INSTITUTE OF TECHNOLOGY AND MANAGEMENT
SKILLS UNIVERSITY,
KHARGHAR, NAVI MUMBAI

DATA STRUCTURES & ALGORITHMS
PROGRAMMING LAB

Prepared by:

Name of Student :Sparsh Sharma

Roll No: 37

Batch: 2023-27

Dept. of CSE

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INSTITUTE OF TECHNOLOGY AND MANAGEMENT
SKILLS UNIVERSITY,
KHARGHAR, NAVI MUMBAI

CERTIFICATE

This is to certify that Mr. / Ms. ____**Sparsh Sharma**____Roll No. ___**37**____ Semester ___ of B.Tech Computer Science & Engineering, ITM Skills University, Kharghar, Navi Mumbai , has completed the term work satisfactorily in subject _____ for the academic year 2024_ - 2025___ as prescribed in the curriculum.

Place: _____

Date: _____

Subject I/C **HOD**

Exp. No	List of Experiment	Date of Submission	Sign
1	Implement Array and write a menu driven program to perform all the operation on array elements		
2	Implement Stack ADT using array.		
3	Convert an Infix expression to Postfix expression using stack ADT.		
4	Evaluate Postfix Expression using Stack ADT.		
5	Implement Linear Queue ADT using array.		
6	Implement Circular Queue ADT using array.		
7	Implement Singly Linked List ADT.		
8	Implement Circular Linked List ADT.		
9	Implement Stack ADT using Linked List		
10	Implement Linear Queue ADT using Linked List		
11	Implement Binary Search Tree ADT using Linked List.		
12	Implement Graph Traversal techniques: a) Depth First Search b) Breadth First Search		
13	Implement Binary Search algorithm to search an element in an array		
14	Implement Bubble sort algorithm to sort elements of an array in ascending and descending order		

Name of Student: ____Sparsh Sharma____

Roll Number: _____37_____

Experiment No: 1

Title: Implement Array and write a menu driven program to perform all the operation on array elements

Theory: This program allows you to perform various operations on an array, such as displaying the array elements, inserting elements at the beginning, end, before, or after an element, deleting elements from the beginning, end, before, or after an element, searching for an element, and exiting the program.

Code:

```
// Implement Array and write a menu driven program to perform all
the operation on array elements
// menu driven array operations program - 1.display array 2.insert
at beginning 3.insert at end 4.insert before an element 5.insert
after an element 6.delete at beginning 7.delete at end 8.delete
before an element 9.delete after an element 10.search an element
11.number of elements

#include <iostream>
#include <algorithm>
using namespace std;
void displayArray(int &a, int arr[])
{
    int count = 0;
    cout << "Array: ";
    for (int i = 0; i < a; i++)
    {
        if (arr[i] != -1)
        {
            cout << arr[i] << " ";
            count++;
        }
        else
        {
```

```
            break;
        }
    }
    cout<<endl
     <<"Number of elements: "<<count<<endl;
}


void insertAtfirst(int &a, int arr[])
{
    if (a >= 45)
    {
        cout<<"Array is full. You Cannot insert at the First ."<<
endl;
        return;
    }
    int b,count=0;
    cout<<"Enter beginning detail:";
    cin>>b;
    for(int i=a-1;i>=0;i--)
    {
        arr[i+1] = arr[i];
    }
    arr[0]=b;
    a++;
    for(int i =0;i<a;i++)
    {
        if (arr[i] == -1)
        {
            break;
        }
        else
        {
            count++;
        }
    }
    for (int i = 0; i < count; i++)
    {
        cout << arr[i] << " ";
    }
}

void insertAtEnd(int &a, int arr[])
{
    if (a>=45)
    {
        cout<<"Array is full. Cannot insert at the end." << endl;
        return;
    }
```

```
    int b,count = 0;
    cout<<"Enter end detail: ";
    cin>>b;
    for(int i=0;i<a;i++)
    {
        if(arr[i] == -1)
        {
            break;
        }
        else
        {
            count++;
        }
    }
    arr[count]=b;
    count++;
    cout << "Size of arra"<<count<<endl;
    a=count;
    for(int i=0; i<count; i++)
    {
        cout<<arr[i]<< " ";
    }
}

void insertAtIndexLocation(int &c, int arr[]) // function to insert
element at specified index location in the array
{
    int a,b;
    cout<<"Enter updated detail: ";
    cin>>b;
    cout<<"Enter index location: ";
    cin>>a;
    int i=c-1;
    while(i>=a)
    {
        arr[i + 1] = arr[i];
        i--;
    }
    c++;
    arr[a]=b;
    for (int i=0;i<c;i++)
    {
        cout<<arr[i]<< " ";
    }
}

void insertBeforeElement(int &c, int arr[])
{
    int b, a, pos;
```

```
    cout << "Enter updated detail: ";
    cin >> b;
    cout << "Enter element to insert before: ";
    cin >> a;
    for (int i = 0; i < c; i++)
    {
        if (arr[i] == a)
        {
            pos = i;
        }
    }
    for (int i = c - 1; i >= pos; i--)
    {
        arr[i + 1] = arr[i];
    }
    c++;
    arr[pos] = b;
    for (int i = 0; i < c; i++)
    {
        cout << arr[i] << " ";
    }
}

void insertAfterElement(int &c, int arr[])
{
    int b, a, pos;
    cout << "Enter updated detail: ";
    cin >> b;
    cout << "Enter element to insert after: ";
    cin >> a;
    for (int i = 0; i < c; i++)
    {
        if (arr[i] == a)
        {
            pos = i;
        }
    }
    for (int i = c - 1; i > pos; i--)
    {
        arr[i + 1] = arr[i];
    }
    c++;
    arr[pos + 1] = b;
    for (int i = 0; i < c; i++)
    {
        cout << arr[i] << " ";
    }
}
```

```
void deleteFromBegin(int &c, int arr[]) // function to delete
element from beginning of the array
{
    for (int i = 0; i < c; i++)
    {
        arr[i] = arr[i + 1];
    }
    c--;
    for (int i = 0; i < c; i++)
    {
        cout << arr[i] << " ";
    }
}

void deleteFromEnd(int &c, int arr[])
{
    if (c <= 0)
    {
        cout << "Array is empty. Cannot delete from the end." <<
endl;
        return;
    }
    arr[c - 1] = arr[c];
    c--;
    for (int i = 0; i < c; i++)
    {
        cout << arr[i] << " ";
    }
}

void deleteBeforeElement(int &c, int arr[])
{
    int b, pos;
    cout << "Enter element to delete after it: ";
    cin >> b;
    for (int i = 0; i < c; i++)
    {
        if (arr[i] == b)
        {
            pos = i;
        }
    }
    for (int i = pos - 1; i < c; i++)
    {
        arr[i] = arr[i + 1];
    }
    c--;
    for (int i = 0; i < c; i++)
    {
```

```
        cout << arr[i] << " ";
    }
}

void deleteAfterElement(int &c, int arr[])
{
    int b, pos;
    cout << "Enter element to delete after it: ";
    cin >> b;
    for (int i = 0; i < c; i++)
    {
        if (arr[i] == b)
        {
            pos = i;
        }
    }
    for (int i = pos + 1; i < c; i++)
    {
        arr[i] = arr[i + 1];
    }
    c--;
    for (int i = 0; i < c; i++)
    {
        cout << arr[i] << " ";
    }
}

void deleteFromArray(int &a, int arr[])
{
    int b, pos;
    cout << "Enter element to delete: ";
    cin >> b;
    for (int i = 0; i < a; i++)
    {
        if (arr[i] == b)
        {
            pos = i;
        }
    }
    for (int i = pos; i < a; i++)
    {
        arr[i] = arr[i + 1];
    }
    a--;
    for (int i = 0; i < a; i++)
    {
        cout << arr[i] << " ";
    }
}
```

```
void searchElement(int &a, int arr[]) // function to search an
element in the array
{
    int b, count = 0;
    cout << "Enter element to search: ";
    cin >> b;
    for (int i = 0; i < a; i++)
    {
        if (arr[i] == b)
        {
            cout << "Element found at index " << i << endl;
            count++;
        }
    }
    if (count == 0)
    {
        cout << "Element not found" << endl;
    }
}

int main()
{
    int arr[46], n, choice;
    fill_n(arr, 46, -1);

    cout << "Enter number of students' details you want to enter
(less than 45): ";
    cin >> n;
    while (n >= 45 || n <= 0)
    {
        cout << "Invalid size. Enter a valid size" << endl;
        cin >> n;
    }
    for (int i = 0; i < n; i++)
    {
        cout << "Enter detail: ";
        cin >> arr[i];
    }

    char ans = 'y';
    while (ans == 'y')
    {
        cout << "Enter your choice:\n1. Insert element at
beginning\n2. Insert element at end\n3. Insert element at a
particular index position\n4. Insert element before an element\n5.
Insert element after an element\n6. Delete element from
beginning\n7. Delete element from end\n8. Delete element before a
particular element\n9. Delete element after a particular
```

```
element\n10. Search an element\n11. Delete element from array\n12.
Display array\n13. Exit\n";
        cin >> choice;

        switch (choice)
        {
        case 1:
            insertAtfirst(n, arr);
            break;
        case 2:
            insertAtEnd(n, arr);
            break;
        case 3:
            insertAtIndexLocation(n, arr);
            break;
        case 4:
            insertBeforeElement(n, arr);
            break;
        case 5:
            insertAfterElement(n, arr);
            break;
        case 6:
            deleteFromBegin(n, arr);
            break;
        case 7:
            deleteFromEnd(n, arr);
            break;
        case 8:
            deleteBeforeElement(n, arr);
            break;
        case 9:
            deleteAfterElement(n, arr);
            break;
        case 10:
            searchElement(n, arr);
            break;
        case 11:
            deleteFromArray(n, arr);
            break;
        case 12:
            displayArray(n, arr);
            break;
        case 13:
            cout << "Exiting..." << endl;
            return 0;
        default:
            cout << "Invalid choice\n";
        }
        cout << "Want to perform another operation? (y/n): ";
```

```
        cin >> ans;
    }
    return 0;
}
```

