

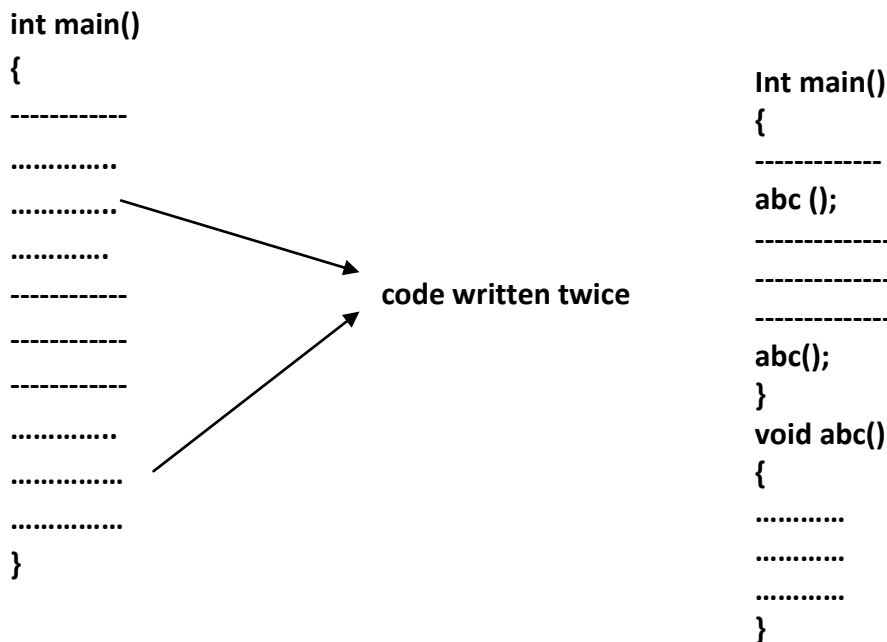
# Unit- 7 Functions

## Introduction:

Function is a block of code or statement to perform some specific (particular) task. Every c program has at least one function, which is main() and all other trivial program can define additional function.

We can divide up our code into separate function, but logically the division in such way that each function performs a specific task.

A single function can be accessed or called from different places with in a program. After execution of the intended task, the control will be returned to the point from which the function was accessed or called.



## Advantage of using functions:

- Functions increase code reusability by avoiding rewriting of the same code over and over.
- If a program is divided into multiple functions then each function can be independently developed. So, program development will be easier
- The program debugging will be easier.
- Length of the program can be reduced by using the function at appropriate place.
- Function reduce program complexity
- Program development will be faster
- With function, it will be easier to understand the logic involved in the program.
- Recursive call is possible through function (A function is allowed to call itself. A function which calls itself directly or indirectly again and again until specified condition is satisfied is known as recursive function.)
- Function increases program readability (clarity)

## Types of Functions:

There are two types of function in C programming:

- Library / Built-in functions.
- User-defined functions.

### 1) Library / Built-in function:

The library function is a predefined function in a header file or pre-processor directive. The programmer can use this function by adding a header file to the program. They cannot change or alter the meaning of this function. They don't have to declare and define these functions.

**For example:**

**printf(), scanf()** are function defined in header file **stdio.h**

**getch(), clrscr()** are defined in header file **conio.h**

**strlen(), strcat(), strrev()** are defined in header file **string .h**

### 2) User-defined function:-

A user defined function is declared, defined, and used by the programmer according to their need.

**Difference between Library and user defined function:**

Library function	User-defined function
1) The library function is predefined function in a header file pre-processor directive	1) A user-defined function is not a predefined function, it is defined by the programmer according to the need.
2) The programmer can simply use this function by including the respective header file.	2) The programmer has to declare define and use this function by them itself.
3) Program development time will be faster.	3) Program development time will be usually slower
4) The program using the library function will be usually simple	4) The program using a user-defined function will be usually complex.
5) This function requires a header file to use it.	5) This function requires a function prototype to use it.
6) Example:- printf(), scanf(), getch(), strlen() etc.	6) Example:- Sum(), area(), factorial() etc.

