# functions in C

The **function in C language** is also known as *procedure* or *subroutine* in other programming languages.

To perform any task, we can create function. A function can be called many times. It provides *modularity* and code *reusability*.

## **Advantage of functions in C**

There are many advantages of functions.

#### 1) Code Reusability

By creating functions in C, you can call it many times. So we don't need to write the same code again and again.

#### 2) Code optimization

It makes the code optimized, we don't need to write much code.

Suppose, you have to check 3 numbers (781, 883 and 531) whether it is prime number or not. Without using function, you need to write the prime number logic 3 times. So, there is repetition of code.

But if you use functions, you need to write the logic only once and you can reuse it several times.

## **Syntax to declare function in C**

The syntax of creating function in c language is given below:

1. return\_type function\_name(data\_type parameter...){
2. //code to be executed
3. }

## **Syntax to call function in C**

The syntax of calling function in c language 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 same number of arguments as defined in the function at the time of declaration or definition.

## **Example of function in C**

Let's see the simple program of function in c language.

1. #include <stdio.h>
2. #include <conio.h>
3. //defining function
4. **int** cube(**int** n){
5. **return** n\*n\*n;
6. }
7. **void** main(){
8. **int** result1=0,result2=0;
9. clrscr();
11. result1=cube(2);//calling function
12. result2=cube(3);
14. printf("%d \n",result1);
15. printf("%d \n",result2);
17. getch();
18. }

#### Output

8

27

# Call by value and call by reference in C

There are two ways to pass value or data to function in C language: *call by value* and *call by reference*. Original value is not modified in call by value but it is modified in call by reference.

Let's understand call by value and call by reference in c language one by one.

## **Call by value in C**

In call by value, **original value is not modified**.

In call by value, value being passed to the function is locally stored by the function parameter in stack memory location. If you change the value of function parameter, it is changed for the current function only. It will not change the value of variable inside the caller method such as main().

Let's try to understand the concept of call by value in c language by the example given below:

1. #include <stdio.h>
2. #include <conio.h>
3. **void** change(**int** num) {
4. printf("Before adding value inside function num=%d \n",num);
5. num=num+100;
6. printf("After adding value inside function num=%d \n", num);
7. }
9. **int** main() {
10. **int** x=100;
11. clrscr();
13. printf("Before function call x=%d \n", x);
14. change(x);//passing value in function
15. printf("After function call x=%d \n", x);
17. getch();
18. **return** 0;
19. }

#### Output

Before function call x=100

Before adding value inside function num=100

After adding value inside function num=200

After function call x=100

## **Call by reference in C**

In call by reference, **original value is modified** because we pass reference (address).

Here, address of the value is passed in the function, so actual and formal arguments shares the same address space. Hence, value changed inside the function, is reflected inside as well as outside the function.

*Note: To understand the call by reference, you must have the basic knowledge of pointers.*

Let's try to understand the concept of call by reference in c language by the example given below:

1. #include <stdio.h>
2. #include <conio.h>
3. **void** change(**int** \*num) {
4. printf("Before adding value inside function num=%d \n",\*num);
5. (\*num) += 100;
6. printf("After adding value inside function num=%d \n", \*num);
7. }
9. **int** main() {
10. **int** x=100;
11. clrscr();
13. printf("Before function call x=%d \n", x);
14. change(&x);//passing reference in function
15. printf("After function call x=%d \n", x);
17. getch();
18. **return** 0;
19. }

#### Output

Before function call x=100

Before adding value inside function num=100

After adding value inside function num=200

After function call x=200

## **Difference between call by value and call by reference in c**

|  |  |  |
| --- | --- | --- |
| **No.** | **Call by value** | **Call by reference** |
| 1 | A copy of value is passed to the function | An address of value is passed to the function |
| 2 | Changes made inside the function is not reflected on other functions | Changes made inside the function is reflected outside the function also |
| 3 | Actual and formal arguments will be created in different memory location | Actual and formal arguments will be created in same memory location |

# Recursion in C

When *function is called within the same function*, it is known as **recursion** in C. The function which calls the same function, is known as **recursive function**.

A function that calls itself, and doesn't perform any task after function call, is know as **tail recursion**. In tail recursion, we generally call the same function with return statement. An example of tail recursion is given below.

Let's see a simple example of recursion.

1. recursionfunction(){
3. recursionfunction();//calling self function
5. }

## **Example of tail recursion in C**

Let's see an example to print factorial number using tail recursion in C language.

1. #include<stdio.h>
2. #include<conio.h>
3. **int** factorial (**int** n)
4. {
5. **if** ( n < 0)
6. **return** -1; /\*Wrong value\*/
7. **if** (n == 0)
8. **return** 1; /\*Terminating condition\*/
9. **return** (n \* factorial (n -1));
10. }
12. **void** main(){
13. **int** fact=0;
14. clrscr();
15. fact=factorial(5);
16. printf("\n factorial of 5 is %d",fact);
18. getch();
19. }

#### Output

factorial of 5 is 120

We can understand the above program of recursive method call by the figure given below: