

# Unit 8: Modular Programming with Functions

Er.Nipun Thapa

# Introduction

- 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.**
- We can divide up our code into separate functions.
- How it divide up our code among different functions is up to us, but logically the division usually is so each function performs a specific task.
- 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.**
- The C standard library provides numerous built-in functions that our program can call.
  - For example, function **strcat()** to concatenate two strings, function **memcpy()** to copy one memory location to another location and many more functions.
- A function is known with various names like a method or a sub-routine or a procedure, etc.

# Introduction

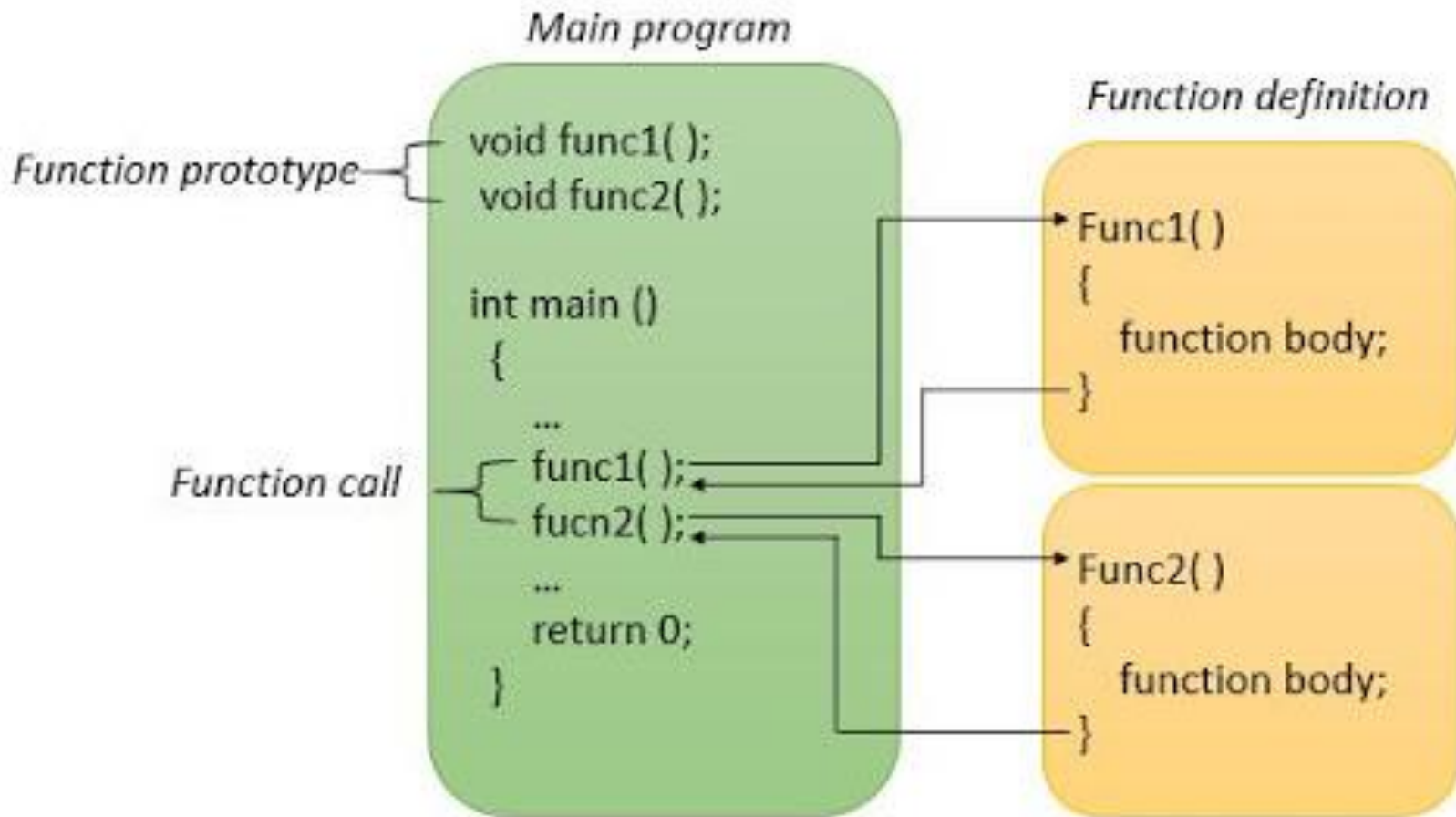
So function in a C program has some properties discussed below.

