C Standard Libraries

1 Abstract

This file lists the contents of the C standard libraries that are part of the ANSI standard. Very brief descriptions are given for each feature; see the man-pages for more information.

The library facilities are divided into groups according to their functionality. The header file that is necessary to get access to the facilites are given in the head of each section (together with the flag that must be given to the C compiler, when applicable).

Every group includes information about the following:

- Functions available (the function prototype is given).
- Constants defined in the header file (constants are C macros that take no arguments).
- Macros defined in the header file.
- Types defined in the header file (including structures).
- Global variables that are available to the programmer.

Note that some often used constants are defined in several header files.

(This list is based on chapters 10 - 20 of the book "C, a Reference Manual" by Samuel P. Harbison and Guy L. Steele jr. (3rd ed., Prentice Hall 1991).)

Contents

2 Common definitions (stddef.h)

Functions:

None	

Constants:

NULL	Null pointer.

Macros:

offsetof(struct, member)	Offset (in bytes) of member in struct.

Types:

ptrdiff_t	Type of subtraction of two pointers.
size_t	Type of result of sizeof operator.
wchar_t	Type for 'wide characters'.

Global variables:

None	

3 Input/output (stdio.h)

Functions:

int remove(const char *filename);	Removes a file.
int rename(const char *oldname, const char *newname);	Renames file oldname to newname.
FILE *tmpfile(void);	Creates a new temporary file.
char *tmpnam(char *buf);	Creates a new unique file name.
int fclose(FILE *stream);	Closes stream.
int fflush(FILE *stream);	Empties stream's buffers.
FILE *fopen(const char *filename, const char *mode);	Opens filename in a certain mode (e.g. "r" for reading, "w" for writing).
FILE *freopen(const char *filename, const char *mode, FILE *stream);	Opens filename in a certain mode and reassociates stream with it.
void setbuf(FILE *stream, char *buf);	Tells system to not buffer stream if buf is NULL; otherwise use strategy _IOFBF with buf as buffer and size BUFSIZE.
int setvbuf(FILE *stream, char *buf, int bufmode, size_t size);	Tells system to use buffering strategy bufmode for stream, using buffer buf of size size.
int fprintf(FILE *stream, const char *format, ...);	Formatted output to stream according to format.
int fscanf(FILE *stream, const char *format,...);	Formatted input from stream according to format.
int printf(const char *format, ...);	Formatted output to stdout according to format.
int scanf(const char *format, ...);	Formatted input from stdin according to format.
int sprintf(char *s, const char *format, ...);	Formatted output to string s according to format.
int sscanf(const char *s, const char *format, ...);	Formatted input from string s according to format.
int vprintf(const char *format, va_list arg);	Like printf(), except that extra arguments are taken from arg.
int vfprintf(FILE *stream, const char *format, va_list arg);	Like fprintf(), except that extra arguments are taken from arg.
int vsprintf(char *s, const char *format, va_list arg);	Like sprintf(), except that extra arguments are taken from arg.
int fgetc(FILE *stream);	Reads next character from stream.
char *fgets(char *s, int n, FILE *stream);	Reads a line from stream (at most n-1 characters) and puts in string s.
int fputc(int c, FILE *stream);	Writes character c on stream.
int fputs(const char *s, FILE *stream);	Writes string s on stream.
int getc(FILE *stream);	Like fgetc(), but may be a macro.
int getchar(void);	Reads next character from stdin.
char *gets(char *s);	Reads a line from stdin and puts in string s.
int putc(int c, FILE *stream);	Like fputc(), but may be a macro.
int putchar(int c);	Writes character c on stdout.
int puts(const char *s);	Writes string s on stdout.

`int ungetc(int c, FILE *stream);`	Puts character c back on stream
`size_t fread(void *data, size_t elt_size, size_t count, FILE *stream);`	Reads count elements of size elt_size from stream and puts them in array data.
`size_t fwrite(const void *data, size_t elt_size, size_t count, FILE *stream);`	Writes count elements of size elt_size from array data to stream.
`int fgetpos(FILE *stream, fpos_t *pos);`	Stores the ‘current position’ of stream in pos.
`int fseek(FILE *stream, long int offset, int wherefrom);`	Sets the current position of stream to offset characters from beginning of file, end of file or current position (wherefrom should be one of SEEK_SET, SEEK_END or SEEK_CUR)
`int fsetpos(FILE *stream, const fpos_t *pos);`	Sets the current position of stream to the position that pos points to.
`long int ftell(FILE *stream);`	Returns current position in stream.
`void rewind(FILE *stream);`	Sets current position of stream to the beginning of file.
`int feof(FILE *stream);`	True if current position of stream is at the end of file.
`int ferror(FILE *stream);`	Returns error status of stream
`void clearerr(FILE *stream);`	Resets the error status of stream.
`void perror(const char *s);`	Writes s followed by an error messsage depending on errno on stderr.

