MODULE 3

FUNCTIONS AND ARRAYS

FUNCTIONS

If the program size is very large, there are many disadvantages like

- Very difficult for writing a large program.
- Very difficult to identify and correct logical errors.
- Very difficult to understand and read the program.
- More prone to errors.
- So, functions will overcome all such disadvantages.

✓ A large program can be divided into a number of individual smaller programs called **modules.**

✓ These modules are called **functions**.

✓ The functions are also called **subprograms.**

✓ The functions can be developed and tested separately.

✓ Every function returns a value.

✓ C enables programmers to break up a program into segments commonly known as functions, each of which can be written more or less independently of the others.

✓ Every function in the program is supposed to perform a well-defined task.

✓ Therefore, the code of one function is completely insulated from the other functions.

TYPES OF FUNCTIONS

➢ C functions can be classified into **two** types,

✓ **Library functions /pre-defined functions /standard functions /built in functions**

✓ **User defined functions**

Library functions /pre-defined functions /standard functions/Built in Functions

➢ These functions are defined in the **library of C compiler** which are used frequently inthe C program.

➢ These functions are written by designers of c compiler.

- C supports many built in functions like
  - Mathematical functions
  - String manipulation functions
  - Input and output functions
  - Memory management functions
  - Character Functions
- EXAMPLE:
  - pow(x,y)-computes $\mathbf{x}^{\mathbf{y}}$
  - sqrt(x)-computes square root of x
  - strcpy(),strcmp()-used to copy and compare the string
  - printf()- used to print the data on the screen
  - scanf()-used to read the data from keyboard.
  - Isalpha(),toLower(),toUpper()-used to check alphabet, lower case, upper case

<u>User Defined Functions</u>

- The functions written by the programmer /user to do the specific tasks are called userdefined function(UDF's).
- The user can construct their own functions to perform some specific task. This type offunctions created by the user is termed as User defined functions.

<u>Advantages fo Functions</u>

- Reusability and Reduction of code size
- Readability of the program can be increased
- Modular programming approach
- Easier Debugging
- Build Library
- Function sharing

Elements of User Defined Function

The Three Elements of User Defined function are :

1.Function Definition

2.Function Declaration

3.Function call

Function Definition:

- A program Module written to achieve a specific task is called as function definition.
- Each function definition consists of two parts:

  i. Function header

  ii. Function body

General syntax of function definition

Function Definition Syntax	Function Definition Example

Datatype function name(parameters)
{
declaration part;
executable part;
return statement;
}
	void add()
	{
	int sum,a,b;
	printf("enter a and b\n");
	scanf("%d%d",&a,&b);
	sum=a+b;
	return sum;
	}

Function header:

Header Consists of data type of value return by the function, name of the function and parameters

Syntax:

datatype function name(parameters)

- **Datatype:**
  - ✓ The data type can be int, float,char,double,void.
  - ✓ This is the data type of the value that the function is expected to return.
- **function name**:
  - ✓ The name of the function.
  - ✓ It should be a valid identifier.
- **parameters**
  - ✓ The parameters are list of variables enclosed within parenthesis.
  - ✓ The list of variables should be separated by comma.

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** and
- ➢ The parameters are **'a' and 'b'** of type integer.

<u>**Function body**</u>

- ➢ The function body consists of the set of instructions enclosed between
- ➢ **{ and } .**
- ➢ The function body consists of following three elements:

a) **Declaration part:** variables used in function body.

b) **Executable part:** set of Statements or instructions that perform specific activity.

c) return : It is a keyword,it is used to **return control back to calling function.**

If a function is not returning value then statement is: **return;**

If a function is returning value then statement is: **return value;**

Syntax with Example:

```
 type function name(Parameters)     int add(int m, int n)              //Function Header
{                                     {
     Declaration part;                       int sum;
      Execution part ;                       sum=m+n               //Function Body
     Return statement ;                      return sum;
}                                      }
```

Function Declaration

- The process of declaring the function before they are used is called as functiondeclaration or function prototype.
    - function declaration Consists of the data type of function, name of the functionand parameter list ending with semicolon.