- Every function has a **unique name**. This name is used to call function from “main()” function. A function can be called from within another function.
- A function is **independent** and it can perform its task without intervention from or interfering with other parts of the program.
- A function performs a **specific task**. A task is a distinct job that your program must perform as a part of its overall operation, such as adding two or more integer, sorting an array into numerical order, or calculating a cube root etc.
- A function returns a **value** to the calling program. This is optional and depends upon the task your function is going to accomplish. Suppose you want to just show few lines through function then it is not necessary to return a value. But if you are calculating area of rectangle and wanted to use result somewhere in program then you have to send back (return) value to the calling function.

# Introduction

- C language is collection of various inbuilt functions.
- If you have written a program in C then it is evident that you have used C's inbuilt functions. Printf, scanf, clrscr etc. all are C's inbuilt functions.
- *You cannot imagine a C program without function.*
- **Function in C programming** is a reusable block of code that makes a program easier to understand, test and can be easily modified without changing the calling program. Functions divide the code and modularize the program for better and effective results. In short, a larger program is divided into various subprograms which are called as functions

# Introduction



# Advantage

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.

# Types of Functions

There are two types of functions in C programming:

- **Library Functions:**

are the functions which are declared in the C header files such as `scanf()`, `printf()`, `gets()`, `puts()`, `ceil()`, `floor()` etc.

- **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.

# Elements or Components of Function

A function usually has three components. They are:

1. Function Prototype/Declaration
2. Function Definition
3. Function Call



# 1. Function Prototype/Declaration

- Function declaration means writing a name of a program.
- It is a compulsory part for using functions in code.
- In a function declaration, we just specify the name of a function that we are going to use in our program like a variable declaration.
- We cannot use a function unless it is declared in a program.
- A function declaration is also called "Function **prototype**."
- Function declaration is a statement that informs the compiler about
  - Name of the function
  - Type of arguments
  - Number of arguments
  - Type of Return value

# 1. Function Prototype/Declaration

The function declarations (called prototype) are usually done above the main () function and take the general form:

**return\_data\_type function\_name (data\_type arguments);**

- The **return\_data\_type**: is the data type of the value function returned back to the calling statement.
- The **function\_name**: is followed by parentheses
- **Arguments** names with their data type declarations optionally are placed inside the parentheses.

# 1. Function Prototype/Declaration

- We consider the following program that shows how to declare a cube function to calculate the cube value of an integer variable

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

```
/*Function declaration*/
```

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

```
/*End of Function declaration*/
```

## 2. Function Definition

- Function definition means just writing the body of a function. A body of a function consists of statements which are going to perform a specific task. A function body consists of a single or a block of statements. It is also a mandatory part of a function.

### Syntax for function definition

```
returntype function_name ([arguments])
{
    statement(s);
    ... ..
}
```

Example :

```
int add(int a,int b) //function body
{
    int c;
    c=a+b;
    return c;
}
```

# 3. Function call

- A function call means calling a function whenever it is required in a program. Whenever we call a function, it performs an operation for which it was designed. A function call is an optional part in a program.

## Syntax for function call

```
function_name ([actual arguments]);
```

## For example,

```
sort(a);
```

```
p = add(x,y);
```

## LAB 6: Q.N.1: Program to find the sum of integer using function

```
#include<stdio.h>
int add(int x,int y);
int main()
{
    int a=10,b=20,sum;
    sum=add(a,b);
    printf("The sum of %d and %d = %d",a,b,sum);
    return 0;
}
int add(int x,int y)
{
    int m;
    m=x+y;
    return m;
}
```

LAB 6: Q.N.2: Program to find greatest number among two number using function

```
#include<stdio.h>
int large(int x,int y)
{
    if(x>y)
        return x;
    else
        return y;
}
int main()
{
    int a,b,c;
    printf("Enter two number:");
    scanf("%d%d",&a,&b);
    c=large(a,b);
    printf("Large value = %d",c);
    return 0;
}
```

## LAB 6: Q.N.3: Program to find greatest number among three number using function

```
#include<stdio.h>
int large(int x,int y)
{
    if(x>y)
        return x;
    else
        return y;
}
int main()
{
    int a,b,c,d,e;
    printf("Enter two number:");
    scanf("%d%d%d",&a,&b,&c);
    d=large(a,b);
    e=large(d,c);
    printf("Large value = %d",e);
    return 0;
}
```



# 4. The return and void statement

The return statement serves two purpose:

- It immediately transfer the control back to the calling function (ie no statements within the function body after the return statement are executed).
- It return the value to the calling function.

