C Header Files

No.	Name	Description
1	stdio.h	Input/Output Functions
2	conio.h	console input/output
3	assert.h	Diagnostics Functions
4	ctype.h	Character Handling Functions
5	cocale.h	Localization Functions
6	math.h	Mathematics Functions
7	setjmp.h	Nonlocal Jump Functions
8	signal.h	Signal Handling Functions
9	stdarg.h	Variable Argument List Functions
10	stdlib.h	General Utility Functions
11	string.h	String Functions
12	time.h	Date and Time Functions
13	complex.h	A set of function for manipulating complex numbers
14	stdalign.h	For querying and specifying the alignment of objects
15	errno.h	For testing error codes

16	locale.h	Defines localization functions
17	stdatomic.h	For atomic operations on data shared between threads
18	stdnoreturn.h	For specifying non-returning functions
19	uchar.h	Types and functions for manipulating Unicode characters
20	fenv.h	A set of functions for controlling floating-point environment
21	wchar.h	Defines wide string handling functions
22	tgmath.h	Type-generic mathematical functions
23	stdarg.h	Accessing a varying number of arguments passed to functions
24	stdbool.h	Defines a boolean data type

C – stdio.h library functions

All C inbuilt functions which are declared in stdio.h header file are given below. The source code for stdio.h header file is also given below for your reference.

LIST OF INBUILT C FUNCTIONS IN STDIO.H FILE:

Function	Description
printf()	This function is used to print the character, string, float, integer, octal and hexadecimal values onto the output screen
scanf()	This function is used to read a character, string, numeric data from keyboard.
getc()	It reads character from file
gets()	It reads line from keyboard
getchar()	It reads character from keyboard
puts()	It writes line to o/p screen

putchar()	It writes a character to screen
clearerr()	This function clears the error indicators
f open()	All file handling functions are defined in stdio.h header file
f close()	closes an opened file
getw()	reads an integer from file
putw()	writes an integer to file
f getc()	reads a character from file
putc()	writes a character to file
f putc()	writes a character to file
f gets()	reads string from a file, one line at a time
f puts()	writes string to a file
f eof()	finds end of file
f getchar	reads a character from keyboard
f getc()	reads a character from file
f printf()	writes formatted data to a file
f scanf()	reads formatted data from a file
f getchar	reads a character from keyboard
f putchar	writes a character from keyboard
f seek()	moves file pointer position to given location
SEEK_SET	moves file pointer position to the beginning of the file

SEEK_CUR	moves file pointer position to given location
SEEK_END	moves file pointer position to the end of file.
f tell()	gives current position of file pointer
rewind()	moves file pointer position to the beginning of the file
putc()	writes a character to file
sprint()	writes formatted output to string
sscanf()	Reads formatted input from a string
remove()	deletes a file
fflush()	flushes a file

C – conio.h library functions

All C inbuilt functions which are declared in conio.h header file are given below. The source code for conio.h header file is also given below for your reference.

LIST OF INBUILT C FUNCTIONS IN CONIO.H FILE:

Functions	Description
clrscr()	This function is used to clear the output screen.
getch()	It reads character from keyboard
getche()	It reads character from keyboard and echoes to o/p screen
textcolor()	This function is used to change the text color
textbackground()	This function is used to change text background

C Library - <assert.h>

The **assert.h** header file of the C Standard Library provides a macro called **assert** which can be used to verify assumptions made by the program and print a diagnostic message if this assumption is false.

The defined macro **assert** refers to another macro **NDEBUG** which is not a part of <assert.h>. If NDEBUG is defined as a macro name in the source file, at the point where <assert.h> is included, the **assert** macro is defined as follows –

Sr.No.	Function & Description
1	**void assert(int expression)** This is actually a macro and not a function, which can be used to add diagnostics in your C program.

C Library - <ctype.h>

The **ctype.h** header file of the C Standard Library declares several functions that are useful for testing and mapping characters.