Function Declaration Syntax
datatypefunctionname(type p1,type p2,………type pn); **Example** **int   add(int a, int b);** **void   add(int a, int b);**

Note: The function declaration should end with a <u>semicolon ;</u>

2. Function Call:

- The method of calling a function to achieve a specific task is called as functioncall.
- 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 theprogram to achieve a specific task.

Ex:

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

FORMAL PARAMETERS AND ACTUAL PARAMETERS

- **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 beseparated 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 ofdata.

- **Differences between Actual and Formal Parameters**

Actual Parameters	**Formal Parameters**
Actual parameters are also called as **argument list.** Ex: add(m,n)	Formal parameters are also called as **dummy parameters.** **Ex:**int add(int a, int b)
The variables used in function call are called as actual parameters	The variables defined in function header are called formal parameters
Actual parameters are used in calling function when a function is called or invoked Ex: add(m,n) Here, m and n are called actual parameters	Formal parameters are used in the function header of a called function. Example: int add(int a, int b) { ……….. } Here, a and b are called formal parameters.
Actual parameters sends data to the formal parameters Example:	Formal parameters receive data from the actual parameters.

➢ **Categories of the functions**

1.Function with no parameters and no return values
2.Function with no parameters and return values.
3.Function with parameters and no return values
4.Function with parameters and return values

1. Function with no parameters and no return values
(void function without parameter)

Calling function	**Called function**

/*program to find sum of two numbers using function*/
#include<stdi
o.h>void
add( );
void main( )
{
    add( );
}
	void add ( )
	{
	    int sum;
	    printf("enter a and b values\n");
	    scanf("%d%d",&a,&b);
	    sum=a+b;
	    printf("\n The sum is %d", sum);return;
	}

- ✓ In this category **no data** is transferred from **calling function to called function**,hence called function cannot receive any values.
- ✓ In the above example,no arguments are passed to user defined function **add( ).**
- ✓ Hence no parameter are defined in function header.
- ✓ When the control is transferred from calling function to called function a ,and bvalues are read,they are added,the result is printed on monitor.
- ✓ When return statement is executed ,control is transferred from called function/add to calling function/main.

2. **Function with parameters and no return values**

(void function with parameter)

Calling function

```
/*program to find sum of two  numbers
using function*/
#include<stdio.h>
void add(int m,
int n);void
main()
{
        int m,n;
        printf("enter values for m and
        n:");scanf("%d %d",&m,&n);
        add(m,n);
}
```

Called function

```
void add(int a, int b)
{
        int
        sum;
        sum =
        a+b;
        printf("sum
         is:%d",sum);return;
}
```

- ✓ In this category, **there is data transfer** from the calling function to the calledfunction using parameters.
- ✓ **But there is no data transfer from** called function to the calling function.
- ✓ The values of actual parameters m and n are copied into formal parameters a and b.
- ✓ The value of a and b are added and result stored in sum is displayed on the screenin called function itself.

3. Function with no parameters and with return values	
Calling function	**Called function**

/*program to find sum of two numbers using function*/
#include<st
dio.h>int
add();
void main()
{
int
result;
result=
add( );
printf("sum is:%d",result);
}
	int add( ) /* function header */
	{
	int a,b,sum;
	printf("enter values for a andb:");
	scanf("%d
	%d",&a,&b);
	sum= a+b;
	return sum;
	}

- ✓ In this category **there is no data transfer from the calling function to thecalled function.**
- ✓ But, there is data transfer from called function to the calling function.
- ✓ No arguments are passed to the function add( ). So, no parameters are defined inthe function header
- ✓ When the function returns a value, the calling function receives one value fromthe called function and assigns to variable result.
- ✓ The result value is printed in calling function.

