G

gcvt() — Extended function (libc)

Convert floating-point numbers to strings
char *gcvt(double *d*, **int** *prec*, **char** **buffer*);

gcvt converts a floating point number into an ASCII string. Its operation resembles that of the **%g** operator to **printf**. **gcvt** converts its argument *d* into a null-terminated string of decimal numerals with a precision (i.e., the number of numerals to the right of the decimal point) of *prec*. Unlike its cousins **ecvt** and **fcvt**, **gcvt** uses a buffer that is defined by the caller. *buffer* must point to a buffer large enough to hold the result; 64 characters will always be sufficient.

When generating its output, **gcvt** will mimic **fcvt** if possible. Otherwise, it mimics **ecvt**.

gcvt returns *buffer*.

Example

For an example of this function, see the entry for **ecvt**.

See Also

ecvt, extended miscellaneous, fcvt, frexp, ldexp, modf, printf

general utilities — Overview

#include <stdlib.h>

The ANSI standard describes a set of general utilities. As its name implies, this set is a grab-bag of utilities that do not fit neatly anywhere else. In accordance with the Standard's principle that every function must be declared in a header, the Committee created the header **stdlib.h** to hold the general utilities and their attendant macros and types.

The general utilities are as follows:

Environment communication

abort	End program immediately
atexit	Register a function to be performed at exit
exit	Terminate a program gracefully
getenv	Get environment variable
system	Suspend program and execute another

Integer arithmetic functions

abs	Compute absolute value of an integer
div	Perform integer division
labs	Compute absolute value of a long integer
ldiv	Perform long integer division

Memory management

calloc	Allocate and clear dynamic memory
free	De-allocate dynamic memory
malloc	Allocate dynamic memory
realloc	Reallocate dynamic memory

Multibyte character functions

mblen	Compute length of a multibyte character
mbstowcs	Convert sequence of multibyte characters to wide characters
mbtowc	Convert multibyte character to wide character
wcstombs	Convert sequence of wide characters to multibyte characters
wctomb	Convert wide character to multibyte character

LEXICON

Pseudo-random number functions

rand	Generate pseudo-random numbers
srand	Seed pseudo-random number generator

Searching-sorting

bsearch	Search an array
qsort	Sort an array

String conversion functions

atof	Convert string to floating-point number
atoi	Convert string to integer
atol	Convert string to long integer
strtod	Convert string to double-precision floating-point number
strtol	Convert string to long integer
strtoul	Convert string to unsigned long integer

Cross-references

Standard, §4.10.1
The C Programming Language, ed. 2, pp. 251*ff*

See Also

div_t, ldiv_t, Library, stdlib.h, wchar_t

getc() — STDIO (stdio.h)

Read a character from a stream
#include <stdio.h>
int getc(FILE ******fp***);**

getc reads a character from the stream pointed to by *fp*. The character is read as an **unsigned char** converted to an **int**.

If all goes well, **getc** returns the character read. If it reads the end of file, it returns **EOF** and sets the end-of-file indicator. If an error occurs, it returns **EOF** and sets the error indicator.

Cross-references

Standard, §4.9.7.5
The C Programming Language, ed. 2, p. 247

See Also

fgetc, getchar, gets, putc, putchar, puts, STDIO, ungetc

Notes

Let's C implements **getc** as a macro, which means that *fp* could be evaluated more than once. Therefore, one should beware of the side-effects of evaluating the argument more than once, especially if the argument itself has side-effects.

getchar() — STDIO (stdio.h)

Read a character from the standard input stream
#include <stdio.h>
int getchar(void);

getchar reads and returns a character from the file or device associated with **stdin**. It is equivalent to:

```
getc(stdin);
```

If **getchar** reads the end of file, it returns **EOF** and sets the file's end-of-file indicator. Likewise, if an error occurs, it returns **EOF** and sets the file's error indicator.

Example

This example copies onto the standard-output device whatever is typed upon the standard-input device. To exit, type **EOF**; what this character is depends upon the operating system that your computer is running.

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

main(void)
{
      int c;

      while((c = getchar()) != EOF)
            putchar(c);
      return(EXIT_SUCCESS);
}
```

Cross-references

Standard, §4.9.7.6
The C Programming Language, ed. 2, p. 247

See Also

getc, gets, putc, putchar, puts, STDIO, ungetc

getenv() — General utility (libc)

Read environmental variable
#include <stdlib.h>
char *getenv(const char ****variable*);**

The environment itself can make information available to a program. This information often is available in the form of an *environment variable*, which is a string that forms a definition. For example, under the UNIX operating system the environment variable **TERM** indicates the type of terminal the user has. The variable **TERM=myterm** indicates that the user is typing on a *myterm* variety of terminal. When a program reads that declaration, it knows to use the coding proper for that terminal.

The environment variables together form the *environment list*. Given the heterogeneous environments under which C is implemented, the Standard does not define the mechanism by which the environment list is passed to a program.

The function **getenv** scans the environment list and looks for the variable that is named in the string pointed to by *variable*.

getenv returns a pointer to the string that defines the variable. It returns NULL if the variable requested cannot be found.

Example

This program looks up words in the environment and displays them.

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

