EXPERIMENT-1

AIM: Write a program to insert and delete an element from an array.

ALGORITHM:

SOURCE CODE:

```
#include <iostream>
using namespace std;

void insertElement(int arr[], int n, int x, int pos)
{
    for (int i = n - 1; i >= pos; i--){
        arr[i + 1] = arr[i];}
    arr[pos] = x;
}
int deleteElement(int arr[], int n, int key){
    int pos;
    for (int i = 0; i < n; i++){
        if (arr[i] == key){
```

```
            pos =i;
            break;}
        else{
            pos=-1; }
    }
    if (pos==-1){
        cout<<"NOT FOUND";
        return n;}
    for (int i = pos; i < n - 1; i++){
        arr[i] = arr[i + 1];}
    return n - 1;
}
int main() {
     int arr[100],size;
     int x,p;
     cout<<"size of the array: "; cin>>size;
     cout<<"elements of the array: "<<endl;
     for (int i =0;i<size;i++){
        cin>>arr[i]; }
    cout<<"ARRAY: "; for (int i=0;i<size;i++){
        cout<<arr[i]<<" ";
    }
    cout<<endl;
    cout<<"Enter the value to be inserted and its position: "; cin>>x>>p;
    insertElement(arr, size, x, p);
    size++;
    cout<<"NEW ARRAY : ";
    for (int i = 0; i < size; i++){
        cout<<arr[i]<<" ";
    }
    cout<<endl;
    cout<<"Enter the value to be deleted "; cin>>x;
    size=deleteElement(arr, size,x);

    cout << "\n\nArray after deletion\n";
    for (int i = 0; i < size; i++){
        cout << arr[i] << " ";
    }
    return 0;
}
```

EXPERIMENT-1

AIM: Write a program to insert and delete an element from an array.

OUTPUT:

```
/tmp/YLtdgCsTkj.o
size of the array: 5
elements of the array:
14
17
19
13
16
ARRAY: 14 17 19 13 16
Enter the value to be inserted and its position: 24 2
NEW ARRAY : 14 17 24 19 13 16
Enter the value to be deleted 17
Array after deletion
14 24 19 13 16
```

EXPERIMENT-2

AIM: Implement sparse matrix using array. Description of program:

a. Read a 2D array from the user.
b. Store it in the sparse matrix form, use array of structures.
c. Print the final array.

ALGORITHM:

SOURCE CODE:

```
#include<iostream>
using namespace std;

int main()
{
   int sparseMatrix[5][5] =
   {
      {0 , 0 , 3 , 0 , 0 },
      {0 , 8 , 0 , 0 , 1 },
      {0 , 0 , 0 , 7 , 0 },
      {0 , 5 , 0 , 0 , 0 },
      {6 , 0 , 9 , 0 , 0 }
   };
```

```
    for (int row = 0; row < 5; row++){
        for (int column = 0; column < 5; column++){
            cout<<sparseMatrix[row][column]<<" ";}
        cout<<endl;}
    cout<<endl;
    int size = 0;
    for (int row = 0; row < 5; row++){
        for (int column = 0; column < 5; column++){
            if (sparseMatrix[row][column] != 0){
                size++;
            }}}
    int resultMatrix[size][3];
    int k = 0;
    for (int row = 0; row < 5; row++)
        for (int column = 0; column < 5; column++)
            if (sparseMatrix[row][column] != 0)
            {
                resultMatrix[k][0] = row;
                resultMatrix[k][1] = column;
                resultMatrix[k][2] = sparseMatrix[row][column];
                k++;}

    cout<<"R-Row C-Column V-Value"<<endl<<endl;
    cout<<"R C V"<<endl;
    for (int row=0; row<size; row++)
    {
        for (int column = 0; column<3; column++){
            cout<<resultMatrix[row][column]<<" ";}
        cout<<endl;
    }
    return 0;
}
```

EXPERIMENT-2

AIM: Implement sparse matrix using array. Description of program:

a. Read a 2D array from the user.
b. Store it in the sparse matrix form, use array of structures.
c. Print the final array.

