Cheat Sheets of the C standard library

Version 1.06 Last updated: 2012-11-28

About

This document is a set of quick reference sheets (or 'cheat sheets') of the ANSI C standard library. It contains function and macro declarations in every header of the library, as well as notes about their usage.

This document covers C++, but does not cover the C99 or C11 standard. A few non-ANSI-standard functions, if they are interesting enough, are also included.

Style used in this document

`Function names, prototypes and their parameters are in monospace.`

Remarks of functions and parameters are marked italic and enclosed in '/' and '*/' like C comments.*

`Data types, whether they are built-in types or provided by the C standard library, are also marked in monospace.` **`Types of parameters and return types are in bold.`**

Type modifiers, like '`const`' and '`unsigned`', have smaller font sizes in order to save space.

Macro constants are marked using proportional typeface, uppercase, and no italics, LIKE_THIS_ONE. One exception is L_tmpnum, which is the constant that uses lowercase letters.

Example:

```
int system      ( const char * command );
 /* system: The value returned depends on the running environment. Usually 0 when executing successfully.
     If command == NULL, system returns whether the command processor exists (0 if not). */
```

License

References

Cplusplus.com – The C++ Resources Network
http://cplusplus.com/reference/clibrary/
(↑ Most of the information in this document is from here.)

The Open Group Base Specifications (Single UNIX Specification)
http://pubs.opengroup.org/onlinepubs/9699919799/

C Runtime Library reference in MSDN
http://msdn.microsoft.com/en-us/library/634ca0c2.aspx

Wikipedia
http://en.wikipedia.org/wiki/C_Standard_Library

Linux man pages
http://linux.die.net/man/

The C Library Reference Guide by Eric Huss
http://www.acm.uiuc.edu/webmonkeys/book/c_guide/index.html

C++ Reference (cppreference.com)
http://en.cppreference.com/w/cpp/io/c

Dinkumware's Libraries Reference
http://www.dinkumware.com/manuals/default.aspx

scanf and printf formats

`%[flags][width][.precision][length]type`

Type – usually for integers:

		example
%d	**D**ecimal (signed)	-12345
%i	scanf: Signed **i**nt, but allows octal, decimal, and hexadecimal input, depending on the prefix.	
%u	Decimal (**u**nsigned)	53191
%o	**O**ctal	147707
%x %X	He**x**adecimal	cfc7

Type – usually for floating points:

%f	**F**ixed-point notation	123000.00
%e %E	**E**xponential notation	1.23e+005
%g %G	%f or %e, whichever is shorter	

Type – usually for text:

%c	Print a **C**haracter	
%s	**S**tring	
%[] %[^]	scanf: Scans only the characters in the set. (%[^] excludes them instead)	%[aeiou] %[^12345]

Type – special:

%%	Single ‘**%**’ character	
%n	Reads and prints nothing, but outputs the **n**umber of characters read/printed so far. (Argument must be an int*)	
%p	**P**ointer address	

Note

For printf, * can be used in the field width or precision (or both). In that case the function takes an additional int argument – preceding the argument to be formatted – to specify the width or precision. (takes 2 arguments if both are *, like %*.*f)

Flags (for printf only, except for the * flag)

%-4d	12	Left-justify the field instead of right.
%+d	+12	Always prepends the sign (+-).
% d (space)	12	Inserts a space if there’s no sign.
%#o %#X	014 0xC	(For o, x, X) Precedes value with ‘0’ or ‘0x’.
%#.0f %#.0e	12000034. 1.e+007	(For f, e) Prints the decimal point even if no digits follow.
%#.3g	1.00e+007	(For g) Keeps trailing zeros, along with decimal point.
%04d	0012	Pads the field with zeros instead of spaces.
%*c		scanf: Retrieves the data but discards it.

Field width

scanf: Maximum number of characters to be read.
printf: Minimum number of characters to be printed.

Precision (for printf only)

%.4d	0012	(For d, u, o, x, X) Minimum number of digits to be printed.
%.4f %.3e	12.3400 1.234e+001	(For f, e, E) Number of digits after the decimal point.
%.4g	12.34	(For g, G) Maximum number of significant digits.
%.4s	Prec	(For s) Maximum number of characters to be printed.

Length

%hd %hf	s**h**ort (i.e. **h**alf length)
%ld %lf	**l**ong (For long double, use %Lf .)

strftime formats

			example
%Y	Year		2001
%y	Year, last two digits	(00–99)	01
%B	Full month name	*[locale-dependant]*	August
%b	Abbreviated month name	*[locale-dependant]*	Aug
%m	Month as a decimal number	(01–12)	08
%U	Week number with Sunday as the first day of week	(00–53)	33
%W	Week number with Monday as the first day of week	(00–53)	34
%d	Day of the month	(01–31)	23
%j	Day of the year	(001–366)	235
%A	Full weekday name	*[locale-dependant]*	Thursday
%a	Abbreviated weekday name	*[locale-dependant]*	Thu
%w	Weekday as a decimal number with Sunday as 0	(0–6)	4
%Z	Timezone name or abbreviation		CDT
%p	AM or PM designation		PM
%I	Hour in 12h format	(01–12)	02
%H	Hour in 24h format	(00–23)	14
%M	Minute	(00–59)	55
%S	Second	(00–61)	02
%x	Date representation	*[locale-dependant]*	08/23/01
%X	Time representation	*[locale-dependant]*	14:55:02
%c	Date and time representation	*[locale-dependant]*	Thu Aug 23 14:55:02 2001
%%	Single ‘**%**’ character		%

cstdio <stdio.h> functions

File access:

```
FILE * fopen   ( const char * filename, const char * mode );
FILE * freopen ( const char * filename, const char * mode, FILE * stream );
```

/ mode parameter:* "r|w|a[b][+]" *(meaning: read/write/append, binary, for update)*
Examples: "rb+", "wb". *Note that "write"* ***erases*** *the file content.*
The system supports at least FOPEN_MAX *files open simultaneously. (*stdin*,* stdout*, and* stderr *included.) */*

```
int fclose     ( FILE * stream );

void setbuf    ( FILE * stream, char * buffer );       /* buffer must have at least BUFSIZ bytes. */
int setvbuf    ( FILE * stream, char * buffer, int mode, size_t size );
int fflush     ( FILE * stream );
```

*/** fclose, setvbuf*, and* fflush *return* 0 *on success.*
mode *parameter:* _IOFBF *(Full buffering),* _IOLBF *(Line buffering),* _IONBF *(No buffering) */*

Formatted input/output:

```
int fscanf     ( FILE * stream,   const char * format %!, ... );
int fprintf    ( FILE * stream,   const char * format %!, ... );
int scanf      (                  const char * format %!, ... ); stdin
int printf     (                  const char * format %!, ... ); stdout
int sscanf     ( const char * str, const char * format %!, ... );
int sprintf    ( char * str,      const char * format %!, ... ); ⚠

int vfprintf   ( FILE * stream,   const char * format %!, va_list arg );
int vprintf    (                  const char * format %!, va_list arg ); stdout
int vsprintf   ( char * str,      const char * format %!, va_list arg ); ⚠
```

⚠: No buffer overflow protection (bound checking). Security issues may occur.

%!: Be careful of format string attacks.

*/** scanf *functions: return the number of items read, or* EOF *if error occurs.*
printf *functions: return the number of characters written, or a negative value if error occurs. */*

Character input/output:

```
int fgetc      (                          FILE * stream );                  /* Alias: getc */
int fputc      ( int character,           FILE * stream );                  /* Alias: putc */
char * fgets   ( char * str, int length,  FILE * stream ); /* length includes terminating '\0'. */
int fputs      ( const char * str,        FILE * stream );
int getchar    ( void ); stdin
int putchar    ( int character ); stdout
char * gets    ( char * str ); stdin ⚠ (Deprecated)     /* Unlike fgets, gets does not scan the '\n'! */
int puts       ( const char * str ); stdout                          /* Appends a '\n' at the end! */

int ungetc     ( int character,           FILE * stream );
```

*/** fgetc, getchar, fputc, putchar*, and* ungetc*: return the same character read/written, as an* int.
fputs *and* puts*: return a non-negative value.*   fgets *and* gets*: return* str.
All return EOF *on error, except for* fgets *and* gets*, which return* NULL. **/*

Direct (binary) input/output:

```
size_t fread   ( void * data,       size_t size, size_t count, FILE * stream );
size_t fwrite  ( const void * data, size_t size, size_t count, FILE * stream );
```

/ Both return the total number of elements successfully read/written. */*

File positioning:

```
int fgetpos    ( FILE * stream, fpos_t * position );              /* Returns 0 on success. */
int fsetpos    ( FILE * stream, const fpos_t * position );        /* Returns 0 on success. */

long int ftell ( FILE * stream );          /* Returns the current position, or -1L if error occurs. */
int fseek      ( FILE * stream, long int offset, int origin );    /* Returns 0 on success. */
```

*/** origin *parameter:* SEEK_SET *(Beginning of file),* SEEK_CUR *(Current position),* SEEK_END *(End of file) */*

```
void rewind    ( FILE * stream );
```

Error-handling:

```
int feof       ( FILE * stream );      /* Can be triggered via Ctrl+Z (DOS/Windows) or Ctrl+D (Unix). */

int ferror     ( FILE * stream );
void perror    ( const char * str ); stderr        /* Outputs "str: <error message (from errno)>\n" */
void clearerr  ( FILE * stream );
```

Operations on files:

```
int rename     ( const char * oldname, const char * newname );        /* Returns 0 on success. */
int remove     ( const char * filename );                             /* Returns 0 on success. */

FILE * tmpfile ( void );                  /* File is created in "wb+" mode. Returns NULL on error. */
char * tmpnam  ( char * str );                          /* str must have at least L_tmpnam bytes. */
```

*/** tmpnam*: returns* str*, or a pointer to an internal buffer (if* str == NULL*), or* NULL *on error. */*

cstdlib <stdlib.h> functions

Limits that the functions may return:

atoi:	INT_MIN,	INT_MAX
atol, strtol:	LONG_MIN,	LONG_MAX
strtoul:	0L,	ULONG_MAX
atof, strtod:	0.0 *(if it underflows),*	±HUGE_VAL

String conversion:

```
int atoi                   ( const char * str );
long int atol              ( const char * str );
double atof                ( const char * str );
double strtod              ( const char * str, char ** endptr );
long int strtol            ( const char * str, char ** endptr, int base );
unsigned long int strtoul  ( const char * str, char ** endptr, int base );
```

/* *All return* 0 *(*0L*, or* 0.0*) if no valid conversion can be done.*
If the converted number is out of range, functions return the limit instead, and set errno = ERANGE. */

Pseudo-random sequence generation:

```
int rand      ( void );                 /* Interval: [0, RAND_MAX]. Usually uses (rand() % range + offset) . */
void srand    ( unsigned int seed );    /* Initial value of seed: 1. Usually uses srand(time(NULL)) . */
```

Dynamic memory management:

```
void * malloc    ( size_t size );
void * calloc    ( size_t num, size_t size );           /* Initializes the memory block to zero. */
void * realloc   ( void * ptr, size_t size );   /* Content is preserved even if the block is moved. */

void free        ( void * ptr );
```

