Functions in C Language

FUNCTIONS

- Modules in C Language are called **functions**.
- Function is a named block of code that performs some action.
- Each function has a unique name that encapsulate piece of code.
- Functions provide structured programming approach or Modular Programming.
- Functions are used to perform similar kind of task again and again.
- The statements written in a function are executed when it is called by its name.

ADVANTAGES OF FUNCTIONS

- **Easier to code**
- **Easier to modify programs**
- **Easier to Debug**
- **Reusability**
- **Less Programming Time**

TYPES OF FUNCTIONS

1. **Built-in Functions or Library Functions**
2. **User Defined Functions**

1) BUILT-IN FUNCTIONS :

- A Type of Function that is Part of Language is called Built-in Functions or Library Functions
- These functions are ready made and stored in different header files.
- Built-in functions make programming easier and faster.
- **Example:**
  - scanf( ) , printf( ) are two examples

2) USER DEFINED FUNCTIONS

- A type of function written by the programmer is known as user-defined function
- A program may contain many user-defined functions
- These functions are written according to the exact need of the user
- **Example:**
  - factorial( ) , table( ) are two examples

USER-DEFINED FUNCTIONS CONSISTS OF

1. **Function Declaration**
2. **function Definition**
3. **Function Call**

1) FUNCTION DECLARATION /FUNCTION PROTOTYPE /FUNCTIONS SIGNATURE

- Function declaration is a model of a function, also known as function prototype
- It provides information to compiler about the structure of the function,
- It consist of function return type ,function name and types of parameter

Syntax:	**Example:**
return-type function-name(parameters);	int add(int a, int b);

2) WHAT IS FUNCTION DEFINITION

- A set of statements that explains what a function does is called function definition
- Function definition consists of function header and function body.
- Function definition can be written before main(), after main() or in a separate file.

Syntax:

```
return-type function-name(parameters)
{
  Function-body;
}
```

❖ WHAT IS FUNCTION HEADER

- First line of function definition is known as function header
- It consist of function return type ,function name and types of parameter

❖ WHAT IS FUNCTION BODY

- Set of statements which are executed inside the function is known as function body.
- These statements are written in curly braces { }
- All logic is implemented in the body of a function.
- Arguments passed to function are also used in function body.

3) WHAT IS FUNCTION CALL OR HOW A FUNCTION IS ACTIVATED

- Functions are **invoked** by a **function call**
- A statement that activates or Triggers a function is known as function call.
- A function is called with its name.
- A function name is followed by necessary parameters in parentheses.
- Empty parentheses are used if there is no parameter, if there are many parameters these are separated by comma's.

Program-1:

```
/* A simple Program that shows Hello World using function */
#include <stdio.h>

void show();          // function declaration

int main ()
{
    show();           //function call
 return 0;
}

void show()
{
    printf("Hello World");

}
```

Output of program-1:

```
Hello World
------------------
```

PARAMETERS

- Parameters are the values that are provided a function when it is called
- Parameters are given in parentheses ()
- Empty parentheses are used if there is no parameter, if there are many parameters these are separated by comma's.
- Parameters are of two types : Actual and Formal

Actual parameter

- Parameter used in function call are called actual parameter.
- These parameters are used to send values to function
- If more than one actual parameters are used, each parameter is separated by comma.

Formal parameter

- Parameters in function header are called formal parameters
- These parameters are used to receive values from the calling function
- If two are more formal parameters are used , each parameter is separated by comma

1. Pass By Value
2. Pass By Reference

1) Passing Parameters/Argument By Value:

A parameter passing mechanism in which the value of **actual parameter** is copied to **formal parameters** is known as **pass by value.** If the function makes any change in formal parameter, it does not affect the values of actual parameters .It is the default method of passing parameters to function.

Separate memory is reserved for actual and formal parameters.

Program-2:

```c
/* This program input Two values in main() function
 Pass these numbers to a function .
 The function displays the  Maximum number  */

#include <stdio.h>
void max(int,int);
int main ()
{
    int x,y;
    printf("Enter 1st Value::");
    scanf("%d",&x);
    printf("Enter 2nd Value::");
    scanf("%d",&y);
    max(x,y);

 return 0;
}

void max(int a,int b)
{
    if(a>b)
        printf("%d is Maximum",a);
    else
        printf("%d is Maximum",b);
}
```

Output of program-2:

```
Enter 1st Value::8
Enter 2nd Value::4
8 is Maximum
------------------------
```

****

Program-3:

```c
/* This program input a number in main() function
 Pass the number  to a function .
 The function displays the  Table of that number  '

#include <stdio.h>
void table(int);
int main ()
{
    int x;
    printf("Enter  Value::");
    scanf("%d",&x);
    table(x);
 return 0;
}

void table(int n)
{
    int c;
    for(c=1;c<=10;c++)
    printf("%d * %d = %d\n",n,c,c*n);
}
```

Program-4:

```c
/* This program input a number in main() function
 Pass the number  to a function .
 The function displays the  Factorial of that number  */
#include <stdio.h>
void factorial(int);
int main ()
{

    int x;
    printf("Enter  Value::");
    scanf("%d",&x);
    factorial(x);
}

void factorial(int n)
{
    int f,c;
    f=1;
    for(c=1;c<=n;c++)
    f=f*c;
    printf("Factorial of %d is %d",n,f);
}
```

****

Program-5:

Write a Program that inputs a number from the user in main() function and find is it Perfect number or Not using function.

```c
#include <stdio.h>
void perfect(int);
int main()
{
    int num;
    printf("Enter a Number: ");
    scanf("%d", &num);
    perfect(num);
}
void perfect(int n)
{
    int sum = 0;
    for (int i = 1; i <= n/2; i++)
    {
        if (n%i == 0)
        {
            sum = sum + i;
        }
    }
    if (sum == n)
        printf("Entered Number is perfect number");
    else
        printf("Entered Number is not a perfect number");
}
```

Output of program-5:

```
Enter a Number: 6
Entered Number is perfect number
--------------------------------
```

2) Passing Parameters/Arguments By Reference

- a parameter passing mechanism in which the address of **actual parameter** is passed to the called function
- **formal parameters** are not created separately in memory
- Formal parameter refers the same memory occupied by actual parameter.

When an argument is passed by reference, the caller allows the called function to modify the original variable's value.

Program-6:

```
/* this program input a number in main() function
and pass this number to cube function by
Passing Argument by reference mechanism.
that calculate the cube.
 */

#include<stdio.h>
void Cube(int*);
int main()
{
    int x;
    printf("Enter Any Number :");
    scanf("%d",&x);
    Cube(&x);
}
void Cube(int *n)
{
    *n=*n * *n * *n;
    printf("Cube is :%d",*n);
}
```

Output of program-6:

```
Enter Any Number :5
Cube is :125
------------------------
```

****

Program-7:

```
/* this program swap two variables by using
Passing Argument by reference mechanism.
 */

#include<stdio.h>
void swap(int* ,int*);
int main()
{
    int x,y;
    printf("Enter Value for x::");
    scanf("%d",&x);
    printf("Enter Value for y::");
    scanf("%d",&y);
    swap(&x,&y);
    printf("AFTER SWAP\n");
    printf("Value of x:: %d\n",x);
    printf("Value of y:: %d",y);

}
void swap(int *m ,int *n)
{
    int temp;
    temp= *m;
    *m=*n;
    *n=temp;
}
```

Output of program-7:

```
Enter Value for x::6
Enter Value for y::7
AFTER SWAP
Value of x:: 7
Value of y:: 6
------------------------
```

RETURNING A VALUE FROM A FUNCTION

- A function can return only single value
- return type in function declaration indicates type of value returned by function
- **return** keyword is used to return a value to a calling function.

Example :

```
return;          // If the function does not return a result
return exp;      // returns the value of expression to the caller.
```

Program-8:

```
/* This program input marks in main() function and pass
to the grade() function that calculate and
return the grade to main()    */

#include<stdio.h>
char grade(int score);
int main()
{
    int m;
    char g;
    printf("your Marks::");
    scanf("%d",&m);
    g=grade(m);
    printf("Your grade is %c",g);
}

char grade(int score)
{
    if(score>=90)
    return 'A';
    else if (score>=80)
    return 'B';
    else if (score>=70)
    return 'C';
    else if (score>=60)
    return 'D';
    else
    return 'F';
}
```

