**Lab 1 - Pointers Recap and Abstraction**

**1.** Sum of `n` numbers using only pointers

**AIM:**

Write a C++ program to find the sum of 'n' integers using only pointers.

**ALGORITHM:**

1. Sum of n numbers provided by user:

**INPUT: N** => number of integers, **N** integers

**OUTPUT: SUM** => Sum of **N** integers

* 1. Get the value of **N**.
  2. Initiate while loop if **N** != 0.
     1. Get new integer and add to **SUM.**
     2. Decrement **N**.
  3. Print **SUM**.

**COMPLEXITY ANALYSIS:**

1. While Loop : O(n)

**CODE:**

//To find the sum of n numbers given by the user

#include <stdio.h>

#include <stdlib.h>

int main(){

    int \*sum,\*n,\*temp;

    n = (int \*)malloc(sizeof(int));

    sum = (int \*)malloc(sizeof(int));

    temp = (int \*)malloc(sizeof(int));

    \*sum = 0;

    printf("\n Enter the value of n : ");

    scanf("%d",n);

    printf("Enter the integers : ");

    while(\*n != 0){

        scanf("%d", temp);

        \*sum += \*temp;

        \*n-=1;

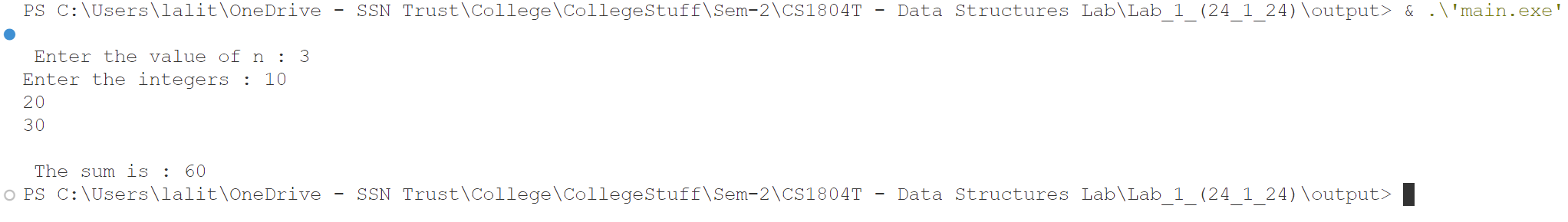
    }

    printf("\n The sum is : %d\n",\*sum);

    return 0;

}

**OUTPUT:**



**2.** Write a C++ program that has functions to calculate the sum, difference, product and difference of two integer numbers.

**AIM:**

Write a C++ program that has functions to calculate the sum, difference, product and difference of two integer numbers.

**ALGORITHM:**

1. Calculator Program

**INPUT: opt** => Option of the user, **num1** => First Integer, **num2** => Second Integer.

**OUTPUT:** Resultant Value.

* + 1. Start Menu options.
    2. **Case Add:**

Print Sum of **num1** and **num2**.

* + 1. **Case SUBTRACT:**

Print Difference of **num1** and **num2**.

* + 1. **Case MULTIPLY:**

Print Product of **num1** and **num2**.

* + 1. **Case DIVIDE:**

Print Quotient of **num1** and **num2**.

* + 1. **Case Add:**

Print Sum of **num1** and **num2**.

* + 1. **Case EXIT:**

Exit Menu

**CODE:**

“**Calculator.h”**

#include <stdio.h>

int Add(int \**num1* , int \**num2*){

    return \**num1* + \**num2*;

}

int Sub(int \**num1* , int \**num2*){

    return \**num1* - \**num2*;

}

int Mul(int \**num1* , int \**num2*){

    return \**num1* \* \**num2*;

}

float Div(int \**num1* , int \**num2*){

    return (float)\**num1* / \**num2*;

}

**“Main.cpp”**

#include "calculator.h"

int main(){

    int num1=1,num2=1,opt = 1;

    while(opt !=6){

        printf("\nChoose Your Option : \n\t1.Set\n\t2.Add\n\t3.Subtract\n\t4.Multiply\n\t5.Divide\n\t6.Exit\n>>");

        scanf("%d",&opt);

        switch (opt)