Output: (screenshot)

```
Enter number of students' details you want to enter (less than 45): 3
Enter detail: 12
Enter detail: 23
Enter detail: 34
Enter your choice:
1. Insert element at beginning
2. Insert element at end
3. Insert element at a particular index position
4. Insert element before an element
5. Insert element after an element
6. Delete element from beginning
7. Delete element from end
8. Delete element before a particular element
9. Delete element after a particular element
10. Search an element
11. Delete element from array
12. Display array
13. Exit
2
Enter end detail: 67
Size of array: 4
12 23 34 67 Want to perform another operation? (y/n):
```

Test Case: Any two (screenshot)

```
Enter your choice:
1. Insert element at beginning
2. Insert element at end
3. Insert element at a particular index position
4. Insert element before an element
5. Insert element after an element
6. Delete element from beginning
7. Delete element from end
8. Delete element before a particular element
9. Delete element after a particular element
10. Search an element
11. Delete element from array
12. Display array
13. Exit
12
Array: 23 34 67
Number of elements: 3
Want to perform another operation? (y/n):
```

```
Want to perform another operation? (y/n): y
Enter your choice:
1. Insert element at beginning
2. Insert element at end
3. Insert element at a particular index position
4. Insert element before an element
5. Insert element after an element
6. Delete element from beginning
7. Delete element from end
8. Delete element before a particular element
9. Delete element after a particular element
10. Search an element
11. Delete element from array
12. Display array
13. Exit
7
23 34 Want to perform another operation? (y/n):
```

Conclusion:

The given C++ program demonstrates a menu-driven approach for performing various operations on an array, including insertion, deletion, and searching.

Name of Student: ____Sparsh Sharma____

Roll Number: _____37_____

Experiment No: 2

Title: Implement Stack ADT using array.

Theory:

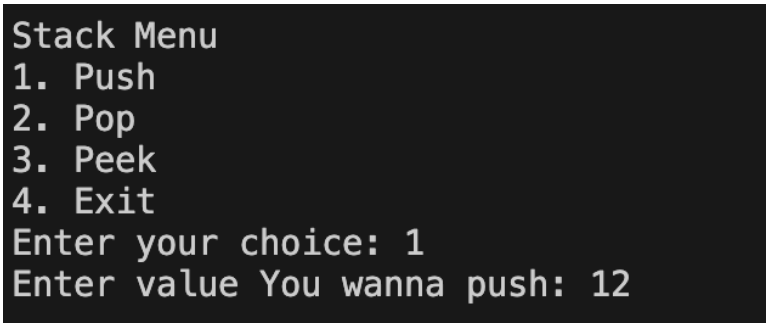
This code implements a stack data structure using an array. It includes methods to **push elements** onto the stack, **pop elements** off the stack, and **peek** at the top element. The main function provides a menu-driven interface for these operations, allowing the user to interact with the stack.

Code:

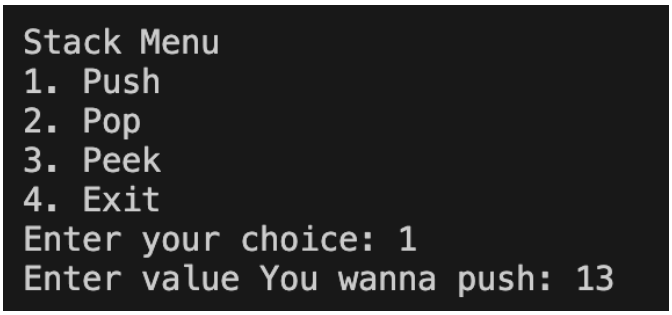
```
#include <iostream>
using namespace std;
#define max_size 100

class Stack {
    int top;
    int arr[max_size];
public:
    Stack(){
        top-1;
    }
    bool empty() {
        return top==-1;
    }
    bool full() {
        return top== max_size - 1;
    }
    void push(int val){
        if (full()){
            cout << "Stack Overflow\n";
            return;
        }
        arr[++top]=val;
    }
    int pop(){
        if (empty()) {
            cout << "Stack Underflow\n";
            return -1;
        }
        return arr[top--];
```

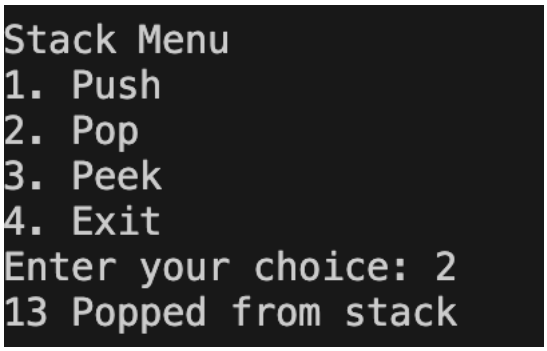
```
    }
    int peek() {
        if (empty()){
            cout<<"Stack is empty\n";
            return -1;
        }
        return arr[top];
    }
};
int main(){
    Stack z;
    int choice, val;
    do{
        cout<<"\nStack Menu\n";
        cout<<"1. Push\n";
        cout<<"2. Pop\n";
        cout<<"3. Peek\n";
        cout<<"4. Exit\n";
        cout<<"Enter your choice: ";
        cin>>choice;

        switch (choice){
            case 1:
                cout<<"Enter value You wanna push: ";
                cin>>val;
                z.push(val);
                break;
            case 2:
                cout<<z.pop() << " Popped from stack\n";
                break;
            case 3:
                cout<<"Top element is : " << z.peek() << endl;
                break;
            case 4:
                cout<<"Exiting program\n";
                break;
            default:
                cout<<"Worng choice\n";
        }
    }while(choice != 4);
    return 0;
}
```

Output: (screenshot)



Test Case: Any two (screenshot)





Conclusion:

The conclusion is a stack using an array, providing basic stack operations like push, pop, and peek. It features a menu-driven interface for user interaction

Name of Student: ____Sparsh Sharma____

Roll Number: _____37_____

Experiment No: 3

Title: Convert an Infix expression to Postfix expression using stack ADT.

Theory:A stack is a data structure that follows the Last In, First Out (LIFO) principle and can be implemented using a Linked List or an Array. It consists of a variable named Top that points to the topmost element of the stack. The main operations in a stack are:

- Push: Insert an element at the top of the stack.
- Pop: Delete and return the top element of the stack.
- Peek: Return the topmost element of the stack without removing it.

One common use of a stack is to convert an infix expression to a postfix expression by pushing operators and brackets onto the stack and operands to the expression. When a closing bracket is encountered, elements are popped from the stack to the expression based on operator precedence.

Code:

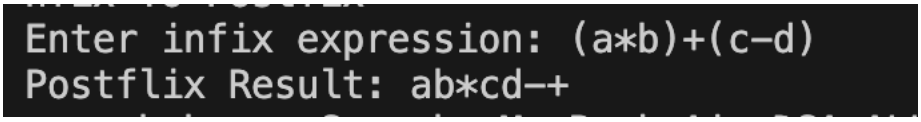
```
// Convert an Infix expression to Postfix expression using stack
ADT.
#include <iostream>
using namespace std;

int precedence(char op)
{
    if (op == '+' || op == '-')
    {
        return 1;
    }
    else if (op == '*' || op == '/' || op == '%')
    {
        return 2;
    }
    else
    {
```

```
        return 0;
    }
}

int main()
{
    string exp, result = "";
    char stack[100];
    int top = -1;
    cout << "Enter infix expression: ";
    getline(cin, exp);
    int n = exp.length();
    char express[n + 2];
    express[0] = '(';
    for (int i = 0; i < n; i++)
    {
        express[i + 1] = exp[i];
    }
    express[n + 1] = ')';
    for (int i = 0; i < n + 2; i++)
    {
        if (express[i] == '(')
        {
            top++;
            stack[top] = express[i];
        }
        else if (express[i] == ')')
        {
            while (stack[top] != '(' && top > -1)
            {
                result += stack[top];
                top--;
            }
            top--;
        }
        else if ((express[i] >= 'a' && express[i] <= 'z') ||
(express[i] >= 'A' && express[i] <= 'Z') || (express[i] >= '0' &&
express[i] <= '9'))
        {
            result += express[i];
        }
        else
        {
            while (top > -1 && precedence(stack[top]) >=
precedence(express[i]))
            {
                result += stack[top];
                top--;
            }
```

```
            top++;
            stack[top] = express[i];
        }
    }
    while (top > -1)
    {
        result += stack[top];
        top--;
    }
    cout << "Postflix Result: " << result << endl;
    return 0;
}
```

Output: (screenshot)

Test Case: Any two (screenshot)



Conclusion:

The conclusion is , using stack ADT, we can convert infix expression to postfix expression by operations like Push and Pop.