Output of program-8:

```
your Marks::70
Your grade is C
------------------
```

****

Program-9:

```
/* This program input a number in main() function and pass
to the factorial() function that calculate and
return the factorial to main()    */
#include <stdio.h>
int factorial(int n);
int main ()
{
    int num;
    long int fact;
    printf("Enter  Value::");
    scanf("%d",&num);
    fact=factorial(num);
    printf("Factorial of %d is %d",num,fact);
}
int factorial(int n)
{
    long int f;
    f=1;
    for(int c=1;c<=n;c++)
    f=f*c;
    return f;
}
```

LOCAL AND GLOBAL VARIABLES

```c
/* this program demonstrate the
working of global variable       */
#include<stdio.h>

void show(void);
int n=0;

int main()
{
    n++;
    show();
    printf("Value of n is : %d",n);
}
void show(void)
{
n++;
}
```

Output of program-10:

```
Value of n is : 2
------------------------
```

FUNCTIONS AND ARRAY

Program-11:

```c
/* this program input 5 integer values in an array
   pass this array and its size to max function.
    function then find the maximum value
    and return to main function
------------------------------------------------ */
#include<stdio.h>
int max(int bs[],int s);  // function declaration

int main()
{
    int arr[5];
    int i,m;
    for(i=0;i<5;i++)
    {
        printf("Enter number::");
        scanf("%d",&arr[i]);
    }
    m=max(arr,5); // calling max function
    printf("Maximum value is :: %d",m);
}

int max(int bs[],int s)
{
    int i,mx;
    mx=bs[0];
    for(i=1;i<s;i++)
        if (mx<bs[i])
        mx=bs[i];

    return mx ;
}
```

Output of program-11:

```
Enter number::4
Enter number::6
Enter number::3
Enter number::8
Enter number::5
Maximum value is :: 8
------------------------
```

Passing array to function using call by reference

When we pass the address of an array while calling a function, this is call by reference. When we pass an address as an argument, the function declaration should have a pointer as a parameter to receive the passed address.

Program-12:

write a program input 5 integer values in an array pass the array to sort() function by refrence ,that sort them using ->Selection sort. main() function then print the sorted array

```
void sort(int*);
#include<stdio.h>
int main()
{
        int arr[5];
        int i;
        for(i=0;i<5;i++)
    {
        printf("Enter number: ");
        scanf("%d",&arr[i]);
    }
    sort(arr);

    printf("Sorted values::\t");
    for(i=0;i<5;i++)
    {
        printf("%d\t",arr[i]);
    }
}
void sort(int *arr)
{

    int i,j,min,temp;

    for (i=0;i<4;i++)
    {
        // Find the minimum element in unsorted array
        min=i;
        for(j=i+1;j<5;j++)
            if(arr[j]<arr[min]) /* check the element to be minimum */
                min=j;
        if(min!=i)
        {
            temp=arr[i];
            arr[i]=arr[min];
            arr[min]=temp;
        }

    } // end of outer for loop
}
```

Output of program-12:

```
Enter number: 9
Enter number: 8
Enter number: 7
Enter number: 6
Enter number: 5
Sorted values:: 5       6       7       8       9
--------------------------------
```

Program-13:

write a program input 5 integer values in an array pass the array to dbl() function by refrence ,that double the values .main() function then print the values.

```
#include<stdio.h>
void dbl(int *ptr);
int main()
{
        int arr[5];
        int i;
        for(i=0;i<5;i++)
    {
        printf("Enter number: ");
        scanf("%d",&arr[i]);
    }
    dbl(arr);

    printf("values in double::");
    for(i=0;i<5;i++)
    {
        printf("%d\t",arr[i]);
    }
}
void dbl(int *ptr)
{
        for (int i=0;i<5;i++)
        {
*ptr=*ptr * 2;
        ptr++;
        }
}
```

Output of program-13:

```
Enter number: 4
Enter number: 6
Enter number: 5
Enter number: 3
Enter number: 7
values in double::8     12      10      6       14
--------------------------------
```

Program-14:

write a program input 5 integer values in an array pass one value each time to evenodd() function that display its status.

```
#include<stdio.h>
void evenodd(int n);
int main()
{
        int arr[5];
        int i;
        for(i=0;i<5;i++)
    {
        printf("Enter number: ");
        scanf("%d",&arr[i]);
    }
    for(i=0;i<5;i++)
    {
        evenodd(arr[i]); //passing single value to function
    }
}
void evenodd(int n)
{
        if (n%2==0)
        printf("%d is even\n",n);
        else
        printf("%d is odd\n",n);
}
```

