**[MAKEFILE](https://www.math.colostate.edu/~yzhou/computer/writemakefile.html)**

[**Basic function and their associated libraries**](https://www.ibm.com/docs/en/i/7.3?topic=extensions-standard-c-library-functions-table-by-name)

**NOMENCLATURE**

**Declaration** – introduces an identifier and its type.

* double f(int, double); // for functions
* extern int value; // for variables
* no memory is allocated

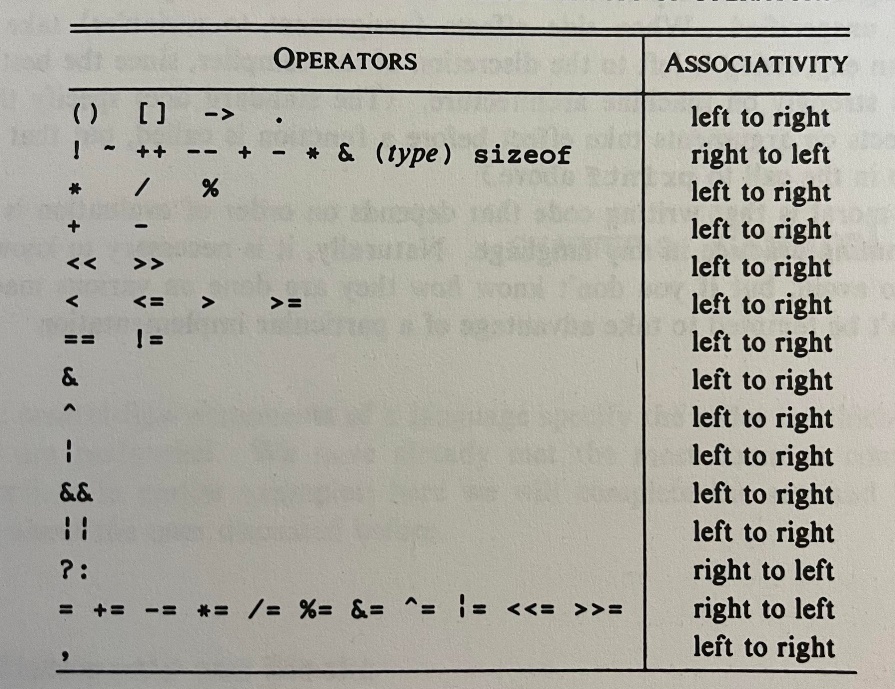
**Definition** – implements an identifier, such as when defining external variables outside functions.

* double f(int i, double d) {return i+d;} // for functions
* int value; // for variables
* memory is allocated

**Initialization** – sets value to each of the variables

* external and static variables are initialized to 0 upon definition.
* Automatic and register variables are not initialized automatically.

**operators –** an operator acting on a variable. Below are the priorities.



* When working with operands of different types, promotes lower operand type to higher before proceeding. Here’s the order from high to low: long double -> double -> float -> int -> short/char

**bitwise Operators** – operator that will strictly work in the bit domain

* &(and) |(inclusive or) ^(exclusive or) <<(left shift) >>(right shift) ~(one’s compliment)

**assignment –** assigning a value to a defined variable. Assignment itself return the value of the entity to the right-hand side of the assignment.

* **Print(“%d\n”, a = 3)** will print 3
* **a = b = c = 1;** is a valid assignment, which will set all variables to 1

[**dynamic memory allocation**](https://www.geeksforgeeks.org/dynamic-memory-allocation-in-c-using-malloc-calloc-free-and-realloc/)– is a way to allocate memory on the fly. One example is to use **malloc()**:

* **int \* nums = (int\*) malloc(n \* sizeof(int));** *this creates an array of ints*
* **int\* nums = (int\*) calloc,(n, sizeof(int));** *another way to create that array*
* **int = realloc(int, m\*sizeof(int));** *will reallocate the space of nums from n\*4 to m\*4 bytes*

**memory freeing** – a way to release the space used by pointer data to be used by other variables.

* To free nums you run free(nums[i]) for I = 0..n and then free(nums)

**stdout** – stands for standard output stream, and is available to the program by default.

* **stdin** and **stderr** are teo of the other streams

**Tokens** – smallest element of a program that is meaningful to the compiler, e.g. keywords (if, for, …) , identifiers (begin with \_) , constants, strings, operators (+, \*, / , …)

**#include** – used to include C packages and is run in the precompile stage

* **<filename>** will search for a filename in the include path
* **“filename”** will search first for a file in current directory first and then In the include path

