D-CC™ & D-C++™
Compiler Suites

C Library Reference Manual

Version 4.2 10/98
Copyright Notice

Copyright 1991-1998, Diab Data, Inc., Foster City, California, USA
All rights reserved. This document may not be copied in whole or in part, or otherwise reproduced, except as specifically permitted under U.S. law, without the prior written consent of Diab Data, Inc.

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

1 Introduction 1
   Document conventions 2
   Library structure 2
   Libraries supplied 3
   Assumptions 5
   Library directory structure 5
   libc.a 7
   Library search paths 8

2 Include Files 13
   Files 13
   Defined Variables, Types, and Constants 15
      errno.h 15
      fcntl.h 16
      float.h 16
      limits.h 16
      math.h 16
      mathf.h 16
      setjmp.h 16
      signal.h 16
      stdarg.h 17
      stddef.h 17
      stdio.h 17
      stdlib.h 17
      string.h 17
      time.h 17

3 Functions 19
   Format of Descriptions 19
      Operating system calls 19
   References 20
   Function Listing 21
difftime 30
div 30
drand48 30
ecvt 31
erf 31
erff 31
erfc 32
erfcf 32
exit 32
__exit 32
exp 33
expf 33
fabs 33
fabsf 33
fclose 34
fcntl 34
fcvt 34
fdopen 34
feof 35
ferror 35
fflush 35
fgetc 35
fgetpos 36
fgets 36
fileno 36
__finite 36
floor 37
floorf 37
fmod 37
fmodf 37
fopen 38
fprintf 38
fputc 39
fputs 39
fread 39
free 40
freopen 40
frexp 40
frexpf 41
fscanf 41
fseek 41
fsetpos 42
fstat 42
ftell 42
fwrite 42
gamma 43
gammaf 43
gcvt 43
getc 44
getchar 44
getenv 44
getopt 44
getpid 45
gets 45
getw 45
gmtime 45
hcreate 46
hdestroy 46
hsearch 46
hypot 47
hypotf 47
irand48 47
isalnum 47
isalpha 47
isascii 48
isatty 48
iscntrl 48
isdigit 48
isgraph 49
islower 49
_isnan 49
isprint 49
ispunct  49
isspace  50
isupper  50
isxdigit  50
j0    50
j0f   51
j1    51
j1f   51
jn    51
jnf   52
jrand48  52
kill   52
krand48  52
l3tol   53
l64a   53
labs   53
lcong48  53
ldexp  53
ldexpf  54
ldexpf  54
ldiv   54
__lessgreater  54
lfind   54
link   55
localeconv  55
localtime  55
log    55
__logb   56
logf   56
log10  56
log10f  56
longjmp  57
lrand48  57
lsearch  57
lseek   57
lto13   58
mallinfo  58
malloc 58
mallopt 59
matherr 59
matherrf 60
mblen 60
mbstowcs 61
mbtowc 61
memccpy 61
memchr 62
memcmp 62
memcpy 62
memmove 62
memset 63
mktemp 63
mktime 63
modf 63
modff 64
mrand48 64
getNextafter 64
nrand48 64
offsetof 65
open 65
perror 65
pow 66
powf 66
printf 66
putc 68
putchar 69
putenv 69
puts 69
putw 69
qsort 70
raise 70
rand 70
read 70
realloc 71
strpbrk 83
strstrchr 83
strspn 83
strstr 84
strtol 84
strtok 84
strtol 85
strtoul 85
strxfrm 85
swab 86
tan 86
tanf 86
tanh 86
tanhf 87
tdelete 87
tell 87
tempnam 87
tfind 88
time 88
tmpfile 88
tmpnam 88
toascii 89
tolower 89
_tolower 89
toupper 89
_toupper 90
tsearch 90
twalk 90
tzset 91
ungetc 91
unlink 91
_unordered 91
vfprintf 92
vfscanf 92
vprintf 92
vscanf 93
List of Tables

