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 (cond)`	abort the program if *cond* is not true; 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 (z), crealf, creall`	real part
`cimag (z)` (*)	imaginary part
`cabs (z)` (*)	magnitude
`carg (z)` (*)	phase angle
`conj (z)` (*)	complex conjugate
`cproj (z)` (*)	projection of Riemann sphere
`cexp (z)` (*)	complex exponential
`clog (z)` (*)	complex natural logarithm
`cpow (z)` (*)	complex power
`csqrt (zb, zp)` (*)	complex square root
`csin (z)` (*)	complex sine
`ccos (z)` (*)	complex cosine
`ctan (z)` (*)	complex tangent
`casin (z)` (*)	complex arc sine
`cacos (z)` (*)	complex arc cosine
`catan (z)` (*)	complex arc tangent
`csinh (z)` (*)	complex hyperbolic sine
`ccosh (z)` (*)	complex hyperbolic cosine
`ctanh (z)` (*)	complex hyperbolic tangent
`casinh (z)` (*)	complex arc hyperbolic sine
`cacosh (z)` (*)	complex arc hyperbolic cosine
`catanh (z)` (*)	complex arc hyperbolic tangent

(*) also presented for `float` and `long double`

Character class tests `ctype.h`

`isalnum (c)`	alphanumeric?
`isalpha (c)`	alphabetic?
`islower (c)`	lower case letter?
`isupper (c)`	upper case letter?
`isdigit (c)`	decimal digit?
`isxdigit (c)`	hexadecimal digit?
`iscntrl (c)`	control character?
`isgraph (c)`	printing character (not incl space)?
`isspace (c)`	space, formfeed, newline, cr, tab, vtab?
`isblank (c)`$^{(C99)}$	blanc character?
`isprint (c)`	printing character (incl space)?
`ispunct (c)`	printing char except space, letter, digit?
`tolower (c)`	convert to lower case
`toupper (c)`	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 (int excepts)`	clear the specified FP status flags
`fetestexcept (int excepts)`	determine which of the specified FP status flags are set
`feraiseexcept (int excepts)`	raise the specified FP exceptions
`fegetexceptflag (fexcept_t* flagp, int excepts), fesetexceptflag (const fexcept_t* flagp, int excepts)`	copy the state of the specified FP status flags from or to the FP env
`fegetround (), fesetround (int round)`	get or set rounding direction
`fegetenv (fenv_t* envp), fesetenv (const fenv_t* envp)`	save or restore the current FP env
`feholdexcept (fenv_t* envp)`	save the env, clear all status flags and ignore all future errors
`feupdateenv (fenv_t* envp)`	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`
`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

(*) also presented for `double` and `long double`

`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 (intmax_t j)`	absolute value
`imaxdiv (intmax_t numer, intmax_t denom)`	division, returns `imaxdiv_t`
`strtoimax (const char* restrict nptr, char** restrict endptr, int base)`	string to integer
`strtoumax (const char* restrict nptr, char** restrict endptr, int base)`	string to unsigned integer
`wcstoimax (const wchar_t* restrict nptr, wchar_t** restrict endptr, int base)`	wide characters to integer
`wcstoumax (const wchar_t* restrict nptr, wchar_t** restrict endptr, int base)`	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(int category, const char* locale)`	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_ILOGBO`$^{(C99)}$, `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 (x)`, `fabsf`$^{(C99)}$, `fabsl`$^{(C99)}$	absolute value
`fmod (x, y)` (*)$^{(C99)}$	remainder of division