Environment:

```
void abort       ( void );          /* Sends SIGABRT. Ignores object destructors and atexit functions! */
void exit        ( int status );    /* Macros constants available: EXIT_SUCCESS and EXIT_FAILURE. */
int atexit       ( void (* function)(void) );
```

/* atexit*: returns* 0 *on success. Registered functions are executed in reversed order as a stack.* */

```
char * getenv    ( const char * name );
```

/* getenv*: returns* NULL *if the environment variable does not exist.*
The string returned is an internal buffer and shall not be modified by the program. */

```
int system       ( const char * command );
```

/* system*: The value returned depends on the running environment. Usually* 0 *when executing successfully.*
If command == NULL, system *returns whether the command processor exists (*0 *if not).* */

Searching and sorting:

```
void * bsearch   ( const void * key, const void * base,  size_t num, size_t size,
                   int (* comparator)(const void *, const void *) );
void qsort       (                    void * base,       size_t num, size_t size,
                   int (* comparator)(const void *, const void *) );
```

/* bsearch*: binary-searches the* key *in the array* base *(returns* NULL *if not found).* qsort*: sorts the array.*
base *should have* num *elements, each element* size *bytes long, sorted / to be sorted by* comparator.
comparator *should return whether its left parameter precedes, equals, or succeeds its right parameter*
in <0, ==0, >0 *respectively. Examples:*

```
int compare (const void * a, const void * b) {
    return ( *(int*)a - *(int*)b );
}
```

```
int compare (const void * a, const void * b) {
    if ( *(int*)a >  *(int*)b ) return 1;
    if ( *(int*)a == *(int*)b ) return 0;
    if ( *(int*)a <  *(int*)b ) return -1;
}
```

*/

Integer arithmetics:

```
int abs          ( int n );
long int labs    ( long int n );

div_t div        ( int numerator,       int denominator );
ldiv_t ldiv      ( long int numerator,  long int denominator );
```

/* div *and* ldiv*: Return a structure with 2 members:* quot *(quotient) and* rem *(remainder).*
In C++, abs *and* div *are overloaded with* long int *type – same as* labs *and* ldiv*, respectively.* */

Multibyte characters:

```
int mblen        (                   const char * pmb,      size_t max );
int mbtowc       ( wchar_t * pwc,    const char * pmb,      size_t max );
int wctomb       ( char * pmb,       wchar_t character );
```

/* *All return the size in bytes of the multibyte character,* 0 *if character is null (*'\0'*), or* -1 *if it's invalid.*
If pmb == NULL*, the functions reset their individual shift states, and then return whether multibyte character encodings are state-dependent (*0 *if not).*
No more than MB_CUR_MAX *bytes are examined in any case.* */

```
size_t mbstowcs  ( wchar_t * wcstr,  const char * mbstr,     size_t max );
size_t wcstombs  ( char * mbstr,     const wchar_t * wcstr,  size_t max );
```

/* mbstowcs*: returns the number of wide characters translated.* max *parameter is in characters.*
wcstombs*: returns the number of bytes translated.* max *parameter is in bytes.*
Both return -1 *when an invalid character is met.* */

cstring <string.h> functions

Copying:

```
void * memcpy    ( void * destination, const void * source, size_t num );
void * memmove   ( void * destination, const void * source, size_t num );

char * strcpy    ( char * destination, const char * source ); ⚠
char * strncpy   ( char * destination, const char * source, size_t max ); NO \0

  /* memmove allows destination and source blocks to overlap, while memcopy doesn't.
     strncpy: pads destination with zeros if a '\0' is found before max characters.
     All return destination. */
```

Concatenation:

```
char * strcat    ( char * destination, const char * source ); ⚠
char * strncat   ( char * destination, const char * source, size_t max ); \0

  /* Both return destination. */
```

Comparison:

⚠: No buffer overflow protection (bound checking). Security issues may occur.

```
int memcmp       ( const void * ptr1, const void * ptr2, size_t num );

int strcmp       ( const char * str1, const char * str2 );
int strcoll      ( const char * str1, const char * str2 );
int strncmp      ( const char * str1, const char * str2, size_t max );

  /* strcoll: Strings are interpreted according to the LC_COLLATE category of the current locale.
     All return 0 if both "memory blocks / strings" are equal, or >0 if the first un-matching byte/character in
       "ptr1 or str1" (pointed data) has a greater value than in "ptr2 or str2", or <0 if it has less. */
```

Searching:

```
void * memchr    ( const void * ptr, int value, size_t num );
char * strchr    ( const char * str, int character );
char * strrchr   ( const char * str, int character );

  /* memchr or strchr: return a pointer to the first occurrence of "value or character" in "ptr or str".
     strrchr: returns the last occurrence.
     All return NULL if value or character is not found. */

char * strpbrk   ( const char * str, const char * chars ); NO \0
char * strstr    ( const char * str, const char * pattern ); \0

  /* strpbrk: returns the first occurrence in str of any of the characters in chars.
     strstr: returns the first occurrence in str of the (sub-)string pattern. */

size_t strspn    ( const char * str, const char * chars );
size_t strcspn   ( const char * str, const char * chars ); \0

  /* strspn: returns the length (span) of the initial portion of str containing only characters in chars.
     strcspn: returns the one not containing any of the characters in chars. */

char * strtok    ( char * str,      const char * delimiters );

  /* strtok: replaces the end of the token with '\0', and then returns the beginning of the token, or NULL if
       no tokens are found.       If str here is NULL, strtok continues tokenizing the last str inputted. */
```

