Module 3

Functions – user defined functions and built-in functions, Function definition, Function call, Function prototypes, Passing arguments to functions.

Modular Programming

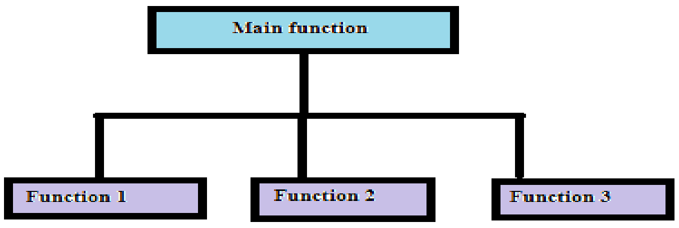
- Modular programming is the process of subdividing a computer program into separate sub programs called functions or procedures.
- A module is a separate software component.
- Module is basically a set of interrelated files that share their implementation details but hide it from the outside world
- It can often be used in a variety of applications and functions with other components of the system.
- It increases the maintainability, readability of the code and to make the program handy to make any changes in future or to correct the errors.

Advantages of Using Modular Programming

1. **Ease of Use** : It allows simplicity, ease in debugging the code and prone to less error.
2. **Reusability**: It allows the user to reuse the functionality with a different interface without typing the whole program again.
3. **Ease of Maintenance**: It helps in less collision at the time of working on modules, helping a team to work with proper collaboration while working on a large application.

Functions:

- A function is a group of statements that perform a specific task.
- A function can also be referred as a method or a sub-routine or a procedure, etc.
- Function 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 general, a larger program is divided into various subprograms which are called as *functions.*
- Every C program has at least one function, which is **main().**



Needs of Functions in C

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

Types of Function :

- There are two types of function in C programming:
  - ✓ ***Standard library functions***
  - ✓ ***User-defined functions***

Standard Library Functions

- Also known as built-in functions.
- These functions are defined in header files.

Example:

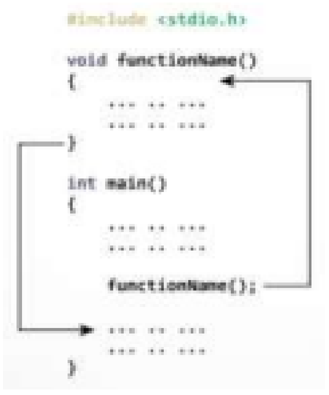
- ***printf()*** is a standard library function to send formatted output to the screen. This function is defined in the stdio.h header file. Hence, to use the printf() function, we need to include the stdio.h header file using #include <stdio.h>.
- ***sqrt()*** function calculates the square root of a number. The function is defined in the math.h header file.

User Defined Functions

- Functions that are defined by the user at the time of writing the program.
- The functions that we create in a program are known as user defined functions.
- In other words, a function created by user is known as user defined function, but later it can be a part of 'C' library.
- Functions are made for code re-usability and for saving time and space.
- In C, main() is the user-defined function and first calling function in any program.
- main() is a special function that tells the compiler to start the execution of a C program from the beginning of the function main().

How user-defined function works?

```
#include <stdio.h>
void functionName()
{ - - -- - - - - -
            - - - - - -----
}
int main()
{ ……….
           functionName();
             ………………
}
```



```
#include <stdio.h>

void functionName()
{
    ... .. ...
    ... .. ...
}

int main()
{
    ... .. ...
    ... .. ...

    functionName();

    ... .. ...
    ... .. ...
}
```

- The execution of a C program begins from the main() function.
- When the compiler encounters functionName();, control of the program jumps to void functionName().
- The compiler starts executing the codes inside the functionName().
- The control of the program jumps back to the main() function once code inside the function definition is executed.

How to Define a Function in C

A function definition in C programming consists of a *function header* and a*function body*.

The general form of a function definition is:

```
return_type function_name (parameter list/argument list)
{
  body of the function – Block of code
}
```

- ***return_type*** − A function may return a value. The **return_type** can be of any data type of the value the function return such as int, double, char, void etc.. 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*.

- ***parameter list*** − A parameter is like a placeholder. Argument list contains variables names along with their data types. These arguments are kind of inputs for the function. 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 C statements that define what the function does.