```
main(void)
{
      for(;;) {
            char buf[80], *is;

            printf("Enter an environmental variable: ");
            fflush(stdout);

            if(gets(buf) == NULL)
                  exit(EXIT_SUCCESS);

            if((is = getenv(buf)) == NULL)
                  printf("Can't find %s\n", buf);
            else
                  printf("%s = %s\n", buf, is);
      }

      return(EXIT_SUCCESS);
}
```

Cross-references

Standard, §4.10.4.4
The C Programming Language, ed. 2, p. 253

See Also

environment list, general utilities

Notes

getenv uses a static area to hold the environment variable requested. This buffer will be overwritten by subsequent calls to **getenv**.

gets() — STDIO (libc)

Read a string from the standard input stream
#include <stdio.h>
char ***gets(char** **buffer**);*

gets reads characters from the standard input stream and stores them in the area pointed to by *buffer*. It stops reading as soon as it detects a newline character or the end of file. **gets** discards the newline or **EOF** and appends a null character onto the end of the string it has built.

If all goes well, **gets** returns *buffer*. When it has encountered the end of file without having placed any characters into *buffer*, it returns NULL and leaves the contents of *buffer* unchanged. If a read error occurs, **gets** returns NULL and the contents of *buffer* may or may not be altered.

Example

This example echoes whatever is typed upon the standard-input device.

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

main(void)
{
      char buf[100];

      while(gets(buf) != NULL)
            puts(buf);
      return(EXIT_SUCCESS);
}
```

Cross-references

Standard, §4.9.7.7
The C Programming Language, ed. 2, p. 247

See Also

fgets, getc, getchar, putc, putchar, puts, STDIO, ungetc

Notes

gets stops reading the input string as soon as it detects a newline character. If a previous read from the standard input stream left a newline character in the standard input buffer, **gets** will read it and immediately stop accepting characters. To the user, it will appear as if **gets** is not working at all.

For example, if **getchar** is followed by **gets**, the first character **gets** will receive is the newline character left behind by **getchar**. A simple statement will remedy this:

```
while (getchar() != '\n')
      ;
```

This discards the newline character left behind by **getchar**. **gets** will now work correctly. You should use this only when you know that a newline will be left in the buffer. Otherwise, the desired line will be lost

getw() — Extended function (libc)

Read word from file stream
#include <xstdio.h>
int getw(FILE *_fp_**);**

getw reads a word (an **int**) from the file stream *fp*, and returns it. It differs from the related function **getc** in that **getc** returns either a **char** promoted to an **int**, or EOF.

getw returns EOF on errors; however, you must call **feof** or **ferror** distinguish this value from a valid end-of-file signal.

Example

For an example of this function, see the entry for **inb**.

See Also

extended STDIO, getc

Notes

getw assumes that the bytes of the word it receives are in the natural byte ordering of the machine. See the entry on **byte ordering** for more information. This means that such files might not be portable between machines.

To conform to the ANSI Standard, this function has been moved from the header **stdio.h** to the header **xstdio.h**. This may require that some code be altered.

getw is not described in the ANSI Standard. A program that uses it does not comply strictly with the Standard, and may not be portable to other compilers or operating systems.

gmtime() — Time function (libc)

Convert calendar time to universal coordinated time
#include <time.h>
struct tm ***gmtime(const time_t** ****caltime**);**

The function **gmtime** takes the calendar time pointed to by *caltime* and breaks it down into a structure of the type **tm**, converting it into universal coordinated time.

LEXICON

gmtime returns a pointer to the structure **tm** that it creates. This structure is defined in the header **time.h**. If universal coordinated time cannot be computed, then **gmtime** returns NULL.

Example

This example shows Universal Coordinated Time in a message of the form "12/22/88 15:27:33".

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

main(void)
{
      time_t now;
      char buffer[80];

      time(&now);
      strftime(buffer, sizeof(buffer),
            "%m/%d/%y %H:%M:%S\n", gmtime(&now));
      printf(buffer);
      return(EXIT_SUCCESS);
}
```

Cross-references

Standard, §4.12.3.3
The C Programming Language, ed. 2, p. 256

See Also

asctime, ctime, date and time, localtime, strftime, tm, universal coordinated time

Notes

The name "gmtime" reflects the term "Greenwich Mean Time." the Standard prefers the term "universal coordinated time," although for all practical purposes the two are identical.

gmtime is useful only on a system whose time is set to UTC rather than to local time. The **Let's C** time routines read the environmental variable **TIMEZONE** to translate UTC automatically into your local time, should you wish. See the entry for **TIMEZONE** for more information on how this works.

gmtime returns a pointer to a statically allocated data area that is overwritten by successive calls.

goto — C keyword

Unconditionally jump within a function
goto *label***;**

The **goto** statement forces a program's execution to jump to the point marked by *label*. A **goto** can jump only to a point within the current function. To jump beyond a function boundary, use the functions **longjmp** and **setjmp**.

The most common use for **goto** is to exit from nested control structures or go to the top of a control block. It is used most often to write "ripcord" routines, i.e., routines that are executed when a error occurs too deeply within a program for the program to disentangle itself correctly.

Example

For an example of this statement, see **name space**.

Cross-references

Standard, §4.6.6.1
The C Programming Language, ed. 2, p. 65

See Also

break, C keywords, continue, label name, non-local jumps, return, statements

Notes

The C Programming Language describes **goto** as "infinitely-abusable." *Caveat utilitor*.