## Syntax

return (expression)

*Where expression, is optional and, if present, it must evaluate to a value of the data type specified in the function header for the return\_type.*

# Function 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. A Parameter is the symbolic name for "data" that goes into a function. There are two ways to pass parameters in C: Pass by Value, Pass by Reference.
- **Pass by Value**
  - Pass by Value, means that a copy of the data is made and stored by way of the name of the parameter. Any changes to the parameter have **NO** affect on data in the calling function.
- **Pass by Reference**
  - A **reference parameter** "refers" to the original data in the calling function. Thus any changes made to the parameter are **ALSO MADE TO THE ORIGINAL** variable.

# Categories of user defined function (Different forms of function)

There can be 4 different types of user-defined functions, they are:

- Function with no arguments and no return value
- Function with no arguments and a return value
- Function with arguments and no return value
- Function with arguments and a return value

# 1.Function with no arguments and no return value

- In this method, We won't pass any arguments to the function while defining, declaring, or calling the function.
- This type of functions in C will not return any value when we call the **function** from main() or any sub-function.
- When we are not expecting any return value, but we need some statements to print as output. Then, this type of function in C is very useful.

# 1.Function with no arguments and no return value

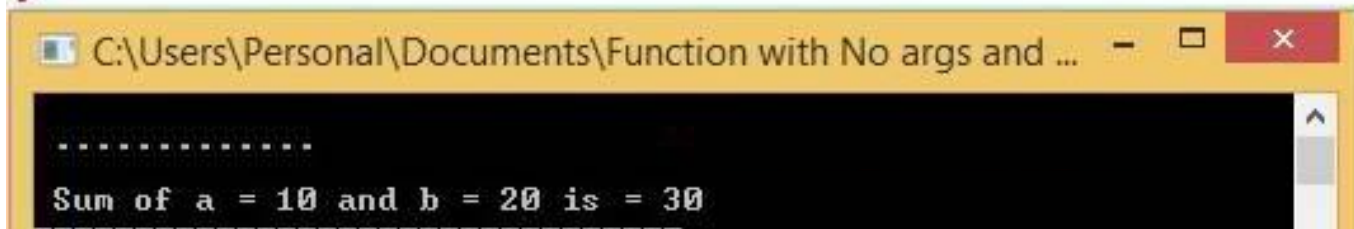
- In these types of Functions in C program, We are going to calculate the Sum of 2 integer values and print the output from the user-defined function itself.

LAB 6: Q.N.4

```
#include<stdio.h>
void Addition();

void main()
{
    printf("\n ..... \n");
    Addition();
}

void Addition()
{
    int Sum, a = 10, b = 20;
    Sum = a + b;
    printf("\n Sum of a = %d and b = %d is = %d", a, b, Sum);
}
```



```
C:\Users\Personal\Documents\Function with No args and ...
.....
Sum of a = 10 and b = 20 is = 30
```

## 2. Function with no arguments and a return value

- In this method, We won't pass any arguments to the function while defining, declaring, or calling the function.
- This type of function will return some value when we call the function from main() or any sub function.
- The Data Type of the return value will depend upon the return type of function declaration.
- For instance, if the return type is int then return value will be int.

## 2. Function with no arguments and a return value

- In this program, We are going to calculate the multiplication of 2 integer values using the user-defined function without arguments and return keyword.

LAB 6: Q.N.5

```
#include<stdio.h>
int Multiplication();
int main()
{
    int Multi;
    Multi = Multiplication();
    printf("\n Multiplication of a and b is = %d \n", Multi );
    return 0;
}
int Multiplication()
{
    int Multi, a = 20, b = 40;
    Multi = a * b;
    return Multi;
}
```

### 3.Function with arguments and no return value

- If you observe the above two methods, No matter how many times you executive, it will give the same output. We don't have any control over the values of the variables a and b because they are fixed values.
- In real-time, we mostly deal with dynamic data means we have to allow the user to enter his own values rather than fixed ones.
- This method allows us to pass the arguments to the function while calling the function. But, This type of function will not return any value when we call the function from main () or any sub function.
- If we want to allow the user to pass his data to the function arguments, but we are not expecting any return value, this type of function is very useful.



