# Module 3

## ARRAYS

### INTRODUCTION:

**Arrays**: Array is a sequential collection of similar data items. 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. Two-dimensional array
    3. 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 location.

Syntax:

```
datatype array_name(size);
```

- datatype can be int, float, char, double.
- array_name is the name of the array and it should be a 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;

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 index.
- 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 initialized and hence values have 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 element before processing is called initialization.

Syntax:

datatype array_name[size] = {v1,v2,v3…,vn};

- 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.

Note: Arrays can be initialized at declaration time.

- Example:

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

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

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** |

Partial array initialization:

If the number of values to be initialized is less than the size of array then it is called as partial array initialization. In such a case element 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** |

Initialization without size:

Arrays can be initialized without specifying its size.

Syntax:

```
int a[ ] = { 10, 20 };
```

| | |
|---|---|
| **10** | **20** |

Assigning values to arrays:

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

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:

```
for(i=0;i<n;i++)
{
scanf("%d",&a[i]);
}
```

To display array elements in output screen we can use printf () function as follows:

```
for(i=0;i<n;i++)
{
printf("%d",a[i]);
}
```

Example Programs:

Program 1: Write a C 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]);
}
}
```

Program 2: Write a C program to find 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=%d",sum);
}
```

Program 3: 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);
}
```

Program 4: Write a C program to generate Fibonacci numbers using arrays.

```
#include<stdio.h>
void main()
{
int i, n, a [10];
a [ 0] = 0;
a [ 1] = 1;
printf (“enter n\n”);
scanf (“%d”, &n);
if(n==1)
{
printf (“%d\t”, a [0]);
}
if(n==2)
{
printf(“%d\t %d\t”,a[0],a[1]);
}
if(n>2)
{
printf(“%d \t %d\t”,a[0],a[1]);
for(i=2;i<n;i++)
{
a[i]=a[i-1]+a[i-2];
printf(“%d\t”,a[i]);
}
}
}
```

Two Dimensional arrays:

In two dimensional arrays, elements will be arranged in rows and columns. To identify two dimensional arrays, we will use two indices (say i and j) where I index indicates row number and j index indicates column number.

Declaration of two-dimensional array:

Syntax:

data_type array_name[exp1][exp2];

Or

data_type array_name[row_size][column_size];

- data_type can be int,float,char,double.
- array_name is the name of the array.
- exp1 and exp2 indicates number of rows and columns

Example:

```
int a [ 3 ] [ 4 ];
```

The above statements allocate memory for 3*4=12 elements i.e 12*2=24 bytes.

Initialization of two-dimensional array:

Assigning or providing the required values to a variable before processing is called initialization.

Syntax:

```
for(i=0;i<n;i++)
{
    for(j=0;j<m;j++)
    {
        scanf("%d",&a[i][j]);
    }
}
```

**OR**

```
data_type array_name[exp1][exp2]={{a1,a2,….an}, {b1,b2,  .bn}, ....................., {z1,z2,…zn}};
```

- data type can be int, float etc.
- exp1 and exp2 are enclosed within square brackets.
- both exp1 and exp2 can be integer constants or constant integer expressions (number of rows and number of columns).
- a1 to an are the values assigned to 1st row, b1 to bn are the values assigned to 2nd row and so on.

Example:

```
int a[3][3]={{10,20,30}, {40,50,60}, {70,80,90}};
```

| | | |
|---|---|---|
| **10** | **20** | **30** |
| **40** | **50** | **60** |
| **70** | **80** | **90** |

Example:

Program 1: Write a c program to read & print 2d array as a Array.

```
#include<stdio.h>
 void main()
{
int m,n,i,j,a[3][3];
printf("enter number of rows and columns\n");
scanf("%d %d",&m,&n);
printf("eneter array elements\n");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
scanf("%d",&a[i][j]);
}
}
printf("array elements are\n");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
printf("%d",a[i][j]);
}
printf("\n");
}
}
```

Program 2: Write a c program to add two matrices.