Example : Sum of two numbers using function

```
#include <stdio.h>
int addNumbers(int a, int b); // function prototype

int main()
{

int n1,n2,sum;

 printf("Enters two numbers: ");
 scanf("%d %d",&n1,&n2);

 sum = addNumbers(n1, n2); // function call
 printf("sum = %d",sum);
return 0;
}

int addNumbers(int a, int b) // function definition
{
 int result;
 result = a+b;
 return result; // return statement
}
```

Elements of user-defined function / Function Activities

· ***C programming has three elements or three activities.***

· They are :

1. **Function Declaration /Function Prototype**
2. **Function Definition**
3. **Function call**

Function Declaration/Function Prototype

- A function declaration is a frame (prototype)of function that contains the function's name, list of parameter and return type and ends with the semicolon.
- It doesn't contain function body.
- A function prototype gives information to the compiler that the function may later be used in the program.
- The function declarations (called prototype) are usually done above the main () function.
- General form is : return_type functionName(type1 argument1, type2 argument2, ...);
  Example: int add(int x, int y); // Function Declaration

1. *function name* – name of the function is **add()**
2. *return type* – the return type of the function is **int**
3. *arguments* – two arguments (x and y) of type int are passed to the function ·

Parameter names are not important in function declaration only their type is required

Example : int max(int, int);

Function Definition

- The function definition is an expansion of function declaration.
- It contains codes in the body part of the function for execution program by the compiler · Function definition contains the block of code 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.

The syntax of the function definition is :

```
return_type function_name(parameter_1, parameter_2,....){
        //statements
        //body of the function
}
```

Example:

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

Calling a Function/Function Call

- A function call means calling a function whenever it is required in a program.
- A function call is an optional part in a program.
- When a function is called, the control of the program is transferred to the function definition and the compiler starts executing the codes inside the body of a function.

Syntax : function_Name(Argument List);

Example: sum(5, 4);

Program to find the maximum of two numbers

```
int max(int num1,int num2); //Function Declaration
int main (){
            int a =100, b = 200, result; //local variable definition
            result = max(a, b); // Calling Function
            printf("Max value is : %d\n", result );
            return0;
}
int max(int num1,int num2){ // Function Definition
```

```
                int result;
                if(num1 > num2)
                        result= num1;
                else
                        result= num2;
                return result;
        }
```

Function Arguments(Parameters)

- A function in C can be called either with arguments or without arguments.
- These function may or may not return values to the calling functions.
- All C functions can be called either with arguments or without arguments in a C program.
- Also, they may or may not return any values.
- A function's arguments are used to receive the necessary values by the function call. · They are matched by position; the first argument is passed to the first parameter, the second to the second parameter and so on.
- A Parameter is the symbolic name for "data" that goes into a function .
- By default, **the arguments are passed by value** in which a copy of data is given to the called function.
- 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.
- Formal parameters behave like other local variables inside the function and are created upon entry into the function and destroyed upon exit.

· Arguments are of two types :

1. **Actual argument**: Argument present in the function call statement.
2. **Formal argument**: Argument present in the function definition.

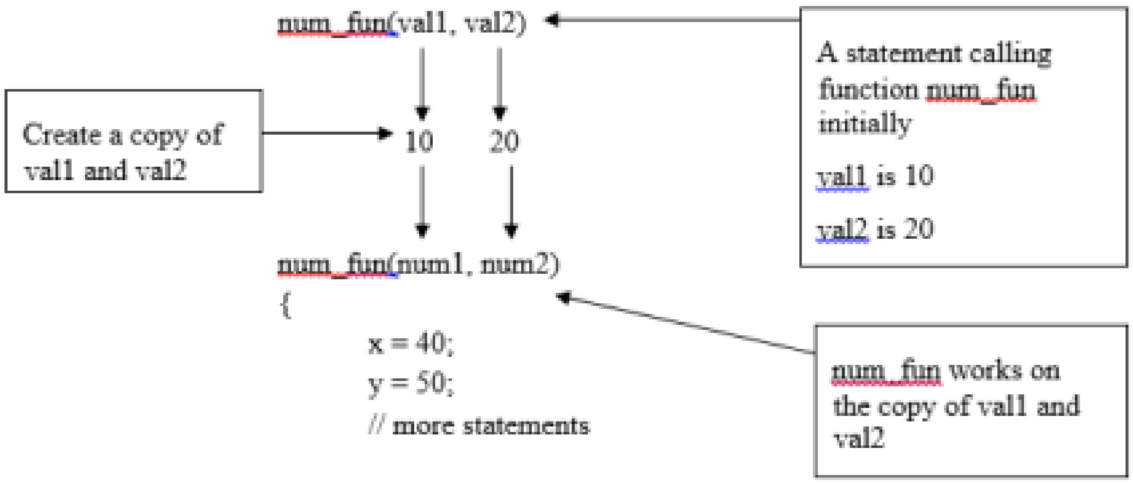
· While calling a function, there are two ways in which arguments can be passed to a function

1. Call by value, and
2. Call by reference

1. Call by value

- Actual parameter is passed to a function.
- This method **copies** the **actual value** of an argument into the **formal parameter** of the function.
- New memory area created for the passed parameters can be used only within the function. · The actual parameter cannot be modified here.
- All changes made to the parameter inside the function have **no effect** on the **actual argument**.
- By default, C programming uses call by value method to pass arguments.