4. Function with parameters and with return values	
Calling function	**Called function**

/*program to find sum of two numbers using function*/
`#include<st`
`dio.h>int`
`add();`
`void main()`
`{`
`    int result,m,n;`
`    printf("enter values for m andn:");`
`    scanf("%d %d",&m,&n);`
`    result=add(m,n);`
`    printf("sum is:%d",result);`
`}`
	`int add(int a, int b) /* function header */`
	`{`
	`    int`
	`    sum;`
	`    sum=`
	`    a+b;`
	`    return`
	`    sum;`
	`}`

- ✓ In this category, there is data transfer between the calling function and calledfunction.
- ✓ When Actual parameters values are passed, the formal parameters in calledfunction can receive the values from the calling function.
- ✓ When the add function returns a value, the calling function receives a value fromthe called function.
- ✓ The values of actual parameters m and n are copied into formal parameters a andb.
- ✓ Sum is computed and returned back to calling function which is assigned to variable result.

PASSING PARAMETERS TO FUNCTIONS OR TYPES OF ARGUMENT PASSING

The different ways of passing parameters to the function are:

- ✓ Pass by value or Call by value
- ✓ Pass by address or Call by address

Call by value:

- In call by value, the values of actual parameters are copied into formal parameters.
- The formal parameters contain only a copy of the actual parameters.
- So, even if the values of the formal parameters changes in the called function, thevalues of the actual parameters are not changed.
- The concept of call by value can be explained by considering the following program.

Example:
```
#include<stdio.h>

        void swap(int a,int b);

        void main()
{
    int m,n;
    printf("enter values for a and b:");scanf("%d %d",&m,&n);
    printf("the values before swapping are m=%d n=%d
    \n",m,n);swap(m,n);
    printf("the values after swapping are m=%d n=%d \n",m,n);
 }
```

- Execution starts from function main( ) and we will read the values for variablesm and n, assume we are reading 10 and 20 respectively.
- We will print the values before swapping it will print 10 and 20.
- The function swap( ) is called with actual parameters m=10 and n=20.
- In the function header of function swap( ), the formal parameters a and breceive the values 10 and 20.
- In the function swap( ), the values of a and b are exchanged.

- But, the values of actual parameters m and n in function main( ) have not beenexchanged.
- The change is not reflected back to calling function.

✓ **Call by Address**

- In Call by **Address,** when a function is called, the addresses of actualparameters are sent.
- In the called function, the formal parameters should be declared as pointerswith the same type as the actual parameters.
- The addresses of actual parameters are copied into formal parameters.
- Using these addresses the values of the actual parameters can be changed.
- This way of changing the actual parameters indirectly using the addresses ofactual parameters is known as pass by address.

Example: #include<stdio.h> void swap(int a,int b);void main()

```
{
        int m,n;
        printf("enter values for a and
        b:");scanf("%d
        %d",&m,&n);
        printf("the values before swapping are m=%d
        n=%d \n",m,n);swap(&m,&n);
        printf("the values after swapping are m=%d n=%d \n",m,n);
  }
```

POINTER:

A pointer is a variable that is used to store the address of another variable.

Syntax: datatype *variablename;

Example: int *p;

Differences between Call by Value and Call by reference

Call by Value	Call by Address
When a function is called the **valuesof variables are passed**	When a function is called the **addresses ofvariables are passed**
The type of formal parameters shouldbe same as type of actual parameters	The type of formal parameters should be same as type of actual parameters, but they have to be declared as **pointers.**
Formal parameters contains	Formal parameters contain the addressesof actual parameters.

Scope and Life time of a variable

Scope of a variable is defined as the region or boundary of the program in whichthe variable is visible. There are two types

(i) Global Scope

(ii) Local Scope

i. Global Scope:

- The variables that are defined outside a block have global scope.
- That is any variable defined in global area of a program is visible from itsdefinition until the end of the program.
- For Example, the variables declared before all the functions are visibleeverywhere in the program and they have global scope.

ii. Local Scope

a. The variables that are defined inside a block have local scope.
b. They exist only from thepoint of their declaration until the end of the block.
c. They are not visible outside the block.

.

❖ Life Span of a variable

- The life span of a variable is defined as the period during which a variable isactive during execution of a program.

For Example

- ❖ The life span of a global variable is the life span of the program.
- ❖ The life span of local variables is the life span of the function, they arecreated.
- ❖ **Storage Classes**

- **There are following storage classes which can be used in a C *Program:***
-

  i. *Global variables*
  ii. *Local variables*
  iii. *Static variables*
  iv. *Register variables*

i. Global variables:

- These are the variables which are defined before all functions in global areaof the program.
- Memory is allocated only once to these variables and initialized to zero.
- These variables can be accessed by any function and are alive and activethroughout the program.
- Memory is deallocated when program execution is over.

ii. *Local variables(automatic variables)*

- These are the variables which are defined within a functions.
- These variables are also called as automatic variables.
- The scope of these variables are limited only to the function in which theyare declared and cannot be accessed outside the function.

iii. Static variables

- The variables that are declared using the keyword static are called staticvariables.

- The static variables can be declared outside the function and inside thefunction. They have the characteristics of both local and global variables.
- Static can also be defined within a function.

  Ex:

  static int a,b;

iv. Register variables

- Any variables declared with the qualifier register is called a register variable.
- This declaration instructs the compiler that the variable under use is to bestored in one of the registers but not in main memory.
- Register access is much faster compared to memory access.Ex:

register int a;

RECURSION

- Recursion is a method of solving the problem where the solution to a problemdepends on solutions to smaller instances of the same problem.
- Recursive function is a function that calls itself during the execution.
- The two types of recursion are

  1. Direct Recursion

  2. Indirect Recursion

<u>**Direct recursion**</u>

- A recursive function that invokes itself is said to have direct recursion.
- For example factorial function calls itself hence it is called direct recursion.

<u>**Indirect recursion**</u>

- A function which contains call to another function which in turn contains callsanother function, and so on.

Design of recursive function

- ***Any recursive function has two elements:***

  i. **Base case**
  ii. **General case**

i. **Base case**

- The statement that solves the problem is called base case.
- Every recursive function must have at least one base case.
- It is a special case whose solution can be obtained without using recursion. A base case serves two purposes:

  i). It **act as a terminating condition**.

  ii). the recursive function obtains **the solution from the base case itreaches**.

ii. **General case:**

- The statement that reduces the size of the problem is called general case.
- This is done by calling the same function with reduced size.

- In general recursive function of factorial problem can be written as

5!=5*4!

4!=4*3!

3!=3*2!

2!=2*1!

1!=1*0!

0!==1 This is a base case

1!=1*0!=1*1=1

2!=2*1!=2*1=2

3!=3*2!, 3!=3*2=6

4!=4*3! , 4!=4*6=24

5!=5*4!, 5*=24,

120(answer)

Example 1.

```
/******* Factorial of a given number using
Recursion ******/#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))
}
```

ARRAYS

Arrays: Array is a sequential collection of similar data items stored sequentially one after the other in contiguous memory locations .

Pictorial representation of an array of 5 integers

10	20	30	40	50
A[0]	A[1]	A[2]	A[3]	A[4]

- An array is a collection of similar data items.
- All the elements of the array share a common name.
- Each element in the array can be accessed by the subscript (or index) and array name.
- The arrays are classified as:
  1. **Single dimensional array**
  2. **Multidimensional array.**

Single Dimensional Array.

- A single dimensional array is a linear list of related data items of same data type.
- In memory, all the data items are stored in contiguous memory locations.

Declaration of one-dimensional array (Single dimensional array) Syntax:

```
datatype  array_name[size];
```

- **datatype** can be int,float,char,double.
- **array_name** is the name of the array and it should be an valid identifier.
- **Size** is the total number of elements in array.

For example:

int a [5];

The above statement allocates 5*2=10 Bytes of memory for the array **a.**

a[0]	a[1]	a[2]	a[3]	a[4]

float b[5];

b[0]	b[1]	b[2]	b[3]	b[5]

The above statement allocates 5*4=20 Bytes of memory for the array **b.**

- Each element in the array is identified using integer number called as i**ndex.**
- If n is the size of array, the array index starts from **0** and ends at **n-1.**

Storing Values in Arrays

- Declaration of arrays only allocates memory space for array. But array elements are not initializedand hence values has to be stored.
- Therefore to store the values in array, there are 3 methods
  1. Initialization
  2. Assigning Values
  3. Input values from keyboard through **scanf()**

Initialization of one-dimensional array

- **Assigning the required values to an array elements before processing is called initialization.**

data type array_name[expression]={v1,v2,v3…,vn};

Where

- ✓ datatype can be char,int,float,double
- ✓ array name is the valid identifier
- ✓ size is the number of elements in array
- ✓ v1,v2,v3…..... vn are values to be assigned.

- Arrays can be initialized

  at declaration time.

  Example:

  int a[5]={2,4,34,3,4};

a[0]	a[1]	a[2]	a[3]	a[4]
2	4	34	3	4

- The various ways of initializing arrays are as follows:
    1. **Initializing all elements of array(Complete array initialization)**
    2. **Partial array initialization**
    3. **Initialization without size**
    4. **String initialization**

1. **Initializing all elements of array:**

- Arrays can be initialized at the time of declaration when their initial values are known in advance.
- In this type of array initialization, initialize all the elements of specified memory size.
- Example:

int a[5]={10,20,30,40,50};

10	**20**	**30**	**40**	**50**

a[0] a[1] a[2] a[3] a[4]

2. **Partial array initialization**
    - If the number of values to be initialized is less than the size of array then it is called as partialarray initialization**.**
    - In such a case elements are initialized in the order from 0th element.
    - The remaining elements will be initialized to **zero automatically by the compiler.**
    - Example:
      int a[5]={10,20};

10	**20**	**0**	**0**	**0**

a[0] a[1] a[2] a[3] a[4]

3. **Initialization without size**

- In the declaration the array size will be set to the total number of initial values specified.
- The compiler will set the size based on the number of initial values.
- Example:

int a[ ]={10,20,30,40,50};

- **In the above example the size of an array is set to 5**

4. String Initialization

- Sequence of characters enclosed within double quotes is called as string.
- The string always ends with NULL character**(\0)**

```
char s[5]=”CITY”;
```

We can observe that string length is 4,but size is 5 because to store NULL character we need one more location.

So pictorial representation of an array **s** is as follows:

C	I	T	Y	\0
S[0]	S[1]	S[2]	S[3]	S[4]

Assigning values to arrays

Using assignment operators, we can assign values to individual elements of arrays.For example:

```
int  a[3];
        a[0]=10;
        a[1]=20;
        a[2]=30;
```

10	20	30
a[0]	a[1]	a[2]

Reading and writing single dimensional arrays.

To read array elements from keyboard we can use **scanf()** function as follows:

To read **0th** element: scanf(“%d”,&a[0]); To read **1st** element: scanf(“%d”,&a[1]); To read **2nd** element: scanf(“%d”,&a[2]);

……

…….

To read **nth** element :
scanf(“%d”,&a[n-1]);

In general

To read **ith** element:

scanf("%d",&a[i]); where i=0; i<n; i++

To print array elements we can use **printf()** function as follows:

To print **0th** element: printf("%d",a[0]); To print **1st** element: printf("%d",a[1]); To print **2nd** element :printf("%d",a[2]);

……..

……..

To **nth** element : printf("%d",&a[n-1]);

In general

To read **ith** element:

printf("%d",a[i]); where i=0; i<n; i++

Write a C program to read N elements from keyboard and to print N elements on screen.

```
/* program to read N elements from keyboard and to print N elements on
screen */
#include<stdio.h>
void main()
{
        int i,n,a[10];
        printf("enter number of array elements\n");
        scanf("%d",&n);
        printf("enter array elements\n");
        for(i=0; i<n;i++)
        {
                scanf("%d",&a[i]);
        }

        Printf("array elements are\n"):
        for(i=0; i<n;i++)
        {
                printf("%d",a[i]);
        }
}
```