**Component of user defined function:**

- Function declaration / function prototype
- Function call
- Function definition

### 1) Function declaration / function prototype:

Function prototype tells the compiler about the name of function, the type of data returned by the function, the number type and order of parameter.

**Syntax:**

returntype functionname(datatype variable1, datatype variable2,.....);

**Eg:**

**void add(int a, int b);**

**int factorial(int n);**

**int add(int a, int b);**

This tells compiler that:-

- Name of function is add
- Function takes two arguments each of integer type.
- This function returns value whose type is integer.

Note: The return type "void" tells the compiler that it does not return any value.

## 2) Function call:

Once function definition is ready, it must be called inside boundary of main function. The function never comes in action if it is not called.

During function call, function name must be matched with function prototype name and the number and order of argument.

### Syntax:

```
functionname(argument1,argument2);
```

## 3) Function definition:-

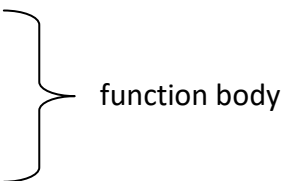
The collection of program statement that describes the specific task to be done by the function is called function definition.

Function definition consists of function header which defines function name returns type and its arguments and consist of function body which is enclosed in curly braces.

### Syntax:-

```
returntype functionname(datatype variable1, datatype variable2, .....)
```

```
{
-----
-----
-----
}
```



### Eg:

```
void add(int a, int b)
{
    int sum;
    sum=a+b;
    printf("sum=%d",sum);
}
```

## Example of c program to add two number using function

```
#include<stdio.h>
void add(int a,int b);    //function prototype or declaration
int main()
{
    int x,y;
    printf("Enter two number");
    scanf("%d %d",&x,&y);
    add(x,y);    //function call
    return 0;
}
void add(int a,int b)    // function definition
{
    int sum;
```

```

        sum=a+b;
        printf("sum=%d",sum);
    }

```

### Types of user-defined function:

Depending on the presence or absence of arguments and whether the value is returned or not, the function can be categorized as:-

- Function with no return type and no argument  
void addition();
- Function with no return type but argument  
void addition(int a, int b);
- Function with return type and argument  
int addition(int a, int b);
- Function with return but no argument  
int addition();

#### 1) Function with no return type and no argument:

Function with no argument means the called function does not receive any data from calling function and Function with no return value means calling function does not receive any data from the called function. So there is no data transfer between calling and called function.

Eg:

```

#include<stdio.h>
void add();
int main()
{
    add();
    return 0;
}
void add()
{
    int a,b,c;
    printf("Enter two number");
    scanf("%d %d",&a,&b);
    c=a+b;
    printf("sum=%d",c);
}

```

#### 2) Function with no return type but argument:

Here function will accept data from the calling function as there are arguments, however, since there is no return type nothing will be returned to the calling program. So it's a one-way type communication.

Eg:

```

#include<stdio.h>
void add(int a,int b);

```

```

int main()
{
    int x,y;
    printf("Enter two number");
    scanf("%d %d",&x,&y);
    add(x,y);
    return 0;
}
void add(int a,int b)
{
    int sum;
    sum=a+b;
    printf("sum=%d",sum);
}

```

### 3) Function with return type and argument:

Function with arguments and return value means both the calling function and called function will receive data from each other. It's like a dual communication.

**Eg:**

```

#include<stdio.h>
int add(int a,int b);
int main()
{
    int x,y,s;
    printf("Enter two number");
    scanf("%d %d",&x,&y);
    s=add(x,y);
    printf("sum=%d",s);
    return 0;
}
int add(int a,int b)
{
    int sum;
    sum=a+b;
    return sum;
}

```

### 4) Function with return type but no argument:

As said earlier function with no arguments means called function does not receive any data from calling function and function with return value means one result will be sent back to the caller from the function.