All the functions accepts **int** as a parameter, whose value must be EOF or representable as an unsigned char.

All the functions return non-zero (true) if the argument c satisfies the condition described, and zero(false) if not.

Sr.No.	Function & Description
1	**int isalnum(int c)** This function checks whether the passed character is alphanumeric.
2	**int isalpha(int c)** This function checks whether the passed character is alphabetic.
3	**int iscntrl(int c)** This function checks whether the passed character is control character.
4	**int isdigit(int c)**

	This function checks whether the passed character is decimal digit.
5	**int isgraph(int c)** This function checks whether the passed character has graphical representation using locale.
6	**int islower(int c)** This function checks whether the passed character is lowercase letter.
7	**int isprint(int c)** This function checks whether the passed character is printable.
8	**int ispunct(int c)** This function checks whether the passed character is a punctuation character.
9	**int isspace(int c)** This function checks whether the passed character is white-space.
10	**int isupper(int c)** This function checks whether the passed character is an uppercase letter.
11	**int isxdigit(int c)** This function checks whether the passed character is a hexadecimal digit.

The library also contains two conversion functions that accepts and returns an "int".

Sr.No.	Function & Description
1	**int tolower(int c)** This function converts uppercase letters to lowercase.
2	**int toupper(int c)** This function converts lowercase letters to uppercase.

C Library - <locale.h>

The **locale.h** header defines the location specific settings, such as date formats and currency symbols. You will find several macros defined along with an important structure **struct lconv** and two important functions listed below.

Sr.No.	Macro & Description
1	**LC_ALL** Sets everything.
2	**LC_COLLATE** Affects strcoll and strxfrm functions.
3	**LC_CTYPE** Affects all character functions.
4	**LC_MONETARY** Affects the monetary information provided by localeconv function.
5	**LC_NUMERIC** Affects decimal-point formatting and the information provided by localeconv function.
6	**LC_TIME** Affects the strftime function.

Library Functions

Following are the functions defined in the header locale.h –

Sr.No.	Function & Description

1	**char *setlocale(int category, const char *locale)** Sets or reads location dependent information.
2	**struct lconv *localeconv(void)** Sets or reads location dependent information.

C – math.h library functions

All C inbuilt functions which are declared in math.h header file are given below. The source code for math.h header file is also given below for your reference.

LIST OF INBUILT C FUNCTIONS IN MATH.H FILE:

- "math.h" header file supports all the mathematical related functions in C language. All the arithmetic functions used in C language are given below.
- Click on each function name below for detail description and example programs.

Function	Description
floor ( )	This function returns the nearest integer which is less than or equal to the argument passed to this function.
round ( )	This function returns the nearest integer value of the float/double/long double argument passed to this function. If decimal value is from ".1 to .5", it returns integer value less than the argument. If decimal value is from ".6 to .9", it returns the integer value greater than the argument.
ceil ( )	This function returns nearest integer value which is greater than or equal to the argument passed to this function.
sin ( )	This function is used to calculate sine value.
cos ( )	This function is used to calculate cosine.
cosh ( )	This function is used to calculate hyperbolic cosine.
exp ( )	This function is used to calculate the exponential "e" to the x^{th} power.
tan ( )	This function is used to calculate tangent.
tanh ( )	This function is used to calculate hyperbolic tangent.
sinh ( )	This function is used to calculate hyperbolic sine.
log ( )	This function is used to calculates natural logarithm.
log10 ( )	This function is used to calculates base 10 logarithm.
sqrt ( )	This function is used to find square root of the argument passed to this function.

pow ( )	This is used to find the power of the given number.
trunc ( )	This function truncates the decimal value from floating point value and returns integer value.

C Library - <setjmp.h>

The **setjmp.h** header defines the macro **setjmp()**, one function **longjmp()**, and one variable type **jmp_buf**, for bypassing the normal function call and return discipline.

