```
<ctype.h>

int isalnum(int c);
    isalpha(c) or isdigit(c)
int isalpha(int c);
    isupper(c) or islower(c)
int iscntrl(int c);
    is control character. In ASCII, control characters are 0x00 (NUL) to 0x1F
(US), and 0x7F (DEL)
int isdigit(int c);
    is decimal digit
int isgraph(int c);
    is printing character other than space
int islower(int c);
    is lower-case letter
int isprint(int c);
    is printing character (including space). In ASCII, printing characters are
0x20 (' ') to 0x7E ('~')
int ispunct(int c);
    is printing character other than space, letter, digit
int isspace(int c);
    is space, formfeed, newline, carriage return, tab, vertical tab
int isupper(int c);
    is upper-case letter
int isxdigit(int c);
    is hexadecimal digit
int tolower(int c);
    return lower-case equivalent
int toupper(int c);
    return upper-case equivalent

<stdio.h>

BUFSIZ
    Size of buffer used by setbuf.
EOF
    Value used to indicate end-of-stream or to report an error.
FILENAME_MAX
    Maximum length required for array of characters to hold a filename.
FOPEN_MAX
    Maximum number of files which may be open simultaneously.
L_tmpnam
```



```
    Number of characters required for temporary filename generated by tmpnam.
NULL
    Null pointer constant.
SEEK_CUR
    Value for origin argument to fseek specifying current file position.
SEEK_END
    Value for origin argument to fseek specifying end of file.
SEEK_SET
    Value for origin argument to fseek specifying beginning of file.
TMP_MAX
    Minimum number of unique filenames generated by calls to tmpnam.
_IOFBF
    Value for mode argument to setvbuf specifying full buffering.
_IOLBF
    Value for mode argument to setvbuf specifying line buffering.
_IONBF
    Value for mode argument to setvbuf specifying no buffering.
stdin
    File pointer for standard input stream. Automatically opened when program
execution begins.
stdout
    File pointer for standard output stream. Automatically opened when program
execution begins.
stderr
    File pointer for standard error stream. Automatically opened when program
execution begins.
FILE
    Type of object holding information necessary to control a stream.
fpos_t
    Type for objects declared to store file position information.
size_t
    Type for objects declared to store result of sizeof operator.
FILE* fopen(const char* filename, const char* mode);
    Opens file named filename and returns a stream, or NULL on failure. mode
may be one of the following for text files:
    "r"
    text reading
    "w"
    text writing
    "a"
    text append
    "r+"
```

text update (reading and writing)
"w+"
text update, discarding previous content (if any)
"a+"
text append, reading, and writing at end
or one of those strings with b included (after the first character), for binary files.

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

Closes file associated with stream, then opens file filename with specified mode and associates it with stream. Returns stream or NULL on error.

```
int fflush(FILE* stream);
```

Flushes stream stream and returns zero on success or EOF on error. Effect undefined for input stream. fflush(NULL) flushes all output streams.

```
int fclose(FILE* stream);
```

Closes stream stream (after flushing, if output stream). Returns EOF on error, zero otherwise.

```
int remove(const char* filename);
```

Removes specified file. Returns non-zero on failure.

```
int rename(const char* oldname, const char* newname);
```

Changes name of file oldname to newname. Returns non-zero on failure.

```
FILE* tmpfile();
```

Creates temporary file (mode "wb+") which will be removed when closed or on normal program termination. Returns stream or NULL on failure.

```
char* tmpnam(char s[L_tmpnam]);
```

Assigns to s (if s non-null) and returns unique name for a temporary file. Unique name is returned for each of the first TMP_MAX invocations.

```
int setvbuf(FILE* stream, char* buf, int mode, size_t size);
```

Controls buffering for stream stream. mode is _IOFBF for full buffering, _IOLBF for line buffering, _IONBF for no buffering. Non-null buf specifies buffer of size size to be used; otherwise, a buffer is allocated. Returns non-zero on error. Call must be before any other operation on stream.

```
void setbuf(FILE* stream, char* buf);
```

Controls buffering for stream stream. For null buf, turns off buffering, otherwise equivalent to (void)setvbuf(stream, buf, _IOFBF, BUFSIZ).

```
int fprintf(FILE* stream, const char* format, ...);
```

Converts (according to format format) and writes output to stream stream. Number of characters written, or negative value on error, is returned. Conversion specifications consist of:

%
(optional) flag:
-
left adjust

+
always sign
space
space if no sign
0
zero pad
#
Alternate form: for conversion character o, first digit will be zero, for [xX], prefix 0x or 0X to non-zero value, for [eEfgG], always decimal point, for [gG] trailing zeros not removed.
(optional) minimum width: if specified as *, value taken from next argument (which must be int).
(optional) . (separating width from precision):
(optional) precision: for conversion character s, maximum characters to be printed from the string, for [eEf], digits after decimal point, for [gG], significant digits, for an integer, minimum number of digits to be printed. If specified as *, value taken from next argument (which must be int).
(optional) length modifier:
h
short or unsigned short
l
long or unsigned long
L
long double
conversion character:
d,i
int argument, printed in signed decimal notation
o
int argument, printed in unsigned octal notation
x,X
int argument, printed in unsigned hexadecimal notation
u
int argument, printed in unsigned decimal notation
c
int argument, printed as single character
s
char* argument
f
double argument, printed with format [-]mmm.ddd
e,E
double argument, printed with format [-]m.dddddd(e|E)(+|-)xx
g,G

double argument
p
void* argument, printed as pointer
n
int* argument : the number of characters written to this point is written into argument
%
no argument; prints %

```
int printf(const char* format, ...);
```

printf(f, ...) is equivalent to fprintf(stdout, f, ...)

```
int sprintf(char* s, const char* format, ...);
```

Like fprintf, but output written into string s, which must be large enough to hold the output, rather than to a stream. Output is NUL-terminated. Returns length (excluding the terminating NUL).

```
int vfprintf(FILE* stream, const char* format, va_list arg);
```

Equivalent to fprintf with variable argument list replaced by arg, which must have been initialised by the va_start macro (and may have been used in calls to va_arg).

```
int vprintf(const char* format, va_list arg);
```

Equivalent to printf with variable argument list replaced by arg, which must have been initialised by the va_start macro (and may have been used in calls to va_arg).

```
int vsprintf(char* s, const char* format, va_list arg);
```

Equivalent to sprintf with variable argument list replaced by arg, which must have been initialised by the va_start macro (and may have been used in calls to va_arg).

```
int fscanf(FILE* stream, const char* format, ...);
```

Performs formatted input conversion, reading from stream stream according to format format. The function returns when format is fully processed. Returns number of items converted and assigned, or EOF if end-of-file or error occurs before any conversion. Each of the arguments following format must be a pointer. Format string may contain:

blanks and tabs, which are ignored

ordinary characters, which are expected to match next non-white-space of input

conversion specifications, consisting of:

%
(optional) assignment suppression character "*"
(optional) maximum field width
(optional) target width indicator:
h
argument is pointer to short rather than int

l
argument is pointer to long rather than int, or double rather than float
L
argument is pointer to long double rather than float
conversion character:
d
decimal integer; int* parameter required
i
integer; int* parameter required; decimal, octal or hex
o
octal integer; int* parameter required
u
unsigned decimal integer; unsigned int* parameter required
x
hexadecimal integer; int* parameter required
c
characters; char* parameter required; white-space is not skipped, and NUL-termination is not performed
s
string of non-white-space; char* parameter required; string is NUL-terminated
e,f,g
floating-point number; float* parameter required
p
pointer value; void* parameter required
n
chars read so far; int* parameter required
[...]
longest non-empty string from specified set; char* parameter required; string is NUL-terminated
[^...]
longest non-empty string not from specified set; char* parameter required; string is NUL-terminated
%
literal %; no assignment

```
int scanf(const char* format, ...);
```

scanf(f, ...) is equivalent to fscanf(stdin, f, ...)

```
int sscanf(char* s, const char* format, ...);
```

Like fscanf, but input read from string s.

```
int fgetc(FILE* stream);
```

Returns next character from (input) stream stream, or EOF on end-of-file or

error.
char* fgets(char* s, int n, FILE* stream);
    Copies characters from (input) stream stream to s, stopping when n-1 characters copied, newline copied, end-of-file reached or error occurs. If no error, s is NUL-terminated. Returns NULL on end-of-file or error, s otherwise.
int fputc(int c, FILE* stream);
    Writes c, to stream stream. Returns c, or EOF on error.
char* fputs(const char* s, FILE* stream);
    Writes s, to (output) stream stream. Returns non-negative on success or EOF on error.
int getc(FILE* stream);
    Equivalent to fgetc except that it may be a macro.
int getchar(void);
    Equivalent to getc(stdin).
char* gets(char* s);
    Copies characters from stdin into s until newline encountered, end-of-file reached, or error occurs. Does not copy newline. NUL-terminates s. Returns s, or NULL on end-of-file or error. Should not be used because of the potential for buffer overflow.
int putc(int c, FILE* stream);
    Equivalent to fputc except that it may be a macro.
int putchar(int c);
    putchar(c) is equivalent to putc(c, stdout).
int puts(const char* s);
    Writes s (excluding terminating NUL) and a newline to stdout. Returns non-negative on success, EOF on error.
int ungetc(int c, FILE* stream);
    Pushes c (which must not be EOF), onto (input) stream stream such that it will be returned by the next read. Only one character of pushback is guaranteed (for each stream). Returns c, or EOF on error.
size_t fread(void* ptr, size_t size, size_t nobj, FILE* stream);
    Reads (at most) nobj objects of size size from stream stream into ptr and returns number of objects read. (feof and ferror can be used to check status.)
size_t fwrite(const void* ptr, size_t size, size_t nobj, FILE* stream);
    Writes to stream stream, nobj objects of size size from array ptr. Returns number of objects written.
int fseek(FILE* stream, long offset, int origin);
    Sets file position for stream stream and clears end-of-file indicator. For a binary stream, file position is set to offset bytes from the position indicated by origin: beginning of file for SEEK_SET, current position for SEEK_CUR, or end of file for SEEK_END. Behaviour is similar for a text stream, but offset must be zero or, for SEEK_SET only, a value returned by ftell.