**Eg:**

```

#include<stdio.h>
int add();
int main()

```

```

{
    int s;
    s=add();
    printf("sum=%d",s);
    return 0;
}
int add()
{
    int a,b,sum;
    printf("Enter two number");
    scanf("%d %d",&a,&b);
    sum=a+b;
    return sum;
}

```

#### **Program 1:**

**Write a program to calculate simple interest using function & print the result inside main.**

```

#include<stdio.h>
float interest(float p,int t,float r);
int main()
{
    float pr,ra,in;
    int ti;
    printf("Enter principal,time and rate");
    scanf("%f %d %f",&pr,&ti,&ra);
    in=interest(pr,ti,ra);
    printf("Interest is %f",in);
    return 0;
}
float interest(float p,int t,float r)
{
    float i;
    i=(p*t*r)/100;
    return i;
}

```

#### **Program 2:**

**Write a program to find maximum of three number using function**

```

#include<stdio.h>
int maximum(int a, int b, int c);
int main()
{
    int x,y,z,max;
    printf("Enter three numbers:");
    scanf("%d%d%d",&x,&y,&z);
    max=maximum(x,y,z);
    printf("maximum is %d",max);
    return 0;
}

```

```

int maximum(int a, int b, int c)
{
    if(a>b&& a>c)
        return a;
    else if(b>c&& b>a)
        return b;
    else
        return c;
}

```

### Program 3:

**Write a program to find factorial of a number using function**

```

#include<stdio.h>
int fact(int num);
int main()
{
    int n,f;
    printf("Enter The number:");
    scanf("%d",&n);
    f=fact(n);
    printf("factorial is %d",f);
    return 0;
}
int fact(int num)
{
    int i,fa=1;
    for(i=1;i<=num;i++)
    {
        fa=fa*i;
    }
    return fa;
}

```

### Program 4:

**Write a program to generate Fibonacci series up to n terms using function**

```

#include<stdio.h>
void fibonacci(int num);
int main()
{
    int n;
    printf("Enter the term number you want to generate:");
    scanf("%d",&n);
    fibonacci(n);
    return 0;
}
void fibonacci(int num)
{
    int a=0,b=1,c,i;
    printf("%d\t%d\t",a,b);
    for(i=1;i<=num-2;i++)

```

```

    {
        c=a+b;
        printf("%d\t",c);
        a=b;
        b=c;
    }
}

```

## Types of Function calls in C

Functions are called by their names, we all know that, if the function does not have any arguments, then to call a function you can directly use its name. But for functions with arguments, we can call a function in two different ways, based on how we specify the arguments, and these two ways are:

- Call by Value
- Call by Reference

### 1) Call by value:

- If the function is called by passing the values of variable as actual argument, the function is called call by value.
- In this call value of each actual argument gets copied into formal arguments of our function definition.
- The content of calling function is not changed, Even if they are changed in called function.
- Pass value as argument.

Eg:

```

#include<stdio.h>
void swap(int a, int b);
int main()
{
    int x=90,y=70;
    printf("Value of X=%d and Y=%d\n",x,y);
    swap(x,y);
    return 0;
}
void swap(int a, int b)
{
    int temp;
    temp=a;
    a=b;
    b=temp;
    printf("After swap values a=%d and b=%d",a,b);
}

```

Output:

```

Value of X=90 and Y=70
After swap values a=70 and b=90

```



## 2) Call by reference:

- If the function is called by passing the address of variable as actual argument, the function is called call by reference.
- In this call, address of each actual argument gets copied into formal argument of function definition.
- The content of calling function is also changed, if any changed in called function.
- Pointer is used as formal argument in call by reference.
- Pass address as argument.

Eg:

```
#include<stdio.h>
void swap(int *a, int *b);
int main()
{
    int x=90,y=70;
    printf("Value of X=%d and Y=%d\n",x,y);
    swap(&x,&y);
    printf("After swap X=%d and Y=%d",x,y);
    return 0;
}
void swap(int *a, int *b)
{
    int temp;
    temp=*a;
    *a=*b;
    *b=temp;
}
```

Output:

```
Value of X=90 and Y=70
After swap X=70 and Y=90
```

## Concept of Recursion (Recursive Function):

Recursion is a programming technique where a function calls itself repeatedly until it reaches a certain condition or base case. A function that uses recursion is called a recursive function.