OUTPUT:

```
0 0 3 0 0
0 8 0 0 1
0 0 0 7 0
0 5 0 0 0
6 0 9 0 0

R-Row C-Column V-Value

R C V
0 2 3
1 1 8
1 4 1
2 3 7
3 1 5
4 0 6
4 2 9
```

EXPERIMENT – 3

AIM: Write a program to demonstrate the functioning of the pointers.

ALGORITHM:

SOURCE CODE:

```
#include <iostream>
using namespace std;
int main() {

    int a;
    cout<<"Enter any integer value: ";
    cin>>a;

    int *b;
    b = &a;
    int **c;
    c = &b;
    cout<<"The value stored in the pointers are:"<<endl;

    cout<<"a:"<<a<<endl;
    cout<<"b: "<<b<<endl;
    cout<<"c: "<<c<<endl;

    cout<<"&a: "<<&a<<endl;
    cout<<"&b: "<<&b<<endl;

    cout<<"*a: "<<*b<<endl;
    cout<<"*b: "<<*c<<endl;

    cout<<"**c: "<<**c<<endl;
    return 0;
}
```

EXPERIMENT-3

AIM: Write a program to demonstrate the functioning of the pointers.

OUTPUT:

```
/tmp/3JdWdCldQf.o
Enter any integer value: 12
The value stored in the pointers are:
a:12
b: 0x7ffd8673f694
c: 0x7ffd8673f698
&a: 0x7ffd8673f694
&b: 0x7ffd8673f698
*a: 12
*b: 0x7ffd8673f694
**c: 12
```

EXPERIMENT-4

AIM: Perform Linear Search and Binary Search on an array. Description of programs:

a. Read an array of type integer.
b. Input element from user for searching.
c. Search the element by passing the array to a function and then returning the position of the element from the function else return -1 if the element is not found.
d. Display the position where the element has been found.

ALGORITHM:

SOURCE CODE:

```
#include <iostream>
using namespace std;

int linearSearch(int arr[10],int size,int e){
   for (int i=0;i<size;i++){
      if (arr[i] == e){
      return i;
      }
   }
   return -1;
}
```

```
int binarySearch(int arr[10],int size,int a){
    int min= 0, max= size-1;int mid;
    while(min<=max){
        mid = (min + max)/2;
        if (a == arr[mid]){
            return mid;
        }
        else if (a > arr[mid]){
            min = mid + 1;
        }

        else{
            max = mid - 1;
        }
    }
}
int main() {

    int arr[] = {1,2,4,6,9,12,14,15,19,23,27,34};
    int a,r,t;
    int size = sizeof(arr)/sizeof(int);
    cout<<"The input array is: ";
    for (int i=0;i<size;i++){
        cout<<arr[i]<<" ";
    }
    cout<<endl;
    cout<<"Element to search: "; cin>>a;
    cout<<"USING: 1.LINEAR SEARCH  2.BINARY SEARCH "<<endl;cin>>t;
    if(t==1){
        r = linearSearch(arr,size,a);
        cout<<"The element is found at index "<<r;
    }
    if(t==2){
        r=binarySearch(arr,size,a);
        cout<<"The element is found at index "<<r;
    }
    return 0;
}
```

EXPERIMENT-4

AIM: Perform Linear Search and Binary Search on an array. Description of programs:

a. Read an array of type integer.
b. Input element from user for searching.
c. Search the element by passing the array to a function and then returning the position of the element from the function else return -1 if the element is not found.
d. Display the position where the element has been found.

OUTPUT:

```
/tmp/3JdWdCldQf.o
The input array is: 1 2 4 6 9 12 14 15 19 23 27 34
Element to search: 23
USING: 1.LINEAR SEARCH  2.BINARY SEARCH
2
The element is found at index 9
```

```
/tmp/3JdWdCldQf.o
The input array is: 1 2 4 6 9 12 14 15 19 23 27 34
Element to search: 11
USING: 1.LINEAR SEARCH  2.BINARY SEARCH
1
The element is found at index -1
```

EXPERIMENT-5

AIM: Create a linked list with nodes having info about a student and perform-

I. Insert a new node at specified position.
II. Delete of a node with the roll number of student specified.
III. Reversal of that linked list