**#define –** used to define symbolic names or constants and is run in the precompile stage

* **#define pi 3.1415926359**
* **#define NAME ROMAN**
  + An example of macro substitution
* **#define forever for (;;)** 
  + Defines an infinite loop
* **#define max(A,B) ((A)>(B)) ? (A) : (B);** 
  + Defines a macro to compute max
  + Parentheses are used deliberately to make sure that when some entity such as 8+1 is plugged in for A and 3 +4 for B the computation is 9>7 and not 8+1>3+4
* **#if / #else / #endif** – offer conditional inclusions that get sorted out in the precompile

**Pointer** – a way to reference a value by pointing to its address.

* b = &a lets b point to address of a
* c = \*b lets c take on the value of the pointer, in this case a
* **using pointer in functions** – void func(\*a,\*b) {\*a = 1; \*b=1;}; func(&a,&b); will pass addresses of a and b and change those values to 1.
* **Arrays** – are implicitly pointer entities
  + **For array a, &a[0]** – returns address of first element in a. this is equivalent to just **a**.

**Arrays** – array represents a pointer to its first element, thus &a[0] is equivalent to a

* To define an array element of string, we can use the following, where in the latter case we have to specify the maximum length of each string and in the first case we are not required to.
  + char \*mystrings[]
  + char mystrings[][15]
* \*++argv[0] will increment address along the first entry in argv, thus returning its elements in turn. **[0]** is processed first, then **++**, then **\*** . **\*** Returns the value, which is a pointer to an array, and [0] indexes that array.
* (\*++argv)[0] will increment address along every entry in argv and return their first element in turn. ++ is processed first, then \*, then [0]. Alternative, equivalent notation would ne \*\*++argv
* [Passing arrays to functions](https://www.geeksforgeeks.org/pass-2d-array-parameter-c/) also page 112 of TCPL

**#undef** – undefines a name

**;** - used to terminate line statements. Can also be used inside empty functions as a null statement.

**EOF** – in terminal is created via CTRL+Z

**Parameter/formal argument** – variable name in the parenthesized function definitions

**Argument/actual argument** – values used to call the function with

**function initialization -**

* Passing strings
  + func(char a[], char b[])
  + func(char \* a, char \* b)

**scope** – part of the program within which the name can be used.

**static** – a declaration applied to external variables that limit the scope of that variable to the file in which it is defined alone. If defined within a function, they will retain a global value between calls to that function. *There are two types of statics so be careful!*

**DATA TYPES**

**size\_t** is the base unsigned integer.

**ssize\_t** is the base signed integer.

**Const char \*** means that the character being pointed to is unchangeable.

**Const \* char** means that the pointer is unchangeable but what it oints to is changeable

**hex –** represented as ‘\x0FF’

**oct** – represented as 077 or ‘\077’

**Enum** – enumerate constant, which is a list of constant values.

* **enum weather {RAINING, WINDY, COLD}**

**typedef** – gives a data type a new name which can be used to initialize variables

* typedef unsinged char BYTE;
  + BYTE b1 = ‘a’;
* typedef struct NAME {char \* name = “Roman”} myname;
  + we can now initialize with: myname newname = “Alex”;

[**struct**](https://stackoverflow.com/questions/1675351/typedef-struct-vs-struct-definitions)–short for ‘structure’ these elements define a new data type with more than one member.

* Struct Example{ int a; int b; } ex1; *creates a struct named ex1.*
* In struct, each member gets a separate space in memory and total memory is equal to the sum of each individual memory sizes
* Struct definition **cannot** reference itself but can reference a pointer to itself.

**union** – is like struct in syntax and allows for string multiple different data type in the same memory location. Unlike a struct, the total memory allocated is based on the largest memory member.



**enum –** is a data type used to associate names to indexes, like dictionaries in python.



**macros** – are inline function definitions. More information can be found [here](https://gcc.gnu.org/onlinedocs/cpp/Macros.html).

**Pointers** – allows you to store a datatype which will point to an address rather than store a value.

* Let **v** be declared a variable. Then **&v** will return its address.
* Let **\*a** be declared as a pointer variable. Then **a** will return its address and \*a will return its value.
* We can now assign **&v** to **a** which means **\*a** will return whatever values **v** takes. Note that reverse assignment is impossible.
* Int arr[10];
  + arr **and** &arr[0] are both addresses of he first element in arr

**NULL** – will evaluate to 0 when casted as an int.

**directives** – commands translated by the preprocessor.

* examples: #include, #define, #error, #ifndef/#ifdef/#elif/#endif

**preprocessor** – is not a part of the compiler and is simply a text substitution tool.

**static** – functions can be accessed only in the file they were created.

**volatile -** This declaration sets a particular type to be volatile meaning that it can change its value due to instances others than within doe, i.e., interrupts. This declaration forces the compiler to pay attention to these variables changing value. This is a very important declaration to be made in embedded code for variables that are fed register values that are subject to change.

