Operator Precedence Table

Operator	Desc.	Direction
`i++ i--`	post-in/decr	$\longrightarrow$
`() [] . ->`	fn, arr, struct, struct thru pointer	
`++i --i`	pre-in/decr	$\longleftarrow$
`+ -`	unary $+, -$	
`! ~`	$\neg$, bitwise NOT	
`(type)`	cast	
`* &`	deref, addr of	
`* / %`	$\times, \div$, mod	$\longrightarrow$
`+ -`	binary $+, -$	
`< <= > >=`	relational $<, \leqslant, >, \geqslant$	
`== !=`	relational $=, \neq$	
`& ^ |`	Bitwise AND,XOR,OR	
`&&`	Logical $\wedge$	
`||`	Logical $\vee$	
`?:`	Ternary	$\longleftarrow$
`= += -= …`	Assignment	
`,`	Comma	$\longrightarrow$

stdlib.h rand, time.h time

```
int rand(void);
```
The `rand()` function computes a sequence of pseudo-random integers in the range of `0` to `RAND_MAX`.

```
void srand(unsigned seed);
```
The `srand()` function sets its argument `seed` as the seed for a new sequence of pseudo-random numbers to be returned by `rand()`. These sequences are repeatable by calling `srand()` with the same seed value.

```
time_t time(time_t *tloc);
```
The `time()` function returns the value of time in seconds since 0 hours, 0 minutes, 0 seconds, January 1, 1970, Coordinated Universal Time, without including leap seconds.

string.h

```
char *strcpy(char *dst, char *src);
```
The `strcpy()` function copies the string `src` to `dst` (including the terminating `` `\0' `` character).

```
char *strcat(char *s, char *append);
```
The `strcat()` function appends a copy of the null-terminated string `append` to the end of the null-terminated string `s`, then add a terminating `` `\0' ``. The string `s` must have sufficient space to hold the result.

```
char *strstr(char *big, char *little);
```
The `strstr()` function locates the first occurrence of the null-terminated string `little` in the null-terminated string `big`. If `little` is an empty string, `big` is returned; if `little` occurs nowhere in `big`, `NULL` is returned; otherwise a pointer to the first character of the first occurrence of `little` is returned.

```
char *strchr(char *s, int c);
```
The `strchr()` function locates the first occurrence of `c` (converted to a `char`) in the string pointed to by `s`. The terminating null character is considered part of the string; therefore if `c` is `` `\0' ``, the functions locate the terminating `` `\0' ``. Returns a pointer to the located character, or `NULL` if the character does not appear in the string.

```
char *strtok(char *str, char *sep);
```
The `strtok()` function is used to isolate sequential tokens in a null- terminated string, `str`. These tokens are separated in the string by at least one of the characters in `sep`. The first time that `strtok()` is called, `str` should be specified; subsequent calls, wishing to obtain further tokens from the same string, should pass a null pointer instead. The separator string, `sep`, must be supplied each time, and may change between calls.

```
char str[80] = "cs1010-abc!def@123";
char sep[10] = "-!@";
char *token;
token = strtok(str, sep);
while (token != NULL) {
    printf("%s\n", token);
    token = strtok(NULL, sep);
}
```

File Processing

```
FILE *fopen(char *path, char *mode);
```

The `fopen()` function opens the file whose name is the string pointed to by `path` and associates a stream with it.

The argument `mode` points to a string beginning with one of the following letters:

`"r"`	Open for reading. The stream is positioned at the beginning of the file. Fail if the file does not exist.
`"w"`	Open for writing. The stream is positioned at the beginning of the file. Create the file if it does not exist.
`"a"`	Open for writing. The stream is positioned at the end of the file. Create the file if it does not exist.

An optional `"+"` following `"r"`, `"w"` or `"a"` opens the file for both reading and writing.

Upon successful completion `fopen()` returns a `FILE` pointer. Otherwise, `NULL` is returned and the global variable `errno` is set to indicate the error.

```
int feof(FILE *stream);
int ferror(FILE *stream);
```

The function `feof()` tests the end-of-file indicator for the stream pointed to by `stream`, returning non-zero if it is set.

The function `ferror()` tests the error indicator for the stream pointed to by `stream`, returning non-zero if it is set.

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

Returns number of input items assigned, which can be fewer than provided for, or even zero, in the event of a matching failure. Zero indicates that, while there was input available, no conversions were assigned; typically this is due to an invalid input charactersuch as an alphabetic character for a '`%d`' conversion. The value `EOF` is returned if an input failure occurs before any conversion such as an end-of-file occurs. If an error or end-of-file occurs after conversion has begun, the number of conversions which were successfully completed is returned.

```
char *fgets(char *str, int size, FILE *stream);
```

The `fgets()` function reads at most one less than the number of characters specified by `size` from the given `stream` and stores them in the string `str`. Reading stops when a newline character is found, at end-of-file or error. The newline, if any, is retained. If any characters are read and there is no error, a `` `\0' `` character is appended to end the string.

```
char *gets(char *str);
```

The `gets()` function is equivalent to `fgets()` with an infinite `size` and a `stream` of `stdin`, except that the newline character (if any) is not stored in the string. It is the caller's responsibility to ensure that the input line, if any, is sufficiently short to fit in the string.

Upon successful completion, `fgets()` and `gets()` return a pointer to the string. If end-of-file occurs before any characters are read, they return `NULL` and the buffer contents remain unchanged. If an error occurs, they return `NULL` and the buffer contents are indeterminate. The `fgets()` and `gets()` functions do not distinguish between end-of-file and error, and callers must use `feof(3)` and `ferror(3)` to determine which occurred.

File I/O Usage

```
FILE *readfp, *writefp;
...
if ((readfp = fopen("infile.in", "r")) == NULL) {
    printf("cannot open `infile.in' for reading\n");
    // bail/exit/return
}
if ((writefp = fopen("outfile.out", "w")) == NULL) {
    printf("cannot open `outfile.out' for writing\n");
    // bail/exit/return
}
/* do I/O on readfp/writefp for example */
char line[LENGTH+1];
while (fgets(line, LENGTH+1, readfp) != NULL) {
    // do stuff
}
if (!feof(readfp))
    // read error: bail/exit/return
fclose(readfp);
...
```

Corner cases

```
/* arrays */
char *str = "string literal";  // READ-ONLY MEMORY!
char str[20] = "modifiable";
```

```
int x[8] = {1,2,3};          // missing elements are all  zeroes
int y[3] = {1,2,3,4};        // 4 will  be truncated

/* 2nd dimension compulsory for matrix */
int m[4][3] = {              // 4x3 matrix
    { 1 },                   // row 0 is  { 1,  0,  0 }
    { 0, 1 },                // row 1 is  { 0,  1,  0 }
};                           // everything else  are zeroes
int m[3][2] = {1,3,2};       // will  automatically be flattened.

int arr[2][3], arr2[3] = {1,1,1}
arr[0] = arr2;   // error:  array type int[3]  is  not assignable

/* Struct */
typedef struct s_t {
    int a;
    float b;
    char* name;
} s_t;
s_t st = { 5, 2.2, "George" };
f(st);                       // structures  are copied

/* I/O */
puts("blabla");              // write(1,  "blabla\n",  7);
fputs("blabla", fp);         // write(?,  "blabla",  6);
```