

# Module 4: User defined function and Structures

# Syllabus

4.1	User Defined Functions: Need, Function Declaration and Definition, Return Values, Function Calls, Passing Arguments to a Function by Value, Recursive functions, String Handling Functions (inbuilt)
4.2	Structures and Unions: Introduction, Declaring and defining Structure, Structure Initialization, Accessing and Displaying Structure Members, Array of Structures
4.3	Introduction to pointers: Pointer declaration and initialization, Pointer addition and subtraction, evaluating pointer expressions Pointers and Functions: Pass by Reference, Returning pointers from functions

# Introduction

- A function is a block of code that performs a specific task.
- C allows you to define functions according to your need.
- These functions are known as user-defined functions.
- **Elements**
- Function Declaration
- Function Definition
- Function Call

# Function prototype

```
returnType functionName(type1 argument1,  
type2 argument2, ...);
```

A function prototype gives information to the compiler that the function may later be used in the program.

```
return_type function_name (argument list)  
{  
    Set of statements – Block of code  
}
```

# Function prototype

<b>return_type</b>	Return type can be of any data type such as int, double, char, void, short etc.
<b>function_name</b>	It can be anything, however it is advised to have a meaningful name for the functions so that it would be easy to understand the purpose of function just by seeing it's name.
<b>argument list</b>	Argument list contains variables names along with their data types. These arguments are kind of inputs for the function. For example – A function which is used to add two integer variables, will be having two integer argument
<b>Block of code</b>	Set of C statements, which will be executed whenever a call will be made to the function.

**What is the difference between Function  
declaration and definition?  
Where to be placed?  
How to call a function?**

# Function call

- The method of calling a function to achieve a specific task is called as function call.
- A function call is defined as function name followed by semicolon.
- A function call is nothing but invoking a function at the required place in the program to achieve a specific task.

Example

```
void main()
{
    add( ); // function call without parameter
}
```

# Formal Parameters

- The variables defined in the function header of function definition are called formal parameters.
- All the variables should be separately declared and each declaration must be separated by commas.
- The formal parameters receive the data from actual parameters.

## Actual Parameters

- The variables that are used when a function is invoked in function call are called actual parameters.
- Using actual parameters, the data can be transferred from calling function. to the called function.
- The corresponding formal parameters in the function definition receive them.
- The actual parameters and formal parameters must match in number and type of data

```
#include <stdio.h>

// Function declaration (prototype)
int add(int a, int b);

int main() {
    int result = add(5, 3); // Function call
    printf("The sum is: %d\n", result);
    return 0;
}

// Function definition
int add(int a, int b) {
    return a + b; // Returns the sum of a and b
}
```

# Example:2

```
#include <stdio.h>

int large(int num1, int num2) {
    if (num1 > num2)
        return num1;
    else
        return num2;
}

int main() {
    int x = 100, y = 90, z;
    z = large(x, y);
    printf("The largest number between %d and %d is %d\n", x, y, z);
    return 0;
}
```

# Example:2

```
#include <stdio.h>

void greeting()
{
    printf("Another beautiful day!\n");
}

int main() {
    greeting();
    return 0;
}
```

Function  
with no  
return  
value

# Types of functions

## Library Functions:

These are built-in functions provided by C, such as [printf\(\)](#), [scanf\(\)](#), [sqrt\(\)](#), and many others. You can use them by including the appropriate [header file](#), like #include <stdio.h> or #include <math.h>.

## User-Defined Functions:

## Recursive function

- Recursion is a method of solving the problem where the solution to a problem depends on solutions to smaller instances of the same problem.
- Recursive function is a function that calls itself during the execution.
- Consider Example for finding factorial of 5

Factorial(5)=n\*fact(n-1)

return 5 \* factorial(4) = 120

  └ return 4 \* factorial(3) = 24

    └ return 3 \* factorial(2) = 6

      └ return 2 \* factorial(1) = 2

       └ return 1 \* factorial(0) = 1

$$1 * 2 * 3 * 4 * 5 = 120$$

```
#include<stdio.h>
int fact(int n);
void main( )
{
    int num, result;
    printf("enter number:");
    scanf("%d", &num);
    result=fact(num);
    printf("The factorial of a number is: %d", result);
}

int fact(int n)
{
    if(n==0)
        return 1;
    else
        return (n*fact(n-1));
}
```

```
enter number:5
The factorial of a number is: 120
```

# C Program to Find G.C.D Using Recursion

```
#include <stdio.h>
int gcd(int n1, int n2);
int main()
{
    int n1, n2;
    printf("Enter two positive integers: ");
    scanf("%d %d", &n1, &n2);
    printf("G.C.D of %d and %d is %d.", n1, n2, gcd(n1, n2));
    return 0;
}

int gcd(int n1, int n2)
{
    if (n2 != 0)
        return gcd(n2, n1 % n2);
    else
        return n1;
}
```

# find the LCM of two numbers

```
#include<stdio.h>
#include<stdlib.h>
int gcd(int n1,int n2);

int main()
{
    int n1, n2;
    printf("Enter two positive integers: ");
    scanf("%d %d", &n1, &n2);
    printf("\nG.C.D of %d and %d is %d\n", n1, n2, gcd(n1, n2));
    printf("L.C.M of %d and %d is %d\n", n1, n2, ((n1*n2)/(gcd(n1, n2))));
    return 0;
}

int gcd(int n1, int n2) {
    if (n2 != 0)
        return gcd(n2, n1 % n2);
    else
        return n1;
}
```