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. In other words, we can say that the collection of functions creates a program. The function is also known as *procedure*or *subroutine* in other programming languages.

Advantage of functions in C

There are the following advantages of C functions.

- 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 an overhead in a C program.

Function Aspects

There are three aspects of a C function.

- **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.
- **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.
- **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.

SN	C function aspects	Syntax
1	Function declaration	return_type function_name (argument list);
2	Function call	function_name (argument_list);
3	Function definition	return_type function_name (argument list) {function body;}

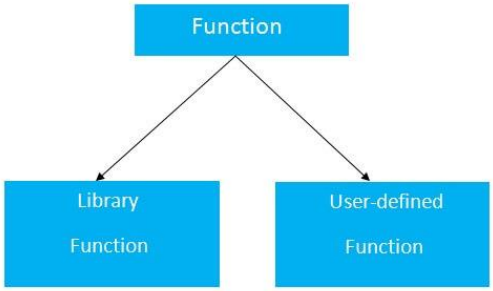
The syntax of creating function in c language is given below:

```
1. return_type function_name(data_type parameter...){
2. //code to be executed
3. }
```

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.

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.

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 return value

Example 1

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

Output

```
Hello VSSUT
```

Example 2

```
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()
```

```
{
    int a,b;
    printf("\nEnter two numbers");
    scanf("%d %d",&a,&b);
    printf("The sum is %d",a+b);
}
```

Output

```
Going to calculate the sum of two numbers:
Enter two numbers 10
24
The sum is 34
```

Example for Function without argument and with return value

Example 1

```
#include<stdio.h>
int sum();
void main()
{
    int result;
    printf("\nGoing to calculate the sum of two numbers:");
    result = sum();
    printf("%d",result);
}
int sum()
{
    int a,b;
    printf("\nEnter two numbers");
    scanf("%d %d",&a,&b);
    return a+b;
}
```

Output

```
Going to calculate the sum of two numbers:

Enter two numbers 10
24

The sum is 34
```

Example 2: program to calculate the area of the square

```
#include<stdio.h>
```

```
int sum();
void main()
{
    printf("Going to calculate the area of the square\n");
    int area = square();
    printf("The area of the square: %f\n",area);
}
int square()
{
    int side;
    printf("Enter the length of the side in meters: ");
    scanf("%d",&side);
    return side * side;
}
```

Output

```
Going to calculate the area of the square
Enter the length of the side in meters: 10
The area of the square: 100
```

Example for Function with argument and without return value

Example 1

```
#include<stdio.h>
void sum(int, int);
void main()
{
    int a,b,result;
    printf("\nGoing to calculate the sum of two numbers:");
    printf("\nEnter two numbers:");
    scanf("%d %d",&a,&b);
    sum(a,b);
}
void sum(int a, int b)
{
    printf("\nThe sum is %d",a+b);
}
```

Output

```
Going to calculate the sum of two numbers:

Enter two numbers 10
24

The sum is 34
```

Example 2: program to calculate the average of five numbers.

```
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. }
```

Output

```
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

Example 1

```
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. }
```

Output

```
Going to calculate the sum of two numbers:
Enter two numbers:10
20
The sum is : 30
```

Example 2: Program to check whether a number is even or odd

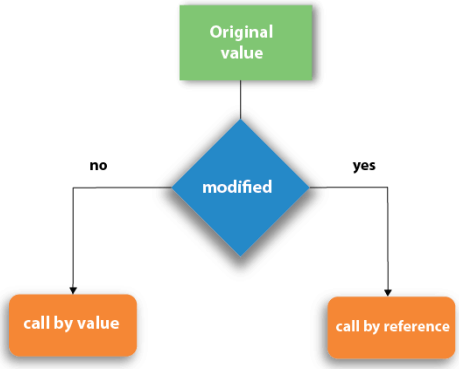
```
#include<stdio.h>
int even_odd(int);
void main()
{
 int n,flag=0;
 printf("\nGoing to check whether a number is even or odd");
 printf("\nEnter the number: ");
 scanf("%d",&n);
 flag = even_odd(n);
if(flag == 0)
{
   printf("\nThe number is odd");
}
else
{
   printf("\nThe number is even");
}
}
int even_odd(int n)
{
   if(n%2 == 0)
   {
      return 1;
   }
   else
   {
      return 0;
   }
}
```

Output

```
Going to check whether a number is even or odd
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*.



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 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.

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

```
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 Value Example: Swapping the values of the two variables

```
1. #include <stdio.h>
2. void swap(int , int); //prototype of the function
3. int main()
4. {
5.    int a = 10;
6.    int b = 20;
7.    printf("Before swapping the values in main a = %d, b = %d\n",a,b); // printing the value of a an
   d b in main
8.    swap(a,b);
9.    printf("After swapping values in main a = %d, b = %d\n",a,b); // The value of actual parameter
   s do not change by changing the formal parameters in call by value, a = 10, b = 20
10. }
11. void swap (int a, int b)
12. {
13.    int temp;
14.    temp = a;
15.    a=b;
16.    b=temp;
17.    printf("After swapping values in function a = %d, b = %d\n",a,b); // Formal parameters, a = 20,
    b = 10
18. }
```

Output

```
Before swapping the values in main a = 10, b = 20
After swapping values in function a = 20, b = 10
After swapping values in main a = 10, b = 20
```

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.

Consider the following example for the call by reference.

```
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
```

Call by reference Example: Swapping the values of the two variables

```
1. #include <stdio.h>
2. void swap(int *, int *); //prototype of the function
3. int main()
4. {
5.     int a = 10;
6.     int b = 20;
7.     printf("Before swapping the values in main a = %d, b = %d\n",a,b); // printing the value of a an
   d b in main
8.     swap(&a,&b);
9.     printf("After swapping values in main a = %d, b = %d\n",a,b); // The values of actual paramete
   rs do change in call by reference, a = 10, b = 20
10. }
11. void swap (int *a, int *b)
12. {
13.    int temp;
14.    temp = *a;
15.    *a=*b;
16.    *b=temp;
17.    printf("After swapping values in function a = %d, b = %d\n",*a,*b); // Formal parameters, a =
   20, b = 10
18. }
```

Output

```
Before swapping the values in main a = 10, b = 20
After swapping values in function a = 20, b = 10
After swapping values in main a = 20, b = 10
```

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