C Functions

In c, we can divide a large program into the basic building blocks known as function. The function contains the set of programming statements enclosed by {}. A function can be called multiple times to provide reusability and modularity to the C program.

## Advantage of functions in C

* By using functions, we can avoid rewriting same logic/code again and again in a program.
* We can call C functions any number of times in a program and from any place in a program.
* We can track a large C program easily when it is divided into multiple functions.
* Reusability is the main achievement of C functions.
* However, Function calling is always a overhead in a C program.

## Function Aspects

* **Function declaration** A function must be declared globally in a c program to tell the compiler about the function name, function parameters, and return type.

**i.e : return\_type function\_name (argument list);**

* **Function call** Function can be called from anywhere in the program. The parameter list must not differ in function calling and function declaration. We must pass the same number of functions as it is declared in the function declaration.

**i.e : 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.

**i.e : return\_type function\_name (argument list) {function body;}**

## 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 the 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.

## Different aspects of function calling

A function may or may not accept any argument. It may or may not return any value. Based on these facts, There are four different aspects of function calls.

* function without arguments and without return value
* function without arguments and with return value
* function with arguments and without return value
* function with arguments and with return value

### Example for Function without argument and without return value

1. #include<stdio.h>
2. **void** printName();
3. **void** main ()
4. {
5. printf("Hello ");
6. printName();
7. }
8. **void** printName()
9. {
10. printf("SOU");
11. }

------------------------------------------------------------------------------------------

1. #include<stdio.h>
2. **void** sum();
3. **void** main()
4. {
5. printf("\nGoing to calculate the sum of two numbers:");
6. sum();
7. }
8. **void** sum()
9. {
10. **int** a,b;
11. printf("\nEnter two numbers");
12. scanf("%d %d",&a,&b);
13. printf("The sum is %d",a+b);
14. }

**Enter two numbers 10**

**24**

**The sum is 34**

### Example for Function without argument and with return value

1. #include<stdio.h>
2. **int** sum();
3. **void** main()
4. {
5. **int** result;
6. printf("\nGoing to calculate the sum of two numbers:");
7. result = sum();
8. printf("%d",result);
9. }
10. **int** sum()
11. {
12. **int** a,b;
13. printf("\nEnter two numbers");
14. scanf("%d %d",&a,&b);
15. **return** a+b;
16. }

Enter two numbers 10

24

The sum is 34

--------------------------------------------------------------------------------------------------------

1. #include<stdio.h>
2. **int** sum();
3. **void** main()
4. {
5. printf("Going to calculate the area of the square\n");
6. **float** area = square();
7. printf("The area of the square: %f\n",area);
8. }
9. **int** square()
10. {
11. **float** side;
12. printf("Enter the length of the side in meters: ");
13. scanf("%f",&side);
14. **return** side \* side;
15. }

Going to calculate the area of the square

Enter the length of the side in meters: 10

The area of the square: 100.000000

### Example for Function with argument and without return value

1. #include<stdio.h>
2. **void** sum(**int**, **int**);
3. **void** main()
4. {
5. **int** a,b,result;
6. printf("\nGoing to calculate the sum of two numbers:");
7. printf("\nEnter two numbers:");
8. scanf("%d %d",&a,&b);
9. sum(a,b);
10. }
11. **void** sum(**int** a, **int** b)
12. {
13. printf("\nThe sum is %d",a+b);
14. }

Going to calculate the sum of two numbers:

Enter two numbers 10 24

The sum is 34

--------------------------------------------------------------------------------------------------------

1. #include<stdio.h>
2. **void** average(**int**, **int**, **int**, **int**, **int**);
3. **void** main()
4. {
5. **int** a,b,c,d,e;
6. printf("\nGoing to calculate the average of five numbers:");
7. printf("\nEnter five numbers:");
8. scanf("%d %d %d %d %d",&a,&b,&c,&d,&e);
9. average(a,b,c,d,e);
10. }
11. **void** average(**int** a, **int** b, **int** c, **int** d, **int** e)
12. {
13. **float** avg;
14. avg = (a+b+c+d+e)/5;
15. printf("The average of given five numbers : %f",avg);
16. }