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

`remainder (x, y)` $^{(C99)}$ (*)$^{(C99)}$	signed remainder of division
`remquo (x, y, int *quo)` $^{(C99)}$ (*)$^{(C99)}$	signed remainder as well as the three last bits of the division
`fma (x, y, z)` $^{(C99)}$ (*)$^{(C99)}$	fused multiply-add operation $x * y + z$
`fmax (x, y)` $^{(C99)}$ (*)$^{(C99)}$	determines larger of two values
`fmin (x, y)` $^{(C99)}$ (*)$^{(C99)}$	determines smaller of two values
`fdim (x, y)` $^{(C99)}$ (*)$^{(C99)}$	positive difference of two floating-point values $max(0, x - y)$
`nan (const char* arg)` $^{(C99)}$ (*)$^{(C99)}$	returns a NaN (not-a-number)
`exp (x)` (*)$^{(C99)}$	e^x
`exp2 (x)` $^{(C99)}$ (*)$^{(C99)}$	2^x
`expm1 (x)` $^{(C99)}$ (*)$^{(C99)}$	$e^x - 1$
`log (x)` (*)$^{(C99)}$	natural (base-e) logarithm lnx
`log10 (x)` (*)$^{(C99)}$	common (base-10) logarithm $log_{10}x$

`log2 (x)` (C99) (*)(C99)	base-2 logarithm $log_2 x$
`log1p (x)` (C99) (*)(C99)	$ln(1+x)$
`pow (x, y)` (*)(C99)	x^y
`sqrt (x)` (*)(C99)	$\sqrt{x}$
`cbrt (x)` (C99) (*)(C99)	$\sqrt[3]{x}$
`hypot (x, y)` (C99) (*)(C99)	$\sqrt{x^2+y^2}$
`sin (x)` (*)(C99)	$\sin x$
`cos (x)` (*)(C99)	$\cos x$
`tan (x)` (*)(C99)	$\tan x$
`asin (x)` (*)(C99)	$\arcsin x$
`acos (x)` (*)(C99)	$\arccos x$
`atan (x)` (*)(C99)	$\arctan x$
`atan2 (y, x)` (*)(C99)	$\arctan x$ using signs to detect quadrants
`sinh (x)` (*)(C99)	$\sinh x$
`cosh (x)` (*)(C99)	$\cosh x$
`tanh (x)` (*)(C99)	$\tanh x$
`asinh (x)` (C99) (*)(C99)	$\text{arcsinh}\ x$
`acosh (x)` (C99) (*)(C99)	$\text{arccosh}\ x$
`atanh (x)` (C99) (*)(C99)	$\text{arctanh}\ x$
`erf (x)` (C99) (*)(C99)	error function
`erfc (x)` (C99) (*)(C99)	complementary error function
`tgamma (x)` (C99) (*)(C99)	gamma function
`lgamma (x)` (C99) (*)(C99)	natural logarithm of gamma function
`ceil (x)` (*)(C99)	smallest integer not less than the given value
`floor (x)` (*)(C99)	largest integer not greater than the given value
`trunc (x)` (C99) (*)(C99)	nearest integer not greater in magnitude
`round (x)` (C99) (*)(C99), `lround` (C99) (*)(C99), `llround` (C99) (*)(C99)	rounds to nearest integer, rounding away from zero in halfway cases
`nearbyint (x)` (C99) (*)(C99)	round to an integer using current rounding mode
`rint (x)` (*)(C99), `lrint` (*)(C99), `llrint` (*)(C99)	round to an integer using current rounding mode with exception if the result differs
`frexp (arg, int* exp)` (*)(C99)	break a number into significand and a power of 2
`ldexp (arg, int exp)` (*)(C99)	multiply a number by 2 raised to a power
`modf (arg, &iptr)` (*)(C99)	break a number into integer and fractional parts
`scalbn (arg, int exp)` (C99) (*)(C99), `scalbln` (C99) (*)(C99)	compute efficiently a number times `FLT_RADIX` raised to a power
`ilogb (x)` (C99) (*)(C99)	extract exponent of the given number
`logb (x)` (C99) (*)(C99)	extract exponent of the given number
`nextafter (from, to)` (C99) (*)(C99), `nexttoward` (C99) (*)(C99)	next representable floating-point value towards the given value
`copysign (x, y)` (C99) (*)(C99)	value with the magnitude of a given value and the sign of another given value
`fpclassify (x)` (C99)	classify the given floating-point value
`isfinite (x)` (C99)	given number has finite value?
`isinf (x)` (C99)	number is infinite?
`isnan (x)` (C99)	number is NaN?
`isnormal (x)` (C99)	number is normal?
`signbit (x)` (C99)	number is negative?
`isgreater (x, y)` (C99)	first argument is greater than second?
`isgreaterequal (x, y)` (C99)	first argument is greater or equal than second?
`isless (x, y)` (C99)	first argument is less than second?
`islessequal (x, y)` (C99)	first argument is less or equal than second?
`islessgreater (x, y)` (C99)	first argument is less or greater than second?
`isunordered (x, y)` (C99)	two values are unordered?

