C Reference Card (ANSI, C99)

Keywords

`auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, inline`$^{(C99)}$`, int, long, register, restrict`$^{(C99)}$`, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while`

Standard libraries

`<assert.h> <complex.h>`$^{(C99)}$ `<ctype.h> <errno.h> <fenv.h>`$^{(C99)}$ `<float.h> <inttypes.h>`$^{(C99)}$ `<iso646.h> <limits.h> <locale.h> <math.h> <setjmp.h> <signal.h> <stdarg.h> <stdbool.h>`$^{(C99)}$ `<stddef.h> <stdint.h>`$^{(C99)}$ `<stdio.h> <stdlib.h> <string.h> <tgmath.h>`$^{(C99)}$ `<time.h> <wchar.h>`$^{(C99)}$ `<wctype.h>`$^{(C99)}$

Asserts `assert.h`

`assert`	abort the program if *cond* is not true;
`(cond)`	skipped if defined `NDEBUG`

Complex numbers `complex.h`$^{(C99)}$

Types

`imaginary`	imaginary type, use as `float imaginary`, `double imaginary`, `long double imaginary`
`complex`	complex type, use with float types as `imaginary`

Constants

`I`	the complex or imaginary unit constant i

Functions

`creal, crealf, creall`	real part
`cimag`$^{(*)}$	imaginary part

$^{(*)}$ also presented for `float` and `long double`

`cabs`$^{(*)}$	magnitude
`carg`$^{(*)}$	phase angle
`conj`$^{(*)}$	complex conjugate
`cproj`$^{(*)}$	projection of Riemann sphere
`cexp`$^{(*)}$	complex exponential
`clog`$^{(*)}$	complex natural logarithm
`cpow`$^{(*)}$	complex power
`csqrt`$^{(*)}$	complex square root
`csin`$^{(*)}$	complex sine
`ccos`$^{(*)}$	complex cosine
`ctan`$^{(*)}$	complex tangent
`casin`$^{(*)}$	complex arc sine
`cacos`$^{(*)}$	complex arc cosine
`catan`$^{(*)}$	complex arc tangent
`csinh`$^{(*)}$	complex hyperbolic sine
`ccosh`$^{(*)}$	complex hyperbolic cosine
`ctanh`$^{(*)}$	complex hyperbolic tangent
`casinh`$^{(*)}$	complex arc hyperbolic sine
`cacosh`$^{(*)}$	complex arc hyperbolic cosine
`catanh`$^{(*)}$	complex arc hyperbolic tangent

Character class tests `ctype.h`

`isalnum`	alphanumeric?
`isalpha`	alphabetic?
`islower`	lower case letter?
`isupper`	upper case letter?
`isdigit`	decimal digit?
`isxdigit`	hexadecimal digit?
`iscntrl`	control character?
`isgraph`	printing character (not incl space)?
`isspace`	space, formfeed, newline, cr, tab, vtab?
`isblank`$^{(C99)}$	blanc character?
`isprint`	printing character (incl space)?
`ispunct`	printing char except space, letter, digit?
`tolower`	convert to lower case
`toupper`	convert to upper case

Error handling `errno.h`

Macros

`errno`	error number
`E2BIG, EACCES, ..., EXDEV`	standard POSIX-compatible error conditions

Floating point environment `fenv.h`$^{(C99)}$

Types

`fenv_t`	entire floating-point environment
`fexcept_t`	all floating-point status flags collectively

Functions

`feclearexcept`	clear the specified FP status flags
`fetestexcept`	determine which of the specified FP status flags are set
`feraiseexcept`	raise the specified FP exceptions
`fegetexceptflag, fesetexceptflag`	copy the state of the specified FP status flags from or to the FP env
`fegetround, fesetround`	get or set rounding direction
`fegetenv, fesetenv`	save or restore the current FP env
`feholdexcept`	save the env, clear all status flags and ignore all future errors
`feupdateenv`	restore the FP env and raise the previously raised exceptions

Macros

`FE_ALL_EXCEPT, FE_DIVBYZERO, FE_INEXACT, FE_INVALID, FE_OVERFLOW, FE_UNDERFLOW`	FP exceptions
`FE_DOWNWARD, FE_TONEAREST, FE_TOWARDZERO, FE_UPWARD`	rounding direction
`FE_DFL_ENV`	default FP env

