## Standard Library

There are 15 standard function libraries in ANSI-C.

| | | | |
|---|---|---|---|
| <assert.h> | <ctype.h> | <errno.h> | <float.h> |
| <limits.h> | <locale.h> | <math.h> | <setjmp.h> |
| <signal.h> | <stdarg.h> | <stddef.h> | <stdio.h> |
| <stdlib.h> | <string.h> | <time.h> | |

You can find these include file at `/usr/include/`.

Check man page section 3C for more information.





## <assert.h>

macro:

```
void assert(int expression);
```

Macro used to add diagnostics. If expression is false, message printed on stderr and abort called to terminate execution. Source file and line number in message come from preprocessor. Ex.

```
assert (x!=NULL);
```

If `x` is `NULL` then the execution will stop and output:

`Assertion failed: x!=NULL, file` *xyz*`, line` *nnn*

If `NDEBUG` is defined where `assert.h` is included, assert macro is ignored.





## <ctype.h>

These functions allow the user to test characters and determine their type (space, number, upper-case, etc.). The functions return a non-zero value (true) when the condition is satisfied, and zero otherwise.

```
int isalpha(int c);      (isupper(c)||islower(c))
int isupper(int c);
int islower(int c);
int isdigit(int c);
int isxdigit(int c);     c ∈ [0-9, A-F, a-f]
int isalnum(int c);      (isalpha(c)||isdigit(c))
int isspace(int c);
int ispunct(int c);
```





```
int isprint(int c);
int isgraph(int c);      (isalnum(c)||isspace(c))
int iscntrl(int c);      (Control Character )
int isascii(int c);      (0 ≤ c ≤ 127)
```

The following two new functions swap a letter's case:

```
int tolower(int c);
int toupper(int c);
```

## <stdio.h>

This header file contains macros, type definitions, and prototypes of functions used to access files. More details refer to ABC A.12, P 653.

```
FILE
FILE* fopen(const char* filename, const char* mode);
int fflush(FILE* stream);
int fclose(FILE* stream);
int fprintf(FILE* stream, const char* format, ...);
int fscanf(FILE* stream, const char* format, ...);
```





## <stdio.h> FILE

`FILE`
Type which records information necessary to control a stream.

When a program begins execution, the following pointer to `FILE` automatically opened:

| | |
|---|---|
| `extern FILE* stdin;` | Standard Input. |
| `extern FILE* stdout;` | Standard Output. |
| `extern FILE* stderr;` | Error information Output. |

In `sdtio.h`, the following are also defined `NULL`, `EOF`, `FILENAME_MAX`, `FOPEN_MAX`,





## fopen, fclose, fflush

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

Opens file filename and returns a stream, or NULL on failure. mode may be (combinations of):

```
"r"  text reading
"w"  text writing; discard previous content
"a"  text append; writing at end
"r+" text update
"w+" text update; discard previous content
"a+" text append; writing at end
```





`int fflush(FILE* stream);`

Flushes stream stream. Effect undefined for input stream. Returns EOF for write error, zero otherwise. fflush(NULL) flushes all output streams.

`int fclose(FILE* stream);`

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

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

The `freopen` function first attempts to flush the `stream` and close any file descriptor associated with `stream`. Then it opens the file whose pathname is the string pointed to by filename and associates the stream pointed to by stream with it. The `mode` argument is used just as in `fopen`.
The `freopen` function is typically used with preopened streams, stdin, stdout and stderr to change buffer-property.





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

returns the next character from the named input `stream` or `stdin`.

```
int putchar(int c);
int putc(int c, FILE *stream);
```

writes a character to `stdout` or `stream`.





```
char *gets(char *s);
char *fgets(char *s, int n, FILE *stream);
```

reads characters from `stream` or `stdin` until a newline character is read or an end-of-file condition is encountered.The newline character is discarded and the string is terminated with a null character.
If a read error occurs, a null pointer is returned and the error indicator for the `stream` is set.

```
int puts(const char *s);
int fputs(const char *s, FILE *stream);
```

writes the string pointed to by `s` followed by a NEWLINE character to `stdout` or `stream`.





## fprintf (I)

```
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 NULL-terminated. Return length does not include the NULL.

## fprintf (II)

```
Length modifier:
        h : short or unsigned short
        l : long or unsigned long
        L : long double
    Conversions:
    d, i : int; signed decimal notation
    o : int; unsigned octal notation
    x,X : int; unsigned hexadecimal notation
    u : int; unsigned decimal notation
    c : int; single character
    s : char*;
    f : double; [-]mmm.ddd
    e,E : double; [-]m.dddddde(+|-)xx
```





```
g,G : double
p : void*; print as pointer
n : int*; number of chars written into arg
% : print %
```





## fscanf

```
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 fflush(FILE *stream);`

causes any unwritten data for that stream to be written to the file.
If stream is NULL, fflush() performs this flushing on all streams.

`int feof(FILE *stream);`

returns non-zero when `EOF` has previously been detected reading the named input `stream` otherwise zero.

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

It causes the array pointed to by buf to be used instead of an automatically allocated buffer. If buf is the NULL pointer, input/output will be completely unbuffered.

`int fgetpos(FILE *stream, fpos_t *pos);`

stores the current value of the file position indicator for the stream pointed to by `stream` into `pos`.

