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

DATA STRUCTURES & ALGORITHMS PROGRAMMING LAB

Prepared by:

Name of Student : Riya Singh

Roll No: 26

Batch: 2023-27

Dept. of CSE

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

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

CERTIFICATE

This is to certify that Mr. / Ms. ____ Roll No. _____________ Semester _____ of B.Tech Computer Science & Engineering, ITM Skills University, Kharghar, Navi Mumbai , has completed the term work satisfactorily in subject ____________________________________ for the academic year 20___ - 20___ as prescribed in the curriculum.

Place: _____________

Date: ______________

Subject I/C **HOD**

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

Name of Student:Riya Singh

Roll Number: 26

Experiment No: 1

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

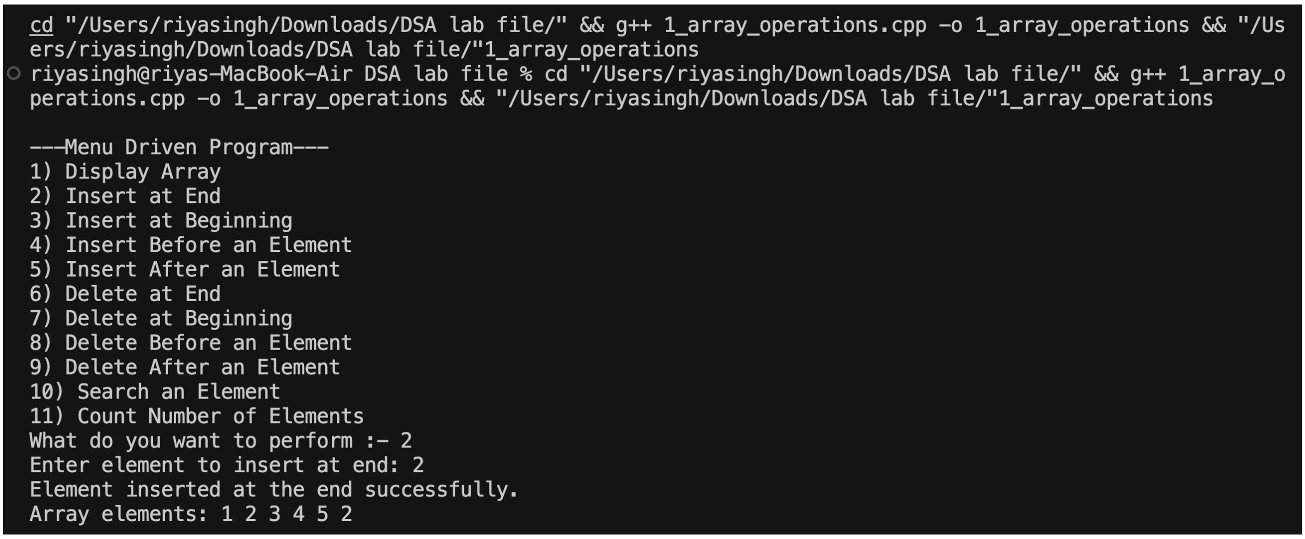
Theory: Array is a collection of elements of similar data types and has a fixed size. We can access an element of the array through it's index. Indexing starts from 0 till n-1(where n=size of array). An element can be inserted in the array by shifting all the elements of the array to the right and making space for the element. Similarly, to delete an element, we need to shift all the elements from the right of the deleted element to the left side in order to overwrite the deleted element. In order to search for an element, we need to traverse through the array and print the appropriate message if 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_SIZE = 100;

class array_operations {
private:
    int arr[MAX_SIZE];
    int size;