Call by value - working


num_fun(val1, val2)
Create a copy of val1 and val2
10
20
num_fun(num1, num2)
{
x = 40;
y = 50;
// more statements
A statement calling function num_fun initially
val1 is 10
val2 is 20
num_fun works on the copy of val1 and val2

Example

```
#include <stdio.h>
int sum (int n);
void main()
 {
 int a = 5;
 printf("\n The value of 'a' before the calling function is = %d", a);
 a = sum(a);
 printf("\n The value of 'a' after calling the function is = %d", a);
 }
int sum (int n)
 {
 n = n + 20;
 printf("\n Value of 'n' in the called function is = %d", n);
```

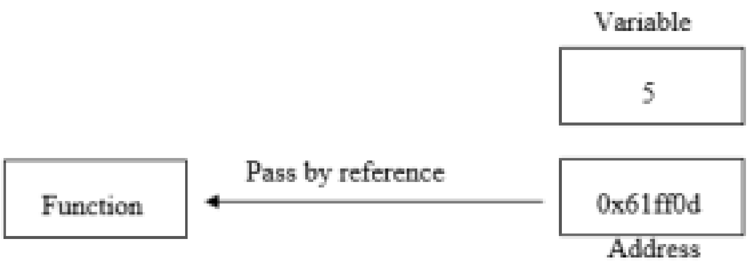
```
    return n;
    }
```

Output

```
The value of 'a' before the calling function is = 5
Value of 'n' in the called function is = 25
The value of 'a' after calling the function is = 25
```

2. Call by reference

- ❖ Here, the address of the variables are passed by the calling function to the called function.
- ❖ The address which is used inside the function is used to access the actual argument used in the call.
- ❖ If there are any changes made in the parameters, they affect the passed argument.
- ❖ For passing a value to the reference, the argument pointers are passed to the functions just like any other value.



There are two ways to make a pass by reference parameter:

1. **Arrays -** Arrays are **always** pass by reference in C. Any change made to the parameter containing the array will change the value of the original array.
2. The ampersand used in the function prototype.

function (**& parameter_name)**

To make a normal parameter into a pass by reference parameter, we use the "& param" notation. The ampersand (&) tells the compiler that any changes made to the parameter also modify the original variable containing the data.

- ❖ Instead of copying variable; an address is passed to function as parameters.
- ❖ Address operator(&) is used in the parameter of the called function.

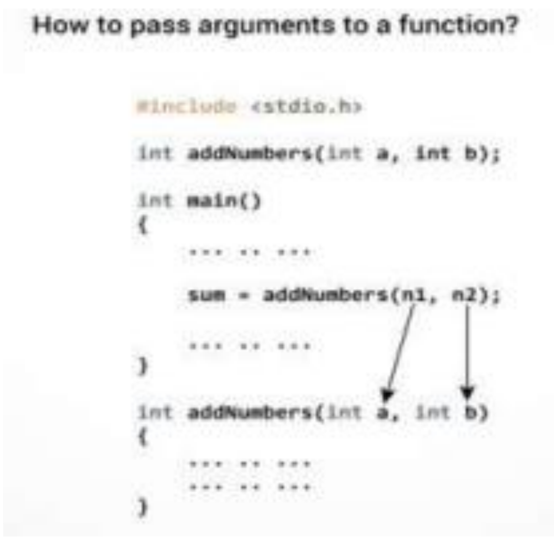
- ❖ Changes in function reflect the change of the original variables.

<u>Example</u>

```
#include <stdio.h>
int sum (int *n);
void main()
{
 int a = 5;
 printf("\n The value of 'a' before the calling function is = %d", a);
 sum(&a);
 printf("\n The value of 'a' after calling the function is = %d", a);
}
int sum (int *n)
{
 *n = *n + 20;
 printf("\n value of 'n' in the called function is = %d", n);
}
```

<u>Output</u>

```
The value of 'a' before the calling function is = 5
value of 'n' in the called function is = -1079041764
The value of 'a' after calling the function is = 25
```

