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

DATA STRUCTURES & ALGORITHMS
PROGRAMMING LAB

Prepared by:

Name of Student: JEEVAN NAIDU

Roll No: 150096723015

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. JEEVAN NAIDU

Roll No. 150096723015 Semester II of B.Tech Computer Science & Engineering, ITM Skills University, Kharghar, Navi Mumbai , has completed the term work satisfactorily in subject DATA STRUCTURES AND ALGORITHMS for the academic year 2023 - 2024 as prescribed in the curriculum.

Place: NAVI MUMBAI
Date: 06 APRIL 2024

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	06 APRIL 2024	
2	Implement Stack ADT using array.	06 APRIL 2024	
3	Convert an Infix expression to Postfix expression using stack ADT.	06 APRIL 2024	
4	Evaluate Postfix Expression using Stack ADT.	06 APRIL 2024	
5	Implement Linear Queue ADT using array.	06 APRIL 2024	
6	Implement Circular Queue ADT using array.	06 APRIL 2024	
7	Implement Singly Linked List ADT.	06 APRIL 2024	
8	Implement Circular Linked List ADT.	06 APRIL 2024	
9	Implement Stack ADT using Linked List	06 APRIL 2024	
10	Implement Linear Queue ADT using Linked List	06 APRIL 2024	
11	Implement Binary Search Tree ADT using Linked List.	06 APRIL 2024	
12	Implement Graph Traversal techniques: a) Depth First Search b) Breadth First Search	06 APRIL 2024	
13	Implement Binary Search algorithm to search an element in an array	06 APRIL 2024	
14	Implement Bubble sort algorithm to sort elements of an array in ascending and descending order	06 APRIL 2024	

Name of Student: JEEVAN NAIDU

Roll Number: 150096723015

Experiment No: 01

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

Theory: Arrays represent a fundamental data structure comprising elements of the same data type, with a predetermined size. Elements within arrays are accessed through indices, starting from 0 to n-1, where n denotes the size of the array. To insert an element into an array, all subsequent elements are shifted to the right to create space. Similarly, deletion involves shifting elements to the left to overwrite the deleted element. For searching an element, the array is traversed, and a message is displayed based on whether the element is found or not.

Code: // Implement Array and write a menu driven program to

```
// perform all the operation on array elements
#include<iostream>
using namespace std;
const int MAX=10; // Change the array size as needed

class Array
{
    private:
        int arr[MAX];
        int size;
    public:
        Array() {
            size = 0;
        }
        void insertBeginning(int num);
        void insertEnd(int num);
        void insertAtPosition(int pos, int num);
        void deleteBeginning();
        void deleteEnd();
        void deleteAtPosition(int pos);
        void insertBefore(int element, int num);
        void insertAfter(int element, int num);
        void deleteBefore(int element);
```

```
        void deleteAfter(int element);
        void reverse();
        void search(int num);
        void display();
};

void Array::insertBeginning(int num)
{
    if(size < MAX) {
        for(int i = size; i > 0; i--)
            arr[i] = arr[i-1];
        arr[0] = num;
        size++;
    } else {
        cout << "Array is full. Cannot insert more elements." << endl;
    }
}

void Array::insertEnd(int num)
{
    if(size < MAX) {
        arr[size++] = num;
    } else {
        cout << "Array is full. Cannot insert more elements." << endl;
    }
}

void Array::insertAtPosition(int pos, int num)
{
    if(pos >= 0 && pos <= size && size < MAX) {
        for(int i = size; i > pos; i--)
            arr[i] = arr[i-1];
        arr[pos] = num;
        size++;
    } else {
        cout << "Invalid position or array is full." << endl;
    }
}

void Array::deleteBeginning()
{
    if(size > 0) {
        for(int i = 0; i < size - 1; i++)
            arr[i] = arr[i+1];
        size--;
    } else {
        cout << "Array is empty. Cannot delete." << endl;
    }
}

void Array::deleteEnd()
{
    if(size > 0) {
        size--;
    } else {
        cout << "Array is empty. Cannot delete." << endl;
    }
}
```

```
void Array::deleteAtPosition(int pos)
{
    if(pos >= 0 && pos < size) {
        for(int i = pos; i < size - 1; i++)
            arr[i] = arr[i+1];
        size--;
    } else {
        cout << "Invalid position." << endl;
    }
}

void Array::insertBefore(int element, int num)
{
    int pos = -1;
    for(int i = 0; i < size; i++) {
        if(arr[i] == element) {
            pos = i;
            break;
        }
    }
    if(pos != -1) {
        insertAtPosition(pos, num);
    } else {
        cout << "Element not found." << endl;
    }
}

void Array::insertAfter(int element, int num)
{
    int pos = -1;
    for(int i = 0; i < size; i++) {
        if(arr[i] == element) {
            pos = i + 1;
            break;
        }
    }
    if(pos != -1) {
        insertAtPosition(pos, num);
    } else {
        cout << "Element not found." << endl;
    }
}

void Array::deleteBefore(int element)
{
    int pos = -1;
    for(int i = 0; i < size; i++) {
        if(arr[i] == element) {
            pos = i - 1;
            break;
        }
    }
    if(pos != -1) {
        deleteAtPosition(pos);
    } else {
        cout << "Element not found or no element before it." << endl;
    }
```

```
}

void Array::deleteAfter(int element)
{
    int pos = -1;
    for(int i = 0; i < size; i++) {
        if(arr[i] == element) {
            pos = i + 1;
            break;
        }
    }
    if(pos != -1 && pos < size) {
        deleteAtPosition(pos);
    } else {
        cout << "Element not found or no element after it." << endl;
    }
}

void Array::reverse()
{
    for(int i = 0; i < size / 2; i++) {
        int temp = arr[i];
        arr[i] = arr[size - i - 1];
        arr[size - i - 1] = temp;
    }
}

void Array::search(int num)
{
    for(int i = 0; i < size; i++) {
        if(arr[i] == num) {
            cout << "Element " << num << " found at position " << i + 1 << endl;
            return;
        }
    }
    cout << "Element " << num << " not found." << endl;
}

void Array::display()
{
    cout << "Array: ";
    for(int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main()
{
    Array a;
    int choice, element, num, pos;

    do {
        cout << "\n------ Menu ------" << endl;
        cout << "1. Insert at beginning" << endl;
        cout << "2. Insert at end" << endl;
        cout << "3. Insert at position" << endl;
        cout << "4. Delete at beginning" << endl;
```

```
        cout << "5. Delete at end" << endl;
        cout << "6. Delete at position" << endl;
        cout << "7. Insert before an element" << endl;
        cout << "8. Insert after an element" << endl;
        cout << "9. Delete before an element" << endl;
        cout << "10. Delete after an element" << endl;
        cout << "11. Reverse the array" << endl;
        cout << "12. Search for an element" << endl;
        cout << "13. Display array" << endl;
        cout << "14. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch(choice) {
            case 1:
                cout << "Enter element to insert at beginning: ";
                cin >> num;
                a.insertBeginning(num);
                break;
            case 2:
                cout << "Enter element to insert at end: ";
                cin >> num;
                a.insertEnd(num);
                break;
            case 3:
                cout << "Enter position to insert at: ";
                cin >> pos;
                cout << "Enter element to insert: ";
                cin >> num;
                a.insertAtPosition(pos, num);
                break;
            case 4:
                a.deleteBeginning();
                break;
            case 5:
                a.deleteEnd();
                break;
            case 6:
                cout << "Enter position to delete from: ";
                cin >> pos;
                a.deleteAtPosition(pos);
                break;
            case 7:
                cout << "Enter element to insert before: ";
                cin >> element;
                cout << "Enter element to insert: ";
                cin >> num;
                a.insertBefore(element, num);
                break;
            case 8:
                cout << "Enter element to insert after: ";
                cin >> element;
                cout << "Enter element to insert: ";
                cin >> num;
                a.insertAfter(element, num);
                break;
            case 9:
                cout << "Enter element to delete before: ";
```

```
                cin >> element;
                a.deleteBefore(element);
                break;
            case 10:
                cout << "Enter element to delete after: ";
                cin >> element;
                a.deleteAfter(element);
                break;
            case 11:
                a.reverse();
                cout << "Array reversed." << endl;
                break;
            case 12:
                cout << "Enter element to search: ";
                cin >> num;
                a.search(num);
                break;
            case 13:
                a.display();
                break;
            case 14:
                cout << "Exiting..." << endl;
                break;
            default:
                cout << "Invalid choice. Please try again." << endl;
        }

    } while(choice != 14);

    return 0;
}
```

Output: (screenshot)