Name of Student: ____Sparsh Sharma____

Roll Number: _____37_____

Experiment No: 4

Title: Evaluate Postfix Expression using Stack ADT.

Theory:

A stack is an Abstract Data Type (ADT) that can be implemented using a Linked List or an Array. It follows the Last In, First Out (LIFO) principle, meaning the last element inserted is the first one to be deleted. There are three main operations in a stack:

- Push: Insert an element at the top of the stack.
- Pop: Delete the topmost element from the stack.
- Peek: Return the topmost element from the stack without removing it.

One common use of a stack is to evaluate a postfix expression. Operands are pushed onto the stack, and when an operator is encountered, operands are popped, the operation is performed, and the result is pushed back onto the stack. After evaluating the entire expression, the topmost element of the stack is the final result.

Code:

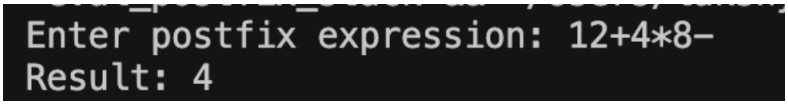
```cpp
//  Evaluate Postfix Expression using Stack ADT.
// evaluating postfix expression using stack(array)
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string expression;
    char stack[100];
    int stack1[100];
    int top = -1, a, b, result = 0;
    cout << "Enter postfix expression: ";
    getline(cin, expression);
    for (int i = 0; i < expression.length(); i++)
    {
```

```
        stack[i] = expression[i];
    }
    stack[expression.length()] = ')';
    int i = 0;
    while (stack[i] != ')')
    {
        if (stack[i] == '*' || stack[i] == '/' || stack[i] == '%'
|| stack[i] == '-' || stack[i] == '+')
        {
            a = stack1[top];
            top--;
            b = stack1[top];
            top--;
            if (stack[i] == '*')
            {
                result = b * a;
            }
            else if (stack[i] == '/')
            {
                if (a != 0)
                {
                    result = b / a;
                }
                else
                {
                    cout << "Error: Division by zero." << endl;
                    return 1;
                }
            }
            else if (stack[i] == '%')
            {
                result = b % a;
            }
            else if (stack[i] == '+')
            {
                result = b + a;
            }
            else
            {
                result = b - a;
            }
            top++;
            stack1[top] = result;
        }
        else
        {
            top++;
            stack1[top] = int(stack[i]) - 48;
        }
```

```
        i++;
    }
    cout << "Result: " << stack1[top] << endl;
    return 0;
}
```

Output: (screenshot)

Test Case: Any two (screenshot)



Conclusion:

The conclusion is , using stack ADT, we can evaluate a postfix expression by operations like Push and Pop.

Name of Student: ____Sparsh Sharma____

Roll Number: _____37_____

Experiment No: 5

Title: Implement Linear Queue ADT using array.

Theory:

An array is a collection of elements of similar data types with a fixed size, accessed by index starting from 0 to n-1 (where n is the size of the array).

A queue is an Abstract Data Type (ADT) that can be implemented using a Linked List or an Array. It follows the First In, First Out (FIFO) principle, meaning the first element inserted is the first one to be deleted.

In a queue, two variables, Front and Rear, point to the first and last elements, respectively. The main operations in a queue are:

- Enqueue: Insert an element at the rear of the queue.
- Dequeue: Delete the frontmost element from the queue.
- Peek: Return the frontmost element from the queue without removing it.

Code:

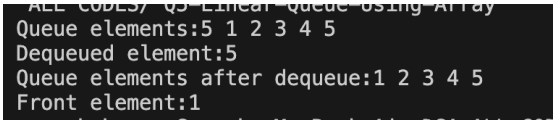
// Implement Linear Queue ADT using array.

```
#include <iostream>
using namespace std;
#define max_size 100
class Queue {
    int front,rear,size;
    int array[max_size];
public:
    Queue(){
        front=rear=-1;
        size=0;
    }
```

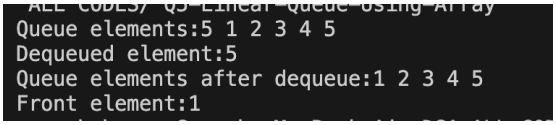
```
    bool empty(){
        return size==0;
    }

    bool full(){
        return size==max_size;
    }

    void enqueue(int item) {
        if (full()){
            cout<<"Queue is full\n";
            return;
        }
        rear=(rear+1)%max_size;
        array[rear]=item;
        size++;
    }

    int dequeue(){
        if (empty()){
            cout<<"Queue is empty\n";
            return -1;
        }
        int item=array[front];
        front=(front+1)%max_size;
        size--;
        return item;
    }

    int peek(){
        if (empty()){
            cout<<"Queue is empty\n";
            return -1;
        }
        return array[front];
    }

    void display(){
        if (empty()){
            cout<<"Queue is empty\n";
            return;
        }
        int i=front;
        while (i!=rear) {
            cout<<array[i]<<" ";
            i=(i + 1)%max_size;
        }
        cout<<array[rear]<<endl;
    }
```

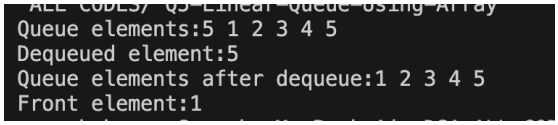
```
};

int main(){
    Queue q;
    q.enqueue(1);
    q.enqueue(2);
    q.enqueue(3);
    q.enqueue(4);
    q.enqueue(5);

    cout<<"Queue elements:";
    q.display();

    cout<<"Dequeued element:" <<q.dequeue()<< endl;

    cout<<"Queue elements after dequeue:";
    q.display();

    cout<<"Front element:"<<q.peek()<<endl;

    return 0;
}
```

Output: (screenshot)



Test Case: Any two (screenshot)





Conclusion:

The conclusion is we have demonstrated how to implement a linear queue ADT using an array in C++. The queue supports basic operations such as enqueue, dequeue, isEmpty, isFull, peek, and display .

Name of Student: ____Sparsh Sharma____

Roll Number: _____37_____

Experiment No: 6

Title: Implement Circular Queue ADT using array.

Theory: A Circular Queue is like a regular queue, but when the rear reaches the end of the array, it wraps around to the beginning.

The implementation uses an array and tracks front and rear indices. Methods check if the queue is full or empty, add elements to the rear (wrapping around if needed), remove elements from the front (also wrapping around), and display the queue contents.

Code:

```
// Implement Circular Queue ADT using array.
#include <iostream>
using namespace std;
#define size 10

class CircularQueue {
private:
    int items[size],front,rear;

public:
    CircularQueue(){
        front=-1;
        rear=-1;
    }
    bool full(){
        if ((front==0 && rear==size-1) || (rear == (front-1) %
(size-1))){
            return true;
        }
        return false;
    }
    bool empty() {
        if(front==-1)
            return true;
        else
            return false;
```

```
    }
    void enQueue(int element) {
        if (full()){
            cout<<"Queue is full";
        }
        else{
            if (front == -1) front=0;
            rear= (rear+1) % size;
            items[rear]=element;
            cout<<endl
                 <<"Inserted"<<element<< endl;
        }
    }
    int deQueue() {
        int element;
        if(empty()){
            cout<<"Queue is empty"<<endl;
            return (-1);
        }
        else{
            element=items[front];
            if(front==rear) {
                front=-1;
                rear=-1;
            } else {
                front=(front+1) % size;
            }
            return (element);
        }
    }

    void displayQueue(){
        if (empty()){
            cout<<"Empty Queue"<<endl;
        } else{
            cout<<"Front -> "<<front<< endl;
            cout<<"Items -> ";
            for(int i=front;i!= rear;i=(i + 1) % size) {
                cout<<items[i]<< " ";
            }
            cout<<items[rear]<<endl;
            cout<<"Rear -> "<<rear<<endl;
        }
    }
};

int main(){
    CircularQueue q;
    q.deQueue();
```

```
    q.enQueue(1);
    q.enQueue(2);
    q.enQueue(3);
    q.enQueue(4);
    q.enQueue(5);
    q.enQueue(6);
    q.displayQueue();
    int elements = q.deQueue();

    if (elements != -1)
        cout << "Deleted Element is " << elements << endl;
    q.displayQueue();
    q.enQueue(7);
    q.displayQueue();

    q.enQueue(8);

    return 0;
}
```

Output: (screenshot)

```
Queue is empty

Inserted1

Inserted2

Inserted3

Inserted4

Inserted5

Inserted6
```

Test Case: Any two (screenshot)

```
Inserted6
Front -> 0
Items -> 1 2 3 4 5 6
Rear -> 5
Deleted Element is 1
Front -> 1
Items -> 2 3 4 5 6
Rear -> 5
```

```
Inserted7
Front -> 1
Items -> 2 3 4 5 6 7
Rear -> 6
```

Conclusion: The conclusion is a Circular Queue ADT using an array. It includes methods to check if the queue is full or empty, to enqueue and dequeue elements, and to display the contents of the queue.