In a recursive function, the function usually breaks down the problem into smaller sub-problems, and then solves each sub-problem using the same function. The function then combines the solutions of the sub-problems to get the final solution.

Recursion is the process through which a function calls repeatedly until some specified condition is satisfied.

For recursive function:-

- The problem must be defined or written in terms of previous result.
- There must be stopping condition.

**Program 1:**

**Write a program to find factorial of a number using recursive function.**

```
#include<stdio.h>
int fact(int num);
int main()
{
    int f,n;
    printf("Enter the number:");
    scanf("%d",&n);
    f=fact(n);
    printf("Factorial is %d",f);
    return 0;
}
int fact(int num)
{
    if(num==0)
        return 1;
    else if(num==1)
        return 1;
    else
        return(num*fact(num-1));
}
```

**Program 2: Write a program to find particular term of Fibonacci series using recursive function.**

```
#include<stdio.h>
int fibo(int n);
int main()
{
    int num,f;
    printf("Enter the particular term you want to generate:");
    scanf("%d",&num);
    f=fibo(num);
    printf("The particular term is %d",f);
    return 0;
}
int fibo(int n)
{
    if(n==1)
        return 0;
    else if(n==2)
        return 1;
    else
        return(fibo(n-1)+fibo(n-2));
}
```

**Program 5: Write a program to print Fibonacci series up to nth term using recursive function.**

```
#include<stdio.h>
int fibonacci(int n);
int main()
{
    int n, i;
    printf("Enter the number of terms in the Fibonacci series: ");
    scanf("%d", &n);
    printf("Fibonacci series:\n");
    for (i = 0; i < n; i++)
    {
        printf("%d ", fibonacci(i));
    }
    return 0;
}
int fibonacci(int n)
{
    if (n <= 1)
        return n;
    else
        return fibonacci(n-1) + fibonacci(n-2);
}
```

**Program 4: Write a program to input a number and calculate the sum of all number up to that number using recursive function.**

```
#include<stdio.h>
int sum(int num);
int main()
{
    int n,s;
    printf("Input a number:");
    scanf("%d",&n);
    s=sum(n);
    printf("the sum upto that number: %d",s);
    return 0;
}
int sum(int num)
{
    if(num==1)
        return 1;
    else
        return (num+sum(num-1));
}
```

### Difference between recursive and iteration/looping:

Recursive	Iteration/looping
1) A function is called form definition of same function to do repeated task	1) In iteration, a function does not call itself
2) In recursion, a function calls itself until some condition is satisfied	2)In iteration, a function does not call itself
3) All problems cannot be solved through recursion	3) All problems can be solved through iteration
4) Recursive function are usually shorter in length	4) The length of iteration are longer than recursion of same program
5) There must be an exclusive if statement is recursive function specifying stopping condition	5) Iteration involves four steps declaration, initialization condition and updating

### Concept of storage class:

The storage class of a variable determines its scope, its storage duration, &its linkage, scope defines where the variable can be referenced in the program. Storage duration defines how long a variable exists in memory.

A storage class in C is used to describe the following things:

- The variable scope.
- The location where the variable will be stored.
- The initialized value of a variable.
- A lifetime of a variable.
- Who can access a variable?

Thus a storage class is used to represent the information about a variable.

There are four storage classes in c language:-

- Automatic storage class (auto)
- External storage class (extern)
- Static storage class (static)
- Register storage class (register)

### Table to understand storage class:

Storage class	Memory	Default value	Scope	Life time
Auto	Ram	Garbage (random +ve or random -ve number according to system)	Within a block(With in function)	Still the block active
Static	RAM	0	Within the block	Till the termination of program
Extern	RAM	0	Anywhere in the program	Till the termination of program
Register	Register	Garbage	Within block (fast access)	Still the block is active

## 1) Automatic storage class (auto):

The auto storage class is the default storage class for all local variables in C. It specifies that the variable has a local scope and a lifetime that lasts only for the duration of the block in which it is defined. The keyword "auto" is rarely used explicitly in C programming.