`int fsetpos(FILE *stream, const fpos_t *pos);`

sets the file position indicator for the stream pointed to by `stream` according to the value of the object pointed to by `pos`.





`long ftell(FILE *stream);`

return the file position in `stream`.

`int fseek(FILE *stream, long offset, int whence);`

sets the file-position indicator for the stream pointed to by `stream` according to `whence`:

SEEK_SET set position equal to offset bytes from the beginning.
SEEK_CUR set position to current location plus `offset`
SEEK_END set position to EOF plus `offset`.

`void rewind(FILE *stream);`

is equivalent to: (void) fseek(stream, 0L, SEEK_SET)





`FILE *tmpfile(void);`
creates a temporary file and opens a corresponding stream
The file will automatically be deleted when all references to the file are closed.

`char *tmpnam(char *s);`
generate file names that can safely be used for a temporary file.
If s is NULL, tmpnam() leaves its result in an internal static area.





## <stdlib.h>

```
double atof(const char* s);
int atoi(const char* s);
long atol(const char* s);
int rand();
void srand(unsigned int seed);
void* calloc(size_t nobj, size_t size);
void* malloc(size_t size);
void* realloc(void* p, size_t size);
void free(void* p);
void abort();
void exit(int status);
```

## <string.h>

```
char *strcat(char *dst, const char *src);
char *strncat(char *dst, const char *src, size_t n);
char *strchr(const char *s, int c);
char *strrchr(const char *s, int c);
int strcmp(const char *s1, const char *s2);
int strncmp(const char *s1, const char *s2, size_t n);
char *strcpy(char *dst, const char *src);
char *strncpy(char *dst, const char *src, size_t n);
char *strdup(const char *s1);
size_t strlen(const char *s);
```





## <string.h> (I)

strcat() appends a copy of string src, including the ter- minating null character, to the end of string dst. strncat() appends at most n characters. Each returns a pointer to the null-terminated result. The initial charac- ter of src overrides the null character at the end of dst.

strchr() returns a pointer to the first occurrence of c (converted to a char) in string s, or a null pointer if c does not occur in the string. strrchr() returns a pointer to the last occurrence of c. The null character terminating a string is considered to be part of the string.





## <strings.h>

```
int strcasecmp(const char *s1, const char *s2);
int strncasecmp(const char *s1, const char *s2, int n);
```

strcasecmp() and strncasecmp() are case-insensitive versions of strcmp() and strncmp() respectively, described below. strcasecmp() and strncasecmp() assume the ASCII character set and ignore differences in case when comparing lower and upper case characters.





## Other C Library

Socket Library

System Library

Regular expressions Library

GLIB

.....

## System Library (I)

Most of these calls have one or more error returns. An error condition is indicated by an otherwise impossible returned value. This is almost always -1 or the null pointer; the individual descriptions specify the details. An error number is also made available in the external variable errno. errno is not cleared on successful calls, so it should be tested only after an error has been indicated.

**Streams** A set of kernel mechanisms that support the development of network services and data communication drivers. It defines interface standards for character input/output within the kernel and between the kernel and user level processes. The STREAMS mechanism is composed of utility routines, kernel facilities and a set of data structures.





## System Library

```
int open(const char *path, int oflag,/* mode_t mode */ ...);
ssize_t read(int fildes, void *buf, size_t nbyte);
ssize_t write(int fildes, const void *buf, size_t nbyte);
off_t lseek(int fildes, off_t offset, int whence);
int close(int fildes);
pid_t fork(void);
int pipe(int fildes[2]);
```





## System Library (Streams functions)

The **open()** function establishes the connection between a file and a file descriptor.

The **close()** function will deallocate the file descriptor indicated by fildes. To deallocate means to make the file descriptor available for return by subsequent calls to open(2) or other functions that allocate file descriptors.

The **read()** function attempts to read nbyte bytes from the file associated with the open file descriptor, fildes, into the buffer pointed to by buf.

The **write()** function attempts to write nbyte bytes from the buffer pointed to by buf to the file associated with the open file descriptor, fildes.





## System Library (Streams functions)

The **lseek()** function sets the file pointer associated with the open file descriptor specified by fildes as follows:

```
If whence is SEEK_SET, the pointer is set to off-set bytes.
If  whence is SEEK_CUR, the pointer is set to its current
 location plus offset.
If whence is SEEK_END, the pointer is set to  the size of
 the file plus offset.
```

Some devices are incapable of seeking. The value of the file pointer associated with such a device is undefined.

## System Library (fork)

The fork() function creates a new process. The new process (child process) is an exact copy of the calling process (parent process). The child process inherits the following attributes from the parent process: UID, GID and ENV etc.

```
.....
if (fork())

   childtask();
   exit(0);

.....
```





## System Library (pipe)

**pipe()** creates an I/O mechanism called a pipe and returns two file descriptors, `fildes[0] and fildes[1]`. The files associated with `fildes[0] and fildes[1]` are streams and are both opened for reading and writing.

A read from `fildes[0]` accesses the data written to `fildes[1]` on a first-in-first-out (FIFO) basis and a read from `fildes[1]` accesses the data written to `fildes[0]` also on a FIFO basis.