```
#include<stdio.h>
void main()
{
int m,n,i,j,a[3][3],b[3][3] ,c[3][3];
printf("enter number of rows and columns\n");
scanf("%d %d",&m,&n);
printf("enter array a elements\n");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
scanf("%d",&a[i][j]);
}
}
printf("enter array b elements\n");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
```

```
scanf("%d",&b[i][j]);
}
}
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
c[i][j]=a[i][j]+b[i][j];
}
}
printf("resultant matrix c is \n");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
printf("%d\t",c[i][j]);
}
printf("\n");
}
}
```

Program 3: Write a c program to copy one 2d array in to another 2d array

```
#include<stdio.h>
void main()
{
int m,n,i,j,a[3][3],b[3][3];
clrscr();
printf("enter number of rows and columns\n");
scanf("%d %d",&m,&n);
printf("enter array a elements\n");
for(i=0; i<m; i++)
{
for(j=0; j<n; j++)
{
scanf("%d", &a[i][j]);
}
}
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
b[i][j]=a[i][j];
}
}
printf("matrix b is \n");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
```

```
{
printf("%d\t",b[i][j]);
}
printf("\n");
}
}
```

Program 4: Write a c program to find biggest element in a matrix or 2D array.

```
#include<stdio.h>
void main()
{
int m,n,i,j,a[3][3];
printf("enter number of rows and columns\n");
scanf("%d %d",&m,&n);
printf("enter array elements\n"); for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
scanf("%d",&a[i][j]);
}
}
big=a[0][0];
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
if(big>a[i][j])
big=a[i][j];
}
}
printf("big is %",big);
}
```

Program 5: Write a C program to implement Matrix Multiplication.

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
int m,n,i,j,sum,p,q,k,a[3][3],b[3][3],c[3][3];
printf("enter number of rows and columns of matrix a \n");
scanf("%d %d",&m,&n);
printf("enter number of rows and columns of matrix b \n");
scanf("%d %d",&p,&q);
if(n!=p)
{
printf("multiplication not possible\n");
exit(0);
```

```
}
printf("enter matrix a elements\n");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
scanf("%d",&a[i][j]);
}
}
printf("enter array b elements\n");
for(i=0;i<p;i++)
{
for(j=0;j<q;j++)
{
scanf("%d",&b[i][j]);
}
}
for(i=0;i<m;i++)
{
for(j=0;j<q;j++)
{
{
c[i][j]=0; for(k=0;k<n;k++)
{
c[i][j]= c[i][j]+a[i][k]*b[k][j];
}
}
}
printf("resultant matrix a is \n");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
printf("%d\t",a[i][j]);
}
printf("\n");
}
printf("resultant matrix a is \n");
for(i=0;i<p;i++)
{
for(j=0;j<q;j++)
{
printf("%d\t",b[i][j]);
}
printf("\n");
}
printf("resultant matrix a is \n");
for(i=0;i<m;i++)
{
```

```
for(j=0;j<q;j++)
{
printf("%d\t",c[i][j]);
}
printf("\n");
}
}
```

Program 6: Write a program to find sum of each row and sum of each column

```
#include<stdio.h>
void main()
{
int m,n,i,j,rsum,csum,a[3][3];
printf("enter number of rows and columns\n");
scanf("%d %d",&m,&n);
printf("enter array elements\n");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
{
scanf("%d",&a[i][j]);
}
}
for(i=0;i<m;i++)
{
rsum=0;
for(j=0;j<n;j++)
{
rsum=rsum+a[i][j];
}
printf("sum is %d",rsum);
}
for(i=0;i<m;i++)
{
csum=0;
for(j=0;j<n;j++)
{
csum=csum+a[j][i];
}
printf("sum is %d",csum);
}
}
```

Program 7: Develop a C Program to create identity matrix

```
#include <stdio.h>
void main()
{
int i, j, n;
printf("Enter the dimension of the matrix: ");
scanf("%d", &n);
for(i = 0; i < n;i++)
{
for(j = 0; j < n;j++)
{
if(i == j)
{
 printf("1 ");
}
else
{
printf("0 ");
}
}
printf("\n");
}
}
```

