# D-CC™ & D-C++™
# Compiler Suites

## *C Library Reference Manual*

**Version 4.2 10/98**

**Power Compiling Solutions™**

000-5416-001

## Copyright Notice



## Disclaimer

Diab Data makes no representations or warranties with respect to the contents of this publication, and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Diab Data reserves the right to revise this publication and make changes from time to time in the content hereof without obligation on the part of Diab Data to notify any person or company of such revision or changes.

In no event shall Diab Data, or others from whom Diab Data has a licensing right, be liable for any indirect, special, incidental, or consequential damages arising out of or connected with a customers possession or use of this product, even if Diab Data or such others has advance notice of the possibility of such damages.

## Trademarks

Diab Data, alone and in combination with D-AS, D-C++, D-CC, D-F77, D-LD , and FastJ are trademarks of Diab Data, Inc. All other trademarks used in this document are the property of their respective owners.

# Contents

# List of Tables

# 1 Introduction



This is a reference manual for the C libraries provided with Diab Data optimizing compilers. It applies to all targets supported by Diab Data.

It is written for the professional programmer and contains descriptions and references for include files, functions, macros, and variables defined in the libraries.

The libraries are compliant with the following standards and definitions:

ANSI X3.159-1989

ISO/IEC 9945-1:1990

POSIX IEEE Std 1003.1

SVID Issue 2

For C++ specific headers, see “Header files” in the chapter “C++ Features and Compatibility” in the *Language User’s Manual*.

## Document conventions

This manual uses the following typographic conventions:

**Table 1-1 Document conventions**

| Example | Description |
|---|---|
| `dcc -o test.c` | This font is used for file and program names, environment variables, examples, user input, and program output. |
| **if**, **main( )**, **#pragma**, **__pack__** | Bold type is used for keywords, operators and other tokens of the language, library routines and entry points, and section names. |
| | Some names begin or end with underscores. These underscores and special characters such as **#** shown in bold are required. |
| *variable*, *filename* | Italic type is used for placeholders for information which you must supply. Italics are also used for emphasis, to introduce new terms, and for titles. |
| [ optional text ] | An item enclosed in brackets is optional. |
| { item1 \| item2 } | Two or more items enclosed in braces and separated by vertical bars means that you *must* choose exactly one of the items. |
| item ...<br>item ,... | An item followed by "..." means that items of that form may be repeated separated by whitespace (spaces or tabs). A character preceding the "..." means that the items are separated by the character, shown here as a comma, and optional whitespace. |
| | The item may be a single token, an optional item enclosed in [ ] brackets (meaning that the item may appear not at all, once, or multiple times), or a set of choices enclosed in { } braces (meaning that a choice must be made from the enclosed items one or more times). |

## Library structure

➤ Libraries are usually selected automatically by the `dctrl` command or the -t option to the linker. This section is provided for user customization of the process and can be skipped for standard use.

The Diab Data library structure is designed to support a wide range of processors, types of floating point support, and execution environments. This section describes that structure and the mechanism used by the linker to select particular libraries.

This discussion is independent of any target, and should be read in conjunction with the following:

- Chapter 2, “Installing the Compiler,” in the *Language User’s Manual*
- Chapter 2, “Selecting a Target and Its Components,” in the *Target User’s Manual*.

These sections describe the location of the components of the tools and the configuration variables (and their equivalents – environment variables and command line options) used to control their operation. That knowledge is assumed here.

## Libraries supplied

The next table shows the archive libraries distributed with the tools. This does not include `libc.a`, which is not an archive library, but is instead a text file which includes other libraries as described following the table.

**Table 1-2 Library files**

| File | Contents |
|---|---|
| `libcfp.a` | Floating point functions called by user code, including, for example, the **printf** and **scanf** formatting functions (but not the actual device input/output code). The version selected depends on the type of floating point selected: hardware, software, or none as described below.<br><br>Typically included automatically by `libc.a`, see below. |
| `libchar.a` | Basic operating system functions using simple character input/output for `stdin` and `stdout` only (`stderr` and named files are not supported). This is an alternative to `libram.a`.<br><br>Sometimes included automatically by `libc.a`, see below. |
| `libcomplex.a` | C++ complex math class library.<br><br>Not automatic; include with an `-lcomplex` option. |

**Table 1-2 Library files *(continued)***

| File | Contents |
|---|---|
| `libd.a` | Additional standard library and support functions delivered with C++ only (`libc.a` is also required).<br><br>Included automatically in the link command generated by `dplus`. If the linker is invoked directly (command `dld`), then must be included by the user with an `-ld` option. |
| `libi.a` | General library containing all standard ANSI C functions except those in `libcfp.a`, `libchar.a`, and `libram.a`.<br><br>Typically included automatically by `libc.a`, see below. |
| `libimpfp.a` | Conversions between floating point and other types. There are three versions: one for use with hardware floating point, one for software floating point, and an empty file when “none” is selected for floating point. |
| `libimpl.a` | Utility functions called by compiler-generated or runtime code for constructs not implemented in hardware, e.g., low-level software floating point (except conversions), 64-bit integer support, and register save/restore when absent in the hardware.<br><br>Typically included automatically by `libc.a`, see below. |
| `libios.a` | C++ iostream class library.<br><br>Not automatic; include with an `-lios` option. |
| `libm.a` | Advanced math function library.<br><br>Not automatic; include with an `-lm` option. |
| `libram.a` | Basic operating system functions using RAM disk file input/output – an alternative to `libchar.a`.<br><br>Sometimes included automatically by `libc.a`, see below. |

The tools accommodate requirements for different floating point and target operating system and input/output support using two mechanisms:

- `libc.a` is a text file which includes a number of the libraries listed above. Several `libc.a` files which include different combinations are delivered for each target.

- The configuration information held in the configuration variables `DTARGET`, `DOBJECT`, `DFP`, and `DENVIRON` causes `dcc` or `dplus` to generate a particular set of paths used by the linker to search for libraries. By setting these configuration variables appropriately, the user can control the search and consequently the particular `libc.a` or other libraries used by the linker to resolve unsatisfied externals.

  As described in Chapter 2, “Selecting a Target and Its Components,” in the *Target User’s Manual*, these four configuration variables are normally set indirectly using the `dctrl` program or the *-ttof:environ* option on the command line used to invoke the compiler, assembler, or linker.

- The `DENVIRON` configuration variable (set from the *environ* part of *-ttof:environ*) designates the “target operating system” environment. The tools use two standard values: `simple` and `cross`, which as shown below, help define the library search paths.

  In addition, the tools may be supplied with directories and files to support other *environ* operating system values. See `relnote.htm` and any relevant *Application Notes* for details for any particular operating system supported by Diab Data.

The remainder of this section describes these mechanisms in more detail.

## Assumptions

To keep this manual independent of any particular host and target, assume that:

- The target processor is the **targ001**, a member of the **targ** family, and it includes hardware floating point support.
- The object module format specifier – the *‘o’* part of the *-ttof:environ* option or its equivalent, is ‘**E**’ for ELF and ‘**D**’ for COFF; the examples will assume ELF. (Actual targets may use different letters for ELF and COFF.)
- The tools have been installed in the *version_path* directory as described in Chapter 2 in the *Language User’s Manual*.

## Library directory structure

Given the above assumptions, and following the pattern described in “Selected startup module and libraries” in Chapter 2 in the *Target User’s Manual*, the libraries of Table 1-2, “Library files,” above will be arranged as follows (see that section in the *Target User’s Manual* for the exact directories for a particular target):

**Table 1-3 Library directory locations**

| Directory / file | Contents |
|---|---|
| `TARGE/` | Directories and files for ELF components (final '`E`' in `TARGE`). |
| `libc.a` | Text file which includes other ELF libraries as described below – no input/output support. |
| `libchar.a` | ELF basic operating system functions using character input/output for `stdin` and `stdout` only (`stderr` and named files are not supported). |
| `libi.a` | ELF standard ANSI C functions. |
| `libimpl.a` | ELF functions called by compiler-generated or runtime code. |
| `libd.a` | ELF additional C++ standard and support functions. |
| `libram.a` | ELF basic operating system functions using RAM-disk input/output. |
| `cross/libc.a` | ELF `libc.a` which includes the RAM-disk input/output library `libram.a`. |
| `simple/libc.a` | ELF `libc.a` which includes the basic character input/output library `libchar.a`. |
| `TARGEN/` | ELF floating point floating point support of "None". |
| `libcfp.a` | Stubs to avoid undefined externals. |
| `libimpfp.a` | Empty file required by different versions of `libc.a`. |
| `TARGEH/` | ELF hardware floating point libraries: |
| `libcfp.a` | Basic floating point functions |
| `libcomplex.a` | Complex number package (not included automatically) |
| `libimpfp.a` | Conversions between floating point and other types |
| `libios.a` | iostream (not included automatically) |
| `libm.a` | Math library (not included automatically) |

**Table 1-3 Library directory locations *(continued)***

| Directory / file | Contents |
|---|---|
| TARGES/ | ELF software floating point libraries parallel to TARGEH. |
| TARGD<br>TARGDN/<br>TARGDS/ | Parallel directories for COFF components (final 'D' in TARGD) |

## libc.a

There are three libc.a files in the table above. Each of these is a short text file which contains -l option lines, each line naming a library. The -l option is the standard command line option to specify a library for the linker to search. When the linker finds that libc.a is a text file, it reads the -l lines in the liba.c and then searches the named libraries for unsatisfied externals. (As with any -l option, only the portion of the name following "lib" is given; thus, -li identifies library libi.a.)

This approach allows the functions in libc.a to be factored into groups for different floating point and input/output requirements. Three of the libc.a files delivered with the tools are:

**Table 1-4 libc.a files delivered with the tools**

| liba.c files | Contents | Use |
|---|---|---|
| TARGE/libc.a | -li<br>-lcfp<br>-limpl<br>-limpfp | Standard C runtime but with no input/output support; if input/output calls are made they will be undefined. |
| TARGE/simple/libc.a | -li<br>-lcfp<br>-lchar<br>-limpl<br>-limpfp | Supports character input/output by adding libchar.a for stdin and stdout only (stderr and named files are not supported). |
| TARGE/cross/libc.a | -li<br>-lcfp<br>-lram<br>-limpl<br>-limpfp | Supports RAM-disk input/output by adding libram.a. |

Notes:

- Only one of the `simple` or `cross` (or similar) libraries should be used.
- The order of the lines in each `liba.c` file determines the order in which the linker will search for unsatisfied externals.

The particular `libc.a` found, as well as the directories for the libraries listed in each `libc.a`, are determined by the search path given to the linker as described in the next section.

## Library search paths

When `dplus` or `dcc` is invoked, it invokes the compiler, assembler, and linker in turn. The generated linker command line includes:

- an -lc option to cause the linker to search for `libc.a`
- for C++, an -ld option to cause the linker to search for `libd.a`
- a -Y P option which specifies the directories to be searched for these libraries and also for the libraries named in the selected `libc.a` (and any others specified by the user with -l *libname1* options).

The -Y P option generated for each target is a function of the -t*tof*:*environ* option or its equivalent environment variables, and is defined in "Selected startup module and libraries" in Chapter 2 in the *Target User's Manual*.

Following the pattern there, the assumptions made here will generate a -Y P option listing the following directories *in the order given* for each setting of the floating point *'f'* part of the -t:*tof* option or its equivalent, and where *environ* is either `simple` or `cross`:

**Table 1-5 Directories searched for libraries**

| *'f'* | **Directories** | **Environment** | **Floating point support** |
|---|---|---|---|
| **N** | *version_path*/`TARGEN`/*environ* | specific | **N**one |
| | *version_path*/`TARGEN` | generic | **N**one |
| | *version_path*/`TARGE`/*environ* | specific | not applicable |
| | *version_path*/`TARGE` | generic | not applicable |
| **H** | *version_path*/`TARGEH`/*environ* | specific | **H**ardware |
| | *version_path*/`TARGEH` | generic | **H**ardware |
| | *version_path*/`TARGE`/*environ* | specific | not applicable |
| | *version_path*/`TARGE` | generic | not applicable |

**Table 1-5 Directories searched for libraries *(continued)***

| *'f'* | **Directories** | **Environment** | **Floating point support** |
|---|---|---|---|
| **S** | *version_path*/TARGES/*environ* | specific | **S**oftware |
| | *version_path*/TARGES | generic | **S**oftware |
| | *version_path*/TARGE/*environ* | specific | not applicable |
| | *version_path*/TARGE | generic | not applicable |

Notes:

- There is no error if a directory given with the -Y P option does not exist.
- The difference between "None" floating point support and "not applicable" is that the directories for the "not applicable" cases do not contain any floating point code, only integer, while the "None" cases will use the `TARGEN/libcfp.a` and `TARGEN/libimpfp.a` libraries. `TARGEN/libcfp.a` provides stubs functions that call **printf** with an error message for floating point externals used by compiler-generated or runtime code so that these externals will not be undefined; `TARGEN/libimpfp` is an empty file needed because each `libc.a` is common to all types of floating point support.

The following table gives examples of the libraries found given the above directory search order. Note that the search for the libraries included by a `libc.a` is independent of the search for `libc.a`. That is, regardless of which directory supplies `libc.a`, the search for the libraries it names begins anew with the first directory in the selected row of Table 1-5, "Directories searched for libraries," above. In all cases, a library is taken from the first directory in which it is found.