Float type limits `float.h`

`FLT_RADIX`	the radix (integer base) used by the representation of all floating-point types
`DECIMAL_DIG`$^{(C99)}$	decimal precision required to (de)serialize long double
`FLT_MIN, DBL_MIN, LDBL_MIN`	minimum normalized positive `float`, `double`, `long double` value
`FLT_MAX`$^{(*)}$	maximum finit value of `float`, `double`, `long double`

$^{(*)}$ also presented for `double` and `long double`

`FLT_EPSILON`$^{(*)}$	smallest x so $1.0\text{f} + x \neq 1.0\text{f}$
`FLT_DIG`$^{(*)}$	number of decimal digits that are guaranteed to be preserved in text - float - text roundtrip
`FLT_MANT_DIG`$^{(*)}$	number of base-`FLT_RADIX` digits that are in the floating-point mantissa
`FLT_MIN_EXP`$^{(*)}$	minimum exponent
`FLT_MIN_10_EXP`$^{(*)}$	minimum exponent
`FLT_MAX_EXP`$^{(*)}$	maximum exponent
`FLT_MAX_10_EXP`$^{(*)}$	maximum exponent
`FLT_ROUNDS`	floating point rounding mode
`FLT_EVAL_METHOD`$^{(C99)}$	specifies in what precision all arithmetic operations are done

Integer Types `inttypes.h`$^{(C99)}$

Types

`imaxdiv_t`	struct, contains quot and rem (result of division)

Functions

imaxabs	absolute value
imaxdiv	division, returns imaxdiv_t
strtoimax	string to integer
strtoumax	string to unsigned integer
wcstoimax	wide characters to integer
wcstoumax	wide characters to unsogned integer

ISO 646 iso646.h$^{(C99)}$

Macros

and	&&
and_eq	&=
bitand	&
bitor	|
compl	~
not	!
not_eq	!=
or	||
or_eq	|=
xor	^
xor_eq	^=

Integer Type Limits limits.h

CHAR_BIT	bits in char
MB_LEN_MAX	maximum number of bytes in a multibyte character
CHAR_MIN	min value of char
CHAR_MAX	max value of char
SCHAR_MIN, SHRT_MIN, INT_MIN, LONG_MIN, LLONG_MIN$^{(C99)}$	minimum value for signed types
SCHAR_MAX, SHRT_MAX, INT_MAX, LONG_MAX, LLONG_MAX$^{(C99)}$	maximum value for signed types
UCHAR_MAX, USHRT_MAX, UINT_MAX, ULONG_MAX, ULLONG_MAX$^{(C99)}$	maximum value for unsigned types

Localization locale.h

Types

lconv	formatting details, returned by localeconv

Constants

NULL	implementation-defined null pointer constant
LC_ALL, LC_COLLATE, LC_CTYPE, LC_MONETARY, LC_NUMERIC, LC_TIME	locale categories for setlocale

Functions

setlocale	gets and sets the current C locale
localeconv	queries numeric and monetary formatting details of the current locale

Mathematical Functions math.h

Types

float_t$^{(C99)}$	most efficient floating-point type at least as wide as float
double_t$^{(C99)}$	most efficient floating-point type at least as wide as double

Constants

HUGE_VALF$^{(C99)}$, HUGE_VAL, HUGE_VALL$^{(C99)}$	indicates value too big to be representable (infinity)
INFINITY$^{(C99)}$	evaluates to positive infinity or the value guaranteed to overflow a float
NAN$^{(C99)}$	evaluates to a quiet NaN of type float
FP_FAST_FMA$^{(C99)}$, FFP_FAST_FMA$^{(C99)}$, FP_FAST_FMAL$^{(C99)}$	indicates that the fma function generally executes about as fast as, or faster than, a multiply and an add of double operands
FP_ILOGB0$^{(C99)}$, FP_ILOGBNAN$^{(C99)}$	evaluates to ilogb(x) if x is zero or NaN, respectively
math_errhandling$^{(C99)}$, MATH_ERRNO$^{(C99)}$, MATH_ERREXCEPT$^{(C99)}$	defines the error handling mechanism used by the common mathematical functions
FP_NORMAL$^{(C99)}$, FP_SUBNORMAL$^{(C99)}$, FP_ZERO$^{(C99)}$, FP_INFINITE$^{(C99)}$, FP_NAN$^{(C99)}$	indicates a floating-point category