Library Variables

Following is the variable type defined in the header setjmp.h –

Sr.No.	Variable & Description
1	**jmp_buf** This is an array type used for holding information for macro **setjmp()** and function **longjmp()**.

Library Macros

There is only one macro defined in this library –

Sr.No.	Macro & Description
1	**int setjmp(jmp_buf environment)** This macro saves the current *environment* into the variable **environment** for later use by the function **longjmp()**. If this macro returns directly from the macro invocation, it returns zero but if it returns from a **longjmp()** function call, then a non-zero value is returned.

Library Functions

Following is the only one function defined in the header setjmp.h –

Sr.No.	Function & Description
1	**void longjmp(jmp_buf environment, int value)** This function restores the environment saved by the most recent call to **setjmp()** macro in the same invocation of the program with the corresponding **jmp_buf** argument.

C Library - <signal.h>

The **signal.h** header defines a variable type **sig_atomic_t**, two function calls, and several macros to handle different signals reported during a program's execution.

Library Variables

Following is the variable type defined in the header signal.h –

Sr.No.	Variable & Description
1	**sig_atomic_t** This is of **int** type and is used as a variable in a signal handler. This is an integral type of an object that can be accessed as an atomic entity, even in the presence of asynchronous signals.

Library Macros

Following are the macros defined in the header signal.h and these macros will be used in two functions listed below. The **SIG_** macros are used with the signal function to define signal functions.

Sr.No.	Macro & Description
1	**SIG_DFL** Default signal handler.

2	**SIG_ERR** Represents a signal error.
3	**SIG_IGN** Signal ignore.

The **SIG** macros are used to represent a signal number in the following conditions –

Sr.No.	Macro & Description
1	**SIGABRT** Abnormal program termination.
2	**SIGFPE** Floating-point error like division by zero.
3	**SIGILL** Illegal operation.
4	**SIGINT** Interrupt signal such as ctrl-C.
5	**SIGSEGV** Invalid access to storage like segment violation.
6	**SIGTERM** Termination request.

Library Functions

Following are the functions defined in the header signal.h –

Sr.No.	Function & Description
1	**void (*signal(int sig, void (*func)(int)))(int)** This function sets a function to handle signal i.e. a signal handler.
2	**int raise(int sig)** This function causes signal **sig** to be generated. The sig argument is compatible with the SIG macros.

C Library - <stdarg.h>

The **stdarg.h** header defines a variable type **va_list** and three macros which can be used to get the arguments in a function when the number of arguments are not known i.e. variable number of arguments.

A function of variable arguments is defined with the ellipsis (,...) at the end of the parameter list.

Library Variables

Following is the variable type defined in the header stdarg.h –

Sr.No.	Variable & Description
1	**va_list** This is a type suitable for holding information needed by the three macros **va_start(), va_arg()** and **va_end()**.

Library Macros

Following are the macros defined in the header stdarg.h –

Sr.No.	Macro & Description
1	**void va_start(va_list ap, last_arg)**

	This macro initializes **ap** variable to be used with the **va_arg** and **va_end** macros. The **last_arg** is the last known fixed argument being passed to the function i.e. the argument before the ellipsis.
2	**type va_arg(va_list ap, type)** This macro retrieves the next argument in the parameter list of the function with type **type**.
3	**void va_end(va_list ap)** This macro allows a function with variable arguments which used the **va_start** macro to return. If **va_end** is not called before returning from the function, the result is undefined.

C Library - <stdlib.h>

The **stdlib.h** header defines four variable types, several macros, and various functions for performing general functions.

Library Variables

Following are the variable types defined in the header stdlib.h –

Sr.No.	Variable & Description
1	**size_t** This is the unsigned integral type and is the result of the **sizeof**keyword.
2	**wchar_t** This is an integer type of the size of a **wide** character constant.
3	**div_t** This is the structure returned by the **div** function.
4	**ldiv_t**

	This is the structure returned by the **ldiv** function.