- Automatic variables are truly local
- When the function is exited, the value of an automatic variable is not retained
- They are recreated each time the function is called
- The default initial value of this variable is garbage i.e. random positive or negative numbers change according to the system
- The automatic variable must begin with the class specifier **auto**. However, it is optional.
- This storage class variable is also called a local variable

Eg:

```
#include<stdio.h>
void fun();
int main()
{
    fun();
    fun();
    fun();
}
void fun()
{
    auto int x=1;
    printf("%d\n",x);
    x=x+1;
}
```

Output:



## 2) Static storage class (static):

- Variables declared as static have a lifetime that lasts for the entire execution of the program.
- Static variables are allocated a fixed amount of memory at the beginning of the program's execution, and that memory is reserved for the entire lifetime of the program.
- Static variables retain their value between function calls, which can be useful for maintaining state.
- The keyword "static" is used to declare a variable as a static variable.
- Static variables are initialized to zero if no initial value is specified.

Eg:

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

```

void fun();
int main()
{
    fun();
    fun();
    fun();
}
void fun()
{
    static int x=1;
    printf("%d\n",x);
    x=x+1;
}

```

**Output:**



```

1
1
1

```

### 3) External storage class (extern):

The variable of the external storage class is declared outside any function. These functions are also called global variable

It has features like

- External variables are declared outside the functions
- All function in the program can be access and modify global variables
- The scope of the variable is global i.e with in the program
- The life-time of the variable is as long as the program's execution does not comes to an end
- The default initial value for this variable is zero
- An extern variable must begin with class specifier extern.
- This storage class variable is also called global variable

**Eg:**

```

#include<stdio.h>
int sum();
int mul();
extern int a=5,b=6;
int main()
{
    printf("\n the sum=%d",sum());
    printf("\n multiplication=%d",mul());
    return 0;
}
int sum()
{
    int s;

```

```

s=a+b;
return(s);
}
int mul()
{
int m;
m=a*b;
return(m);
}

```

**Output:**

```

the sum=11
multiplication=30

```

#### 4) Register Storage class (register):

The register storage class in C is used to define variables that can be stored in the CPU registers instead of RAM memory.

- The keyword **register** is used to declare a register storage class.
- The variables declared using register storage class has lifespan throughout the function.
- The variable is limited to the particular block.
- It is similar to the auto storage class. The only difference is that the variables declared using register storage class are stored inside CPU registers instead of a RAM.
- Register has faster access than that of the main memory.
- The default initial value of this variable is garbage i.e. random positive or negative number change according to system

**Eg:**

```

#include<stdio.h>
int main()
{
    register int a=5;
    printf("%d",a);
    return 0;
}

```

#### Local and global variables:

In C programming language, there are two main types of variables based on their scope: local variables and global variables.

##### Local Variables:

- Variables that are declared within or inside a function block are known as Local variables.
- These variables can only be accessed within the function in which they are declared.
- The lifetime of the local variable is within its function only, which means the variable exists till the function executes. Once function execution is completed, local variables are destroyed and no longer exist outside the function.

**Global variables:**

- Global variables are those variables which are declared outside of all the functions or block and can be accessed globally in a program.
- It can be accessed by any function present in the program.
- Once we declare a global variable, its value can be varied as used with different functions.
- The lifetime of the global variable exists till the program executes. These variables are stored in fixed memory locations given by the compiler and do not automatically clean up.
- Global variables are mostly used in programming and useful for cases where all the functions need to access the same data.

**Example of local and global variables:**



## Passing Array to Function in C

In C, there are various general problems which require passing more than one variable of the same type to a function. For example, consider a function which sorts the 10 elements in ascending order. Such a function requires 10 numbers to be passed as the actual parameters from the main function. Here, instead of declaring 10 different numbers and then passing into the function, we can declare and initialize an array and pass that into the function. This will resolve all the complexity since the function will now work for any number of values.

Consider the following syntax to pass an array to the function.

**functionname(arrayname);**

### Methods to declare a function that receives an array as an argument

There are 3 ways to declare the function which is intended to receive an array as an argument.

#### First way:

return\_type function(datatype arrayname[]) ;

Declaring blank subscript notation [] is the widely used technique.