Functions

fabs, fabsf$^{(C99)}$, fabsl$^{(C99)}$	absolute value
fmod$^{(*)(C99)}$	remainder of division
remainder$^{(C99)}$ $^{(*)(C99)}$	signed remainder of division
remquo$^{(C99)}$ $^{(*)(C99)}$	signed remainder as well as the three last bits of the division
fma$^{(C99)}$ $^{(*)(C99)}$	fused multiply-add operation $x*y+z$
fmax$^{(C99)}$ $^{(*)(C99)}$	determines larger of two values
fmin$^{(C99)}$ $^{(*)(C99)}$	determines smaller of two values
fdim$^{(C99)}$ $^{(*)(C99)}$	positive difference of two floating-point values $max(0, x-y)$
nan$^{(C99)}$ $^{(*)(C99)}$	returns a NaN (not-a-number)
exp$^{(*)(C99)}$	e^x
exp2$^{(C99)}$ $^{(*)(C99)}$	2^x
expm1$^{(C99)}$ $^{(*)(C99)}$	$e^x - 1$
log$^{(*)(C99)}$	natural (base-e) logarithm lnx
log10$^{(*)(C99)}$	common (base-10) logarithm $log_{10}x$
log2$^{(C99)}$ $^{(*)(C99)}$	base-2 logarithm log_2x
log1p$^{(C99)}$ $^{(*)(C99)}$	$ln(1+x)$
pow$^{(*)(C99)}$	x^y
sqrt$^{(*)(C99)}$	$\sqrt{x}$
cbrt$^{(C99)}$ $^{(*)(C99)}$	$\sqrt[3]{x}$
hypot$^{(C99)}$ $^{(*)(C99)}$	$\sqrt{x^2+y^2}$
sin$^{(*)(C99)}$	$\sin x$
cos$^{(*)(C99)}$	$\cos x$
tan$^{(*)(C99)}$	$\tan x$
asin$^{(*)(C99)}$	$\arcsin x$
acos$^{(*)(C99)}$	$\arccos x$
atan$^{(*)(C99)}$	$\arctan x$
atan2$^{(*)(C99)}$	$\arctan x$ using signs to detect quadrants
sinh$^{(*)(C99)}$	$\sinh x$
cosh$^{(*)(C99)}$	$\cosh x$
tanh$^{(*)(C99)}$	$\tanh x$
asinh$^{(C99)}$ $^{(*)(C99)}$	$\text{arcsinh}\ x$
acosh$^{(C99)}$ $^{(*)(C99)}$	$\text{arccosh}\ x$
atanh$^{(C99)}$ $^{(*)(C99)}$	$\text{arctanh}\ x$
erf$^{(C99)}$ $^{(*)(C99)}$	error function
erfc$^{(C99)}$ $^{(*)(C99)}$	complementary error function
tgamma$^{(C99)}$ $^{(*)(C99)}$	gamma function
lgamma$^{(C99)}$ $^{(*)(C99)}$	natural logarithm of gamma function
ceil$^{(*)(C99)}$	smallest integer not less than the given value
floor$^{(*)(C99)}$	largest integer not greater than the given value
trunc$^{(C99)}$ $^{(*)(C99)}$	nearest integer not greater in magnitude
round$^{(C99)}$ $^{(*)(C99)}$, lround$^{(C99)}$ $^{(*)(C99)}$, llround$^{(C99)}$ $^{(*)(C99)}$	rounds to nearest integer, rounding away from zero in halfway cases