2.	Write a C program to find sum of n array elements .

```
/* program to find the sum of n array elements.*/
#include<stdio.h>
void main()
{
        int i,n,a[10],sum=0;
        printf("enter number of array elements\n");
        scanf("%d",&n);
        printf("enter array elements\n");
        for(i=0; i<n; i++)
        {
                scanf("%d",&a[i]);
        }
        for(i=0; i<n;i++)
        {
                sum=sum+ a[i];
        }
        printf("sum is %d\n",sum):

    }
```

Write a c program to find largest of n elements stored in an array a.

```
#include<stdio.h>
void main()
{
        int i,n,a[10],big;
        printf("enter number of array elements\n");scanf("%d",&n);
        printf("enter array elements\n");
        for(i=0; i<n;i++)
        {
                scanf("%d",&a[i]);
        }
        big=a[0];
        for(i=0; i<n;i++)
        {
                if(a[i]>big)
                 big=a[i];
        }
        printf("the biggest element in an array is %d\n",big);
}
```

USING ARRAYS WITH FUNCTIONS:

In large programs that use functions we can pass Arrays as parameters. Two ways of passing arrays tofunctions are:

1. Pass individual elements of array as parameter
2. Pass complete array as parameter

Pass individual elements of array as parameter	**Pass complete array as parameter**
Here, each individual element of array is passed to function separately.	Here, the complete array is passed to the function.

Example:
 	
#include<stdio.h>
int square(int);
	Example:
	
	#include<stdio.h>
	int sum(int [ ]);

```
int main( )
{
    int num[5], i;
    num[5] ={1, 2, 3, 4, 5};
    for(i=0; i<5; i++)
    {
            square(num[i]);
    }
}
int square(int n)
{
  int sq;
  sq= n * n;
  printf(“%d  ”, sq);
}
```
	```
	int main( )
	{
	    int marks[5], i;
	    marks[5] ={10, 20, 30, 40, 50};
	    sum(marks);
	}
	int sum(int n[ ])
	{
	    int i, sum=0;
	    for(i=0; i<5; i++)
	    {
	        sum = sum+n[i];
	     }
	     printf(“Sum = %d ”, sum);
	}
	```

Operations On One Dimensional Arrays

Searching

- The process of finding a particular item in the large amount of data is called searching.
- The element to be searched is

called key element.There are two methods of searching:

1] Linear search.

2] Binary search.

1] Linear Search:

- Linear search also called sequential search is a simple searching technique.
- In this technique we search for a given key item in linear order i.e,one after the other fromfirst element to last element.
- The search may be successful or unsuccessful.

- If key item is present, the search is successful, otherwise unsuccessful search.

1. **Program to implement linear search.**

```
#include<stdio.h>
void main()
{
        int i,n,a[10],key;
        clrscr( );
        printf(“enter array elements\n”);
        scanf(“%d”,&n);
        printf(“enter array elements\n”);
        for(i=0; i<n;i++)
        {
                scanf(“%d”,&a[i]);
        }
        printf(“enter the key element\n”);
        scanf(“%d”„&key);

        for(i=0; i<n;i++)
        {
        if(key==a[i])
                {
                        printf(“successful search\n”);
                        exit(0);
                }
        }
        printf(“unsuccessful search\n”);
}
```

Advantages of linear search

- Very simple Approach.
- Works well for small arrays.
- Used to search when elements are not sorted.

Disadvantages of linear search

- Less efficient if array size is large
- If the elements are already sorted, linear search is not efficient.

2] Binary Search:

- Binary search is a simple and very efficient searching technique which can be applied if the items arearranged in either ascending or descending order.

- In binary search first element is considered as low and last element is considered as high.
- Position of middle element is found by taking first and last
  element is as follows.mid=(low+high)/2

- Mid element is compared with key element, if they are same, the search is successful.
- Otherwise if key element is less than middle element then searching continues in left part of the array.
- If key element is greater than middle element then searching continues in right part of the array.
- The procedure is repeated till key item is found or key item is not found.