Library Macros

Following are the macros defined in the header stdlib.h –

Sr.No.	Macro & Description
1	**NULL** This macro is the value of a null pointer constant.
2	**EXIT_FAILURE** This is the value for the exit function to return in case of failure.
3	**EXIT_SUCCESS** This is the value for the exit function to return in case of success.
4	**RAND_MAX** This macro is the maximum value returned by the rand function.
5	**MB_CUR_MAX** This macro is the maximum number of bytes in a multi-byte character set which cannot be larger than MB_LEN_MAX.

Library Functions

Following are the functions defined in the header stdio.h –

Sr.No.	Function & Description
1	**double atof(const char *str)** Converts the string pointed to, by the argument *str* to a floating-point number (type double).
2	**int atoi(const char *str)**

	Converts the string pointed to, by the argument *str* to an integer (type int).
3	**long int atol(const char *str)** Converts the string pointed to, by the argument *str* to a long integer (type long int).
4	**double strtod(const char *str, char **endptr)** Converts the string pointed to, by the argument *str* to a floating-point number (type double).
5	**long int strtol(const char *str, char **endptr, int base)** Converts the string pointed to, by the argument *str* to a long integer (type long int).
6	**unsigned long int strtoul(const char *str, char **endptr, int base)** Converts the string pointed to, by the argument *str* to an unsigned long integer (type unsigned long int).
7	**void *calloc(size_t nitems, size_t size)** Allocates the requested memory and returns a pointer to it.
8	**void free(void *ptr** Deallocates the memory previously allocated by a call to *calloc, malloc,* or *realloc*.
9	**void *malloc(size_t size)** Allocates the requested memory and returns a pointer to it.
10	**void *realloc(void *ptr, size_t size)** Attempts to resize the memory block pointed to by ptr that was previously allocated with a call to *malloc* or *calloc*.

11	**void abort(void)** Causes an abnormal program termination.
12	**int atexit(void (*func)(void))** Causes the specified function **func** to be called when the program terminates normally.
13	**void exit(int status)** Causes the program to terminate normally.
14	**char *getenv(const char *name)** Searches for the environment string pointed to by name and returns the associated value to the string.
15	**int system(const char *string)** The command specified by string is passed to the host environment to be executed by the command processor.
16	**void *bsearch(const void *key, const void *base, size_t nitems, size_t size, int (*compar)(const void *, const void *))** Performs a binary search.
17	**void qsort(void *base, size_t nitems, size_t size, int (*compar)(const void *, const void*))** Sorts an array.
18	**int abs(int x)** Returns the absolute value of x.
19	**div_t div(int numer, int denom)** Divides numer (numerator) by denom (denominator).
20	**long int labs(long int x)**

	Returns the absolute value of x.
21	**ldiv_t ldiv(long int numer, long int denom)** Divides numer (numerator) by denom (denominator).
22	**int rand(void)** Returns a pseudo-random number in the range of 0 to *RAND_MAX*.
23	**void srand(unsigned int seed)** This function seeds the random number generator used by the function **rand**.
24	**int mblen(const char *str, size_t n)** Returns the length of a multibyte character pointed to by the argument *str*.
25	**size_t mbstowcs(schar_t *pwcs, const char *str, size_t n)** Converts the string of multibyte characters pointed to by the argument *str* to the array pointed to by *pwcs*.
26	**int mbtowc(whcar_t *pwc, const char *str, size_t n)** Examines the multibyte character pointed to by the argument *str*.
27	**size_t wcstombs(char *str, const wchar_t *pwcs, size_t n)** Converts the codes stored in the array *pwcs* to multibyte characters and stores them in the string *str*.
28	**int wctomb(char *str, wchar_t wchar)** Examines the code which corresponds to a multibyte character given by the argument *wchar*.

C – string.h library functions

All C inbuilt functions which are declared in string.h header file are given below. The source code for string.h header file is also given below for your reference.