**compound Literal** – can be used to initialize a struct. From the struct type above, we can instantiate one with the following line:

* struct Example example = {1,2};

**stdin, stdout, stderr** – are three pointers which can interact with a program and putc/getc/printf functions

**FUNCTIONS**

**main()** – is the function where the program will begin executing at beginning of main

**printf()** – a nonstandard output stream function

* **printf(“%3d”, num)** will print num as an integer, right justified, with at least 3 characters (numbers smaller than 999 will simply be filled with empty space where there is no digit)
* **printf(“%3.2f”, num)** will print num as a float with at least 3 numbers to the left of decimal, and exactly 2 numbers after the decimal

**for –** a method fir iteratively looping. **For(initialization; test condition; function if test is true) { *stuff* }**

* the loop terminates when test condition becomes false

**continue** is a statement which can be passed within if() to exit out of a for loop iteration and move on to the next one. Functions similar to break.

**?** expression is used to reduce the code size of implementing if then statement. See example:

* string num\_text[] = (s[i] == 0) ? ‘zero’ : ‘nonzero’;

**memcpy() -** copies over the contents that a given pointer points to, to a new location.

void \*memcpy(void \*dest, const void \* src, size\_t n)

**char c = getchar()** – will get the next character input by the input stream and save it under c

**putchar(c)** – will place c into the output stream

**switch** – a multiway decision test

* **switch (expression) {**

**case const-expression : statements**

**case const-expression : statements**

**default: statements**

**}**

* must use **break** or **return** statements to get out of the decision test upon hitting ‘true’

**do-while** – one of the loop statements in C, where the body is executed *at least once*

* **do**

**statements**

**while (expression);**

**break** – causes innermost enclosing loop or switch to exit immediately

**continue** – causes the next iteration of enclosing loop to begin, does not apply to switch

**goto –** used to hop out of a loop to a set location within the code

* **for (){**

**for (){**

**if (){**

**goto handlegoto;**

**}**

**}**

**}**

**statements**

**handlegoto:**

**statements**

* goto will skip all other statements between where it is called and where is implemented when and only when handled.

**function** – minimal function is:

* dummy () { }

**getc(stdin) / getc(\*FILENAME) / getchar()** – will get the next character from the terminal or file

**fscanf(\*FILENAME, \*FORMAT)** – will read a file under a specified format string, e.g., “%s”

**sscanf(char \*string, char \*format, arg1, arg2, ..)** – will take the string and populate arg1, arg2, … pointers with entries in string.

**Scanf(char \* format, arg1, arg2, …)** – will scan stdin and populate arg1, arg2, … pointers according to format string.

* Scanf will ignore spaces and tabs but other formatting, such as dashes, lines, etc.
* Scanf and sscanf will return the number of elements successfully picked out from the string

**exit()** – terminates the entire program immediately, unlike **return** which returns from current functions only. **exit(0)** terminates without errors whereas **exit(1)** terminates with an error.

**char \*fgets(char \*line, int maxline, FILE \*fp)** – gets the next input line from the fp into the character array.

**Int fputs(char \*line, FILE \*fp)** – will print line to the file and return EOF if there is an error and 0 otherwise

**Long lseek(int fd, long offset, int origin)** – will current position in file fd to a place specified by origin with offset.

MACROS

[**How to define multiline macro**](https://stackoverflow.com/questions/1067226/c-multi-line-macro-do-while0-vs-scope-block)

GENERAL NOTES

**register –** initializer tells the compiler that the variable will be heavily used and should be kept in the processor register.

**Open files** – are terminated when the program exists via return in main or exit() elsewhere.

**Variadic functions** – are those that take variable number of inputs.

EMBEDDED SPECIFIC

**Volatile** – declaration must be made on variables that are subject to change suddenly (via an interrupt or alike) otherwise the compiler may optimize the code to not detect change in variable values and not recompute equations where is used.

**Atomic** – actions that cannot be interrupted by anything else in the system. These exist to eliminate race conditions.

**Pitchfork -** is a popular build system[layout](https://github.com/vector-of-bool/pitchfork)

* **tests** – contains unit tests.
* **src** contains the makefile and potentially all source files
  + alternatively, **lib** will contain all the library files
* README

PRINTING

**%o** – prints octal

**%x** – prints hexadecimal

**execvp()** – executes a command

**fork()** – starts a child process to run concurrently to parent

**pid\_t** – data type associated with