        {

        case 1:

            printf("Set the value of Num 1 : ");

            scanf("%d",&num1);

            printf("Set the value of Num 2 : ");

            scanf("%d",&num2);

            break;

        case 2:

            printf("Sum Of %d and %d is %d",num1,num2,Add(&num1,&num2));

            break;

        case 3:

            printf("Difference Of %d and %d is %d",num1,num2,Sub(&num1,&num2));

            break;

        case 4:

            printf("Product Of %d and %d is %d",num1,num2,Mul(&num1,&num2));

            break;

        case 5:

            printf("Quotient Of %d and %d is %f",num1,num2,Div(&num1,&num2));

            break;

        case 6:

        printf("Program ended\n");

            return 0;

            break;

        default:

            printf("Incorrect Option");

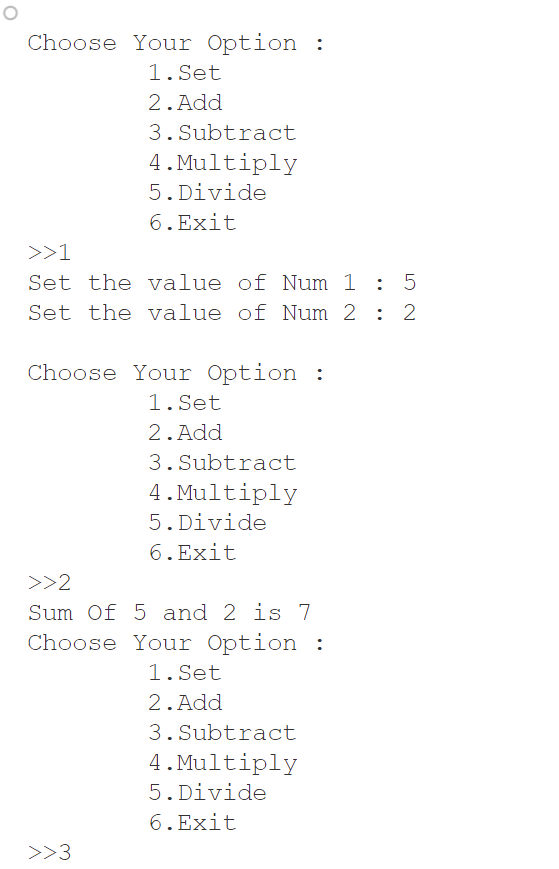
            break;

        }

    }

}

**OUTPUT :**

A screenshot of a computer program

Description automatically generated

**Lab 2 - Searching and Sorting**

**1.**Presence of an element in an Array.

**AIM:**

Write a C++ program to search for the presence of a number in an array.

**ALGORITHM:**

1. Linear Search in array

**INPUT: N** => Number of elements, **search** => Element to search for, **array** => array of elements.

**OUTPUT:** Index of Element if found , **-1** if element not found

* + 1. Get number of elements **N**
    2. Create the array of elements with for loop insertion.
    3. Get the element to search for **search**.
    4. Initialize a for loop to access array elements.
       1. If current element = **Search** , print index
       2. Else continue
    5. Print element not found in array

**CODE:**

// Program to find a number in an array

#include <stdio.h>

#include <stdlib.h>

int main(){

    int \*Array,\*n,\*search,\*var;

    n = (int \*)malloc(sizeof(int));

    search = (int \*)malloc(sizeof(int));

    var = (int \*)malloc(sizeof(int));

    printf("Enter the number of elements in the array : ");

    scanf("%d",n);

    Array = (int \*)malloc(sizeof(int) \* (\*n));

    for(\*var  = 0;\*var<\*n;(\*var)++){

        printf("Enter the number - %d : ",(\*var) + 1);

        scanf("%d",(Array + (\*var)));

    }

    printf("Enter the number to search for : ");

    scanf("%d",search);

    for(\*var = 0;\*var < \*n;(\*var)++){

        if(\*(Array+ (\*var)) == \*search){

            printf("Found The Element at index : %d\n",\*var);

            return 0;

        }

    }

    printf("Element not found in list\n");

    return 0;

}

**OUTPUT:**

A screenshot of a computer code

Description automatically generated

**2.**Sort elements in an Array.

**AIM:**

Write a C++ menu-driven program to sort an array of numbers in ascending or descending order.

