Function

A function is a group of statements that together perform a task. Every C program has at least one function, which is **main()**, and all the most trivial programs can define additional functions. A function is a certain line of block of code that performs a specific task. The functions which are created by programmer are called user-defined functions. The functions which are implemented in header libraries are known as library functions. Function which is not returning any value from function, their return type is void.

While using function, three things are important

Function Declarations

A function **declaration** tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately. Like variables, all the functions must be declared before they are used. The function declaration is also known as function prototype or function signature. It consists of four parts,

1) Function type (return type).
2) Function name.
3) Parameter list. Or argument list of a function
4) Terminating semicolon

A function declaration has the following parts:

Syntax**:** **Return_type FunctionName (Argument1, Argument2, Argument3……);**

Example: int sum(int , int);

In this example, function return type is int, name of function is sum, 2 parameters are passed to function and both are integer.

Why we need functions

Functions are used because of following reasons –

a) To improve the readability of code.
b) Improves the reusability of the code, same function can be used in any program rather than writing the same code from scratch.
c) Debugging of the code would be easier if you use functions as errors are easy to be traced.
d) Reduces the size of the code, duplicate set of statements are replaced by function calls.

Function Definition

Function Definition is also called function implementation. It has mainly two parts.

i. Function header : It is same as function declaration but with argument name.

ii. Function body : It is actual logic or coding of the function

The general form of a function definition in C programming language is as follows:

return_type function_name(argument list1, argument list 2,……)

{

body of the function // logic of a function that we want to perform

}

Example:

```
int max(int num1, int num2)
{
    /* local variable declaration */
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
```

Function definition in C programming consists of a *function header* and a *function body*. Here are all the parts of a function:

- **Return Type**: A function may return a value. The **return_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword **void**.
- **Function Name:** This is the actual name of the function. The function name and the parameter list together constitute the function signature.
- **Parameters:** A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
- Function Body: The function body contains a collection of statements that define what the function does.

1. Explain different categories of functions in details.

Functions can be classified in one of the following category based on whether arguments are present or not, whether a function is returning any value or not.

So function is categorized in to following categories.

I. No argument , not return value
II. Has argument , not return value

III. No argument , has return value
IV. Has argument and has return value

We will see each and every category in brief with example.

I. No argument ,not return value

In this type of category a function does not have any argument, it means its parameter list is empty, and functions will not return any value so return type of such categories function is **void.**

In this type of function will not return any value to calling function or not receive any value from calling function.

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

void  sum()
{
        int no1,no2,ans;
        printf("enter no1,no2:");
        scanf("%d %d",&no1,&no2);
        ans = no1+no2;
        printf("sum of two nos. =%d", ans);
}
void main()
{
        clrscr();
        sum();
        getch();
}
```

II. Has Argument not Return Value

In this type of category a function have any number argument, it means its parameter list is not empty, and functions will not return any value so return type of such categories function is **void.**

In this type of function the function will receive the data from the main function (calling function) but not return any value to calling function.

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

void  sum(int no1,int no2)
{
        int ans;
        ans = no1+no2;
        printf("sum of two nos. =%d", ans);
}
```

```
void main()
{
        int no1,no2;
        clrscr();
        printf("enter no1,no2:");
        scanf("%d%d",&no1,&no2);
        sum(no1,no2);
        getch();
}
```

III. No argument and has return value

In this type of category a function does not have any argument, it means its parameter list is empty, and functions will return any value so return type of such categories function is not **void** it can be any other valid data type**.**

In this type of function the function will not receive the data from the main function (calling function) but will return any value to calling function.

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

int  sum()
{
        int no1,no2,ans;
        printf("enter no1,no2:");
        scanf("%d %d",&no1,&no2);
        ans = no1+no2;
        return ans;
}
void main()
{
        int res;
        clrscr();
        res=sum();
        printf("sum of two nos is %d" ,res);
        getch();
}
```

IV. Has Argument and Has Return Value

In this type of category a function does have any number of argument, it means its parameter list is not empty, and functions will return any value so return type of such categories function is not **void** it can be any other valid data type**.**

In this type of function the function will receive the data from the main function (calling function) and also will return any value to calling function.