------ Menu ------

1. Insert at beginning

2. Insert at end

3. Insert at position

4. Delete at beginning

5. Delete at end

6. Delete at position

7. Insert before an element

8. Insert after an element

9. Delete before an element

10. Delete after an element

11. Reverse the array

12. Search for an element

13. Display array

14. Exit

Enter your choice: 1

Enter element to insert at beginning: 1

------ Menu ------

1. Insert at beginning

2. Insert at end

3. Insert at position

4. Delete at beginning

5. Delete at end

6. Delete at position

7. Insert before an element

8. Insert after an element

9. Delete before an element

10. Delete after an element

11. Reverse the array

12. Search for an element

13. Display array

14. Exit

Enter your choice: 1

Enter element to insert at beginning: 2

------ Menu ------

1. Insert at beginning

2. Insert at end

3. Insert at position

4. Delete at beginning

5. Delete at end

6. Delete at position

7. Insert before an element

8. Insert after an element

9. Delete before an element

10. Delete after an element

11. Reverse the array

12. Search for an element

13. Display array

14. Exit

Enter your choice: 1

Enter element to insert at beginning: 3

------ Menu ------

1. Insert at beginning

2. Insert at end

3. Insert at position

4. Delete at beginning

5. Delete at end

6. Delete at position

7. Insert before an element

8. Insert after an element

9. Delete before an element

10. Delete after an element

11. Reverse the array

12. Search for an element

13. Display array

14. Exit

Enter your choice: 1

Enter element to insert at beginning: 4

------ Menu ------

1. Insert at beginning

2. Insert at end

3. Insert at position

4. Delete at beginning

5. Delete at end

6. Delete at position

7. Insert before an element

8. Insert after an element

9. Delete before an element

10. Delete after an element

11. Reverse the array

12. Search for an element

13. Display array

14. Exit

Enter your choice: 2

Enter element to insert at end: 5

------ Menu ------

1. Insert at beginning

2. Insert at end

3. Insert at position

4. Delete at beginning

5. Delete at end

6. Delete at position

7. Insert before an element

8. Insert after an element

9. Delete before an element

10. Delete after an element

11. Reverse the array

12. Search for an element

13. Display array

14. Exit

Enter your choice: 2

Enter element to insert at end: 6

------ Menu ------

1. Insert at beginning

2. Insert at end

3. Insert at position

4. Delete at beginning

5. Delete at end

6. Delete at position

7. Insert before an element

8. Insert after an element

9. Delete before an element

10. Delete after an element

11. Reverse the array

12. Search for an element

13. Display array

14. Exit

Enter your choice: 2

Enter element to insert at end: 7

------ Menu ------

1. Insert at beginning

2. Insert at end

3. Insert at position

4. Delete at beginning

5. Delete at end

6. Delete at position

7. Insert before an element

8. Insert after an element

9. Delete before an element

10. Delete after an element

11. Reverse the array

12. Search for an element

13. Display array

14. Exit

Enter your choice: 2

Enter element to insert at end: 8

------ Menu ------

1. Insert at beginning

2. Insert at end

3. Insert at position

4. Delete at beginning

5. Delete at end

6. Delete at position

7. Insert before an element

8. Insert after an element

9. Delete before an element

10. Delete after an element

11. Reverse the array

12. Search for an element

13. Display array

14. Exit

Enter your choice: 13

Array: 4 3 2 1 5 6 7 8

------ Menu ------

1. Insert at beginning

2. Insert at end

3. Insert at position

4. Delete at beginning

5. Delete at end

6. Delete at position

7. Insert before an element

8. Insert after an element

9. Delete before an element

10. Delete after an element

11. Reverse the array

12. Search for an element

13. Display array

14. Exit

Enter your choice: 10

Enter element to delete after: 4

------ Menu ------

1. Insert at beginning

2. Insert at end

3. Insert at position

4. Delete at beginning

5. Delete at end

6. Delete at position

7. Insert before an element

8. Insert after an element

9. Delete before an element

10. Delete after an element

11. Reverse the array

12. Search for an element

13. Display array

14. Exit

Enter your choice: 11

Array reversed.

------ Menu ------

1. Insert at beginning

2. Insert at end

3. Insert at position

4. Delete at beginning

5. Delete at end

6. Delete at position

7. Insert before an element

8. Insert after an element

9. Delete before an element

10. Delete after an element

11. Reverse the array

12. Search for an element

13. Display array

14. Exit

Enter your choice: 13

Array: 8 7 6 5 1 2 4

Test Case: Any two (screenshot)

Conclusion: Utilizing switch cases, we can execute diverse operations such as insertion, deletion, and searching within an array by traversing through indices. This approach provides a versatile and efficient means of managing array elements, contributing to enhanced understanding and proficiency in array manipulation and programming logic.

Name of Student: JEEVAN NAIDU

Roll Number: 150096723015

Experiment No: 02

Title: Implement Stack ADT using array.

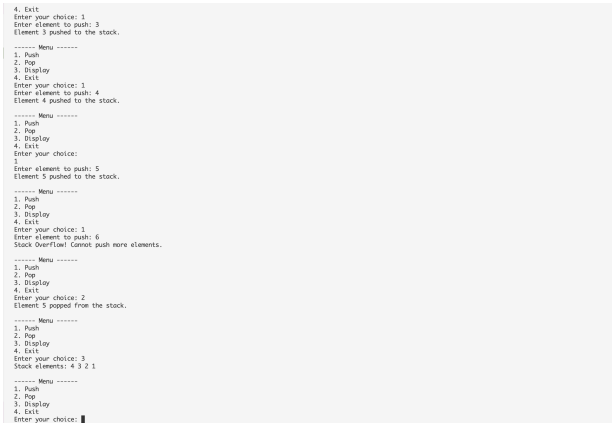
Theory: A stack is a linear data structure that follows the Last In, First Out (LIFO) principle. In this implementation, an array is used to represent the stack, with a fixed maximum size. The top of the stack indicates the index of the last inserted element, initialized as -1 for an empty stack. Key operations include push, which adds an element to the top of the stack, and pop, which removes the topmost element from the stack. Additionally, methods for checking if the stack is empty or full, and for displaying its contents, are provided.

Code:

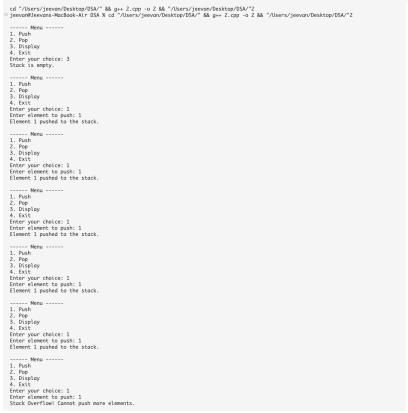
```
#include<iostream>

using namespace std;

const int MAX_SIZE = 10; // Maximum size of the stack

class Stack {
private:
    int arr[MAX_SIZE];
    int top;

public:
    Stack() {
        top = -1; // Initialize top of stack to -1
    }

    bool isEmpty() {
        return top == -1;
    }

    bool isFull() {
        return top == MAX_SIZE - 1;
    }

    void push(int val) {
        if (isFull()) {
            cout << "Stack Overflow! Cannot push more elements." << endl;
        } else {
            arr[++top] = val;
            cout << "Element " << val << " pushed to the stack." << endl;
        }
    }

    void pop() {
        if (isEmpty()) {
            cout << "Stack Underflow! Cannot pop from an empty stack." << endl;
```

```
        } else {
            cout << "Element " << arr[top--] << " popped from the stack." << endl;
        }
    }

    void display() {
        if (isEmpty()) {
            cout << "Stack is empty." << endl;
        } else {
            cout << "Stack elements: ";
            for (int i = top; i >= 0; i--) {
                cout << arr[i] << " ";
            }
            cout << endl;
        }
    }
};

int main() {
    Stack stack;
    int choice, element;

    do {
        cout << "\n------ Menu ------" << endl;
        cout << "1. Push" << endl;
        cout << "2. Pop" << endl;
        cout << "3. Display" << endl;
        cout << "4. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter element to push: ";
                cin >> element;
                stack.push(element);
                break;
            case 2:
                stack.pop();
                break;
            case 3:
                stack.display();
                break;
            case 4:
                cout << "Exiting..." << endl;
                break;
            default:
                cout << "Invalid choice. Please try again." << endl;
        }

    } while (choice != 4);

    return 0;
}
```

Output: (screenshot)



Test Case: Any two (screenshot)



Conclusion:

This program demonstrates the functionality of a stack through a menu-driven interface. By utilizing push and pop operations, elements can be efficiently added to and removed from the stack. The display function offers visibility into the current state of the stack. Through this implementation, a clear understanding of stack operations and their utilization in programming is acquired, contributing to enhanced problem-solving skills and proficiency in data structure concepts.

Name of Student: JEEVAN NAIDU

Roll Number: 150096723015

Experiment No: 03

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

Theory:The provided program demonstrates the conversion of an infix expression to a postfix expression using a stack-based approach. In this implementation, a custom Stack class is utilized, which employs a dynamic array to store characters. The program utilizes the rules of infix to postfix conversion, where operands are directly added to the output string, while operators are pushed onto the stack according to their precedence. When encountering a closing parenthesis, operators are popped from the stack and appended to the output string until an opening parenthesis is encountered, ensuring correct expression evaluation.

Code:#include<iostream>

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

class Stack {
private:
    char* arr;
    int top;
    int capacity;

public:
    Stack(int size) {
        capacity = size;
        arr = new char[capacity];
        top = -1;
    }

    ~Stack() {
        delete[] arr;
    }

    bool isEmpty() {
        return top == -1;
    }

    bool isFull() {
        return top == capacity - 1;
    }
```

```
    void push(char ch) {
        if (isFull()) {
            cout << "Stack Overflow! Cannot push more elements." << endl;
        } else {
            arr[++top] = ch;
        }
    }

    char pop() {
        if (isEmpty()) {
            cout << "Stack Underflow! Cannot pop from an empty stack." << endl;
            return '\0';
        } else {
            return arr[top--];
        }
    }

    char peek() {
        if (isEmpty()) {
            cout << "Stack is empty." << endl;
            return '\0';
        } else {
            return arr[top];
        }
    }
};

bool isOperator(char ch) {
    return ch == '+' || ch == '-' || ch == '*' || ch == '/';
}

int precedence(char op) {
    if (op == '+' || op == '-')
        return 1;
    if (op == '*' || op == '/')
        return 2;
    return 0;
}

string infixToPostfix(const string& infix) {
    Stack st(infix.length());
    string postfix = "";

    for (char ch : infix) {
        if (isalnum(ch)) {
            postfix += ch;
        } else if (ch == '(') {
            st.push(ch);
        } else if (ch == ')') {
            while (!st.isEmpty() && st.peek() != '(') {
                postfix += st.pop();
            }
            st.pop(); // Pop '('
        } else { // Operator
            while (!st.isEmpty() && precedence(st.peek()) >= precedence(ch)) {
                postfix += st.pop();
            }
            st.push(ch);
        }
    }

    while (!st.isEmpty()) {
        postfix += st.pop();
    }

    return postfix;
}
```

```
int main() {
    string infix, postfix;

    cout << "Enter the infix expression: ";
    cin >> infix;

    postfix = infixToPostfix(infix);
    cout << "Postfix expression: " << postfix << endl;

    return 0;
}
```

Output: (screenshot)

```
cd "/Users/jeevan/Desktop/DSA/" && g++ 3.cpp -o 3 && "/Users/jeevan/Desktop/DSA/"3
jeevan@Jeevans-MacBook-Air DSA % cd "/Users/jeevan/Desktop/DSA/" && g++ 3.cpp -o 3 && "/Users/jeevan/Desktop/DSA/"3
3.cpp:73:18: warning: range-based for loop is a C++11 extension [-Wc++11-extensions]
    for (char ch : infix) {
                 ^
1 warning generated.
Enter the infix expression: (A+B)*(C*D)
Postfix expression: AB+CD**
jeevan@Jeevans-MacBook-Air DSA %
```

Test Case: Any two (screenshot)

```
cd "/Users/jeevan/Desktop/DSA/" && g++ 3.cpp -o 3 && "/Users/jeevan/Desktop/DSA/"3
jeevan@Jeevans-MacBook-Air DSA % cd "/Users/jeevan/Desktop/DSA/" && g++ 3.cpp -o 3 && "/Users/jeevan/Desktop/DSA/"3
3.cpp:73:18: warning: range-based for loop is a C++11 extension [-Wc++11-extensions]
    for (char ch : infix) {
                 ^
1 warning generated.
Enter the infix expression: (A*B)+(C*D)*P-(Q-D))
Stack Underflow! Cannot pop from an empty stack.
Postfix expression: AB*CD*P*+QD--
jeevan@Jeevans-MacBook-Air DSA %
```

```
cd "/Users/jeevan/Desktop/DSA/" && g++ 3.cpp -o 3 && "/Users/jeevan/Desktop/DSA/"3
jeevan@Jeevans-MacBook-Air DSA % cd "/Users/jeevan/Desktop/DSA/" && g++ 3.cpp -o 3 && "/Users/jeevan/Desktop/DSA/"3
3.cpp:73:18: warning: range-based for loop is a C++11 extension [-Wc++11-extensions]
    for (char ch : infix) {
                 ^
1 warning generated.
Enter the infix expression: (R*Q)*(Y+W)
Postfix expression: RQ*YW+*
jeevan@Jeevans-MacBook-Air DSA %
```

Conclusion: Through this program, the conversion of infix expressions to postfix expressions is efficiently accomplished. The stack-based algorithm enables the handling of operators and parentheses, ensuring proper precedence and order of

evaluation. By leveraging the stack data structure, this implementation offers a clear demonstration of how fundamental data structures can be utilized to solve complex problems in expression evaluation and parsing.

Name of Student: JEEVAN NAIDU

Roll Number: 150096723015

Experiment No: 04

Title: Evaluate Postfix Expression using Stack ADT.

Theory: This program showcases the evaluation of a postfix expression using a stack-based approach. It utilizes a custom Stack class implemented with dynamic array storage. The postfix expression is iterated character by character, and operands are pushed onto the stack. When encountering an operator, the required number of operands is popped from the stack, and the operation is performed, with the result pushed back onto the stack. This process continues until the entire expression is evaluated.

Code:

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

class Stack {
private:
    int* arr;
    int top;
    int capacity;

public:
    Stack(int size) {
        capacity = size;
        arr = new int[capacity];
        top = -1;
    }

    ~Stack() {
        delete[] arr;
    }

    bool isEmpty() {
        return top == -1;
    }

    bool isFull() {
```

```
        return top == capacity - 1;
    }

    void push(int value) {
        if (isFull()) {
            cout << "Stack Overflow! Cannot push more elements." << endl;
            return;
        }
        top++;
        arr[top] = value;
    }

    int pop() {
        if (isEmpty()) {
            cout << "Stack Underflow! Cannot pop from an empty stack." << endl;
            return -1; // or throw an exception
        }
        int value = arr[top];
        top--;
        return value;
    }

    int peek() {
        if (isEmpty()) {
            cout << "Stack is empty." << endl;
            return -1; // or throw an exception
        }
        return arr[top];
    }
};

bool isOperator(char ch) {
    return ch == '+' || ch == '-' || ch == '*' || ch == '/';
}

int evaluatePostfix(const string& postfix) {
    int len = postfix.length();
    Stack st(len);

    for (int i = 0; i < len; i++) {
        char ch = postfix[i];
        if (isdigit(ch)) {
            st.push(ch - '0');
        } else if (isOperator(ch)) {
            int operand2 = st.pop();
            int operand1 = st.pop();

            switch (ch) {
                case '+':
                    st.push(operand1 + operand2);
                    break;
                case '-':
                    st.push(operand1 - operand2);
                    break;
                case '*':
                    st.push(operand1 * operand2);
                    break;
                case '/':
```

```
                    st.push(operand1 / operand2);
                    break;
                }
            }
        }

    return st.pop();
}

int main() {
    string postfix;

    cout << "Enter the postfix expression: ";
    cin >> postfix;

    int result = evaluatePostfix(postfix);
    cout << "Result of postfix expression: " << result << endl;

    return 0;
}
```

Output: (screenshot)

```
cd "/Users/jeevan/Desktop/DSA/" && g++ 4.cpp -o 4 && "/Users/jeevan/Desktop/DSA/"4
jeevan@Jeevans-MacBook-Air DSA % cd "/Users/jeevan/Desktop/DSA/" && g++ 4.cpp -o 4 && "/Users/jeevan/Desktop/DSA/"4
Enter the postfix expression: 13-51**
Result of postfix expression: -10
jeevan@Jeevans-MacBook-Air DSA %
```

Test Case: Any two (screenshot)

```
cd "/Users/jeevan/Desktop/DSA/" && g++ 4.cpp -o 4 && "/Users/jeevan/Desktop/DSA/"4
jeevan@Jeevans-MacBook-Air DSA % cd "/Users/jeevan/Desktop/DSA/" && g++ 4.cpp -o 4 && "/Users/jeevan/Desktop/DSA/"4
Enter the postfix expression: 13-90*0+-
Result of postfix expression: -2
jeevan@Jeevans-MacBook-Air DSA % cd "/Users/jeevan/Desktop/DSA/" && g++ 4.cpp -o 4 && "/Users/jeevan/Desktop/DSA/"4
Enter the postfix expression: 50*23+-
Result of postfix expression: -5
jeevan@Jeevans-MacBook-Air DSA %
```

Conclusion: Through this program, the evaluation of postfix expressions is efficiently demonstrated using the stack data structure. By leveraging stack operations for operand and operator manipulation, the program ensures correct expression evaluation based on postfix notation. This implementation exemplifies the versatility and utility of stacks in solving mathematical problems, providing a clear understanding of stack-based algorithms in expression evaluation.

Name of Student: JEEVAN NAIDU

Roll Number: 150096723015

Experiment No: 05

Title: Implement Linear Queue ADT using array.

Theory:The provided program implements a queue using an array-based approach. Queues represent a linear data structure that follows the First In, First Out (FIFO) principle. This implementation utilizes an array to store elements, with a fixed maximum size defined by the capacity variable. The front and rear pointers indicate the front and rear of the queue, respectively. Key operations include enqueue, which adds an element to the rear of the queue, and dequeue, which removes an element from the front of the queue. Additional functionalities such as peeking at the front element and displaying the entire queue are also implemented.

Code:

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

const int MAX_SIZE = 5; // Maximum size of the queue

class Queue {
private:
    int* arr;
    int front;
    int rear;
    int capacity;

public:
    Queue(int size) {
        capacity = size;
        arr = new int[capacity];
        front = rear = -1;
    }

    ~Queue() {
        delete[] arr;
    }
```

```
    bool isEmpty() {
        return front == -1;
    }

    bool isFull() {
        return (rear + 1) % capacity == front;
    }

    void enqueue(int value) {
        if (isFull()) {
            cout << "Queue Overflow! Cannot enqueue more elements." << endl;
            return;
        }
        if (isEmpty())
            front = 0;
        rear = (rear + 1) % capacity;
        arr[rear] = value;
        cout << "Enqueued element: " << value << endl;
    }

    int dequeue() {
        if (isEmpty()) {
            cout << "Queue Underflow! Cannot dequeue from an empty queue." << endl;
            return -1; // or throw an exception
        }
        int value = arr[front];
        if (front == rear) // If there is only one element in the queue
            front = rear = -1;
        else
            front = (front + 1) % capacity;
        cout << "Dequeued element: " << value << endl;
        return value;
    }

    int peek() {
        if (isEmpty()) {
            cout << "Queue is empty." << endl;
            return -1; // or throw an exception
        }
        return arr[front];
    }

    void display() {
        if (isEmpty()) {
            cout << "Queue is empty." << endl;
            return;
        }
        cout << "Queue elements: ";
        int i = front;
        do {
            cout << arr[i] << " ";
            i = (i + 1) % capacity;
        } while (i != (rear + 1) % capacity);
        cout << endl;
    }
};

int main() {
    Queue queue(MAX_SIZE);
    int choice, element;

    // Menu-driven interface
    do {
```

```
        cout << "\n------ Menu ------" << endl;
        cout << "1. Enqueue" << endl;
        cout << "2. Dequeue" << endl;
        cout << "3. Peek" << endl;
        cout << "4. Display" << endl;
        cout << "5. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter element to enqueue: ";
                cin >> element;
                queue.enqueue(element);
                break;
            case 2:
                queue.dequeue();
                break;
            case 3:
                element = queue.peek();
                if (element != -1)
                    cout << "Front element: " << element << endl;
                break;
            case 4:
                queue.display();
                break;
            case 5:
                cout << "Exiting..." << endl;
                break;
            default:
                cout << "Invalid choice. Please try again." << endl;
        }

    } while (choice != 5);

    return 0;
}
```

Output: (screenshot)

Test Case: Any two (screenshot)

```
cd "/Users/jeevan/Desktop/DSA/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/Users/jeevan/Desktop/DSA/"tempCodeRunnerFile
jeevan@Jeevans-MacBook-Air DSA % cd "/Users/jeevan/Desktop/DSA/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/Users/jeevan/Desktop/DSA/"tempCodeRunnerFile

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter element to enqueue: 1
Enqueued element: 1

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter element to enqueue: 2
Enqueued element: 2

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 4
Queue elements: 1 2

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 2
Dequeued element: 1

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 4
Queue elements: 2
```

```
cd "/Users/jeevan/Desktop/DSA/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/Users/jeevan/Desktop/DSA/"tempCodeRunnerFile
jeevan@Jeevans-MacBook-Air DSA % cd "/Users/jeevan/Desktop/DSA/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/Users/jeevan/Desktop/DSA/"tempCodeRunnerFile

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 2
Queue Underflow! Cannot dequeue from an empty queue.