Other:

```
void * memset    ( void * ptr, int value, size_t num ); /* Returns ptr. All bytes become value. */
char * strerror  ( int errnum );                         /* Returns a pointer to an internal buffer. */
size_t strlen    ( const char * str );
size_t strxfrm   ( char * destination, const char * source, size_t max );

  /* strxfrm: transforms the string according to the current locale (LC_COLLATE category), to destination
       and returns its length (excluding '\0'). */
```

Non-standard functions: (OpenBSD, FreeBSD, Solaris, and Mac OS X)

```
size_t strlcpy   ( char * destination, const char * source, size_t size ); \0
size_t strlcat   ( char * destination, const char * source, size_t size ); \0

  /* Alternatives to strncpy and strncat. Differences: (a) Always append a terminating null character.
       (b) Return the size required for the destination string (including '\0') instead.
     strlcpy: Unlike strncpy, strlcpy doesn't pad zeros. */
```

Notes

- For writing functions marked with `\0`, these append a terminating null character (`'\0'`) after the process; for searching functions with `\0`, `'\0'` is included during the search. `NO \0` means the opposite.
- All `max` parameters **exclude** the terminating `'\0'`. All `size` parameters **include** the `'\0'`.
- In C++, the functions `memchr`, `strchr`, `strrchr`, `strpbrk`, `strstr` have declarations in different form:

```
const Type * Function   ( const Type *,  «other parameters» );
Type * Function         ( Type *,        «other parameters» );
// Overloaded. Type should be replaced with char or void. Function should be replaced with memchr, strchr, etc.
```

cmath <math.h> functions

Trigonometric functions:

```
double sin   ( double x ); <complex> <valarray>
double cos   ( double x ); <complex> <valarray>
double tan   ( double x ); <complex> <valarray>
```

/* x *is expressed in radians (for* sin, cos, *and* tan). */

```
double asin  ( double x ); <valarray>        /* Intervals: x ∈ [−1, +1], return value ∈ [−π/2, +π/2]  */
double acos  ( double x ); <valarray>        /* Intervals: x ∈ [−1, +1], return value ∈ [0, π]        */
double atan  ( double x ); <valarray>                     /* Interval: return value ∈ [−π/2, +π/2]  */

double atan2 ( double y, double x ); <valarray>           /* Interval: return value ∈ [−π, +π]      */
```

/* *The return value is in radians (for* asin, acos, atan, *and* atan2).
atan2: *If* (x==0 && y==0), *it sets* errno = EDOM. */

Hyperbolic functions:

```
double sinh  ( double x ); <complex> <valarray>
double cosh  ( double x ); <complex> <valarray>
double tanh  ( double x ); <complex> <valarray>
```

/* cosh *and* sinh: *If the magnitude is too large, they return* HUGE_VAL *with appropriate sign and set* errno = ERANGE. */

Exponential and logarithmic functions:

```
double exp   ( double x ); <complex> <valarray>                                   /* Base e */
double log   ( double x ); <complex> <valarray>                                   /* Base e */
double log10 ( double x ); <complex> <valarray>                                  /* Base 10 */
```

/* exp: *If the magnitude is too large, it returns* HUGE_VAL *and sets* errno = ERANGE.
log, log10: *If* x==0, *they return* −HUGE_VAL *and set* errno = ERANGE. *If* x<0, *they set* errno = EDOM. */

```
double frexp ( double x,           int * exp );   /* Returns the significand, in the interval [1/2, 1). */
double ldexp ( double significand, int exp );                 /* Returns: (significand × 2^exp). */

double modf  ( double x, double * intpart );
```

/* modf: *returns the fractional part. Both* intpart *and return value have the same sign as* x. */

Power functions:

```
double pow   ( double base, double exponent ); <complex> <valarray>
```

/* pow: *If the magnitude is too large, it returns* HUGE_VAL *with appropriate sign and sets* errno = ERANGE.
If base<0 *and* exponent *is non-integer, or if* base==0 *and* exponent<0, *it sets* errno = EDOM. */

```
double sqrt  ( double x ); <complex> <valarray>                 /* If x<0, this sets errno = EDOM. */
```

Rounding, absolute value and remainder functions:

```
double ceil  ( double x );
double floor ( double x );

double fabs  ( double x );
```

/* fabs: *In C++,* abs *is also declared in this header with the same behavior, except that* abs *is overloaded in* <cstdlib>, <complex>, *and* <valarray>. */

```
double fmod  ( double numerator, double denominator );               /* Returns the remainder. */
```

Notes in C++

- All functions taking 1 or 2 double-type arguments are overloaded with float and long double types:

```
float Function       ( float );              float Function       ( float, float );
long double Function ( long double );        long double Function ( long double, long double );
```

- Other functions are overloaded in these ways:

```
float frexp          ( float, int * );       float modf           ( float, float * );
long double frexp    ( long double, int * ); long double modf     ( long double, long double * );

float ldexp          ( float, int );
long double ldexp    ( long double, int );
```

- Functions marked with <complex> or <valarray> are overloaded in <complex> or <valarray> respectively:

```
              template<class T>
complex<T> pow
          ( const complex<T>&, const complex<T>& );
complex<T> pow     ( const complex<T>&, const T& );
complex<T> pow     ( const T&, const complex<T>& );
complex<T> pow     ( const complex<T>&, int      );

T abs                       ( const complex<T>& );

/* For all other functions: */
complex<T> Function         ( const complex<T>& );
```

```
              template<class T>
/* For atan2 and pow functions: */
valarray<T> Function
          ( const valarray<T>&, const valarray<T>& );
valarray<T> Function ( const valarray<T>&, const T& );
valarray<T> Function ( const T&, const valarray<T>& );

/* For all other functions: */
valarray<T> Function          ( const valarray<T>& );
```

cctype <ctype.h> functions and ASCII table

/ In C++, a locale-specific template version of every function below exists in* <locale>. **/*