$^{(*)(C99)}$ also presented for float and long double, added in C99

nearbyint$^{(C99)}$ $^{(*)(C99)}$	round to an integer using current rounding mode
rint$^{(*)(C99)}$, lrint$^{(*)(C99)}$, llrint$^{(*)(C99)}$	round to an integer using current rounding mode with exception if the result differs
frexp$^{(*)(C99)}$	break a number into significand and a power of 2
ldexp$^{(*)(C99)}$	multiply a number by 2 raised to a power
modf$^{(*)(C99)}$	break a number into integer and fractional parts
scalbn$^{(C99)}$ $^{(*)(C99)}$, scalbln$^{(C99)}$ $^{(*)(C99)}$	compute efficiently a number times FLT_RADIX raised to a power
ilogb$^{(C99)}$ $^{(*)(C99)}$	extract exponent of the given number
logb$^{(C99)}$ $^{(*)(C99)}$	extract exponent of the given number
nextafter$^{(C99)}$ $^{(*)(C99)}$, nexttoward$^{(C99)}$ $^{(*)(C99)}$	next representable floating-point value towards the given value
copysign$^{(C99)}$ $^{(*)(C99)}$	value with the magnitude of a given value and the sign of another given value
fpclassify$^{(C99)}$	classify the given floating-point value
isfinite$^{(C99)}$	given number has finite value?
isinf$^{(C99)}$	number is infinite?
isnan$^{(C99)}$	number is NaN?
isnormal$^{(C99)}$	number is normal?
signbit$^{(C99)}$	number is negative?
isgreater$^{(C99)}$	first argument is greater than second?
isgreaterequal$^{(C99)}$	first argument is greater or equal than second?
isless$^{(C99)}$	first argument is less than second?
islessequal$^{(C99)}$	first argument is less or equal than second?
islessgreater$^{(C99)}$	first argument is less or greater than second?
isunordered$^{(C99)}$	two values are unordered?

Program Support Utilities `setjmp.h`

Types

jmp_buf	execution context type

Functions

setjmp	save context
longjmp	jump to specified location

Program Support `signal.h`

Types

sig_atomic_t	integer type that can be accessed as an atomic entity from an asynchronous signal handler

Macros

SIGABRT, SIGFPE, SIGILL,SIGINT, SIGSEGV, SIGTERM	signal types
SIG_DFL, SIG_IGN	signal handling strategies
SIG_ERR	error was encountered

Functions

signal	set signal handler for particular signal
raise	run signal handler for particular signal

Variable Argument Lists `stdarg.h`

Function definition: `type name(t1 arg1, ...)`

Types

va_list	information needed by all macros

Macros

va_start	initialize argument pointer ap, lastarg - last named argument
va_arg	access next argument
va_copy$^{(C99)}$	copy arguments
va_end	end traversal

Boolean Type `stdbool.h`

Macros

bool	boolean type definition
true	integer 1
false	integer 0

Types Support `stddef.h`

Types

ptrdiff_t	signed int, result of two pointers subtraction
size_t	unsigned int returned by sizeof, offsetof

Constants

NULL	implementation-defined null pointer constant

Macros

offsetof	byte offset from the beginning of a struct type to specified member

Integer Type Support `stdint.h`$^{(C99)}$

Types

int8_t, int16_t, int32_t, int64_t	signed int with exact width
int_fast8_t, ..., int_fast64_t	fastest signed int with width at least 8, 16, ...
int_least8_t, ..., int_least64_t	smallest signed int with width at least 8, 16, ...
intmax_t	maximum width integer type
intptr_t	integer type capable of holding a pointer
uint8_t, ..., uint64_t	unsigned int with exact width
uint_fast8_t, ..., uint_fast64_t	fastest unsigned int with width at least 8, 16, ...
uint_least8_t, ..., uint_least64_t	smallest unsigned int with width at least 8, 16, ...
uintmax_t	maximum width unsigned integer type
uintptr_t	unsigned integer type capable of holding a pointer

Constants

INT8_MIN, ..., INT64_MIN	minimum value of object of corresponding type
INT_FAST8_MIN, ..., INT_FAST64_MIN	minimum value of object of corresponding type
INT_LEAST8_MIN, ..., INT_LEAST64_MIN	minimum value of object of corresponding type
INTPTR_MIN	minimum value of intptr_t object
INTMAX_MIN	minimum value of intmax_t object
INT8_MAX, ..., INT64_MAX	maximum value of object of corresponding type
INT_FAST8_MAX, ..., INT_FAST64_MAX	maximum value of object of corresponding type
INT_LEAST8_MAX, ..., INT_LEAST64_MAX	maximum value of object of corresponding type
INTPTR_MAX	maximum value of intptr_t object
INTMAX_MAX	maximum value of intmax_t object
UINT8_MAX, ..., UINT64_MAX	maximum value of object of corresponding type
UINT_FAST8_MAX, ..., UINT_FAST64_MAX	maximum value of object of corresponding type
UINT_LEAST8_MAX, ..., UINT_LEAST64_MAX	maximum value of object of corresponding type
UINTPTR_MAX	maximum value of uintptr_t object
UINTMAX_MAX	maximum value of uintmax_t object