### 3.Function with arguments and no return value

- These Types of Functions in C program allows the user to enter 2 integer values. Next, We are going to pass those values to the user-defined function to calculate the sum.

LAB 6: Q.N.6

```
#include<stdio.h>
void Addition(int a, int b);
void main()
{
    int a, b;
    printf("\n Please Enter two integer values \n");
    scanf("%d %d",&a, &b);
    //Calling the function with dynamic values
    Addition(a, b);
}
void Addition(int a, int b)
{
    int Sum;
    Sum = a + b;
    printf("\n Additiontion of %d and %d is = %d \n", a, b, Sum);
}
```

## 4. Function with arguments and a return value

- This method allows us to pass the arguments to the function while calling the function.
- This type of function will return some value when we call the function from main () or any sub function.
- Data Type of the return value will depend upon the return type of function declaration. For instance, if the return type is int then return value will be int.
- This type of user-defined function is called a fully dynamic function, and it provides maximum control to the end-user.

## 4. Function with arguments and a return value

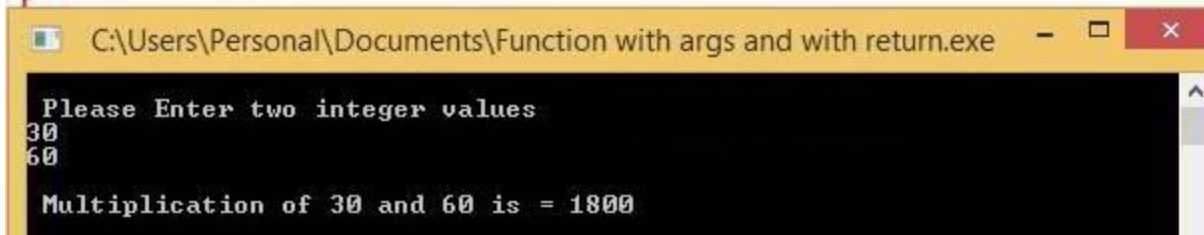
- This Types of Functions in C program allows the user to enter 2 integer values. And then, We are going to pass those values to the user-defined function to multiply those values and return the value using the return keyword.

LAB 6: Q.N.7

```
#include<stdio.h>
int Multiplication(int, int);

int main()
{
    int a, b, Multi;
    printf("\n Please Enter two integer values \n");
    scanf("%d %d",&a, &b);

    Multi = Multiplication(a, b);
    printf("\n Multiplication of %d and %d is = %d \n", a, b, Multi);
    return 0;
}
int Multiplication(int a, int b)
{
    int Multi;
    Multi = a * b;
    return Multi;
}
```



```
C:\Users\Personal\Documents\Function with args and with return.exe
Please Enter two integer values
30
60
Multiplication of 30 and 60 is = 1800
```

## Example 1: No arguments passed and no return value

LAB 6: Q.N.8

```
void checkPrimeNumber();
int main()
{
    checkPrimeNumber();
    return 0;
}
void checkPrimeNumber()
{
    int n, i, flag = 0;
    printf("Enter a positive integer: ");
    scanf("%d",&n);
```

```
    for(i=2; i <= n/2; ++i)
    {
        if(n%i == 0)
        {
            flag = 1;
            break;
        }
    }
    if (flag == 1)
        printf("%d is not a prime number.", n);
    else
        printf("%d is a prime number.", n);
}
```

## Example 2: No arguments passed but a return value

LAB 6: Q.N.9

```
#include <stdio.h>

int getInteger();

int main()
{
    int n, i, flag = 0;
    n = getInteger();
    for(i=2; i<=n/2; ++i)
    {
        if(n%i==0){
            flag = 1;
            break;
        }
    }
    if (flag == 1)
        printf("%d is not a prime number.", n);
    else
        printf("%d is a prime number.", n);
    return 0;
}
```

```
int getInteger()
{
    int n;
    printf("Enter a positive integer: ");
    scanf("%d",&n);
    return n;
}
```