Program 8: Develop a C Program to transpose a matrix

```
#include <stdio.h>
void main()
{
  int a[10][10], transpose[10][10], r, c;
  printf("Enter rows and columns: ");
  scanf("%d %d", &r, &c);
  printf("\nEnter matrix elements:\n");
  for (int i = 0; i < r; ++i)
  for (int j = 0; j < c; ++j)
  {
    printf("Enter element a%d%d: ", i + 1, j + 1);
    scanf("%d", &a[i][j]);
  }
  printf("\nEntered matrix: \n");
  for (int i = 0; i < r; ++i)
  for (int j = 0; j < c; ++j)
  {
    printf("%d ", a[i][j]);
    if (j == c - 1)
    printf("\n");
```

```
    }
    for (int i = 0; i < r; ++i)
    for (int j = 0; j < c; ++j)
    {
      transpose[j][i] = a[i][j];
    }
    printf("\nTranspose of the matrix:\n");
    for (int i = 0; i < c; ++i)
    for (int j = 0; j < r; ++j)
    {
      printf("%d ", transpose[i][j]);
      if (j == r - 1)
      printf("\n");
    }
  }
```

Multidimensional Array:

In C, we can define multidimensional arrays in simple words as an array of arrays. Data in multidimensional arrays are stored in tabular form (in row-major order).

The general form of declaring N-dimensional arrays:

data_type  array_name[size1][size2]... [sizeN];

- data_type: Type of data to be stored in the array.
- array_name: Name of the array
- size1, size2,... ,sizeN: Sizes of the dimensions

Example:

int a[10][10][10];

The total number of elements that can be stored in a multidimensional array can be calculated by multiplying the size of all the dimensions.

The array int a[10][20] can store total (10*20) = 200 elements.

Similarly array int a[10][10][10] can store total (10*10*10) = 1000 elements.

# Character Arrays and Strings:

A character array is a sequence of characters, just as a numeric array is a sequence of numbers. A typical use is to store short pieces of text as character vectors, such as c = 'ABC'.

String is a sequence of characters that is treated as a single data item and terminated by null character'\0'. Remember that C language does not support strings as a data type. A string is actually one-dimensional array of characters in C language. These are often used to create meaningful and readable programs.

Is character array same as strings?

Both Character Arrays and Strings are a collection of characters but are different in terms of properties. String refers to a sequence of characters represented as a single data type. Character Array is a sequential collection of data type char. Strings are immutable.

The following 4 points are good enough to make use of strings.

- In C, strings are arrays of characters.
- Every string in a C program ends with '\0' (Null) character.
- Strings are always enclosed within double quotes.
- If the program uses the string handling functions, it must define under string.h header file.

Declaration and Initialization of String Variable:

The general syntax for declaring a variable as a String is as follows:

```
char string_variable_name [array_size];
```

The classic Initialization of strings can be done as follow:

```
char string_name[string_length] = "string";
```

The size of an array must be defined while declaring a C String variable because it is used to calculate how many characters are going to be stored inside the string variable in C.

Example:

```
char first_name[15] = "ANTHONY";
char first_name[15] = {'A','N','T','H','O','N','Y','\0'};
```

```
/* NULL character '\0' is required at end in this declaration */

char string1 [6] = "hello";
/* string size = 'h'+'e'+'l'+'l'+'o'+"NULL" = 6 */
```

Reading and writing Strings:

The strings can be read and write by using following I/O functions.

1. scanf() and printf()
2. gets() and puts()

The scanf() reads a string until white space/blank space found from a keyboard, whereas gets() reads a string until user press ENTER KEY

The printf() helps to write(display) strings on the monitor screen in different formats with appropriate messages, whereas puts() helps to display only the string on monitor screen without format.

## String Manipulating/Handling functions:

C supports different string handling functions to perform different operations on strings. All the string handling functions are stored in <string.h> header file.

Some of the most common used string handling functions are:

1. strlen() – Returns number of characters in the given string
2. strcpy() – Copies the given source string to another destination string variable
3. strcmp() – Compares two strings for their similarities
4. strcat() – Concatenates(joins) two strings into single string
5. strupr() – Converts characters into uppercase
6. strlwr() – Converts characters into lowercase
7. strrev() – Reverse a given string

strlen( ) Function :

strlen( ) function is used to find the length of a character string.