public:
    array_operations() {
        size = 0;
    }
    array_operations(int initial_array[], int initial_size) {
        if (initial_size <= MAX_SIZE) {
            for (int i = 0; i < initial_size; ++i) {
                arr[i] = initial_array[i];
            }
            size = initial_size;
        } else {
            cerr << "Initial array size exceeds maximum size." <<
endl;
```

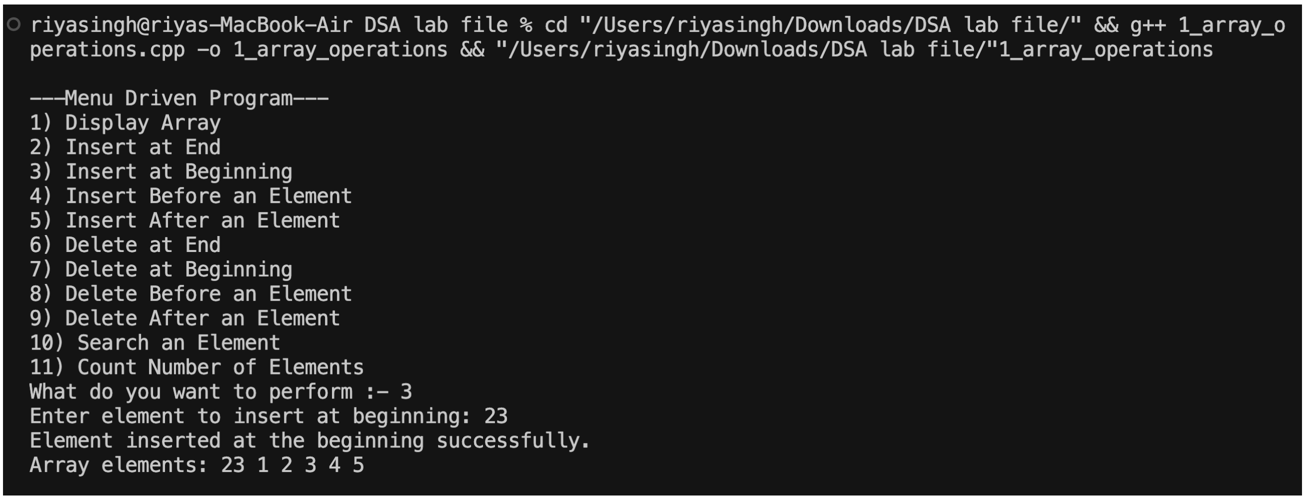
```
        }
    }
    void display_array() {
        if (size == 0) {
            cout << "Array is empty." << endl;
            return;
        }
        cout << "Array elements: ";
        for (int i = 0; i < size; i++) {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
    void insert_at_end(int element) {
        if (size < MAX_SIZE) {
            arr[size++] = element;
            cout << "Element inserted at the end successfully." <<
endl;
        } else {
            cout << "Array is full. Cannot insert element." <<
endl;
        }
    }
    void insert_at_beginning(int element) {
        if (size < MAX_SIZE) {
            for (int i = size; i > 0; i--) {
                arr[i] = arr[i - 1];
            }
            arr[0] = element;
            size++;
            cout << "Element inserted at the beginning 
successfully." << endl;
        } else {
            cout << "Array is full. Cannot insert element." <<
endl;
        }
    }
    void insert_before_element(int element, int target) {
        int index = -1;
        for (int i = 0; i < size; i++) {
            if (arr[i] == target) {
                index = i;
                break;
            }
        }
        if (index != -1) {
            if (size < MAX_SIZE) {
                for (int i = size; i > index; i--) {
                    arr[i] = arr[i - 1];
                }
                arr[index] = element;
                size++;
                cout << "Element inserted before " << target << " successfully." << endl;
            } else {
                cout << "Array is full. Cannot insert element." << endl;
            }
        } else {
            cout << "Element not found in the array." << endl;
        }
    }
    void insert_after_element(int element, int target) {
        int index = -1;
        for (int i = 0; i < size; i++) {
            if (arr[i] == target) {
                index = i;
                break;
            }
        }
        if (index != -1) {
            if (size < MAX_SIZE) {
                for (int i = size; i > index + 1; i--) {
                    arr[i] = arr[i - 1];
                }
                arr[index + 1] = element;
                size++;
                cout << "Element inserted after " << target << " successfully." << endl;
            } else {
                cout << "Array is full. Cannot insert element." << endl;
            }
        } else {
            cout << "Element not found in the array." << endl;
        }
    }
    void delete_at_end() {
        if (size > 0) {
            size--;
            cout << "Element deleted from the end successfully." << endl;
        } else {
            cout << "Array is empty. Cannot delete element." << endl;
        }
    }
    void delete_at_beginning() {
        if (size > 0) {
            for (int i = 0; i < size - 1; i++) {
                arr[i] = arr[i + 1];
            }
            size--;
            cout << "Element deleted from the beginning 
successfully." << endl;
        } else {
            cout << "Array is empty. Cannot delete element." << 
endl;
        }
    }
    void delete_before_element(int target) {
        int index = -1;
        for (int i = 0; i < size; i++) {
            if (arr[i] == target) {
                index = i;
                break;
            }
        }
        if (index != -1 && index > 0) {
            for (int i = index - 1; i < size - 1; i++) {
                arr[i] = arr[i + 1];
            }
            size--;
            cout << "Element deleted before " << target << " 
successfully." << endl;
        } else {
            cout << "Element not found in the array or no element 
before it." << endl;
        }
    }
    void delete_after_element(int target) {
        int index = -1;
        for (int i = 0; i < size; i++) {
            if (arr[i] == target) {
                index = i;
                break;
            }
        }
        if (index != -1 && index < size - 1) {
            for (int i = index + 1; i < size - 1; i++) {
                arr[i] = arr[i + 1];
            }
            size--;
            cout << "Element deleted after " << target << " 
successfully." << endl;
        } else {
            cout << "Element not found in the array or no element 
after it." << endl;
        }
    }
    void search_element(int element) {
        bool found = false;
        for (int i = 0; i < size; i++) {
            if (arr[i] == element) {
                found = true;
                break;
            }
        }
        if (found) {
            cout << "Element " << element << " found in the array." 
<< endl;
        } else {
            cout << "Element " << element << " not found in the 
array." << endl;
        }
    }
    void count_elements() {
        cout << "Number of elements in the array: " << size << 
endl;
    }
};

int main() {
    int initial_array[] = {1, 2, 3, 4, 5};
    int initial_size = sizeof(initial_array) / 
sizeof(initial_array[0]);

    array_operations array(initial_array, initial_size);
    int choice, element, target;
    do {
        cout << "\n---Menu Driven Program---" << endl;
        cout << "1) Display Array" << endl;
        cout << "2) Insert at End" << endl;
        cout << "3) Insert at Beginning" << endl;
        cout << "4) Insert Before an Element" << endl;
        cout << "5) Insert After an Element" << endl;
        cout << "6) Delete at End" << endl;
        cout << "7) Delete at Beginning" << endl;
        cout << "8) Delete Before an Element" << endl;
        cout << "9) Delete After an Element" << endl;
        cout << "10) Search an Element" << endl;
        cout << "11) Count Number of Elements" << endl;
        cout << "What do you want to perform :- ";
        cin >> choice;
        switch (choice) {
            case 1:
                array.display_array();
                break;
            case 2:
                cout << "Enter element to insert at end: ";
                cin >> element;
                array.insert_at_end(element);
                array.display_array();
                break;
            case 3:
                cout << "Enter element to insert at beginning: ";
                cin >> element;
                array.insert_at_beginning(element);
                array.display_array();
                break;
            case 4:
                cout << "Enter element to insert: ";
                cin >> element;
                cout << "Enter element before which to insert: ";
                cin >> target;
                array.insert_before_element(element, target);
                array.display_array();
                break;
            case 5:
                cout << "Enter element to insert: ";
                cin >> element;
                cout << "Enter element after which to insert: ";
                cin >> target;
                array.insert_after_element(element, target);
                array.display_array();
                break;
            case 6:
                array.delete_at_end();
                array.display_array();
                break;
            case 7:
                array.delete_at_beginning();
                array.display_array();
                break;
            case 8:
                cout << "Enter element before which to delete: ";
                cin >> target;
                array.delete_before_element(target);
                array.display_array();
                break;
            case 9:
                cout << "Enter element after which to delete: ";
                cin >> target;
                array.delete_after_element(target);
                array.display_array();
                break;
            case 10:
                cout << "Enter element to search: ";
                cin >> element;
                array.search_element(element);
                array.display_array();
                break;
            case 11:
                array.count_elements();
                break;
            default:
                cout << "Invalid choice. Please try again." <<
endl;
        }
    } while (true);
    return 0;
}
```

Output: (screenshot)

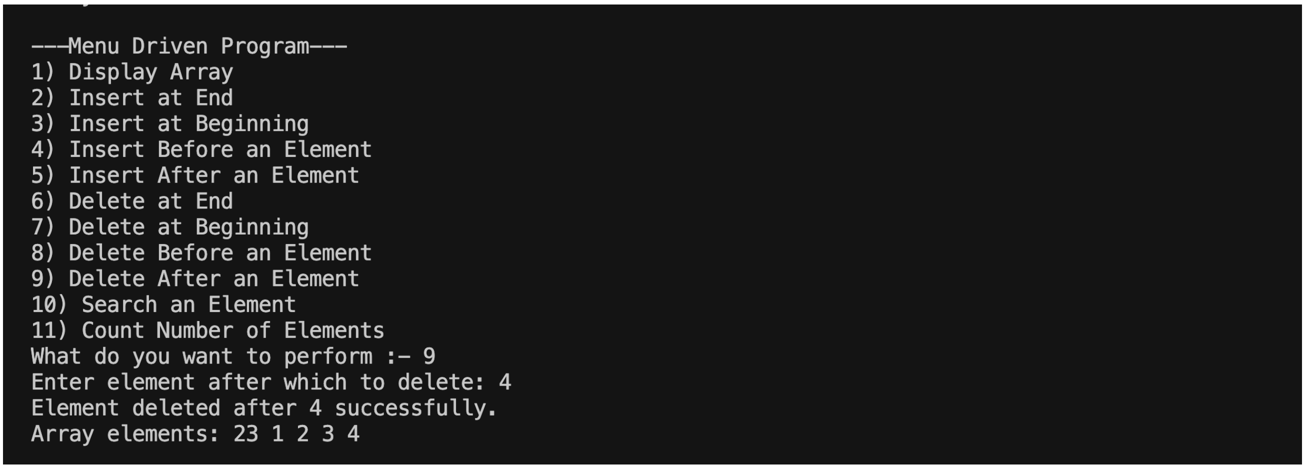


