

A function is a self-contained program segment that carries out some specific well-defined tasks or a group of statements that together perform a task. In C/C++, we can construct several functions according to our requirement. The user-defined functions are defined by a user as per its own requirement and library functions come with compiler. Library functions are also called Predefined functions.

### Advantages of functions:

- 1) **Program development made easy:** Work can be divided among project members thus implementation can be completed in parallel.
- 2) **Program testing becomes easy :** Easy to locate and isolate a faulty function for further investigation
- 3) **Code sharing becomes possible:** A function may be used later by many other programs this means that a C/C++ programmer can use function written by others, instead of starting over from scratch.
- 4) **Code re-usability increases:** A function can be used to keep away from rewriting the same block of codes which we are going to use two or more locations in a program. This is especially useful if the code involved is long or complicated.
- 5) **Avoid un-necessary repetition of code:** suppose you have a program that calculates square root of a number, later if you want to calculate square root of another number, you don't want to write same instructions all over again. Simply make a function that calculates square root & call it whenever need it.
- 6) **Increases program readability:** It makes possible top down modular programming. In this style of programming, the high level logic of the overall problem is solved first while the details of each lower level functions is addressed later. The length of the source program can be reduced by using functions at appropriate places.
- 7) **Function facilitates procedural abstraction:** Once a function is written, it serves as a black box. All that a programmer would have to know to invoke a function would be to know its name, and the parameters that it expects
- 8) **Functions facilitate the factoring of code:** Every C/C++ program consists of one **main( )** function typically invoking other functions, each having a well-defined functionality.

### Syntax:

```
Return-type Name-of-Function (argument(s))
{
    Function_body ;
}
```

- ❖ **Return Type** – A function may return a value. The **return type** is the data type of the value the function returns. 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.
- ❖ **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.
- ❖ **Function Body** – the function body contains a collection of statements that define what the function does.

**Structure of a Function:** Each function is comprises of three following parts.

1) **Function Declaration /Prototype**

Function prototype tells compiler that we are going to use a function which will have given name, return type and parameters. It is **ended** with the **semicolon**.

OR The function declaration (or prototype, the terms means the same thong) tells the compiler the name of the function, the data type of the function returns (if any) and the number and data types of the function's arguments (if any)

2) **Function Call:**

This causes the transfer of control to the code in the definition of function. Or compiler jumps to that function. It is **ended** with the **semicolon**.

3) **Function Definition:**

Function definition on the other hand is just writing logic of any function. For example you have one function prototype in C/C++ program for adding two integer numbers (i.e. `int add(int, int)`) but along with prototype you also have to write logic how that function will behave (respect to above prototype; how you will utilize those two passed integer value and how you will return value) when you will call that function.

**Types of functions:**

A function may belong to any one of the following categories:

- 1) Functions with no arguments and no return values.
- 2) Functions with arguments and no return values.
- 3) Functions with arguments and return values.
- 4) Functions with no arguments and return values.

**1) Functions with no arguments and no return value.**

A C/C++ function without any arguments means you cannot pass data (values like `int`, `char` etc) to the called function. Similarly, function with no return type does not pass back data to the **calling function**. It is one of the simplest types of function in C/C++. This type of function which does not return any value cannot be used in an expression it can be used only as independent statement.

**Example**

```

#include<stdio.h>
#include<conio.h>
void line(void);
void main() // function prototype ended with semicolon above void main()
{
    line(); // function call ended with semicolon
    printf("\n Computer System\n ");
    line();
} // main ends here
void line(void) // function definition
{ // function body
    for(int i=0; i<=20; i++)
        printf("*");
}

```

**2) Functions with arguments and no return value.**

A C/C++ function with arguments can perform much better than previous function type. This type of function can accept data from calling function. In other words, you send data to the called function from calling function but you cannot send result data back to the calling function. Rather, it displays the result on the terminal. But we can control the output of function by providing various values as arguments.

**Example**

```

void add(int,int); // function prototype ,It has 2 integer arguments
void main void()
{
    int x,y;
    printf("enter value of x"); scanf("%d",&x);
    printf("enter value of y"); scanf("%d",&y);
    add(x,y); // function call
} // main ends here

void add(int x, int y) //function definition
{
    int sum;
    sum = x+y;
    printf("the sum is %d",sum);
}

```

### 3. Functions with arguments and return value.

This type of function can send arguments (data) from the calling function to the called function and wait **for** the result to be returned back from the called function back to the calling function. And this type of function is mostly used in programming world because it can do two way communications; it can accept data as arguments as well as can send back data as return value. The data returned by the function can be used later in our program **for** further calculations.

#### Example

```
int add(int x, int y); //function prototype ,function has 2 arguments and return an integer value
void main( )
{
    clrscr();
    int x=30, y=20 , z;

    z = add(x,y); //function call ,the return value of function will stored to z

    printf("Result %d.\n\n",z); // result of function will display here

    getch();
}

int add(int x, int y) // function definition
{
    int result; result = x+y;
    return(result);
}
```

### 4. Functions with no arguments but returns value.

This type of function which does not take any argument but only returns values to the calling function

## Example

```

#include<stdio.h>
#include<conio.h>
int send(); //function prototype ,function has NO arguments but return an integer value
void main()
{
    clrscr();
    int z;
    z = send(); //function call ,the return value of function will stored to z

    printf("\n You entered : %d.", z); // result of function will display here
    getch();
}

int send() // function definition
{
    int no1;
    printf("Enter a no : ");
    scanf("%d",&no1);
    return(no1);
}

```

## Recursion:

A function that calls itself again and again until some test pattern remain true , is called “Recursive “ function and this process is known as “Recursion”.

## Example

```

#include<stdio.h>
#include<conio.h>

int add (int i)
{
    int n;
    if (i==0)
        return 0;
    else
        n = i+ add(i-1);
    return n;
}

void main()
{
    int x;
    clrscr();
    printf(" Enter any Number: ");scanf(" %d " ,&x);
    printf("\n Sum : %d", add(x));
}

```

**Explanation:**

User input "x" passes to "add", Function will check either it is zero or not, **if** yes, then it will return "0" to main, otherwise it will add integer values until it becomes zero.

**Advantages:**

- It is easy to use.
- It represents compact programming structures.

**Disadvantage:**

- It is slower than that of looping statements because each time function is called.

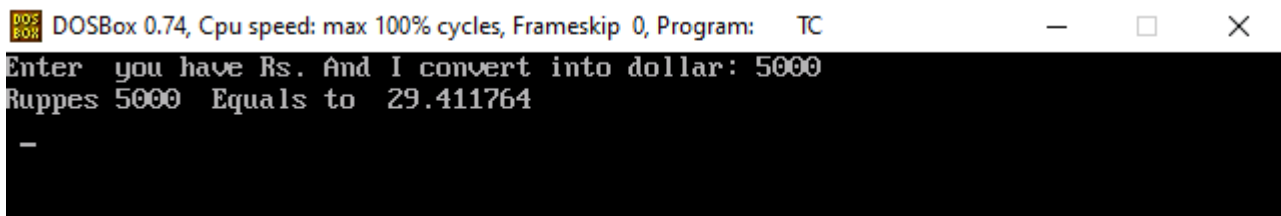
**Points to Remember**

- There should be at least one if statement used to terminate recursion.
- It does not contain any looping statements.

**Local and Global Variables**

Local variable is declared inside a function whereas Global variable is declared outside the function. Local variables are created when the function has started execution and is lost when the function terminates, on the other hand, Global variable is created as execution starts and is lost when the program ends.

```
#include <conio.h>
#include <stdio.h>
int dollar =170;    //Global variable dollar
RupeesInToDollar(int rs) //function prototype declaration with definition
{
    int LocalDollar; // Local variable Localdollar
    LocalDollar = (float)rs / (float)dollar;
    printf("Rupes %d Equals to %f \n ",rs, LocalDollar);
}
void main()
{
    int rs;          // it is local variable
    printf("Enter you have Rs. And I convert into dollar: ");
    scanf("%d",&rs);
    RupeesInToDollar(rs);
}
```



```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter you have Rs. And I convert into dollar: 5000
Rupes 5000 Equals to 29.411764
```

## Exercise

## Theory Questions

1. Write four advantages of using functions.
2. Explain why some functions do not have a **return** statement.
3. Explain the difference between a local variable and global variable.
4. Describe the recursion?
5. What is the difference between the user-defined and pre-defined functions in C/C++.

## Practical Questions

1. Write a program to display your complete profile, using at least 3 functions.  
**Hint:** If user press 1: your bio-data will display onto the screen  
If user press 2: your academic profile will display onto screen
2. Write a program to find;  
**a)** Surface area ( $A=4\pi r^2$ ) **b)** Area of rectangle ( $A= \text{width} \times \text{Length}$ ).
3. Write a function to calculate if a number is prime, Return 1 if it is prime and 0 if it is not a prime.
4. Write a program to calculate factorial of a number input by the user using Recursion.
5. Make a calculator using functions **sum( )**, **sub( )**, **mul( )**, **div( )** for calculations.
6. Write a function **pow(x,y)** to calculate the value of x raised to y.

## Objective MCQ's

1. A variable that is declared outside a function is called a \_\_\_\_ variable.  
a) Local  
b) Global  
c) Program  
d) Class
2. A local variable must be declared \_\_\_\_\_.  
a) Before a function  
b) After a function  
c) Within the braces of a function definition.  
d) With the local keyword.
3. A (n) \_\_\_\_\_ allows you to treat a related group of C/C++ commands as a single unit.  
a) Statement  
b) Variable  
c) Event  
d) Function
4. When creating a user-defined function, you should use \_\_\_\_\_ with the function name.  
a) function  
b) function return type and function parameters (if any)  
c) procedure  
d) nothing all

5. The function calls itself this process is called \_\_\_\_\_.
- Repetitive
  - Decision
  - Recursion
  - loop
6. A program module in C/C++ is called \_\_\_\_\_.
- Variable
  - Function
  - Sub-routine
  - Event
7. Function is invoked with a \_\_\_\_\_.
- Return to function.
  - Function Definition.
  - Function Call.
  - Function Prototype.
8. A global variable is defined in a declaration.
- In main () only
  - In the first function that uses it
  - Outside the any function
  - Outside the main function.
9. A variable that is known only with in the function in which it is defined is called a\_\_\_\_\_.
- Global variable
  - Function
  - Local variable
  - Variable scope
10. Which of these are valid reasons for using functions?
- They use less memory than repeating the same code.
  - They run faster.
  - They keep different program activities separate.
  - They keep variables safe from other parts of the program.