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter element to enqueue: 2
Enqueued element: 2

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 4
Queue elements: 2

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 5
Exiting...
jeevan@Jeevans-MacBook-Air DSA %
```

Conclusion: Through this program, the functionality of a queue is effectively demonstrated using an array-based approach. The menu-driven interface offers users the flexibility to enqueue, dequeue, peek at the front element, and display the queue's contents. By leveraging fundamental queue operations and array manipulation, the program provides insights into the implementation and utilization of queues in various applications, enhancing understanding and proficiency in data structure concepts and programming logic.

Name of Student: JEEVAN NAIDU

Roll Number: 150096723015

Experiment No: 06

Title: Implement Circular Queue ADT using array.

Theory:The program demonstrates the implementation of a circular queue, a variant of the traditional queue data structure. In a circular queue, elements are stored in a circular manner, allowing efficient utilization of memory. This implementation employs an array to store queue elements, with pointers for front and rear indicating the current positions. Key operations such as enqueue, dequeue, peek, and display are provided, ensuring efficient management of elements in the circular queue. Additionally, the circular nature of the queue ensures optimal usage of available space, enabling seamless insertion and removal of elements.

Code: #include <iostream>

```
using namespace std;

const int MAX_SIZE = 5; // Maximum size of
the circular queue

class CircularQueue {
private:
    int* arr;
    int front;
    int rear;
    int capacity;
```

```
public:
    CircularQueue(int size) {
        capacity = size;
        arr = new int[capacity];
        front = rear = -1;
    }

    ~CircularQueue() {
        delete[] arr;
    }

    bool isEmpty() {
        return front == -1;
    }

    bool isFull() {
        return (rear + 1) % capacity ==
front;
    }

    void enqueue(int value) {
        if (isFull()) {
            cout << "Circular Queue Overflow!
Cannot enqueue more elements." << endl;
            return;
        }
```

```
        if (isEmpty())
            front = 0;
        rear = (rear + 1) % capacity;
        arr[rear] = value;
        cout << "Enqueued element: " << value
<< endl;
    }

    int dequeue() {
        if (isEmpty()) {
            cout << "Circular Queue
Underflow! Cannot dequeue from an empty
queue." << endl;
            return -1; // or throw an
exception
        }
        int value = arr[front];
        if (front == rear) // If there is
only one element in the queue
            front = rear = -1;
        else
            front = (front + 1) % capacity;
        cout << "Dequeued element: " << value
<< endl;
        return value;
    }
```

```
    int peek() {
        if (isEmpty()) {
            cout << "Circular Queue is
empty." << endl;
            return -1; // or throw an
exception
        }
        return arr[front];
    }

    void display() {
        if (isEmpty()) {
            cout << "Circular Queue is
empty." << endl;
            return;
        }
        cout << "Circular Queue elements: ";
        int i = front;
        do {
            cout << arr[i] << " ";
            i = (i + 1) % capacity;
        } while (i != (rear + 1) % capacity);
        cout << endl;
    }
};

int main() {
```

```
    CircularQueue circularQueue(MAX_SIZE);
    int choice, element;

    // Menu-driven interface
    do {
        cout << "\n------ Menu ------" <<
endl;
        cout << "1. Enqueue" << endl;
        cout << "2. Dequeue" << endl;
        cout << "3. Peek" << endl;
        cout << "4. Display" << endl;
        cout << "5. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter element to
enqueue: ";
                cin >> element;

circularQueue.enqueue(element);
                break;
            case 2:
                circularQueue.dequeue();
                break;
            case 3:
```

```
                element =
circularQueue.peek();
                if (element != -1)
                    cout << "Front element: "
<< element << endl;
                break;
            case 4:
                circularQueue.display();
                break;
            case 5:
                cout << "Exiting..." << endl;
                break;
            default:
                cout << "Invalid choice.
Please try again." << endl;
        }

    } while (choice != 5);

    return 0;
}
```

Output: (screenshot)

```
cd "/Users/jeevan/Desktop/DSA/" && g++ 6.cpp -o 6 && "/Users/jeevan/Desktop/DSA/"6
jeevan@Jeevans-MacBook-Air DSA % cd "/Users/jeevan/Desktop/DSA/" && g++ 6.cpp -o 6 && "/Users/jeevan/Desktop/DSA/"6

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter element to enqueue: 1
Enqueued element: 1

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 2
Dequeued element: 1

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter element to enqueue: 3
Enqueued element: 3

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter element to enqueue: 5
Enqueued element: 5

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 4
Circular Queue elements: 3 5
```

Test Case: Any two (screenshot)

```
cd "/Users/jeevan/Desktop/DSA/" && g++ 6.cpp -o 6 && "/Users/jeevan/Desktop/DSA/"6
jeevan@Jeevans-MacBook-Air DSA % cd "/Users/jeevan/Desktop/DSA/" && g++ 6.cpp -o 6 && "/Users/jeevan/Desktop/DSA/"6

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 2
Circular Queue Underflow! Cannot dequeue from an empty queue.

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter element to enqueue: 4
Enqueued element: 4

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter element to enqueue: 5
Enqueued element: 5

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 4
Circular Queue elements: 4 5

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice:
```

```
cd "/Users/jeevan/Desktop/DSA/" && g++ 6.cpp -o 6 && "/Users/jeevan/Desktop/DSA/"6
jeevan@Jeevans-MacBook-Air DSA % cd "/Users/jeevan/Desktop/DSA/" && g++ 6.cpp -o 6 && "/Users/jeevan/Desktop/DSA/"6

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter element to enqueue: 1
Enqueued element: 1

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter element to enqueue: 2
Enqueued element: 2

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter element to enqueue: 3
Enqueued element: 3

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice:
4
Circular Queue elements: 1 2 3

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice:
```

Conclusion:Through this program, the functionality and benefits of a circular queue are showcased. The menu-driven interface offers users a convenient means to enqueue, dequeue, peek at the front element, and display the contents of the circular queue. By leveraging circular queue operations and array manipulation, the program demonstrates the efficiency and versatility of circular queues in various applications. This implementation enhances understanding and proficiency in data structure concepts and programming logic, contributing to effective problem-solving skills.

Name of Student: **JEEVAN NAIDU**

Roll Number: **150096723015**

Experiment No: 07

Title: Implement Singly Linked List ADT.

Theory: The program implements a singly linked list data structure in C++, providing various operations such as insertion, deletion, searching, and display. The linked list is a dynamic data structure where each element (node) contains a data field and a pointer to the next node. The Linklist class defines methods to perform operations on the linked list, including insertion at the beginning, end, or at a specific location, deletion from the beginning or end, deletion before or after a given element, and searching for an element. Additionally, the program provides a user-friendly menu-driven interface to interact with the linked list, enabling users to perform desired operations conveniently.

Code:

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

class Linklist
{
    private:
        struct Node
        {
            int data;
            Node*link;
        }*START;
    public:
        Linklist();
        void insert (int no);
```

```
        void display ();
        void insertbeg(int num);
        void insert_at_location(int loc,int
num);
        void insert_before_element(int
bnum,int num);
        void deletebeg();
        void deletend();
        void delete_before(int num);
        void delete_after(int num);
        void search(int num);
        ~Linklist ();
};
Linklist::Linklist()
{
    START=NULL;
}
Linklist::~Linklist()
{
    Node*q;
    while (START!=NULL)
    {
        q=START->link;
        delete START;
        START=q;
    }
}
```

```
void Linklist::display()
{
    Node*PTR=START;
    while(PTR!=0)
    {
        cout<<PTR->data<<" ";
        PTR=PTR->link;
    }
    cout<<endl;
}
void Linklist::insert (int num)
{
    Node*PTR,*r;
    if(START==NULL)
    {
        PTR=new Node;
        PTR->data=num;
        PTR->link=NULL;
        START=PTR;
    }
    else
    {
        PTR=START;
        while (PTR->link !=NULL)
            PTR=PTR->link;

        r=new Node;
```

```
        r->data=num;
        r->link=NULL;
        PTR->link=r;
    }
}
void Linklist::insertbeg(int num)
{
    Node*PTR;
    PTR=new Node;
    PTR->data=num;
    PTR->link=START;
    START=PTR;
}
void Linklist::insert_at_location(int loc,
int num)
{
    Node*PTR,*r;
    PTR=START;
    for(int i=0;i<loc;i++)
    {
        PTR=PTR->link;
        if(PTR==NULL)
        {
            cout<<"There are less than
"<<loc<<" elements in list "<<endl;
            return ;
        }
```

```
    }
    r=new Node;
    r->data=num;
    r->link=PTR->link;
    PTR->link=r;
}
void Linklist::insert_before_element(int
bnum,int num)
{
    Node*PTR,*r;
    PTR=START;
    Node* prev=nullptr;
    if(PTR !=nullptr && PTR->data==bnum)
    {
        r=new Node;
        r->data=num;
        r->link=PTR;
        START=r;
        return;
    }
    while (PTR!=nullptr &&PTR->data !=bnum)
    {
        prev=PTR;
        PTR=PTR->link;
    }
    if(PTR==nullptr)
    {
```

```
        cout<<"Element " << bnum << " not
found in the linked list." << endl;
        return;
    }
    r=new Node;
    r->data=num;
    prev->link=r;
    r->link=PTR;
}
void Linklist::deletebeg()
{
    Node*PTR,*r;
    if(START ==nullptr)
    {
        cout<<"Linked list is empty. Cannot
delete from an empty list." << endl;
        return;
    }
    r=START;
    START=START->link;
    delete r;
}
void Linklist::deletend()
{
    if (START == nullptr) {
        cout << "Linked list is empty. Cannot
delete from an empty list." << endl;
```

```
        return;
    }
    if (START->link == nullptr) {
        delete START;
        START = nullptr;
        return;
    }
    Node* prev = nullptr;
    Node*PTR;
    PTR=START;
    while (PTR->link != nullptr)
    {
        prev = PTR;
        PTR = PTR->link;
    }
    prev->link = nullptr;
    delete PTR;
}
void Linklist::delete_before(int num)
{
    Node*PTR,*prev;
    cout<<"Enter the number of which before
element has to be deleted: ";
    cin>>num;
    PTR=START;
    prev=nullptr;
```

```
    if (START == nullptr || START->link ==
nullptr)
    {
        cout << "Cannot delete before. Linked
list does not have enough elements." << endl;
        return;
    }
    while (PTR != nullptr && PTR->data !=
num)
    {
        prev = PTR;
        PTR = PTR->link;
    }
    if (PTR == nullptr || PTR == START)
    {
        cout << "Element " << num << " not
found in the linked list." << endl;
        return;
    }
    if (prev == nullptr)
    {
        cout << "No node to delete before the
first node." << endl;
        return;
    }
    Node* toDelete = prev->link;
    prev->link = PTR;
```

```
    delete toDelete;
}
void Linklist::delete_after(int num)
{
    Node*PTR,*after;
    cout<<"Enter the number of which after
element has to be deleted: ";
    cin>>num;
    PTR=START;
    after=nullptr;
    if (START == nullptr || START->link ==
nullptr)
    {
        cout << "Cannot delete after. Linked
list does not have enough elements." << endl;
        return;
    }
    while (PTR != nullptr && PTR->data !=
num)
    {
        PTR = PTR->link;
    }
    if (PTR == nullptr)
    {
        cout << "Element " << num << " not
found in the linked list." << endl;
```

```
        return;
    }
    if (PTR->link == nullptr) {
        cout << "No node to delete after the
last node." << endl;
        return;
    }
    after = PTR->link;
    PTR->link = after->link;

    delete after;
}
void Linklist::search(int num)
{
    Node*PTR;
    PTR=START;
    int count;
    cout<<"Enter the element you search for:
";
    while(PTR->data!=num)
    {
        count++;
        PTR=PTR->link;
    }
    cout<<"Element "<<num<<" found at
"<<count;
}
```

```
void all_opp(Linklist& l)
{
    int choice, num, loc, bnum;
    do {
        cout << "1. Insert\n";
        cout << "2. Insert at Beginning\n";
        cout << "3. Insert at Location\n";
        cout << "4. Insert Before Element\n";
        cout << "5. Delete from Beginning\n";
        cout << "6. Delete from End\n";
        cout << "7. Delete Before Element\n";
        cout << "8. Delete After Element\n";
        cout << "9. Search\n";
        cout << "10. Display\n";
        cout << "11. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice)
        {
        case 1:
            cout << "Enter the number to
insert: ";
            cin >> num;
            l.insert(num);
            break;
        case 2:
```

```
            cout << "Enter the number to
insert at beginning: ";
            cin >> num;
            l.insertbeg(num);
            break;
        case 3:
            cout << "Enter the number to
insert: ";
            cin >> num;
            cout << "Enter the location: ";
            cin >> loc;
            l.insert_at_location(loc, num);
            break;
        case 4:
            cout << "Enter the number to
insert before: ";
            cin >> bnum;
            cout << "Enter the number to
insert: ";
            cin >> num;
            l.insert_before_element(bnum,
num);
            break;
        case 5:
            l.deletebeg();
            break;
        case 6:
```

```
            l.deletend();
            break;
        case 7:
            cout << "Enter the number of
which before element has to be deleted: ";
            cin >> num;
            l.delete_before(num);
            break;
        case 8:
            cout << "Enter the number of
which after element has to be deleted: ";
            cin >> num;
            l.delete_after(num);
            break;
        case 9:
            cout << "Enter the element you
want to search: ";
            cin >> num;
            l.search(num);
            break;
        case 10:
            cout << "The linked list
contains: ";
            l.display();
            break;
        case 11:
            cout << "Exiting...\n";
```

```
            break;
        default:
            cout << "Invalid choice. Please
enter a valid option.\n";
            break;
        }
    } while (choice != 11);
}

int main()
{
    Linklist l;
    int num;
    cout << "Enter the number of elements to
store: ";
    cin >> num;

    for (int i = 0; i < num; i++)
    {
        int a;
        cout<<"Enter the "<<i+1<< " element:
";
        cin>>a;
        l.insert(a);
    }
    all_opp(l);
    return 0;
```

```
}
```