ALGORITHM:

SOURCE CODE:

```
#include <bits/stdc++.h>
using namespace std;
class Node {
public:
    int roll;
    string Name;
```

```
    string Dept;
    int Marks;
    Node* next;
};

Node* head = new Node();

bool check(int x)
{
    if (head == NULL)
        return false;
    Node* t = new Node;
    t = head;
    while (t != NULL) {
        if (t->roll == x)
            return true;
        t = t->next;
    }

    return false;
}
void Insert_Record(int roll, string Name,string Dept, int Marks)
{

    if (check(roll)) {
        cout << "Student with this \n"<< "record Already Exists\n";
        return;
    }

    Node* t = new Node();
```

```
    t->roll = roll;
    t->Name = Name;
    t->Dept = Dept;
    t->Marks = Marks;
    t->next = NULL;
    if (head == NULL|| (head->roll >= t->roll)) {
        t->next = head;
        head = t;
    }
    else {
        Node* c = head;
        while (c->next != NULL && c->next->roll < t->roll) {
            c = c->next;
        }
        t->next = c->next;
        c->next = t;
    }
    cout << "Record Inserted "<< "Successfully\n";
}
int Delete_Record(int roll)
{
    Node* t = head;
    Node* p = NULL;
  if (t != NULL&& t->roll == roll) {
        head = t->next;
        delete t;
        cout << "Record Deleted "<< "Successfully\n";
        return 0; }
    while (t != NULL && t->roll != roll) {
        p = t;
```

```
        t = t->next; }
    if (t == NULL) {
        cout << "Record does not Exist\n";
        return -1; }
    else{
        p->next = t->next;
        delete t;
        cout << "Record Deleted "
            << "Successfully\n";
        return 0;}
}

void Show_Record()
{
    Node* p = head;
    if (p == NULL) {
        cout << "No Record "<< "Available\n"; }
    else {
        cout << "Index\tName\tCourse"<< "\tMarks\n";
        while (p != NULL) {
            cout << p->roll << "    \t" << p->Name << "\t" << p->Dept << "\t" << p->Marks <<
endl;
            p = p->next; }
    }
}

Node* reverse()
    {
        Node* current = head;
        Node *prev = NULL, *next = NULL;
        while (current != NULL) {
```

```
            next = current->next;
            current->next = prev;
            prev = current;
            current = next; }
        head = prev;
        return head;
    }

int main()
{
    head = NULL;
    string Name, Course;
    int Roll, Marks;
    while (true) {
    cout << "1 create a new Record\n"
                "2 delete a student record\n"
                "3 view all students record\n"
                "4 revese the link list\n"
                "5 Exit\n";
        cout << "\nEnter your Choice\n";
        int Choice;
        cin >> Choice;
        if (Choice == 1) {
            cout << "Enter Name of Student\n";
            cin >> Name;
            cout << "Enter Roll Number of Student\n";
            cin >> Roll;
            cout << "Enter Course of Student \n";
            cin >> Course;
            cout << "Enter Total Marks of Student\n";
```

```
            cin >> Marks;
            Insert_Record(Roll, Name, Course, Marks);
        }
        else if (Choice == 2) {
            cout << "Enter Roll Number of Student whose record is to be deleted\n";
            cin >> Roll;
            Delete_Record(Roll);
        }
        else if (Choice == 3) {
            Show_Record();
        }
        else if (Choice == 4) {
          Node*newhead=reverse();
          Show_Record();
        }
        else if (Choice == 5) {
            exit(0);
        }
        else {
            cout << "Invalid Choice " << "Try Again\n";
        }
    }
    return 0;
}
```

EXPERIMENT-5

AIM: Create a linked list with nodes having info about a student and perform-

I. Insert a new node at specified position.
II. Delete of a node with the roll number of student specified.
III. Reversal of that linked list

OUTPUT:

```
1 create a new Record
2 delete a student record
3 view all students record
4 revese the link list
5 Exit

Enter your Choice
1
Enter Name of Student
Gautam
Enter Roll Number of Student
41
Enter Course of Student
CSE
Enter Total Marks of Student
499
Record Inserted Successfully
```

```
Enter your Choice
3
Index    Name     Course  Marks
41       Gautam   CSE 499
```

```
Enter Name of Student
Ajay
Enter Roll Number of Student
31
Enter Course of Student
IT
Enter Total Marks of Student
436
Record Inserted Successfully
```

```
Enter your Choice
2
Enter Roll Number of Student whose record is to be deleted
31
Record Deleted Successfully
```

EXPERIMENT-6

AIM: Create doubly linked list with nodes having information about an employee and perform Insertion at front of doubly linked list and perform deletion at end of that doubly linked list.