Table 1-1 Document conventions 2
Table 1-2 Library files 3
Table 1-3 Library directory locations 6
Table 1-4 libc.a files delivered with the tools 7
Table 1-5 Directories searched for libraries 8
Table 1-6 Examples of libraries found for different -t options 10
Table 2-1 Standard Include Files 14
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>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dcc -o test.c</td>
<td>This font is used for file and program names, environment variables, user input, and program output.</td>
</tr>
<tr>
<td>if, main(), #pragma,</td>
<td>Bold type is used for keywords, operators and other tokens of the language, library routines and entry points, and section names.</td>
</tr>
<tr>
<td><strong>pack</strong></td>
<td>Some names begin or end with underscores. These underscores and special characters such as # shown in bold are required.</td>
</tr>
<tr>
<td>variable, filename</td>
<td>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.</td>
</tr>
<tr>
<td>[ optional text ]</td>
<td>An item enclosed in brackets is optional.</td>
</tr>
<tr>
<td>{ item1</td>
<td>item2 }</td>
</tr>
<tr>
<td>item ...</td>
<td>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.</td>
</tr>
<tr>
<td>item ,...</td>
<td>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).</td>
</tr>
</tbody>
</table>

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*

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>
<thead>
<tr>
<th>File</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>libcfp.a</em></td>
<td>Floating point functions called by user code, including, for example, the <em>printf</em> and <em>scanf</em> 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. Typically included automatically by <em>libc.a</em>, see below.</td>
</tr>
<tr>
<td><em>libchar.a</em></td>
<td>Basic operating system functions using simple character input/output for <em>stdin</em> and <em>stdout</em> only (<em>stderr</em> and named files are not supported). This is an alternative to <em>libram.a</em>. Sometimes included automatically by <em>libc.a</em>, see below.</td>
</tr>
<tr>
<td><em>libcomplex.a</em></td>
<td>C++ complex math class library. Not automatic; include with an <em>-lcomplex</em> option.</td>
</tr>
</tbody>
</table>
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 `-t tof:environ` option on the command line used to invoke the compiler, assembler, or linker.

• The `DENVIRON` configuration variable (set from the `environ` part of `-t tof: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 `-t tof: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):
### Library directory locations

<table>
<thead>
<tr>
<th>Directory / file</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TARGE/</strong></td>
<td>Directories and files for ELF components (final ‘e’ in TARGE).</td>
</tr>
<tr>
<td>libc.a</td>
<td>Text file which includes other ELF libraries as described below — no input/output support.</td>
</tr>
<tr>
<td>libchar.a</td>
<td>ELF basic operating system functions using character input/output for stdin and stdout only (stderr and named files are not supported).</td>
</tr>
<tr>
<td>libi.a</td>
<td>ELF standard ANSI C functions.</td>
</tr>
<tr>
<td>libimpl.a</td>
<td>ELF functions called by compiler-generated or runtime code.</td>
</tr>
<tr>
<td>libd.a</td>
<td>ELF additional C++ standard and support functions.</td>
</tr>
<tr>
<td>libram.a</td>
<td>ELF basic operating system functions using RAM-disk input/output.</td>
</tr>
<tr>
<td>cross/libc.a</td>
<td>ELF libc.a which includes the RAM-disk input/output library libram.a.</td>
</tr>
<tr>
<td>simple/libc.a</td>
<td>ELF libc.a which includes the basic character input/output library libchar.a.</td>
</tr>
<tr>
<td><strong>TARGEN/</strong></td>
<td>ELF floating point floating point support of “None”.</td>
</tr>
<tr>
<td>libcfp.a</td>
<td>Stubs to avoid undefined externals.</td>
</tr>
<tr>
<td>libimpfp.a</td>
<td>Empty file required by different versions of libc.a.</td>
</tr>
<tr>
<td><strong>TARGEN/</strong></td>
<td>ELF hardware floating point libraries:</td>
</tr>
<tr>
<td>libcfp.a</td>
<td>Basic floating point functions</td>
</tr>
<tr>
<td>libcomplex.a</td>
<td>Complex number package (not included automatically)</td>
</tr>
<tr>
<td>libimpfp.a</td>
<td>Conversions between floating point and other types</td>
</tr>
<tr>
<td>libios.a</td>
<td>iostream (not included automatically)</td>
</tr>
<tr>
<td>libm.a</td>
<td>Math library (not included automatically)</td>
</tr>
</tbody>
</table>
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 1-3 Library directory locations (continued)**

<table>
<thead>
<tr>
<th>Directory / file</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>TARGES/</td>
<td>ELF software floating point libraries parallel to TARGEN.</td>
</tr>
<tr>
<td>TARGD</td>
<td>Parallel directories for COFF components (final ‘D’ in TARGD)</td>
</tr>
<tr>
<td>TARGDN/</td>
<td></td>
</tr>
<tr>
<td>TARGDS/</td>
<td></td>
</tr>
</tbody>
</table>