**ALGORITHM:**

1. Ascending order bubble sort

**INPUT: array** => array of integers, **N** => Array size

**OUTPUT: array** => Sorted array in ascending order.

* 1. Initialize **first** for loop from **0** to **N-1**
     1. Initialize **second** for loop from **first** to **N**.
        1. If **first > second**:
           1. Swap **First** and **Second**.
  2. Return **array**

1. Descending order bubble sort

**INPUT: array** => array of integers, **N** => Array size

**OUTPUT: array** => Sorted array in ascending order.

* 1. Initialize **first** for loop from **0** to **N-1**
     1. Initialize **second** for loop from **first** to **N**.
        1. If **first < second**:
           1. Swap **First** and **Second**.
  2. Return **array**

**CODE:**

// Program to sort a list in either ascending or descending form

//This program uses selection sort

#include <stdio.h>

#include <stdlib.h>

void ascending(int \**arr*,int \**n*){

    int \*first,\*second,\*tmp;

    first = (int \*)malloc(sizeof(int));

    second = (int \*)malloc(sizeof(int));

    tmp = (int \*)malloc(sizeof(int));

    for(\*first = 0;\*first <\**n* -1;(\*first)++){

        for(\*second = \*first + 1;\*second < \**n*;(\*second)++){

            if(\*(*arr* + (\*first)) > \*(*arr* + (\*second))){

                \*tmp = \*(*arr* + (\*first)) ;

                \*(*arr* + (\*first)) = \*(*arr* + (\*second));

                \*(*arr* + (\*second)) = \*tmp;

            }

        }

    }

}

void descending(int \**arr*,int \**n*){

    int \*first,\*second,\*tmp;

    first = (int \*)malloc(sizeof(int));

    second = (int \*)malloc(sizeof(int));

    tmp = (int \*)malloc(sizeof(int));

    for(\*first = 0;\*first <\**n* -1;(\*first)++){

        for(\*second = \*first + 1;\*second < \**n*;(\*second)++){

            if(\*(*arr* + (\*first)) < \*(*arr* + (\*second))){

                \*tmp = \*(*arr* + (\*first)) ;

                \*(*arr* + (\*first)) = \*(*arr* + (\*second));

                \*(*arr* + (\*second)) = \*tmp;

            }

        }

    }

}

int main(){

    int \*n,\*Array,\*var;

    n = (int \*)malloc(sizeof(int));

    var = (int \*)malloc(sizeof(int));

    printf("Enter the number of elements in the array : ");

    scanf("%d",n);

    Array = (int \*)malloc(sizeof(int) \* (\*n));

    for(\*var = 0; \*var < \*n;(\*var)++){

        printf("Enter the element - %d : ",\*var + 1);

        scanf("%d",Array + (\*var));

    }

    printf("1-Ascending\n2-Descending\n>>");

    scanf("%d",var);

    if(\*var == 1){

        ascending(Array,n);

    }else{

        descending(Array,n);

    }

    printf("Sorted array = ");

    for(\*var = 0; \*var < \*n;(\*var)++){

        printf("%d ",\*(Array + (\*var)));

    }

    printf("\n");

    return 0;

}

**OUTPUT:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**3.**Find the presence of a students Roll No. with Binary Search.

**AIM:**

Imagine a class with a 'n' number of students. Their integer roll numbers are stored in an array in ascending order. There are gaps in the roll number series.  Now, get a roll number from the user and search whether it is present in the array. If present, return the array index + 1. Follow the below approach for searching.

**ALGORITHM:**

**INPUT: N** => Number of students, **find** => The roll number to search for.

**OUTPUT: arr** => Sorted array of roll numbers in ascending order**,**Position of the searched roll number (if found)

1. Get the number of students **N** from the user.
2. Create an array **arr** of size **N** to store the roll numbers.
3. Initialize a loop from **0** to **N-1**:
   1. Get the roll number from the user and store it in array index.
4. Sort in Ascending Order:
   1. Initialize **first** to 0.
   2. Initialize a loop from **first** to **N-2**:
      1. Initialize **second** to **first + 1**.
      2. Initialize a loop from **second** to **N-1**:
         1. If **arr[first] > arr[second]**, swap **arr[first] and arr[second]**
5. Print the sorted array of roll numbers.
6. Get the roll number to search for from the user as **find**.
7. Search For Roll Number:
   1. Initialize **Start** to **0** and **End** to **N-1**.
   2. While **Start <= End**:
      1. Calculate mid as (Start + End) / 2.
      2. If arr[mid] == find, print the position (mid + 1) and return mid.
      3. If arr[mid] < find, update Start to mid + 1.
      4. If arr[mid] > find, update End to mid - 1.
   3. If the loop exits without finding the roll number, print "Roll number not found" and return mid.