Function Macro

INT8_C, ..., INT64_C	expands to an int const expression with the type int_least8_t, ...
INTMAX_C	expands to an int const expression with the type intmax_t
UINT8_C, ..., UINT64_C	expands to an int const expression with the type uint_least8_t, ...
UINTMAX_C	expands to an int const expression with the type uintmax_t

Standard input/output stdio.h

Types

FILE	object type, capable of holding all information needed to control a C I/O stream
fpos_t	non-array complete object type, capable of uniquely specifying a position and multibyte parser state in a file

Predefined standard streams

stdin, stdout, stderr	expression of type FILE * associated with corresponding stream

Functions

fopen	open file, returns FILE *
freopen	open an existing stream FILE *fp with a different name, returns FILE *
fclose	close a file
fflush	synchronizes an output stream with the actual file
setbuf	sets the buffer for a file stream
setvbuf	sets the buffer and its size for a file stream
fread	reads from a file count objects of size size to buffer
fwrite	writes to a file count objects of size size from buffer
fgetc, getc	gets a character from a file stream
fgets	gets a character string with length count - 1 from a file stream
fputc, putc	writes a character to a file stream
fputs	writes a character string to a file stream
getchar	reads a character from stdin, equivalent to getc (stdin)
gets	reads a character string from stdin until newline or EOF
putchar	writes a character to stdout, equivalent to putc (ch, stdout)
puts	writes a character string + \n to stdout
ungetc	puts a character back into a file stream
scanf, fscanf, sscanf	reads formatted input from stdin, a file stream or a buffer
vscanf$^{(C99)}$, vfscanf$^{(C99)}$, vsscanf$^{(C99)}$	reads formatted input from stdin, a file stream or a buffer using variable argument list
printf, fprintf, sprintf, snprintf$^{(C99)}$	prints formatted output to stdout, a file stream or a buffer
vprintf, vfprintf, vsprintf, vsnprintf$^{(C99)}$	prints formatted output to stdout, a file stream or a buffer using variable argument list
ftell	returns the current file position indicator, useful for fseek
fgetpos	gets the file position indicator, useful for fsetpos
fseek	moves the file position indicator to a specific location in a file, origin values: SEEK_SET, SEEK_CUR, SEEK_END
fsetpos	moves the file position indicator to a specific location in a file
rewind	moves the file position indicator to the beginning in a file
clearerr	clears errors
feof	checks for the end-of-file
ferror	checks for a file error
perror	displays a character string corresponding of the current error to stderr
remove	erases a file
rename	renames a file
tmpfile	returns a pointer to a temporary file
tmpnam	returns a unique filename

Macro constants

EOF	integer constant expression of type int and negative value
FOPEN_MAX	maximum number of files that can be open simultaneously
FILENAME_MAX	size needed for an array of char to hold the longest supported file name
BUFSIZ	size of the buffer used by setbuf
_IOFBF, _IOLBF, _IONBF	argument to setvbuf indicating fully buffered, line buffered, unbuffered I/O
SEEK_SET, SEEK_CUR, SEEK_END	argument to fseek indicating seeking from beginning, current position, end of the file
TMP_MAX	maximum number of unique filenames that can be generated by tmpnam
L_tmpnam	size needed for an array of char to hold the result of tmpnam

Standard library stdlib.h

Macros

EXIT_SUCCESS, EXIT_FAILURE	indicates program execution execution status
MB_CUR_MAX	maximum number of bytes in a multibyte character, in the current locale
RAND_MAX	maximum possible value generated by rand()