Example:

```
int n;
char st[20] = “Bangalore”;
n = strlen(st);
```

- This will return the length of the string 9 which is assigned to an integer variable n.
- Note that the null character “\0“ available at the end of a string is not counted.

Program 1: Write a C program to demonstrate strlen () function.

```
#include <stdio.h>
#include <string.h>
void main()
{
    char str1[20] = "BeginnersBook";
    printf("Length of string str1: %d", strlen(str1));
}
```

Output:

Length of string str1: 13

strcpy ( ) Function :

strcpy( ) function copies contents of one string into another string. Syntax for strcpy function is given below.

Syntax :

char strcpy (char destination, char source);

Example 1:

strcpy ( str1, str2) – It copies contents of str2 into str1.

strcpy ( str2, str1) – It copies contents of str1 into str2.

Example 2:

```
char city[15];
strcpy(city, “BANGALORE”) ;
```

This will assign the string “BANGALORE” to the character variable city.

Program: Write a C program to demonstrate strcpy () function.

```
#include <stdio.h>
#include <string.h>
void main()
{
    char s1[30] = "string 1";
    char s2[30] = "string 2 : I'm gonna copied into s1";
    strcpy(s1,s2);
    printf("String s1 is: %s", s1);
}
```

Output:

String s1 is: string 2: I'm gonna copied into s1

<u>strcat( ) Function:</u>

strcat( ) function in C language concatenates two given strings. It concatenates source string at the end of destination string.

Syntax :

```
char  strcat ( char  destination, char  source );
```

Example :

strcat ( str2, str1 ); - str1 is concatenated at the end of str2.

strcat ( str1, str2 ); - str2 is concatenated at the end of str1.

- As you know, each string in C is ended up with null character ('\0′).
- In strcat( ) operation, null character of destination string is overwritten by source string's first character and null character is added at the end of new destination string which is created after strcat( ) operation.

Program: Write a C program to demonstrate strcat () function.

```
#include <stdio.h>
#include <string.h>
void main()
{
    char s1[10] = "Hello";
    char s2[10] = "World";
    strcat(s1,s2);
    printf("Output string after concatenation: %s", s1);
}
```

Output:

Output string after concatenation: HelloWorld

strcmp( ) Function :

strcmp( ) function in C compares two given strings and returns zero if they are same. If length of string1 < string2, it returns < 0 value. If length of string1 > string2, it returns > 0 value.

Syntax :

```
int strcmp ( char  str1, char  str2 );
```

strcmp( ) function is case sensitive. i.e., “A” and “a” are treated as different characters.

Example :

```
char city[20] = “Madras”;
char town[20] = “Mangalore”;
    strcmp(city, town);
```

This will return an integer value “-1“.

Program: Write a C program to demonstrate strcmp () function.

```
#include <stdio.h>
#include <string.h>
void main()
{
    char s1[20] = "BeginnersBook";
    char s2[20] = "BeginnersBook.COM";
    if (strcmp(s1, s2) ==0)
    {
      printf("string 1 and string 2 are equal");
    }
else
     {
       printf("string 1 and 2 are different");
     }
  }
```

Output:

```
string 1 and 2 are different
```

strlwr() function :

strlwr() function converts a given string into lowercase.

Syntax :

```
char  strlwr(char  string);
```

Example:

```
char str[10] = "SAHYADRI"
strlwr(str);
```

strlwr() function will give "sahyadri" as output for above example.

Program: Write a C program to demonstrate strlwr () function

```
#include<stdio.h>
#include<string.h>
void main()
{
char str[10] = "SAHYADRI"
strlwr(str);
printf("%s",str);
}
```

Output:

```
sahyadri
```

<u>strupr() function</u>:

strupr() function converts a given string into uppercase.

<u>Syntax :</u>

```
char strupr(char string);
```

Example:

```
char str[10] = "sahyadri"
strupr(str);
```

strupr() function will give "SAHYADRI" as output for above example.

Program: Write a C program to demonstrate strupr () function

```
#include<stdio.h>
#include<string.h>
void main()
{
char str[10] = "sahyadri"
strupr(str);
printf("%s",str);
}
```

Output:

```
SAHYADRI
```

strrev () function:

strrev() function is used to reverse a given string.