/* C program to search a name in a list of names using Binarysearching technique*/

```
#include<stdio.h>
void main()
{
        int i, n, low, high, mid,a[50],key;
        printf(“enter the number of elements\n”);
        scanf(“%d”,&n);
        printf(“enter the elements\n”);
        for(i=0;i<n;i++);
        {
                Scanf(“%d”,&a[i]);
        }
        printf(“enter the key element to be searched\n”);
        scanf(“%d”,&key);
      low=0; high=n-1;
        while(low<=high)
        {
                mid=(low+high)/2;
                if(key==a[mid])
                {
                        printf(“successful search\n”);
                        exit(0);
                }
                if(key<a[mid])
                {
                        high=mid-1;
                }
                else
                {
                        low=mid+1;
                }
        }
```

```
                printf("unsuccesfull seasrch\n");
                }
```

Advantages of binary search

1. Simple technique
2. Very efficient
   searching
   technique
   Disadvantages
1. The elements should be sorted.
2. It is necessary to obtain the middle element, which are stored in array. If the elements are stored in linkedlist, this method cannot be used.

Sorting

➢ The process of arranging elements in either ascending order or descending order is called Sorting.

Bubble Sort

➢ This is the simplest and easiest sorting technique.
➢ In this technique two successive elements of an array such as a[j] and a[j+1] are compared.
➢ If a[j]>=a[j+1] the they are exchanged, this process repeats till all elements of an array are arranged inascending order.
➢ After each pass the largest element in the array is sinks at the bottom and the smallest element in thearray is bubble towards top. So this sorting technique is also called as sinking sort and bubble sort.

```
#include<stdi
o.h>void
main()
{
        int a[20],n,.temp,i,j;
        printf("enter the number of elements\n"):
        scanf("%d",&n);
        printf("enter the unsorted array
        elements\n");for(i=0;i<n;i++)
                scanf("%d",
        &a[i]);
        for(i=0;i<n;i++)
        {
                for(j=0;j<n-i;j++)
```

```
                        {
                             If( a[ i ] > a[ i+1 ] )
                             {
                                  temp=a
                                  [j];
                                  a[j]=a[j
                                  +1];
                                  a[j+1]=
                                  temp;
                              }
                        }
               }
                   printf("the sorted elements
               arer\n");for(i=0;i<n;i++)
               printf(" %d\t", a[i]);
}
```

Traversal

```
#include<stdio.h>
Main()
        {
Int a[50],n,i;
Printf("Enter number of array elements");
Scanf("%d",&n);
Printf("Enter elements of array");
For(i=0;i<n;i++)
{
Scanf("%d",&a[i]);
}
Printf("Elements  in array are:");
For(i=0;i<n;i++)
{
Printf("%d",a[i]);
}
}
```

Inserting an Element at Specific Position

```
#include<stdio.h>
Main()
{
int a[50],size,num,pos,i;
Printf("Enter number of array elements");
Scanf("%d",&size);
Printf("Enter elements of array");
For(i=0;i<n;i++)
{
```

```
Scanf("%d",&a[i]);
}
Printf("Enter element to be inserted");
Scanf("%d",&num);
Printf("Enter the position to insert element");
Scanf("%d",&pos);

For(i=size-1; i>=pos-1; i--)
{
    a[i+1]=a[i];
}
   a[pos-1]=num;
    size++;
for(i=0; i<size; i++)
{
 Printf("%d",a[i]);
}
}
```

<u>Deleting an Element from Specific Position</u>

```
#include<stdio.h>
Main()
{
int a[50],size,pos,item,i;
Printf("Enter number of array elements");
Scanf("%d",&size);
Printf("Enter elements of array");
For(i=0;i<n;i++)
{
Scanf("%d",&a[i]);
}
Printf("Enter position to delete the element");
Scanf("%d",&pos);

If(pos < = 0 || pos > size)
{
Printf("Invalid  position");
}
Else
{
Item=a[pos-1];
For(i=pos-1; i<size-1; i++)
{
  a[i]=a[i+1];
}
Size--;
For(i=0; i<size; i++)
{
Printf("%d",a[i]);
}
}
}
```