#### Second way:

return\_type function(datatype arrayname[SIZE]) ;

Optionally, we can define size in subscript notation [].

#### Third way:

return\_type function(datatype \*arrayname);

You can also use the concept of a pointer. In pointer chapter, we will learn about it.

### Eg1: Passing single element at a time

```
#include<stdio.h>
void display(int x);
int main()
{
    int i;
    int a[5]={5,10,15,20,25};
    for(i=0;i<5;i++)
    {
        display(a[i]);
    }
    return 0;
}
void display(int x)
{
    printf("%d\t",x);
}
```

### Eg2: Passing whole array at a time

```
#include<stdio.h>
void display(int d[]);
int main()
{
    int a[5]={10,20,30,40};
    display(a);
    return 0;
}
```

```

}
void display(int d[])
{
    int i;
    for(i=0;i<5;i++)
    {
        printf("%d\t",d[i]);
    }
}

```

#### Program 1:

Write a program to find sum of all number in an array using function.

```

#include<stdio.h>
int i;
void display(int d[],int m);
void sum(int s[],int p);
int main()
{
    int a[100],n;
    printf("Enter how many elements:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("Enter the element of a[%d]:",i);
        scanf("%d",&a[i]);
    }
    display(a,n);
    sum(a,n);
    return 0;
}
void display(int d[],int m)
{
    printf("Element of array are\n");
    for(i=0;i<m;i++)
    {
        printf("%d\t",d[i]);
    }
}
void sum(int s[],int p)
{
    int sum1=0;
    for(i=0;i<p;i++)
    {
        sum1=sum1+s[i];
    }
    printf("\nsum of all element =%d",sum1);
}

```

## Program 2:

Write a program to input n number in an array and sort them using fuction

```
#include<stdio.h>
void display(int d[],int m);
void sorting(int s[],int p);
int i;
int main()
{
    int a[100],n;
    printf("Enter how many element:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("Enter the element of a[%d]",i);
        scanf("%d",&a[i]);
    }
    display(a,n);
    sorting(a,n);
    return 0;
}
void display(int d[],int m)
{
    printf("\nArray element are\n");
    for(i=0;i<m;i++)
    {
        printf("%d\t",d[i]);
    }
}
void sorting(int s[],int p)
{
    int j,temp;
    for(i=0;i<p-1;i++)
    {
        for(j=i+1;j<p;j++)
        {
            if(s[i]>s[j])
            {
                temp=s[i];
                s[i]=s[j];
                s[j]=temp;
            }
        }
    }
    display(s,p);
}
```

Passing 2-D array to the function:

Program 3: Write a program to input 3\*3 order matrix and display it using function

```
#include<stdio.h>
int i,j;
void display(int d[3][3]);
int main()
{
    int a[3][3];
    printf("Enter the element of matrix:\n");
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("Enter the Element of a[%d][%d]",i,j);
            scanf("%d",&a[i][j]);
        }
    }
    display(a);
    return 0;
}
void display(int d[3][3])
{
    printf("Entered matrix is:\n");
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("%d\t",d[i][j]);
        }
        printf("\n");
    }
}
```

Program 4:

Write a program to add two 3\*3 matrices using function. Three separate function for input, display and addition.

```
#include<stdio.h>
void input(int l[3][3]);
void display(int d[3][3]);
void addition(int m1[3][3],int m2[3][3]);
int i,j;
int main()
{
    int a[3][3],b[3][3];
    printf("Enter the element of a:\n");
    input(a);
    printf("Enter the element of b:\n");
    input(b);
    printf("first matrix:\n");
```

```

display(a);
printf("second matrix:\n");
display(b);
addition(a,b);
return 0;
}

void input(int l[3][3])
{
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
scanf("%d",&l[i][j]);
}
}
}

void addition(int m1[3][3],int m2[3][3])
{
int c[3][3];
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
c[i][j]=m1[i][j]+m2[i][j];
}
}
printf("Resultant matrix is \n");
display(c);
}

void display(int d[3][3])
{
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
printf("%d\t",d[i][j]);
}
printf("\n");
}
}

```