**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**

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

<table>
<thead>
<tr>
<th>‘f’</th>
<th>Directories</th>
<th>Environment</th>
<th>Floating point support</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>version_path/TARGEN/environ</td>
<td>specific</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>version_path/TARGEN</td>
<td>generic</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>version_path/TARGE/environ</td>
<td>specific</td>
<td>not applicable</td>
</tr>
<tr>
<td></td>
<td>version_path/TARGE</td>
<td>generic</td>
<td>not applicable</td>
</tr>
<tr>
<td>H</td>
<td>version_path/TARGEH/environ</td>
<td>specific</td>
<td>Hardware</td>
</tr>
<tr>
<td></td>
<td>version_path/TARGEH</td>
<td>generic</td>
<td>Hardware</td>
</tr>
<tr>
<td></td>
<td>version_path/TARGEH/environ</td>
<td>specific</td>
<td>not applicable</td>
</tr>
<tr>
<td></td>
<td>version_path/TARGE</td>
<td>generic</td>
<td>not applicable</td>
</tr>
</tbody>
</table>
Library structure

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>
<thead>
<tr>
<th>‘f’</th>
<th>Directories</th>
<th>Environment</th>
<th>Floating point support</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>version_path/TARGES/environ</td>
<td>specific</td>
<td>Software</td>
</tr>
<tr>
<td></td>
<td>version_path/TARGES</td>
<td>generic</td>
<td>Software</td>
</tr>
<tr>
<td></td>
<td>version_path/TARGES/environ</td>
<td>specific</td>
<td>not applicable</td>
</tr>
<tr>
<td></td>
<td>version_path/TARGES</td>
<td>generic</td>
<td>not applicable</td>
</tr>
</tbody>
</table>

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.
### Library structure

<table>
<thead>
<tr>
<th>-t option</th>
<th>Libraries found</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>-tTARGEN:simple</td>
<td>TARGE/simple/libc.a&lt;br&gt; TARGE/libi.a&lt;br&gt; TARGEN/libcfp.a&lt;br&gt; TARGE/libchar.a&lt;br&gt; TARGE/libimpl.a&lt;br&gt; TARGEN/libimpfp.a</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• libi.a and libimpl.a are common to all TARGE systems and are found in the fourth directory TARGE.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The floating point support is independent of the environment and comes from the second directory TARGEN.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 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.</td>
</tr>
<tr>
<td>-tTARGES:cross</td>
<td>TARGE/cross/libc.a&lt;br&gt; TARGE/libi.a&lt;br&gt; TARGES/libcfp.a&lt;br&gt; TARGE/libram.a&lt;br&gt; TARGE/libimpl.a&lt;br&gt; TARGES/libimpfp.a</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• libi.a and libimpl.a are in the fourth directory TARGE as before.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The software floating point library libcfp.a is from the second directory, now TARGES.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• This time libram.a has been selected by TARGE/cross/libc.a instead of libcchar.a (but still from the fourth directory TARGE as before).</td>
</tr>
</tbody>
</table>
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:

- li
- lcfp
- lchar
- limpl
- limpfp

Thus, based on the search order implied by the -tTARGET:cust option, the standard libraries TARGE/libi.a, TARGE/libimpl.a, TARGES/libcfp.a, and TARGES/libimpfp.a will be searched.

In addition, the library TARGE/cust/libchar.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.

<table>
<thead>
<tr>
<th>-t option</th>
<th>Libraries found</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>-tTARGET:cust</td>
<td>TARGE/cust/libc.a, TARGE/libi.a, TARGES/libcfp.a, TARGE/cust/libchar.a, TARGE/libimpl.a, TARGES/libimpfp.a</td>
<td>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: -li -lcfp -lchar -limpl -limpfp Thus, based on the search order implied by the -tTARGET:cust option, the standard libraries TARGE/libi.a, TARGE/libimpl.a, TARGES/libcfp.a, and TARGES/libimpfp.a will be searched. In addition, the library TARGE/cust/libchar.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.</td>
</tr>
</tbody>
</table>
2 Include 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

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

<table>
<thead>
<tr>
<th>Standard</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSI</td>
<td>The function/macro is defined in ANSI X3.159-1989.</td>
</tr>
<tr>
<td>ANSI 754</td>
<td>The function is define in ANSI/IEEE Std 754-1985.</td>
</tr>
<tr>
<td>DCC</td>
<td>The function/macro is added to D-CC.</td>
</tr>
<tr>
<td>POSIX</td>
<td>The function/macro is defined in IEEE Std 1003.1-1990.</td>
</tr>
<tr>
<td>SVID</td>
<td>The function/macro is defined in System V Interface Definition 2.</td>
</tr>
<tr>
<td>UNIX</td>
<td>The function/macro is provided to be compatible with Unix V.3.</td>
</tr>
</tbody>
</table>