Constants:

_IOFBF	Buffering strategy: Full buffering.
_IOLBF	Buffering strategy: Buffer one line.
_IONBF	Buffering strategy: No buffering.
BUFSIZ	Buffer size
EOF	End of file.
FILENAME_MAX	Maximum length of a filename.
FOPEN_MAX	Maximum number of streams that can be open simultaneously.
L_tmpnam	Maximum length of a temporary file name (from tmpnam()).
TMP_MAX	Number of successive calls to tmpnam() that generates unique names.
SEEK_CUR	fseek(): seek from current position.
SEEK_END	fseek(): seek from end of file.
SEEK_SET	fseek(): seek from beginning of file.
NULL	Null pointer.

Macros:

None

Types:

FILE	Type for streams.
fpos_t	Type for file positions.
size_t	Type of result of sizeof operator.

Global variables:

stdin	Standard input stream(‘normal input’).
stdout	Standard output stream (‘norma output).
stderr	Stream for error messages.

4 General utilities (stdlib.h)

Functions:

double atof(const char *str);	Converts string str to double.
int atoi(const char *str);	Converts string str to integer.
long int atol(const char *str);	Converts string str to long.
double strtod(const char *str, char **ptr);	Converts string str to double, and sets ptr to point to next character in str after the number.
long int strtol(const char *str, char **ptr, int base);	Converts string str to integer, and sets ptr to point to next character in str after the number. The number in str should be radix base.
unsigned long int strtoul(const char *str, char **ptr, int base);	Like strtol(), but converts str to unsigned long.
int rand(void);	Returns a pseudo random number between 0 and RAND_MAX. (This function may not be as good as it ought to. It may often be necessary to use some other random number generator, e.g. drand48.)
void srand(unsigned int seed);	Initializes rand() with seed.
void *malloc(size_t size);	Allocates memory for an object of size size.
void *calloc(size_t elt_count, size_t elt_size);	Allocates memory for elt_count elements of size elt_size.
void free(void *ptr);	Deallocates memory pointed to by ptr.
void *realloc(void *ptr, size_t size);	Changes the size of the allocated memory pointed to by ptr to size.
void exit(int status);	Terminate the program and return status.
void abort(void);	Aborts the program.
int atexit(void (*func) (void));	Tells the system to call the function func when the program terminates.
char *getenv(const char *name);	Returns the value of environment variable name.
int system(const char *command);	Execute command in the shell.
void *bsearch(const void *key, const void *base, size_t count, size_t size, int (*compar) (const void *the_key, const void *a_value));	Searches an array (which must be sorted) of count elements (of size size) starting at base for key. The function compar is used to compare elements with the key.
void qsort(void *base, size_t count, size_t size, int (*compar) (const void *element1, const void *element2));	Sorts an array of count elements (of size size) starting at base. The function compar gives the ordering.
int abs(int x);	Returns the absolute value of x.
long int labs(long int x);	Returns the absolute value of x.
div_t div(int n, int d);	Returns the quotient and remainder of the division of n by d.
ldiv_t ldiv(long int n, long int d);	Returns the quotient and remainder of the division of n by d.
int mblen(const char *s, size_t n);	Returns the size of multibyte character s, which is not more than n characters.
int mbtowc(wchar_t *pwc, const char *s, size_t n);	Converts multibyte character s, which is not more than n characters, to a wide character pointed to by pwc.
int wctomb(char *s, wchar_t wchar);	Converts the wide character wchar to multibyte character s.
size_t mbstowcs(wchar_t *pwcs, const char *s, size_t n);	Converts n multibyte characters in s to wide characters, and stores them in the array pwcs.
size_t wcstombs(char *s, const wchar_t *pwcs, size_t n);	Converts n wide characters in pwcs to multibyte characters, and stores them in the array s.

Constants:

EXIT_FAILURE	Return value of program if it failed.
EXIT_SUCCESS	Return value of program if it succeeded.
MB_CUR_MAX	Maximum length of a multibyt character.
NULL	Null pointer.
RAND_MAX	Maximum value returned by rand().

Macros:

None

Types:

div_t	Structure containing quotient and remainder (integers).
ldiv_t	Structure containing quotient and remainder (long integers).
size_t	Type of result of sizeof operator.
wchar_t	Type for ‘wide characters’.