## Example 3: Argument passed but no return value

LAB 6: Q.N.10

```
#include <stdio.h>
void checkPrimeAndDisplay(int n);
int main()
{
    int n;
    printf("Enter a positive integer: ");
    scanf("%d",&n);
    checkPrimeAndDisplay(n);
    return 0;
}
```

```
void checkPrimeAndDisplay(int n)
{
    int i, flag = 0;
    for(i=2; i <= n/2; ++i)
    {
        if(n%i == 0){
            flag = 1;
            break;
        }
    }
    if(flag == 1)
        printf("%d is not a prime number.",n);
    else
        printf("%d is a prime number.", n);
}
```

## Example 4: Argument passed and a return value

LAB 6: Q.N.11

```
#include <stdio.h>

int checkPrimeNumber(int n);

int main()
{
    int n, flag;
    printf("Enter a positive integer: ");
    scanf("%d",&n);
    flag = checkPrimeNumber(n);
    if(flag == 1)
        printf("%d is not a prime number",n);
    else
        printf("%d is a prime number",n);
    return 0;
}
```

```
int checkPrimeNumber(int n)
{
    int i;
    for(i=2; i <= n/2; ++i)
    {
        if(n%i == 0)
            return 1;
    }
    return 0;
}
```

# Recursion and Recursive function

- Recursion is the process of repeating items in a self-similar way. In programming languages, if a program allows you to call a function inside the same function, then it is called a recursive call of the function.

```
void recursion()  
{  
    recursion(); /* function calls itself */  
}  
int main()  
{  
    recursion();  
}
```

- The C programming language supports recursion, i.e., a function to call itself. But while using recursion, programmers need to be careful to define an exit condition from the function, otherwise it will go into an infinite loop.
- Recursive functions are very useful to solve many mathematical problems, such as calculating the factorial of a number, generating Fibonacci series, etc.

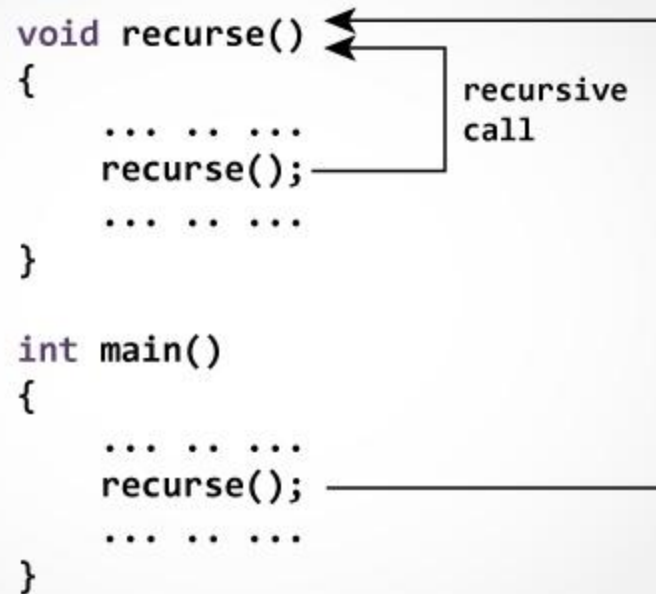


# Recursion and Recursive function

How does recursion work?

```
void recurse()
{
    ... ..
    recurse();
    ... ..
}

int main()
{
    ... ..
    recurse();
    ... ..
}
```



## LAB 6 Q.N.12 : Program to compute the factorial of a number using recursion.

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

### LAB 6 Q.N.13: Program to find Fibonacci series of a number using recursion.

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

# Comparison of iteration and recursion

Property	Recursion	Iteration
<b>Definition</b>	Function calls itself.	A set of instructions repeatedly executed.
<b>Application</b>	For functions.	For loops.
<b>Termination</b>	Through base case, where there will be no function call.	When the termination condition for the iterator ceases to be satisfied.
<b>Usage</b>	Used when code size needs to be small, and time complexity is not an issue.	Used when time complexity needs to be balanced against an expanded code size.
<b>Code Size</b>	Smaller code size	Larger Code Size.
<b>Time Complexity</b>	Very high(generally exponential) time complexity.	Relatively lower time complexity(generally polynomial-logarithmic).
<b>Example</b>	<pre>int fact(int n) {   If(n==0)     return 1;   else     return (n* fact(n-1)); }</pre>	<pre>int fact(int n) { int l, result=1; If(n==0 )     return 1; else     for(i=1;i&lt;=n;i++)     result=result * l; return result; }</pre>

# Passing Array to function

- Just like variables, array can also be passed to a function as an argument .