**CODE:**

// Program to sort,insert,find student roll numbers

#include <stdio.h>

#include <stdlib.h>

void ascending(int \**arr*,int \**n*){

    int \*first,\*second,\*tmp;

    first = (int \*)malloc(sizeof(int));

    second = (int \*)malloc(sizeof(int));

    tmp = (int \*)malloc(sizeof(int));

    for(\*first = 0;\*first <\**n* -1;(\*first)++){

        for(\*second = \*first + 1;\*second < \**n*;(\*second)++){

            if(\*(*arr* + (\*first)) > \*(*arr* + (\*second))){

                \*tmp = \*(*arr* + (\*first)) ;

                \*(*arr* + (\*first)) = \*(*arr* + (\*second));

                \*(*arr* + (\*second)) = \*tmp;

            }

        }

    }

}

int \*search(int \**array*,int \**n*,int \**find*){

    int \*Start,\*End,\*mid;

    Start = (int \*)malloc(sizeof(int));

    End = (int \*)malloc(sizeof(int));

    \*Start = 0;

    \*End = \**n*;

    while(\*Start <= \*End){

        \*mid = (int)((\*Start + \*End) /2);

        if(\*(*array* + \*mid) == \**find*){

            printf("Found the Roll Number in pos - %d\n",\*mid + 1);

            return mid;

        }else {

            if(\*(*array* + \*mid) < \**find*){

                \*Start = \*mid  + 1;

            }else{

                \*End = \*mid - 1;

            }

        }

    }

    printf("Roll number not found :(\n");

    return mid;

}

int main(){

    int \*n,\*array,\*var,\*find;

    n = (int \*)malloc(sizeof(int));

    printf("Enter the number of students : ");

    scanf("%d",n);

    var = (int \*)malloc(sizeof(int));

    find = (int \*)malloc(sizeof(int));

    array = (int \*)malloc(sizeof(int) \* (\*n));

    for(\*var  = 0;\*var<\*n;(\*var)++){

        printf("Enter the roll - %d : ",(\*var) + 1);

        scanf("%d",(array + (\*var)));

    }

    ascending(array,n);

    printf("Sorted Roll Numbers = ");

    for(\*var = 0; \*var < \*n;(\*var)++){

        printf("%d ,",\*(array + (\*var)));

    }

    printf("\nEnter the Roll of the Person to search for : ");

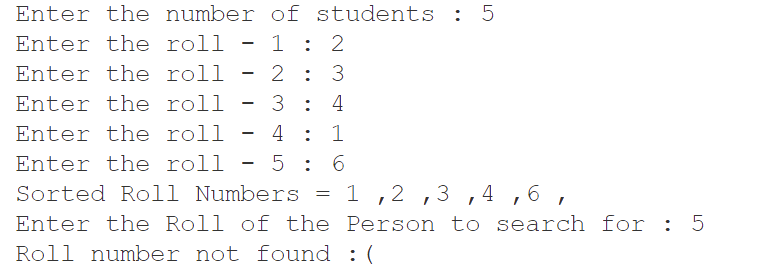
    scanf("%d",find);

    search(array,n,find);

    return 0;

}

**OUTPUT:**



**Lab 3 - List ADT - Array Implementation**

**1.**List ADT using Array.

**AIM:**

Write a C++ menu driven program to implement List ADT using arrays. Maintain proper boundary conditions and follow good coding practices.