Global variables:

None

5 String handling (string.h)

Functions:

`size_t strlen(const char *s);`	Returns the number of characters in the string s.
`int strcmp(const char *s1, const char *s2);`	Lexicographically compares the two strings s1 and s2.
`int strncmp(const char *s1, const char *s2, size_t n);`	Lexicographically compares up to n characters of string s1 with up to n characters of string s2.
`char *strcpy(char *dest, const char *src);`	Copies the string src to the string dest.
`char *strncpy(char *dest, const char *src, size_t n);`	Copies n characters from the string src to the string dest.
`char *strcat(char *dest, const char *src);`	Appends the string src to the end of the string dest.
`char *strncat(char *dest, const char *src, size_t n);`	Appends up to n characters from the string s to the end of the string dest.
`char *strchr(const char *s, int c);`	Searches the string s for character c. Returns pointer to first occurence.
`char *strrchr(const char *s, int c);`	Searches the string s for character c. Returns pointer to last occurence.
`size_t strspn(const char *s, const char *set);`	Searches the string s for the first character that is not included in set. Returns the number of characters that were skipped.
`size_t strcspn(const char *s, const char *set);`	Searches the string s for the first character that is included in set. Returns the number of characters that were skipped.
`char *strpbrk(const char *s, const char *set);`	Like strcspn(), but returns a pointer to the first character found from set.
`char *strstr(const char *src, const char *sub);`	Locates the first occurence of string sub in the string src.
`char *strtok(char *str, const char *set);`	Separates the string str into tokens separated by characters from set.
`void *memcpy(void *dest, const void *src, size_t len);`	Copies len bytes from src to dest.
`void *memmove(void *dest, const void *src, size_t len);`	Copies len bytes from src to dest (src and dest may overlap).
`int memcmp(const void *ptr1, const void *ptr2, size_t len);`	Compares len bytes beginning at ptr1 with len bytes beginning at ptr2.
`void *memchr(const void *ptr, int val, size_t len);`	Searches for val in the len bytes beginning at ptr.
`void *memset(void *ptr, int val, size_t len);`	Copies val into len bytes starting at ptr.
`char *strerror(int errnum);`	Returns a pointer to an error message that explains error errnum.
`int strcoll(const char *s1, const char *s2);`	Compares the two strings s1 and s2 according to the current locale.
`size_t strxfrm(char *dest, const char *src, size_t len);`	Transforms the string src so that strcmp() can be used instead of strcoll() and puts the result in dest.

Constants:

`NULL`	Null pointer.

Macros:

None

Types:

`size_t`	Type of result of sizeof operator.

Global variables:

None

6 Character handling (ctype.h)

Functions:

`int isalnum(int c);`	True if c is alphanumeric character.
`int isalpha(int c);`	True if c is alphabetic character.
`int iscntrl(int c);`	True if c is 'control character'.
`int isdigit(int c);`	True if c is decimal digit.
`int isgraph(int c);`	True if c is printable and not space.
`int islower(int c);`	True if c is lowercase letter.
`int isprint(int c);`	True if c is printable.
`int ispunct(int c);`	True if c is 'punctuation character'.
`int isspace(int c);`	True if c is 'whitespace character'.
`int isupper(int c);`	True if c is uppercase letter.
`int isxdigit(int c);`	True if c is hexadecimal digit.
`int tolower(int c);`	Returns c converted to lowercase.
`int toupper(int c);`	Returns c converted to uppercase.

Constants:

None

Macros:

None

Types:

None

Global variables:

None

7 Date and time (time.h)

Functions:

`clock_t clock(void);`	Returns the processor time used by the current process.
`time_t time(time_t *tptr);`	Returns the the current calendar time, and stores it at tptr.
`double difftime(time_t t1, time_t t0);`	Returns the number of seconds from time t0 to time t1.
`time_t mktime(struct tm *tmptr);`	Converts a time value from a tm structure to type time_t.
`struct tm *gmtime(const time_t *);`	Converts a time value of type time_t to a tm structure (Greenwich mean time).
`struct tm *localtime(const time_t *);`	Converts a time value of type time_t to a tm structure (local time).
`char *asctime(const struct tm *ts);`	Returns a pointer to a string that contains time ts in printable form.
`char *ctime(const time_t *timptr);`	Returns a pointer to a string that contains time timptr in printable form.
`size_t strftime(char *s, size_t maxsize, const char *format, const struct tm *timeptr);`	Stores the time value at timeptr in printable form in s according to format. At most maxsize characters are written in s.

Constants:

CLOCKS_PER_SEC	Number of time units per second.
NULL	Null pointer.