Case conversion:

```
int tolower  ( int c );                    /* If conversion is impossible, returns c unchanged. */
int toupper  ( int c );                    /* If conversion is impossible, returns c unchanged. */
```

Classification:

/ All 11 classifying functions (*`iscntrl`, `isspace`, `isprint`, `isgraph`, `ispunct`, `isalnum`, `isxdigit`, `isdigit`, `isalpha`, `isupper`, `islower`*) are in this form: */*

```
int Function ( int c );
```

/ All return non-zero (true) if* `c` *belongs to the category,* `0` *(false) otherwise. */*

ASCII values	Characters	iscntrl	isspace	isprint	isgraph	ispunct	isalnum	isxdigit	isdigit	isalpha	isupper	islower
0x00–0x08	\0 *(other control chars)*	●										
0x09–0x0D	\t\n\v\f\r	●	●									
0x0E–0x1F	*(control chars)*	●										
0x20	*(Sp)*		●	●								
0x21–0x2F	!"#$%&'()*+,-./			●	●	●						
0x30–0x39	0123456789			●	●		●	●	●			
0x3A–0x40	:;<=>?@			●	●	●						
0x41–0x46	ABCDEF			●	●		●	●		●	●	
0x47–0x5A	GHIJKLMNOPQRSTUVWXYZ			●	●		●			●	●	
0x5B–0x60	[\]^_`			●	●	●						
0x61–0x66	abcdef			●	●		●	●		●		●
0x67–0x7A	ghijklmnopqrstuvwxyz			●	●		●			●		●
0x7B–0x7E	{|}~			●	●	●						
0x7F	*(Del)*	●										

●: function will return non-zero.
(blank): function will return zero.

ASCII table

↓Binary	Oct.	**Dec.**	Hex.	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000 0000	000_8	**0**	0x00	*Nul*	*SoH*	*STx*	*ETx*	*EoT*	*Enq*	*Ack*	*Bel*	*BS*	*HT*	*LF*	*VT*	*FF*	*CR*	*SO*	*SI*
001 0000	020_8	**16**	0x10	*DLE*	*DC1*	*DC2*	*DC3*	*DC4*	*NAk*	*Syn*	*ETB*	*Can*	*EM*	*Sub*	*Esc*	*FS*	*GS*	*RS*	*US*
010 0000	040_8	**32**	0x20	*Sp*	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
011 0000	060_8	**48**	0x30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
100 0000	100_8	**64**	0x40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
101 0000	120_8	**80**	0x50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
110 0000	140_8	**96**	0x60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
111 0000	160_8	**112**	0x70	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	*Del*

Nul	^@	'\0' Null character	*BS*	^H	'\b' Backspace	*DLE*	^P	Data Link Escape	*Can*	^X	Cancel
SoH	^A	Start of Header	*HT*	^I	'\t' Horizontal Tab	*DC1*	^Q	Device Control 1	*EM*	^Y	End of Medium
STx	^B	Start of Text	*LF*	^J	'\n' Line feed	*DC2*	^R	Device Control 2	*Sub*	^Z	Substitute
ETx	^C	End of Text	*VT*	^K	'\v' Vertical Tab	*DC3*	^S	Device Control 3	*Esc*	^[	Escape
EoT	^D	End of Transmission	*FF*	^L	'\f' Form Feed	*DC4*	^T	Device Control 4	*FS*	^\	File Separator
Enq	^E	Enquiry	*CR*	^M	'\r' Carriage Return	*NAk*	^U	Negative Acknowledge	*GS*	^]	Group Separator
Ack	^F	Acknowledgment	*SO*	^N	Shift Out	*Syn*	^V	Synchronous idle	*RS*	^^	Record Separator
Bel	^G	'\a' Bell	*SI*	^O	Shift In	*ETB*	^W	End of Transmission Block	*US*	^_	Unit Separator

(Note: '\\' is used to output a single backslash \.)

Sp		Space	*Del*	^?	Delete

clocale <locale.h> functions

```
char * setlocale ( int category, const char * locale );
```

/* `setlocale`*: returns a string identifying the locale currently set for the* `category`*, or* NULL *on error.*
`locale` *parameter: name of a locale, which are system-specific, except for these two:*
`"C"` *(Minimal "C" locale; all C programs set this by default), and* `""` *(Environment's default locale)*
If `locale` == NULL, `setlocale` *doesn't change the locale but returns the current locale name.* */

```
struct lconv * localeconv ( void );
```

/* `localeconv`*: returns the formatting parameters for quantities in the current locale. The pointed data shouldn't be modified as it may be overridden by further calls to* `localeconv` *or* `setlocale`*.* */

Locale categories in C: (for `category` parameter)

LC_ALL	The entire locale
LC_COLLATE	Affects the behavior of `strcoll` and `strxfrm`.
LC_CTYPE	Affects character handling functions (all functions of <cctype>, except `isdigit` and `isxdigit`), and the multibyte and wide character functions.
LC_MONETARY	Affects the monetary formatting information returned by `localeconv`.
LC_NUMERIC	Affects the decimal-point character in formatted input/output operations and string formatting functions, as well as non-monetary information returned by `localeconv`.
LC_TIME	Affects the behavior of `strftime`.
LC_MESSAGES	*(POSIX.1, not in C standard)* Affects what strings are expected or given (or both) by commands and utilities as affirmative or negative responses.

Types:

`struct lconv {`

	Value in "C" locale	Description

`char * decimal_point;`
`char * mon_decimal_point;`
	`"."`
	`""`
		Decimal-point separator.

`char * thousands_sep;`
`char * mon_thousands_sep;`
	`""`
	`""`
		Separators used to delimit group of digits to the left of the decimal point.