Going to calculate the average of five numbers:

Enter five numbers:10 20 30 40 50

The average of given five numbers: 30.000000

### Example for Function with argument and with return value

1. #include<stdio.h>
2. **int** sum(**int**, **int**);
3. **void** main()
4. {
5. **int** a,b,result;
6. printf("\nGoing to calculate the sum of two numbers:");
7. printf("\nEnter two numbers:");
8. scanf("%d %d",&a,&b);
9. result = sum(a,b);
10. printf("\nThe sum is : %d",result);
11. }
12. **int** sum(**int** a, **int** b)
13. {
14. **return** a+b;
15. }

Enter two numbers:10 20

The sum is : 30

--------------------------------------------------------------------------------------------------------

1. #include<stdio.h>
2. **int** even\_odd(**int**);
3. **void** main()
4. {
5. **int** n,flag=0;
6. printf("\nGoing to check whether a number is even or odd");
7. printf("\nEnter the number: ");
8. scanf("%d",&n);
9. flag = even\_odd(n);
10. **if**(flag == 0)
11. {
12. printf("\nThe number is odd");
13. }
14. **else**
15. {
16. printf("\nThe number is even");
17. }
18. }
19. **int** even\_odd(**int** n)
20. {
21. **if**(n%2 == 0)
22. {
23. **return** 1;
24. }
25. **else**
26. {
27. **return** 0;
28. }
29. }

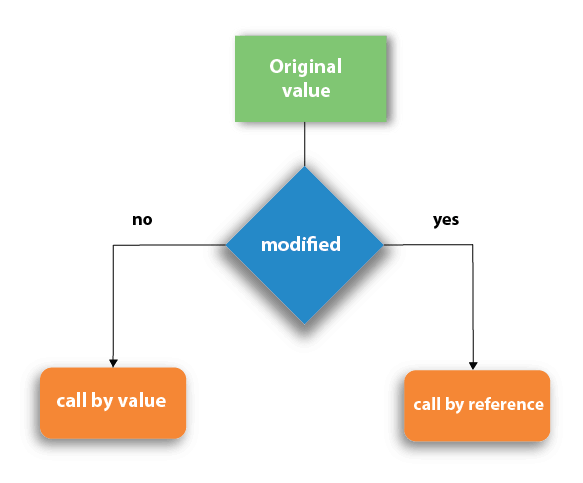
Enter the number: 100

The number is even

--------------------------------------------------------------------------------------------------------

Call by value and Call by reference in C

There are two methods to pass the data into the function in C language, i.e., *call by value* and *call by reference*.



## Call by value in C

* In call by value method, the value of the actual parameters is copied into the formal parameters. In other words, we can say that the value of the variable is used in the function call in the call by value method.
* In call by value method, we can not modify the value of the actual parameter by the formal parameter.
* In call by value, different memory is allocated for actual and formal parameters since the value of the actual parameter is copied into the formal parameter.
* The actual parameter is the argument which is used in the function call whereas formal parameter is the argument which is used in the function definition.

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

#### 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, the address of the variable is passed into the function call as the actual parameter.
* The value of the actual parameters can be modified by changing the formal parameters since the address of the actual parameters is passed.
* In call by reference, the memory allocation is similar for both formal parameters and actual parameters. All the operations in the function are performed on the value stored at the address of the actual parameters, and the modified value gets stored at the same address.

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

#### 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 the value is passed into the function | An address of value is passed into the function |
| 2 | Changes made inside the function is limited to the function only. The values of the actual parameters do not change by changing the formal parameters. | Changes made inside the function validate outside of the function also. The values of the actual parameters do change by changing the formal parameters. |
| 3 | Actual and formal arguments are created at the different memory location | Actual and formal arguments are created at the same memory location |