

Module 4

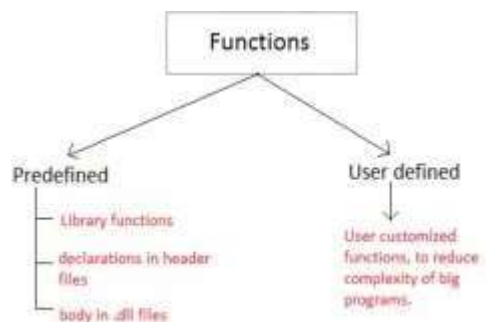
USER – DEFINED FUNCTIONS

INTRODUCTION:

A function is a block of code that performs a particular task.

C functions can be classified into two categories:

1. Library functions
2. User-defined functions



Library functions are those functions which are already defined in C library, example printf (), scanf (), strcat () etc. You just need to include appropriate header files to use these functions. These are already declared and defined in C libraries.

A User-defined functions on the other hand, are those functions which are defined by the user at the time of writing program. These functions are made for code reusability and for saving time and space.

Benefits of Using Functions

1. It provides modularity to your program's structure.
2. It makes your code reusable. You just have to call the function by its name to use it, wherever required.
3. In case of large programs with thousands of code lines, debugging and editing becomes easier if you use functions.
4. It makes the program more readable and easier to understand.

Elements of user-defined function in C

There are 3 key elements of UDF (User defined function)

1. Function Declaration (or Function Prototype)
2. Function Call
3. Function Definition

Function Declaration (or Function Prototype):

A function declaration is a frame (prototype) of function that contains the function's name, list of parameters and return type and ends with the semicolon, but it doesn't contain the function body. Function declaration is not a mandatory field in the program.

Syntax:

return_Type function_Name (formal parameter1, parameter2,.....);

Every function declaration must contain the following 3 parts and ends with the semicolon in C language

1. function name – name of the function is sum_Num ()
2. return type – the return type of the function is int
3. arguments – two arguments of type int are passed to the function

Example:

Example of the function declaration

int sum_Num (int x, int y,.....);

In the above example, int sum_Num (int x, int y); is a function declaration which contains the following information.

Function Call:

When a function is called, control of the program gets transferred to the function.

Syntax:


function_Name (Actual Argument list);

Example:

sum_Num (10,20);

When the function is called, the control flow of the program move to function definition and statements executes the inside body of the function

```
int main(){
.....
sum=sumNum(); //function call
.....
}
int sumNum() { //function definition
//function body
}
```

A diagram with a horizontal line above the function call 'sum=sumNum();' and another horizontal line below the function definition 'int sumNum() {'. A vertical line connects these two horizontal lines, with an arrow pointing from the upper line down to the lower line, indicating the transfer of control from the call site to the function definition.

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 – it contains the block of code for the special task.

The syntax of the function definition

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

1. function name – name of the function is sum_Num ()
2. return type – the return type of the function is int
3. arguments – two arguments of type int are passed to the function
4. body of function – specifies actions to be performed

return statements

Return statements terminate the execution of the function and return value to calling the function. Also, return statements control of the program control and moved to the calling function.

Syntax:

```
return (expression);
```

Example:

```
return (result);  
  
return (x+y);
```

Program Examples:

1. Write a C program to perform division of two numbers using functions.

```
#include <stdio.h>  
int division(int x,int y);           //function declaration or prototype  
void main()  
{  
    int a, b, div;  
    printf("Please enter 2 numbers for division\n");  
    scanf("%d%d",&a,&b);  
    div=division(a, b);              //function call  
    printf("The result of division is :%d",div);  
}
```

```
int division (int a, int b)
{
    int result;
    result=a/b;
    return result;           //return statements
}
```

2. Write a C program to add two numbers using function.

```
#include <stdio.h>
int addNumbers(int a, int b);    // function prototype
void main()
{
    int n1,n2,sum;
    printf("Enter two numbers: ");
    scanf("%d %d",&n1,&n2);
    sum = addNumbers(n1, n2);    // function call
    printf("sum = %d",sum);
}

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

3. Write C program to find SUM and AVERAGE of two integer Numbers using User Define Functions.

```
#include <stdio.h>
int sumTwoNum(int, int);        /*to get sum*/
float averageTwoNum(int, int);  /*to get average*/

void main()
{
    int number1, number2;
    int sum;
    float avg;

    printf("Enter the first integer number: ");
    scanf("%d", &number1);

    printf("Enter the second integer number: ");
    scanf("%d", &number2);
```

```
/*function calling*/
sum = sumTwoNum(number1, number2);
avg = averageTwoNum(number1, number2);

printf("Number1: %d, Number2: %d\n", number1, number2);
printf("Sum: %d, Average: %f\n", sum, avg);
}
int sumTwoNum(int x, int y)
{
    int sum; /*x and y are the formal parameters*/
    sum = x + y;
    return sum;
}
float averageTwoNum(int x, int y)
{
    float average; /*x and y are the formal parameters*/
    return ((float)(x) + (float)(y)) / 2;
}
```

4. Write C program to print Table of an Integer Number using User Define Function.

```
#include <stdio.h>
void printTable(int); /*function declaration*/
void main()
{
    int number;

    printf("Enter an integer number: ");
    scanf("%d", &number);

    printf("Table of %d is:\n", number);
    printTable(number);

    return 0;
}
void printTable(int num)
{
    int i;

    for (i = 1; i <= 10; i++)
        printf("%5d\n", (num * i));
}
```

Types of the user-defined function in C language:

User defined function in C can be categorized into four types:

1. function with no return value and without argument
2. function with no return value and with an argument
3. function with a return value and without argument
4. function with a return value and with an argument

Function with no return value and without argument

In this method, we won't pass any arguments to the function while defining, declaring, or calling it. 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.

```
#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);
}
```

Function with no return value and with an argument

In this category, function has some arguments. It receives data from the calling function, but it doesn't return a value to the calling function. The calling function doesn't receive any data from the called function. So, it is one way data communication between called and calling functions.

```
#include<stdio.h>
void Addition(int, int);
void main()
{
    int a, b;
    printf("\n Please Enter two integer values \n");
    scanf("%d %d",&a, &b);
    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);
}
```

Function with a return value and without argument

In this category, the functions have no arguments and it doesn't receive any data from the calling function, but it returns a value to the calling function. The calling function receives data from the called function. So, it is one way data communication between calling and called functions.

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

Function with a return value and with an argument

In this category, functions have some arguments and it receives data from the calling function. Similarly, it returns a value to the calling function. The calling function receives data from the called function. So, it is two-way data communication between calling and called functions.

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

```
int Multiplication(int a, int b)
{
    int Multi;
    Multi = a * b;
    return Multi;
}
```

Difference between call by value and call by reference

Call By Value	Call By Reference
While calling a function, we pass values of variables to it. Such functions are known as “Call by Values”.	While calling a function, instead of passing the values of variables, we pass address of variables (location of variables) to the function known as “Call by References
In this method, the value of each variable in calling function is copied into corresponding dummy variables of the called function.	In this method, the address of actual variables in the calling function are copied into the dummy variables of the called function.
With this method, the changes made to the dummy variables in the called function have no effect on the values of actual variables in the calling function.	With this method, using addresses we would have an access to the actual variables and hence we would be able to manipulate them.
In call by values, we cannot alter the values of actual variables through function calls.	In call by reference, we can alter the values of variables through function calls.
Values of variables are passes by Simple technique.	Pointer variables are necessary to define to store the address values of variables

Formal Parameter and Actual 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.

Functions and Arrays:

In C programming, you can pass an entire array to functions. Before we learn that, let's see how you can pass individual elements of an array to functions.

Pass Individual Array Elements

Passing array elements to a function is similar to passing variables to a function.

```
#include <stdio.h>
void display(int age1, int age2)
{
    printf("%d\n", age1);
    printf("%d\n", age2);
}
void main()
{
    int ageArray[] = {2, 8, 4, 12};
    display(ageArray[1], ageArray[2]);
}
```



Pass Arrays to Functions

Here we will be passing entire array to function.

Write a C program to calculate the sum, mean and standard deviation of an array elements using functions.

```
#include<stdio.h>
#include<math.h>
int sum(int a[], int n);
int sum1(int a[], int mean, int n);
void main()
{
    int i, n, a[100];
    float m, v, sd, sum, sum1;
    printf("Enter total number of elements\n");
    scanf("%d", &n);
    printf("Enter array elements\n");
    for(i=0;i<n;i++)
    {
        scanf("%d", &a[i]);
    }
    sum = sum(a, n);
    printf("Sum of the elements = %d\n", sum);
    m = sum/n;
    printf("Mean = %f\n", m);
    sum1 = sum1(a, m, n);
    v = sum1/n;
    sd = sqrt(v);
    printf("Standard deviation = %f\n", sd);
}
int sum(int a[], int n)
{
    int i, sum=0;
    for(i=0;i<n;i++)
    {
        sum = sum + a[i];
    }
    return sum;
}
```

```
int sum1(int a[], int m, int n)
{
    int sum1=0, i;
    for(i=0;i<n;i++)
    {
        sum1 = sum1 + pow((a[i] - m), 2);
    }
    return sum1;
}
```

Recursion

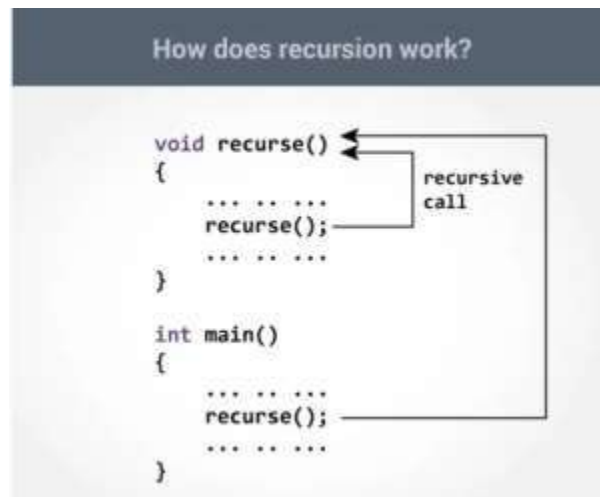
INTRODUCTION

In C, when a function calls a copy of itself then the process is known as Recursion. To put it short, when a function calls itself then this technique is known as Recursion. And the function is known as a recursive function.

Syntax:

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

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



Example: Write a C program to find the Sum of Natural Numbers Using Recursion

```
#include <stdio.h>
int sum(int n);
void main()
{
    int number, result;
    printf("Enter a positive integer: ");
    scanf("%d", &number);
    result = sum(number);
    printf("sum = %d", result);
}
int sum(int n)
{
    if (n != 0)
        return n + sum(n-1);
    else
        return n;
}
```

Write a C program to generate Fibonacci series

```
#include<stdio.h>
int Fibonacci(int);
void main()
{
    int n, i = 0, c;
    printf("Enter n value\n");
```

```
scanf("%d",&n);

printf("Fibonacci series\n");

for ( c = 1 ; c <= n ; c++ )
{
    printf("%d\n", Fibonacci(i));
    i++;
}
}
int Fibonacci(int n)
{
    if ( n == 0 )
        return 0;
    else if ( n == 1 )
        return 1;
    else
        return ( Fibonacci(n-1) + Fibonacci(n-2) );
}
```

Write a C program to find the factorial of a number

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