**Table 1-6 Examples of libraries found for different -t options**

| -t **option** | **Libraries found** | **Notes** |
|---|---|---|
| `-tTARGEN:simple` | `TARGE/simple/libc.a`<br><br>`TARGE/libi.a`<br>`TARGEN/libcfp.a`<br>`TARGE/libchar.a`<br>`TARGE/libimpl.a`<br>`TARGEN/libimpfp.a` | `libc.a` is specific to the environment, but never to the floating point support. It is found in the third directory searched. It names four libraries:<br>• `libi.a` and `libimpl.a` are common to all `TARGE` systems and are found in the fourth directory `TARGE`.<br>• The floating point support is independent of the environment and comes from the second directory `TARGEN`.<br>• The character input/output support is independent of the floating point support, and while it has been selected because of the `simple` environment setting, it resides in the generic fourth directory `TARGE`. |
| `-tTARGES:cross` | `TARGE/cross/libc.a`<br><br>`TARGE/libi.a`<br>`TARGES/libcfp.a`<br>`TARGE/libram.a`<br>`TARGE/libimpl.a`<br>`TARGES/libimpfp.a` | Again, `libc.a` is specific to the environment but not the floating point support, and is found in the third directory `TARGE/cross`. It again names four libraries:<br>• `libi.a` and `libimpl.a` are in the fourth directory `TARGE` as before.<br>• The software floating point library `libcfp.a` is from the second directory, now `TARGES`.<br>• This time `libram.a` has been selected by `TARGE/cross/libc.a` instead of `libchar.a` (but still from the fourth directory `TARGE` as before). |

**Table 1-6 Examples of libraries found for different -t options *(continued)***

| -t **option** | **Libraries found** | **Notes** |
|---|---|---|
| `-tTARGES:cust` | `TARGE/cust/libc.a`<br><br>`TARGE/libi.a`<br>`TARGES/libcfp.a`<br>`TARGE/cust/libchar.a`<br>`TARGE/libimpl.a`<br>`TARGES/libimpfp.a` | The customer has defined a new `libc.a` in a new `TARGE/cust` directory for a C++ project using software floating point. This `libc.a` text file consists of the following five lines:<br><br>`-li`<br>`-lcfp`<br>`-lchar`<br>`-limpl`<br>`-limpfp`<br><br>Thus, based on the search order implied by the `-tTARGES:cust` option, the standard libraries `TARGE/libi.a`, `TARGE/libimpl.a`, `TARGES/libcfp.a`, and `TARGES/libimpfp.a` will be searched.<br><br>In addition, the library `TARGE/cust/lib-char.a`, a special character I/O package for the customer's target environment, will also be searched. Because directory `TARGES/cust` is searched before `TARGE`, the linker will find the customer's `libchar.a` library rather than the standard `TARGE/libchar.a`. |

# 2 Include Files

2



## Files

The following list is a subset of the include files provided. Each is enclosed in angle brackets, < >, whenever used in text to emphasize their inclusion in the standard C library.

All include files are found in *version_path*`/include`. See “Installation and compiler components” in Chapter 2, “Installing the Compiler,” in the *Language User’s Manual* for additional information.

➤ In this manual, some paths are given using UNIX format, that is, using a '/' separator. For DOS, substitute a'\' separator; for MPW, use '{ }' and ':' as required.

**Table 2-1 Standard Include Files**

| Name | Description |
|---|---|
| <aouthdr.h> | COFF optional header |
| <ar.h> | archive header |
| <assert.h> | **assert( )** macro |
| <ctype.h> | character handling macros |
| <dcc.h> | prototypes not found elsewhere |
| <errno.h> | error macros and **errno** variable |
| <fcntl.h> | **creat( )**, **fcntl( )**, and **open( )** definitions |
| <filehdr.h> | COFF file header |
| <float.h> | floating point limits |
| <limits.h> | limits of processor and operating system |
| <linenum.h> | COFF line number definitions |
| <locale.h> | locale definitions |
| <malloc.h> | old **malloc( )** definitions. Use <stdlib.h> |
| <math.h> | defines the constant **HUGE_VAL** and declares math functions |
| <mathf.h> | single precision versions of <math.h> functions |
| <memory.h> | old declarations of **mem*( )**. Use <string.h> |
| <mon.h> | monitor( ) definitions |
| <regexp.h> | regular expression handling |
| <reloc.h> | COFF relocation entry definitions |
| <scnhdr.h> | COFF section header definitions |

**Table 2-1 Standard Include Files *(continued)***

| Name | Description |
|---|---|
| <search.h> | search routine declarations |
| <setjmp.h> | **setjmp( )** and **longjmp( )** definitions |
| <signal.h> | signal handling |
| <stdarg.h> | ANSI variable arguments handling |
| <stddef.h> | ANSI definitions |
| <stdio.h> | **stdio** library definitions |
| <stdlib.h> | ANSI definitions |
| <storclass.h> | COFF storage classes |
| <string.h> | **str*( )** and **mem*( )** declarations |
| <syms.h> | COFF symbol table definitions |
| <sys/types.h> | type definitions |
| <time.h> | time handling definitions |
| <unistd.h> | prototypes for UNIX system calls |
| <values.h> | old limits definitions. Use <limits.h> and <float.h> |
| <varargs.h> | old variable arguments handling. Use <stdarg.h> |

## Defined Variables, Types, and Constants

The following list is a subset of the variables, types, and constants defined in the include files in the D-CC libraries.

### errno.h

Declares the variable **errno** holding error codes. Defines error codes; all starting with **E**. See the file for more information.

## fcntl.h

Defines the following constants used by **open( )** and **fcntl( )**:

| | |
|---|---|
| **O_RDONLY** | open for reading only |
| **O_WRONLY** | open for writing only |
| **O_RDWR** | open for reading and writing |
| **O_NDELAY** | no blocking |
| **O_APPEND** | append all writes at the end of the file |

## float.h

Defines constants handling the precision and range of floating point values. See the ANSI C standard for reference.

## limits.h

Defines constants defining the range of integers and operating system limits. See the ANSI C and POSIX 1003.1 standards for reference.

## math.h

Defines the value **HUGE_VAL** that is set to IEEE double precision infinity.

## mathf.h

Defines the value **HUGE_VAL_F** that is set to IEEE single precision infinity.

## setjmp.h

Defines the type **jmpbuf**, used by **setjmp( )** and **longjmp( )**.
Defines the type **sigjmpbuf**, used by **sigsetjmp( )** and **siglongjmp( )**.

## signal.h

Defines the signal macros starting with SIG.
Defines the volatile type **sig_atomic_t** that can be used by signal handlers.
Defines the type **sigset_t**, used by POSIX signal routines.

### stdarg.h

Defines the type **va_list** used by the macros **va_start**, **va_arg**, and **va_end**.

### stddef.h

Defines **ptrdiff_t** which is the result type of subtracting two pointers.
Defines **size_t** which is the result type of the **sizeof** operator.
Defines **NULL** which is the null pointer constant.

### stdio.h

Defines **size_t** which is the result type of the **sizeof** operator.
Defines **fpos_t** which is the type used for file positioning.
Defines **FILE** which is the type used by stream and file input and output.
Defines the **BUFSIZ** constant which is the size used by **setbuf( )**.
Defines the **EOF** constant which indicates end-of-file.
Defines **NULL** which is the null pointer constant.
Declares **stdin** as a pointer to the **FILE** associated with standard input.
Declares **stdout** as a pointer to the **FILE** associated with standard output.
Declares **stderr** as a pointer to the **FILE** associated with standard error.

### stdlib.h

Defines **size_t** which is the result type of the **sizeof** operator.
Defines **div_t** and **ldiv_t** which are the types returned by **div( )** and **ldiv( )**.
Defines **NULL** which is the null pointer constant.
Defines the **EXIT_FAILURE** and **EXIT_SUCCESS** constants returned by **exit( )**.

### string.h

Defines **NULL** which is the null pointer constant.
Defines **size_t** which is the result type of the **sizeof** operator.

### time.h

Defines **CLOCKS_PER_SEC** constant which is the number of clock ticks per second.

# 3 Functions




## Format of Descriptions

This chapter describes the functions and function-like macros provided in the D-CC libraries. The descriptions are not a complete definition of the functions, but rather a brief explanation for the experienced user.

Each function description is formatted as follows:

**name** **include files**
**prototype definition**

brief description

OS calls: optional; see below

Reference: see below

### Operating system calls

Some of the functions described in this chapter make calls on operating system functions that are standard in UNIX environments. In embedded environments, such functions cannot be used unless the embedded environment includes a real-time operating system providing these operating system functions.

The functions which call operating system functions, directly or indirectly, have all the required operating system functions listed. The non-UNIX user can employ this list to see what system functions need to be provided in order to use a particular function.

Some functions refer to standard input, output, and error – the standard input/output streams

found in UNIX and DOS environments. For embedded environments, see the *Target User's Manual* for suggestions for file system support.

## References

All functions have references to the following standards and definitions:

| | |
|---|---|
| ANSI | The function/macro is defined in ANSI X3.159-1989. |
| ANSI 754 | The function is define in ANSI/IEEE Std 754-1985. |
| DCC | The function/macro is added to D-CC. |
| POSIX | The function/macro is defined in IEEE Std 1003.1-1990. |
| SVID | The function/macro is defined in System V Interface Definition 2. |
| UNIX | The function/macro is provided to be compatible with Unix V.3. |

Other references:

| | |
|---|---|
| MATH | The math libraries must be specified at link time with the -lm option. |
| SYS | The function must be provided by the operating system or emulated in a stand-alone system. |
| REENT | The function is reentrant. It does not use any static or global data. |
| REERR | The function might modify **errno** and is reentrant only if all processes ignore that variable |

Most functions in the libraries have a synonym to conform to various standards. For example, the function **read( )** has the synonym **_read( )**. In ANSI C, **read( )** is not defined, which means that the user is free to define **read( )** as a new function. To avoid conflicts with such user-defined functions, library functions, e.g., **fread( )**, call the synonym defined with the leading underscore, e.g., **_read( )**.

# Function Listing

## a64l

**#include <stdlib.h>**
**long a64l(const char ****s***);**

Converts the base-64 number, pointed to by *s, to a long value.

Reference: SVID, REENT.

## abort

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

Same as **exit( )**, but also causes the signal **SIGABRT** to be sent to the calling process. If **SIGABRT** is neither caught nor ignored, all streams are flushed prior to the signal being sent and a core dump results.

OS calls: **close**, **getpid**, **kill**, **sbrk**, **write**.

Reference: ANSI.

## abs

**#include <stdlib.h>**
**int abs(int** *i***);**

Returns the absolute value of its integer operand.

Reference: ANSI, REENT.

## access

**#include <unistd.h>**
**int access(char ****path***, int** *amode***);**

Determines accessibility of a file.

The D-CC libraries provide an interface to this operating system call. Please see your OS manual for a complete definition.

Reference: POSIX, SYS.

## acos

**#include <math.h>**
**double acos(double** *x***);**

Returns the arc cosine of *x* in the range [0, π]. *x* must be in the range [-1, 1]. Otherwise zero is returned, **errno** is set to **EDOM**, and a message indicating a domain error is printed on the standard error output.

OS calls: **write**.

Reference: ANSI, MATH, REERR.

## acosf

**#include <mathf.h>**
**float acosf(float** *x***);**

Returns the arc cosine of *x* in the range [0, π]. *x* must be in the range [-1, 1]. Otherwise zero is returned, **errno** is set to **EDOM**, and a message indicating a domain error is printed on the standard error output. This is the single precision version of **acos( )**.

OS calls: **write**.

Reference: DCC, MATH, REERR.

## advance

**#include <regexp.h>**
**int advance(char ****string***, char ****expbuf***);**

Does pattern matching given the string *string* and a compiled regular expression in *expbuf*. See SVID for more details.

Reference: SVID.

## alloca

**#include <dcc.h>**
**void *alloca(size_t** *size***)**

Allocates temporary local stack space for an object of size *size*. Returns a pointer to the start of the object. The allocated memory will be released at return from the current function.

Reference: DCC, REENT.

## asctime

**#include <time.h>**
**char *asctime(const struct tm ***timeptr***);**

Converts time in *timeptr* into a string in the form exemplified by

```
"Sun Sep 16 01:03:52 1973\n".
```

Reference: ANSI.

## asin

**#include <math.h>**
**double asin(double** *x)**;**

Returns the arc sine of x in the range [-π/2, π/2]. x must be in the range [-1, 1]. Otherwise zero is returned, **errno** is set to **EDOM** and a message indicating a domain error is printed on the standard error output.

OS calls: **write**.

Reference: ANSI, MATH, REERR.

## asinf

**#include <mathf.h>**
**float asinf(float** *x***);**

Returns the arc sine of *x* in the range [-π/2, π/2]. *x* must be in the range [-1, 1]. Otherwise zero is returned, **errno** is set to **EDOM** and a message indicating a domain error is printed on the standard error output. This is the single precision version of **asin( )**.

OS calls: **write**.

Reference: DCC, MATH, REERR.

## assert

**#include <assert.h>**
**void assert(int** *expression***);**

Puts diagnostics into programs. If *expression* is false, **assert( )** writes information about the particular call that failed (including the text of the argument, the name of the source file, and the source line number – the latter are respectively the values of the preprocessing macros **__FILE__** and **__LINE__**) on the standard error file. It then calls the **abort( )**

function. **assert( )** is implemented as a macro. If the preprocessor macro **NDEBUG** is defined at compile time, the **assert( )** macro will not generate any code.

OS calls: **close**, **getpid**, **kill**, **sbrk, write**.

Reference: ANSI.

## atan

**#include <math.h>**
**double atan(double** *x***);**

Returns the arc tangent of *x* in the range [-π/2, π/2].

OS calls: **write**.

Reference: ANSI, MATH, REERR.

## atanf

**#include <mathf.h>**
**float atan(float** *x***);**

Returns the arc tangent of *x* in the range [-π/2, π/2]. This is the single precision version of **atan( )**.

OS calls: **write**.

Reference: DCC, MATH, REERR.

## atan2

**#include <math.h>**
**double atan2(double** *x***, double** *y***);**

Returns the arc tangent of *y/x* in the range [-π, π], using the signs of both arguments to determine the quadrant of the return value. If both arguments are zero, then zero is returned, **errno** is set to **EDOM** and a message indicating a domain error is printed on the standard error output.