The implementation ensures that the queue operates in a circular manner are working properly or not.

Name of Student: ____Sparsh Sharma____

Roll Number: _____37_____

Experiment No: 7

Title: Implement Singly Linked List ADT

Theory: A linked list is a data structure consisting of nodes, each containing data and a pointer to the next node. It uses dynamic memory allocation and a start pointer to the first node. The last node's pointer is NULL, indicating the end of the list.

Code:

// menu driven linked list

```
#include <iostream>
using namespace std;
class Node
{
public:
    int data;
    Node *next;
    Node()
    {
        cout << "Enter data: ";
        cin >> data;
        next = NULL;
    }
};
Node *createList(int n)
{
    Node *start = NULL;
    Node *ptr = NULL;
    for (int i = 0; i < n; i++)
    {
        Node *new_node = new Node();
        if (start == NULL)
        {
            start = new_node;
            ptr = start;
        }
        else
        {
            ptr->next = new_node;
            ptr = new_node;
```

```
        }
    }
    return start;
}
void insertAtStart(Node *&a)
{
    Node *new_node = new Node();
    if (new_node == NULL)
    {
        cout << "Overflow";
        return;
    }
    else
    {
        new_node->next = a;
        a = new_node;
    }
}
void insertAtEnd(Node *&a)
{
    Node *new_node = new Node();
    Node *ptr = a;
    while (ptr->next != NULL)
    {
        ptr = ptr->next;
    }
    ptr->next = new_node;
    new_node->next = NULL;
}
void insertAfterElement(Node *&a)
{
    int n;
    cout << "Enter element after which to add a node: ";
    cin >> n;
    Node *new_node = new Node();
    Node *ptr = a;
    if (new_node == NULL)
    {
        cout << "Overflow" << endl;
        return;
    }
    else
    {
        while (ptr->data != n)
        {
            ptr = ptr->next;
            if (ptr == NULL)
            {
                cout << "No element found" << endl;
```

```
                return;
            }
        }
        new_node->next = ptr->next;
        ptr->next = new_node;
    }
}
void insertBeforeElement(Node *&a)
{
    int b;
    cout << "Enter element to add a node before it: ";
    cin >> b;
    Node *new_node = new Node();
    Node *ptr = a;
    if (new_node == NULL)
    {
        cout << "Overflow" << endl;
        return;
    }
    else
    {
        if (ptr->data == b)
        {
            new_node->next = ptr;
            a = new_node;
            return;
        }
        Node *preptr = ptr;
        ptr = ptr->next;
        while (ptr != NULL)
        {
            if (ptr->data == b)
            {
                new_node->next = ptr;
                preptr->next = new_node;
                return;
            }
            preptr = ptr;
            ptr = ptr->next;
        }
        cout << "Element not found" << endl;
    }
}
void deleteFirstNode(Node *&a)
{
    Node *ptr = a;
    if (a == NULL)
    {
        cout << "Underflow" << endl;
```

```
        return;
    }
    else
    {
        a = ptr->next;
        delete ptr;
    }
}
void deleteLastNode(Node *&a)
{
    Node *ptr = a;
    Node *preptr = ptr;
    if (a == NULL)
    {
        cout << "Underflow" << endl;
        return;
    }
    else
    {
        while (ptr->next != NULL)
        {
            preptr = ptr;
            ptr = ptr->next;
        }
        if (preptr == ptr)
        {
            delete ptr;
            a = NULL;
        }
        else
        {
            preptr->next = NULL;
            delete ptr;
        }
    }
}
void deleteBeforeElement(Node *&a)
{
    int b;
    cout << "Enter element to delete a node before it: ";
    cin >> b;
    Node *ptr = a;
    Node *preptr = NULL;
    Node *temp = a;
    if (a == NULL)
    {
        cout << "Underflow" << endl;
        return;
    }
```

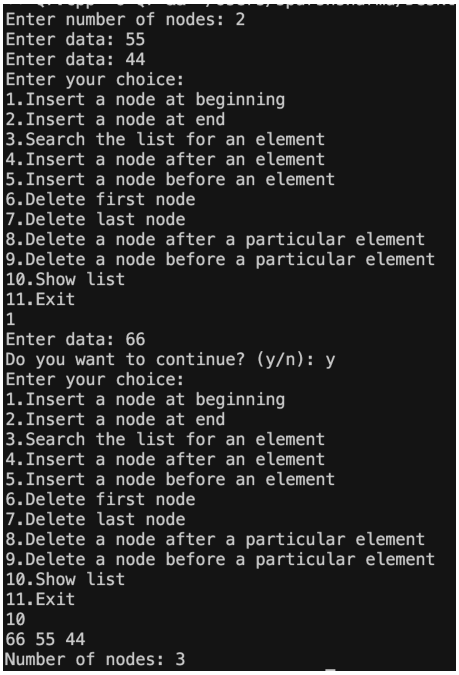
```
    else
    {
        if (a->data == b)
        {
            cout << "No element found" << endl;
            return;
        }
        while (ptr->data != b)
        {
            temp = preptr;
            preptr = ptr;
            ptr = ptr->next;
            if (ptr == NULL)
            {
                cout << "Element not found" << endl;
                return;
            }
        }
        if (preptr == NULL)
        {
            cout << "Element not found" << endl;
        }
        else
        {
            if (preptr == a)
            {
                a = ptr;
            }
            else
            {
                temp->next = ptr;
            }
            delete preptr;
        }
    }
}
void deleteAfterElement(Node *&a)
{
    int b;
    cout << "Enter element to delete node after: ";
    cin >> b;
    Node *ptr = a;
    Node *preptr = a;
    if (a == NULL)
    {
        cout << "Underflow" << endl;
        return;
    }
    else
```

```
    {
        while (preptr->data != b)
        {
            preptr = ptr;
            ptr = ptr->next;
            if (ptr == NULL)
            {
                cout << "Element not found" << endl;
                return;
            }
        }
        if (ptr == NULL)
        {
            cout << "Element not found" << endl;
        }
        else
        {
            if (ptr->next == NULL)
            {
                cout << "No element to delete" << endl;
            }
            else
            {
                preptr = ptr;
                ptr = ptr->next;
                preptr->next = ptr->next;
                delete ptr;
            }
        }
    }
}
void searchElement(Node *a, int b)
{
    Node *ptr = a;
    Node *pos = NULL;
    while (ptr != NULL)
    {
        if (ptr->data == b)
        {
            pos = ptr;
            break;
        }
        else
        {
            ptr = ptr->next;
        }
    }
    if (pos == NULL)
    {
```

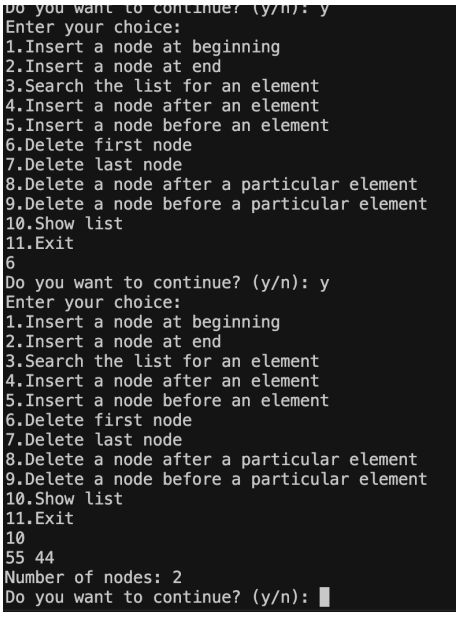
```
        cout << "Element not found" << endl;
    }
    else
    {
        cout << "Element " << pos->data << " found at " << pos <<
endl;
    }
}
void showList(Node *a)
{
    int count = 0;
    Node *ptr = a;
    while (ptr != NULL)
    {
        cout << ptr->data << " ";
        ptr = ptr->next;
        count++;
    }
    cout << endl
         << "Number of nodes: " << count << endl;
}
void deleteList(Node *&a)
{
    Node *ptr = a;
    Node *temp = NULL;
    while (ptr != NULL)
    {
        temp = ptr;
        ptr = ptr->next;
        delete temp;
    }
    a = NULL;
}
int main()
{
    int n;
    cout << "Enter number of nodes: ";
    cin >> n;
    Node *start = createList(n);
    int choice;
    char ans = 'y';
    do
    {
        cout << "Enter your choice: \n1.Insert a node at
beginning\n2.Insert a node at end\n3.Search the list for an
element\n4.Insert a node after an element\n5.Insert a node before
an element\n6.Delete first node\n7.Delete last node\n8.Delete a
node after a particular element\n9.Delete a node before a
particular element\n10.Show list\n11.Exit\n";
```