**ALGORITHM:**

1. Class Definition:
   * Define a class named **List**.
   * Declare private data members:
     + **C\_Ind**: Current index (initialized to -1).
     + **T\_Ind**: Temporary index (initialized to 0).
     + **L\_arr**: Array to store elements (initialized to size List\_Size\_Given with all elements initialized to 0).
2. Constructor:
   * Define a constructor for the List class:
     + **Initialize C\_Ind to -1**.
3. IsEmpty Method:
   * Define a method named IsEmpty:
     + Check if **C\_Ind** is -1.
     + If true, return 1 (indicating the list is empty).
     + If false, return 0.
4. IsFull Method:
   * Define a method named IsFull:
     + Check if **C\_Ind** is equal to **List\_Size\_Given - 1**.
     + If true, return 1 (indicating the list is full).
     + If false, return 0.
5. InsertBeg Method:
   * Define a method named **InsertBeg** with parameter **val** (value to be inserted at the beginning):
     + Check if the list is not full.
     + If true, shift all elements to the right by one position.
     + Insert the value **val** at index 0.
     + Increment **C\_Ind**.
     + Return 1 (indicating successful insertion).
     + If the list is full, return 0.
6. Append Method:
   * Define a method named Append with parameter val (value to be appended):
     + Check if the list is not full.
     + If true, increment **C\_Ind** and insert the value **val** at index C**\_Ind**.
     + Return 1 (indicating successful append).
     + If the list is full, return 0.
7. Insert Method:
   * Define a method named Insert with parameters **val** (value to be inserted) and **pos** (position to insert):
     + Check if the list is not full.
     + Check if **pos** is within the current range of the list.
     + If true, shift elements to the right from position **pos** onwards.
     + Insert the value **val** at position **pos**.
     + Increment **C\_Ind**.
     + Return 1 (indicating successful insertion).
     + If the list is full, return 0.
     + If **pos** is out of range, return -1.
8. DeleteBeg Method:
   * Define a method named DeleteBeg:
     + Check if the list is not empty.
     + If true, shift all elements to the left by one position.
     + Decrement **C\_Ind**.
     + Return 1 (indicating successful deletion).
     + If the list is empty, return 0.
9. Pop Method:
   * Define a method named Pop:
     + Check if the list is not empty.
     + If true, decrement **C\_Ind**.
     + Return 1 (indicating successful pop).
     + If the list is empty, return 0.
10. Delete Method:

* Define a method named Delete with parameter pos (position to delete):
  + Check if the list is not empty.
  + Check if **pos** is within the current range of the list.
  + If true, shift elements to the left from position **pos + 1** onwards.
  + Decrement **C\_Ind**.
  + Return 1 (indicating successful deletion).
  + If the list is empty, return 0.
  + If pos is out of range, return 0.

1. Search Method:
   * Define a method named Search with parameter **val** (value to search):
     + Check if the list is not empty.
     + **Iterate** through the list to find the position of val.
     + If **val** is found, return its index.
     + If **val** is not found, return -1.
2. Display Method:
   * Define a method named Display:
     + Print all elements of the list.

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#define List\_Size\_Given 10

class **List**{

    private:

        int C\_Ind;

        int T\_Ind = 0;

        int L\_arr[List\_Size\_Given] ={0};

    public:

        List(){

            C\_Ind = -1;

        }

        // Function to Check if the array is empty

        int IsEmpty(){

            if (C\_Ind == -1){

                return 1;

            }else{

                return 0;

            }

        }

        // Funciton to check if array is full

        int IsFull(){

            if(C\_Ind == List\_Size\_Given-1){

                return 1;

            }else{

                return 0;

            }

        }

// Function to shift all elements to the right and insert a new value in index : 0

        int InsertBeg(int *val*){

            if(IsFull()==0){

                for(T\_Ind = C\_Ind;T\_Ind > -1;T\_Ind--){

                    L\_arr[T\_Ind+1]=L\_arr[T\_Ind];

                }

                L\_arr[0]=(int)*val*;

                C\_Ind++;

                return 1;

            }

            return 0;

        }

        // Function to Add an element to that last of the array

        int Append(int *val*){

            if(IsFull()==0){

                C\_Ind++;

                L\_arr[C\_Ind] = *val*;

                return 1;

            }

            return 0;

        }

        // Function to insert an element in the given position

        int Insert(int *val*,int *pos*){

            if(IsFull()==0){

                if(*pos*<=C\_Ind){

                    for(T\_Ind = C\_Ind;T\_Ind >= *pos*;T\_Ind--){

                        L\_arr[T\_Ind+1]=L\_arr[T\_Ind];