Program-15:

```c
/* write a program that declares a structure to store Roll no ,
 Marks and GPA of a student . the main() define a
 structure variable, input the values and pass to function
 that display these values */

#include<stdio.h>
struct Student
{
    int rno;
    int marks;
    float gpa;
};
void output(Student st);
int main()
{
    struct Student s;   //defining a variable s of type student
    printf("Enter your Roll No:");
    scanf("%d",&s.rno);
    printf("Enter your Marks:");
    scanf("%d",&s.marks);
    printf("Enter your GPA:");
    scanf("%f",&s.gpa);
    output(s);
}
void output(Student st)
{
    printf("\n-----------------------\n");
    printf("Your Roll Is :%d\n",st.rno);
    printf("Your Marks are :%d\n",st.marks);
    printf("Your GPA :%.2f\n",st.gpa);

}
```

****

```c
/* write a program that declares a structure to store Roll no ,
 Marks and GPA of a student .
 and  write main(),input(),output() function   */
#include<stdio.h>
struct Student
{
    int rno;
    int marks;
    float avg;
};
void input(Student*);
void output(Student*);
int main()
{
    struct Student s;   //defining a variable s of type student
    input(&s);
    output(&s);
}
void input(Student *x)
{
    printf("Enter your Roll No:");
    scanf("%d",&x->rno);
    printf("Enter your Marks:");
    scanf("%d",&x->marks);
    printf("Enter your GPA:");
    scanf("%f",&x->avg);
}
void output(Student *y)
{
    printf("\n-----------------------\n");
    printf("Your Roll Is :%d\n",y->rno);
    printf("Your Marks are :%d\n",y->marks);
    printf("Your GPA is :%f\n",y->avg);
}
```

2. **What is function**
   - Function is a named block of code that performs some action. The statements written in a function are executed when it is called by its name.
   - Each function has a unique name that encapsulate piece of code.
   - Functions provide structured programming approach.
3. **What is importance of using functions? OR Why we use functions in a program.**
   - Functions provide modular way of writing programs in C.
   - Functions divide a program into different parts , and each part implement different functionality
   - A piece of code that is executed repeatedly is stored in a separate function.
4. **List any four benefits of using functions in C.**
   - Easier to code
   - Easier to modify a program
   - Reusability
   - Less programming time
5. **List different types of functions in C**
   - C language provide two types of functions
     - ❖ User-defined function
     - ❖ Built-in function
6. **What is user defined function ,give example**
   - A type of function written by the programmer is known as user-defined function
   - A program may contain many user-defined functions
   - These functions are written according to the exact need of the user
   - **Example:**
     - ❖ factorial( ) , table( ) are two examples
7. **What is built-in function ,give example**
   - A type of function that is part of language is called **built-in function or library function**
   - These functions are ready made and stored in different header files.
   - Built-in functions make programming easier and faster.
   - **Example:**
     - ❖ scanf( ) , printf( ) are two examples
8. **What is function prototype /functions signature / function declaration**
   - Function declaration is a model of a function, also known as function prototype
   - It provides information to compiler about the structure of the function,
   - It consist of function return type ,function name and types of parameter

Syntax:	**Example:**
return-type function-name(parameters);	int add(int a, int b);

9. **What is function definition**
   - A set of statements that explains what a function does is called function definition
   - Function definition consists of function header and function body.
   - Function definition can be written before main(), after main() or in a separate file.

Syntax:

```
return-type function-name(parameters)
{
  Function-body;
}
```

10. **What is function header**
    - First line of function definition is known as function header
    - It consist of function return type ,function name and types of parameter
11. **What is function body**
    - Set of statements which are executed inside the function is known as function body.
    - These statements are written in curly braces { }
    - All logic is implemented in the body of a function.
    - Arguments passed to function are also used in function body.

12. What is function call *OR* how a function is activated

- A statement that activates a function is known as function call. A function is called with its name.
- A function name is followed by necessary parameters in parentheses .
- Empty parentheses are used if there is no parameter, if there are many parameters these are separated by comma's.