Functions

abort	causes abnormal program termination (without cleaning up)
exit	causes normal program termination with cleaning up
_Exit$^{(C99)}$	causes normal program termination without cleaning up
atexit	registers a function to be called on exit() invocation
system	calls the host environment's command processor
getenv	access to the list of environment variables
malloc	allocates memory
calloc	allocates and zeroes memory
realloc	expands previously allocated memory block
free	deallocates previously allocated memory
atof	converts a byte string to a floating-point value
atoi, atol, atoll$^{(C99)}$	converts a byte string to an integer value
strtol, strtoll$^{(C99)}$	converts a byte string to an integer value
strtoul, strtoull$^{(C99)}$	converts a byte string to an unsigned integer value
strtof$^{(C99)}$, strtod, strtold$^{(C99)}$	converts a byte string to a floating-point value
mblen	returns the number of bytes in the next multibyte character

`mbtowc`	converts the next multibyte character to wide character
`wctomb`	converts a wide character to its multibyte representation
`mbstowcs`	converts a narrow multibyte character string to wide string
`wcstombs`	converts a wide string to narrow multibyte character string
`rand`	generates a pseudo-random number
`srand`	seeds pseudo-random number generator
`qsort`	sorts a range of elements with unspecified type
`bsearch`	searches an array for an element of unspecified type

NULL-terminated strings `string.h`

Macros

`NULL`	implementation-defined null pointer constant

Types

`size_t`	unsigned integer type returned by the sizeof operator

Functions

`strcpy`	copies one string to another
`strncpy`	copies a certain amount of characters from one string to another
`strcat`	concatenates two strings
`strncat`	concatenates a certain amount of characters of two strings
`strxfrm`	transform a string so that strcmp would produce the same result as strcoll
`strlen`	returns the length of a given string
`strcmp`	compares two strings
`strncmp`	compares a certain amount of characters of two strings
`strcoll`	compares two strings in accordance to the current locale
`strchr`	finds the first occurrence of a character
`strrchr`	finds the last occurrence of a character
`strspn`	returns the length of the maximum initial segment that consists of only the characters found in another byte string
`strcspn`	returns the length of the maximum initial segment that consists of only the characters not found in another byte string
`strpbrk`	finds the first location of any character in one string, in another string
`strstr`	finds the first occurrence of a substring of characters
`strtok`	finds the next token in a byte string
`memchr`	searches an array for the first occurrence of a character
`memcmp`	compares two buffers
`memset`	fills a buffer with a character
`memcpy`	copies one buffer to another
`memmove`	moves one buffer to another
`strerror`	returns a text version of a given error code

Type generic math `tgmath.h`$^{(C99)}$

Determines, which real or complex function to call for the provided arguments

Date and time utilities `time.h`

Types

`tm`	calendar time type
`time_t`	calendar time since epoch type
`clock_t`	processor time since era type

Constants

`CLOCKS_PER_SEC`	number of processor clock ticks per second

Functions

`difftime`	computes the difference between times
`time`	returns the current calendar time of the system as time since epoch
`clock`	returns raw processor clock time since the program is started
`asctime`	converts a tm object to a textual representation
`ctime`	converts a time_t object to a textual representation
`strftime`	converts a tm object to custom textual representation
`gmtime`	converts time since epoch to calendar time expressed as Coordinated Universal Time (UTC)
`localtime`	converts time since epoch to calendar time expressed as local time
`mktime`	converts calendar time to time since epoch

Wide character support `wchar.h`$^{(C99)}$

Types

`mbstate_t`	conversion state information necessary to iterate multibyte character strings
`wchar_t`	integer type that can hold any valid wide character
`wint_t`	integer type that can hold any valid wide character and at least one more value

Macros

`WEOF`	a non-character value of type `wint_t` used to indicate errors
`WCHAR_MIN`	the smallest valid value of `wchar_t`
`WCHAR_MAX`	the largest valid value of `wchar_t`