OS calls: **write**.

Reference: ANSI, MATH, REERR.

## atan2f

**#include <mathf.h>**
**float atan2(float** *x***, float** *y***);**

Returns the arc tangent of *y/x* in the range [-π, π], using the signs of both arguments to determine the quadrant of the return value. If both arguments are zero, then zero is returned, **errno** is set to **EDOM** and a message indicating a domain error is printed on the standard error output. This is the single precision version of **atan2( )**.

OS calls: **write**.

Reference: DCC, MATH, REERR.

## atexit

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

Registers the function whose address is *func* to be called by **exit( )**.

Reference: ANSI.

## atof

**#include <stdlib.h>**
**double atof(const char *****nptr***);**

Converts an ASCII number string *nptr* into a **double**.

Reference: ANSI, REERR.

## atoi

**#include <stdlib.h>**
**int atoi(const char *****nptr***);**

Converts an ASCII decimal number string *nptr* into an **int**.

Reference: ANSI, REENT.

## atol

**#include <stdlib.h>**
**long atol(const char ****nptr***);**

Converts an ASCII decimal number string *nptr* into a **long**.

Reference: ANSI, REENT.

## bsearch

**#include <stdlib.h>**
**void *bsearch(const void ****key***, const void ****base***, size_t** *nel***, size_t** *size***,**
**int (****compar***)( ));**

Binary search routine which returns a pointer into a table indicating where a datum may be found. The table must be previously sorted in increasing order. *key* points to a datum instance to search for in the table, *base* points to the element at the base of the table, *nel* is the number of elements in the table. *compar* is a pointer to the comparison function, which is called with two arguments that point to the elements being compared.

Reference: ANSI, REENT.

## calloc

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

Allocates space for an array of *nmemb* objects of the size *size*. Returns a pointer to the start (lowest byte address) of the object. The array is initialized to zero. See **malloc( )** for more information.

OS calls: **sbrk**, **write**.

Reference: ANSI.

## ceil

**#include <math.h>**
**double ceil(double** *x***);**

Returns the smallest integer not less than *x*.

OS calls: **write**.

Reference: ANSI, MATH, REENT.

## ceilf

**#include <mathf.h>**
**float ceilf(float** *x***);**

Returns the smallest integer not less than *x*. This is the single precision version of **ceil( )**.

OS calls: **write**.

Reference: DCC, MATH, REENT.

## _chgsign

**#include <math.h>**
**double _chgsign(double** *x***);**

Returns *x* copies with its sign reversed, not 0 - *x*. The distinction is germane when *x* is +- or -0 or NaN. Consequently, it is a mistake to use the sign bit to distinguish signaling NaNs from quite NaNs.

Reference: ANSI 754, MATH, REENT.

## clearerr

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

Resets the error and EOF indicators to zero on the named *stream*.

Reference: ANSI.

## clock

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

Returns the number of clock ticks of elapsed processor time, counting from a time related to program start-up. The constant **CLOCKS_PER_SEC** is the number of ticks per second.

OS calls: **times**.

Reference: ANSI.

## close

**#include <unistd.h>**
**int close(int** *fildes***);**

Closes the file descriptor *fildes*.

The D-CC libraries provide an interface to this operating system call. Please see your OS manual for a complete definition.

Reference: POSIX, SYS.

## compile

**#include <regexp.h>**
**int compile(char ****instring***, char ****expbuf***, char ****endbuf***, int** *eof***);**

Compiles the regular expression in *instring* and produces a compiled expression that can be used by **advance( )** and **step( )** for pattern matching.

Reference: SVID.

## _copysign

**#include <math.h>**
**double _copysign(double** *x***, double** *y***);**

Returns *x* with the sign of *y*. Hence, `abs(x) = _copysign(x, 1.0)` even if *x* is NaN.

Reference: ANSI 754, MATH, REENT.

## cos

**#include <math.h>**
**double cos(double** *x***);**

Returns the cosine of *x* measured in radians. Accuracy is reduced with large argument values.

OS calls: **write**.

Reference: ANSI, MATH, REERR.

## cosf

**#include <mathf.h>**
**float cosf(float** *x***);**

Returns the cosine of *x* measured in radians. Accuracy is reduced with large argument values. This is the single precision version of **cos( )**.

OS calls: **write**.

Reference: DCC, MATH, REERR.

## cosh

**#include <math.h>**
**double cosh(double** *x***);**

Returns the hyperbolic cosine of *x* measured in radians. Accuracy is reduced with large argument values.

OS calls: **write**.

Reference: ANSI, MATH, REERR.

## coshf

**#include <mathf.h>**
**float coshf(float** *x***);**

Returns the hyperbolic cosine of *x* measured in radians. Accuracy is reduced with a large argument values. This is the single precision version of **cosh( )**.

OS calls: **write**.

Reference: DCC, MATH, REERR.

## creat

**#include <sys/types.h>**
**#include <sys/stat.h>**
**#include <fcntl.h>**
**int creat(char ****path***, mode_t** *mode***);**

Creates the new file *path*.

The D-CC libraries provide an interface to this operating system call. Please see your OS manual for a complete definition.

Reference: POSIX, SYS.

## ctime

**#include <time.h>**
**char *ctime(const time_t **timer*);**

Equivalent to calling **asctime(localtime(*timer*))**.

Reference: ANSI.

## difftime

**#include <time.h>**
**double difftime(time_t t1, time_t *t0*);**

Returns the difference in seconds between the calendar time *t0* and the calendar time *t1*.

Reference: ANSI, REENT.

## div

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

Divides *numer* by *denom* and returns the quotient and the remainder as a **div_t** structure.

Reference: ANSI, REENT.

## drand48

**#include <stdlib.h>**
**double drand48(*void*);**

Generates pseudo-random, non-negative, double-precision floating-point numbers uniformly distributed over the half open interval [0.0, 1.0[ (i.e. excluding 1.0), using the linear congruential algorithm and 48-bit integer arithmetic. It must be initialized using the **srand48( )**, **seed48( )**, or **lcong48( )** functions.

Reference: SVID.

**#include <unistd.h>**
**int dup(int** *fildes***);**

Duplicates the open file descriptor *fildes*.

The D-CC libraries provide an interface to this operating system call. Please see your OS manual for a complete definition.

Reference: POSIX, SYS.

## ecvt

**#include <dcc.h>**
**char *ecvt(double** *value***, int** *ndigit***, int *****decpt***, int *****sign***);**

Converts *value* to a null-terminated string of *ndigit* digits and returns a pointer to it. The high-order digit is non-0 unless *value* is zero. The low-order digit is rounded to the nearest value (5 is rounded up). The position of the decimal point relative the beginning of the string is stored through *decpt* (negative means to the left of the returned digits). If the sign of the result is negative, the integer pointed to by *sign* is set to one, otherwise it is set to zero.

Reference: DCC.

## erf

**#include <math.h>**
**double erf(double** *x***);**

Returns the error function of *x*.

Reference: SVID, MATH, REENT.

## erff

**#include <mathf.h>**
**float erff(float** *x***);**

Returns the error function of *x*. This is the single precision version of **erf( )**.

Reference: DCC, MATH, REENT.

## erfc

**#include <math.h>**
**double erfc(double** *x***);**

Complementary error function = 1.0 - **erf(***x***)**. Provided because of the extreme loss of relative accuracy if **erf(***x***)** is called for large *x* and the result subtracted from 1.0.

Reference: SVID, MATH, REENT.

## erfcf

**#include <mathf.h>**
**float erfcf(float** *x***);**

Complementary error function = 1.0 - **erff**(*x*). Provided because of the extreme loss of relative accuracy if **erff(***x***)** is called for large *x* and the result subtracted from 1.0. This is the single precision version of **erfc( )**.

Reference: DCC, MATH, REENT.

## exit

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

Normal program termination. Flushes all open files. Executes all functions submitted by the **atexit( )** function. Does not return to its caller. The following *status* constants are provided:

| | |
|---|---|
| **EXIT_FAILURE** | unsuccessful termination |
| **EXIT_SUCCESS** | successful termination |

OS calls: **_exi**t, **close**, **sbrk, write**.

Reference: ANSI.

## _exit

**#include <unistd.h>**
**void _exit(int** *status***);**

Program termination. All files are closed. Does not return to its caller.

The D-CC libraries provide an interface to this operating system call. Please see your OS manual for a complete definition.

Reference: POSIX, SYS.

## exp

**#include <math.h>**
**double exp(double** *x***);**

Returns the exponential function of *x*. Returns **HUGE_VAL** when the correct value would overflow or 0 when the correct value would underflow, and sets **errno** to **ERANGE**.

OS calls: **write**.

Reference: ANSI, MATH, REERR.

## expf

**#include <mathf.h>**
**float expf(float** *x***);**

Returns the exponential function of *x*. Returns **HUGE_VAL** when the correct value would overflow or 0 when the correct value would underflow and sets **errno** to **ERANGE**. This is the single precision version of **exp( )**.

OS calls: **write**.

Reference: DCC, MATH, REERR.

## fabs

**#include <math.h>**
**double fabs(double** *x***);**

Returns the absolute value of *x*.

Reference: ANSI, MATH, REENT.

## fabsf

**#include <mathf.h>**
**float fabsf(float** *x***);**

Returns the absolute value of *x*. This is the single precision version of **fabs( )**.

Reference: DCC, MATH, REENT.

## fclose

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

Causes any buffered data for the named *stream* to be written out, and the stream to be closed.

OS calls: **close**, **sbrk, write**.

Reference: ANSI.

## fcntl

**#include <fcntl.h>**
**int fcntl(int** *fildes***, int** *cmd***, ...);**

Controls the open file *fildes*.

The D-CC libraries provide an interface to this operating system call. Please see your OS manual for a complete definition.

Reference: POSIX, SYS.

## fcvt

**#include <dcc.h>**
**char *fcvt(double** *value***, int** *ndigit***, int ****decpt***, int ****sign***);**

Rounds the correct digit for **printf** format "**%f**" (FORTRAN F-format) output according to the number of digits specified. See **ecvt( )**.

Reference: DCC.

## fdopen

**#include <stdio.h>**
**FILE *fdopen(int** *fildes***, const char ****type***);**

See **fopen( )**. **fdopen( )** associates a stream with a file descriptor, obtained from **open( )**, **dup( )**, **creat( )**, or **pipe( )**. The *type* of stream must agree with the mode of the open file.

OS calls: **fcntl**, **lseek**.

Reference: POSIX.

## feof

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

Returns non-zero when end-of-file has previously been detected reading the named input *stream*.

Reference: ANSI.

## ferror

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

Returns non-zero when an input/output error has occurred while reading from or writing to the named *stream*.

Reference: ANSI.

## fflush

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

Causes any buffered data for the named *stream* to be written to the file, and the *stream* remains open.

OS calls: **write**.

Reference: ANSI.

## fgetc

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

Behaves like the macro **getc( )**, but is a function. Runs more slowly than **getc( )**, takes less space, and can be passed as an argument to a function.

OS calls: **isatty**, **read**, **sbrk**, **write**.

Reference: ANSI.

## fgetpos

**#include <stdio.h>**
**int fgetpos(FILE *_stream_, fpos_t *_pos_);**

Stores the file position indicator for *stream* in **pos*. If unsuccessful, it stores a positive value in **errno** and returns a nonzero value.

OS calls: **lseek**.

Reference: ANSI.

## fgets

**#include <stdio.h>**
**char *fgets(char *_s_, int _n_, FILE *_stream_);**

Reads characters from *stream* into the array pointed to by *s*, until *n*-1 characters are read, or a new-line character is read and transferred to *s*, or an EOF is encountered. The string is terminated with a null character.

OS calls: **isatty**, **read**, **sbrk**, **write**.

Reference: ANSI.

## fileno

**#include <stdio.h>**
**int fileno (FILE *_stream_);**

Returns the integer file descriptor associated with the named *stream*; see **open( )**.

Reference: POSIX.

## _finite

**#include <math.h>**
**double _finite(double _x_);**

Returns a non-zero value if $-\infty < x < +\infty$, and returns 0 otherwise.

Reference: ANSI 754, MATH, REENT

## floor

**#include <math.h>**
**double floor(double** *x***);**

Returns the largest integer (as a double-precision number) not greater than *x*.

Reference: ANSI, MATH, REENT.

## floorf

**#include <mathf.h>**
**float floorf(float** *x***);**

Returns the largest integer (as a single-precision number) not greater than *x*. This is the single precision version of **floor( )**.

Reference: DCC, MATH, REENT.

## fmod

**#include <math.h>**
**double fmod(double** *x***, double** *y***);**

Returns the floating-point remainder of the division of *x* by *y*, zero if *y* is zero or if *x*/*y* would overflow. Otherwise the number is f with the same sign as *x*, such that *x*=i*y*+f for some integer i, and absolute value of f is less than absolute value of *y*.

Reference: ANSI, MATH, REENT.

## fmodf

**#include <mathf.h>**
**float fmodf(float** *x***, float** *y***);**

Returns the floating-point remainder of the division of *x* by *y*, zero if *y* is zero or if *x*/*y* would overflow. Otherwise the number is f with the same sign as x, such that *x*=i*y*+f for some integer i, and absolute value of f is less than absolute value of *y*. This is the single precision version of **fmod( )**.

Reference: DCC, MATH, REENT.

## fopen

**#include <stdio.h>**
**FILE *fopen(const char *_filename_, const char *_type_);**

Opens the file named by *filename* and associates a stream with it. Returns a pointer to the **FILE** structure associated with the stream. *type* is a character string having one of the following values:

| | |
|---|---|
| **"r"** | open for reading |
| **"w"** | truncate or create for writing |
| **"a"** | append; open for writing at EOF, or create for writing |
| **"r+"** | open for update (read and write) |
| **"w+"** | truncate or create for update |
| **"a+"** | append; open or create for update at EOF |

A **"b"** can also be specified as the second or third character in the above list, to indicate a binary file on systems where there is a difference between text files and binary files. Examples: **"rb"**, **"wb+"**, and **"a+b"**.