Other references:

<table>
<thead>
<tr>
<th>Standard</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATH</td>
<td>The math libraries must be specified at link time with the -lm option.</td>
</tr>
<tr>
<td>SYS</td>
<td>The function must be provided by the operating system or emulated in a stand-alone system.</td>
</tr>
<tr>
<td>REENT</td>
<td>The function is reentrant. It does not use any static or global data.</td>
</tr>
<tr>
<td>REERR</td>
<td>The function might modify errno and is reentrant only if all processes ignore that variable</td>
</tr>
</tbody>
</table>

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

```c
#include <math.h>

double acos(double x);
```

Returns the arc cosine of \( x \) in the range \([0, \pi]\). \( x \) must be in the range \([-1, 1]\). Otherwise zero is returned, \texttt{errno} is set to \texttt{EDOM}, and a message indicating a domain error is printed on the standard error output.

OS calls: \texttt{write}.

Reference: ANSI, MATH, REERR.

acosf

```c
#include <mathf.h>

float acosf(float x);
```

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

OS calls: \texttt{write}.

Reference: DCC, MATH, REERR.

advance

```c
#include <regexp.h>

int advance(char *string, char *expbuf);
```

Does pattern matching given the string \texttt{string} and a compiled regular expression in \texttt{expbuf}. See SVID for more details.

Reference: SVID.

alloca

```c
#include <dcc.h>

void *alloca(size_t size)
```

Allocates temporary local stack space for an object of size \texttt{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 \([-\pi/2, \pi/2]\). 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 \([-\pi/2, \pi/2]\). 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

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

Returns the arc tangent of \(x\) in the range \([-\pi/2, \pi/2]\).

OS calls: `write`.

Reference: ANSI, MATH, REERR.

### atanf

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

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

OS calls: `write`.

Reference: DCC, MATH, REERR.

### atan2

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

Returns the arc tangent of \(y/x\) in the range \([-\pi, \pi]\), 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

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

Returns the arc tangent of \(y/x\) in the range \([-\pi, \pi]\), 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

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

Registers the function whose address is `func` to be called by `exit`.

Reference: ANSI.

### atof

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

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

Reference: ANSI, REERR.

### atoi

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

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

Reference: ANSI, REENT.
atol

```c
#include <stdlib.h>
long atol(const char *nptr);
```
Converts an ASCII decimal number string `nptr` into a `long`.

Reference: ANSI, REENT.

bsearch

```c
#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

```c
#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

```c
#include <math.h>
double ceil(double x);
```
Returns the smallest integer not less than `x`.

OS calls: `write`.

Reference: ANSI, MATH, REENT.
**ceilf**

```c
#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**

```c
#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**

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

Resets the error and EOF indicators to zero on the named `stream`.  
Reference: ANSI.

**clock**

```c
#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

```c
#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

```c
#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

```c
#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

```c
#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: _exit, 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

```c
#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

```c
#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

```c
#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

```c
#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**

```c
#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**

```c
#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**

```c
#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**

```c
#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

```c
#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

```c
#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

```c
#include <stdio.h>
int fileno (FILE *stream);
```
Returns the integer file descriptor associated with the named stream; see open();
Reference: POSIX.

_finite

```c
#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=iy+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=iy+f for
some integer i, and absolute value of f is less than absolute value of y. This is the single pre-
cision 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.
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.

cfreopen

#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.

cfrexp

#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.
frexp

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

Given that every non-zero number can be expressed as \( x \times (2^n) \), where \( 0.5 \leq |x| < 1.0 \) 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 \texttt{frexp( )}.

Reference: DCC, MATH, REENT.

fscanf

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

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

OS calls: isatty, read, sbrk, write.

Reference: ANSI.

fseek

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

Sets the position of the next input or output operation on the \texttt{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 \texttt{whence}. The next operation on a file opened for update may be either input or output. whence has one of the following values:

- \texttt{SEEK_SET}  
  offset is absolute position from beginning of file.
- \texttt{SEEK_CUR}  
  offset is relative distance from current position.
- \texttt{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**

```c
#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**

```c
#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**

```c
#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

```c
#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

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

Same as `getc`, but defined as `getc(stdin)`.
OS calls: `isatty`, `read`, `sbrk`, `write`.
Reference: ANSI.

getenv

```c
#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

```c
#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`.
getpid

```c
#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

```c
#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

```c
#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

```c
#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**

```c
#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**

```c
#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**

```c
#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^2 + y^2), taking precautions against unwarranted overflows.

Reference: UNIX, MATH, REERR.

hypotf

#include <mathf.h>

float hypotf(float x, float y);

Returns sqrt(x^2 + y^2), 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 inter-
val [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.
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**

```c
#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.