                    }

                    L\_arr[*pos*]=*val*;

                    C\_Ind++;

                    return 1;

                }

                return(-1);

            }

            return 0;

        }

        // Function To Delete Beginning element of the array

        int DeleteBeg(){

            if(IsEmpty()==0){

                for(T\_Ind = 1;T\_Ind <= C\_Ind;T\_Ind++){

                    L\_arr[T\_Ind-1]=L\_arr[T\_Ind];

                }

                C\_Ind--;

                return 1;

            }

            return 0;

        }

        // Function to Pop the last element of the array

        int Pop(){

            if(IsEmpty()==0){

                C\_Ind--;

                return 1;

            }

            return 0;

        }

        // Function to delete theelement in the given position

        int Delete(int *pos*){

            if(IsEmpty()==0){

                for(T\_Ind = *pos*+1;T\_Ind <= C\_Ind;T\_Ind++){

                    L\_arr[T\_Ind-1]=L\_arr[T\_Ind];

                }

                C\_Ind--;

                return 1;

            }

            return 0;

        }

// Function to search for an element and return the index of that element

        int Search(int *val*){

            if(IsEmpty()==0){

                for(T\_Ind = 0;T\_Ind<=C\_Ind;T\_Ind++){

                    if(L\_arr[T\_Ind]==*val*){

                        return T\_Ind;

                    }

                }

            }

            return -1;

        }

        // Function to display all the elements of the list

        void Display(){

            printf("\n[");

            for(T\_Ind=0;T\_Ind<C\_Ind;T\_Ind++){

                printf("%d,",L\_arr[T\_Ind]);

            }

            printf("%d]",L\_arr[C\_Ind]);

        }

};

int main(){

**List** a;

    int opt,var,pos,variable;

    while(1==1){

        system("cls");

printf("\n-------------------------------------------------\nOptions : \n\t0-Exit\n\t1-Insert At Beginning\n\t2-Appending\n\t3-Insertion\n\t4-Delete beginning\n\t5-Pop\n\t6-Deletetion\n\t7-Search\n\t8-Display\n>>> ");

        scanf("%d",&opt);

        system("cls");

        switch (opt)

        {

        case 0:

            return 0;

        case 1:

            printf("\nValue : ");

            scanf("%d",&var);

            if(a.InsertBeg(var)==1){

                printf("\nInserted Successful");

            }else{

                printf("\nList is Full :(");

            }

            break;

        case 2:

            printf("Value to append :  ");

            scanf("%d",&var);

            if(a.Append(var)==1){

                printf("\nInserted Successful");

            }else{

                printf("\nList is Full :(");

            }

            break;

        case 3:

            printf("Value to Insert :  ");

            scanf("%d",&var);

            printf("Position to Insert :  ");

            scanf("%d",&pos);

            variable = a.Insert(var,pos);

            if(variable==1){

                printf("\nInserted Successful");

            }else if(variable==0){

                printf("\nList is Full :(");

            }else{

                printf("Invalid Position");

            }

            break;

        case 4:

            if(a.DeleteBeg()==1){

                printf("\nDeleted Successfully");

            }else{

                printf("\nList is Empty :(");

            }

            break;

        case 5:

            if(a.Pop()==1){

                printf("\nPopped Successfully");

            }else{

                printf("\nList is Empty :(");

            }

            break;

        case 6:

            printf("Position to Remove :  ");

            scanf("%d",&pos);

            if(a.Delete(pos)==1){

                printf("\nDeleted Successfully");

            }else{

                printf("\nList is Empty :(");

            }

            break;

        case 7:

            printf("Element to search :  ");

            scanf("%d",&var);

            pos = a.Search(var);

            if(pos!=-1){

                printf("\nFound Element In index :%d",pos);

            }else{

                printf("\nElement Not Found in List");

            }

            break;

        case 8:

            printf("Values in List : ");

            a.Display();

            break;

        default:

            printf("Invalid Choise\n");

            break;

        }

        printf("\n(Press any key to continue)");

        getchar();

        getchar();

    }

}

**OUTPUT:**