LIST OF INBUILT C FUNCTIONS IN STRING.H FILE:

String functions	Description
strcat ( )	Concatenates str2 at the end of str1
strncat ( )	Appends a portion of string to another
strcpy ( )	Copies str2 into str1
strncpy ( )	Copies given number of characters of one string to another
strlen ( )	Gives the length of str1
strcmp ( )	Returns 0 if str1 is same as str2. Returns <0 if strl < str2. Returns >0 if str1 > str2
strcmpi ( )	Same as strcmp() function. But, this function negotiates case. "A" and "a" are treated as same.
strchr ( )	Returns pointer to first occurrence of char in str1
strrchr ( )	last occurrence of given character in a string is found
strstr ( )	Returns pointer to first occurrence of str2 in str1
strrstr ( )	Returns pointer to last occurrence of str2 in str1
strdup ( )	Duplicates the string
strlwr ( )	Converts string to lowercase
strupr ( )	Converts string to uppercase
strrev ( )	Reverses the given string

strset ( )	Sets all character in a string to given character
strnset ( )	It sets the portion of characters in a string to given character
strtok ( )	Tokenizing given string using delimiter

string functions	Description
memset()	It is used to initialize a specified number of bytes to null or any other value in the buffer
memcpy()	It is used to copy a specified number of bytes from one memory to another
memmove()	It is used to copy a specified number of bytes from one memory to another or to overlap on same memory.
memcmp()	It is used to compare specified number of characters from two buffers
memicmp()	It is used to compare specified number of characters from two buffers regardless of the case of the characters
memchr()	It is used to locate the first occurrence of the character in the specified string

C Library - <time.h>

The **time.h** header defines four variable types, two macro and various functions for manipulating date and time.

Library Variables

Following are the variable types defined in the header time.h –

Sr.No.	Variable & Description
1	**size_t** This is the unsigned integral type and is the result of the **sizeof**keyword.

2	**clock_t** This is a type suitable for storing the processor time.
3	**time_t is** This is a type suitable for storing the calendar time.
4	**struct tm** This is a structure used to hold the time and date.

C Library - <complex.h>

Basic operations

`cabs`	computes absolute value
`carg`	computes argument of a complex number
`cimag`	computes imaginary part of a complex number
`creal`	computes real part of a complex number
`conj`	computes complex conjugate
`cproj`	computes complex projection into the Riemann sphere

Exponentiation operations

`cexp`	computes complex exponential
`clog`	computes complex logarithm
`csqrt`	computes complex square root

`cpow`	computes complex power

Trigonometric operations

`csin`	computes complex sine
`ccos`	computes complex cosine
`ctan`	computes complex tangent
`casin`	computes complex arc sine
`cacos`	computes complex arc cosine
`catan`	computes complex arc tangent

Hyperbolic operations

`csinh`	computes complex hyperbolic sine
`ccosh`	computes complex hyperbolic cosine
`ctanh`	computes complex hyperbolic tangent
`casinh`	computes complex hyperbolic arc sine
`cacosh`	computes complex hyperbolic arc cosine
`catanh`	computes complex hyperbolic arc tangent

<stdalign.h> Header – C

The **<stdalign.h>** header is part of the standard library of the C programming language that provides four convenient macros for dealing with alignments of objects. This header was added in C11.

The header provides a single macro definition:

Macro Name	Description	Since
alignas	expand to _Alignas	C11
alignof	expand to _Alignof	C11
__alignas_is_defined	expand to the integer constant 1	C11
__alignof_is_defined	expand to the integer constant 1	C11

C Library - <errno.h>

The **errno.h** header file of the C Standard Library defines the integer variable **errno**, which is set by system calls and some library functions in the event of an error to indicate what went wrong. This macro expands to a modifiable lvalue of type int, therefore it can be both read and modified by a program.

The **errno** is set to zero at program startup. Certain functions of the standard C library modify its value to other than zero to signal some types of error. You can also modify its value or reset to zero at your convenience.