OS calls: **lseek**, **open**.

Reference: ANSI.

## fprintf

**#include <stdio.h>**
**int fprintf(FILE *_stream_, const char *_format_, ... );**

Places output argument on named output stream. See **printf( )**.

---

➤ By default in most environments, **fprintf** buffers its output until a newline is output. To cause output character-by-character without waiting for a newline, call **setbuf** (page 74) with a NULL buffer pointer after opening but before writing to the stream:

```
setbuf(*stream, 0);
```

---

OS calls: **isatty**, **sbrk**, **write**.

Reference: ANSI.

## fputc

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

Behaves like the macro **putc( )**, but is a function. Therefore, it runs more slowly, takes up less space, and can be passed as an argument to a function.

OS calls: **isatty**, **sbrk**, **write**.

Reference: ANSI.

## fputs

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

Writes the null-terminated string pointed to by s to the named output *stream*.

OS calls: **isatty**, **sbrk**, **write**.

Reference: ANSI.

## fread

**#include <stdio.h>**
**#include <sys/types.h>**
**int fread(void ****ptr***, size_t** *size***, int** *nitems***, FILE ****stream***);**

Copies *nitems* items of data from the named input *stream* into an array pointed to by *ptr*, where an item of data is a sequence of bytes of length *size*. It leaves the file pointer in *stream* pointing to the byte following the last byte read.

OS calls: **isatty**, **read**, **sbrk**, **write**.

Reference: ANSI.

## free

**#include <stdlib.h>**
**void free(void *_ptr_);**
**extern int _ _no_malloc_warning;**

Object pointed to by *ptr* is made available for further allocation. *ptr* must previously have been assigned a value from **malloc( )**, **calloc( )**, or **realloc( )**.

If the pointer *ptr* was freed or not allocated by **malloc( )**, a warning is printed on the **stderr** stream. The warning can be suppressed by assigning a non-zero value to the integer **__no_malloc_warning**. See **malloc( )** for more information.

OS calls: **sbrk**, **write**.

Reference: ANSI.

## freopen

**#include <stdio.h>**
**FILE *freopen(const char *_filenam_, const char *_type_, FILE *_stream_);**

See **fopen( )**. **freopen( )** opens the named file in place of the open *stream*. The original stream is closed, and a pointer to the **FILE** structure for the new stream is returned.

OS calls: **close**, **lseek**, **open**, **sbrk**, **write**.

Reference: ANSI.

## frexp

**#include <math.h>**
**double frexp(double** *value*, **int *_eptr_);**

Given that every non-zero number can be expressed as $x*(2^n)$, where $0.5<=|x|< 1.0$ and n is an integer, this function returns x for a *value* and stores n in the location pointed to by *eptr*.

Reference: ANSI, REENT.

## frexpf

**#include <mathf.h>**
**float frexpf(float** *value***, int ****eptr***);**

Given that every non-zero number can be expressed as x*($2^n$), where 0.5<=|x|< 1.0 and n is an integer, this function returns x for a *value* and stores n in the location pointed to by *eptr*. This is the single precision version of **frexp( )**.

Reference: DCC, MATH, REENT.

## fscanf

**#include <stdio.h>**
**int fscanf(FILE ****stream***, const char ****format***, ...);**

Reads formatted data from the named input *stream* and optionally assigns converted data to variables specified by the *format* string. Returns the number of successful conversions (or **EOF** if input is exhausted). See **scanf( )**.

OS calls: **isatty**, **read**, **sbrk**, **write**.

Reference: ANSI.

## fseek

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

Sets the position of the next input or output operation on the *stream*. The new position is at the signed distance *offset* bytes from the beginning, from the current position, or from the end of the file, according to *whence*. The next operation on a file opened for update may be either input or output. whence has one of the following values:

| | |
|---|---|
| **SEEK_SET** | offset is absolute position from beginning of file. |
| **SEEK_CUR** | offset is relative distance from current position. |
| **SEEK_END** | offset is relative distance from the end of the file. |

OS calls: **lseek**, **write**.

Reference: ANSI.

## fsetpos

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

Sets the file position indicator for *stream* to **pos* and clears the EOF indicator for *stream*. If unsuccessful, stores a positive value in **errno** and returns a nonzero value.

OS calls: **lseek**, **write**.

Reference: ANSI.

## fstat

**#include <sys/types.h>**
**#include <sys/stat.h>**
**int fstat(int** *fildes***, struct stat ***buf***);**

Gets file status for the file descriptor *fildes*.

The D-CC libraries provide an interface to this operating system call. Please see your OS manual for a complete definition.

Reference: POSIX, SYS.

## ftell

**#include <stdio.h>**
**long ftell(FILE ***stream***);**

See **fseek( )**. Returns the offset of the current byte relative to the beginning of the file associated with the named *stream*.

OS calls: **lseek**.

Reference: ANSI.

## fwrite

**#include <stdio.h>**
**#include <sys/types.h>**
**int fwrite(const void ***ptr***, size_t** *size***, int** *nitems***, FILE ***stream***);**

Appends at most *nitems* items of data from the array pointed to by *ptr* to the named output *stream*. See **fread( )**.

OS calls: **isatty**, **sbrk**, **write**.

Reference: ANSI.

## gamma

**#include <math.h>**
**double gamma(double** *x***);**
**extern int** *signgam***;**

Returns the natural logarithm of the absolute value of the gamma function of *x*. The argument *x* must be a positive integer. The sign of the gamma function is returned as -1 or 1 in *signgam*.

OS calls: **write**.

Reference: UNIX, MATH, REERR.

## gammaf

**#include <mathf.h>**
**float gammaf(float** *x***);**
**extern int** *signgamf***;**

Returns the natural logarithm of the absolute value of the gamma function of *x*. The argument *x* must be a positive integer. The sign of the gamma function is returned as -1 or 1 in *signgamf*. This is the single precision version of **gamma( )**.

OS calls: **write**.

Reference: DCC, MATH, REERR.

## gcvt

**#include <dcc.h>**
**char *gcvt(double** *value***, int** *ndigit***, char *****buf***);**

See **ecvt( )**. Converts *value* to a null-terminated string in the array pointed to by *buf* and returns *buf*. Produces *ndigit* significant digits in FORTRAN F-format if possible, otherwise E-format. Any minus sign or decimal point will be included as part of the string. Trailing zeros are suppressed.

Reference: DCC.

## getc

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

Returns the next character (i.e. byte) from the named input *stream*. Moves the file pointer, if defined, ahead one character in *stream*.

OS calls: **isatty**, **read**, **sbrk**, **write**.

Reference: ANSI.

## getchar

**#include <stdio.h>**
**int getchar(void);**

Same as **getc**, but defined as **getc(stdin)**.

OS calls: **isatty**, **read**, **sbrk**, **write**.

Reference: ANSI.

## getenv

**#include <stdlib.h>**
**char getenv(char *** *name* **);**

Searches the environment list for a string of the form *name=value*, and returns a pointer to value if present, otherwise a null pointer.

Reference: ANSI, REENT.

## getopt

**#include <stdio.h>**
**int getopt(int** *argc***, char *const *** *argv***, const char *** *optstring* **);**
**extern char *** *optarg* **;**
**extern int** *optind***,** *opterr* **;**

Returns the next option letter in *argv* that matches a letter in *optstring*, and supports all the rules of the command syntax standard. *optarg* is set to point to the start of the option-argument on return from **getopt(** ). **getopt( )** places the *argv* index of the next argument to be processed in *optind*. Error message output may be disabled by setting *opterr* to 0.

OS calls: **write**.

Reference: SVID.

## getpid

**#include <unistd.h>**
**pid_t getpid(void);**

Gets process ID.

The D-CC libraries provide an interface to this operating system call. Please see your OS manual for a complete definition.

Reference: POSIX, SYS.

## gets

**#include <stdio.h>**
**char *gets(char **s*);**

Reads characters from **stdin** into the array pointed to by *s*, until a new-line character is read or an EOF is encountered. The new-line character is discarded and the string is terminated with a null character. The user is responsible for allocating enough space for the array *s*.

OS calls: **isatty**, **read**, **sbrk**, **write**.

Reference: ANSI.

## getw

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

Returns the next word (i.e., the next integer) from the named input *stream*, and increments the file pointer, if defined, to point to the next word.

OS calls: **isatty**, **read**, **sbrk**, **write**.

Reference: SVID.

## gmtime

**#include <time.h>**
**struct tm *gmtime(const time_t **timer*);**

Breaks down the calendar time *timer* into sections, expressed as Coordinated Universal Time.

Reference: ANSI.

## hcreate

**#include <search.h>**
**int hcreate(unsigned** *nel***);**

Allocates sufficient space for a hash table. See **hsearch( )**. The hash table must be allocated before **hsearch( )** is used. *nel* is an estimate of the maximum number of entries the table will contain.

OS calls: **sbrk**.

Reference: SVID.

## hdestroy

**#include <search.h>**
**void hdestroy(void);**

Destroys the hash table, and may be followed by another call to **hcreate( )**. See **hsearch( )**.

OS calls: **sbrk**, **write**.

Reference: SVID.

## hsearch

**#include <search.h>**
**ENTRY *hsearch(ENTRY** *item***, ACTION** *action***);**

Hash table search routine which returns a pointer into the hash table, indicating the location where an entry can be found. *item.key* points to a comparison key, and *item.data* points to any other data for that key. *action* is either **ENTER** or **FIND** and indicates the disposition of the entry if it cannot be found in the table. **ENTER** means that *item* should be inserted into the table and **FIND** indicates that no entry should be made.

OS calls: **sbrk**.

Reference: SVID.

## hypot

**#include <math.h>**
**double hypot(double** *x***, double** *y***);**

Returns sqrt(*x* * *x* + *y* * *y*), taking precautions against unwarranted overflows.

Reference: UNIX, MATH, REERR.

## hypotf

**#include <mathf.h>**
**float hypotf(float** *x***, float** *y***);**

Returns sqrt(*x* * *x* + *y* * *y*), taking precautions against unwarranted overflows. This is the single precision version of **hypot( )**.

Reference: DCC, MATH, REERR.

## irand48

**#include <stdlib.h>**
**long irand48(unsigned short** *n***);**

Generates pseudo-random non-negative long integers uniformly distributed over the interval [0, n-1], using the linear congruential algorithm and 48-bit integer arithmetic. Must be initialized using **srand48( )**, **seed48( )**, or **lcong48( )** functions.

Reference: UNIX.

## isalnum

**#include <ctype.h>**
**int isalnum(int** *c***);**

Tests for any letter or digit. Returns non-zero if test is true.

Reference: ANSI, REENT.

## isalpha

**#include <ctype.h>**
**int isalpha(int** *c***);**

Tests for any letter. Returns non-zero if test is true.

Reference: ANSI, REENT.

## isascii

**#include <ctype.h>**
**int isascii(int** *c***);**

Tests for ASCII character, code between 0 and 0x7f. Returns non-zero if test is true.

Reference: SVID, REENT.

## isatty

**#include <unistd.h>**
**int isatty(int** *fildes***);**

Tests for a terminal device. Returns non-zero if *fildes* is associated with a terminal device.

Although not a system call in the UNIX environment, it needs to be implemented as such in an embedded environment using the **stdio** functions.

Reference: POSIX.

## iscntrl

**#include <ctype.h>**
**int iscntrl(int** *c***);**

Tests for control character (0x7f or less than 0x20). Returns non-zero if test is true.

Reference: ANSI, REENT.

## isdigit

**#include <ctype.h>**
**int isdigit(int** *c***);**

Tests for digit [0-9]. Returns non-zero if test is true.

Reference: ANSI, REENT.

## isgraph

**#include <ctype.h>**
**int isgraph(int *c*);**

Tests for printable character not including space. Returns non-zero if test is true.

Reference: ANSI, REENT.

## islower

**#include <ctype.h>**
**int islower(int *c*);**

Tests for lower case letter. Returns non-zero if test is true.

Reference: ANSI, REENT.

## _isnan

**#include <math.h>**
**double _isnan(double *x*);**

Returns a non-zero value if *x* is a NaN, and returns 0 otherwise.

Reference: ANSI 754, MATH, REENT

## isprint

**#include <ctype.h>**
**int isprint(int *c*);**

Tests for printable character (including space). Returns non-zero if test is true.

Reference: ANSI, REENT.

## ispunct

**#include <ctype.h>**
**int ispunct(int *c*);**

Tests for printable punctuation character. Returns non-zero if test is true.

Reference: ANSI, REENT.

## isspace

**#include <ctype.h>**
**int isspace(int** *c***);**

Tests for space, tab, carriage return, new-line, vertical tab, or form-feed. Returns non-zero if test is true.

Reference: ANSI, REENT.

## isupper

**#include <ctype.h>**
**int isupper(int** *c***);**

Tests for upper-case letters. Returns non-zero if test is true.

Reference: ANSI, REENT.

## isxdigit

**#include <ctype.h>**
**int isxdigit(int** *c***);**

Tests for hexadecimal digit (0-9, a-f, A-F). Returns non-zero if test is true.

Reference: ANSI, REENT.

## j0

**#include <math.h>**
**double j0(double** *x***);**

Returns the Bessel function of *x* of the first kind of order 0.

OS calls: **write**.

Reference: UNIX, MATH, REERR.

## j0f

**#include <mathf.h>**
**float j0f(float** *x***);**

Returns the Bessel function of *x* of the first kind of order 0. This is the single precision version of **j0( )**.

OS calls: **write**.

Reference: DCC, MATH, REERR.

## j1

**#include <math.h>**
**double j1(double** *x***);**

Returns the Bessel function of *x* of the first kind of order 1.

OS calls: **write**.