13. Define parameters.

- Parameters are the values that are provided a a function when it is called
- Parameters are given in parentheses ()
- Empty parentheses are used if there is no parameter, if there are many parameters these are separated by comma's.
- Parameters are of two types : actual and formal

14. What is use of actual parameter

- Parameter used in function call are called actual parameter .
- These parameters are used to send values to function
- If more than one actual parameters are used, each parameter is separated by comma.

15. What is use of formal parameter

- Parameters in function declaration are called formal parameters
- These parameters are used to receive values from the calling function
- If two are more formal parameters are used , each parameter is separated by comma

16. How a function return a value

- A function can return only single value
- return type in function declaration indicates type of value returned by function
- **return** keyword is used to return a value to a calling function.

17. What is local variable

- A variable declared inside a function is known as local variable
- Local variables are called also called **automatic variable**

18. What is life time of variable

- The time period for which a variable exist in a memory is called **lifetime of a variable**
- Lifetime of a local variable starts when control enters the function
- Lifetime of a local variable ends when control moves back to the calling function

19. What is scope of local variable

- Scope is the area in which variable can be accessed
- Local variable can be used in the function in which it is declared
- If a local variable is used outside a function compiler generate error about its scope

20. What is global variable

- A variable declared outside of any function is known as global variable.
- Global variable is used by all functions in a program
- The value of these variables is shared among different functions

21. What is life time of a global variable OR How long global variable exit in a memory?

- Global variables exist in the memory as long as the program is running
- These variables are destroyed from memory when the program terminates
- These variables occupy memory longer than local variable

22. What is scope of global variable

- Global variables can be used by all functions in the program
- These variables are globally access from any part of the program
- Normally global variables are declared before the main function

23. what is pass by value mechanism for parameter passing

- a parameter passing mechanism in which the value of **actual parameter** is copied to **formal parameters** is known as **pass by value**
- it is the default method of passing parameters to function

24. what is pass by reference mechanism for parameter passing

- a parameter passing mechanism in which the address of **actual parameter** is passed to the called function
- **formal parameters** are not created separately in memory
- Formal parameter refers the same memory occupied by actual parameter.

- A function that calls itself is known as a recursive function.
- A **recursive function** *calls itself* either directly or indirectly through another function.

Calculating Factorial

The factorial of a nonnegative integer n, written $n!$ (pronounced "n factorial"), is the product

$$n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 1$$

with 1! equal to 1, and 0! defined to be 1. For example, 5! is the product 5 * 4 * 3 * 2 * 1, which is equal to 120.

Recursively Calculating Factorials

Recrsive Function Factorial

$$fact(0) = 1$$

$$fact(n) = n*fact(n-1) \ \ | (n > 0)$$

Program-17:

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

Output of program-17:

```
Enter number : 6
Factorial of 6 is 720
--------------------------------
```

Fibonacci Series

Default

0 1 1 2 3 5

0 + 1 = 1
1 + 1 = 2
1 + 2 = 3
2 + 3 = 5

Recursively Calculating Fibonacci

$$f(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ f(n-1) + f(n-2) & n > 1 \end{cases}$$

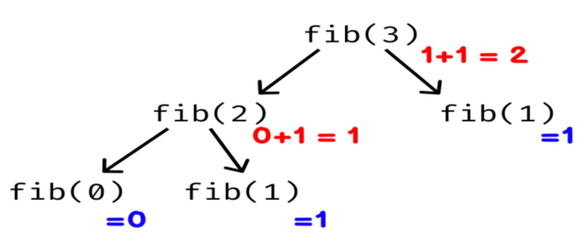
Program-17:

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

Output of program-17:

```
How many terms:5
0       1       1       2       3
--------------------------------
```

RECURSIVE FIBONACCI FUNCTION



The **fib()** function will recursively call itself until it reaches the base case

Program-18:

```c
#include<stdio.h>
void DecToBinary(int);
int main()
{
    int n;
    printf("Enter number: ");
    scanf("%d",&n);
    DecToBinary(n);
}

void DecToBinary(int m)
{
    if(m > 0){
    DecToBinary(m/2);
    printf("%d",m%2);
    }
}
```

Output of program-18:

```
Enter number: 14
1110
------------------------
```