The **errno.h** header file also defines a list of macros indicating different error codes, which will expand to integer constant expressions with type **int**.

Library Macros

Following are the macros defined in the header errno.h –

Sr.No.	Macro & Description
1	**extern int errno** This is the macro set by system calls and some library functions in the

	event of an error to indicate what went wrong.
2	**EDOM Domain Error** This macro represents a domain error, which occurs if an input argument is outside the domain, over which the mathematical function is defined and errno is set to EDOM.
3	**ERANGE Range Error** This macro represents a range error, which occurs if an input argument is outside the range, over which the mathematical function is defined and errno is set to ERANGE.

C Library - <locale.h>

The **locale.h** header defines the location specific settings, such as date formats and currency symbols. You will find several macros defined along with an important structure **struct lconv** and two important functions listed below.

Library Macros

Following are the macros defined in the header and these macros will be used in two functions listed below –

Sr.No.	Macro & Description
1	**LC_ALL** Sets everything.
2	**LC_COLLATE** Affects strcoll and strxfrm functions.
3	**LC_CTYPE** Affects all character functions.

4	**LC_MONETARY** Affects the monetary information provided by localeconv function.
5	**LC_NUMERIC** Affects decimal-point formatting and the information provided by localeconv function.
6	**LC_TIME** Affects the strftime function.

Library Functions

Following are the functions defined in the header locale.h –

Sr.No.	Function & Description
1	**char *setlocale(int category, const char *locale)** Sets or reads location dependent information.
2	**struct lconv *localeconv(void)** Sets or reads location dependent information.

C Programming/stdatomic.h

The **stdatomic.h** header defines several macros and declares several types and functions for performing atomic operations on data shared between threads.

Initialization

`atomic_init`	initializes an atomic object

Fences

`atomic_thread_fence`	
`atomic_signal_fence`	

Lock-Free Property

`atomic_is_lock_free`	indicates whether or not an atomic object is lock-free

Operations

`atomic_store`	replaces the value of an atomic object
`atomic_load`	returns the value of an atomic object
`atomic_exchange`	replaces and returns the value of an atomic object
`atomic_compare_exchange_strong`	
`atomic_compare_exchange_weak`	
`atomic_fetch_key`	
`atomic_flag_test_and_set`	
`atomic_flag_clear`	

<stdnoreturn.h> Header – C

The **<stdnoreturn.h>** header is part of the standard library of the C programming language that provides a single convenient macro for non-returning functions. This header was added in C11.

The header provides a single macro definition:

Macro Name	Description	Since
noreturn	expands to _Noreturn	C11

<cuchar> (uchar.h)

Unicode characters

This header provides support for 16-bit and 32-bit characters, suitable to be encoded using UTF-16 and UTF-32.

Types

In C, this header defines two macros: `char16_t` and `char32_t`, which map to unsigned integral types of the appropriate size (the same as uint_least16_t and uint_least32_t, respectively).

In C++, `char16_t` and `char32_t` are fundamental types (and thus this header does not define such macros in C++).

Macros

In C++, the following macros are defined by this header:

Macro	description
`__STD_UTF_16__`	If defined, values of type `char16_t` have UTF-16 encoding. Otherwise, the encoding of `char16_t` is unspecified. (In C11, the macro expands to `1` when defined)
`__STD_UTF_32__`	If defined, values of type `char32_t` have UTF-32 encoding. Otherwise, the encoding of `char32_t` is unspecified. (In C11, the macro expands to `1` when defined)

Functions

c16rtomb

Convert 16-bit character to multibyte sequence (function)

c32rtomb

Convert 32-bit character to multibyte sequence (function)

mbrtoc16

Convert multibyte sequence to 16-bit character (function)

mbrtoc32

Convert multibyte sequence to 32-bit character (function)

<cfenv> (fenv.h)

Floating-point environment

This header declares a set of functions and macros to access the *floating-point environment*, along with specific types.