Reference: UNIX, MATH, REERR.

## j1f

**#include <mathf.h>**
**float j1f(float** *x***);**

Returns the Bessel function of *x* of the first kind of order 1. This is the single precision version of **j1( )**.

OS calls: **write**.

Reference: DCC, MATH, REERR.

## jn

**#include <math.h>**
**double jn(double** *n***, double** *x***);**

Returns the Bessel function of *x* of the first kind of order *n*.

OS calls: **write**.

Reference: UNIX, MATH, REERR.

## jnf

**#include <mathf.h>**
**float jnf(float** *n***, float** *x***);**

Returns the Bessel function of *x* of the first kind of order *n*. This is the single precision version of **jn( )**.

OS calls: **write**.

Reference: DCC, MATH, REERR.

## jrand48

**#include <stdlib.h>**
**long jrand48(unsigned short** *xsubi***[3]);**

Generates pseudo-random non-negative long integers uniformly distributed over the interval $[-2^{31}, 2^{31}-1]$, using the linear congruential algorithm and 48-bit integer arithmetic. The calling program must place the initial value Xi into the *xsubi* array and pass it as an argument.

Reference: SVID.

## kill

**#include <signal.h>**
**int kill(int** *pid***, int** *sig***);**

Sends the signal *sig* to the process *pid*.

The D-CC libraries provide an interface to this operating system call. Please see your OS manual for a complete definition.

Reference: POSIX, SYS.

## krand48

**#include <stdlib.h>**
**long krand48(unsigned short** *xsubi***[3], unsigned short** *n***);**

Generates pseudo-random non-negative long integers uniformly distributed over the interval [0, n-1], using the linear congruential algorithm and 48-bit integer arithmetic.

Reference: UNIX.

## l3tol

**#include <dcc.h>**
**void l3tol(long ****lp***, char ****cp***, int** *n***);**

Converts the list of *n* three-byte integers packed into the character string pointed to by *cp* into a list of long integers pointed to by **lp*.

Reference: UNIX, REENT.

## l64a

**#include <stdlib.h>**
**char *l64a(long** *l***);**

Converts the long integer *l* to a base-64 character string.

Reference: SVID.

## labs

**#include <stdlib.h>**
**long labs(long** *i***);**

Returns the absolute value of *i*.

Reference: ANSI, REENT.

## lcong48

**#include <stdlib.h>**
**void lcong48(unsigned short** *param***[7]);**

Initialization entry point for **drand48( )**, **lrand48( )**, and **mrand48( )**. Allows the user to specify parameters in the random equation: Xi is *param*[0-2], multiplier a is *param*[3-5], and addend c is *param*[6].

Reference: UNIX.

## ldexp

**#include <math.h>**
**double ldexp(double** *value***, int** *exp***);**

Returns the quantity: *value* * ($2^{exp}$). See also **frexp( )**.

Reference: UNIX, REERR.

## ldexpf

**#include <mathf.h>**
**float ldexpf(float** *value***, int** *exp***);**

Returns the quantity: *value* * ($2^{exp}$). See also **frexpf( )**. This is the single precision version of **ldexp( )**.

Reference: DCC, MATH, REERR.

## ldiv

**#include <stdlib.h>**
**ldiv_t ldiv(long int** *numer***, long int** *denom***);**

Similar to **div( )**, except that arguments and returned items all have the type **long int**.

Reference: ANSI, REENT.

## _lessgreater

**#include <math.h>**
**double _lessgreater(double** *x***, double** *y***);**

The value of $x <> y$ is non-zero only when $x < y$ or $x > y$, and is distinct from NOT($x = y$) per Table 4 of the ANSI 754 standard.

Reference: ANSI 754, MATH, REENT.

## lfind

**#include <stdio.h>**
**#include <search.h>**
**void *lfind(const void *** *key***, const void *** *base***, unsigned *** *nelp***, int** *size***,**
**int (*** *compar***)( ));**

Same as **lsearch( )** except that if datum is not found, it is not added to the table. Instead, a null pointer is returned.

Reference: UNIX, REENT.

## link

**#include <unistd.h>**
**int link(const char **path1*, const char **path2*);**

Creates a new link *path2* to the existing file *path1*.

The D-CC libraries provide an interface to this operating system call. Please see your OS manual for a complete definition.

Reference: SYS.

## localeconv

**#include <locale.h>**
**struct lconv *localeconv(void);**

Loads the components of an object of the type **struct lconv** with values appropriate for the formatting of numeric quantities (monetary and otherwise) according to the rules of the current locale. See also **setlocale( )**.

Reference: ANSI.

## localtime

**#include <time.h>**
**struct tm *localtime(const time_t **timer*);**

Breaks down the calendar time *timer* into sections, expressed as local time.

Reference: ANSI.

## log

**#include <math.h>**
**double log(double** *x*);

Returns the natural logarithm of a positive *x*.

OS calls: **write**.

Reference: ANSI, MATH, REERR.

## _logb

**#include <math.h>**
**double _logb(double** *x***);**

Returns the unbiased exponent of *x*, a signed integer in the format of *x*, except that **logb(NaN)** is NaN, **logb(infinity)** is +∞, and **logb(0)** is -∞ and signals the division by zero exception. When *x* is positive and finite the expression **scalb(***x***, -logb(***x***))** lies strictly between 0 and 2; it is less than 1 only when *x* is denormalized.

Reference: ANSI 754, MATH, REENT.

## logf

**#include <mathf.h>**
**float logf(float** *x***);**

Returns the natural logarithm of a positive *x*. This is the single precision version of **log( )**.

OS calls: **write**.

Reference: DCC, MATH, REERR.

## log10

**#include <math.h>**
**double log10(double** *x***);**

Returns the logarithm with base ten of a positive *x*.

OS calls: **write**.

Reference: ANSI, MATH, REERR.

## log10f

**#include <mathf.h>**
**float log10f(float** *x***);**

Returns the logarithm with base ten of a positive *x*. This is the single precision version of **log10( )**.

OS calls: **write**.

Reference: DCC, MATH, REERR.

## longjmp

**#include <setjmp.h>**
**void longjmp(jmp_buf** *env***, int** *val***);**

Restores the environment saved in *env* by a corresponding **setjmp( )** function call. Execution will continue as if the **setjmp( )** had just returned with the value *val*. If *val* is 0 it will be set to 1 to avoid conflict with the return value from **setjmp( )**.

Reference: ANSI, REENT.

## lrand48

**#include <stdlib.h>**
**long lrand48(void);**

Generates pseudo-random non-negative long integers uniformly distributed over the interval [0, $2^{31}$-1], using the linear congruential algorithm and 48-bit integer arithmetic. Must be initialized using **srand48( )**, **seed48( )**, or **lcong48( )** functions.

Reference: SVID.

## lsearch

**#include <stdio.h>**
**#include <search.h>**
**void *lsearch(const void *** *key***, const void *** *base***, unsigned *** *nelp***, int size,**
**int (*** *compar***)( ));**

Linear search routine which returns a pointer into a table indicating where a datum may be found. If the datum is not found, it is added to the end of the table. *base* points to the first element in the table. *nelp* points to an integer containing the number of elements in the table. *compar* is a pointer to the comparison function which the user must supply (for example, **strcmp( )**).

Reference: SVID, REENT.

## lseek

**#include <unistd.h>**
**off_t lseek(int** *fildes***, off_t** *offset***, int** *whence***);**

Moves the file pointer for the file *fildes* to the file offset *offset*. *whence* has one of the following values:

| | |
|---|---|
| **SEEK_SET** | offset is absolute position from beginning of file |
| **SEEK_CUR** | offset is relative distance from current position |
| **SEEK_END** | offset is relative distance from the end of the file |

The D-CC libraries provide an interface to this operating system call. Please see your OS manual for a complete definition.

Reference: SYS.

## ltol3

**#include <dcc.h>**
**void ltol3(char ****cp***, long ****lp***, int** *n***);**

Converts a list of long integers to three-byte integers. It is the inverse of **l3tol( )**.

Reference: UNIX, REENT.

## mallinfo

**#include <malloc.h>**
**struct mallinfo mallinfo(void)**

Used to determine the best setting of **malloc( )** parameters for an application. Must not be called until after **malloc( )** has been called.

Reference: SVID.

## malloc

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

Allocates space for an object of size *size*. Returns a pointer to the start (lowest byte address) of the object. Returns a null pointer if no more memory can be obtained by the OS.

The first time **malloc( )** is called, it checks the following environment variables:

| | |
|---|---|
| `DMALLOC_INIT=`*n* | If set, **malloc( )** initializes allocated memory with the byte value *n*. This is useful when debugging programs that may depend on **malloc( )** areas always being set to zero. |

| | |
|---|---|
| `DMALLOC_CHECK` | If set, **malloc( )** and **free( )** check the free-list every time they are called. This is useful when debugging programs that may trash the free-list. |

OS calls: **sbrk**.

Reference: ANSI.

## mallopt

**#include <malloc.h>**
**int mallopt(int** *cmd***, int** *value***);**

Used to allocate small blocks of memory quickly by allocating a large group of small blocks at one time. This function exists in order to be compatible to SVID, but its use is not recommended, since the **malloc( )** function is already optimized to be fast.

Reference: SVID.

## matherr

**#include <math.h>**
**int matherr(struct exception ****x***);**

Invoked by math library routines when errors are detected. Users may define their own procedure for handling errors, by including a function named **matherr( )** in their programs. The function **matherr( )** must be of the form described above. When an error occurs, a pointer to the exception structure *x* will be passed to the user-supplied **matherr( )** function. This structure, which is defined by the <`math.h`> header file, includes the following members:

```
int      type;
char     *name;
double   arg1, arg2, retval;
```

The member `type` is an integer describing the type of error that has occurred from the following list defined by the <`math.h`> header file:

| | |
|---|---|
| **DOMAIN** | argument domain error |
| **SING** | argument singularity |
| **OVERFLOW** | overflow range error |
| **UNDERFLOW** | underflow range error |

| | |
|---|---|
| **TLOSS** | total loss of significance |
| **PLOSS** | partial loss of significance |

The member `name` points to a string containing the name of the routine that incurred the error. The members `arg1` and `arg2` are the first and second arguments with which the routine was invoked.

The member `retval` is set to the default value that will be returned by the routine unless the user's **matherr( )** function sets it to a different value.

If the user's **matherr( )** function returns non-zero, no error message will be printed, and **errno** will not be set.

If the function **matherr( )** is not supplied by the user, the default error-handling procedures, described with the math library routines involved, will be invoked upon error. **errno** is set to **EDOM** or **ERANGE** and the program continues.

Reference: SVID, MATH.

## matherrf

**#include <mathf.h>**
**int matherrf(struct exceptionf *_x_);**

This is the single precision version of **matherr( )**.

Reference: DCC, MATH.

## mblen

**#include <stdlib.h>**
**int mblen(const char *_s_, size_t _n_);**

If s is not a null pointer, the function returns the number of bytes in the string s that constitute the next multi-byte character, or -1 if the next *n* (or the remaining bytes) do not compromise a valid multi-byte character. A terminating null character is not included in the character count. If *s* is a null pointer and the multi-byte characters have a state-dependent encoding in current locale, the function returns nonzero; otherwise, it returns zero.

Reference: ANSI, REENT.

## mbstowcs

**#include <stdlib.h>**
**size_t mbstowcs(wchar_t *** *pwc*, **const char *** *s*, **size_t** *n*);

Stores a wide character string in the array whose first element has the address *pwc*, by converting the multi-byte characters in the string *s*. It converts as if by calling **mbtowc( )**. It stores at most *n* wide characters, stopping after it stores a null wide character. It returns the number of wide characters stored, not counting the null character.

Reference: ANSI, REENT.

## mbtowc

**#include <stdlib.h>**
**int mbtowc(wchar_t *** *pwc*, **const char *** *s*, **size_t** *n*);

If *s* is not a null pointer, the function returns the number of bytes in the string *s* that constitute the next multi-byte character. (The number of bytes cannot be greater than **MB_CUR_MAX**). If *pwc* is not a null pointer, the next multi-byte character is converted to the corresponding wide character value and stored in **pwc*. The function returns -1 if the next *n* or the remaining bytes do not constitute a valid multi-byte character. If *s* is a null pointer and multi-byte characters have a state-dependent encoding in current locale, the function stores an initial shift state in its internal static duration data object and returns non-zero; otherwise it returns zero.

Reference: ANSI, REENT.

## memccpy

**#include <string.h>**
**void *memccpy(void *** *s1*, **const void *** *s2*, **int** *c*, **size_t** *n*);

Copies characters from *s2* into *s1*, stopping after the first occurrence of character *c* has been copied, or after *n* characters, whichever comes first.

Reference: SVID, REENT.

## memchr

**#include <string.h>**
**void *memchr(const void *s, int c, size_t n);**

Locates the first occurrence of *c* (converted to unsigned char) in the initial *n* characters of the object pointed to by *s*. Returns a null pointer if *c* is not found.

Reference: ANSI, REENT.

## memcmp

**#include <string.h>**
**int memcmp(const void *s1, const void *s2, size_t n);**

Compares the first *n* character of *s1* to the first *n* characters of *s2*. Returns an integer greater than, equal to, or less than zero according to the relationship between *s1* and *s2*.

Reference: ANSI, REENT.

## memcpy

**#include <string.h>**
**void *memcpy(void *s1, const void *s2, size_t n);**

Copies *n* character from the object pointed to by *s2* into the object pointed to by *s1*. The behavior is undefined if the objects overlap. Returns the value of *s1*.

Reference: ANSI, REENT.

## memmove

**#include <string.h>**
**void *memmove(void *s1, const void *s2, size_t n);**

Copies *n* characters from the object pointed by *s2* into the object pointed to by *s1*. It can handle overlapping while copying takes place as if the *n* characters were first copied to a temporary array, then copied into *s1*. Returns the value of *s1*.