Output: (screenshot)

```
cd "/Users/jeevan/Desktop/DSA/" && g++ 7.cpp -o 7 && "/Users/jeevan/Desktop/DSA/"7
jeevan@Jeevans-MacBook-Air DSA % cd "/Users/jeevan/Desktop/DSA/" && g++ 7.cpp -o 7 && "/Users/jeevan/Desktop/DSA/"7
Enter the number of elements to store: 5
Enter the 1 element: 1
Enter the 2 element: 2
Enter the 3 element: 3
Enter the 4 element: 4
Enter the 5 element:
5
1. Insert
2. Insert at Beginning
3. Insert at Location
4. Insert Before Element
5. Delete from Beginning
6. Delete from End
7. Delete Before Element
8. Delete After Element
9. Search
10. Display
11. Exit
Enter your choice: 10
The linked list contains: 1 2 3 4 5
1. Insert
2. Insert at Beginning
3. Insert at Location
4. Insert Before Element
5. Delete from Beginning
6. Delete from End
7. Delete Before Element
8. Delete After Element
9. Search
10. Display
11. Exit
Enter your choice: 5
1. Insert
2. Insert at Beginning
3. Insert at Location
4. Insert Before Element
5. Delete from Beginning
6. Delete from End
7. Delete Before Element
8. Delete After Element
9. Search
10. Display
11. Exit
Enter your choice: 10
The linked list contains: 2 3 4 5
```

Test Case: Any two (screenshot)

```
cd "/Users/jeevan/Desktop/DSA/" && g++ 7.cpp -o 7 && "/Users/jeevan/Desktop/DSA/"7
jeevan@Jeevans-MacBook-Air DSA % cd "/Users/jeevan/Desktop/DSA/" && g++ 7.cpp -o 7 && "/Users/jeevan/Desktop/DSA/"7
Enter the number of elements to store: 3
Enter the 1 element: 1
Enter the 2 element: 2
Enter the 3 element: 3
1. Insert
2. Insert at Beginning
3. Insert at Location
4. Insert Before Element
5. Delete from Beginning
6. Delete from End
7. Delete Before Element
8. Delete After Element
9. Search
10. Display
11. Exit
Enter your choice: 9
Enter the element you want to search: 2
Enter the element you search for: Element 2 found at 11. Insert
```

```
cd "/Users/jeevan/Desktop/DSA/" && g++ 7.cpp -o 7 && "/Users/jeevan/Desktop/DSA/"7
jeevan@Jeevans-MacBook-Air DSA % cd "/Users/jeevan/Desktop/DSA/" && g++ 7.cpp -o 7 && "/Users/jeevan/Desktop/DSA/"7
Enter the number of elements to store: 3
Enter the 1 element: 1
Enter the 2 element: 2
Enter the 3 element: 3
1. Insert
2. Insert at Beginning
3. Insert at Location
4. Insert Before Element
5. Delete from Beginning
6. Delete from End
7. Delete Before Element
8. Delete After Element
9. Search
10. Display
11. Exit
Enter your choice: 7
Enter the number of which before element has to be deleted: 2
Enter the number of which before element has to be deleted: 2
1. Insert
2. Insert at Beginning
3. Insert at Location
4. Insert Before Element
5. Delete from Beginning
6. Delete from End
7. Delete Before Element
8. Delete After Element
9. Search
10. Display
11. Exit
Enter your choice: 10
The linked list contains: 1 0
```

Conclusion:

Through this program, the functionality and versatility of singly linked lists are demonstrated, showcasing their ability to efficiently manage dynamic data. Users can easily manipulate the linked list by inserting, deleting, or searching for elements based on their requirements. The menu-driven interface enhances user experience, providing a seamless interaction with the linked list operations. By leveraging the power of dynamic memory allocation and pointer manipulation, the program offers a practical implementation of linked lists, fostering a deeper understanding of fundamental data structure concepts and programming techniques.

Name of Student: JEEVAN NAIDU

Roll Number: 150096723015

Experiment No: 08

Title: Implement Circular Linked List ADT.

Theory: The program implements a circular linked list in C++, where each node contains an integer data field and a pointer to the next node. Circular linked lists are similar to singly linked lists, but the last node points back to the first node, creating a circular structure. The CircularLinkedList class provides methods to perform various operations on the circular linked list, including insertion, deletion, searching, display, getting the count of nodes, and deleting the entire list. Nodes can be inserted at the end of the list, and elements can be deleted based on their value. Additionally, the program offers a user-friendly menu-driven interface to interact with the circular linked list, enabling users to perform desired operations conveniently.

Code:

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

class CircularLinkedList {
private:
    struct Node {
        int data;
        Node* next;
    }* head;

public:
    CircularLinkedList() {
        head = nullptr;
    }
```

```
    // Function to insert a new node at the
end of the circular linked list
    void insert(int value) {
        Node* newNode = new Node;
        newNode->data = value;
        newNode->next = nullptr;

        if (head == nullptr) {
            head = newNode;
            head->next = head; // Pointing
back to itself for circularity
        }
        else {
            Node* temp = head;
            while (temp->next != head) {
                temp = temp->next;
            }
            temp->next = newNode;
            newNode->next = head; // Pointing
back to head for circularity
        }
    }

    // Function to display the elements of
the circular linked list
    void display() {
        if (head == nullptr) {
```

```
            cout << "Circular Linked List is
empty" << endl;
            return;
        }

        Node* temp = head;
        do {
            cout << temp->data << " ";
            temp = temp->next;
        } while (temp != head);
        cout << endl;
    }

    // Function to delete a node with the
given value from the circular linked list
    void deleteNode(int value) {
        if (head == nullptr) {
            cout << "Circular Linked List is
empty. Cannot delete." << endl;
            return;
        }

        Node* temp = head;
        Node* prev = nullptr;

        // Traverse the circular linked list
to find the node with the given value
```

```
        do {
            if (temp->data == value)
                break;

            prev = temp;
            temp = temp->next;
        } while (temp != head);

        // If node with given value is not
found
        if (temp == head && temp->data !=
value) {
            cout << "Element " << value << "
not found in the Circular Linked List." <<
endl;
            return;
        }

        // If the node to be deleted is the
head node
        if (temp == head) {
            // If head is the only node in
the circular linked list
            if (head->next == head) {
                delete head;
                head = nullptr;
            }
```

```
            else {
                prev = head;
                while (prev->next != head)
                    prev = prev->next;
                prev->next = head->next;
                delete head;
                head = prev->next;
            }
        }
        else {
            prev->next = temp->next;
            delete temp;
        }

        cout << "Element " << value << "
deleted from the Circular Linked List." <<
endl;
    }

    // Function to search for a value in the
circular linked list
    void search(int value) {
        if (head == nullptr) {
            cout << "Circular Linked List is
empty. Cannot search." << endl;
            return;
        }
```

```
        Node* temp = head;
        int position = 1;
        bool found = false;

        // Traverse the circular linked list
to search for the value
        do {
            if (temp->data == value) {
                cout << "Element " << value
<< " found at position " << position << "."
<< endl;
                found = true;
                break;
            }
            temp = temp->next;
            position++;
        } while (temp != head);

        if (!found)
            cout << "Element " << value << "
not found in the Circular Linked List." <<
endl;
    }

    // Function to get the count of nodes in
the circular linked list
```

```
    int getCount() {
        if (head == nullptr)
            return 0;

