

Functions in C Programming

A function is a block of statements that performs a specific task. Suppose you are building an application in C language and in one of your program, you need to perform a same task more than once. In such case you have two options –

- a) Use the same set of statements every time you want to perform the task
- b) Create a function to perform that task, and just call it every time you need to perform that task.

Using option (b) is a good practice and a good programmer always uses functions while writing codes in C.

Types of functions

1) **Predefined standard library functions** – such as `puts()`, `gets()`, `printf()`, `scanf()` etc – These are the functions which already have a definition in header files (.h files like `stdio.h`), so we just call them whenever there is a need to use them.

2) **User Defined functions** – The functions that we create in a program are known as user defined functions.

In this guide, we will learn how to create user defined functions and how to use them in C Programming

Why we need functions in C

Functions are used because of following reasons –

- a) To improve the readability of code.
- b) Improves the reusability of the code, same function can be used in any program rather than writing the same code from scratch.
- c) Debugging of the code would be easier if you use functions, as errors are easy to be traced.
- d) Reduces the size of the code, duplicate set of statements are replaced by function calls.

Syntax of a function

```
return_type function_name (argument list)
{
    Set of statements – Block of code
}
```

return_type: Return type can be of any data type such as `int`, `double`, `char`, `void`, `short` etc. Don't worry you will understand these terms better once you go through the examples below.

function_name: It can be anything, however it is advised to have a meaningful name for the functions so that it would be easy to understand the purpose of function just by seeing it's name.

argument list: Argument list contains variables names along with their data types. These arguments are kind of inputs for the function. For example – A function which is used to add two integer variables, will be having two integer argument.

Block of code: Set of C statements, which will be executed whenever a call will be made to the function.

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.

Function Declarations

A function declaration tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.

A function declaration has the following parts –

```
return_type function_name( parameter list );
```

For the above defined function max(), the function declaration is as follows –

```
int max(int num1, int num2);
```

Parameter names are not important in function declaration only their type is required, so the following is also a valid declaration –

```
int max(int, int);
```

Function declaration is required when you define a function in one source file and you call that function in another file. In such case, you should declare the function at the top of the file calling the function.

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 –

Sr.No.	Call Type & Description
1	Call by value 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

	<p>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.</p>
--	---