

Functions in C Language

What is a Function?

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

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

Here option (b) is obviously a good practice.

Importance functions in C:

Functions are used because of following reasons –

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

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

How to use a function?

To use a function the following three steps we need to follow:

Function declaration/ prototype: A function prototype is simply the declaration of a function that specifies function's name, parameters and return type. It doesn't contain function body. A function prototype gives information to the compiler that the function may later be used in the program.

Syntax:

```
return_type function_name (data_type parameter...)  
{  
    //code to be executed  
}
```

Function call: Function can be called from anywhere in the program. The parameter list must be same in function calling and function declaration. We must pass the same number of parameters as it is declared in the function declaration.

Syntax: function_name (argument_list);

Function definition: It contains the actual statements which are to be executed. It is the most important aspect to which the control comes when the function is called. Here, we must notice that only one value can be returned from the function. It is also called function body part.

Syntax: return_type function_name (argument list)
{
 //function body;
}

Functions in C Language

Passing arguments to a function:

In programming, argument refers to the variable passed to the function. In the example below, two variables **n1** and **n2** are passed during the function call. The parameters, '**a**' and '**b**' accepts the arguments in the function definition. These arguments are called **formal parameters** of the function.

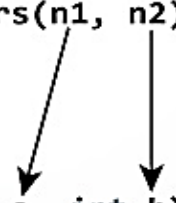
```
#include <stdio.h>

int addNumbers(int a, int b);

int main()
{
    ... .. ...

    sum = addNumbers(n1, n2);
    ... .. ...
}

int addNumbers(int a, int b)
{
    ... .. ...
    ... .. ...
}
```



'return' Statement:

The **return** statement terminates the execution of a function and returns a value to the calling function. The program control is transferred to the calling function after the return statement. In the example shown below, the value of the result variable is returned to the **main()** function. The sum variable in the **main()** function is assigned this value.

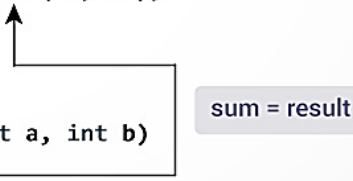
```
#include <stdio.h>

int addNumbers(int a, int b);

int main()
{
    ... .. ...

    sum = addNumbers(n1, n2);
    ... .. ...
}

int addNumbers(int a, int b)
{
    ... .. ...
    return result;
}
```



Functions in C Language

Actual Parameters and Formal Parameters:

Actual Parameters: The values/variables passed while calling a function are called actual parameters.

Formal Parameters: These are the variables declared in function definition/prototype, and receive their values when a call to that function is made.

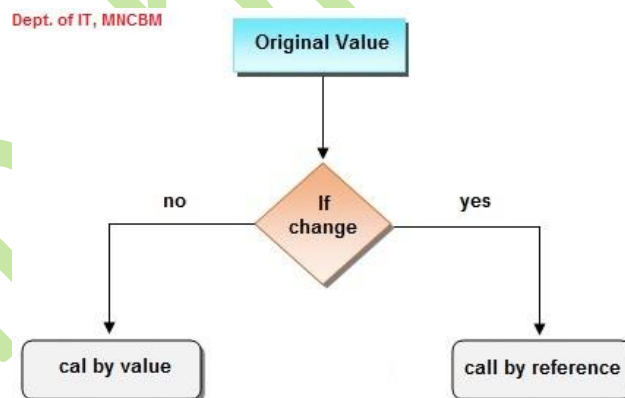
The value(s) of the **actual** parameters are copied to **formal** parameters when the call to that function is made. The following example shows it clearly.

```
#include<stdio.h>
int sum(int a, int b)           //Function definition, here a and b are formal parameters
{
    return a+b;
}
void main()
{
    int x=10,y=20;
    int s = sum(x,y);          //Function call, here x and y are actual parameters
}
```

Parameter passing techniques:

There are two techniques through which we can pass parameters to a function:

- Call by Value
- Call by Reference



Call by value: In call by value, a copy of actual arguments is passed to formal arguments of the called function and any change made to the formal arguments in the called function have no effect on the values of actual arguments in the calling function.

Call by reference: In call by reference, the location (address) of actual arguments is passed to formal arguments of the called function. This means by accessing the addresses of actual arguments we can alter them within from the called function.

Functions in C Language

Difference between call by value and call by reference:

call by value	call by reference
This method copy original value into function as an argument.	This method copy address of argument into function as an argument.
Changes made to the parameter inside the function have no effect on the argument.	Changes made to the parameter affect the argument. Because address is used to access the actual argument.
Actual and formal arguments will be created in different memory location	Actual and formal arguments will be created in same memory location

Example of call by value:

```
#include<stdio.h>
#include<conio.h>
void swap(int a, int b)
{
    int temp;
    temp=a;
    a=b;
    b=temp;
}
void main()
{
    int a=100, b=200;
    clrscr();
    swap(a, b);           // passing value to function
    printf("\nValue of a: %d",a);
    printf("\nValue of b: %d",b);
    getch();
}
```

Example of call by reference:

```
#include<stdio.h>
#include<conio.h>
void swap(int *a, int *b)
{
    int temp;
    temp=*a;
    *a=*b;
    *b=temp;
}
```

Functions in C Language

```
void main()
{
    int a=100, b=200;
    clrscr();
    swap(&a, &b);        // passing value to function
    printf("\nValue of a: %d",a);
    printf("\nValue of b: %d",b);
    getch();
}
```

mncbm.ac.in