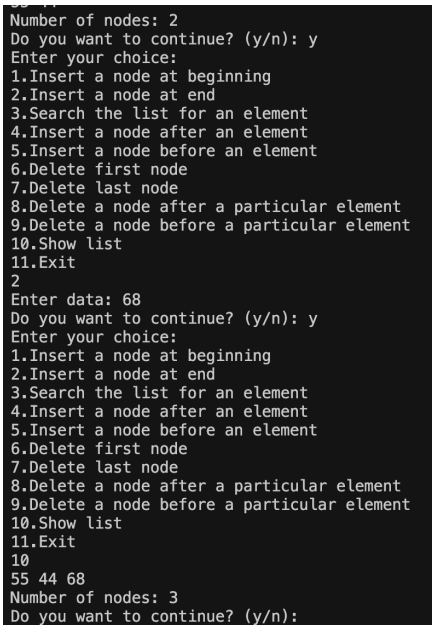
```
        cin >> choice;
        switch (choice)
        {
        case 1:
            insertAtStart(start);
            break;
        case 2:
            insertAtEnd(start);
            break;
        case 3:
            int element;
            cout << "Enter the element to search for: ";
            cin >> element;
            searchElement(start, element);
            break;
        case 4:
            insertAfterElement(start);
            break;
        case 5:
            insertBeforeElement(start);
            break;
        case 6:
            deleteFirstNode(start);
            break;
        case 7:
            deleteLastNode(start);
            break;
        case 8:
            deleteAfterElement(start);
            break;
        case 9:
            deleteBeforeElement(start);
            break;
        case 10:
            showList(start);
            break;
        case 11:
            cout << "Exiting...\n";
            return 0;
        default:
            cout << "Wrong choice" << endl;
        }
        cout << "Do you want to continue? (y/n): ";
        cin >> ans;
    } while (ans == 'y');
    deleteList(start);
    return 0;
}
```

Output: (screenshot)



Test Case: Any two (screenshot)





Conclusion: The conclusion is , we can implement a linked list by using class or structure and allocate heap memory for the node by using new operator or malloc function. We can deallocate memory for the node by using free function or delete operator.

Name of Student: ____Sparsh Sharma____

Roll Number: _____37_____

Experiment No: 8

Title: Implement Circular Linked List ADT.

Theory:A linked list is a data structure where each node contains data and a pointer to the next node. It uses dynamic memory allocation and a start pointer to the first node. In a circular linked list, the last node's pointer points back to the first node.

Code:

// circular linked list menu

```
#include <iostream>
using namespace std;
class Node
{
public:
 int data;
 Node *next;
 Node()
 {
 cout << "Enter data: ";
 cin >> data;
 next = NULL;
 }
};
Node *createList(int n)
{
 Node *start = NULL;
 Node *ptr = start;
 for (int i = 0; i < n; i++)
 {
 Node *newNode = new Node();
 if (start == NULL)
 {
 start = newNode;
 ptr = newNode;
 }
 else
 {
 ptr->next = newNode;
```

```
 ptr = ptr->next;
 }
 }
 ptr->next = start;
 return start;
}
void searchElement(Node *&a)
{
 int element;
 cout << "Enter element to search: ";
 cin >> element;
 Node *ptr = a;
 Node *preptr = ptr;
 Node *temp = NULL;
 while (preptr->next != a)
 {
 if (ptr->data == element)
 {
 cout << "Element " << ptr->data << " found in node " <<
ptr << endl;
 temp = ptr;
 break;
 }
 preptr = ptr;
 ptr = ptr->next;
 }
 if (temp == NULL)
 {
 cout << "No element found" << endl;
 }
}
void traverseList(Node *&a)
{
 int count = 0;
 Node *ptr = a;
 Node *preptr = ptr;
 cout << "Circular Linked List: " << endl;
 while (preptr->next != a)
 {
 cout << ptr->data << endl;
 preptr = ptr;
 ptr = ptr->next;
 count++;
 }
 cout << "Number of nodes: " << count << endl;
}
void insertAtBegin(Node *&a)
{
 Node *ptr = a;
```

```
 Node *newNode = new Node();
 if (newNode == NULL)
 {
 cout << "Overflow" << endl;
 return;
 }
 newNode->next = a;
 while (ptr->next != a)
 {
 ptr = ptr->next;
 }
 ptr->next = newNode;
 a = newNode;
}
void insertAtEnd(Node *&a)
{
 Node *ptr = a;
 Node *newNode = new Node();
 if (newNode == NULL)
 {
 cout << "Overflow" << endl;
 return;
 }
 newNode->next = a;
 while (ptr->next != a)
 {
 ptr = ptr->next;
 }
 ptr->next = newNode;
}
void insertBeforeElement(Node *&a)
{
 Node *ptr = a;
 Node *preptr = a;
 Node *newNode = new Node();
 if (newNode == NULL)
 {
 cout << "Overflow" << endl;
 return;
 }
 else
 {
 int element;
 cout << "Enter element to insert a node before it: ";
 cin >> element;
 if (ptr->data == element)
 {
 newNode->next = a;
 while (ptr->next != a)
```

```
 {
 ptr = ptr->next;
 }
 ptr->next = newNode;
 a = newNode;
 return;
 }
 else
 {
 do
 {
 if (ptr->data == element)
 {
 preptr->next = newNode;
 newNode->next = ptr;
 return;
 }
 preptr = ptr;
 ptr = ptr->next;
 } while (ptr != a);
 if (ptr == a)
 {
 cout << "Element not found" << endl;
 return;
 }
 }
 }
}
void insertAfterElement(Node *&a)
{
 Node *ptr = a;
 Node *preptr = ptr;
 Node *newNode = new Node();
 if (newNode == NULL)
 {
 cout << "Overflow" << endl;
 return;
 }
 else
 {
 int element;
 cout << "Enter element to insert a node after it: ";
 cin >> element;
 do
 {
 if (preptr->data == element)
 {
 if (preptr == a)
 {
```

```
 newNode->next = a->next;
 a->next = newNode;
 return;
 }
 if (preptr->next == a)
 {
 preptr->next = newNode;
 newNode->next = a;
 a = newNode;
 return;
 }
 preptr->next = newNode;
 newNode->next = ptr;
 return;
 }
 preptr = ptr;
 ptr = ptr->next;
 } while (ptr != a);
 if (ptr == a)
 {
 cout << "Element not found" << endl;
 return;
 }
 }
}
void deleteAtBegin(Node *&a)
{
 Node *ptr = a;
 if (a == NULL)
 {
 cout << "Underflow" << endl;
 return;
 }
 while (ptr->next != a)
 {
 ptr = ptr->next;
 }
 Node *temp = a;
 ptr->next = temp->next;
 a = temp->next;
 delete temp;
}
void deleteAtEnd(Node *&a)
{
 Node *ptr = a;
 Node *preptr = ptr;
 if (a == NULL)
 {
 cout << "Underflow" << endl;
```

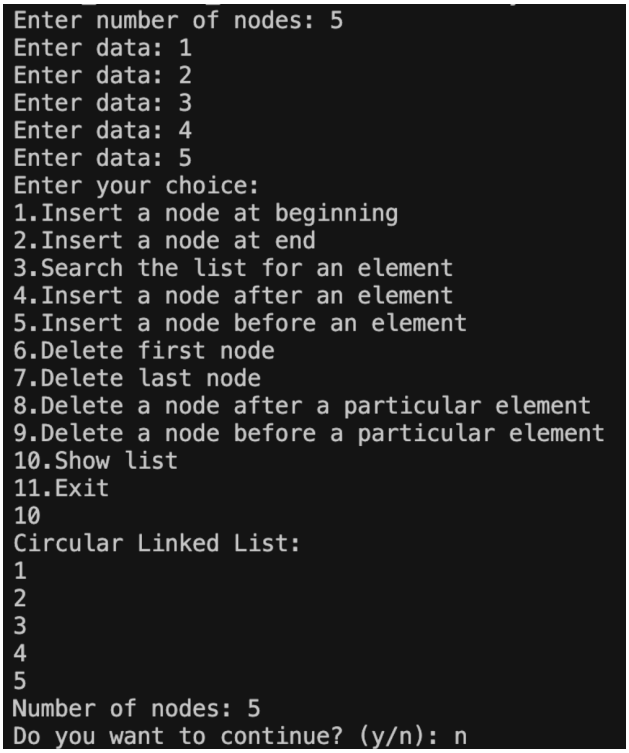
```
 return;
 }
 while (ptr->next != a)
 {
 preptr = ptr;
 ptr = ptr->next;
 }
 preptr->next = a;
 delete ptr;
}
void deleteBeforeElement(Node *&a)
{
 int element;
 cout << "Enter element to delete node before it: ";
 cin >> element;
 Node *ptr = a;
 Node *preptr = NULL;
 Node *temp = NULL;
 if (a == NULL)
 {
 cout << "Underflow" << endl;
 return;
 }
 else
 {
 if (element == a->data)
 {
 cout << "Cannot delete before first element" << endl;
 return;
 }
 else
 {
 do
 {
 if (ptr->data == element)
 {
 if (temp == NULL)
 {
 ptr = a;
 while (ptr->next != a)
 {
 ptr = ptr->next;
 }
 temp = a;
 ptr->next = temp->next;
 a = temp->next;
 delete temp;
 return;
 }
```

```
 temp->next = ptr;
 delete preptr;
 }
 temp = preptr;
 preptr = ptr;
 ptr = ptr->next;
 } while (ptr != a);
 return;
 if (ptr == a)
 {
 cout << "Element not found" << endl;
 return;
 }
 }
 }
}
void deleteAfterElement(Node *&a)
{
 int element;
 cout << "Enter element to delete node after it: ";
 cin >> element;
 if (a == NULL)
 {
 cout << "Underflow" << endl;
 return;
 }
 Node *ptr = a;
 Node *preptr = NULL;
 do
 {
 if (ptr->data == element)
 {
 if (ptr->next == a)
 {
 Node *temp = ptr->next;
 ptr->next = temp->next;
 delete temp;
 a = ptr->next;
 return;
 }
 else
 {
 preptr = ptr;
 ptr = ptr->next;
 preptr->next = ptr->next;
 delete ptr;
 return;
 }
 }
```

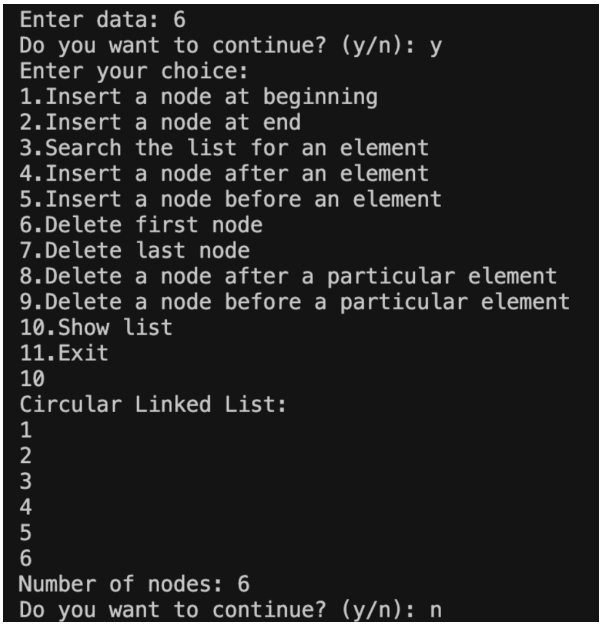
```
 preptr = ptr;
 ptr = ptr->next;
 } while (ptr != a);
 return;
 cout << "Element not found" << endl;
}
void deleteList(Node *&a)
{
 Node *ptr = a;
 Node *preptr = ptr;
 while (ptr->next != a)
 {
 preptr = ptr;
 ptr = ptr->next;
 delete preptr;
 }
 a = NULL;
}
int main()
{
 int n;
 cout << "Enter number of nodes: ";
 cin >> n;
 Node *start = createList(n);
 int choice;
 char ans = 'y';
 do
 {
 cout << "Enter your choice: \n1.Insert a node at
beginning\n2.Insert a node at end\n3.Search the list for an
element\n4.Insert a node after an element\n5.Insert a node before
an element\n6.Delete first node\n7.Delete last node\n8.Delete a
node after a particular element\n9.Delete a node before a
particular element\n10.Show list\n11.Exit\n";
 cin >> choice;
 switch (choice)
 {
 case 1:
 insertAtBegin(start);
 break;
 case 2:
 insertAtEnd(start);
 break;
 case 3:
 searchElement(start);
 break;
 case 4:
 insertAfterElement(start);
 break;
```

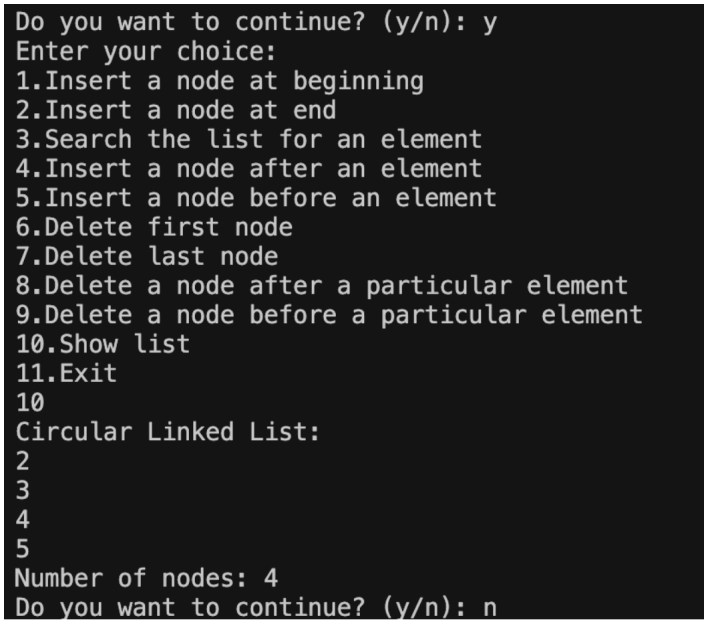
```
 case 5:
 insertBeforeElement(start);
 break;
 case 6:
 deleteAtBegin(start);
 break;
 case 7:
 deleteAtEnd(start);
 break;
 case 8:
 deleteAfterElement(start);
 break;
 case 9:
 deleteBeforeElement(start);
 break;
 case 10:
 traverseList(start);
 break;
 case 11:
 cout << "Exiting...\n";
 return 0;
 default:
 cout << "Wrong choice" << endl;
 }
 cout << "Do you want to continue? (y/n): ";
 cin >> ans;
 } while (ans == 'y');
 deleteList(start);
 return 0;
}
```

Output: (screenshot)



Test Case: Any two (screenshot)





Conclusion:

The Conclusion is we can implement a circular linked list by using class or structure and allocate heap memory for the node by using new operator or malloc function. We can deallocate memory for the node by using free function or delete operator.

Name of Student: ____Sparsh Sharma____

Roll Number: _____37_____

Experiment No: 9

Title: Implement Stack ADT using Linked List.

Theory:

A stack is like a stack of plates: you can only add or remove the top plate. It follows the Last In, First Out (LIFO) principle. We can implement a stack using a linked list by inserting and deleting elements from the beginning.

A linked list is a data structure where each element (node) contains data and a pointer to the next node. It uses dynamic memory and has a start pointer in stack memory that points to the first node in heap memory. The last node's pointer points back to the first node in a circular linked list.

Code:

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

class Node {
public:
    int element;
    Node *next;

    Node() {
        cout << "Enter element: ";
        cin >> element;
        next = nullptr;
    }
};

void pushList(Node *&a) {
    Node *newnode = new Node();
    if (a == nullptr) {
        a = newnode;
    } else {
        newnode->next = a;
        a = newnode;
    }
    cout << "Element pushed successfully\n";
}
```

```
void popList(Node *&a) {
    if (a == nullptr) {
        cout << "Stack is empty\n";
    } else {
        Node *ptr = a;
        cout << "Element " << a->element << " popped
successfully\n";
        a = a->next;
        delete ptr;
    }
}

void peekList(Node *&a) {
    if (a == nullptr) {
        cout << "Stack is empty\n";
    } else {
        cout << "Top element: " << a->element << endl;
    }
}

void seeList(Node *&a) {
    Node *ptr = a;
    if (a == nullptr) {
        cout << "Empty stack\n";
    } else {
        while (ptr) {
            cout << ptr->element << endl;
            ptr = ptr->next;
        }
    }
}

int main() {
    Node *top = nullptr;
    int choice;
    while (true) {
        cout << "\nStack operation:
\n1.Push\n2.Pop\n3.Peek\n4.Exit\n";
        cin >> choice;
        switch (choice) {
            case 1:
                pushList(top);
                break;
            case 2:
                popList(top);
                break;
            case 3:
                peekList(top);
```

```
                break;
            case 4:
                cout << "Exiting...\n";
                return 0;
            default:
                cout << "Wrong choice\n";
                break;
        }
    }
}
```

Output: (screenshot)

```
Stack operation:
1.Push
2.Pop
3.Peek
4.Exit
1
Enter element: 22
Element pushed successfully
```

Test Case: Any two (screenshot)

```
Stack operation:
1.Push
2.Pop
3.Peek
4.Exit
2
Element 22 popped successfully
```

```
Element pushed successfully

Stack operation:
1.Push
2.Pop
3.Peek
4.Exit
3
Top element: 55
```

Conclusion: The conclusion is , we can implement Stack by linked list by using class or structure and allocate heap memory for the node by using new operator or malloc function. We can deallocate memory for the node by using free function or delete operator. We can implement push and pop operations through insertion at beginning and deletion from beginning algorithms.

Name of Student: ____Sparsh Sharma____

Roll Number: _____37_____

Experiment No: 10

Title: Implement Linear Queue ADT using Linked List.

Theory:A queue is an Abstract Data Type (ADT) that follows the FIFO (First In, First Out) principle. It can be implemented using a linked list or an array, with operations like Enqueue (insertion from the rear), Dequeue (deletion from the front), and Peek (returning the frontmost element). Linked list is a data structure consisting of nodes with data and a next pointer, storing the address of the next node. It uses dynamic memory allocation, with a start pointer pointing to the first node and the last node's next pointer pointing back to the first node, creating a circular linked list.

Code:

```
#include <iostream>

using namespace std;

class Node {
public:
    int data;
    Node *next;

    Node() {
        cout << "Enter data: ";
        cin >> data;
        next = nullptr;
    }
};

void enqueue(Node *&start, Node *&end) {
    Node *newnode = new Node();
    if (start == nullptr) {
        start = newnode;
        end = newnode;
    } else {
        end->next = newnode;
        end = newnode;
    }
    cout << "Element added successfully.\n";
```