```
#include<stdio.h>
#inlcude<conio.h>
int  sum( int no1, int no2)
{
```

```
        int ans;
        ans = no1+no2;
        return ans;
}
void main()
{
        int res,no1,no2;
        clrscr();
        printf("enter no1,no2:");
        scanf("%d %d",&no1,&no2);
        res=sum(no1,no2);
        printf("sum of two nos is %d" ,res);
        getch();
}
```

2. **Explain actual argument and formal argument with example.**
   - Arguments passed to the function during function calling are called actual arguments or actual parameters.
   - Arguments received in the definition of a function are called formal arguments or formal parameters.

```
#include<stdio.h>
#inlcude<conio.h>
int  sum( int no1, int no2)                    // no1 and no2 here treated as formal argument.
{
        int ans;
        ans = no1+no2;
        return ans;
}
void main()
{
        int res,no1,no2;
        clrscr();
        printf("enter no1,no2:");
        scanf("%d %d",&no1,&no2);
        res=sum(no1,no2);                      // no1 and no2 here treated as actual argument
        printf("sum of two nos is %d" ,res);
        getch();
}
```

3. **Explain Call By value and call by reference with Example**
   OR **How Parameters are passed to the function Explain with Example?**

If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the **formal parameters** of the function. We can pass parameters to a function by any of the two ways:

I. Call by Value

II. Call by Reference

I. Call by Value

The **call by value** method of passing arguments to a function copies the actual parameter's value of an argument into the formal parameter of the function. In this case, changes made to the formal parameter inside the function have no effect on the actual parameter argument.

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

/* function definition to swap the values */
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 */
}
void 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 );
        swap(a, b);
        printf("After swap, value of a : %d\n", a );
        printf("After swap, value of b : %d\n", b );
        getch();
}
```

II. Call by Reference

The **call by reference** method of passing arguments to a function copies the address of an actual argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. It means the changes made to the formal parameter affect the passed actual argument.

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

/* function definition to swap the values */
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 */
}
void 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 );
        swap(&a, &b);
        printf("After swap, value of a : %d\n", a );
        printf("After swap, value of b : %d\n", b );
        getch();
}
```

4. **What is recursion? Explain With Example.**

Recursion is a process in which a function called itself. So when a function called itself is called recursion. So each recursive function should have terminating condition otherwise recursive function will go to infinite loop.

Recursive functions are very useful to solve many mathematical problems, such as calculating the factorial of a number, generating Fibonacci series, etc.

```
#include<stdio.h>
#include<conio.h>
long fact(long);
void main()
{
        long f,n;
        printf("enter  number:");
        scanf("%ld",&n);
        f=fact(n);
        printf("\n factorial=%ld",f);
}
long fact(long n)
{
        int f=1;
        if(n==1)
```

```
        {
                return 1;
        }
        else
        {
                return (n*fact(n-1));
        }
}
```

Advantages

- Easy solution for recursively defined solution.
- Complex programs can be easily written with less code.

Disadvantages

- Recursive code is difficult to understand and debug.
- Terminating condition is must; otherwise it will go in an infinite loop.
- Execution speed decreases because of function call and return activity many times.

4 What is scope, lifetime and visibility of variable?

A scope in any programming is a region of the program where a defined variable can have its existence and beyond that variable it cannot be accessed. The scope of variable can be defined as that a part of a program where the particular variable is accessible.

A variable is divided in to mainly 4 categories.

I. Local Variable or Automatic Variable

Variables that are declared inside a function or block are called local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own.

They are stored in a memory block of computer. **Initial value of a local variable is garbage value.**

They are created when a function is called and destroyed when function is terminated so they are called **Automatic variable**.

II. Global Variable:

Variables that are declared outside a function or block are called global variables. We will use key word extern to declare a global variable. **Initial value of a global variable is 0.** They are accessible anywhere in program and life time of a global variable is entire program.

III. Register Variable

Register variable are those variable which are stored in side computer's register for faster access. This variable having capacity to execute faster than global variable and local variable.

IV. Static variable

Static variable are the variable whose value remains static in program.

These variables have initial value 0 by default. This variable is called local static or global static depending upon their declaration. This variable remains its value or in existence after a function call.