Syntax:

```
char strrev(char string);
```

Example:

```
char str[10] = "Sahyadri"
strrev(str);
```

Output of above example is "irdayhaS"

Program: Write a C program to demonstrate strrev () function.

```
#include<stdio.h>
#include<string.h>
void main()
{
char str[10] = "Sahyadri"
strrev(str);
printf("%s",str);
}
```

Output:

"irdayhaS"

## Basic Algorithms of Searching and Sorting:

Searching Algorithms are designed to retrieve an element from any data structure where it is used. A Sorting Algorithm is used to arranging the data of list or array into some specific order.

Searching Algorithm:

There are 2 types of searching algorithms:

1. Linear search Algorithm
2. Binary search Algorithm

Linear search Algorithm:

Linear search is a very simple search algorithm. In this type of search, a sequential search is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data collection.

Algorithm:

Algorithm Name: Linear Search (Array A, Value x)

Step 1: START

Step 2: Set i to 1

Step 3: if i > n then go to step 8

Step 4: if A[i] = x then go to step 7

Step 5: Set i to i + 1

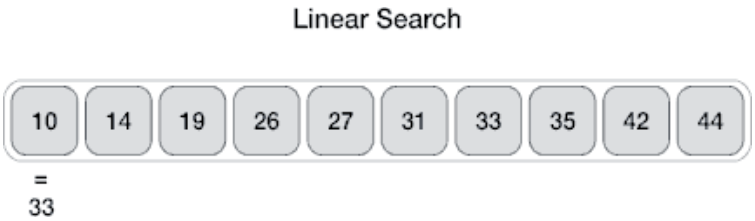
Step 6: Go to Step 3

Step 7: Print Element x Found at index i and go to step 8

Step 8: Print element not found

Step 9: STOP

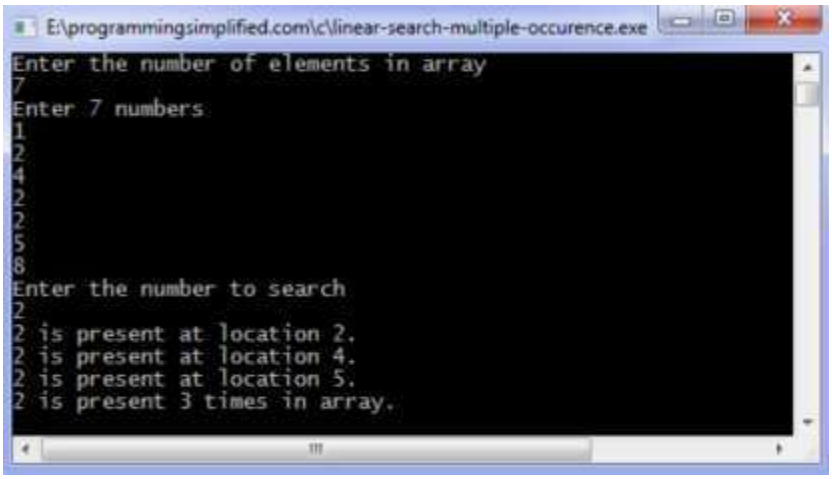
Demonstration:




Program: Develop a program to demonstrate Linear search algorithm.

```
#include <stdio.h>
void main()
{
 int array[100], search, c, n;
 printf("Enter number of elements in array\n");
 scanf("%d", &n);
 printf("Enter %d integer(s)\n", n);
 for (c = 0; c < n; c++)
{
 scanf("%d", &array[c]);
 }
 printf("Enter a number to search\n");
 scanf("%d", &search);
 for (c = 0; c < n; c++)
 {
 if (array[c] == search)
 {
  printf("%d is present at location %d.\n", search, c+1);
  break;
  }
  }
  if (c == n)
  printf("%d isn't present in the array.\n", search);
 }
```

Output:




Advantages of Linear Search Algorithm:

1. Will perform fast searches of small to medium lists.
2. The list does not need to sorted.
3. Not affected by insertions and deletions.