```
}

void dequeue(Node *&start, Node *&end) {
    Node *ptr = nullptr;
    if (start == nullptr) {
        cout << "Queue is empty.\n";
        return;
    } else {
        ptr = start;
        start = start->next;
        cout << "Element " << ptr->data << " deleted successfully.
\n";
        delete ptr;
    }
}

void peek(Node *&start) {
    if (start == nullptr) {
        cout << "Queue is empty.\n";
        return;
    } else {
        cout << "Front element: " << start->data << endl;
    }
}

void deleteQueue(Node *&start) {
    Node *ptr = start;
    Node *temp = nullptr;
    while (ptr != nullptr) {
        temp = ptr;
        ptr = ptr->next;
        delete temp;
    }
    start = nullptr;
}

void showQueue(Node *start) {
    Node *ptr = start;
    while (ptr != nullptr) {
        cout << ptr->data << endl;
        ptr = ptr->next;
    }
}

int main() {
    Node *front = nullptr;
    Node *rear = nullptr;
    int choice;
    while (true) {
```

```
        cout << "\nQueue Operations:
\n1.Enqueue\n2.Dequeue\n3.Peek\n4.Exit\n";
        cin >> choice;
        switch (choice) {
            case 1:
                enqueue(front, rear);
                break;
            case 2:
                dequeue(front, rear);
                break;
            case 3:
                peek(front);
                break;
            case 4:
                cout << "Exiting...\n";
                deleteQueue(front);
                return 0;
            default:
                cout << "Wrong choice.\n";
                break;
        }
    }
    return 0;
}
```

Output: (screenshot)

```
Queue Operations:
1.Enqueue
2.Dequeue
3.Peek
4.Exit
1
Enter data: 55
Element added successfully.
```

Test Case: Any two (screenshot)

```
Queue Operations:
1.Enqueue
2.Dequeue
3.Peek
4.Exit
2
Element 55 deleted successfully.
```

```
Element 55 deleted successfully.

Queue Operations:
1.Enqueue
2.Dequeue
3.Peek
4.Exit
3
Front element: 59
```

Conclusion:

The conclusion is , we can implement Linear Queue by linked list by using class or structure and allocate heap memory for the node by using new operator or malloc function. We can deallocate memory for the node by using free function or delete operator. We can implement enqueue and dequeue operations through insertion at end and deletion from beginning algorithms.

Name of Student: ____Sparsh Sharma____

Roll Number: _____37_____

Experiment No: 11

Title:Implement Binary Search Tree ADT using Linked List.

Theory:A binary tree is a data structure with a root node and each parent node having at most two child nodes. In a binary search tree, nodes with values less than the root are in the left subtree, and nodes with values greater than or equal to the root are in the right subtree.

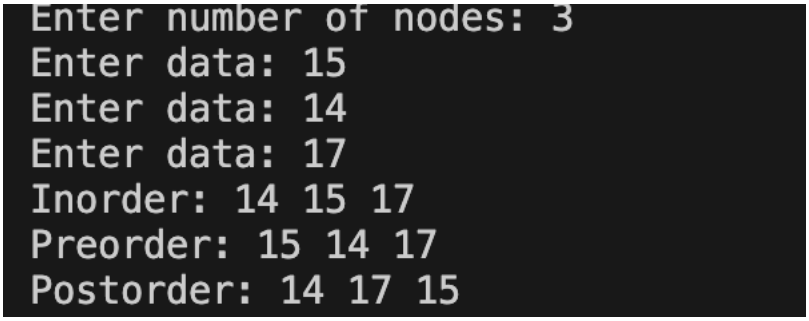
Code:

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

class Node
{
public:
    int data;
    Node *left;
    Node *right;
    Node()
    {
        cout << "Enter data: ";
        cin >> data;
        left = right = NULL;
    }
};

class BST
{
public:
    Node *root;
    BST()
    {
        root = NULL;
    }
    ~BST()
    {
        deleteTree(root);
    }
    void insert(Node *node)
    {
        if (root == NULL)
```

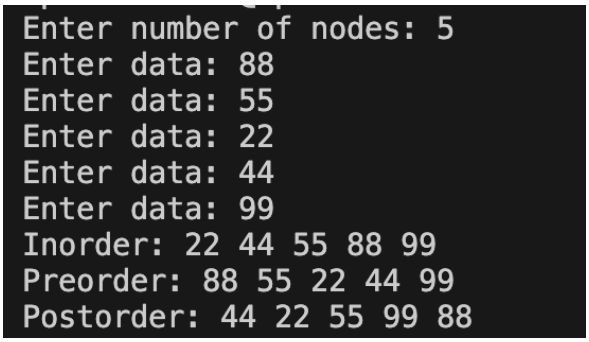
```
        {
            root = node;
            return;
        }
        Node *temp = root;
        while (temp != NULL)
        {
            if (node->data < temp->data)
            {
                if (temp->left == NULL)
                {
                    temp->left = node;
                    return;
                }
                temp = temp->left;
            }
            else
            {
                if (temp->right == NULL)
                {
                    temp->right = node;
                    return;
                }
                temp = temp->right;
            }
        }
    }
    void inorder(Node *node)
    {
        if (node == NULL)
            return;
        inorder(node->left);
        cout << node->data << " ";
        inorder(node->right);
    }
    void preorder(Node *node)
    {
        if (node == NULL)
            return;
        cout << node->data << " ";
        preorder(node->left);
        preorder(node->right);
    }
    void postorder(Node *node)
    {
        if (node == NULL)
            return;
        postorder(node->left);
        postorder(node->right);
```

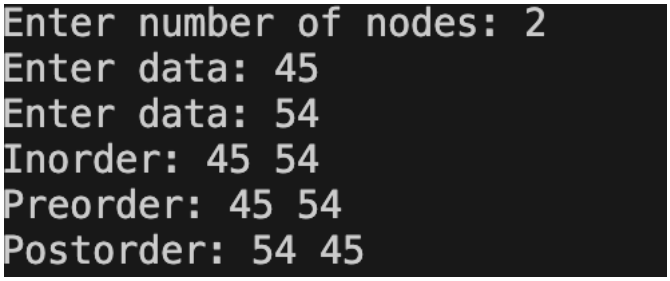
```
        cout << node->data << " ";
    }
    void deleteTree(Node *node)
    {
        if (node == NULL)
            return;
        deleteTree(node->left);
        deleteTree(node->right);
        delete node;
    }
};

int main()
{
    BST bst;
    int n;
    cout << "Enter number of nodes: ";
    cin >> n;
    for (int i = 0; i < n; i++)
    {
        Node *node = new Node();
        bst.insert(node);
    }
    cout << "Inorder: ";
    bst.inorder(bst.root);
    cout << endl;
    cout << "Preorder: ";
    bst.preorder(bst.root);
    cout << endl;
    cout << "Postorder: ";
    bst.postorder(bst.root);
    cout << endl;
    return 0;
}
```

Output: (screenshot)



Test Case: Any two (screenshot)





Conclusion: The conclusion is , we can implement Binary Search Tree ADT using Linked List.

Name of Student: ____Sparsh Sharma____

Roll Number: _____37_____

Experiment No: 12

Title: Implement Graph Traversal techniques: a) Depth First Search b) Breadth First Search

Theory : A Graph is a non-linear data structure which can have parent-child as well as other complex relationships between the nodes. It is a set of edges and vertices, where vertices are the nodes, and the edges are the links connecting the nodes. We can implement a graph using adjacency matrix or adjacency list.

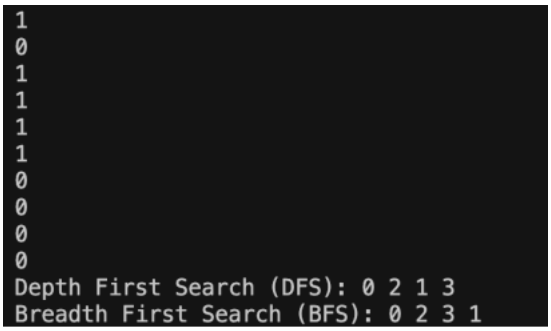
Code:

```
// Graph Traversal techniques: a) Depth First Search b) Breadth
First Search
#include <iostream>
#include <queue>

using namespace std;

const int MAXN = 100;


void dfs(int graph[][MAXN], bool visited[], int n, int node)
{
    visited[node] = true;
    cout << node << " ";

    for (int i = 0; i < n; ++i)
    {
        if (graph[node][i] == 1 && !visited[i])
        {
            dfs(graph, visited, n, i);
        }
    }
}
```