Macros:

None

Types:

clock_t	Type for processor time.
time_t	Type for calendar time.
size_t	Type of result of sizeof operator.
struct tm	Structure for calendar time.

Global variables:

None

8 Mathematics (math.h, -lm)

Functions:

`double acos(double x);`	Returns arc cosine of x.
`double asin(double x);`	Returns arc sine of x.
`double atan(double x);`	Returns arc tangent of x.
`double atan2(double y, double x);`	Returns arc tangent of y/x.
`double cos(double x);`	Returns cosine of x.
`double sin(double x);`	Returns sine of x.
`double tan(double x);`	Returns tangent of x.
`double cosh(double x);`	Returns hyperbolic cosine of x.
`double sinh(double x);`	Returns hyperbolic sine of x.
`double tanh(double x);`	Returns hyperbolic tangent of x.
`double exp(double x);`	Returns e to the power of x.
`double frexp(double x, int *nptr);`	Splits x into fraction and exponent.
`double ldexp(double x, int n);`	Returns x times 2^n.
`double log(double x);`	Returns the natural logarithm of x.
`double log10(double x);`	Returns the base-10 logarithm of x.
`double modf(double x, double *nptr);`	Splits x into fraction and integer.
`double pow(double x, double y);`	Returns x to the power of y.
`double sqrt(double x);`	Returns the square root of x (>= 0).
`double ceil(double x);`	Returns footnotesizeest integer >= x.
`double fabs(double x);`	Returns the absolute value of x.
`double floor(double x);`	Returns largest integer <= x.
`double fmod(double x, double y);`	Returns f so that x = ky+f, for some k.

Constants:

`HUGE_VAL`	Returned if value is too large to be represented.

Macros:

None

Types:

None

Global variables:

None

9 Errors (errno.h)

Functions:

None

Constants:

`EDOM`	Error ‘argument not in domain’.
`ERANGE`	Error ‘out of range’.
...	(Many other error codes often defined).

Macros:

None

Types:

None

Global variables:

`errno`	Latest error code.

10 Cs (assert.h)

Functions:

None

Constants:

NDEBUG	Define to turn off assert facility.

Macros:

assert(expression)	Check that expression is true (!= 0).

Types:

None

Global variables:

None

11 Arguments (stdarg.h)

Functions:

None

Constants:

None

Macros:

va_start(ap, last_fixed_arg)	Initializes the va_list ap to point to the next argument after last_fixed_arg
va_arg(ap, type)	Returns next argument which should be of type type.
va_end(ap)	Cleans up after last call to va_arg.

Types:

va_list	Local state variable used to traverse the arguments.

Global variables:

None

12 Signal handling (signal.h)

Functions:

```
void (*signal(int sig, void (*func) (int)))(int);
                                            Associates signal handler func with
                                            signal sig. Returns old handler.
int raise(int sig);                         Raises signal sig.
```

Constants:

SIG_DFL	Default handling of signals.
SIG_ERR	Error in signal().
SIG_IGN	Ignore signal.
SIGABRT	Signal: abort.
SIGFPE	Signal: division by zero.
SIGILL	Signal: illegal instruction.
SIGINT	Signal: interrupt.
SIGSEGV	Signal: segmentation fault.
SIGTERM	Signal: termination.
...	(Many other signals often defined.)

Macros:

None

Types:

sig_atomic_t	?

Global variables:

None

13 Non-local jumps (setjmp.h)

Functions:

int setjmp(jmp_buf env);	Records caller's environment in 'jump buffer' env and returns 0.
void longjmp(jmp_buf env, int status);	Jumps to 'jump buffer' env and returns status.

Constants:

None

Macros:

None

Types:

jmp_buf	'Jump buffer'

Global variables:

None

14 Localization (locale.h)

Functions:

`char *setlocale(int category, const char *locale);`	Change locale-specific behavior of category to locale.
`struct lconv *localeconv(void);`	Returns info. about how numeric and monetary values are handled in the current locale.

Constants:

LC_ALL	Category: all behavior.
LC_COLLATE	Category: strcoll() and strxfrm().
LC_CTYPE	Category: character functions.
LC_MONETARY	Category: monetary information.
LC_NUMERIC	Category: numeric information.
LC_TIME	Category: strftime().
NULL	Null pointer.

Macros:

None

Types:

struct lconv	Contains info. about how numeric and monetary values are handled in the current locale.

Global variables:

None

15 Facilities

Functions:

None

Constants:

__DATE__	String with date of compilation.
__FILE__	String with name of source file.
__LINE__	Integer with current line number.
__TIME__	String with time of compilation.
__STDC__	Defined in ANSI C compilers.

Macros:

None

Types:

None

Global variables:

None