Disadvantages of Linear Search Algorithm:

1. Slow searching of large lists.

Binary Search Algorithm:

Binary Search is a searching algorithm used in a sorted array by repeatedly dividing the search interval in half.

The basic steps to perform Binary Search are:

1. Begin with an interval covering the whole array.
2. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half.
3. Otherwise, narrow it to the upper half.
4. Repeatedly check until the value is found or the interval is empty.

Algorithm:

Given an array A of elements with values or records A0, A1, A2, …….An-1sorted such that A0<A1<A2<…..An-1, and target value T, the following subroutine uses binary search to find the index of T in A.

Step 1: START

Step 2: Set L to 0 and R to n-1.

Step 3: If L>R, the search terminates as unsuccessful.

Step 4: Set m (the position of the middle element) to the floor of L+R/2, which is the greatest integer less than or equal to L+R/2.

Step 5: If A(m)<T, set L to m+1 and go to step 3.

Step 6: If A(m)>T, set R to m-1 and go to step 3.

Step 7: Now A(m)=T, the search is done; return m

Step 8: STOP

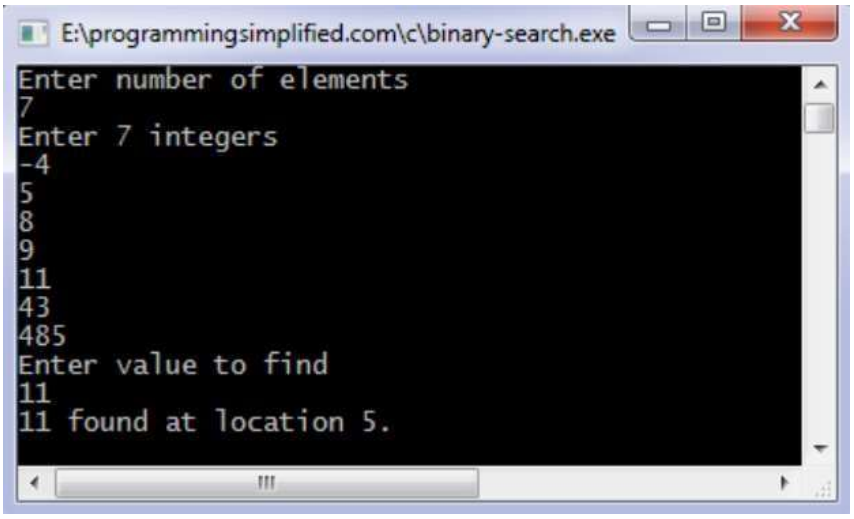
Demonstration:


Program: Write a C program to perform binary search on the array of integers.

```
#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");
}
```

Output:




<u>Advantages of Binary Search Algorithm:</u>

1. Simple technique
2. Very efficient searching technique

<u>Disadvantages of Binary Search Algorithm</u>:

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 linked list, this method cannot be used.

Sorting Algorithm:

There are 2 types of sorting algorithms

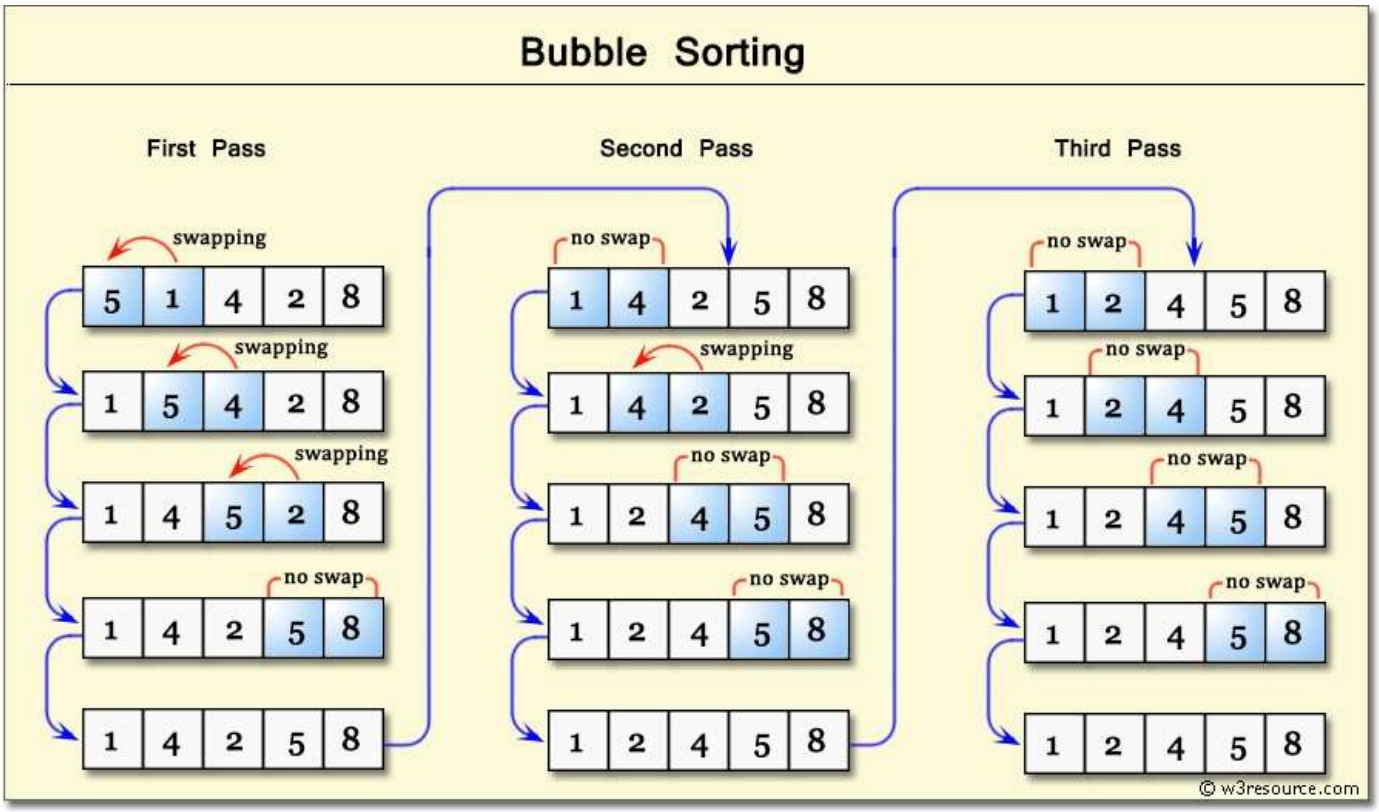
1. Bubble sort Algorithm
2. Selection sort Algorithm

Bubble Sort Algorithm:

Bubble Sort in C is a sorting algorithm where we repeatedly iterate through the array and swap adjacent elements that are unordered. We repeat this until the array is sorted.

Technique:

- 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 in ascending order.
- After each pass the largest element in the array is sinks at the bottom and the smallest element in the array is bubble towards top. So, this sorting technique is also called as sinking sort and bubble sort.



Program: Program to illustrate bubble sort

```
#include<stdio.h>
void main ()
{
  int i, j,temp;
   int a[10] = { 10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
    for(i = 0; i<10; i++)
    {
        for(j = i+1; j<10; j++)
        {
          if(a[j] > a[i])
            {
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
          }
    }
   printf("Printing Sorted Element List ...\n");

    for(i = 0; i<10; i++)
    {
        printf("%d\n",a[i]);
    }
}
```

Output:

Printing Sorted Element List …

7 9  10  12  23  23  34  44  78  101

<u>Selection Sort Algorithm:</u>

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning.

Technique:

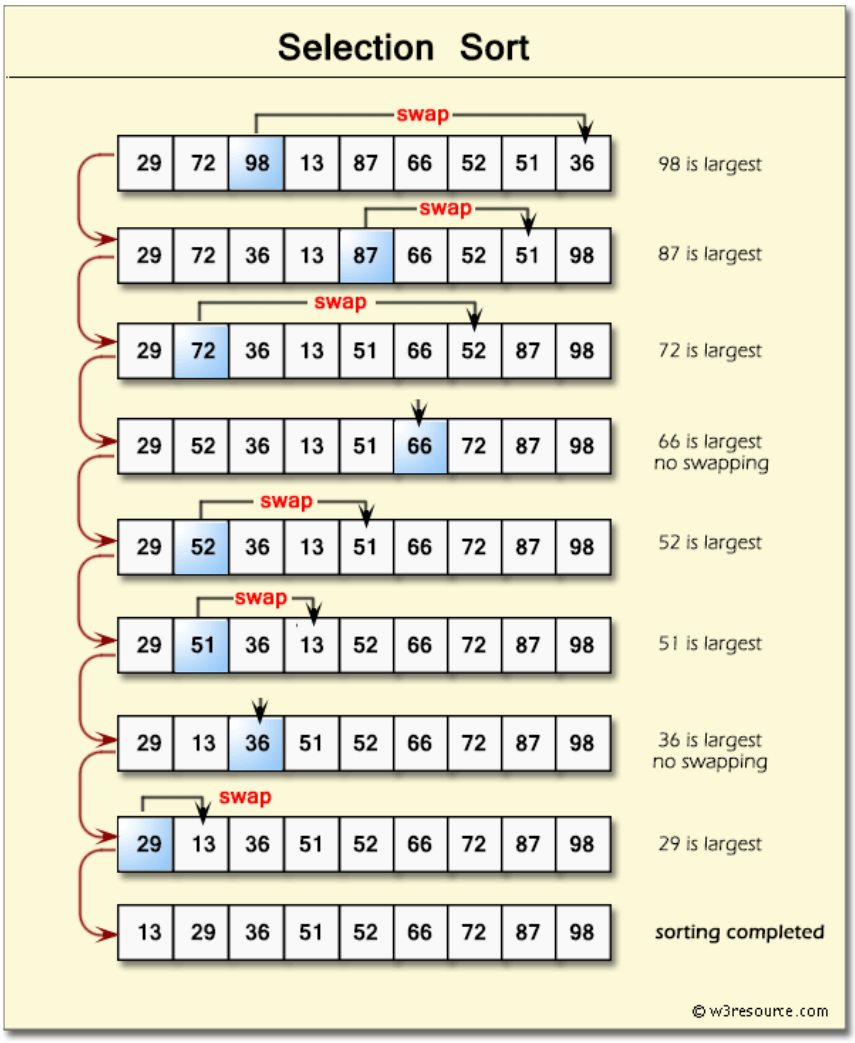
Step 1 − Set min to the first location

Step 2 − Search the minimum element in the array

Step 3 – swap the first location with the minimum value in the array

Step 4 – assign the second element as min.

Step 5 − Repeat the process until we get a sorted array.




Program: Program to illustrate selection sort

```
#include <stdio.h>
void main()
{
int a[100], n, i, j, position, swap;
printf("Enter number of elementsn");
scanf("%d", &n);
printf("Enter %d Numbersn", n);
for (i = 0; i < n; i++)
scanf("%d", &a[i]);
for(i = 0; i < n - 1; i++)
{
position=i;
for(j = i + 1; j < n; j++)
{
if(a[position] > a[j])
position=j;
}
if(position != i)
{
swap=a[i];
a[i]=a[position];
```

```
a[position=swap;
}
}
printf("Sorted Array:n");
for(i = 0; i < n; i++)
printf("%dn", a[i]);
}
```

Output:


## Dynamic Arrays:

Dynamic array in C using malloc library function. Program example will create an integer array of any length dynamically by asking the array size and array elements from user and display on the screen.

Dynamic array in C example

```
#include  <stdio.h>
#include <stdlib.h>
void main()
{
        int *ptr=NULL;
        int i,len=0;
        printf("ENTER THE SIZE OF ARRAY:");
        scanf("%d",&len);
        ptr=(int*)malloc(len*sizeof(int));
        printf("ENTER THE ELEMENTS:");
        for(i=0;i<len;++i)
        {
                scanf("%d", &ptr[i]);
        }
        printf("Array elements are \n");
        for(i=0;i<len;++i)
        {
                printf("%d,", ptr[i]);
        }
        free(ptr);
}
```

Output:

```
ENTER THE SIZE OF ARRAY:5
ENTER THE ELEMENTS:
2
3
4
10
6
Array elements are
2 3 4 10 6
```