`char * grouping;`
`char * mon_grouping;`
	`""`
	`""`
		The size of each group of digits. (From right to left, starting at the decimal point.)
		The last number before the ending `0` (`'\0'`) is used over and over for the remaining groups. If this number is CHAR_MAX, no further grouping would be performed.
		Example: `"\3\2\1"` → `1,0,0,0,00,000` (If `thousand_sep` == `","`)

`char * positive_sign;`
`char * negative_sign;`
	`""`
	`""`
		Sign to be used for monetary quantities.

`char * int_curr_symbol;`
`char * currency_symbol;`
	`""`
	`""`
		`int_curr_symbol` consists of the 3-letter ISO 4217 code, followed by a character (usually space) that separates the code from the monetary quantity.
		`currency_symbol`: Local currency symbol, like "$".

`char int_frac_digits;`
`char frac_digits;`
	CHAR_MAX
	CHAR_MAX
		Amount of fractional digits after the decimal point for monetary quantities.

`char p_cs_precedes;`
`char n_cs_precedes;`
	CHAR_MAX
	CHAR_MAX
		Whether the currency symbol should precede or follow the monetary quantities.
		1: Precede $\boxed{\$\ }$123 0: Follow 123$\boxed{\ \$}$

`char p_sep_by_space;`
`char n_sep_by_space;`
	CHAR_MAX
	CHAR_MAX
		Whether a space should appear between the currency symbol and the monetary quantities.
		1: Yes $\boxed{\$\ }$123 0: No $\boxed{\$}$123

`char p_sign_posn;`
`char n_sign_posn;`
	CHAR_MAX
	CHAR_MAX
		Position of the sign.
		`0`: Currency symbol and quantity surrounded by parentheses ($\boxed{\$\ 123}$)
		1: Sign before quantity and currency symbol -$\boxed{\$\ 123}$
		2: Sign after quantity and currency symbol $\boxed{\$\ 123}$-
		3: Sign right before currency symbol -$\boxed{\$}$123
		4: Sign right after currency symbol $\boxed{\$\ }$-123

`};`

Notes

- `decimal_point`, `thousands_sep`, and `grouping` are for non-monetary quantities; the rest are monetary.
- `int_curr_symbol` and `int_frac_digits` are for monetary quantities in the international format.
- Member names with "`p_`" prefix are for positive or zero quantities; those with "`n_`" are for negatives.
- CHAR_MAX or `""` (empty string) indicates that the value is unspecified.

ctime <time.h> functions

Time manipulation:

```
clock_t clock    ( void );
```

/* clock: *returns the number of clock ticks elapsed since the program starts, or* -1 *on error. The initial moment of reference used by* clock *as the beginning of the program execution may vary between platforms. To calculate the actual processing times of a program, the value returned by* clock *should be compared to a value returned by an initial call to* clock.
Macro constant available: CLOCKS_PER_SEC *(number of clock ticks in a second).* */

```
time_t time      ( time_t * timer );
```

/* time: *returns the current calendar time, or* -1 *on error. It also stores the return value to the location pointed by* timer *argument (if it is not* NULL*).* */

```
time_t mktime    ( struct tm * timeptr );
```

/* *Returns the converted time, or* -1 *on error.* */

/* mktime: *interprets the structure pointed by* timeptr *as local time. The members* tm_wday *and* tm_yday *are not read, but set to appropriate values. Other members are also set to values within the range.* */

```
double difftime  ( time_t latter, time_t earlier );
```

/* *Returns:* (latter − earlier) *in seconds.* */

Conversion:

```
struct tm * gmtime     ( const time_t * timer );
struct tm * localtime  ( const time_t * timer );
```

/* *Returned time is expressed as UTC.* */
/* *Returned time is local time.* */

/* gmtime *and* localtime: *return a pointer to a structure that is statically allocated and shared by both functions. Each time either one of these functions is called the content is overwritten.* */

```
char * asctime   ( const struct tm * timeptr );
char * ctime     ( const time_t * timer );
```

/* *The returned string has this format: "Www Mmm dd hh:mm:ss yyyy\n" (≈"%a %b %d %H:%M:%S %Y\n" in "C" locale). Terminating* '\0' *included. The array that holds the string is statically allocated and shared by both* asctime *and* ctime. */

```
size_t strftime ( char * str, size_t maxsize, const char * format, const struct tm * timeptr );
```

/* strftime: *returns the number of characters written to* str *(excluding* '\0'*).* maxsize *includes* '\0'. *If the resulting string doesn't fit,* strftime *returns* 0 *and the contents of* str *are indeterminate.* */

Types:

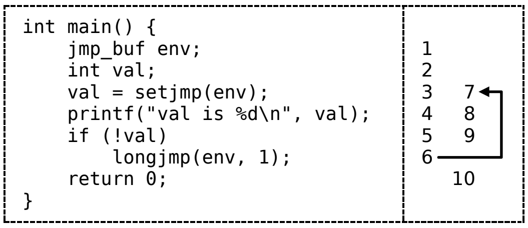
```
struct tm {
```

	Description	Range
int tm_sec;	Seconds	0–61
int tm_min;	Minutes	0–59
int tm_hour;	Hours	0–23
int tm_mday;	Day of the month	1–31
int tm_mon;	Month since January	0–11
int tm_year;	Years since 1900	
int tm_wday;	Days since Sunday	0–6
int tm_yday;	Days since January 1	0–365
int tm_isdst;	Daylight Saving Time (DST) flag >0: DST is in effect. ==0: DST is not in effect. <0: Information is not available.	

```
};
```

/* *The member* tm_sec *allows up to 2 leap seconds (if supported by the system), although 1 is enough for UTC.* */