```
cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ 1_array_operations.cpp -o 1_array_operations && "/Us
ers/riyasingh/Downloads/DSA lab file/"1_array_operations
riyasingh@riyas-MacBook-Air DSA lab file % cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ 1_array_o
perations.cpp -o 1_array_operations && "/Users/riyasingh/Downloads/DSA lab file/"1_array_operations

---Menu Driven Program---
1) Display Array
2) Insert at End
3) Insert at Beginning
4) Insert Before an Element
5) Insert After an Element
6) Delete at End
7) Delete at Beginning
8) Delete Before an Element
9) Delete After an Element
10) Search an Element
11) Count Number of Elements
What do you want to perform :- 2
Enter element to insert at end: 2
Element inserted at the end successfully.
Array elements: 1 2 3 4 5 2
```

Test Case: Any two (screenshot)



```
○ riyasingh@riyas-MacBook-Air DSA lab file % cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ 1_array_o
  perations.cpp -o 1_array_operations && "/Users/riyasingh/Downloads/DSA lab file/"1_array_operations

  ---Menu Driven Program---
  1) Display Array
  2) Insert at End
  3) Insert at Beginning
  4) Insert Before an Element
  5) Insert After an Element
  6) Delete at End
  7) Delete at Beginning
  8) Delete Before an Element
  9) Delete After an Element
  10) Search an Element
  11) Count Number of Elements
  What do you want to perform :- 3
  Enter element to insert at beginning: 23
  Element inserted at the beginning successfully.
  Array elements: 23 1 2 3 4 5
```



```
  ---Menu Driven Program---
  1) Display Array
  2) Insert at End
  3) Insert at Beginning
  4) Insert Before an Element
  5) Insert After an Element
  6) Delete at End
  7) Delete at Beginning
  8) Delete Before an Element
  9) Delete After an Element
  10) Search an Element
  11) Count Number of Elements
  What do you want to perform :- 9
  Enter element after which to delete: 4
  Element deleted after 4 successfully.
  Array elements: 23 1 2 3 4
```

Conclusion: Therefore, using switch cases, we can perform multiple operations like insertion, deletion, and searching for an element in an array through traversal using index.

Name of Student: Riya Singh

Roll Number: 26

Experiment No: 2

Title: Implement Stack ADT using Array.

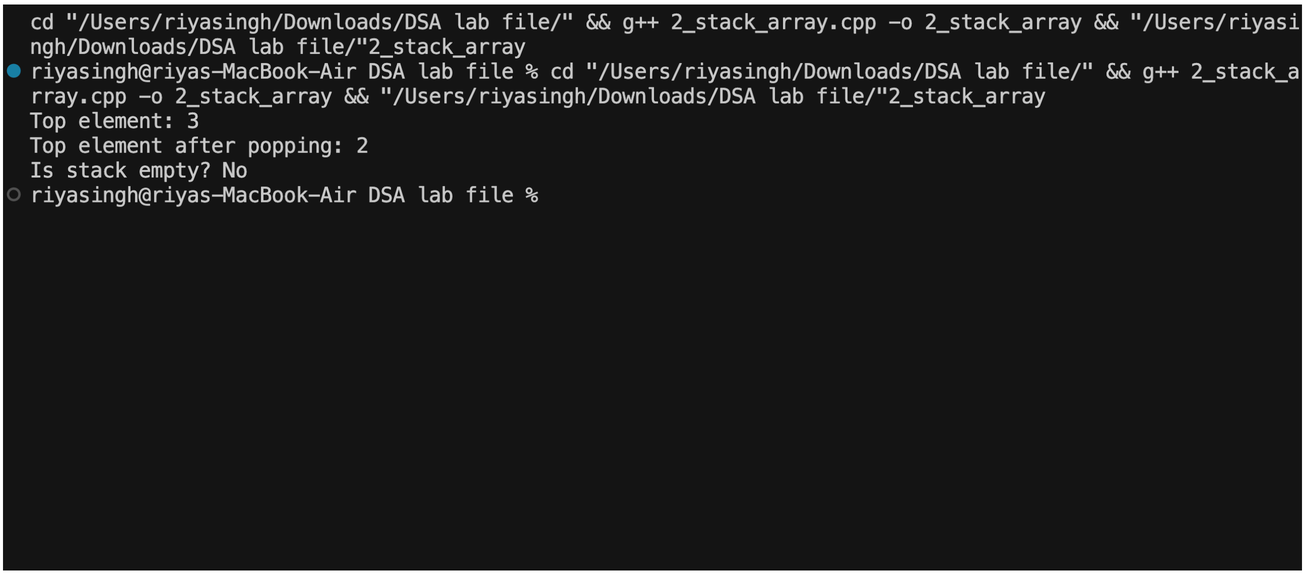
Theory: Array is a collection of elements of similar data types and has a fixed size. We can access an element of the array through it's index. Indexing starts from 0 till n-1(where n=size of array).

Stack is an Abstract Data Type which can be implemented using Linked List or Array. It consists of a variable named Top which points to the topmost element of the stack. Stack follows LIFO principle(Last In, First Out) which means that the element which is inserted last will be deleted first. There are three operations in Stack: Push- insertion from top, Pop- deletion from top, Peek- returning the topmost element from the stack.

Code:

```cpp
//Implement Stack ADT using array.
#include <iostream>
const int MAX_SIZE = 100;
class Stack {
private:
    int arr[MAX_SIZE];
    int top;
public:
    Stack() {
        top = -1;
    }
    void push(int value) {
        if (top == MAX_SIZE - 1) {
            std::cout << "Stack Overflow\n";
            return;
        }
        arr[++top] = value;
    }
    void pop() {
        if (top == -1) {
            std::cout << "Stack Underflow\n";
            return;
        }
        top--;
    }
    int peek() {
        if (top == -1) {
            std::cout << "Stack is empty\n";
            return -1;
        }
        return arr[top];
    }
    bool isEmpty() {
        return top == -1;
    }
};
int main() {
    Stack stack;

    stack.push(1);
    stack.push(2);
    stack.push(3);

    std::cout << "Top element: " << stack.peek() << std::endl;

    stack.pop();
    std::cout << "Top element after popping: " << stack.peek() <<
std::endl;

    std::cout << "Is stack empty? " << (stack.isEmpty() ? "Yes" :
"No") << std::endl;

    return 0;
}
```

Output: (screenshot)



```
  cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ 2_stack_array.cpp -o 2_stack_array && "/Users/riyasi
  ngh/Downloads/DSA lab file/"2_stack_array
● riyasingh@riyas-MacBook-Air DSA lab file % cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ 2_stack_a
  rray.cpp -o 2_stack_array && "/Users/riyasingh/Downloads/DSA lab file/"2_stack_array
  Top element: 3
  Top element after popping: 2
  Is stack empty? No
○ riyasingh@riyas-MacBook-Air DSA lab file %
```

Test Case: Any two (screenshot)

Conclusion: Therefore, using switch cases, we can perform multiple operations like push, pop, and peek in a stack using array.

Name of Student:Riya Singh

Roll Number: 26

Experiment No: 3

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

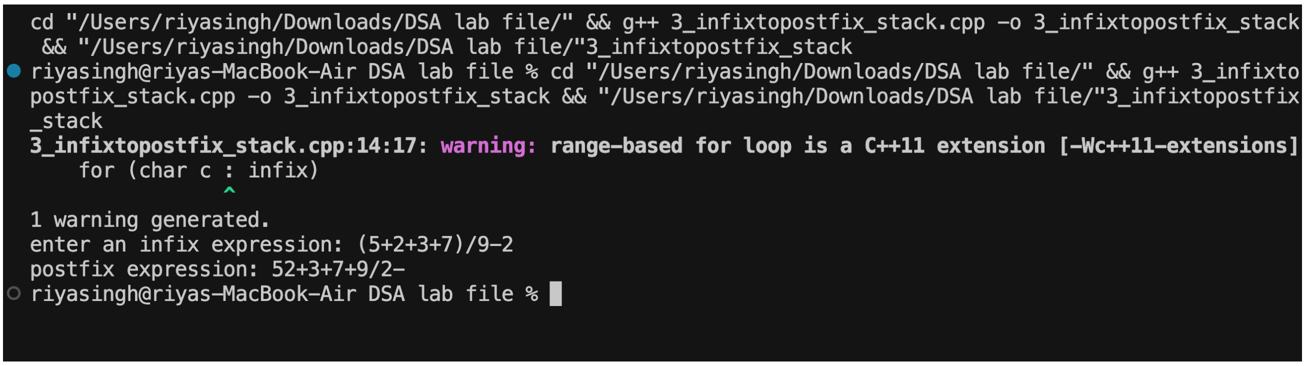
Theory: Stack is an Abstract Data Type which can be implemented using Linked List or Array. It consists of a variable named Top which points to the topmost element of the stack. Stack follows LIFO principle(Last In, First Out) which means that the element which is inserted last will be deleted first. There are three operations in Stack: Push- insertion from top, Pop- deletion from top, Peek- returning the topmost element from the stack. Using stack, we can convert an infix expression to postfix expression by pushing the operators and brackets in the stack and the operands to the expression and popping the elements to the expression through operator precedence after encountering a closing bracket.

Code:

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

int main() 
{
```

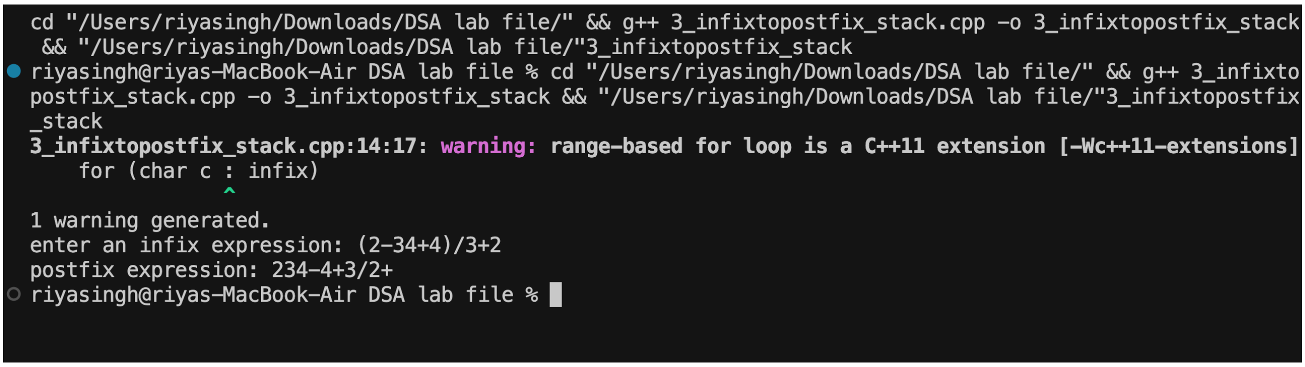
```cpp
    stack<char> s;
    string infix, postfix;

    cout << "enter an infix expression: ";
    getline(cin, infix);

    for (char c : infix)
    {
        if (isalnum(c))
        {
            postfix += c;
        }

        else if (c == '(')
        {
            s.push(c);
        }

        else if (c == ')')
        {
            while (s.top() != '(')
            {
                postfix += s.top(); s.pop();
            }
            s.pop();
        }
        else
        {
            while (!s.empty() && s.top() != '(' && ((c == '+' || c == '-') ? 1 : 2) <= ((s.top() == '+' || s.top() == '-') ? 1 : 2))
            {
                postfix += s.top(); s.pop();
            }
            s.push(c);
        }
    }

    while (!s.empty())
    {
        postfix += s.top(); s.pop();
    }

    cout << "postfix expression: " << postfix << endl;

    return 0;
}
```