**isspace**

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

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

Reference: ANSI, REENT.

**isxdigit**

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

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

Reference: ANSI, REENT.

**j0**

```c
#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.
Function Listing

jnf

```c
#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

```c
#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 \( X_i \) into the \( xsubi \) array and pass it as an argument.

Reference: SVID.

kill

```c
#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

```c
#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().
ldexpf

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

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

Reference: DCC, MATH, REERR.

ldiv

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

Similar to \( \text{div}() \), except that arguments and returned items all have the type \( \text{long int} \).

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 \( \text{lsearch}() \) 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 
\( \logb(\text{NaN}) \) is NaN, \( \logb(\text{infinity}) \) is \( +\infty \), and \( \logb(0) \) is \( -\infty \) and signals the division by zero exception. When \( x \) is positive and finite the expression \( \text{scalb}(x, -\logb(x)) \) 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.
Function Listing

**sbrk**
OS calls: `sbrk`.
Reference: ANSI.

**mallopt**

```c
#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**

```c
#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
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

```c
#include <mathf.h>
int matherrf(struct exceptionf *x);
```

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

Reference: DCC, MATH.

### mblen

```c
#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 con-
verting 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 consti-
tute 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

```c
#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

```c
#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

```c
#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

```c
#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**

```c
#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**

```c
#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 Xs, which will be replaced with a letter and the current process ID.

OS calls: access, getpid.

Reference: SVID.

**mktime**

```c
#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**

```c
#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

```c
#include <mathf.h>
float modff(float value, float *intptr);
```

Returns the fractional part of `value` and stores the integral part in the location pointed to by `intptr`. 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

```c
#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

```c
#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 \(+2^{\text{emin}}\) and \(-2^{\text{emin}}\). In both cases, inexact is signaled. Reference: ANSI 754, MATH, REENT.

nrand48

```c
#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

```c
#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

```c
#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

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

```c
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)

zero padding to field width (for d, i, o, u, x, X, e, E, f, g, and G conversions)

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.

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.

The following length modifiers are used:

Used before d, i, o, n, u, x, or X conversions to denote a short int or unsigned short int value.

Used before d, i, o, n, u, x, or X conversions to denote a long int or unsigned long int value.

Used before e, E, f, g or G conversions to denote a long double value.

The following conversion specifiers are used:

Write signed decimal integer value.

Write signed decimal integer value.

Write unsigned octal integer value

Write unsigned decimal integer value

Write unsigned hexadecimal (0-9, abc...) integer value

Write unsigned hexadecimal (0-9, ABC...) integer value.

Write floating point value: [-]d.ddde+dd.

Write floating point value: [-]d.dddE+dd.

Write floating point value: [-]ddd.ddd.
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**

```c
#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

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

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

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

Reference: ANSI.

putenv

```c
#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

```c
#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

```c
#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

```c
#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

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

Sends the signal `sig` to the executing program.

OS calls: `getpid`, `kill`.

Reference: ANSI.

rand

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

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

Reference: ANSI.

read

```c
#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

```c
#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

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

Returns `y * 2^N` for integral values `N` without computing `2^N`.
Reference: ANSI 754, MATH, REENT.

scanf

```c
#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).

Maximum field to be scanned (default is until no match occurs).

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.

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
_IOCBF causes stream to be line buffered
_IOLBF 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

```c
#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

```c
#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

```c
#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

```c
#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**

```c
#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**

```c
#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**

```c
#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**

```c
#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**

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

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

Reference: ANSI.

**srand48**

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

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

Reference: SVID.

**sscanf**

```c
#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**

```c
#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

```c
#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

```c
#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

```c
#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

```c
#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**

```c
#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**

```c
#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**

```c
#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**

```c
#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)
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**

```c
#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**

```c
#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**

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

Locates the last occurrence of \( c \) within the string pointed to by \( s \).

Reference: ANSI, REENT.

**strspn**

```c
#include <string.h>
size_t strspn(const char *, const char *s2);
```

Computes the length of the maximum initial segment of \( s1 \) which consists entirely of characters from \( s2 \).

