**Functions in C**

A function is a group of statements that together perform a task. Every C program has at least one function, which is **main()**, and all the most trivial programs can define additional functions.

You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division is such that each function performs a specific task.

A function **declaration** tells the compiler about a function's name, return type, and parameters. A function **definition** provides the actual body of the function.

The C standard library provides numerous built-in functions that your program can call. For example, **strcat()** to concatenate two strings, **memcpy()** to copy one memory location to another location, and many more functions.

A function can also be referred as a method or a sub-routine or a procedure, etc.

Defining a Function

The general form of a function definition in C programming language is as follows −

return\_type function\_name( parameter list ) {

body of the function

}

A function definition in C programming consists of a *function header* and a *function body*. Here are all the parts of a function −

* **Return Type** − A function may return a value. The **return\_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return\_type is the keyword **void**.
* **Function Name** − This is the actual name of the function. The function name and the parameter list together constitute the function signature.
* **Parameters** − A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
* **Function Body** − The function body contains a collection of statements that define what the function does.

## Calling a Function:

## ===================

While creating a C function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task.

When a program calls a function, the program control is transferred to the called function. A called function performs a defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program.

To call a function, you simply need to pass the required parameters along with the function name, and if the function returns a value, then you can store the returned value.

## Function Arguments

If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the **formal parameters** of the function.

Formal parameters behave like other local variables inside the function and are created upon entry into the function and destroyed upon exit.

While calling a function, there are two ways in which arguments can be passed to a function −

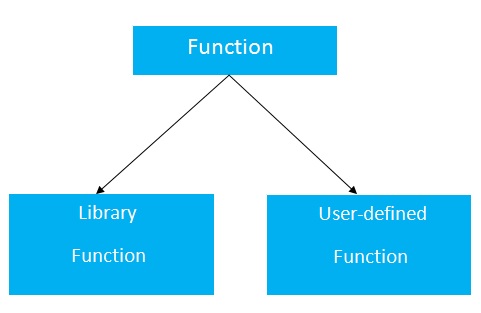
|  |  |
| --- | --- |
| **S.N.** | **Call Type & Description** |
| 1 | **[Call by value](https://www.tutorialspoint.com/cprogramming/c_function_call_by_value.htm)**  This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument. |
| 2 | **[Call by reference](https://www.tutorialspoint.com/cprogramming/c_function_call_by_reference.htm)**  This method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument. |

By default, C uses **call by value** to pass arguments. In general, it means the code within a function cannot alter the arguments used to call the function.

Types of Functions

There are two types of functions in C programming:

1. **Library Functions**: are the functions which are declared in the C header files such as scanf(), printf(), gets(), puts(), ceil(), floor() etc.
2. **User-defined functions**: are the functions which are created by the C programmer, so that he/she can use it many times. It reduces complexity of a big program and optimizes the code.



Return Value

A C function may or may not return a value from the function. If you don't have to return any value from the function, use void for the return type.

Let's see a simple example of C function that doesn't return any value from the function.

**Example without return value:**

1. **void** hello(){
2. printf("hello c");
3. }

If you want to return any value from the function, you need to use any data type such as int, long, char etc. The return type depends on the value to be returned from the function.

Let's see a simple example of C function that returns int value from the function.

**Example with return value:**

1. **int** get(){
2. **return** 10;
3. }

In the above example, we have to return 10 as a value, so the return type is int. If you want to return floating-point value (e.g. 10.2, 3.1, 54.5 etc), you need to use float as the return type of the method.

1. **float** get(){
2. **return** 10.2;
3. }

Now, you need to call the function, to get the value of the function.

Parameters in C Function

A c function may have 0 or more parameters. You can have any type of parameter in C program such as int, float, char etc. The parameters are also known as **formal arguments**.

**Example of a function that has 0 parameter:**

1. **void** hello(){
2. printf("hello c");
3. }

**Example of a function that has 1 parameter:**

1. **int** cube(**int** n){
2. **return** n\*n\*n;
3. }

**Example of a function that has 2 parameters:**

1. **int** add(**int** a, **int** b){
2. **return** a+b;
3. }

Calling a function in C

If a function returns any value, you need to call function to get the value returned from the function. The syntax of calling a function in c programming is given below:

1. variable=function\_name(arguments...);

**1) variable:** The variable is not mandatory. If function return type is *void*, you must not provide the variable because void functions doesn't return any value.

**2) function\_name:** The function\_name is name of the function to be called.

**3) arguments:** You need to provide arguments while calling the C function. It is also known as **actual arguments**.

**Example to call a function:**

1. hello();//calls function that doesn't return a value
2. **int** value=get();//calls function that returns value
3. **int** value2=add(10,20);//calls parameterized function by passing 2 values

#### Recursion

Recursion is a special of nesting functions, where a function calls itself inside it. We must have certain condition to break out of the recursion, otherwise recursion is infinite.

**function1()**

{

**function1() ;**

//statements

}