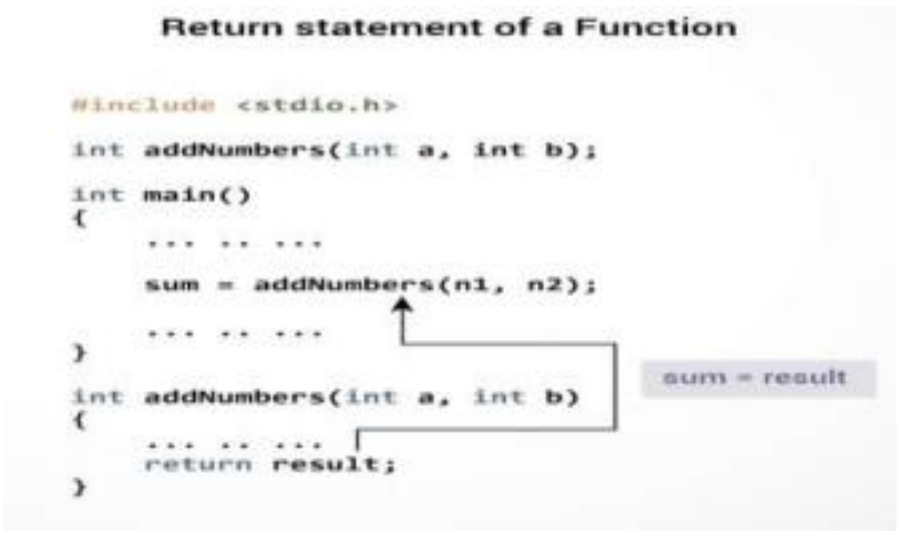

How to pass arguments to a function?

```
#include <stdio.h>
int addNumbers(int a, int b);
int main()
{
     ... .. ...
     sum = addNumbers(n1, n2);
     ... .. ...
}
int addNumbers(int a, int b)
{
     ... .. ...
     ... .. ...
}
```

<u>Return Statement</u>

- ➢ The return statement terminates the execution of a function and returns a value to the calling function.
- ➢ The program control is transferred to the calling function after the return statement.
  Syntax of return statement

return (expression);
Example : return a;
return (a+b);


Return statement of a Function

```
#include <stdio.h>
int addNumbers(int a, int b);
int main()
{
    ... .. ...
    sum = addNumbers(n1, n2);
    ... .. ...
}
int addNumbers(int a, int b)
{
    ... .. ...
    return result;
}
```

<u>Scope of Variables</u>

- ➢ The scope is the region in any program where the defined variable has its existence.
- ➢ It cannot be accessed beyond this scope.

· In C programming language, there are three places where the variables can be declared:

1. Local variables which are inside a function or a block.
2. Global variables which are outside all the functions.
3. Formal parameters which are defined in the function parameters.

<u>Local Variables</u>

- Variables that are declared inside a function or block.
- They can be used only by statements that are inside that function or block of code.
- They are not known to functions outside.

<u>Example</u>

```
int main (){
            /* local variable declaration */
            int a, b;
            int c;
            /* actual initialization */
             a =10; b =20;
             c = a + b;
            printf("value of a = %d, b = %d and c = %d\n", a, b, c);
```

```
                    return0;
    }
```

Global Variables

- Global variables are defined outside a function or block, usually on top of the program.
- Global variables hold their values throughout the lifetime of your program.
- They can be accessed anywhere in the program and in any function.

Example

```
    #include<stdio.h>
    int g; /* global variable declaration */
    int main (){
                    int a, b; /* local variable declaration */
                    /* actual initialization */
     a =10; b =20;
     g = a + b;
                    printf("value of a = %d, b = %d and g = %d\n", a, b, g);
                    return0;
    }
```

A program can have same name for local and global variables but the value of local variable inside a function will take preference.

```
    #include<stdio.h>

    int g =20; /* global variable declaration */

    int main (){

            int g =10; /* local variable declaration */

            printf("value of g = %d\n", g);

            return0;

    }
```

Formal parameters

- They are the local variables within the function and take a precedence over the global variables.
- These variables behave like local variables in the function.
- They can be accessed inside the function but not outside the function.

Types of user defined functions in C

- Depending upon the presence of arguments and the return values, user defined functions can be classified into ***five categories***.

· They are:

1. ***Function with no arguments and no return values***
2. ***Function with no arguments and return value***
3. ***Function with arguments and no return values***
4. ***Function with arguments and return value***

1.Function with no arguments and no return value

- Such functions neither receives any data from the calling function nor returns a value.
- In other words, the calling function does not get any value from the called function.
- Such functions can either be used to display information because they are completely dependent on user inputs.
- In this type, each function is independent.
- So there is no data transfer between calling and called function.

