Unit-1 Function

1.1 Definition and Need of Function –

In programming, set of statements that solves a particular task is called function or module. Every C program should have at least one function that is main function. The function contains the set of programming statements enclosed by {}.You can divide up your code into separate functions. A function **declaration** tells the compiler about a function's name, return type, and parameters. A function **definition** provides the actual body of the function. 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.

Why do we need functions?

- Enables reusability and reduces redundancy
- Makes a code modular
- Provides abstraction functionality
- The program becomes easy to understand and manage
- Breaks an extensive program into smaller and simpler pieces

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 a overhead in a C program.

1.2 Declaration and Prototypes-

- **Function aspect**

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;}

Basic Syntax of Functions -

The basic syntax of functions in C programming is:

return_type function_name(arg1, arg2, … argn){

Body of the function //Statements to be processed

}

In the above syntax:

- return_type: Here, we declare the data type of the value returned by functions. However, not all functions return a value. In such cases, the

keyword void indicates to the compiler that the function will not return any value.

- function_name: This is the function's name that helps the compiler identify it whenever we call it.
- arg1, arg2, ...argn: It is the argument or parameter list that contains all the parameters to be passed into the function. The list defines the data type, sequence, and the number of parameters to be passed to the function. A function may or may not require parameters. Hence, the parameter list is optional.
- Body: The function's body contains all the statements to be processed and executed whenever the function is called.

• 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.

For example, `main()` is a function, which is used to execute code, and `printf()` is a function; used to output/print text to the screen:

Example

```
int main() {
 printf("Hello World!");
 return 0;
}
```

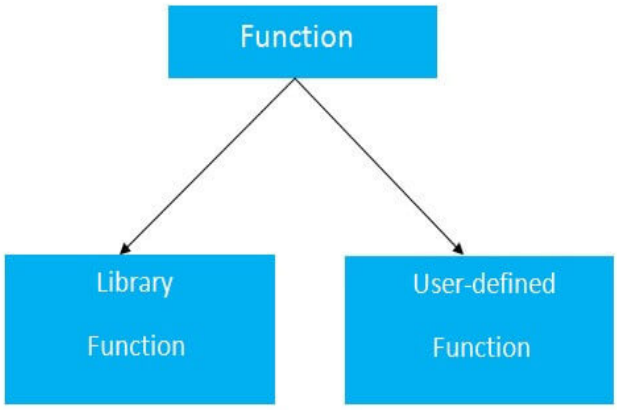
2. **User-defined functions**: are the functions which are created by the C programmer, so that he/she can use it many times. It reduces the complexity of a big program and optimizes the code.

   Ex.

```
// Create a function
void myFunction() {
  printf("I just got executed!");
}

int main() {
```

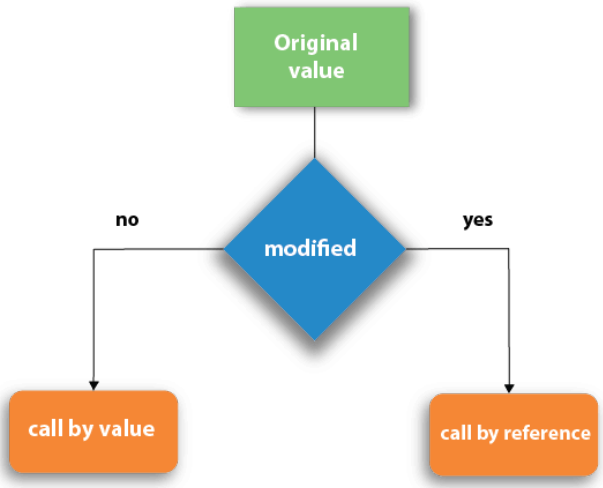
```
  myFunction(); // call the function
  return 0;
}
```



1.3Function calling (Call by value, call by reference)

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.

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

```
#include<stdio.h>

void change(int num) {

   printf("Before adding value inside function num=%d \n",num);

   num=num+100;

   printf("After adding value inside function num=%d \n", num);
```

```
}

int main() {

   int x=100;

   printf("Before function call x=%d \n", x);

   change(x);//passing value in function

   printf("After function call x=%d \n", x);

return 0;

}
```

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

```
#include <stdio.h>
void swap(int , int); //prototype of the function
int main()
{
   int a = 10;
   int b = 20;
   printf("Before swapping the values in main a = %d, b = %d\n",a,b); // printing th
e value of a and b in main

   swap(a,b);

   printf("After swapping values in main a = %d, b = %d\n",a,b);
 // The value of actual parameters do not change by changing the formal parameter
s in call by value, a = 10, b = 20
}
```

```
void swap (int a, int b)
{
   int temp;
   temp = a;
   a=b;
   b=temp;
   printf("After swapping values in function a = %d, b = %d\n",a,b); // Formal para
meters, a = 20, b = 10
}
```

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.