Returns non-zero on error.
long ftell(FILE* stream);
    Returns current file position for stream stream, or -1 on error.
void rewind(FILE* stream);
    Equivalent to fseek(stream, 0L, SEEK_SET); clearerr(stream).
int fgetpos(FILE* stream, fpos_t* ptr);
    Stores current file position for stream stream in *ptr. Returns non-zero on error.
int fsetpos(FILE* stream, const fpos_t* ptr);
    Sets current position of stream stream to *ptr. Returns non-zero on error.
void clearerr(FILE* stream);
    Clears end-of-file and error indicators for stream stream.
int feof(FILE* stream);
    Returns non-zero if end-of-file indicator is set for stream stream.
int ferror(FILE* stream);
    Returns non-zero if error indicator is set for stream stream.
void perror(const char* s);
    Prints s (if non-null) and strerror(errno) to standard error as would: fprintf(stderr, "%s: %s\n", (s != NULL ? s : ""), strerror(errno))

<string.h>

NULL
    Null pointer constant.
size_t
    Type for objects declared to store result of sizeof operator.
char* strcpy(char* s, const char* ct);
    Copies ct to s including terminating NUL and returns s.
char* strncpy(char* s, const char* ct, size_t n);
    Copies at most n characters of ct to s. Pads with NUL characters if ct is of length less than n. Note that this may leave s without NUL-termination. Return s.
char* strcat(char* s, const char* ct);
    Concatenate ct to s and return s.
char* strncat(char* s, const char* ct, size_t n);
    Concatenate at most n characters of ct to s. NUL-terminates s and return it.
int strcmp(const char* cs, const char* ct);
    Compares cs with ct, returning negative value if cs<ct, zero if cs==ct, positive value if cs>ct.
int strncmp(const char* cs, const char* ct, size_t n);

Compares at most (the first) n characters of cs and ct, returning negative value if cs<ct, zero if cs==ct, positive value if cs>ct.

```
int strcoll(const char* cs, const char* ct);
```

Compares cs with ct according to locale, returning negative value if cs<ct, zero if cs==ct, positive value if cs>ct.

```
char* strchr(const char* cs, int c);
```

Returns pointer to first occurrence of c in cs, or NULL if not found.

```
char* strrchr(const char* cs, int c);
```

Returns pointer to last occurrence of c in cs, or NULL if not found.

```
size_t strspn(const char* cs, const char* ct);
```

Returns length of prefix of cs which consists of characters which are in ct.

```
size_t strcspn(const char* cs, const char* ct);
```

Returns length of prefix of cs which consists of characters which are not in ct.

```
char* strpbrk(const char* cs, const char* ct);
```

Returns pointer to first occurrence in cs of any character of ct, or NULL if none is found.

```
char* strstr(const char* cs, const char* ct);
```

Returns pointer to first occurrence of ct within cs, or NULL if none is found.

```
size_t strlen(const char* cs);
```

Returns length of cs.

```
char* strerror(int n);
```

Returns pointer to implementation-defined message string corresponding with error n.

```
char* strtok(char* s, const char* t);
```

Searches s for next token delimited by any character from ct. Non-NULL s indicates the first call of a sequence. If a token is found, it is NUL-terminated and returned, otherwise NULL is returned. ct need not be identical for each call in a sequence.

```
size_t strxfrm(char* s, const char* ct, size_t n);
```

Stores in s no more than n characters (including terminating NUL) of a string produced from ct according to a locale-specific transformation. Returns length of entire transformed string.

```
void* memcpy(void* s, const void* ct, size_t n);
```

Copies n characters from ct to s and returns s. s may be corrupted if objects overlap.

```
void* memmove(void* s, const void* ct, size_t n);
```

Copies n characters from ct to s and returns s. s will not be corrupted if objects overlap.

```
int memcmp(const void* cs, const void* ct, size_t n);
```


Compares at most (the first) n characters of cs and ct, returning negative value if cs<ct, zero if cs==ct, positive value if cs>ct.

```
void* memchr(const void* cs, int c, size_t n);
```

Returns pointer to first occurrence of c in first n characters of cs, or NULL if not found.

```
void* memset(void* s, int c, size_t n);
```

Replaces each of the first n characters of s by c and returns s.