Example usage of setjmp and longjmp:



```
int main() {
    jmp_buf env;                          1
    int val;                              2
    val = setjmp(env);                    3   7
    printf("val is %d\n", val);           4   8
    if (!val)                             5   9
        longjmp(env, 1);                  6
    return 0;                                10
}
```

csetjmp <setjmp.h>

```
/* macro */ int setjmp  ( jmp_buf env );
```

/* setjmp: *saves the calling environment to* env *for later use by* longjmp.
setjmp *returns* 0 *on its direct invocation, and returns a non-zero value on a later return from* longjmp. */

```
void longjmp         ( jmp_buf env, int val );
```

/* longjmp: *restores the environment from* env, *and pass the argument* val *to* setjmp. *Program execution continues as if the corresponding call of* setjmp *had just returned the value* val.
(If val==0, setjmp *returns* 1 *instead.)* */

Non-standard functions: (POSIX.1)

```
int sigsetjmp    ( sigjmp_buf env, int savesigs );
void siglongjmp  ( sigjmp_buf env, int val );
```

/* *If* savesigs *argument is non-zero (true),* sigsetjmp *also saves the set of blocked signals to* env, *which will then be restored by* siglongjmp. */

cassert <assert.h>

/* macro */ `void assert ( int expression );`

/* `assert`*: If* `expression==0`*, it outputs a message to* `stderr` *and then calls* `abort`*, terminating the program. The error message is usually like this: "Assertion failed:* `expression`*, file* __FILE__*, line* __LINE__*" Adding the line* `#define NDEBUG` *before the inclusion of* <assert.h> *disables the* `assert` *macro.* */

cerrno <errno.h>

/* macro */ `int errno = 0;`

/* `errno`*: Last error number, modified by certain functions to signal some types of error. You may modify it. In C++,* `errno` *is always declared as a macro, but in C compilers it may also be implemented as an* `int` *object with external linkage.*
Macro constants available: EDOM *(Domain error),* ERANGE *(Range error), and* EILSEQ *(Illegal byte sequence).* */

csignal <signal.h> functions

`void (* signal ( int sig, void (* handler)(int) ) )(int);`

/* `signal`*: returns the signal handler function* ***before*** *this call, or* SIG_ERR *on error.*
`handler` *parameter: signal handler function, which may be* SIG_DFL *(Default handling),* SIG_IGN *(Ignore the signals), or a user-defined function.*
For maximum portability, a signal handler should only make calls (that succeed) to the function `signal`*, assign values to objects of type* `volatile sig_atomic_t`*, and return control to its caller.* */

`int raise ( int sig );` /* *Returns* `0` *on success.* */

ANSI standard signals: (for `sig` parameter)

SIGABRT	Abort (from `abort` function)	SIGINT	Interrupt (generated by user pressing interrupt key such as Ctrl+C)
SIGFPE	Erroneous arithmetic operation (formerly: Floating-Point Exception)	SIGSEGV	Segmentation Violation (invalid memory reference, or segmentation fault)
SIGILL	Illegal Instruction	SIGTERM	Termination request

Each compiler implementation may provide additional signal number macro constants to be used by functions.

Notes

- Equivalent, yet human-readable prototype of `signal`:

```
/* In FreeBSD: */
typedef void (* sig_t)(int);
sig_t signal ( int sig, sig_t handler );
```

```
/* In GNU C Library: */
typedef void (* sighandler_t)(int);
sighandler_t signal ( int sig, sighandler_t handler );
```

- <signal.h> also defines **`sig_atomic_t`**, the integral type of an object that can be accessed as an atomic entity, even in the presence of asynchronous interrupts. It is used as a variable in signal handlers.

cstdarg <stdarg.h>

```
/* macro */ void va_start  ( va_list ap, lastparam );
/* macro */ Type va_arg    ( va_list ap, Type );      /* Type is expanded to return type of the macro. */
/* macro */ void va_end    ( va_list ap );
```

/* `lastparam` *parameter: Name of the last named parameter (just before the ellipsis).* */

Example usage

```
void PrintArgs (int amount, ...) {
    int value, i;
    va_list vl;
    va_start(vl, amount);
    for (i = 0; i<amount; i++) {
        value = va_arg(vl, int);
        printf("%d ", value);
    }
    va_end(vl);
}
```

```
void WriteFormatted (char * format, ...) {
    va_list args;
    va_start(args, format);
    vprintf(format, args);
    va_end(args);
}
```

/* `vfprintf`, `vprintf`*, and* `vsprintf` *do not automatically call the* `va_end` *macro.* */

cstddef <stddef.h>

/* <stddef.h> *also define these types:* **`ptrdiff_t`** *(result of pointer subtraction),* **`size_t`**, *and* **`wchar_t`**. */

/* macro */ `size_t offsetof(Type,member);`

/* `offsetof`*: returns the offset value of* `member` *in the structure* `Type`*.*
`offsetof` *is not a function and cannot be described as a C prototype. In C++, the use of* `offsetof` *is restricted to "POD types", which for classes, more or less corresponds to the C concept of* `struct`*.* */

