# Functions in the C programming Language

The C language is similar to most modern programming languages in that it allows the use of functions, self contained "modules" of code that take inputs, do a computation, and produce outputs. C functions must be TYPED (the return type and the type of all parameters specified).

# Functions in C

As always, a function is a module of code that takes information in (referring to that information with local symbolic names called parameters), does some computation, and (usually) returns a new piece of information based on the parameter information.

---

## Basic Function Design Pattern

For the basic syntax of a function in C, please refer to the C Function Design Pattern chapter.

---

## Dot C files

The "recipe" for a function (the function's code) is always stored in a ".C" file. In C there can be many functions written in a single file.

---

## Ordering of functions in a file

The order of functions inside a file is arbitrary. It **does not matter** if you put function one at the top of the file and function two at the bottom, or vice versa.

***Caveat:*** *In order for one function to "see" (use) another function, the "prototype" of the function must be seen in the file before the usage. If a function uses another function that is textually written above it in the file, then this will automatically be true. If the function uses a function that is "below it" in a file, then the prototype should occur at the top of the file... see prototypes below.*

---

## A Function Prototype

In C, all functions must be written to return a specific TYPE of information and to take in specific types of data (parameters). This information is communicated to the compiler via a function prototype.

Here is the syntax for the function declaration or **Prototype**:

```
_TYPE name_of_function ( PARAMETER_TYPE name_of_param,PARAMETER_TYPE name_of_param, etc);

e are some examples of prototypes used at the top of a file:
sqrt( float x );

average( int grades[], int length );
```

A Prototype can occur at the top of a C source code file to describe what the function returns and what it takes (return type and parameter list). When this is the case (occuring at the top of the file), the function prototype **should be followed by a semi-colon**

The function prototype is also used at the beginning of the code for the function. Thus the prototype **can occur twice** in a C source code file. When the prototype occurs with the code **NO semicolon is used**.

---

## The Main Function

In C, the "main" function is treated the same as every function, it has a return type (and in some cases accepts inputs via parameters). The only difference is that the main function is "called" by the operating system when the user runs the program. Thus the main function is always the first code executed when a program starts.

```
int                          // the main function will usually returns a 0 if successful
main()                       // this is the name, in this case: main
{

                             // this is the body of the function (lots of code can go here)
}
```

---

## Examples of C Functions:

```
double                                  // this is the return type
max( double param1, double param2)      // this is the name, followed by the parameters
{
  if (param1 > param2)
    {
      return param1;  // Notice: that param1 is of type double and the return
                      //         type is also of type double
    }
  else
    {
      return param2;
    }
}



void          // This is the return type (void means no value is computed and returned by t
print_happy_birthday( int age )
{
   printf("Congratulations on your %d th Birthday\n", age);
   return;  // you can "terminate" a void function by using return.
   // HERE it is REDUNDANT because the function is over anyway.
}
```

---

## Return Type of a C function

Every C function must specify the type of data that is being generated. For example, the max function above returns a value of type "double". Inside the function, the line "return X;" must be found, where X is a value or variable containing a value of the given type.

### The return statement

When a line of code in a function that says: "return X;" is executed, the function "ends" and no more code in the function is executed. The value of X (or the value in the variable represented by X) becomes the result of the function.

---

## Calling a C function (aka invoke a function)

When one piece of code invokes or calls a function, it is done by the following syntax:

```
        variable = function_name ( args, ...);
```

The function name must match exactly the name of the function in the function prototype. The args are a list of values (or variables containing values) that are "passed" into the function.

The number of args "passed" into a function **must exactly match** the number of parameters required for the function. The type of each arg **must exactly match** the type of each parameter. The return variable type **must exactly match** the return type of the function.

The "variable" in the example above must have a type equivalent to the return type of the function. Inside the function, somewhere will be the line "return X". The value of X is then copied into the "variable".

---

## Parameters in C functions

A Parameter is the symbolic name for "data" that goes into a function. There are two ways to pass parameters in C: Pass by Value, Pass by Reference.

- **Pass by Value**

  Pass by Value, means that a copy of the data is made and stored by way of the name of the parameter. Any changes to the parameter have **NO** affect on data in the calling function.

- **Pass by Reference**

  A **reference parameter** "refers" to the original data in the calling function. Thus any changes made to the parameter are **ALSO MADE TO THE ORIGINAL** variable.

  There are two ways to make a pass by reference parameter:

  1. **ARRAYS**

     Arrays are **always** pass by reference in C. Any change made to the parameter containing the array will change the value of the original array.