Reference: ANSI, REENT.
Function Listing

ynchronously

strstr

```c
#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 \textit{s2} within the string pointed to by \textit{s1}.

Reference: ANSI, REENT.

strtod

```c
#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 \textit{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 \texttt{e} or \texttt{E} followed by an optional sign or space, followed by an integer. At return, the pointer at \texttt{*endptr} is set to the first unrecognized character.

Reference: ANSI, REERR.

strtok

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

Searches string \textit{s1} for address of the first element that equals none of the elements in string \textit{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 \textit{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

```c
#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

```c
#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

```c
#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 <dcl.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

```c
#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

```c
#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

```c
#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

```c
#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`. 

Reference: ANSI.
OS calls: `access`, `getpid`.
Reference: ANSI.

toascii

```c
#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

```c
#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

```c
#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

```c
#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

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

tzset() uses the contents of the environment variable \texttt{TZ} to override the value of the different external variables for the time zone. It scans the contents of \texttt{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

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

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

Reference: ANSI.

### unlink

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

Removes the directory entry \texttt{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

```c
#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.
vfprintf

```c
#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

```c
#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

```c
#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

```c
#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

```c
#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**

```c
#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**

```c
#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**

```c
#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**

```c
#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`

```c
#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`

```c
#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

achinery Reference Manual

%f format specifier 34

A

64l function 21
abort function 21
abs function 21
access function 21
acos function 22
acosh function 22
advance function 22, 28
alloca function 22
ANSI 754 reference 20
ANSI reference 20
asctime function 23, 91
asin function 23
assinf function 23
assert function 23
assert macro 14, 24
atan function 24
atan2 function 24
atan2f function 25
atanf function 24
atexit function 25
atoi function 25
atol function 26

B

bsearch function 26
BUFSIZ constant 17, 74–75

C

calloc function 26, 40, 71
ceil function 26–27
ceilf function 27
 chágsign function 27

class library

C++ complex.a 3
C++ iostream.a 4
clearerr function 27
clock function 27
CLOCKS_PER_SEC constant 17, 27
close function 28
COFF object module format
crt0.o startup module and libraries, parallel to
ELF 7
compile function 28
complex arithmetic library, libcomplex.a 6
complex.a C++ complex math class library 3

constants

BUFSIZ 17, 74–75
CLOCKS_PER_SEC 17, 27
DOMAIN 59
EDOM 22–24, 60
ENTER 46
EOF 17, 41, 72, 78, 91
ERANGE 33, 60
EXIT_FAILURE 17, 32
EXIT_SUCCESS 17, 32
FIND 46
HUGE_VAL 14, 16, 33
HUGE_VAL_F 16
_IOFBF 75
_IOFBF 75
_ICONBF 75
LC_ALL 75
LC_COLLATE 75, 79
LC_MONETARY 75
LC_NUMERIC 75
LC_TIME 75
L_tmpnam 88
MB_CUR_MAX 61
NULL
in stddef.h 17
in stdio.h 17
in stdlib.h 17
in string.h 17
O_APPEND 16
O_NDELAY 16
O_RDONLY 16, 65
O_RDWR 16, 65
OVERFLOW 59
O_WRONLY 16, 65
PLOSS 60
RAND_MAX 70
SEEK_CUR 58
SEEK_END 58
SEEK_SET 58
SING 59
TLOSS 60
UNDERFLOW 59