Let us now call the function **swap()** by passing values by reference as in the following example −

```
#include <stdio.h>

int main () {

  /* local variable definition */
  int a = 100;
  int b = 200;

  printf("Before swap, value of a : %d\n", a );
  printf("Before swap, value of b : %d\n", b );
```

```
   /* calling a function to swap the values */
   swap(&a, &b);

   printf("After swap, value of a : %d\n", a );
   printf("After swap, value of b : %d\n", b );

   return 0;
}
void swap(int *x, int *y) {

   int temp;

   temp = *x; /* save the value of x */
   *x = *y;    /* put y into x */
   *y = temp; /* put temp into y */

   return;
}
```

Let us put the above code in a single C file, compile and execute it, to produce the following result −

```
Before swap, value of a : 100
Before swap, value of b : 200
After swap, value of a : 200
After swap, value of b : 100
```

1.4 Function with return and Function with argument –

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. }
```

Example for Function without argument and return value

Example 1

```
#include<stdio.h>
void printName();
void main ()
{
   printf("Hello ");
   printName();
}
void printName()
{
   printf("Javatpoint");
}
```

Output

```
Hello Javatpoint
```

Example 2

```
#include<stdio.h>
void sum();
void main()
{
   printf("\nGoing to calculate the sum of two numbers:");
   sum();
}
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");
   float area = square();
   printf("The area of the square: %f\n",area);
}
int square()
{
   float side;
   printf("Enter the length of the side in meters: ");
   scanf("%f",&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: 100.000000
```

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.

```
#include<stdio.h>

void average(int, int, int, int, int);

void main()

{

   int a,b,c,d,e;

   printf("\nGoing to calculate the average of five numbers:");

   printf("\nEnter five numbers:");

   scanf("%d %d %d %d %d",&a,&b,&c,&d,&e);

   average(a,b,c,d,e);

}

void average(int a, int b, int c, int d, int e)

{

   float avg;

   avg = (a+b+c+d+e)/5;

   printf("The average of given five numbers : %f",avg);

}
```

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

```
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. }
```

Output

```
Going to check whether a number is even or odd
Enter the number: 100
The number is even
```

Program to check number is positive and negative with function

Example

```
#include<stdio.h>

void check(int num)
{
    if(num == 0)
        printf("Neither positive nor negative\n");
    else if(num > 0)
        printf("Positive\n");
    else
        printf("Negative\n");
}

int main()
{
    int num;

    //for first number
    scanf("%d",&num);
    check(num);

    //for second number
    scanf("%d",&num);
    check(num);

    return 0;
```

```
}
```

C Library Functions

Library functions are the inbuilt function in C that are grouped and placed at a common place called the library. Such functions are used to perform some specific operations. For example, printf is a library function used to print on the console. The library functions are created by the designers of compilers. All C standard library functions are defined inside the different header files saved with the extension **.h**. We need to include these header files in our program to make use of the library functions defined in such header files. For example, To use the library functions such as printf/scanf we need to include stdio.h in our program which is a header file that contains all the library functions regarding standard input/output.

The list of mostly used header files is given in the following table.

SN	Header file	Description
1	stdio.h	This is a standard input/output header file. It contains all the library functions regarding standard input/output.
2	conio.h	This is a console input/output header file.
3	string.h	It contains all string related library functions like gets(), puts(),etc.
4	stdlib.h	This header file contains all the general library functions like malloc(), calloc(), exit(), etc.
5	math.h	This header file contains all the math operations related functions like sqrt(), pow(), etc.
6	time.h	This header file contains all the time-related functions.
7	ctype.h	This header file contains all character handling functions.
8	stdarg.h	Variable argument functions are defined in this header file.
9	signal.h	All the signal handling functions are defined in this header file.
10	setjmp.h	This file contains all the jump functions.
11	locale.h	This file contains locale functions.
12	errno.h	This file contains error handling functions.
13	assert.h	This file contains diagnostics functions.