Example – To find the sum of two numbers using function

```
#include<stdio.h>
void add(); // function declaration
int main() {
        add(); // function call and argument is not passed
        return 0; }
void add() //function definition. Return type is void, doesn’t return any value {
int a =10,b = 100, sum;
                sum = a + b;
        printf(”Sum = %d”,sum);
}
```

2.Function with no arguments and return value

- This type of functions, arguments are passed through the calling function to called function but the called function returns value.

- Function with return value means result will be sent back to the caller from the function.
- Such type of function are independent.

Example - Program to calculate the area of square using the function

```
#include <stdio.h>
int area(); //function prototype with return type int.
int main()
 {
 int square_area;
 square_area = area(); // function call
 printf("Area of square = %d", square_area);
 return 0;
 }
int area() // return type is int means it returns some value
 {
 int side, area;
 printf("\n Enter the side of the square :");
 scanf("%d", &sid);
 area = side * side;
 return area;
 }
```

3.Function with arguments and no return value

- In this, arguments are passed through the calling function to called function but does not return value.
- So it's a one-way type communication.
- Such type of function practically dependent on each other.

Example – Program to find the largest of two numbers using function

```
#include <stdio.h>

void greatNum(int x, int y); // Function prototype
int main()
{
            int num1, num2;
            printf("Enter two integers :");
            scanf("%d %d",&num1, &num2);
            greatNum(num1, num2); //Function called and argument is passed
            return 0;
 }
 // return type is void meaning it return no value
```

```
void greatNum(int x, int y)
{
                if(x > y){
               printf("The greater number is %d", x);
}
else
{
                printf("The greater number is %d", y);
 }
}
```

4.Function with arguments and a return value

- Both the calling function and called function will receive data from each other.
- This is the best type, as this makes the function completely independent of inputs and outputs, and only the logic is defined inside the function body.
- It’s like a dual communication.
- Both functions are dependent on each other.

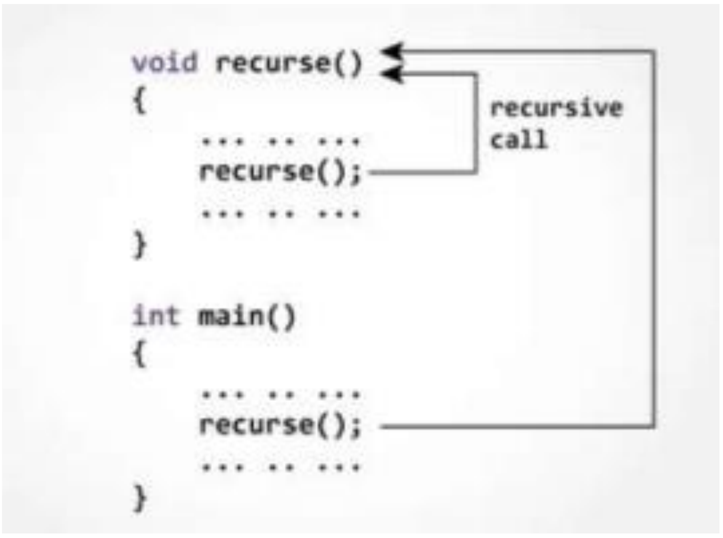
Example – Program to find the area of circle

```
#include <stdio.h>

int area(int radius); //function prototype with return type int

int main()

{

 int area, radius;

 printf("Enter the radius of circle :");

 scanf("%d",&radius);

 area = area(radius); //function call

 printf("Area of circle = %d", area);

 return 0;

}

int area(int r)

{

 int area;

 area = 3.14 * r * r;

 return area;

}
```

Recursive Function

- Recursion is the process of repeating items in a self-similar way.
- In programming languages, if a program allows to call a function inside the same function, then it is called a recursive call of the function.
- 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.
- ***A function that calls itself is known as a recursive function. And, this technique is known as recursion.***

Recursive Working



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

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

Example - Program to find the factorial using recursion

```
#include <stdio.h>
int factorial(int i)
{
        if(i <= 1) {
        return 1;
 }
 return i * factorial(i - 1);
}
int main()
 {
        int n;
       printf("\n Enter a number : ");
        scanf ("%d", &n);
       printf("Factorial of %d is %d\n", n, factorial(n));
       return 0;
}
```

1. Program to find the greatest of two numbers using functions.

2. Program to find the factorial of a number using functions.

3. Write a c program to find the sum of two integer numbers and two floating pint numbers using function.

4. Write a c program to find the area of a triangle, circle, square and rectangle using function.