        Node* temp = head;
        int count = 0;

        // Traverse the circular linked list
and count nodes
        do {
            count++;
            temp = temp->next;
        } while (temp != head);

        return count;
    }

    // Function to delete the entire circular
linked list
    void deleteList() {
        if (head == nullptr) {
            cout << "Circular Linked List is
already empty." << endl;
            return;
        }

        Node* temp = head;
```

```
        Node* nextNode;

        // Traverse the circular linked list
and delete nodes
        do {
            nextNode = temp->next;
            delete temp;
            temp = nextNode;
        } while (temp != head);

        head = nullptr;
        cout << "Circular Linked List deleted
successfully." << endl;
    }

    ~CircularLinkedList() {
        deleteList();
    }
};

int main() {
    CircularLinkedList circularList;
    int choice, value;

    do {
        cout << "\n------ Menu ------" <<
endl;
```

```
        cout << "1. Insert" << endl;
        cout << "2. Delete" << endl;
        cout << "3. Search" << endl;
        cout << "4. Display" << endl;
        cout << "5. Get Count" << endl;
        cout << "6. Delete List" << endl;
        cout << "7. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter value to
insert: ";
                cin >> value;
                circularList.insert(value);
                break;
            case 2:
                if (circularList.getCount()
== 0) {
                    cout << "Circular Linked
List is empty. Cannot delete." << endl;
                    break;
                }
                cout << "Enter value to
delete: ";
                cin >> value;
```

```
circularList.deleteNode(value);
                break;
            case 3:
                if (circularList.getCount()
== 0) {
                    cout << "Circular Linked
List is empty. Cannot search." << endl;
                    break;
                }
                cout << "Enter value to
search: ";
                cin >> value;
                circularList.search(value);
                break;
            case 4:
                if (circularList.getCount()
== 0) {
                    cout << "Circular Linked
List is empty." << endl;
                    break;
                }
                cout << "Circular Linked
List: ";
                circularList.display();
                break;
            case 5:
```

```
                cout << "Number of nodes in
Circular Linked List: " <<
circularList.getCount() << endl;
                break;
            case 6:
                circularList.deleteList();
                break;
            case 7:
                cout << "Exiting..." << endl;
                break;
            default:
                cout << "Invalid choice.
Please enter a valid option." << endl;
                break;
        }
    } while (choice != 7);

    return 0;
}
```

Output: (screenshot)

```
cd "/Users/jeevan/Desktop/DSA/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/Users/jeevan/Desktop/DSA/"tempCodeRunnerFile
jeevan@Jeevans-MacBook-Air DSA % cd "/Users/jeevan/Desktop/DSA/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/Users/jeevan/Desktop/DSA/"tempCodeRunnerFile

------ Menu ------
1. Insert
2. Delete
3. Search
4. Display
5. Get Count
6. Delete List
7. Exit
Enter your choice: 1
Enter value to insert: 1

------ Menu ------
1. Insert
2. Delete
3. Search
4. Display
5. Get Count
6. Delete List
7. Exit
Enter your choice: 1
Enter value to insert: 2

------ Menu ------
1. Insert
2. Delete
3. Search
4. Display
5. Get Count
6. Delete List
7. Exit
Enter your choice: 1
Enter value to insert: 3

------ Menu ------
1. Insert
2. Delete
3. Search
4. Display
5. Get Count
6. Delete List
7. Exit
Enter your choice: 4
Circular Linked List: 1 2 3
```

Test Case: Any two (screenshot)

```
cd "/Users/jeevan/Desktop/DSA/" && g++ 8.cpp -o 8 && "/Users/jeevan/Desktop/DSA/"8
jeevan@Jeevans-MacBook-Air DSA % cd "/Users/jeevan/Desktop/DSA/" && g++ 8.cpp -o 8 && "/Users/jeevan/Desktop/DSA/"8

------ Menu ------
1. Insert
2. Delete
3. Search
4. Display
5. Get Count
6. Delete List
7. Exit
Enter your choice: 2
Circular Linked List is empty. Cannot delete.

------ Menu ------
1. Insert
2. Delete
3. Search
4. Display
5. Get Count
6. Delete List
7. Exit
Enter your choice: 1
Enter value to insert: 1

------ Menu ------
1. Insert
2. Delete
3. Search
4. Display
5. Get Count
6. Delete List
7. Exit
Enter your choice: 1
Enter value to insert: 3

------ Menu ------
1. Insert
2. Delete
3. Search
4. Display
5. Get Count
6. Delete List
7. Exit
Enter your choice: 2
Enter value to delete: 2
Element 2 not found in the Circular Linked List.

------ Menu ------
1. Insert
2. Delete
3. Search
4. Display
5. Get Count
6. Delete List
7. Exit
Enter your choice: 4
Circular Linked List: 1 3

------ Menu ------
1. Insert
2. Delete
3. Search
4. Display
5. Get Count
6. Delete List
7. Exit
```

```
cd "/Users/jeevan/Desktop/DSA/" && g++ 8.cpp -o 8 && "/Users/jeevan/Desktop/DSA/"8
jeevan@Jeevans-MacBook-Air DSA % cd "/Users/jeevan/Desktop/DSA/" && g++ 8.cpp -o 8 && "/Users/jeevan/Desktop/DSA/"8

------ Menu ------
1. Insert
2. Delete
3. Search
4. Display
5. Get Count
6. Delete List
7. Exit
Enter your choice: 2
Circular Linked List is empty. Cannot delete.

------ Menu ------
1. Insert
2. Delete
3. Search
4. Display
5. Get Count
6. Delete List
7. Exit
Enter your choice: 1
Enter value to insert: 1

------ Menu ------
1. Insert
2. Delete
3. Search
4. Display
5. Get Count
6. Delete List
7. Exit
Enter your choice: 1
Enter value to insert: 3

------ Menu ------
1. Insert
2. Delete
3. Search
4. Display
5. Get Count
6. Delete List
7. Exit
Enter your choice: 2
Enter value to delete: 2
Element 2 not found in the Circular Linked List.

------ Menu ------
1. Insert
2. Delete
3. Search
4. Display
5. Get Count
6. Delete List
7. Exit
Enter your choice: 4
Circular Linked List: 1 3

------ Menu ------
1. Insert
2. Delete
3. Search
4. Display
5. Get Count
6. Delete List
7. Exit
```

Conclusion: In conclusion, the program demonstrates the functionality and versatility of circular linked lists, showcasing their ability to efficiently manage dynamic data structures. Users can easily manipulate the circular linked list by inserting, deleting, searching for elements, and obtaining information about the list's size. The menu-driven interface enhances user experience, providing a seamless interaction with the circular linked list operations. By leveraging pointer manipulation and dynamic memory allocation, the program offers a practical implementation of circular linked lists, fostering a deeper understanding of fundamental data structure concepts and programming techniques.

Name of Student: JEEVAN NAIDU

Roll Number: 150096723015

Experiment No: 09

Title: Implement Stack ADT using Linked List

Theory: The program implements a stack data structure in C++ using a linked list. A stack is a linear data structure that follows the Last In, First Out (LIFO) principle, where elements are inserted and removed from the same end, known as the top. The Stack class utilizes a linked list representation, where each element is stored in a node containing an integer data field and a pointer to the next node. The operations supported by the stack include push (inserting an element onto the stack), pop (removing the top element from the stack), peek (viewing the top element without removing it), display (printing all elements of the stack), and deleteStack (deleting the entire stack).

Code:

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

class Stack {
private:
    struct Node {
        int data;
        Node* next;
    }* top;

public:
    Stack() {
        top = nullptr;
```

```
    }

    // Function to push an element onto the stack
    void push(int value) {
        Node* newNode = new Node;
        newNode->data = value;
        newNode->next = top;
        top = newNode;
        cout << "Element " << value << " pushed onto the stack." << endl;
    }

    // Function to pop an element from the stack
    int pop() {
        if (isEmpty()) {
            cout << "Stack Underflow! Cannot pop from an empty stack." << endl;
            return -1; // or throw an exception
        }
        int value = top->data;
        Node* temp = top;
        top = top->next;
        delete temp;
        return value;
```

```
    }

    // Function to peek at the top element of
the stack
    int peek() {
        if (isEmpty()) {
            cout << "Stack is empty." <<
endl;
            return -1; // or throw an
exception
        }
        return top->data;
    }

    // Function to check if the stack is
empty
    bool isEmpty() {
        return top == nullptr;
    }

    // Function to display the elements of
the stack
    void display() {
        if (isEmpty()) {
            cout << "Stack is empty." <<
endl;
            return;
```

```
        }
        cout << "Stack elements: ";
        Node* temp = top;
        while (temp != nullptr) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }

    // Function to delete the entire stack
    void deleteStack() {
        Node* temp;
        while (top != nullptr) {
            temp = top;
            top = top->next;
            delete temp;
        }
        cout << "Stack deleted successfully."
<< endl;
    }

    ~Stack() {
        deleteStack();
    }
};
```

```
int main() {
    Stack stack;
    int choice, value;

    do {
        cout << "\n------ Menu ------" <<
endl;
        cout << "1. Push" << endl;
        cout << "2. Pop" << endl;
        cout << "3. Peek" << endl;
        cout << "4. Display" << endl;
        cout << "5. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter value to push:
";
                cin >> value;
                stack.push(value);
                break;
            case 2:
                value = stack.pop();
                if (value != -1)
                    cout << "Popped element:
" << value << endl;
```

```
                break;
            case 3:
                value = stack.peek();
                if (value != -1)
                    cout << "Top element: "
<< value << endl;
                break;
            case 4:
                stack.display();
                break;
            case 5:
                cout << "Exiting..." << endl;
                break;
            default:
                cout << "Invalid choice.
Please enter a valid option." << endl;
                break;
        }
    } while (choice != 5);

    return 0;
}
```

Output: (screenshot)

```
cd "/Users/jeevan/Desktop/DSA/" && g++ 9.cpp -o 9 && "/Users/jeevan/Desktop/DSA/"9
jeevan@Jeevans-MacBook-Air DSA % cd "/Users/jeevan/Desktop/DSA/" && g++ 9.cpp -o 9 && "/Users/jeevan/Desktop/DSA/"9

------ Menu ------
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to push: 1
Element 1 pushed onto the stack.

------ Menu ------
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to push: 3
Element 3 pushed onto the stack.

------ Menu ------
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to push: 5
Element 5 pushed onto the stack.

------ Menu ------
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 4
Stack elements: 5 3 1
```