Output: (screenshot)

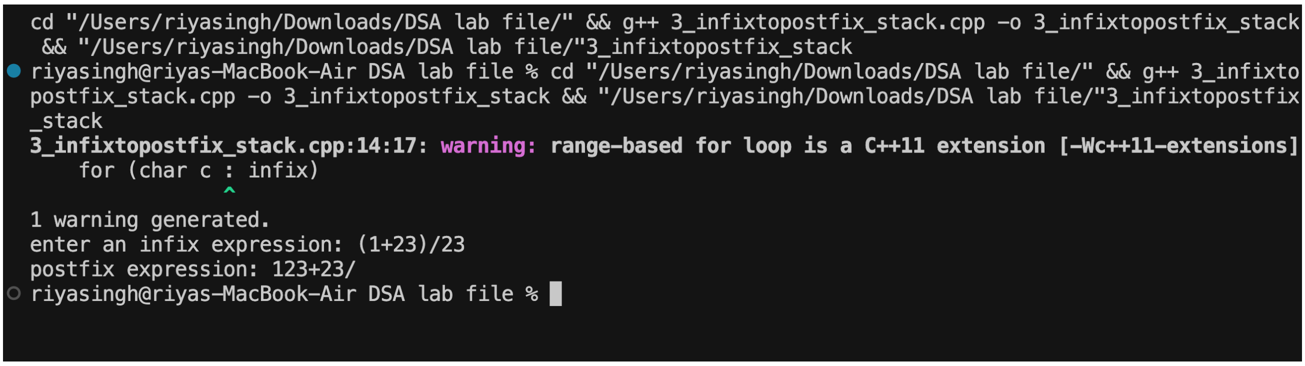


```
  cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ 3_infixtopostfix_stack.cpp -o 3_infixtopostfix_stack
   && "/Users/riyasingh/Downloads/DSA lab file/"3_infixtopostfix_stack
● riyasingh@riyas-MacBook-Air DSA lab file % cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ 3_infixto
  postfix_stack.cpp -o 3_infixtopostfix_stack && "/Users/riyasingh/Downloads/DSA lab file/"3_infixtopostfix
  _stack
  3_infixtopostfix_stack.cpp:14:17: warning: range-based for loop is a C++11 extension [-Wc++11-extensions]
      for (char c : infix)
                  ^

  1 warning generated.
  enter an infix expression: (5+2+3+7)/9-2
  postfix expression: 52+3+7+9/2-
○ riyasingh@riyas-MacBook-Air DSA lab file % █
```

Test Case: Any two (screenshot)



```
  cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ 3_infixtopostfix_stack.cpp -o 3_infixtopostfix_stack
   && "/Users/riyasingh/Downloads/DSA lab file/"3_infixtopostfix_stack
● riyasingh@riyas-MacBook-Air DSA lab file % cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ 3_infixto
  postfix_stack.cpp -o 3_infixtopostfix_stack && "/Users/riyasingh/Downloads/DSA lab file/"3_infixtopostfix
  _stack
  3_infixtopostfix_stack.cpp:14:17: warning: range-based for loop is a C++11 extension [-Wc++11-extensions]
      for (char c : infix)
                  ^

  1 warning generated.
  enter an infix expression: (2-34+4)/3+2
  postfix expression: 234-4+3/2+
○ riyasingh@riyas-MacBook-Air DSA lab file % █
```



```
  cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ 3_infixtopostfix_stack.cpp -o 3_infixtopostfix_stack
   && "/Users/riyasingh/Downloads/DSA lab file/"3_infixtopostfix_stack
● riyasingh@riyas-MacBook-Air DSA lab file % cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ 3_infixto
  postfix_stack.cpp -o 3_infixtopostfix_stack && "/Users/riyasingh/Downloads/DSA lab file/"3_infixtopostfix
  _stack
  3_infixtopostfix_stack.cpp:14:17: warning: range-based for loop is a C++11 extension [-Wc++11-extensions]
      for (char c : infix)
                  ^

  1 warning generated.
  enter an infix expression: (1+23)/23
  postfix expression: 123+23/
○ riyasingh@riyas-MacBook-Air DSA lab file % █
```

Conclusion: Therefore, using stack ADT, we can convert infix expression to postfix expression by operations like Push and Pop.

Name of Student: Riya Singh

Roll Number: 26

Experiment No: 4

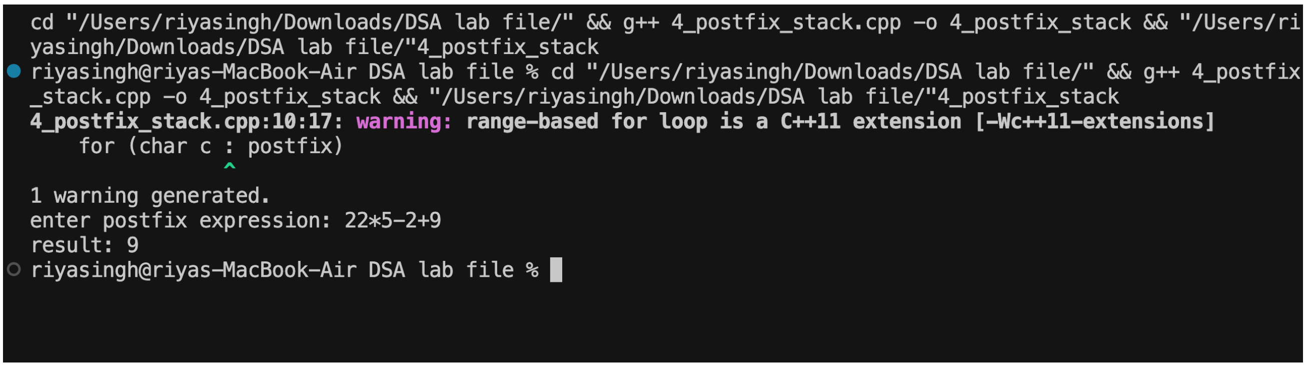
Title: Evaluate Postfix expression using Stack ADT.

Theory: Stack is an Abstract Data Type which can be implemented using Linked List or Array. It consists of a variable named Top which points to the topmost element of the stack. Stack follows LIFO principle(Last In, First Out) which means that the element which is inserted last will be deleted first. There are three operations in Stack: Push- insertion from top, Pop- deletion from top, Peek- returning the topmost element from the stack. Using stack, we can evaluate a postfix expression by pushing the operands in the stack and popping them and evaluating them when an operator is encountered and popping the result back in the stack and printing the topmost element after the whole expression is evaluated.