Reference: ANSI, REENT.

## memset

**#include <string.h>**
**void *memset(void *_s_, int _c_, size_t _n_);**

Copies the value of *c* into each of the first *n* characters of the object pointed to by *s*. Returns the value of *s*.

Reference: ANSI, REENT.

## mktemp

**#include <stdio.h>**
**char *mktemp (char *_template_);**

Replaces the contents of the string pointed to by *template* with a unique file name, and returns the address of *template*. The *template* string should look like a filename with six trailing **X**s, which will be replaced with a letter and the current process ID.

OS calls: **access**, **getpid**.

Reference: SVID.

## mktime

**#include <time.h>**
**time_t mktime(struct tm *_timeptr_);**

Converts the local time stored in *timeptr* into a calendar time with the same encoding as values returned by the **time( )** function, but with all values within their normal ranges. It sets the structure members `tm_mday`, `tm_wday`, `tm_yday`.

Reference: ANSI, REENT.

## modf

**#include <math.h>**
**double modf(double** *value*, **double ***_iptr_**);**

Returns the fractional part of *value* and stores the integral part in the location pointed to by *iptr*. Both the fractional and integer parts have the same sign as *value*. See also **frexp( )**.

Reference: ANSI, REENT.

## modff

**#include <mathf.h>**
**float modff(float** *value***, float ****iptr***);**

Returns the fractional part of *value* and stores the integral part in the location pointed to by *iptr*. Both the fractional and integer parts have the same sign as *value*. See also **frexpf( )**. This is the single precision version of **modf( )**.

Reference: DCC, MATH, REENT.

## mrand48

**#include <stdlib.h>**
**long mrand48(void);**

Generates pseudo-random non-negative long integers uniformly distributed over the interval $[-2^{31}, 2^{31}-1]$, using the linear congruential algorithm and 48-bit integer arithmetic. Must be initialized using **srand48( )**, **seed48( )**, or **lcong48( )** functions.

Reference: SVID.

## _nextafter

**#include <math.h>**
**double _nextafter(double** *x***, double** *y***);**

Returns the next representable neighbor of *x* in the direction toward *y*. The following special cases arise: if $x = y$, then the result is *x* without any exception being signaled; otherwise, if either *x* or *y* is a quiet NaN, then the result is one or the other of the input NaNs. Overflow is signaled when *x* is finite but **_nextafter(***x***,** *y***)** lies strictly between $+2^{Emin}$ and $-2^{Emin}$. In both cases, inexact is signaled.

Reference: ANSI 754, MATH, REENT.

## nrand48

**#include <stdlib.h>**
**long nrand48(unsigned short xsubi[3]);**

Generates pseudo-random non-negative long integers uniformly distributed over the interval $[0, 2^{31}-1]$, using the linear congruential algorithm and 48-bit integer arithmetic.

Reference: SVID.

## offsetof

**#include <stddef.h>**
**size_t offsetof(***type***,** *member***);**

Returns the offset of the member *member* in the structure *type*. Implemented as a macro.

Reference: ANSI, REENT.

## open

**#include <fcntl.h>**
**int open(const char ****path***, int** *oflag***, int** *mode***);**

Opens the file *path* for reading or writing according to *oflag*. Usual values of *oflag* are:

| | |
|---|---|
| **O_RDONLY** | open for reading only |
| **O_WRONLY** | open for writing only |
| **O_RDWR** | open for reading and writing |

The D-CC libraries provide an interface to this operating system call. Please see your OS manual for a complete definition.

Reference: POSIX, SYS.

## perror

**#include <stdio.h>**
**void perror(const char ***s**);**

**extern int errno;**
**extern char *sys_errlist[];**
**extern int sys_nerr;**

Produces a message on the standard error output describing the last error encountered during a call to a system or library function. The array of message strings **sys_errlist[]** may be indexed by **errno** to access the message string directly without the new-line. **sys_nerr** is the number of messages in the table. See **strerror( )**.

OS calls: **write**.

Reference: ANSI.

## pow

**#include <math.h>**
**double pow(double** *x***, double** *y***);**

Returns the value of $x^y$. If *x* is zero, *y* must be positive. If *x* is negative, *y* must be an integer.

OS calls: **write**.

Reference: ANSI, MATH, REERR.

## powf

**#include <mathf.h>**
**float powf(float** *x***, float** *y***);**

Returns the value of $x^y$. If *x* is zero, *y* must be positive. If *x* is negative, *y* must be an integer. This is the single precision version of **pow( )**.

OS calls: **write**.

Reference: DCC, MATH, REERR.

## printf

**#include <stdio.h>**
**int printf(const char ***_format_**, ... );**

Places output arguments on **stdout**, controlled by *format*. Returns the number of characters transmitted or a negative value if there was an error. A summary of the **printf( )** conversion specifiers is shown below. Each conversion specification is introduced by the character **%**. Conversion specifications within brackets are optional.

**%** {*flags*} {*field_width*} {**.***precision*} {*length_modifier*} *conversion*

| | |
|---|---|
| *flags* | Single characters which modify the operation of the format as follows: |
| - | left adjusted field |
| + | signed values will always begin with plus or minus sign |
| space | values will always begin with minus or space |

| | |
|---|---|
| **#** | Alternate form. Has the following effect: For **o** (octal) conversion, the first digit will always be a zero **G**, **g**, **E**, **e** and **f** conversions will always print a decimal point. **G** and **g** conversions will also keep trailing zeros. **X**, **x** (hex) and **p** conversions will prepend non-zero values with **0x** (or **0X**) |
| **0** | zero padding to field width (for **d**, **i**, **o**, **u**, **x**, **X**, **e**, **E**, **f**, **g**, and **G** conversions) |
| *field_width* | Number of characters to be printed in the field. Field width will be padded with space if needed. If given as '*****', the next argument should be an integer holding the field width. |
| *.precision* | Minimum number of digits to print for integers (**d**, **i**, **o**, **u**, **x** and **X**). Number of decimals printed for floating point values (**e**, **E**, and **f**). Maximum number of significant digits for **g** and **G** conversions. Maximum number of characters for **s** conversion. If given as '*****' the next argument should be an integer holding the precision. |
| *length_modifier* | The following length modifiers are used: |
| **h** | Used before **d**, **i**, **o**, **n**, **u**, **x**, or **X** conversions to denote a **short int** or **unsigned short int** value. |
| **l** | Used before **d**, **i**, **o**, **n**. **u**, **x**, or **X** conversions to denote a **long int** or **unsigned long int** value. |
| **L** | Used before **e**, **E**, **f**, **g** or **G** conversions to denote a **long double** value. |
| *conversion* | The following conversion specifiers are used: |
| **d** | Write signed decimal integer value. |
| **i** | Write signed decimal integer value. |
| **o** | Write unsigned octal integer value |
| **u** | Write unsigned decimal integer value |
| **x** | Write unsigned hexadecimal (0-9, abc...) integer value |
| **X** | Write unsigned hexadecimal (0-9, ABC...) integer value. |
| **e** | Write floating point value: [-]d.ddde+dd . |
| **E** | Write floating point value: [-]d.dddE+dd . |
| **f** | Write floating point value: [-]ddd.ddd . |

| | |
|---|---|
| **g** | Write floating point value in **f** or **e** notation depending on the size of the value (“best” fit conversion). |
| **G** | Write floating point value in **f** or **E** notation depending on the size of the value (“best” fit conversion). |
| **c** | Write a single character. |
| **s** | Write a string. |
| **p** | Write a pointer value (address). |
| **n** | Store current number of characters written so far. The argument should be a pointer to integer. |
| **%** | Write a percentage character. |

The floating point values Infinity and Not-A-Number are printed as `inf`, `INF`, `nan`, and `NAN` when using the **e**, **E**, **f**, **g**, or **G** conversions.

---

➤ By default in most environments, **printf** buffers its output until a newline is output. To cause output character-by-character without waiting for a newline, call **setbuf** (page 74) with a NULL buffer pointer after opening but before writing to the stream:

```
setbuf(*stream, 0);
```

---

OS calls: **isatty**, **sbrk**, **write**.

Reference: ANSI.

## putc

**#include <stdio.h>**
**int putc(int** *c***, FILE ****stream***)**

Writes the character *c* onto the output *stream* at the position where the file pointer, if defined, is pointing.

OS calls: **isatty**, **sbrk**, **write**.

Reference: ANSI.

## putchar

**#include <stdio.h>**
**int putchar(int** *c***)**

Similar to **putc( )** but writes to **stdout**.

OS calls: **isatty**, **sbrk**, **write**.

Reference: ANSI.

## putenv

**#include <stdlib.h>**
**int putenv(char ***_string_**);**

*string* points to a string of the form *name=value*, and **putenv( )** makes the value of the environmental variable *name* equal to *value*. The string pointed to by *string* becomes part of the environment, so altering *string* alters the environment.

OS calls: **sbrk**, **write**.

Reference: SVID.

## puts

**#include <stdio.h>**
**int puts(const char ***_s_**);**

Writes the null-terminated string pointed to by *s*, followed by a new-line character, to **stdout**.

OS calls: **isatty**, **sbrk**, **write**.

Reference: ANSI.

## putw

**#include <stdio.h>**
**int putw(int** *w***, FILE ***_stream_**)**

Writes the word (i.e., integer) *w* to the output *stream* at the position at which the file pointer, if defined, is pointing.

OS calls: **isatty**, **sbrk**, **write**.

Reference: SVID.

## qsort

**#include <stdlib.h>**
**void qsort(void *** *base***, size_t** *nel***, size_t** *size***, int (*compar)( ));**

Sorts a table in place using the quick-sort algorithm. *base* points to the element at the base of the table, *nel* is the number of elements. *size* is the size of each element. *compar* is a pointer to the user supplied comparison function, which is called with two arguments that point to the elements being compared.

Reference: ANSI, REENT.

## raise

**#include <signal.h>**
**int raise(int** *sig***);**

Sends the signal *sig* to the executing program.

OS calls: **getpid**, **kill**.

Reference: ANSI.

## rand

**#include <stdlib.h>**
**int rand(void);**

Returns a pseudo random number in the interval [0, **RAND_MAX**].

Reference: ANSI.

## read

**#include <unistd.h>**
**int read(int** *fildes***, void *** *buf***, unsigned** *nbyte***);**

Reads max *nbyte* bytes from the file associated with the file descriptor *fildes* to the buffer pointed to by *buf*.

The D-CC libraries provide an interface to this operating system call. Please see your OS manual for a complete definition.

Reference: SYS.

## realloc

**#include <stdlib.h>**
**void *realloc(void *_ptr_, size_t _size_);**
**extern int _ _no_malloc_warning;**

Changes the size of the object pointed to by *ptr* to the size *size*. *ptr* must have received its value from **malloc( )**, **calloc( )**, or **realloc( )**. Returns a pointer to the start address of the possibly moved object, or a null pointer if no more memory can be obtained from the OS.

If the pointer *ptr* was freed or not allocated by **malloc( )**, a warning is printed on the **stderr** stream. The warning can be suppressed by assigning a non-zero value to the integer variable **__no_malloc_warning**. See **malloc( )** for more information.

OS calls: **sbrk**, **write**.

Reference: ANSI.

## remove

**#include <stdio.h>**
**int remove(const char *_filename_);**

Removes the file *filename*. Once removed, the file cannot be opened as an existing file.

OS calls: **unlink**.

Reference: ANSI.

## rename

**#include <stdio.h>**
**int rename(const char *_old_, const char *_new_);**

Renames the file *old* to the file *new*. Once renamed, the file *old* cannot be opened again.

OS calls: **link**, **unlink**.

Reference: ANSI.

## rewind

**#include <stdio.h>**
**void rewind(FILE *_stream_);**

Same as **fseek(***stream***, 0L, 0)**, except that no value is returned.

OS calls: **isatty**, **read**, **sbrk**, **write**.

Reference: ANSI.

## sbrk

**#include <unistd.h>**
**void *sbrk(int** *incr***);**

Gets *incr* bytes of memory from the operating system.

The D-CC libraries provide an interface to this operating system call. Please see your OS manual for a complete definition.

Reference: UNIX, SYS.

## _scalb

**#include <math.h>**
**double _scalb(double** *x***, int** *N***);**

Returns $y * 2^N$ for integeral values *N* without computing $2^N$.

Reference: ANSI 754, MATH, REENT.

## scanf

**#include <stdio.h>**
**int scanf(const char ***format***, ...);**

Reads formatted data from **stdin** and optionally assigns converted data to variables specified by the *format* string. Returns the number of successful conversions (or **EOF** if input is exhausted).

If the format string contains white-space characters, input is scanned until a non-white-space character is found.

A conversion specification is introduced by the character **%**.

If the format string neither contains a white-space nor a **%**, the format string and the input characters must match exactly.

A summary of the **scanf( )** conversion specifiers is shown below. Conversion specifications within braces are optional.

**%** {*} {*field_width*} {*length_modifier*} *conversion*

| | |
|---|---|
| ***** | No assignment should be done (just scan the field). |
| *field_width* | Maximum field to be scanned (default is until no match occurs). |
| *length_modifier* | The following length modifiers are used: |
| **l** | Used before **d**, **i**, or **n** to indicate **long int** or before **o**, **u**, **x** to denote the presence of an **unsigned long int**. For **e**, **E**, **g**, **G**, and **f** conversions the l character implies a **double** operand. |
| **h** | Used before **d**, **i**, or **n** to indicate **short int** or before **o**, **u**, or **x** to denote the presence of an **unsigned short int**. |
| **L** | For **e**, **E**, **g**, **G**, and **f** conversions the **L** character implies a **long double** operand. |
| *conversion* | The following conversions are available: |
| **d** | Read an optionally signed decimal integer value. |
| **i** | Read an optionally signed integer value in standard C notation. Default is decimal notation, but octal (0n) and hex (0xn, 0Xn) notations are also recognized. |
| **o** | Read an optionally signed octal integer. |
| **u** | Read an unsigned decimal integer. |
| **x**, **X** | Read an optionally signed hexadecimal integer. |
| **f**, **e**, **E**, **g**, **G** | Read a floating point constant. |
| **s** | Read a character string. |
| **c** | Read *field_width* number of characters (1 is default). |
| **n** | Store the number of characters read so far. The argument should be a pointer to an integer. |
| **p** | Read a pointer value (address). |
| **[** | Read characters as long as they match any of the characters that are within the terminating ]. If the first character after [ is a ^, the matching condition is reversed. If the [ is immediately followed by ] or ^], the ] is assumed to belong to the matching sequence, and there must be another terminating character. A range of characters may be represented by first-last, thus [a-f] equals [abcdef]. |
| **%** | Read a **%** character. |

Notes: Except for the **[**, **c**, or **n** specifiers leading white-space characters are skipped. Variables must always be expressed as addresses in order to be assignable by **scanf**.