Program Support Utilities `setjmp.h`

Types	
`jmp_buf`	execution context type
Functions	
`setjmp (jmp_buf env)`	save context
`longjmp (jmp_buf env, int status)`	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 (int sig, void (*handler)(int))`	set signal handler for particular signal
`raise (int sig)`	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 (va_list ap, lastarg)`	initialize argument pointer `ap`, `lastarg` - last named argument
`va_arg (ap, type)`	access next argument
`va_copy (va_list dest, va_list src)` (C99)	copy arguments
`va_end (ap)`	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(type, member)`	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_fast16_t, int_fast32_t, int_fast64_t	fastest signed int with width at least 8, 16, ...
int_least8_t, int_least16_t, int_least32_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, uint16_t, uint32_t, uint64_t	unsigned int with exact width
uint_fast8_t, uint_fast16_t, uint_fast32_t, uint_fast64_t	fastest unsigned int with width at least 8, 16, ...
uint_least8_t, uint_least16_t, uint_least32_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, INT16_MIN, INT32_MIN, INT64_MIN	minimum value of object of corresponding type
INT_FAST8_MIN, INT_FAST16_MIN, INT_FAST32_MIN, INT_FAST64_MIN	minimum value of object of corresponding type
INT_LEAST8_MIN, INT_LEAST16_MIN, INT_LEAST32_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, INT16_MAX, INT32_MAX, INT64_MAX	maximum value of object of corresponding type
INT_FAST8_MAX, INT_FAST16_MAX, INT_FAST32_MAX, INT_FAST64_MAX	maximum value of object of corresponding type
INT_LEAST8_MAX, INT_LEAST16_MAX, INT_LEAST32_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, UINT16_MAX, UINT32_MAX, UINT64_MAX	maximum value of object of corresponding type
UINT_FAST8_MAX, UINT_FAST16_MAX, UINT_FAST32_MAX, UINT_FAST64_MAX	maximum value of object of corresponding type
UINT_LEAST8_MAX, UINT_LEAST16_MAX, UINT_LEAST32_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 (x), INT16_C, INT32_C, INT64_C	expands to an int const expression with the type int_least8_t, ...
INTMAX_C (x)	expands to an int const expression with the type intmax_t
UINT8_C (x), UINT16_C, UINT32_C, UINT64_C	expands to an int const expression with the type uint_least8_t, ...
UINTMAX_C (x)	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 ("filename", "mode")	open file, returns FILE *fp
freopen ("filename", "mode", fp)	open an existing stream FILE *fp with a different name, returns FILE *
fclose (fp)	close a file
fflush (fp)	synchronizes an output stream with the actual file
setbuf (fp, char *buffer)	sets the buffer for a file stream
setvbuf (fp, char *buffer, int mode, size_t size)	sets the buffer and its size for a file stream
fread (void *buffer, size_t size, size_t count, fp)	reads from a file count objects of size size to buffer
fwrite (const void* buffer, size_t size, size_t count, fp)	writes to a file count objects of size size from buffer
fgetc (fp), getc (fp)	gets a character from a file stream
fgets (char *str, int count, fp)	gets a character string with length count - 1 from a file stream
fputc (int ch, fp), putc (int ch, fp)	writes a character to a file stream
fputs (char *str, fp)	writes a character string to a file stream
getchar ()	reads a character from stdin, equivalent to getc (stdin)
gets (char *str)	reads a character string from stdin until newline or EOF
putchar (int ch)	writes a character to stdout, equivalent to putc (ch, stdout)
puts (char* str)	writes a character string + \n to stdout
ungetc (int ch, fp)	puts a character back into a file stream
scanf (const char *format, ...), fscanf (fp, const char *format, ...), sscanf (const char *buffer, const char *format, ...)	reads formatted input from stdin, a file stream or a buffer