ALGORITHM:

SOURCE CODE:

```
#include<iostream>
using namespace std;

class node{
```

```
    public:
    string name;
    int phoneno;
    int ID;
    node* prev;
    node* next;
    };
node* head=new node;
bool check(int x){
    if (head == NULL)
        return false;
    node* t = new node;
    t = head;
    while (t != NULL) {
        if (t->ID == x)
            return true;
        t = t->next;}
    return false;
}
void push(string name,int id,int phone){
    if (check(id)) {
        cout << "Employee with this record Already Exists\n";
        return;}
    node* new_rec=new node;
    new_rec->name=name;
    new_rec->phoneno=phone;
    new_rec->ID=id;
    new_rec->next=head;
    new_rec->prev=NULL;
```

```
    if (head!=NULL){
        head->prev=new_rec;
        new_rec->next=head;}
    head=new_rec;
}

void pop(int id){
    node* del=new node;
    del=head;
    while(del!=NULL){
        if (del->ID== id)
            break;
        del = del->next;}
    if (head == NULL || del == NULL)
        return;

    if (head == del){
        head = del->next;}
    if (del->next != NULL)
        del->next->prev = del->prev;
    if (del->prev != NULL)
        del->prev->next = del->next;
    free(del);
    return;
}

int main(){
    head=NULL;
    string name;
    int id,phone;
```

```
    while(true){
        cout<<"1 Insert a record"<<endl;
        cout<<"2 Delete a record"<<endl;
        cout<<"3 Exit"<<endl;
        int ch;
        cout<<"Enter your choice:"<<endl;
        cin>>ch;
        if (ch==1){
            cout<<"Name of Employee:"<<endl;
            cin>>name;
            cout<<"Phone number:"<<endl;
            cin>>phone;
            cout<<"Employee id:"<<endl;
            cin>>id;
        }
        else if (ch==2){
            cout<<"Enter id to delete:"<<endl;
            int n;
            cin>>n;
            pop(n);}
        else if (ch == 3) {
            break;}
        else {
            cout << "Invalid Choice " << "Try Again\n";
            break;}
    }
    return 0;
}
```

EXPERIMENT-6

AIM: Create doubly linked list with nodes having information about an employee and perform Insertion at front of doubly linked list and perform deletion at end of that doubly linked list.

OUTPUT:

```
1 Insert a record
2 Delete a record
3 Exit
Enter your choice:
1
Name of Employee:
Vivek
Phone number:
1415112
Employee id:
12
```

```
Name of Employee:
Ajay
Phone number:
4154135
Employee id:
25
```

```
Enter your choice:
2
Enter id to delete:
25
```

EXPERIMENT-7

Aim: Implement two stacks using a single array

ALGORITHM:

SOURCE CODE:

```
#include <stdio.h>
#include <liamits.h>
#include<iostream>
using namespace std;
#define LEFT_STACK 0
#define RIGHT_STACK 1

struct st {
    int array[100];
```

```
    int top1, top2;
} st;

void initialize() {
    st.top1 = -1;
    st.top2 = 100;
}
void push(int stack, int num) {
    if(stack == LEFT_STACK) {
        if (st.top1 < st.top2-1) {
            st.top1++;
            st.array[st.top1] = num;
        } else {
            printf("Left Stack Full");
            return;}
    } else if(stack == RIGHT_STACK) {
       if (st.top1 < st.top2-1) {
           st.top2--;
           st.array[st.top2] = num;
       } else {
           printf("Right Stack Full");
           return;}}
}

int pop(int stack) {
    if(stack == LEFT_STACK) {
        if(st.top1 >= 0){
            return st.array[st.top1--];
        } else {
            printf("Left Stack is Empty");
            return INT_MIN;}
        }else if(stack == RIGHT_STACK) {
        if(st.top2 <= 99){
        return st.array[st.top2++];
```

```
        } else {
            printf("Right Stack is Empty");
            return INT_MIN;}}
}

int main() {
    initialize();
    int a,b,c;
    while (true){
        cout<<"1.Left Stack  2.Right Stack  3.EXIT"<<endl;
        cout<<"CHOOSE: ";cin>>a;
        if (a==1){
            cout<<"1.Push  2.Pop"<<endl;
            cout<<"CHOOSE: ";cin>>b;
            if (b==1){
                cout<<"Enter Number:";cin>>c;
                push(LEFT_STACK,c);
            }
            else if(b==2){
                printf("Pop from left stack %d\n", pop(LEFT_STACK));}
        }
        else if(a==2){
            cout<<"1.Push  2.Pop"<<endl;
            cout<<"CHOOSE: ";cin>>b;
            if (b==1){
                cout<<"Enter Number:";cin>>c;
                push(RIGHT_STACK,c);}
            else if(b==2){
                printf("Pop from right stack %d\n", pop(RIGHT_STACK));}
        }
        else {
            exit(0);}}
    return 0;
}
```

EXPERIMENT-7

Aim: Implement two stacks using a single array.

OUTPUT:

```
1.Left Stack  2.Right Stack  3.EXIT
CHOOSE: 1
1.Push  2.Pop
CHOOSE: 1
Enter Number:12
1.Left Stack  2.Right Stack  3.EXIT
CHOOSE: 2
1.Push  2.Pop
CHOOSE: 1
Enter Number:14
1.Left Stack  2.Right Stack  3.EXIT
CHOOSE: 1
1.Push  2.Pop
CHOOSE: 2
Pop from left stack 12
1.Left Stack  2.Right Stack  3.EXIT
CHOOSE: 2
1.Push  2.Pop
CHOOSE: 2
Pop from right stack 14
```

EXPERIMENT-8

Aim: Create a stack and perform Push , Pop , and Display on stack using Linked List.

ALGORITHM:

SOURCE CODE:

```
#include <iostream>
using namespace std;
struct Node {
   int data;
   struct Node *next;
};
struct Node* top = NULL;
```

```
void push(int val) {
  struct Node* newnode = (struct Node*) malloc(sizeof(struct Node));
  newnode->data = val;
  newnode->next = top;
  top = newnode;
}
void pop() {
  if(top==NULL)
  cout<<"Stack Underflow"<<endl;
  else {
    cout<<"The popped element is "<< top->data <<endl;
    top = top->next;
  }
}
void display() {
  struct Node* ptr;
  if(top==NULL)
  cout<<"EMPTY STACK";
  else {
    ptr = top;
    cout<<"Elements : ";
    while (ptr != NULL) {
      cout<< ptr->data <<" ";
      ptr = ptr->next;
    }
  }
  cout<<endl;
}
int main() {
  int ch, val;
  cout<<"1 Push"<<endl;
  cout<<"2 Pop"<<endl;
  cout<<"3 Display"<<endl;
  cout<<"4 Exit"<<endl;
```

```
  do {
    cout<<"Enter choice: ";
    cin>>ch;
    switch(ch) {
      case 1: {
        cout<<"Enter value to be pushed:";
        cin>>val;
        push(val);
        break;
      }
      case 2: {
        pop();
        break;
      }
      case 3: {
        display();
        break;
      }
      case 4: {
        cout<<"Exit"<<endl;
        break;
      }
      default: {
        cout<<"Invalid Choice"<<endl;
      }
    }
  }while(ch!=4);
  return 0;
}
```

EXPERIMENT-8

Aim: Create a stack and perform Push , Pop , and Display on stack using Linked List.

OUTPUT:

```
1 Push
2 Pop
3 Display
4 Exit
Enter choice: 1
Enter value to be pushed:14
Enter choice: 1
Enter value to be pushed:31
Enter choice: 2
The popped element is 31
```