Functions

`fwide`	switches a file stream between wide character I/O and narrow character I/O
`fgetwc`, `getwc`	gets a wide character from a file stream
`fgetws`	gets a wide string of length `count - 1` from a file stream
`fputwc`, `putwc`	writes a wide character to a file stream
`fputws`	writes a wide string to a file stream
`getwchar`	reads a wide character from stdin
`putwchar`	writes a wide character to stdout
`ungetwc`	puts a wide character back into a file stream
`wscanf`, `fwscanf`, `swscanf`	reads formatted wide character input from stdin, a file stream or a buffer
`vwscanf`, `vfwscanf`, `vswscanf`	reads formatted wide character input from stdin, a file stream or a buffer using variable argument list
`wprintf`, `fwprintf`, `swprintf`	prints formatted wide character output to stdout, a file stream or a buffer
`vwprintf`, `vfwprintf`, `vswprintf`	prints formatted wide character output to stdout, a file stream or a buffer using variable argument list
`mbsinit`	checks if the mbstate_t object represents initial shift state
`btowc`	widens a single-byte narrow character to wide character, if possible
`wctob`	narrows a wide character to a single-byte narrow character, if possible
`mbrlen`	returns the number of bytes in the next multibyte character, given state
`mbrtowc`	converts the next multibyte character to wide character, given state
`wcrtomb`	converts a wide character to its multibyte representation, given state
`mbsrtowcs`	converts a narrow multibyte character string to wide string, given state
`wcsrtombs`	converts a wide string to narrow multibyte character string, given state

`wcstol`, `wcstoll`	converts a wide string to an integer value
`wcstoul`, `wcstoull`	converts a wide string to an unsigned integer value
`wcstof`, `wcstod`, `wcstold`	converts a wide string to a floating-point value
`wcscpy`	copies one wide string to another
`wcsncpy`	copies a certain amount of wide characters from one string to another
`wcscat`	appends a copy of one wide string to another
`wcsncat`	appends a certain amount of wide characters from one wide string to another
`wcsxfrm`	transform a wide string so that `wcscmp` would produce the same result as `wcscoll`
`wcslen`	returns the length of a wide string
`wcscmp`	compares two wide strings
`wcsncmp`	compares a certain amount of characters from two wide strings
`wcscoll`	compares two wide strings in accordance to the current locale
`wcschr`	finds the first occurrence of a wide character in a wide string
`wcsrchr`	finds the last occurrence of a wide character in a wide string
`wcsspn`	returns the length of the maximum initial segment that consists of only the wide characters found in another wide string
`wcscspn`	returns the length of the maximum initial segment that consists of only the wide chars not found in another wide string
`wcspbrk`	finds the first location of any wide character in one wide string, in another wide string
`wcsstr`	finds the first occurrence of a wide string within another wide string
`wcstok`	finds the next token in a wide string
`wmemcpy`	copies a certain amount of wide characters between two non-overlapping arrays
`wmemmove`	copies a certain amount of wide characters between two, possibly overlapping, arrays
`wmemcmp`	compares a certain amount of wide characters from two arrays
`wmemchr`	finds the first occurrence of a wide character in a wide character array
`wmemset`	copies the given wide character to every position in a wide character array

Wide character support `wctype.h`$^{(C99)}$

Types

`wint_t`	integer type that can hold any valid wide character and at least one more value
`wctrans_t`	scalar type that holds locale-specific character mapping
`wctype_t`	scalar type that holds locale-specific character classification

Macros

`WEOF`	a non-character value of type `wint_t` used to indicate errors

Functions

`iswalnum`	checks if a wide character is alphanumeric
`iswalpha`	checks if a wide character is alphabetic
`iswlower`	checks if a wide character is an lowercase character
`iswupper`	checks if a wide character is an uppercase character
`iswdigit`	checks if a wide character is a digit
`iswxdigit`	checks if a wide character is a hexadecimal character
`iswcntrl`	checks if a wide character is a control character
`iswgraph`	checks if a wide character is a graphical character
`iswspace`	checks if a wide character is a space character
`iswblank`	checks if a wide character is a blank character
`iswprint`	checks if a wide character is a printing character
`iswpunct`	checks if a wide character is a punctuation character
`iswctype`	classifies a wide character according to the specified texttttLC_CTYPE category
`wctype`	looks up a character classification category in the current C locale
`towlower`	converts a wide character to lowercase
`towupper`	converts a wide character to uppercase
`towctrans`	performs character mapping according to the specified `LC_CTYPE` mapping category
`wctrans`	looks up a character mapping category in the current C locale