Test Case: Any two (screenshot)

```
cd "/Users/jeevan/Desktop/DSA/" && g++ 9.cpp -o 9 && "/Users/jeevan/Desktop/DSA/"9
jeevan@Jeevans-MacBook-Air DSA % cd "/Users/jeevan/Desktop/DSA/" && g++ 9.cpp -o 9 && "/Users/jeevan/Desktop/DSA/"9

------ Menu ------
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to push: 1
Element 1 pushed onto the stack.

------ Menu ------
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to push: 3
Element 3 pushed onto the stack.

------ Menu ------
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to push: 5
Element 5 pushed onto the stack.

------ Menu ------
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 4
Stack elements: 5 3 1
```

```
cd "/Users/jeevan/Desktop/DSA/" && g++ 9.cpp -o 9 && "/Users/jeevan/Desktop/DSA/"9
jeevan@Jeevans-MacBook-Air DSA % cd "/Users/jeevan/Desktop/DSA/" && g++ 9.cpp -o 9 && "/Users/jeevan/Desktop/DSA/"9

------ Menu ------
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to push: 1
Element 1 pushed onto the stack.

------ Menu ------
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 2
Popped element: 1

------ Menu ------
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 2
Stack Underflow! Cannot pop from an empty stack.

------ Menu ------
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter your choice: 4
Stack is empty.
```

Conclusion: In conclusion, the program demonstrates the implementation and functionality of a stack data structure using a linked list in C++. Users can interact with the stack through a menu-driven interface, allowing them to push elements onto the stack, pop elements from the stack, view the top element, display all elements, and exit the program. The use of linked lists enables dynamic memory allocation and efficient management of stack operations, making it suitable for scenarios where the size of the stack may vary dynamically. The program provides a practical illustration of stack operations and serves as a foundation for understanding more complex data structures and algorithms that utilize stacks.

Name of Student: **JEEVAN NAIDU**

Roll Number: **150096723015**

Experiment No: 10

Title: Implement Linear Queue ADT using Linked List

Theory:

The program demonstrates the implementation of a queue data structure in C++. A queue is a linear data structure that follows the First In, First Out (FIFO) principle, where elements are inserted at the rear and removed from the front. The Queue class is defined using a linked list representation, where each element is stored in a node containing an integer data field and a pointer to the next node. The class provides methods for enqueueing (inserting an element into the queue), dequeuing (removing an element from the front of the queue), peeking (viewing the front element without removing it), displaying all elements of the queue, and checking if the queue is empty.

Code:

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

class Queue {
private:
    struct Node {
        int data;
```

```
        Node* next;
    }* front, *rear;

public:
    Queue() {
        front = rear = nullptr;
    }

    // Function to check if the queue is
empty
    bool isEmpty() {
        return front == nullptr;
    }

    // Function to enqueue an element into
the queue
    void enqueue(int value) {
        Node* newNode = new Node;
        newNode->data = value;
        newNode->next = nullptr;
        if (isEmpty()) {
            front = rear = newNode;
        } else {
            rear->next = newNode;
            rear = newNode;
        }
```

```
        cout << "Element " << value << "
enqueued into the queue." << endl;
    }

    // Function to dequeue an element from
the queue
    int dequeue() {
        if (isEmpty()) {
            cout << "Queue Underflow! Cannot
dequeue from an empty queue." << endl;
            return -1; // or throw an
exception
        }
        int value = front->data;
        Node* temp = front;
        if (front == rear) {
            front = rear = nullptr;
        } else {
            front = front->next;
        }
        delete temp;
        return value;
    }

    // Function to peek at the front element
of the queue
    int peek() {
```

```
        if (isEmpty()) {
            cout << "Queue is empty." <<
endl;
            return -1; // or throw an
exception
        }
        return front->data;
    }

    // Function to display the elements of
the queue
    void display() {
        if (isEmpty()) {
            cout << "Queue is empty." <<
endl;
            return;
        }
        cout << "Queue elements: ";
        Node* temp = front;
        while (temp != nullptr) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }

    ~Queue() {
```

```
        // Delete all remaining nodes
        Node* temp;
        while (front != nullptr) {
            temp = front;
            front = front->next;
            delete temp;
        }
    }
};

int main() {
    Queue queue;
    int choice, value;

    do {
        cout << "\n------ Menu ------" <<
endl;
        cout << "1. Enqueue" << endl;
        cout << "2. Dequeue" << endl;
        cout << "3. Peek" << endl;
        cout << "4. Display" << endl;
        cout << "5. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
```

```
                cout << "Enter value to
enqueue: ";
                cin >> value;
                queue.enqueue(value);
                break;
            case 2:
                value = queue.dequeue();
                if (value != -1)
                    cout << "Dequeued
element: " << value << endl;
                break;
            case 3:
                value = queue.peek();
                if (value != -1)
                    cout << "Front element: "
<< value << endl;
                break;
            case 4:
                queue.display();
                break;
            case 5:
                cout << "Exiting..." << endl;
                break;
            default:
                cout << "Invalid choice.
Please enter a valid option." << endl;
                break;
```

```
        }
    } while (choice != 5);

    return 0;
}
```

Output: (screenshot)

```
cd "/Users/jeevan/Desktop/DSA/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/Users/jeevan/Desktop/DSA/"tempCodeRunnerFile
jeevan@Jeevans-MacBook-Air DSA % cd "/Users/jeevan/Desktop/DSA/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/Users/jeevan/Desktop/DSA/"tempCodeRunnerFile

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue: 1
Element 1 enqueued into the queue.

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 2
Dequeued element: 1

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 2
Queue Underflow! Cannot dequeue from an empty queue.

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 4
Queue is empty.
```

Test Case: Any two (screenshot)


```
cd "/Users/jeevan/Desktop/DSA/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/Users/jeevan/Desktop/DSA/"tempCodeRunnerFile
jeevan@Jeevans-MacBook-Air DSA % cd "/Users/jeevan/Desktop/DSA/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/Users/jeevan/Desktop/DSA/"tempCodeRunnerFile

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue: 1
Element 1 enqueued into the queue.

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 2
Dequeued element: 1

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 2
Queue Underflow! Cannot dequeue from an empty queue.

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 4
Queue is empty.
```

```
cd "/Users/jeevan/Desktop/DSA/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/Users/jeevan/Desktop/DSA/"tempCodeRunnerFile
jeevan@Jeevans-MacBook-Air DSA % cd "/Users/jeevan/Desktop/DSA/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/Users/jeevan/Desktop/DSA/"tempCodeRunnerFile

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue: 1
Element 1 enqueued into the queue.

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue: 2
Element 2 enqueued into the queue.

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue: 3
Element 3 enqueued into the queue.

------ Menu ------
1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 4
Queue elements: 1 2 3
```

Conclusion: In conclusion, the program offers a practical implementation of a queue data structure using a linked list in C++. Users can interact with the queue through a menu-driven interface, allowing them to enqueue elements, dequeue elements, peek at the front element, display all elements, and exit the program. The use of linked lists enables dynamic memory allocation and efficient management of queue operations, making it suitable for scenarios where the size of the queue may vary dynamically. The program serves as a fundamental example of queue operations and lays the groundwork for understanding more complex data structures and algorithms that utilize queues.

Name of Student: **JEEVAN NAIDU**

Roll Number: **150096723015**

Experiment No: 11

Title: Implement Binary Search Tree ADT using Linked List.

Theory:

A Binary Search Tree (BST) is a binary tree data structure in which each node has at most two children, referred to as the left child and the right child. The key property of a BST is that the value of each node in the left subtree is less than the value of the node, and the value of each node in the right subtree is greater than the value of the node. This property allows for efficient searching, insertion, and deletion operations.

CODE:

```
// Binary Search Tree using Linked List
#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;
    }
    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 << " ";
    }
};

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:

```
cd "/Users/jeevan/Desktop/DSA/" && g++ tempCodeRunnerFile.CPP -o tempCodeRunnerFile && "/Users/jeevan/Desktop/DSA/"tempCodeRunnerFile
jeevan@Jeevans-MacBook-Air DSA % cd "/Users/jeevan/Desktop/DSA/" && g++ tempCodeRunnerFile.CPP -o tempCodeRunnerFile && "/Users/jeevan/Desktop/DSA/"tempCodeRunnerFile
Enter number of nodes: 5
Enter data: 1
Enter data: 2
Enter data: 3
Enter data: 4
Enter data: 5
Inorder: 1 2 3 4 5
Preorder: 1 2 3 4 5
Postorder: 5 4 3 2 1
jeevan@Jeevans-MacBook-Air DSA %
```

Test Case: Any two (screenshot)

```
cd "/Users/jeevan/Desktop/DSA/" && g++ tempCodeRunnerFile.CPP -o tempCodeRunnerFile && "/Users/jeevan/Desktop/DSA/"tempCodeRunnerFile
jeevan@Jeevans-MacBook-Air DSA % cd "/Users/jeevan/Desktop/DSA/" && g++ tempCodeRunnerFile.CPP -o tempCodeRunnerFile && "/Users/jeevan/Desktop/DSA/"tempCodeRunnerFile
Enter number of nodes: 4
Enter data: 3
Enter data: 6
Enter data: 2
Enter data: 8
Inorder: 2 3 6 8
Preorder: 3 2 6 8
Postorder: 2 8 6 3
jeevan@Jeevans-MacBook-Air DSA % cd "/Users/jeevan/Desktop/DSA/" && g++ tempCodeRunnerFile.CPP -o tempCodeRunnerFile && "/Users/jeevan/Desktop/DSA/"tempCodeRunnerFile
Enter number of nodes: 7
Enter data: 30
Enter data: 45
Enter data: 23
Enter data: 14
Enter data: 25
Enter data: 64
Enter data: 53
Inorder: 14 23 25 30 45 53 64
Preorder: 30 23 14 25 45 64 53
Postorder: 14 25 23 53 64 45 30
jeevan@Jeevans-MacBook-Air DSA %
```

CONCLUSION: The program demonstrates the implementation of a Binary Search Tree using linked list nodes. It allows users to input the number of nodes they want to insert into the BST and then prompts for the data value for each node. After inserting the nodes, the program displays the BST in three different traversal orders: inorder, preorder, and postorder.