Copysign function 28
cos function 28
cosf function 29
cosh function 29
coshf function 29
creat function 14, 29, 34
cftime function 30

D
DCC reference 20
defined variables, types, and constants 15–17
difftime function 30
div function 17, 30, 54
div_t type 17, 30
DMALLOC_CHECK environment variable 59
DMALLOC_INIT environment variable 58
DOMAIN constant 59
drand48 function 30, 53, 78
dup function 34

E
ecvt function 31
EDOM constant 22–24, 60
ENTER constant 46
environment variables
DMALLOC_CHECK 59
DMALLOC_INIT 58
EOF constant 17, 41, 72, 78, 91
ERANGE constant 33, 60
erf function 31
erfc function 32
erfcf function 32
erff function 31
errno variable 14–15, 20, 22–24, 33, 36, 42, 60, 65
__exit function 32
exit function 17, 21, 25, 32
EXIT_FAILURE constant 17, 32
EXIT_SUCCESS constant 17, 32
exp function 33
expf function 33

F
fabs function 33
fabsf function 33
fclose function 34
fcntl function 14, 16, 34
fcvt function 34
fdopen function 34
feof function 35
ferror function 35
ferror function 35
fflush function 35
fgetc function 35
fgetpos function 36
fgets function 36
__FILE__ preprocessor macro 23
FILE structure 38, 40, 88
file type 38
fileno function 36
files
  include 13
  stderr 17, 40, 71
  stdin 17, 45, 72
  stdout 17, 66, 69
FIND constant 46
__finite function 36
floating point
  libcfp.a hardware library 6
  libcfp.a software library 7
  libcfp.a stubs library 6
floor function 37
floorf function 37
fmod function 37
fmodf function 37
fopen function 38
fpos_t type 17
fprintf function 38, 92
fputc function 39
fputs function 39
fread function 39
free function 40, 59
freopen function 40
frexp function 40
frexpf function 41, 64
fscanf function 41, 92
fseek function 41, 71
fsetpos function 42
fstat function 42
ftel function 42
function standards and definitions, table of 20
functions
  See specific function name
  modifies errno marked by REERR 20
  reentrant marked by REENT 20
fwrite 42

G
gamma function 43
gammaf function 43
gcvt function 43
getc function 35, 44, 91
getchar function 44
getenv function 44
getopt function 44
getpid function 45
gets function 45
getw function 45
gmtime function 45

H
hcreate function 46
hdestroy function 46
hsearch function 46
HUGE_VAL constant 14, 16, 33
HUGE_VAL_F constants 16
hypot function 47
hypotf function 47

I
include files 13
include files, standard, table of 14
input/output
  basic character input/output library, part of simple/libc.a 6
  RAM-disk library, part of cross/libc.a 6
#include files
  _IOFBF constant 75
  _IOLBF constant 75
  _IONBF constant 75
  iostream C++ library, libios.a 6
  iostream.a C++ iostream class library 4
  irand48 function 47
  isalnum function 47
  isalpha function 47
  isascii function 48
  isattf function 48
  iscntrl function 48
  isdigit function 48
  isgraph function 49
  islower function 49
  isnan function 49
  isprint function 49
  ispunct function 49
  isspace function 50
  isupper function 50
  isxdigit function 50

J
j0 function 50
j0f function 51
j1 function 51
j1f function 51
jmpbuf type 16
jn function 51
jnf function 52
jrand48 function 52

K
keyword, sizeof 17
keywords
sizeof 17
kill function 52
krand48 function 52

L
l3tol function 53, 58
l64a function 53
labs function 53
LC_ALL constant 75
LC_COLLATE constant 75, 79
LC_MONETARY constant 75
LC_NUMERIC constant 75
LC_TIME constant 75
ldexp function 53
ldexpf function 54
ldiv function 17, 54
ldiv_t type 17
_lessgreater function 54
lfind function 54
libcfp.a floating point library 3
libchar.a basic character I/O library 3
libd.a C++ additional standard library 4
libi.a standard C library 4
libimfpf.a compiler support library 4, 6
libimpl.a compiler support library 4
libm.a math library 4
libram.a RAM disk I/O library 4
libraries
complex.a, C++ complex math class 3
iostream.a, C++ iostream class 4
libcfp.a, floating point 3
libchar.a, basic character I/O 3
libd.a, additional standard C++ 4
libi.a, standard C 4
libimfpf.a, compiler support 4, 6
libimpl.a, compiler support 4
libm.a, math 4
libram.a, RAM disk I/O 4

library
basic character input output, part of libc.a 6
complex arithmetic, libcomplex.a 6
floating point hardware, libcfp.a 6
floating point software, libcfp.a 7
floating point stubs, libcfp.a 6
iostream C++, libios.a 6
libc.a 6
RAM-disk input output, part of libc.a 6
_LINE__ preprocessor macro 23

link function 55
-lnm option 20
localeconv function 55, 75
localtime function 55
location of include files, version_path/include 13
