcsrchr	_mbsrchr	wcsrchr	Win 95, NT
strrev	_tcsrev	_mbsrev	_wcsrev	Win 95, NT
strset	_tcsset	_mbsset	_wcsset	Win 95, NT
strspn	_tcsspn	_mbsspn	wcsspn	Win 95, NT
strstr	_tcsstr	_mbsstr	wcsstr	Win 95, NT
strtod	_tcstod		wcstod	Win 95, NT
strtok	_tcstok	_mbstok	wcstok	Win 95, NT
strtol	_tcstol		wcstol	Win 95, NT
strtoul	_tcstoul		_wcstoul	Win 95, NT
strupr	_tcsupr	_mbsupr	_wcsupr	Win 95, NT
strxfrm	_tcsxfrm		wcsxfrm	Win 95, NT
system	_tsystem		_wsystem	Win NT
_tempnam	_ttempnam		_wtempnam	Win 95, NT
tmpnam	_ttmpnam		_wtmpnam	Win 95, NT

tolower	_totlower	_mbctolower	towlower	Win 95, NT
toupper	_totupper	_mbctoupper	towupper	Win 95, NT
tzset	_ttzset		_wtzset	Win 95, NT
ultoa	_ultot		_ultow	Win 95, NT
ungetc	_ungettc		ungetwc	Win 95, NT
_unlink	_tunlink		_wunlink	Win NT
_utime	_tutime		_wutime	Win 95, NT
vfprintf	_vftprintf		vfwprintf	Win 95, NT
vprintf	_vtprintf		vwprintf	Win 95, NT
vsprintf	_vstprintf		vswprintf	Win 95, NT
WinMain	_tWinMain		wWinMain	Win NT

Unicode macros

See also

By default, these Unicode routines are available as a macro. To get the function version, you must undefine the macro.

iswalpha

iswascii

iswcntrl

iswdigit

iswgraph

iswlower

iswprint

iswpunct

iswspace

iswupper

iswxdigit

International API formatted I/O

See also

There are now versions of some runtime library functions that take wide strings (**wchar_t***) instead of narrow strings (**char***). These wide functions have similar names as their narrow counter parts but with a *w* placed in it. For example: along with *printf* and *scanf* there are now *wprintf* and *wscanf* functions. The file TCHAR.H has **#define** names that map to either the narrow versions (for normal ANSI **char**'s) or wide versions (for Unicode support) based on the setting of the _UNICODE macro.

The standard functions operate on regular strings, and the wide versions operate on wide strings. The *printf* and *scanf* family of functions allow you to input or output similar width or opposite width strings with some new format conversion characters and prefixes.

The narrow versions of the functions take narrow format strings and default to reading/writing narrow strings and chars. The wide versions of the functions take wide format strings and default to reading/writing wide strings and chars.

Note: The capitol letter version of **%s** and **%c** (**%S** and **%C**) mean "*use the opposite width than the default for the function that was called*". This means that **%S** in *wprintf* will write to a narrow string.

Also, **%l** and **%h** force the width to be either long (wide) or short (narrow).

Summary of formatted I/O functions

See also

Here is a summary of the current *printf* and *scanf* family of functions.

ANSI function	Unicode function	Description
cprintf	{None}	Console output
cscanf	{None}	Console input
fprintf	fwprintf	FILE * stream output
fscanf	fwscanf	FILE * stream input
printf	wprintf	STDOUT output
scanf	wscanf	STDIN input
sprintf	swprintf	string/memory output
sscanf	swscanf	string/memory input
vfprintf	vfwprintf	VA_LIST FILE* stream output
vfscanf	vfwscanf	VA_LIST FILE* stream input
vprintf	vwprintf	VA_LIST STDOUT output
vscanf	vwscanf	VA_LIST STDIN input

Unicode output format specifiers

See also

The following table shows the formatted ouput specifiers for the Unicode family of functions. The table shows how the format specifier is used by *printf* and the Unicode family of output functions to output strings and characters.

Format specifier	***printf*** **function**	**Unicode function**
%c	narrow	wide
%C	wide	narrow
%hc	narrow	narrow
%hC	narrow	narrow
%lc	wide	wide
%lC	wide	wide
%s	narrow	wide
%S	wide	narrow
%hs	narrow	narrow
%hS	narrow	narrow
%ls	wide	wide
%lS	wide	wide

Unicode family of output functions

The Unicode output family of functions includes the following.

_snprintf

fprintf

sprintf

vfprintf

vprintf

vsprintf

_snprintf

fwprintf

swprintf

vfwprintf

vwprintf

vswprintf

Unicode input format specifiers

See also

The following table shows the formatted ouput specifiers for the Unicode family of functions. The table shows how the format specifier is used by *scanf* and the Unicode family of input functions to input strings and characters.

Format specifier	*scanf* function	Unicode function
%c	narrow	wide
%C	wide	narrow
%hc	narrow	narrow
%hC	narrow	narrow
%lc	wide	wide
%lC	wide	wide
%s	narrow	wide
%S	wide	narrow
%hs	narrow	narrow
%hS	narrow	narrow
%ls	wide	wide
%lS	wide	wide

Unicode family of input functions

The Unicode input family of functions includes the following.

sscanf

swscanf

Extended types formatted I/O

See also

The following table shows new format specifiers implemented in C++Builder for the *printf* and *scanf* family of functions. This implementation allows the input and output of 64-bit integers and provides greater I/O flexibility for other types.

Format character	Functionality
%Ld	**__int64**
%I8d	8-bit wide integer (**char**)
%I16d	16-bit wide integer (**short**)
%I32d	32-bit wide integer (**long**)
%I64d	64-bit wide integer (**__int64**)

Note that the above table uses the **%d** format as an example. The **I8**, **I16**, **I32**, **I64** prefixes can be used with the **d**, **i**, **o**, **x**, **X** formats, as well as the new **L** prefix previously allowed only on **float** to specify **long double** type.