OS calls: **isatty**, **read**, **sbrk**, **write**.

Reference: ANSI.

## seed48

**#include <stdlib.h>**
**unsigned short *seed48(unsigned short seed16v[3]);**

Initialization entry point for **drand48( )**, **lrand48( )**, and **mrand48( )**.

Reference: SVID.

## setbuf

**#include <stdio.h>**
**void setbuf(FILE *** *stream***, char *** *buf***);**

May be used after the *stream* has been opened but before reading or writing to it. It causes the array pointed to by *buf* to be used instead of an automatically allocated buffer. If *buf* is the null pointer, then input/output will be unbuffered. The constant **BUFSIZ** in `<stdio.h>` defines the required size of *buf*.

OS calls: **isatty**, **sbrk**, **write**.

Reference: ANSI.

## setjmp

**#include <setjmp.h>**
**int setjmp(jmp_buf** *env***);**

Saves the current execution environment in *env* for use by the **longjmp( )** function. Returns 0 when invoked by **setjmp( )** and a non-zero value when returning from a **longjmp( )** call.

Reference: ANSI, REENT.

## setlocale

**#include <locale.h>**
**char *setlocale(int** *category***, const char ***locale***);**

Selects the appropriate portion of the program's locale as specified by the *category* and *locale* arguments. Can be used to change or query the program's entire locale with the category **LC_ALL**; the other values for *category* name only portions of the program's locale. **LC_COLLATE** affects the behavior of the **strcoll( )** and **strxfrm( )** functions. **LC_CTYPE** affects the behavior of the character handling functions and the multi-byte functions. **LC_MONETARY** affects the monetary formatting information returned by the **localeconv( )** function. **LC_NUMERIC** affects the decimal-point character for the formatted input/output functions and the string conversion functions, as well as the non-monetary formatting information returned by the **localeconv( )** function. **LC_TIME** affects the behavior of the **strftime( )** function.

A value of **"C"** for *locale* specifies the minimal environment for C translation; a value of "" for *locale* specifies the implementation-defined native environment. Other implementation-defined strings may be passed as the second argument to **setlocale( )**.

At program start-up, the equivalent of `setlocale(LC_ALL, "C")` is executed.

The D-CC currently only supports the **"C"** locale.

Reference: ANSI.

## setvbuf

**#include <stdio.h>**
**void setvbuf(FILE ***stream***, char ***buf***, int** *type***, size_t** *size***);**

See setbuf( ). *type* determines how the *stream* will be buffered:

| | |
|---|---|
| **_IOFBF** | causes stream to be fully buffered |
| **_IOLBF** | causes stream to be line buffered |
| **_IONBF** | causes stream to be unbuffered |

*size* specifies the size of the buffer to be used; **BUFSIZ** in <stdio.h> is the suggested size.

OS calls: **sbrk**, **write**.

Reference: ANSI.

## signal

**#include <signal.h>**
**void (*signal(int** *sig*, **void (*func)( ))(void);**

Specifies the action on delivery of a signal. When the signal *sig* is delivered, a signal handler specified by *func* is called.

The D-CC libraries provide an interface to this operating system call. Please see your OS manual for a complete definition.

Reference: ANSI, SYS.

## sin

**#include <math.h>**
**double sin(double** *x***);**

Returns the sine of *x* measured in radians. It loses accuracy with a large argument value.

OS calls: **write**.

Reference: ANSI, MATH, REERR.

## sinf

**#include <mathf.h>**
**float sinf(float** *x***);**

Returns the sine of *x* measured in radians. It loses accuracy with a large argument value. This is the single precision version of **sin( )**.

OS calls: **write**.

Reference: DCC, MATH, REERR.

## sinh

**#include <math.h>**
**double sinh(double** *x***);**

Returns the hyperbolic sine of *x* measured in radians. It loses accuracy with a large argument value.

Reference: ANSI, MATH, REERR.

## sinhf

**#include <mathf.h>**
**float sinhf(float** *x***);**

Returns the hyperbolic sine of *x* measured in radians. It loses accuracy with a large argument value. This is the single precision version of **sinh( )**.

Reference: DCC, MATH, REERR.

## sprintf

**#include <stdio.h>**
**int sprintf(char *s, const char *format , ...);**

Places output arguments followed by the null character in consecutive bytes starting at *s; the user must ensure that enough storage is available. See **printf( )**.

Reference: ANSI, REENT.

## sqrt

**#include <math.h>**
**double sqrt(double** *x***);**

Returns the non-negative square root of *x*. The argument must be non-negative.

OS calls: **write**.

Reference: ANSI, MATH, REERR.

## sqrtf

**#include <mathf.h>**
**float sqrtf(float** *x***);**

Returns the non-negative square root of *x*. The argument must be non-negative. This is the single precision version of **sqrt( )**.

OS calls: **write**.

Reference: DCC, MATH, REERR.

## srand

**#include <stdlib.h>**
**void srand(unsigned** *seed***);**

Resets the random-number generator to a random starting point. See **rand( )**.

Reference: ANSI.

## srand48

**#include <stdlib.h>**
**void srand48(long** *seedval***);**

Initialization entry point for **drand48( )**, **lrand48( )**, and **mrand48( )**.

Reference: SVID.

## sscanf

**#include <stdio.h>**
**int sscanf(const char ****s***, const char ****format***, ...);**

Reads formatted data from the character string *s*, optionally assigning converted data to variables specified by the *format* string. It returns the number of successful conversions (or **EOF** if input is exhausted). See **scanf( )**.

Reference: ANSI, REENT.

## step

**#include <regexp.h>**
**int step(char ****string***, char ****expbuf***);**

Does pattern matching given the string *string* and a compiled regular expression *expbuf*. See SVID for more details.

Reference: SVID.

## strcat

**#include <string.h>**
**char *strcat(char *_s1_, const char *_s2_);**

Appends a copy of the string pointed to by *s2* (including a null character) to the end of the string pointed to by *s1*. The initial character of *s2* overwrites the null character at the end of *s1*. The behavior is undefined if the objects overlap.

Reference: ANSI, REENT.

## strchr

**#include <string.h>**
**char *strchr(const char *_s_, int _c_);**

Locates the first occurrence of *c* in the string pointed to by *s*.

Reference: ANSI, REENT.

## strcmp

**#include <string.h>**
**int strcmp(const char *_s1_, const char *_s2_);**

Compares *s1* to *s2*. Returns an integer greater than, equal to, or less than zero according to the relationship between *s1* and *s2*.

Reference: ANSI, REENT.

## strcoll

**#include <string.h>**
**int strcoll(const char *_s1_, const char *_s2_);**

Compares *s1* to *s2*, both interpreted as appropriate to the **LC_COLLATE** category of the current locale. Returns an integer greater than, equal to, or less than zero according to the relationship between *s1* and *s2*.

Reference: ANSI, REENT.

## strcpy

**#include <string.h>**
**char *strcpy(char **s1*, const char **s2*);**

Copies the string pointed to by *s2* (including a terminating null character) into the array pointed to by *s1*. The behavior is undefined if the objects overlap.

Reference: ANSI, REENT.

## strcspn

**#include <string.h>**
**size_t strcspn(const char **s1*, const char **s2*);**

Computes the length of the maximum initial segment of *s1* which consists entirely of characters not from *s2*.

Reference: ANSI, REENT.

## strdup

**#include <string.h>**
**char *strdup(const char **s1*);**

Returns a pointer to a new string which is a duplicate of *s1*.

OS calls: sbrk.

Reference: SVID.

## strerror

**#include <string.h>**
**char *strerror(int** *errnum*);

Maps the error number in *errnum* to an error message string.

Reference: ANSI, REENT.

## strftime

**#include <time.h>**
**size_t strftime(char *** *s*, **size_t** *maxsize*, **const char *** *format*,
**const struct tm *** *timeptr*);

Uses the format *format* and values in the structure *timeptr* to generate formatted text. Generated characters are stored in successive locations in the array pointed to by *s*. It stores a null character in the next location in the array. Each non-**%** character is stored in the array. For each **%** followed by a character, a replacement character sequence is stored as shown below. Examples are in parenthesis.

| | |
|---|---|
| **%a** | abbreviated weekday name (Mon) |
| **%A** | full weekday name (Monday) |
| **%b** | abbreviated month name (Jan) |
| **%B** | full month name (January) |
| **%c** | date and time (Jan 03 07:22:43 1990) |
| **%d** | day of the month (04) |
| **%H** | hour of the 24-hour day (13) |
| **%I** | hour of the 12-hour day (9) |
| **%j** | day of the year, Jan 1 = 001 (322) |
| **%m** | month of the year (11) |
| **%M** | minutes after the hour (43) |
| **%p** | AM/PM indicator (PM) |
| **%S** | seconds after the minute (37) |
| **%U** | Sunday week of the year, from 00 (34) |
| **%w** | weekday number, Sunday = 0 (3) |
| **%W** | Monday week of the year, from 00 (23) |
| **%x** | date (Jan 23 1990) |
| **%X** | time (23:33:45) |
| **%y** | year of the century (90) |
| **%Y** | year (1990) |

| | |
|---|---|
| **%Z** | time zone name (PST) |
| **%%** | percent character (%) |

Reference: ANSI, REENT.

## strlen

**#include <string.h>**
**size_t strlen(const char *_s_);**

Computes the length of the string *s*.

Reference: ANSI, REENT.

## strncat

**#include <string.h>**
**char *strncat(char *_s1_, const char *_s2_, size_t _n_);**

Appends not more than *n* characters from the string pointed to by *s2* to the end of the string pointed to by *s1*. The initial character of *s2* overwrites the null character at the end of *s1*. The behavior is undefined if the objects overlap. A terminating null character is always appended to the result.

Reference: ANSI, REENT.

## strncmp

**#include <string.h>**
**int strncmp(const char *_s1_, const char *_s2_, size_t _n_);**

Compares not more than *n* characters (characters after a null character are ignored) in *s1* to *s2*. Returns an integer greater than, equal to, or less than zero according to the relationship between *s1* and *s2*.

Reference: ANSI, REENT.

## strncpy

**#include <string.h>**
**char *strncpy(char *_s1_, const char *_s2_, size_t _n_);**

Copies not more than *n* characters from the string pointed to by *s2* (including a terminating null character) into the array pointed to by *s1*. The behavior is undefined if the objects overlap. If *s2* is shorter than *n*, null characters are appended.

Reference: ANSI, REENT.

## strpbrk

**#include <string.h>**
**char *strpbrk(const char *_s1_, const char *_s2_);**

Locates the first occurrence of any character from the string pointed to by *s2* within the string pointed to by *s1*.

Reference: ANSI, REENT.

## strrchr

**#include <string.h>**
**char *strrchr(const char *_s_, int _c_);**

Locates the last occurrence of *c* within the string pointed to by *s*.

Reference: ANSI, REENT.

## strspn

**#include <string.h>**
**size_t strspn(const char *_s1_, const char *_s2_);**

Computes the length of the maximum initial segment of *s1* which consists entirely of characters from *s2*.

Reference: ANSI, REENT.

## strstr

**#include <string.h>**
**char *strstr(const char *_s1_, const char *_s2_);**

Locates the first occurrence of the sequence of characters (not including a null character) in the string pointed to by *s2* within the string pointed to by *s1*.

Reference: ANSI, REENT.

## strtod

**#include <stdlib.h>**
**double strtod(const char *_str_, char **_endptr_);**

Returns as a double-precision floating-point number the value represented by the character string pointed to by *str*. The string is scanned to the first unrecognized character. Recognized characters include optional white-space character(s), optional sign, a string of digits optionally containing a decimal point, optional **e** or **E** followed by an optional sign or space, followed by an integer. At return, the pointer at **endptr* is set to the first unrecognized character.

Reference: ANSI, REERR.

## strtok

**#include <string.h>**
**char *strtok(char *_s1_, const char *_s2_);**

Searches string *s1* for address of the first element that equals none of the elements in string *s2*. If the search does not find an element, it stores the address of the terminating null character in the internal static duration data object and returns a null pointer. Otherwise, searches from found address to address of the first element that equals any one of the elements in string *s2*. If it does not find element, it stores address of the terminating null character in the internal static duration data object. Otherwise, it stores a null character in the element whose address was found in second search. Then it stores address of the next element after end in the internal duration data object (so next search starts at that address) and returns address found in initial search.

Reference: ANSI.

## strtol

**#include <stdlib.h>**
**long strtol(const char ****str***, char *****endptr***, int** *base***);**

Returns as a long integer the value represented by the character string pointed to by *str*. The string is scanned to the first character inconsistent with the base. Leading white-space characters are ignored. At return, the pointer at **endptr* is set to the first unrecognized character.

If *base* is positive and less then 37, it is used as the base for conversion. After an optional sign, leading zeros are ignored, and "**0x**" or "**0X**" is ignored if *base* is 16.