Passing array to function in C

```
void func( int a[] , int size )  
{  
}  
  
int main( )  
{  
    int n=5;  
    int arr[5] = { 1, 2, 3, 4, 5 };  
    func( arr , n);  
    return 0;  
}
```

Pointer a takes the base address of array arr

Pointer to arr

Length of arr

The length of arr is passed. It is compulsory to pass size as is just a pointer

## LAB 6: Q.N.13. : Program to read 10 number in an array and find their sum and display using the function

```
#include <stdio.h>
int sum(int n[]);
void display(int n[]);
int main()
{
    int i,a[10];
    printf("Enter 10 number:");
    for(i=0;i<10;i++)
        scanf("%d",&a[i]);
    display(a);
    printf("\nThe sum of 10 number=%d",sum(a));
    return 0;
}
```

```
int sum(int n[])
{
    int i,sum=0;
    for(i=0;i<10;i++)
        sum=sum+n[i];
    return sum;
}
void display(int n[])
{
    int i;
    printf("Your 10 numbers are:\n");
    for(i=0;i<10;i++)
        printf("%d\t",n[i]);
}
```

## LAB 6.Q.N.14: Passing two-dimensional arrays

```
#include <stdio.h>
void displayNumbers(int num[2][2]);
int main()
{
    int num[2][2];
    printf("Enter 4 numbers:\n");
    for (int i = 0; i < 2; ++i)
        for (int j = 0; j < 2; ++j)
            scanf("%d", &num[i][j]);

    // passing multi-dimensional array to a
    // function
    displayNumbers(num);
    return 0;
}
```

```
void displayNumbers(int num[2][2])
{
    printf("Displaying:\n");
    for (int i = 0; i < 2; ++i) {
        for (int j = 0; j < 2; ++j) {
            printf("%d\n", num[i][j]);
        }
    }
}
```

# Different types of function calls

The arguments in function can be passed in two ways:

1. Pass arguments by value
2. Pass argument by address or reference or pointers.



# Pass arguments by value

LAB 6: Q.N.15: program to swap two number using call by value

```
#include <stdio.h>
void swap(int , int );
int main()
{
    int a,b;
    a=100;
    b=55;
    printf("\nBefore swapping a=%d\t b=%d :",a,b);
    swap(a,b);
    printf("\nAfter swapping a=%d\t b=%d :",a,b);
    return 0;
}
void swap(int a,int b)
{
    int temp;
    temp=a;
    a=b;
    b=temp;
    printf("\nThe function within function are:a=%d\tb=%d",a,b);
}
```

# Pass argument by address or reference or pointers.

LAB 6: Q.N.16: program to swap two number using call by reference

```
#include <stdio.h>
void swap(int * , int * );
int main()
{
    int a,b;
    a=100;
    b=55;
    printf("\nBefore swapping a=%d\t b=%d :",a,b);
    swap(&a,&b);
    printf("\nAfter swapping a=%d\t b=%d :",a,b);
    return 0;
}
void swap(int *a,int *b)
{
    int temp;
    temp=*a;
    *a=*b;
    *b=temp;
    printf("\nThe function within function are:a=%d\tb=%d" ,*a,*b);
}
```

# Pass arguments by value

LAB 6: Q.N.17: program to illustrate using call by value

```
#include<stdio.h>
int calc(int x);
int main()
{
    int x = 10;
    x = calc(x);
    printf("value of x is %d", x);
    return 0;
}
int calc(int x)
{
    x = x + 10 ;
    return x;
}
```

# Pass argument by address or reference or pointers.

LAB 6: Q.N.18: program to illustrate using call by reference

```
#include<stdio.h>
int calc(int *x);
int main()
{
    int x = 10;
    x = calc(&x);
    printf("value of x is %d", x);
    return 0;
}
int calc(int *x)
{
    *x = *x + 10 ;
    return *x;
}
```

# Finished

# Unit 8