log function 55
log10 function 56
log10f function 56
_log10 function 56
logf function 56

lrand4 function 78
lrand48 function 53, 57
lsearch function 54, 57
lseek function 57, 87
L_tmpnam constant 88
ltol3 function 58

M
macros
assert 14, 24
va_arg 17
va_end 17
va_start 17, 92–93
mallinfo function 58
malloc function 14, 40, 58, 71
mallopt function 59
MATH functions require math library 20
matherr function 59
matherrf function 60
MB_CUR_MAX constant 61
mblen function 60
mbstowcs function 61
mbtowc function 61
memccpy function 61
memchr function 62
memcmp function 62
memcpy function 62
memmove function 62
memory allocation functions 14–15
memset function 63
mktemp function 63
mktime function 63
modf function 63
modff function 64
mrand48 function 53, 64, 78

N
NDDEBUG preprocessor macro 24
__nextafter function 64
__no_malloc_warning 40
__no_malloc_warning variable 71
mrand48 function 64
NULL constant
   in stdlib.h 17
   in stddef.h 17
   in stdio.h 17
   in string.h 17

O
O_APPEND constant 16
offsetof function 65
O_NDELAY constant 16
open function 14, 16, 34, 65
operator, sizeof 17
option, -lm 20
O_RDONLY constant 16, 65
O_RDWR constant 16
OVERFLOW constant 59
O_WRONLY constant 16, 65

P
perror function 65
pipe function 34
PLOSS constant 60
POSIX reference 20
pow function 66
powf function 66
preprocessor macros
   __FILE__ 23
   __LINE__ 23
   NDEBUG 24
printf function 34, 66
ptrdiff_t type 17
putc function 39, 68
putchar function 69
putenv function 69
puts function 69
putw function 69

Q
qsort function 70

R
raise function 70
rand function 70
RAND_MAX constant 70
read function 20, 70
realloc function 40, 71
REENT functions are reentrant 20
reentrant function marked by REENT 20
REERR functions modify errno 20
remove function 71
rename function 71
Index

C Library Reference Manual

rewind function 71

S
sbrk function 72
__scalb function 72
scanf function 72, 93
seed4 function 30
seed48 function 47, 57, 64, 74
SEEK_CUR constant 58
SEEK_END constant 58
SEEK_SET constant 58
setbuf function 17, 74
setjmp function 15–16, 57, 74
setlocale function 75
setvbuf function 75
SIGABRT signal 21
sig_atomic_t type 16
sigjmpbuf type 16
siglongjmp function 16
signal function 76
signals
    SIGABRT 21
sigsetjmp function 16
sigset_t type 16
sin function 76
sinf function 76
SING constant 59
sinh function 76
sinhf function 77
sizeof keyword 17
sizeof operator 17
size_t type
    in stddef.h 17
    instdio.h 17
    instdlib.h 17
    instring.h 17
sscanf function 77, 93
sqrt function 77
sqrtf function 77
strd48 function 30, 47, 57, 64, 78
sscanf function 78, 94

standard include files, table of 14
stderr file 17, 40, 71
stdin file 17, 45, 72
stdio 15, 48
stdout file 17, 66, 69
step function 28, 78
strcat function 79
strchr function 79
strcmp function 57, 79, 85
strcoll function 75, 79, 85
strncpy function 80
strerror function 80
strcat function 82
strcat function 82
strncmp function 82
strncat function 82
strchr function 83
strchr function 83
strerror function 83
strerror function 83
strerror function 83
strerror function 83
strftime function 75, 81
string functions 15
strerror function 82
strerror function 82
strerror function 82
struct Iconv 55
strxfrm function 75, 85
SVID reference 20
swab function 86
SYS functions provided by system 20
sys_errlist variable 65
sys_nerr variable 65

T
tan function 86
tanf function 86
tanh function 86
tanhf function 87
tdelete function 87
tell function 87
 tempnam function 87
tfind function 88
time function 63, 88
TLOSS constant 60
tmpfile function 88
tmpnam function 88
toascii function 89
_tolower function 89
tolower function 89
_toupper function 90
toupper function 89
tsearch function 90
twalk function 90

types 17
 div_t 17, 30
 fpos_t 17
 jmpbuf 16
 ldiv_t 17
 ptdiff 17
 sig_atomic_t 16
 sigjmpbuf 16
 sigset_t 16
 size_t
 in stddef.h 17
 in stdio.h 17
 in stdlib.h 17
 in string.h 17
 VISIT 90

va_list type 17

variables
 errno 14–15, 20, 22–24, 33, 36, 42, 60, 65
 __no_malloc_warning 71
 sys_errlist 65
 sys_nerr 65

va_start macro 17, 92–93
vfprintf function 92
vscanf function 92
VISIT type 90
vprintf function 92
vsprintf function 93
vsscanf function 94

W
 westombs function 94
wctomb function 94
write function 94

Y
 y0 function 95
 y0f function 95
 y1 function 95
 y1f function 95
 yn function 96
 ynf function 96

U
 UNDERFLOW constant 59
 ungetc function 91
 UNIX reference 20
 unlink function 91
 unordered function 91

V
 va_arg macro 17
 va_end macro 17