`vscanf (const char *restrict format, va_list vlist)`$^{(C99)}$, `vfscanf (fp, const char *restrict format, va_list vlist)`$^{(C99)}$, `vsscanf (const char *restrict buffer, const char *restrict format, va_list vlist)`$^{(C99)}$	reads formatted input from stdin, a file stream or a buffer using variable argument list
`printf (const char* format, ...)`, `fprintf (fp, const char* format, ...)`, `sprintf (char* buffer, const char* format, ...)`, `snprintf (char* restrict buffer, size_t bufsz, const char* restrict format, ...)`$^{(C99)}$	prints formatted output to stdout, a file stream or a buffer
`vprintf (const char* format, va_list vlist)`, `vfprintf (fp, const char* format, va_list vlist)`, `vsprintf (char* buffer, const char* format, va_list vlist)`, `vsnprintf (char* restrict buffer, size_t bufsz, const char* restrict format, va_list vlist)`$^{(C99)}$	prints formatted output to stdout, a file stream or a buffer using variable argument list
`ftell (fp)`	returns the current file position indicator, useful for `fseek`
`fgetpos (fp, fpos_t* pos)`	gets the file position indicator, useful for `fsetpos`
`fseek (fp, long offset, int origin)`	moves the file position indicator to a specific location in a file, origin values: `SEEK_SET`, `SEEK_CUR`, `SEEK_END`
`fsetpos (fp, const fpos_t* pos)`	moves the file position indicator to a specific location in a file
`rewind (fp)`	moves the file position indicator to the beginning in a file
`clearerr(fp)`	clears errors
`feof (fp)`	checks for the end-of-file
`ferror (fp)`	checks for a file error
`perror (const char *s)`	displays a character string corresponding of the current error to stderr
`remove (const char* pathname)`	erases a file
`rename (const char* old_filename, const char* new_filename)`	renames a file
`tmpfile ()`	returns a pointer to a temporary file
`tmpnam (char *filename)`	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

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

Functions

`fwide (fp, int mode)`	switches a file stream between wide character I/O and narrow character I/O
`fgetwc (fp)`, `getwc (fp)`	gets a wide character from a file stream
`fgetws (wchar_t* str, int count, fp)`	gets a wide string of length `count - 1` from a file stream
`fputwc (wchar_t ch, fp)`, `putwc (wchar_t ch, fp)`	writes a wide character to a file stream
`fputws (wchar_t *str, fp)`	writes a wide string to a file stream
`getwchar ()`	reads a wide character from stdin
`putwchar (wchar_t ch)`	writes a wide character to stdout
`ungetwc (wint_t ch, fp)`	puts a wide character back into a file stream
`wscanf (const wchar_t *format, ...)`, `fwscanf (fp, const wchar_t *format, ...)`, `swscanf (const wchar_t *buffer, const wchar_t *format, ...)`	reads formatted wide character input from stdin, a file stream or a buffer
`vwscanf (const wchar_t *restrict format, va_list vlist)`, `vfwscanf (fp, const wchar_t *restrict format, va_list vlist)`, `vswscanf (const wchar_t *restrict buffer, const wchar_t *restrict format, va_list vlist)`	reads formatted wide character input from stdin, a file stream or a buffer using variable argument list

`wprintf (const wchar_t* format, ...), fwprintf (fp, const wchar_t* format, ...), swprintf (wchar_t* buffer, size_t bufsz, const wchar_t* format, ...)`	prints formatted wide character output to stdout, a file stream or a buffer
`vwprintf (const wchar_t* format, va_list vlist), vfwprintf (fp, const wchar_t* format, va_list vlist), vswprintf (wchar_t* buffer, size_t bufsz, const wchar_t* format, va_list vlist)`	prints formatted wide character output to stdout, a file stream or a buffer using variable argument list

Integer Type Limits `limits.h`

Functions

`CHAR_BIT`	bits in `char`

Integer Type Limits `limits.h`

Functions

`CHAR_BIT`	bits in `char`