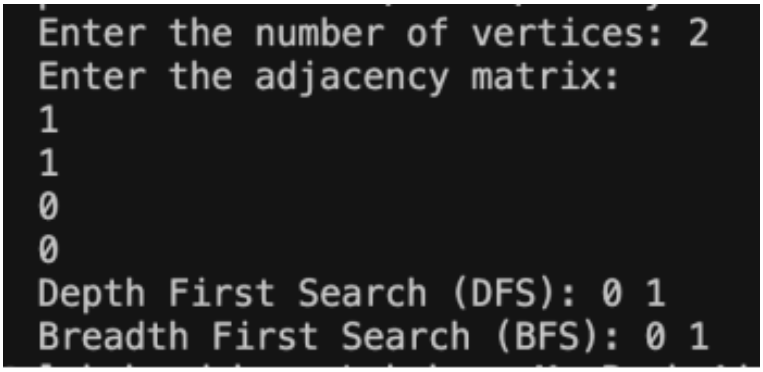
```
void bfs(int graph[][MAXN], bool visited[], int n, int start)
{
    queue<int> q;
    q.push(start);
    visited[start] = true;

    while (!q.empty())
    {
        int node = q.front();
        q.pop();
        cout << node << " ";

        for (int i = 0; i < n; ++i)
        {
            if (graph[node][i] == 1 && !visited[i])
            {
                q.push(i);
                visited[i] = true;
            }
        }
    }
}

int main()
{
    int n;
    cout << "Enter the number of vertices: ";
    cin >> n;

    int graph[MAXN][MAXN]; // Adjacency matrix
    cout << "Enter the adjacency matrix:" << endl;
    for (int i = 0; i < n; ++i)
    {
        for (int j = 0; j < n; ++j)
        {
            cin >> graph[i][j];
        }
    }

    bool visited[MAXN] = {false}; // Visited array to keep track of
visited nodes

    cout << "Depth First Search (DFS): ";
    for (int i = 0; i < n; ++i)
    {
        if (!visited[i])
        {
            dfs(graph, visited, n, i);
```

```
        }
    }
    cout << endl;

    fill(visited, visited + n, false);

    cout << "Breadth First Search (BFS): ";
    for (int i = 0; i < n; ++i)
    {
        if (!visited[i])
        {
            bfs(graph, visited, n, i);
        }
    }
    cout << endl;

    return 0;
}
```

Output: (screenshot)



Test Case: Any two (screenshot)



Conclusion: The conclusion is , we can implement Graph Traversal techniques by Depth First and Breadth First using adjacency matrix.

Name of Student: ____Sparsh Sharma____

Roll Number: _____37_____

Experiment No: 13

Title: Implement Binary Search algorithm to search an element in the array

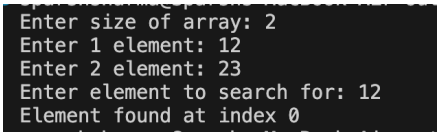
Theory: Binary Search is a searching algorithm which is used in a sorted array by repeatedly dividing the search interval in half. The idea of binary search is to use the information that the array is sorted and reduce the time complexity to O(log N).

Code:

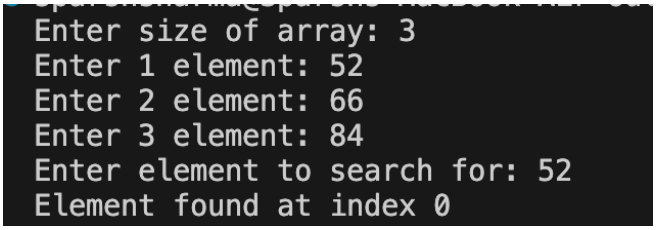
```
// Binary Search algorithm to search an element in an array
#include <iostream>
using namespace std;

int binarySearch(int arr[], int n, int a)
{
    int l = 0, r = n - 1;
    while (l <= r)
    {
        int m = l + (r - l) / 2;
        if (arr[m] == a)
        {
            return m;
        }
        if (arr[m] < a)
        {
            l = m + 1;
        }
        else
        {
            r = m - 1;
        }
    }
    return -1;
}

int main()
```

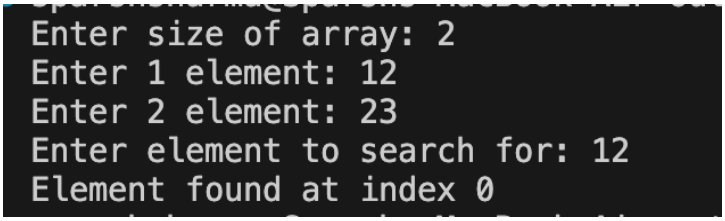
```
{
    int n, a;
    cout << "Enter size of array: ";
    cin >> n;
    int arr[n];
    for (int i = 0; i < n; i++)
    {
        cout << "Enter " << i + 1 << " element: ";
        cin >> arr[i];
    }
    cout << "Enter element to search for: ";
    cin >> a;
    int b = binarySearch(arr, n, a);
    if (b == -1)
    {
        cout << "Element not found." << endl;
    }
    else
    {
        cout << "Element found at index " << b << endl;
    }
    return 0;
}
```

Output: (screenshot)



Test Case: Any two (screenshot)





Conclusion: The conclusion is , we can implement Binary Search algorithm in a sorted array to search the index location of an element present in the array in an efficient manner.

Name of Student: ____Sparsh Sharma____

Roll Number: _____37_____

Experiment No: 14

Title: Implement Bubble Sort algorithm to sort elements of an array in ascending and descending order.

Theory:

In Bubble Sort algorithm, we traverse from left and compare adjacent elements and the higher one is placed at right side. In this way, the largest element is moved to the rightmost end at first. This process is then continued to find the second largest and place it and so on until the data is sorted.

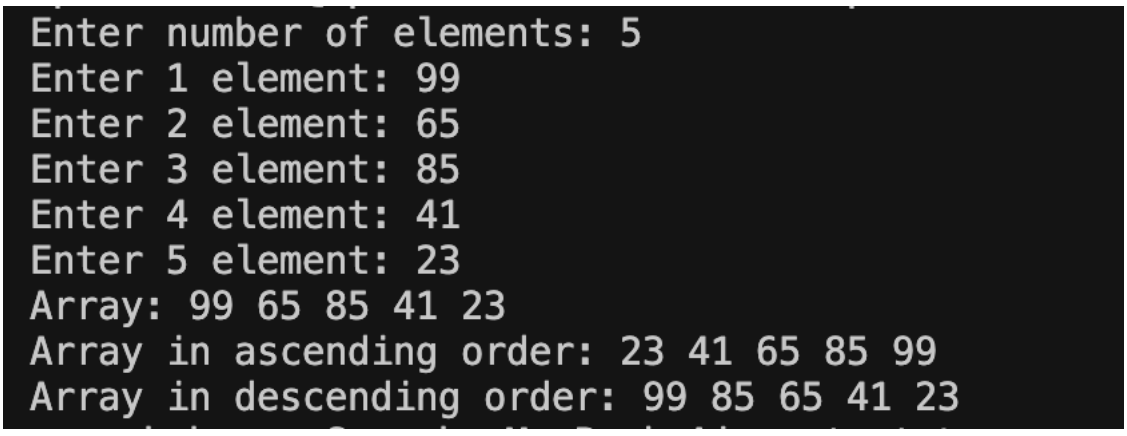
Code:

```
// bubble sort algorithm to sort array in ascending and descending
order
#include <iostream>
using namespace std;

int main()
{
    int n;
    cout << "Enter number of elements: ";
    cin >> n;
    int arr[n];
    for (int i = 0; i < n; i++)
    {
        cout << "Enter " << i + 1 << " element: ";
        cin >> arr[i];
    }
    cout << "Array: ";
    for (int i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }

    // ascending order
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - 1; j++)
        {
            if (arr[j] > arr[j + 1])
```

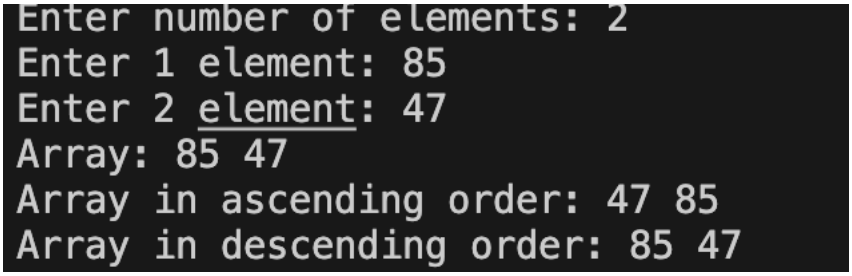
```
            {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
    cout << "\nArray in ascending order: ";
    for (int i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }
```

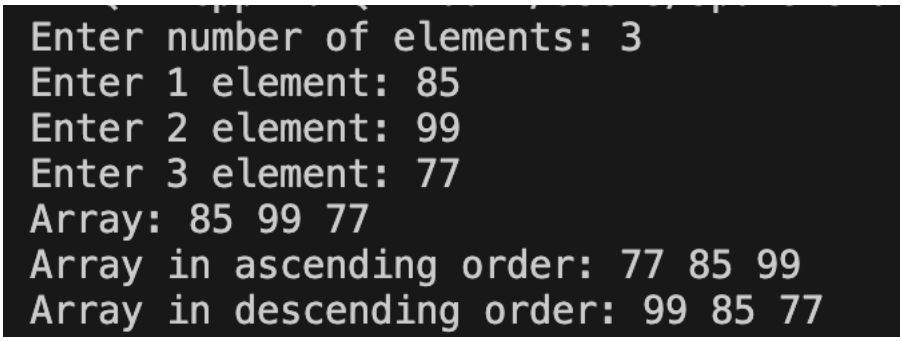
```
    // descending order
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - 1 - i; j++)
        {
            if (arr[j] < arr[j + 1])
            {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
    cout << "\nArray in descending order: ";
    for (int i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }
    cout<<endl;
    return 0;
}
```

Output: (screenshot)



Test Case: Any two (screenshot)





Conclusion:

The conclusion is , we can implement Bubble Sort algorithm to sort the array in ascending or descending order by traversing through the array and comparing the elements to the adjacent elements.