FUNCTIONS

- Concept of function, using functions, call by value and call by reference mechanism to working with functions-example programs, passing arrays to functions, scope and extent, storage classes, recursion.

User Defined Functions

Even though many functions are provided by C compiler, these functions are not enough to perform customized functions. The user can construct their own functions to perform some specific task. This type of functions created by the user is termed as User defined functions.

Function Structure

The function structure consists of two parts

1. Function Header
2. Function Body

1. Function header

It consists of three parts

Syntax: **return_type function_name(parameter or without parameters);**

- The return value of the function
- The name of the function
- The parameters of the functions may be included or excluded

Ex: void add(int a, int b);

In the above example

- ❖ The return type of the function is void,
- ❖ The name of the function is add,
- ❖ The parameters are 'a' and 'b' of type integer.

2. Function body

The function body consists of the set of instructions enclosed between { and }.

Generally while representing the function we can include the functions in the program as

- ❖ Function Call
- ❖ Function Definitions
- ❖ Function Declaration

✓ **Function Call**

A function call also termed as signature of the function, it is the statement which calls the function. It is used in the main program. A function call format is function name followed by semicolon. Like ***add(a,b);***

Ex:

```
void main()
{
        add(a,b);
}
```

Note:

A function call is nothing but invoking a function at the required place in the program to achieve a specific task.

✓ **Function Definition or Function Body**

A function definition is the body of the function which would consist of series of instructions embedded within the flower braces after the function name.

Ex:

```
void add( int a, int b)
    {
      // instructions
    }
```

✓ **Function Declaration**

Function declaration also termed as function prototyping consists of the return type of function, name of the function and parameter list ending with semicolon.

Ex: void add(int a, int b);

Note:

The function declaration should end with a semicolon

✓ Function prototyping is not compulsory, if the definition of the function appears before the call to the function. Generally the function call will be inserted in the main function, if the function

definition is present before the coding of main function, then no need of function prototyping. Otherwise it is must to specify the function prototyping to make the function code reachable.

Example:

Function to add two numbers

- **Without prototyping**

```
#include<stdio.h>
#include<conio.h>
void add(int a, int b) // function definition
{
        int x;
        x=a+b;
        printf("\n the sum is %d", x);
}
void main()
{
int a, b;
printf("\n enter the two numbers a and b");
scanf("%d%d", &a, &b);
add(a,b);  // function call
getch();
}
```

- **With prototyping**

```
#include<stdio.h>
#include<conio.h>
void add(int a, int b); // function prototyping
void main()
{
int a, b;
printf("\n enter the two numbers a and b");
scanf("%d%d", &a, &b);
add(a,b);  // function call
getch();
}
void add(int a, int b) // function definition
{
        int x;
        x=a+b;
        printf("\n the sum is %d", x);
}
```

Note:

The function prototyping can be specified in different ways if parameters are passed

1. Only the data type can be specified instead of specifying the variable name with data type.

   Ex: ***void add(int a, int b);*** can also be written as ***void add(int, int);***

2. If arrays are used as parameters then we have to specify the ranging values.

   Ex: ***void add(int a[ ], int b[ ]);*** can be specified as ***void add(int [ ], int [ ]);***

Calling Function and Called Function

A function that is invoked or called when the program is being executed is called ***"called function"*** and the invoking function is called ***"calling function".***

Ex:

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

void add(int a, int b); // function prototyping

// main function is the calling function
void main()
{
        int a, b;
        printf("\n enter the two numbers a and b");
        scanf("%d%d", &a, &b);
        add(a,b);  // function call
        getch();
}
// add is the called function
void add(int a, int b) // function definition
{
        int x;
        x=a+b;
        printf("\n the sum is %d", x);
}
```

- ✓ In the above program we are calling a function **add()** from the main function, that means we are invoking a function **add()**.
- ✓ We are calling the function **add()** from the main function then main() is "**calling function**" and invoked function add() is called as **"called function".**

Formal Parameters and Actual Parameters

Formal Parameters:

The variables defined in the function header or 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 are called **actual** parameters. Using actual parameters, the data can be transferred to the function. The corresponding **formal** arguments in the **function definition** receive them. The **actual** parameters and **formal** parameters must match in number and type of data.

Ex:

```
#include<stdio.h>
void swap(int,int);
void main()
{
        int a,b;
        printf("enter values for a and b:");
        scanf("%d%d",&a,&b);
        printf("the values before swapping are a=%d b=%d \n",a,b);
        swap(a,b); // The variables in Function call are known as Actual parameters
        printf("the values after swapping are a=%d b=%d \n",a,b);
}

void swap(int c, int d) // The variables in function header are known as Formal parameters
{
        int temp;
```

```
        temp=c;
        c=d;
        d=temp;
}
```

Differences between Actual and Formal Parameters

Actual Parameters	Formal Parameters
Actual parameters are also called as argument list. Ex: y=gcd(10,12)	Formal parameters are also called as parameter list. **Ex:** int gcd(int x, int y)
Actual parameters are used in calling function when a function is called or invoked Example: c= sub(a,b); Here, a and b are called actual parameters	Formal parameters are used in the function header of a called function. Example: int sub(int m,int n) { …….. } Here, m and n are called formal parameters.
Actual parameters can be constants, variables or expressions **Example:** C=sub(4,a+4);	Formal parameters should be only variables. Expressions and constants are not allowed **Example:** int sub(int m, int n) { …. }

Actual parameters sends values to the formal parameters
Example:
C=sub(4,5);
	Formal parameters receive values from the actual parameters.
	Example:
	int sub(int m, int n)
	{
	}
	Here, m will have the value of 4 and n will have the value 5.

Addresses of actual parameters can be sent to formal parameters.	If formal parameters contain addresses, they should be declared as pointers.