Code:

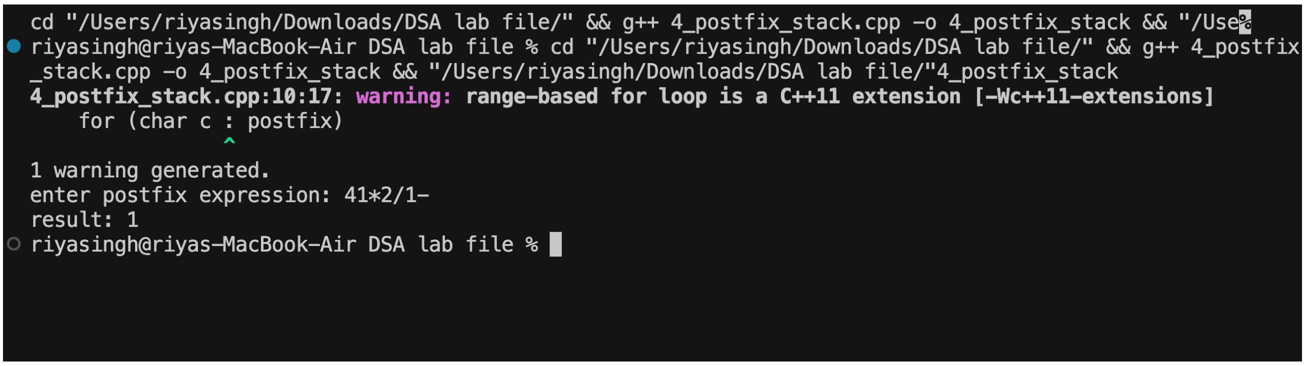
```
// Evaluate Postfix Expression using Stack ADT.
#include <iostream>
#include <stack>
using namespace std;

int evaluate(const string& postfix)
{
    stack<int> s;
    for (char c : postfix)
    {
        if (isdigit(c))
        {
            s.push(c - '0');
        }
        else
        {
            int operand2 = s.top(); s.pop();
            int operand1 = s.top(); s.pop();

            switch(c)
            {
                case '+': s.push(operand1 + operand2);
                break;
                case '-': s.push(operand1 - operand2);
                break;
                case '*': s.push(operand1 * operand2);
                break;
                case '/': s.push(operand1 / operand2);
                break;
            }
        }
    }

    return s.top();
}

int main()
{
    string postfixexp;

    cout << "enter postfix expression: ";
    getline(cin, postfixexp);

    cout << "result: " << evaluate(postfixexp) << endl;

    return 0;
}
```

Output: (screenshot)

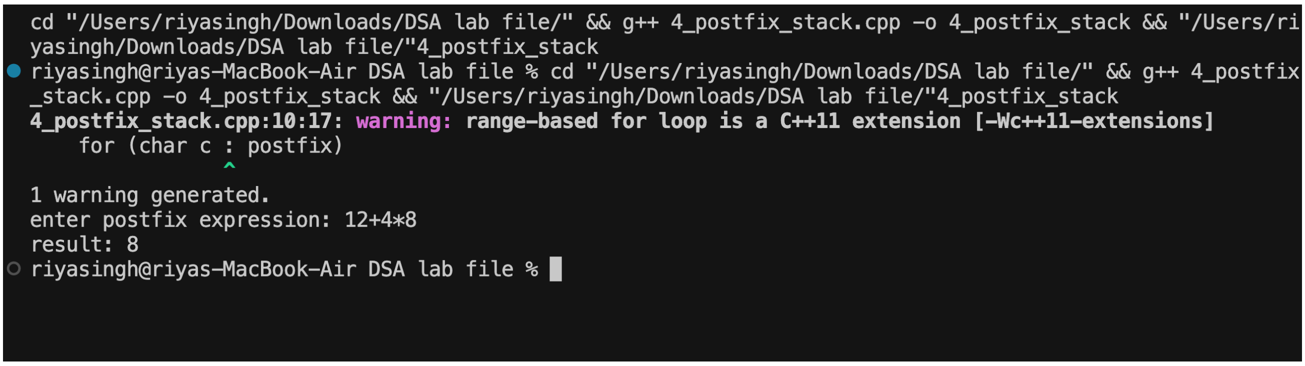


```
cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ 4_postfix_stack.cpp -o 4_postfix_stack && "/Users/ri
yasingh/Downloads/DSA lab file/"4_postfix_stack
riyasingh@riyas-MacBook-Air DSA lab file % cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ 4_postfix
_stack.cpp -o 4_postfix_stack && "/Users/riyasingh/Downloads/DSA lab file/"4_postfix_stack
4_postfix_stack.cpp:10:17: warning: range-based for loop is a C++11 extension [-Wc++11-extensions]
    for (char c : postfix)
                ^
1 warning generated.
enter postfix expression: 22*5-2+9
result: 9
riyasingh@riyas-MacBook-Air DSA lab file % 
```

Test Case: Any two (screenshot)



```
 cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ 4_postfix_stack.cpp -o 4_postfix_stack && "/Use%
● riyasingh@riyas-MacBook-Air DSA lab file % cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ 4_postfix
 _stack.cpp -o 4_postfix_stack && "/Users/riyasingh/Downloads/DSA lab file/"4_postfix_stack
 4_postfix_stack.cpp:10:17: warning: range-based for loop is a C++11 extension [-Wc++11-extensions]
     for (char c : postfix)
                 ^
 1 warning generated.
 enter postfix expression: 41*2/1-
 result: 1
○ riyasingh@riyas-MacBook-Air DSA lab file %
```



```
 cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ 4_postfix_stack.cpp -o 4_postfix_stack && "/Users/ri
 yasingh/Downloads/DSA lab file/"4_postfix_stack
● riyasingh@riyas-MacBook-Air DSA lab file % cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ 4_postfix
 _stack.cpp -o 4_postfix_stack && "/Users/riyasingh/Downloads/DSA lab file/"4_postfix_stack
 4_postfix_stack.cpp:10:17: warning: range-based for loop is a C++11 extension [-Wc++11-extensions]
     for (char c : postfix)
                 ^
 1 warning generated.
 enter postfix expression: 12+4*8
 result: 8
○ riyasingh@riyas-MacBook-Air DSA lab file %
```

Conclusion: Therefore, using stack ADT, we can evaluate a postfix expression by operations like Push and Pop.

Name of Student: Riya Singh

Roll Number: 26

Experiment No: 5

Title: Implement Linear Queue ADT using array.

Theory: Array is a collection of elements of similar data types and has a fixed size. We can access an element of the array through it's index. Indexing starts from 0 till n-1(where n=size of array).