The *floating-point environment* maintains a series of *status flags* and specific *control modes*. Specific about the contents of the *floating-point environment* depend on the implementation, but the *status flags* generally include the *floating-point exceptions* and their associated information, and the *control modes* include at least the *rounding direction*.

Functions

Floating-point exceptions

feclearexcept

Clear floating-point exceptions (function)

feraiseexcept

Raise floating-point exception (function)

fegetexceptflag

Get floating-point exception flags (function)

fesetexceptflag

Set floating-point exception flags (function)

Rounding direction

fegetround

Get rounding direction mode (function)

fesetround

Set rounding direction mode (function)

Entire environment

fegetenv

Get floating-point environment (function)

fesetenv

Set floating-point environment (function)

feholdexcept

Hold floating-point exceptions (function)

feupdateenv

Update floating-point environment (function)

Other

fetestexcept

Test for floating-point exceptions (function)

Types

fenv_t

Floating-point environment type (type)

fexcept_t

Floating-point exceptions type (type)

Macro constants

Floating-point exceptions

FE_DIVBYZERO

Pole error exception (macro)

FE_INEXACT

Inexact result exception (macro)

FE_INVALID

Invalid argument exception (macro)

FE_OVERFLOW

Overflow range error exception (macro)

FE_UNDERFLOW

Underflow range error exception (macro)

FE_ALL_EXCEPT

All exceptions (macro)

Rounding directions

FE_DOWNWARD

Downward rounding direction mode (macro)

FE_TONEAREST

To-nearest rounding direction mode (macro)

FE_TOWARDZERO

Toward-zero rounding direction mode (macro)

FE_UPWARD

Upward rounding direction mode (macro)

Entire evnironment

FE_DFL_ENV

Default environment (macro)

Pragmas

FENV_ACCESS

Access to Floating-point environment (pragma)

<cwchar> (wchar.h)

Wide characters

This header file defines several functions to work with *C wide strings*.

Functions

Input/Output: (mostly wide versions of <cstdio> functions)

fgetwc

Get wide character from stream (function)

fgetws

Get wide string from stream (function)

fputwc

Write wide character to stream (function)

fputws

Write wide string to stream (function)

fwide

Stream orientation (function)

fwprintf

Write formatted data to stream (function)

fwscanf

Read formatted data from stream (function)

getwc

Get wide character from stream (function)

getwchar

Get wide character from stdin (function)

putwc

Write wide character to stream (function)

putwchar

Write wide character to stdout (function)

swprintf

Write formatted data to wide string (function)

swscanf

Read formatted data from string (function)

ungetwc

Unget wide character from stream (function)

vfwprintf

Write formatted data from variable argument list to stream (function)

vfwscanf

Read formatted data from stream into variable argument list (function)

vswprintf

Write formatted data from variable argument list to sized buffer (function)

vswscanf

Read formatted data from wide string into variable argument list (function)

vwprintf

Print formatted data from variable argument list to stdout (function)

vwscanf

Read formatted data into variable argument list (function)

wprintf

Print formatted data to stdout (function)

wscanf

Read formatted data from stdin (function)

General utilities: (wide versions of <cstdlib> functions)

wcstod

Convert wide string to double (function)

wcstof

Convert wide string to float (function)

wcstol

Convert wide string to long integer (function)

wcstold

Convert wide string to long double (function)

wcstoll

Convert wide string to long long integer (function)

wcstoul

Convert wide string to unsigned long integer (function)

wcstoull

Convert wide string to unsigned long long integer (function)

Character/string conversion: (mostly extended versions of <cstdlib> functions)

btowc

Convert single byte character to wide character (function)

mbrlen

Get length of multibyte character (function)

mbrtowc

Convert multibyte sequence to wide character (function)

mbsinit

Check if initial conversion state (function)

mbsrtowcs

Convert multibyte string to wide-character string (function)

wcrtomb

Convert wide character to multibyte sequence (function)

wctob

Convert wide character to single byte (function)

wcsrtombs

Convert wide-character string to multibyte string (function)