  2. The ampersand used in the function prototype.

function ( **& parameter_name** )

To make a normal parameter into a pass by reference parameter, we use the "& param" notation. The ampersand (&) is the syntax to tell C that any changes made to the parameter also modify the original variable containing the data.

---

## Pass by Value Example:

In C, the default is to pass by value. For example:

```
//
// C function using pass by value. (Notice no &)
//
void
doit( int x )
{
     x = 5;
}

//
// Test function for passing by value (i.e., making a copy)
//
int
main()
{
  int z = 27;
  doit( z );
  printf("z is now %d\n", z);

  return 0;
}
```

## Pass by Reference Example:

### Warning: C++

I suggest that you use a C++ compiler such as g++ which allows the following pass by reference syntax (a much more modern style). **The Syntax is to use the '&' in front of the parameter name in the function declaration**. The calling code and usage inside the function are the same as before. For example:

```
//
// C++ using Reference Parameter!
//
void
doit( int & x )
{
     x = 5;
}


//
// Test code for passing by a variable by reference
```

```
//
int
main()
{
  int z = 27;
  doit( z );
  printf("z is now %d\n", z);

  return 0;
}
```

## Warning: Standard C - Using "Pointers"

With standard C you have to put the & in the calling location as opposed to next to the parameter in the function declaration; further, you must use a '*' in the parameter list, and use a '*' whenever using the parameter inside the function.

**The '*' is used to define a "pointer", a discussion of which is beyond the scope of this simple example.** Feel free to Google "Pointers in C" for a long treatise on how to use them... or take my advice, and (as a beginning programmer) avoid them.

```
//
// "Pure" C code using Reference Parameter! (aka pointers)
//
void
doit( int * x )
{
    *x = 5;
}


//
// Test code for passing by a variable by reference
// Note the use of the & (ampersand) in the function call.
//
int
main()
{
  int z = 27;
  doit( & z );
  printf("z is now %d\n", z);

  return 0;
}
```

In summary, if you use a reference parameter, any changes to the parameter inside the function are reflected "outside" of the function (i.e., in the calling function)! If you don't use the & (pass by reference), then we get the same behavior as in Matlab (i.e., the value is changed inside the called function, but maintains its original value in the calling function.

One reason to use reference parameters is to make the program more "efficient". Consider passing in a structure as a parameter. If the structure is very big, and we copy all of it, then we are using a lot of unnecessary memory.

## Array Parameter Example (ALWAYS pass by reference)

Arrays are always **passed by reference** in C. They **do not** use the '&' notation, but are pass by reference none the less. For example:

```
//
// Initialize an array with values 1,2,3,...,length_of_array
//
// Notice: Any changes made to "array_variable" are reflected in
//         the calling code! Arrays are pass by reference!
//
// Notice: There is no return statement, but still the array is changed
//         and can be said to be "returned" to the calling function.
//
void
build_array( int array_variable[], int length_of_array )
{
    for (int i=0; i<length_of_array; i++)
      {
        array_variable[i] = i;
      }
}


//
// Test code for passing an array by reference
//
int
main()
{
  int values[50];

  printf("the value at location 7 starts as %d\n", values[7]);

  build_array(values, 50);

  printf("the value at location 7 is now %d\n", values[7]);

  return 0;
}
```

## Constant Reference

To protect from accidentally changing a reference parameter, when we really want it not to be changed (we just want to save time/memory) we can use the C keyword **const**. For example:

```
//
// C Code using a CONSTANT reference Parameter
//
void
doit( const int & x )
{
    x = 5; // ILLEGAL
}

//
```

```
          // Main Function
          //
          int
          main()
          {
            int z = 27;
            doit( z );
            printf("z is now %d\n", z);

            return 0;
          }
```

## Void Functions

If a function does not return a value, then a special "TYPE" is used to tell the computer this. The return type is "void" (all lower case).

Void functions are mostly used in two classes of functions.

1. The first is a function that prints information for the user to read. For example (for our purposes), the printf function is treated as a void function. (In actuality, printf returns an integer which is the number of characters printed... but we almost always ignore this value.)

2. The second use of void functions is with "reference" parameters (e.g., Arrays). A reference parameter is **not** a copy of the input data, as is so often the case. A reference parameter is an "alias" for the same bucket in memory as the input data. Thus any change made to a reference parameter **is in fact made to the original variable!**

Back to Topics List