If *base* is zero, the string itself determines the base: after an optional leading sign a leading zero indicates octal, a leading "**0x**" or "**0X**" indicates hexadecimal, else decimal conversion is used.

Reference: ANSI, REERR.

## strtoul

**#include <stdlib.h>**
**long strtoul(const** *char* ***, char *****endptr***, int** *base***);**

Returns as an unsigned long integer the value represented by the character string pointed to by *s*. The string is scanned to the first character inconsistent with the base. Leading white-space characters are ignored. This is the same as **strtol( )**, except that it reports a range error only if the value is too large to be represented as the type **unsigned long**.

Reference: ANSI, REERR.

## strxfrm

**#include <string.h>**
**size_t strxfrm(char ****s1***, char ****s2***, size_t** *n***);**

Transforms *s2* and places the result in *s1*. No more than *n* characters are put in *s1*, including the terminating null character. The transformation is such that if **strcmp( )** is applied to the two strings, it returns a value greater than, equal to, or less than zero, corresponding to the result of the **strcoll( )** function applied to the same two original strings. Copying between objects that overlap causes undefined results.

Reference: ANSI, REENT.

## swab

**#include <dcc.h>**
**void swab(const char *_from_, char *_to_, int _nbytes_)**

Copies *nbytes* bytes pointed to by *from* to the array pointed to by *to*. *nbytes* must be even and non-negative. Adjacent even and odd bytes are exchanged.

Reference: SVID, REENT.

## tan

**#include <math.h>**
**double tan(double** *x***);**

Returns the tangent of *x* measured in radians.

OS calls: **write**.

Reference: ANSI, MATH, REERR.

## tanf

**#include <mathf.h>**
**float tanf(float** *x***);**

Returns the tangent of *x* measured in radians. This is the single precision version of **tan( )**.

OS calls: **write**.

Reference: DCC, MATH, REERR.

## tanh

**#include <math.h>**
**double tanh(double** *x***);**

Returns the hyperbolic tangent of *x* measured in radians.

Reference: ANSI, MATH, REENT.

## tanhf

**#include <mathf.h>**
**float tanhf(float** *x***);**

Returns the hyperbolic tangent of *x* measured in radians. This is the single precision version of **tanh( )**.

Reference: DCC, MATH, REENT.

## tdelete

**#include <search.h>**
**void *tdelete(const void** **key***, void ***rootp***, int (****compar***)( ));**

The **tdelete( )** function deletes a node from a binary search tree. The value for *rootp* will be changed if the deleted node was the root of the tree. Returns a pointer to the parent of the deleted node. See **tsearch( )**.

Reference: SVID.

## tell

**#include <dcc.h>**
**long tell(int** *fildes***);**

Returns the current location in the file descriptor *fildes*. This is the same as **lseek(***fildes***,0L,1)**.

OS calls: lseek.

Reference: DCC.

## tempnam

**#include <stdio.h>**
**char *tempnam(const char ***dir***, const char ***pfx***);**

Creates a unique file name, allowing control of the choice of directory. If the `TMPDIR` variable is specified in the user's environment, it is used as the temporary file directory. Otherwise, the argument *dir* points to the name of the directory in which the file is to be created. If *dir* is invalid, the path-prefix **P_tmpdir** (`<stdio.h>`) is used. If **P_tmpdir** is invalid, **/tmp** is used. See **tmpnam( )**.

Reference: SVID.

## tfind

**#include <search.h>**
**void *tfind(void *_key_, void *const *_rootp_, int (*_compar_)( ));**

**tfind( )** will search for a datum in a binary tree, and return a pointer to it if found, otherwise it returns a null pointer. See **tsearch( )**.

Reference: SVID, REENT.

## time

**#include <time.h>**
**time_t time(time_t *_timer_);**

Returns the system time. If *timer* is not a null pointer, the time value is stored in **timer*.

The D-CC libraries provide an interface to this operating system call. Please see your OS manual for a complete definition.

Reference: ANSI, SYS.

## tmpfile

**#include <stdio.h>**
**FILE *tmpfile(void);**

Creates a temporary file using a name generated by **tmpnam( )** and returns the corresponding **FILE** pointer. File is opened for update (**"w+"**), and is automatically deleted when the process using it terminates.

OS calls: **lseek**, **open**, **unlink**.

Reference: ANSI.

## tmpnam

**#include <stdio.h>**
**char *tmpnam(char *_s_);**

Creates a unique file name using the path-prefix defined as **P_tmpdir** in `<stdio.h>`. If *s* is a null pointer, **tmpnam( )** leaves the result in an internal static area and returns a pointer to that area. At the next call to **tmpnam( )**, it will destroy the contents of the area. If s is not a null pointer, it is assumed to be the address of an array of at least **L_tmpnam** bytes (defined in `<stdio.h>`); **tmpnam( )** places the result in that array and returns *s*.

OS calls: **access**, **getpid**.

Reference: ANSI.

## toascii

**#include <ctype.h>**
**int toascii(int** *c***);**

Turns off all bits in the argument *c* that are not part of a standard ASCII character; for compatibility with other systems.

Reference: SVID, REENT.

## tolower

**#include <ctype.h>**
**int tolower(int** *c***);**

Converts an upper-case letter to the corresponding lower-case letter. The argument range is -1 through 255, any other argument is unchanged.

Reference: ANSI, REENT.

## _tolower

**#include <ctype.h>**
**int _tolower(int** *c***);**

Converts an upper-case letter to the corresponding lower-case letter. Arguments outside lower-case letters return undefined results. The speed is somewhat faster than **tolower( )**.

Reference: SVID, REENT.

## toupper

**#include <ctype.h>**
**int toupper(int** *c***);**

Converts a lower-case letter to the corresponding upper-case letter. The argument range is -1 through 255, any other argument is unchanged.

Reference: ANSI, REENT.

## _toupper

**#include <ctype.h>**
**int _toupper(int** *c***);**

Converts a lower-case letter to the corresponding upper-case letter. Arguments outside lower-case letters return undefined results. The speed is somewhat faster than **toupper( )**.

Reference: SVID, REENT.

## tsearch

**#include <search.h>**
**void *tsearch(const void *** *key***, void ** *rootp***, int (*** *compar***)( ));**

Used to build and access a binary tree. The user supplies the routine *compar* to perform comparisons. *key* is a pointer to a datum to be accessed or stored. If a datum equal to **key* is in the tree, a pointer to that datum is returned. Otherwise, **key* is inserted, and a pointer to it is returned. *rootp* points to a variable that points to the root of the tree.

Reference: SVID.

## twalk

**#include <search.h>**
**void twalk(void *** *root***, void (*** *action***)( ));**

**twalk( )** traverses a binary tree. *root* is the root of the tree to be traversed, and any node may be the root for a walk below that node. *action* is the name of the user supplied routine to be invoked at each node, and is called with three arguments. The first argument is the address of the node being visited. The second argument is a value from the enumeration data type **typedef enum {preorder, postorder, endorder, leaf} VISIT** (see `<search.h>`), depending on whether this is the first, second, or third time the node has been visited (during a depth-first, left-to-right traversal of the tree), or whether the node is a leaf. The third argument is the level of the node in the tree, with the root as level zero. See **tsearch( )**.

Reference: SVID, REENT.

## tzset

**#include <sys/types.h>**
**#include <time.h>**
**void tzset(void);**

**tzset( )** uses the contents of the environment variable TZ to override the value of the different external variables for the time zone. It scans the contents of TZ and assigns the different fields to the respective variable. **tzset( )** is called by **asctime( )** and may be called explicitly by the user.

Reference: POSIX.

## ungetc

**#include <stdio.h>**
**int ungetc(int** *c***, FILE ****stream***);**

Inserts character *c* into the buffer associated with input *stream*. The argument *c* will be returned at the next **getc( )** call on that stream. **ungetc( )** returns *c* and leaves the file associated with *stream* unchanged. If *c* equals **EOF**, **ungetc( )** does nothing to the buffer and returns **EOF**. Only one character of push-back is guaranteed.

Reference: ANSI.

## unlink

**#include <unistd.h>**
**int unlink(const char ****path***);**

Removes the directory entry *path*.

The D-CC libraries provide an interface to this operating system call. Please see your OS manual for a complete definition.

Reference: POSIX, SYS.

## _unordered

**#include <math.h>**
**double _unordered(double** *x***, double** *y***);**

Returns a non-zero value if *x* is unordered with *y*, and returns zero otherwise. See Table 4 of the ANSI 754 standard for the meaning of *unordered*.

Reference: ANSI 754, MATH, REENT.

## vfprintf

**#include <stdarg.h>**
**#include <stdio.h>**
**int vfprintf(FILE ****stream***, const char ****format***, va_list** *arg***);**

This is equivalent to **fprintf( )**, but with the argument list replaced by *arg*, which must have been initialized with the **va_start** macro.

---

➤ By default in most environments, **vfprintf** buffers its output until a newline is output. To cause output character-by-character without waiting for a newline, call **setbuf** (page 74) with a NULL buffer pointer before after opening but before writing to the stream:

```
setbuf(*stream, 0);
```

---

OS calls: **isatty**, **sbrk**, **write**.

Reference: ANSI.

## vfscanf

**#include <stdarg.h>**
**#include <stdio.h>**
**int vfscanf(FILE ****stream***, const char ****format***, va_list** *arg***);**

This is equivalent to **fscanf( )**, but with the argument list replaced by *arg*, which must have been initialized with the **va_start** macro.

OS calls: **isatty**, **read**, **sbrk**, **write**.

Reference: DCC.

## vprintf

**#include <stdarg.h>**
**#include <stdio.h>**
**int vprintf(const char ****format***, va_list** *arg***);**

This is equivalent to **printf( )**, but with the argument list replaced by *arg*, which must have been initialized with the **va_start** macro.

➤ By default in most environments, **vprintf** buffers its output until a newline is output. To cause output character-by-character without waiting for a newline, call **setbuf** (page 74) with a NULL buffer pointer before after opening but before writing to the stream:

```
setbuf(*stream, 0);
```

OS calls: **isatty**, **sbrk**, **write**.

Reference: ANSI.

## vscanf

**#include <stdarg.h>**
**#include <stdio.h>**
**int vscanf(const char *_format_, va_list _arg_);**

This is equivalent to **scanf( )**, but with the argument list replaced by *arg*, which must have been initialized with the **va_start** macro.

OS calls: **isatty**, **read**, **sbrk**, **write**.

Reference: DCC.

## vsprintf

**#include <stdarg.h>**
**#include <stdio.h>**
**int vsprintf(char *_s_, const char *_format_, va_list _arg_);**

This is equivalent to **sprintf( )**, but with the argument list replaced by *arg*, which must have been initialized with the **va_start** macro.

OS calls: **isatty**, **sbrk**, **write**.

Reference: ANSI, REENT.

## vsscanf

**#include <stdarg.h>**
**#include <stdio.h>**
**int vsscanf(const char ****s***, const char *****format***, va_list** *arg***);**

This is equivalent to **sscanf( )**, but with the argument list replaced by *arg*, which must have been initialized with the **va_start** macro.

OS calls: **isatty**, **read**, **sbrk**, **write**.

Reference: DCC, REENT.

## wcstombs

**#include <stdlib.h>**
**size_t wcstombs(char ***s***, const wchar_t *****wcs***, size_t** *n***);**

Stores a multi-byte character string in the array whose first element has the address s by converting each of the characters in the string *wcs*. It converts as if calling **wctomb( )**. It stores no more than *n* characters, stopping after it stores a null character. It returns the number of characters stored, not counting the null character; unless there is an error, in which case it returns -1.

Reference: ANSI.

## wctomb

**#include <stdlib.h>**
**int wctomb(char *****s***, wchar_t** *wchar***);**

If s is not a null pointer, the function determines the number of bytes needed to represent the multi-byte character corresponding to the wide character *wchar*. It converts *wchar* to the corresponding multi-byte character and stores it in the array whose first element has the address *s*. It returns the number of bytes required, not counting the terminating null character; unless there is an error, in which case it returns -1.

Reference: ANSI.

## write

**#include <unistd.h>**
**int write(int** *fildes***, const void *****buf***, unsigned** *nbyte***);**

Writes *nbyte* bytes from the buffer *buf* to the file *fildes*.

The D-CC libraries provide an interface to this operating system call. Please see your OS manual for a complete definition.

Reference: POSIX, SYS.

## y0

**#include <math.h>**
**double y0(double** *x***);**

Returns the Bessel function of positive *x* of the second kind of order 0.

OS calls: **write**.

Reference: UNIX, MATH, REERR.

## y0f

**#include <mathf.h>**
**float y0f(float** *x***);**

Returns the Bessel function of positive *x* of the second kind of order 0. This is the single precision version of **y0( )**.

OS calls: **write**.

Reference: DCC, MATH, REERR.

## y1

**#include <math.h>**
**double y1(double** *x***);**

Returns the Bessel function of positive *x* of the second kind of order 1.

OS calls: **write**.

Reference: UNIX, MATH, REERR.

## y1f

**#include <mathf.h>**
**float y1f(float** *x***);**

Returns the Bessel function of positive *x* of the second kind of order 1. This is the single precision version of **y1( )**.

OS calls: **write**.

Reference: DCC, MATH, REERR.

## yn

**#include <math.h>**
**double yn(double** *n***, double** *x***);**

Returns the Bessel function of positive *x* of the second kind of order *n*.

OS calls: **write**.

Reference: UNIX, MATH, REERR.

## ynf

**#include <mathf.h>**
**float ynf(float** *n***, float** *x***);**

Returns the Bessel function of positive *x* of the second kind of order *n*. This is the single precision version of **yn( )**.

OS calls: **write**.

Reference: DCC, MATH, REERR.

# Index

## Symbols



## A



## B



## C

## D



## E



## F

## G



## H



## I



## J

## K



## L



## M

## N



## O



## P



## Q



## R

## S



## T

## U



## V



## W



## Y