Strings: (wide versions of <cstring> functions)

wcscat

Concatenate wide strings (function)

wcschr

Locate first occurrence of character in wide string (function)

wcscmp

Compare two strings (function)

wcscoll

Compare two wide strings using locale (function)

wcscpy

Copy wide string (function)

wcscspn

Get span until character in wide string (function)

wcslen

Get wide string length (function)

wcsncat

Append characters from wide string (function)

wcsncmp

Compare characters of two wide strings (function)

wcsncpy

Copy characters from wide string (function)

wcspbrk

Locate characters in wide string (function)

wcsrchr

Locate last occurrence of character in wide string (function)

wcsspn

Get span of character set in wide string (function)

wcsstr

Locate substring of wide string (function)

wcstok

Split wide string into tokens (function)

wcsxfrm

Transform wide string using locale (function)

wmemchr

Locate character in block of wide characters (function)

wmemcmp

Compare two blocks of wide characters (function)

wmemcpy

Copy block of wide characters (function)

wmemmove

Move block of wide characters (function)

wmemset

Fill array of wide characters (function)

Time: (a wide version of a <ctime> function)

wcsftime

Format time as wide string (function)

Types

mbstate_t

Multibyte conversion state (type)

size_t

Unsigned integral type (type)

struct tm

Time structure (type)

wchar_t

Wide character (type)

wint_t

Wide int type (type)

Macro constants

NULL

Null pointer (macro)

WCHAR_MAX

Maximum value of wchar_t (constant)

WCHAR_MIN

Minimum value of wchar_t (constant)

WEOF

Wide end-of-file (constant)

<ctgmath> (tgmath.h)

This header defines macro functions that correspond to the functions in <math.h>, but which can take other non-*floating point* types as arguments:

Every function in <math.h> that takes at least one double as argument (except modf) is defined in <tgmath.h> as a macro with the same semantics but taking *generic parameters* instead:

Each of the arguments provided for these *generic parameters* that is of an integer type is casted to a double; Arguments of *floating-point types* are used without transformation (directly as float, double or long double).

This header automatically includes <math.h> and <complex.h>: The type-generic function can also take *complex* values if the function exists in <complex.h> (prefixed with a "c" character).

C Library - <stdarg.h>

The **stdarg.h** header defines a variable type **va_list** and three macros which can be used to get the arguments in a function when the number of arguments are not known i.e. variable number of arguments.

A function of variable arguments is defined with the ellipsis (,...) at the end of the parameter list.

Library Variables

Following is the variable type defined in the header stdarg.h –

Sr.No.	Variable & Description
1	**va_list** This is a type suitable for holding information needed by the three macros **va_start(), va_arg()** and **va_end()**.

Library Macros

Following are the macros defined in the header stdarg.h –

Sr.No.	Macro & Description
1	**void va_start(va_list ap, last_arg)** This macro initializes **ap** variable to be used with

	the **va_arg** and **va_end** macros. The **last_arg** is the last known fixed argument being passed to the function i.e. the argument before the ellipsis.
2	**type va_arg(va_list ap, type)** This macro retrieves the next argument in the parameter list of the function with type **type**.
3	**void va_end(va_list ap)** This macro allows a function with variable arguments which used the **va_start** macro to return. If **va_end** is not called before returning from the function, the result is undefined.

Use of bool in C

The C99 standard for C language supports bool variables. Unlike C++, where no header file is needed to use bool, a header file "stdbool.h" must be included to use bool in C. If we save the below program as .c, it will not compile, but if we save it as .cpp, it will work fine.

```
int main()
{
  bool arr[2] = {true, false};
  return 0;
}
```

If we include the header file "stdbool.h" in the above program, it will work fine as a C program.

```
#include <stdbool.h>
int main()
{
  bool arr[2] = {true, false};
  return 0;
}
```