- **Inorder traversal displays the nodes in sorted order.**
- **Preorder traversal displays the root node first, followed by the left subtree and then the right subtree.**
- **Postorder traversal displays the left subtree first, followed by the right subtree, and then the root node.**

Binary Search Trees are widely used in various applications such as database indexing, searching, and sorting algorithms. They provide efficient search and retrieval operations when the data is stored in a sorted manner.

Name of Student: JEEVAN NAIDU

Roll Number: 150096723015

Experiment No: 12

Title: Implement Graph Traversal techniques:

a) Depth First Search b) Breadth First Search

Theory: The program demonstrates two fundamental graph traversal algorithms: Depth First Search (DFS) and Breadth First Search (BFS). Graph traversal is a process of visiting all the vertices or nodes of a graph in a systematic manner. DFS and BFS are two popular methods for traversing graphs and are used in various applications such as path finding, cycle detection, and connectivity analysis.

Code: #include <iostream>

```
#include <queue>

using namespace std;

const int MAXN = 100;

// Depth First Search (DFS)
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);
        }
    }
}

// Breadth First Search (BFS)
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)
        {
```

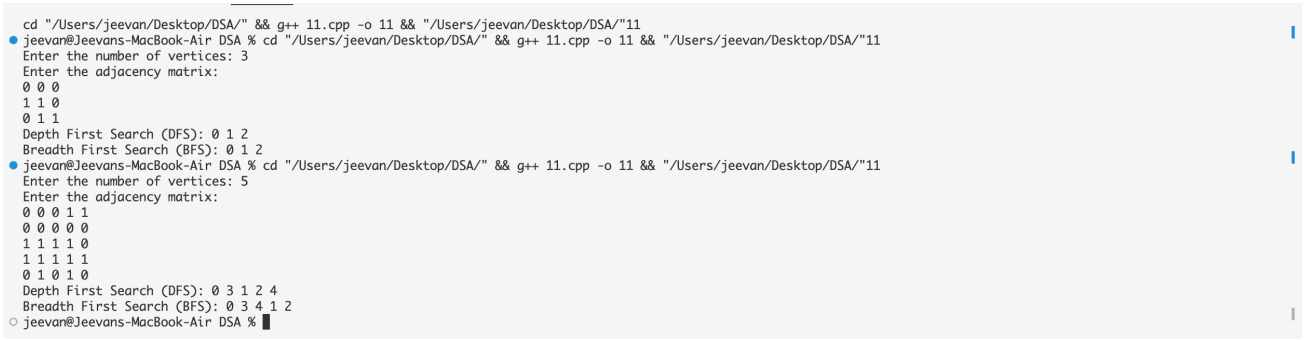
```
            if (graph[node][i] == 1 && !
visited[i])
            {
                q.push(i);
                visited[i] = true;
            }
        }
    }
}

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;

    // Resetting visited array for BFS
    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: In summary, the program allows users to input an adjacency matrix representing a graph and then performs DFS and BFS traversal on the graph. DFS explores as far as possible along each branch before backtracking, while BFS explores neighbors of a node before moving on to the next level. The program utilizes recursion for DFS and a queue for BFS to efficiently traverse the graph and print the order of visited nodes. These algorithms provide fundamental tools for understanding and analyzing graph structures and are essential in various graph-related problems and applications.

Name of Student: JEEVAN NAIDU

Roll Number: 150096723015

Experiment No: 13

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

Theory: Binary search is an efficient algorithm for finding a target value within a sorted array. It works by repeatedly dividing the search interval in half until the target value is found or the interval is empty. Binary search compares the target value to the middle element of the array and eliminates half of the remaining elements based on the comparison result. The time complexity of binary search is O(log n), where n is the number of elements in the array.

Code:

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

// Function to perform binary search on a sorted array
int binarySearch(int arr[], int n, int target) {
    int left = 0, right = n - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (arr[mid] == target) {
```

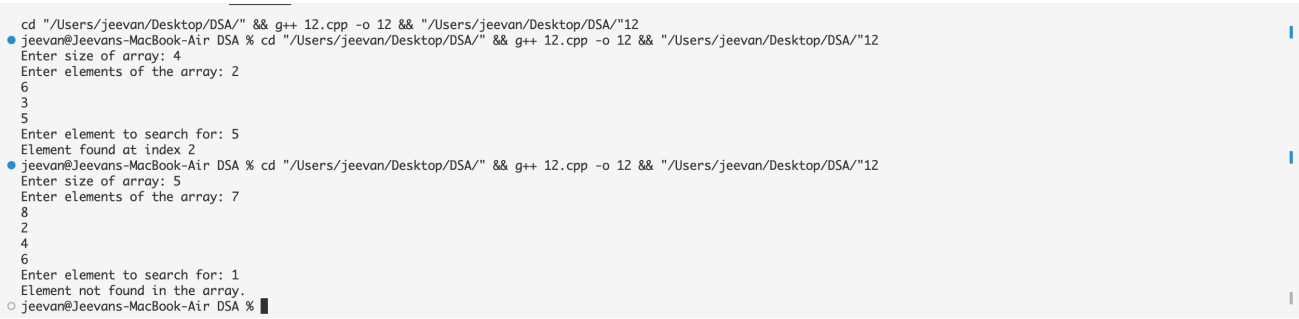
```
            return mid; // Element found,
return its index
        } else if (arr[mid] < target) {
            left = mid + 1; // Search in the
right half
        } else {
            right = mid - 1; // Search in the
left half
        }
    }

    return -1; // Element not found
}

// Function to perform bubble sort on an
array
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                // Swap arr[j] and arr[j + 1]
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
```

```
}

int main() {
    int n;
    cout << "Enter size of array: ";
    cin >> n;

    int arr[n];
    cout << "Enter elements of the array: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    // Sorting the array using bubble sort
    bubbleSort(arr, n);

    int target;
    cout << "Enter element to search for: ";
    cin >> target;

    int index = binarySearch(arr, n, target);

    if (index != -1) {
        cout << "Element found at index " <<
index << endl;
    } else {
```

```
        cout << "Element not found in the
array." << endl;
    }

    return 0;
}
```

Output: (screenshot)

Test Case: Any two (screenshot)



Conclusion: The provided C++ program first prompts the user to enter the size of the array and then accepts the elements of the array. It sorts the array using the bubble sort algorithm and then prompts the user to enter the target element to search for. The binary search algorithm is then applied to find the target element in the sorted array. If the element is found, the program outputs its index; otherwise, it notifies the user that the element was not found in the array.

Name of Student: JEEVAN NAIDU

Roll Number: 150096723015

Experiment No: 14

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

Theory:

- **Bubble Sort Algorithm:**

- **Bubble sort is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted. Bubble sort is named for the way smaller elements "bubble" to the top of the list with each pass. Despite its simplicity, bubble sort is not efficient for large lists and is mainly used for educational purposes.**

- **Ascending Order Sorting:**

- **Sorting in ascending order means arranging elements from the smallest to the largest. In the context of bubble sort, this involves iterating through the array and swapping adjacent elements if they are in the wrong order (i.e., if the current element is greater than the next element).**

- **Descending Order Sorting:**

- **Sorting in descending order means arranging elements from the largest to the smallest. In the context of bubble sort, this involves iterating through the array and swapping adjacent elements if they are in the wrong order (i.e., if the current element is less than the next element).**

Code:

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

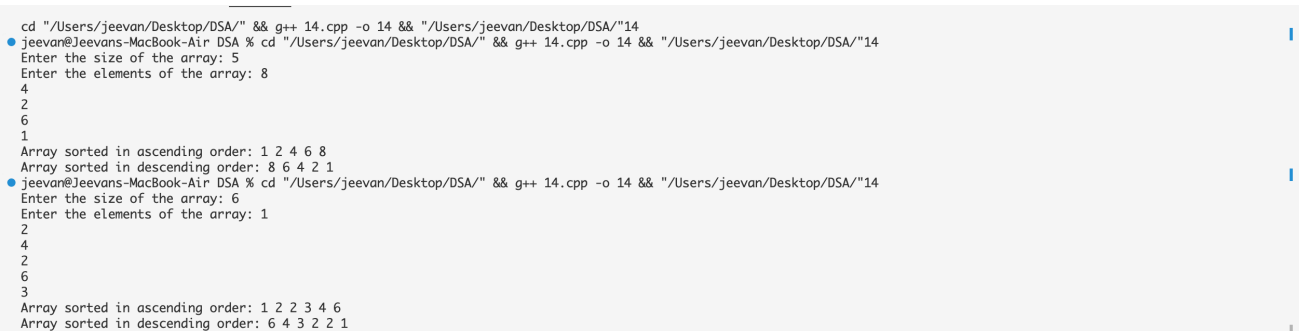
```
// Function to perform bubble sort in
ascending order
void bubbleSortAscending(int arr[], int n) {
    for (int i = 0; i < n - 1; ++i) {
        for (int j = 0; j < n - i - 1; ++j) {
            if (arr[j] > arr[j + 1]) {
                // Swap arr[j] and arr[j + 1]
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

// Function to perform bubble sort in
descending order
void bubbleSortDescending(int arr[], int n) {
    for (int i = 0; i < n - 1; ++i) {
        for (int j = 0; j < n - i - 1; ++j) {
            if (arr[j] < arr[j + 1]) {
                // Swap arr[j] and arr[j + 1]
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
```

```
            }
        }
    }

    // Function to print the elements of an array
    void printArray(int arr[], int n) {
        for (int i = 0; i < n; ++i) {
            cout << arr[i] << " ";
        }
        cout << endl;
    }

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

        int arr[n];
        cout << "Enter the elements of the array:
    ";
        for (int i = 0; i < n; ++i) {
            cin >> arr[i];
        }

        // Sorting in ascending order
        bubbleSortAscending(arr, n);
```

```
    cout << "Array sorted in ascending order:
";
    printArray(arr, n);

    // Sorting in descending order
    bubbleSortDescending(arr, n);
    cout << "Array sorted in descending
order: ";
    printArray(arr, n);

    return 0;
}
```

Output: (screenshot)

Test Case: Any two (screenshot)



Conclusion: The provided C++ program demonstrates the implementation of bubble sort for sorting an array in both ascending and descending orders. It prompts the user to enter the size of the array and the elements of the array, sorts the array using bubble sort in ascending order, and then prints the sorted array. After that, it sorts the same array in descending order using bubble sort and prints the sorted array again.

This program serves as a practical example of how bubble sort can be used to sort arrays in both ascending and descending orders, providing a clear illustration of the algorithm's operation and its application in sorting algorithms.