climits <limits.h>

Name	ANSI Minimum magnitude	Description
CHAR_BIT	≥+8	Number of **bits** in a **`char`** (or a byte)
MB_LEN_MAX	≥+1	**Maximum length** of a **multibyte** character across all locales (in bytes)
CHAR_MIN	SCHAR_MIN *or* 0	**Minimum** value for a **`char`**
CHAR_MAX	SCHAR_MAX *or* UCHAR_MAX	**Maximum** value for a **`char`**
SCHAR_MIN	≤− 127	**Minimum** value for a **`signed char`**
SCHAR_MAX	≥+ 127	**Maximum** value for a **`signed char`**
UCHAR_MAX	≥+ 255	**Maximum** value for an **`unsigned char`**
SHRT_MIN	≤− 32 767	**Minimum** value for a **`short int`**
SHRT_MAX	≥+ 32 767	**Maximum** value for a **`short int`**
USHRT_MAX	≥+ 65 535	**Maximum** value for an **`unsigned short int`**
INT_MIN	≤− 32 767	**Minimum** value for an **`int`**
INT_MAX	≥+ 32 767	**Maximum** value for an **`int`**
UINT_MAX	≥+ 65 535	**Maximum** value for an **`unsigned int`**
LONG_MIN	≤− 2 147 483 647	**Minimum** value for a **`long int`**
LONG_MAX	≥+ 2 147 483 647	**Maximum** value for a **`long int`**
ULONG_MAX	≥+ 4 294 967 295	**Maximum** value for an **`unsigned long int`**

cfloat <float.h>

Name	Minimum magnitude	Description
FLT_RADIX	≥2	Base (i.e. **radix**) for all floating-point types (`float`, `double` and `long double`).
FLT_ROUNDS		Rounding mode for floating-point addition: -1: Indeterminable 0: Toward zero 1: To nearest (default) 2: Toward positive infinity 3: Toward negative infinity (other): Non-standard mode
FLT_MANT_DIG DBL_MANT_DIG LDBL_MANT_DIG		Number of **digits** that conform the significand (**mantissa**), i.e. precision of significand. (in the FLT_RADIX base)
FLT_DIG DBL_DIG LDBL_DIG	≥6 ≥10 ≥10	Number of decimal **digits** that can be rounded into a floating-point and back without change in the number of decimal digits.
FLT_MIN_EXP DBL_MIN_EXP LDBL_MIN_EXP		**Minimum** negative integer value for the **exponent** that generates a normalized floating-point number. (in base FLT_RADIX)
FLT_MIN_10_EXP DBL_MIN_10_EXP LDBL_MIN_10_EXP	≤−37 ≤−37 ≤−37	**Minimum** negative integer value for the **exponent** of a **base-10** expression that would generate a normalized floating-point number.
FLT_MAX_EXP DBL_MAX_EXP LDBL_MAX_EXP		**Maximum** integer value for the **exponent** that generates a normalized floating-point number. (in base FLT_RADIX)
FLT_MAX_10_EXP DBL_MAX_10_EXP LDBL_MAX_10_EXP	≥+37 ≥+37 ≥+37	**Maximum** integer value for the **exponent** of a **base-10** expression that would generate a normalized floating-point number.
FLT_EPSILON DBL_EPSILON LDBL_EPSILON	≤1E−5 ≤1E−9 ≤1E−9	**Epsilon** (Difference between 1 and the least value greater than 1 that is representable.)
FLT_MIN DBL_MIN LDBL_MIN	≤1E−37 ≤1E−37 ≤1E−37	**Minimum** positive representable floating-point number.
FLT_MAX DBL_MAX LDBL_MAX	≥1E+37 ≥1E+37 ≥1E+37	**Maximum** finite representable floating-point number.

The minimum magnitudes refer to those in ANSI standard. Actual values may vary between implementations.

Other macro constants (that are not described above)

NULL	Null pointer	<stddef.h> <stdio.h> <stdlib.h> <string.h> <time.h>
This macro expands to a null pointer constant. A null pointer is generally used to signify that a pointer does not point to any object. In C++, NULL expands either to `0` or `0L`.		
EOF	End-of-File	<stdio.h>
This macro expands to a negative integral constant expression. It is used as the value returned by several <stdio.h> functions to indicate failure, either because the End-of-File has been reached in a reading operation or because an error happened.		
FILENAME_MAX	Maximum length of file names	<stdio.h>
This macro constant expands to an integral expression corresponding to the size needed for an array of `char` elements to hold the longest file name string allowed by the system. Or, if the system imposes no such restriction, it is set to the recommended size for character arrays intended to hold any file name.		
TMP_MAX	Number of temporary files	<stdio.h>
This macro expands to the minimum number of unique temporary file names that are granted to be possible to generate using `tmpnam` or `tmpfile`.		
EXIT_FAILURE	Failure termination code	<stdlib.h>
This macro expands to a system-dependent integral expression that, when used as the argument for function `exit`, should signify that the application failed. The opposite meaning can be specified with EXIT_SUCCESS (see below).		
EXIT_SUCCESS	Success termination code	<stdlib.h>
This macro expands to a system-dependent integral expression that, when used as the argument for function `exit`, should signify that the application was successful.		
MB_CUR_MAX	Maximum size of multibyte characters	<stdlib.h>
This macro expands to a positive integer expression, the value of which is the maximum number of bytes in a multibyte character with the current locale. Its value is never greater than MB_LEN_MAX (macro defined in <limits.h>).		
RAND_MAX	Maximum value returned by `rand`	<stdlib.h>
This macro expands to an integral constant expression whose value is the maximum value returned by the `rand` function. This value is library dependent, but is granted to be at least 32767.		
HUGE_VAL	Huge value	<math.h>
A function returns this value when the result of a mathematical operation yields a value that is so large in magnitude that it is not representable with its return type. This is one of the possible range errors, and is signaled by setting `errno` to ERANGE. Functions can return either a positive or a negative HUGE_VAL to at least indicate the sign of the result.		
CLOCKS_PER_SEC	Clock ticks per second	<time.h>
This macro expands to an expression representing the number of clock ticks in a second, as returned by the function `clock`. Dividing a count of clock ticks by this expression yields the number of seconds. CLK_TCK is an obsolete alias of this macro.		

ciso646 <iso646.h>

/ In C++, these words are reserved and treated as aliases of their respective operator. */*

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