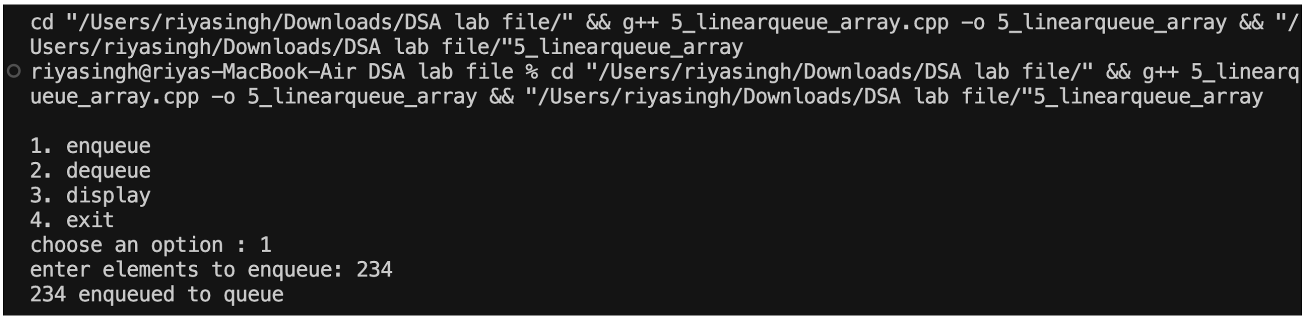
Queue is an Abstract Data Type which can be implemented using Linked List or Array. It consists of two variables named Front and Rear which point to the first and last elements of the stack, respectively. Queue follows FIFO principle(First In, First Out) which means that the element which is inserted first will be deleted first. There are three operations in Stack: Enqueue- insertion from rear, Dequeue- deletion from front, Peek- returning the frontmost element from the queue.

Code:

```
// Implement Linear Queue ADT using array.
#include <iostream>
using namespace std;

const int MAX_SIZE = 100;
```

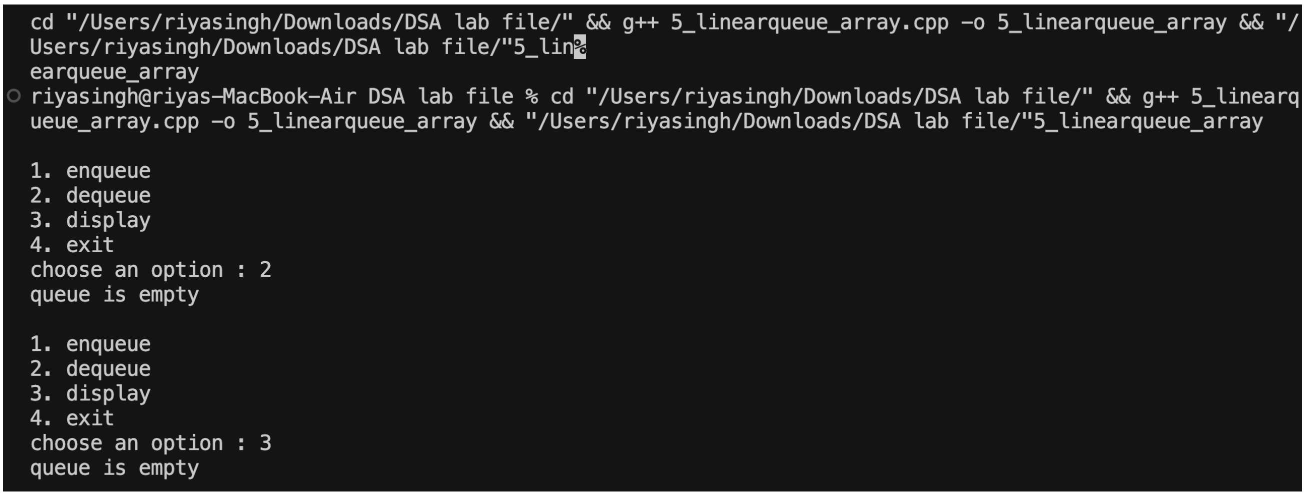
```
class Queue
{
private:
    int front, rear;
    int arr[MAX_SIZE];

public:
    Queue()
    {
        front = -1;
        rear = -1;
    }

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

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

    void enqueue(int data)
    {
        if (isFull())
        {
            cout << "queue is full" << endl;
            return;
        }
        else if (isEmpty())
        {
            front = rear = 0;
        }
        else
        {
            rear++;
        }

        arr[rear] = data;
        cout << data << " enqueued to queue" << endl;
    }

    void dequeue()
    {
        if (isEmpty())
        {
            cout << "queue is empty" << endl;
            return;
        }
        else if (front == rear)
        {
            cout << arr[front] << " dequeued from queue" << endl;
            front = rear = -1;
        }
        else
        {
            cout << arr[front] << " dequeued from queue." << endl;
            front++;
        }
    }

    void display()
    {
        if (isEmpty())
        {
            cout << "queue is empty" << endl;
            return;
        }

        cout << "elements in the queue: ";

        for (int i = front; i <= rear; i++)
        {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
};

int main()
{
    Queue q;
    int choice, data;

    do {
        cout << "\n1. enqueue\n";
        cout << "2. dequeue\n";
        cout << "3. display\n";
        cout << "4. exit\n";
        cout << "choose an option : ";
        cin >> choice;

        switch(choice)
        {
            case 1:
                cout << "enter elements to enqueue: ";
                cin >> data;
                q.enqueue(data);
                break;
            case 2:
                q.dequeue();
                break;
            case 3:
                q.display();
                break;
            case 4:
                cout << "exited!" << endl;
                break;
            default:
                cout << "invalid choice." << endl;
        }
    }
    while (choice != 4);

    return 0;
}
```

Output: (screenshot)



```
 cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ 5_linearqueue_array.cpp -o 5_linearqueue_array && "/
 Users/riyasingh/Downloads/DSA lab file/"5_linearqueue_array
○ riyasingh@riyas-MacBook-Air DSA lab file % cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ 5_linearq
 ueue_array.cpp -o 5_linearqueue_array && "/Users/riyasingh/Downloads/DSA lab file/"5_linearqueue_array

 1. enqueue
 2. dequeue
 3. display
 4. exit
 choose an option : 1
 enter elements to enqueue: 234
 234 enqueued to queue
```

Test Case: Any two (screenshot)



```
 cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ 5_linearqueue_array.cpp -o 5_linearqueue_array && "/
 Users/riyasingh/Downloads/DSA lab file/"5_lin%
 earqueue_array
○ riyasingh@riyas-MacBook-Air DSA lab file % cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ 5_linearq
 ueue_array.cpp -o 5_linearqueue_array && "/Users/riyasingh/Downloads/DSA lab file/"5_linearqueue_array

 1. enqueue
 2. dequeue
 3. display
 4. exit
 choose an option : 2
 queue is empty

 1. enqueue
 2. dequeue
 3. display
 4. exit
 choose an option : 3
 queue is empty
```

Conclusion: Therefore, using array, we can implement a linear queue and perform operations like Enqueue, Dequeue and Peek.

Name of Student: Riya Singh

Roll Number: 26

Experiment No: 6

Title: Implement Circular Queue ADT using array.

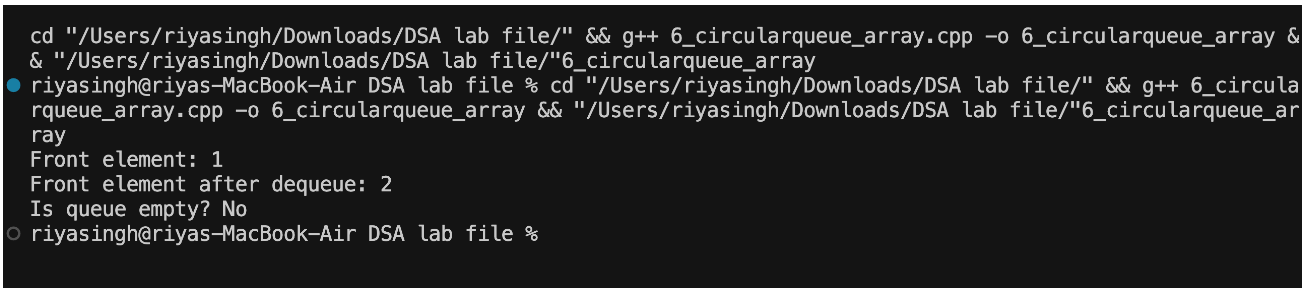
Theory: Array is a collection of elements of similar data types and has a fixed size. We can access an element of the array through it's index. Indexing starts from 0 till n-1(where n=size of array).

Queue is an Abstract Data Type which can be implemented using Linked List or Array. It consists of two variables named Front and Rear which point to the first and last elements of the stack, respectively. Queue follows FIFO principle(First In, First Out) which means that the element which is inserted first will be deleted first. There are three operations in Stack: Enqueue- insertion from rear, Dequeue- deletion from front, Peek- returning the frontmost element from the queue. As size of array is fixed, in order to overcome the challenges, we can move the rear pointer to the start of the array if rear=n-1 and front is not at first index, so we can continue to insert elements.

Code:

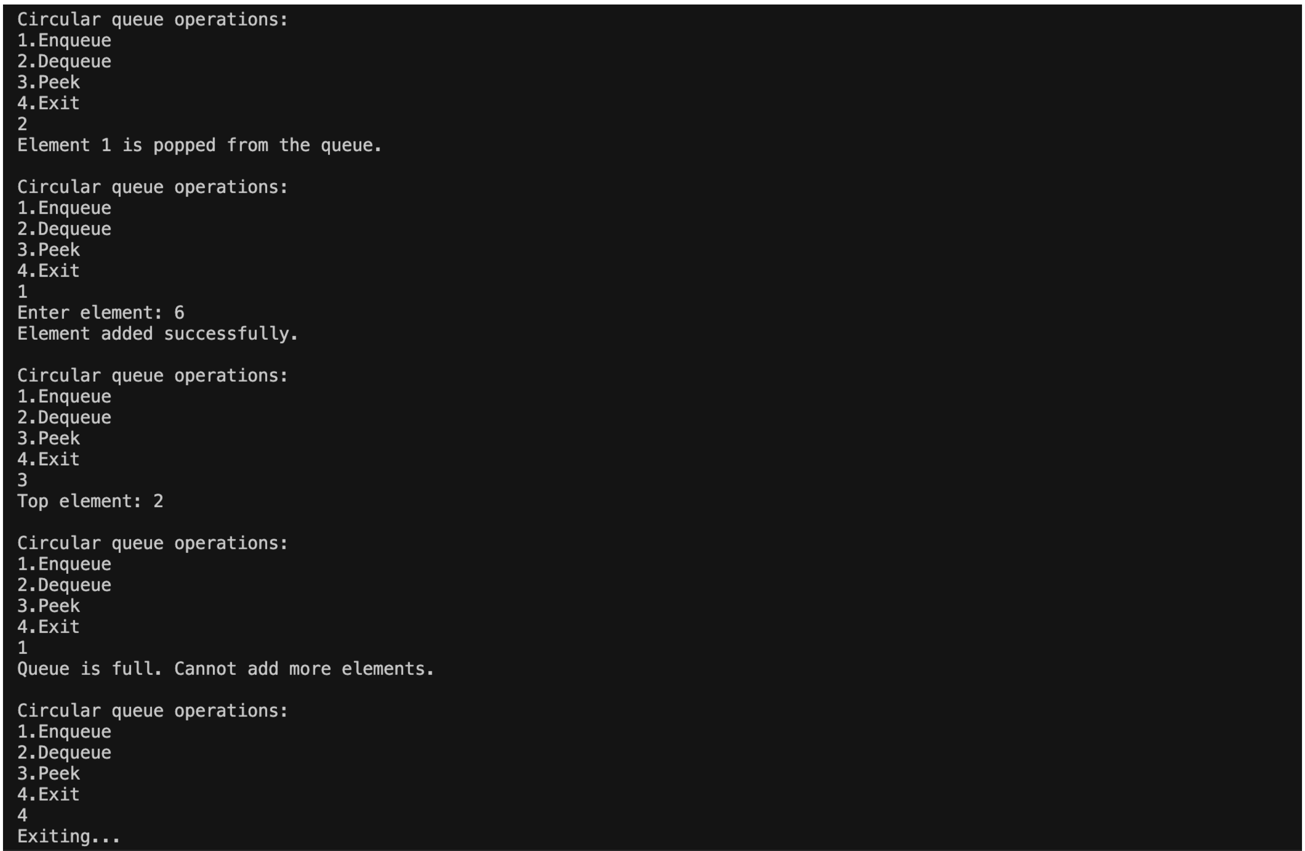
```
//Implement Circular Queue ADT using array.
#include <iostream>
const int MAX_SIZE = 100;
class CircularQueue {
private:
    int arr[MAX_SIZE];
    int front;
    int rear;

public:
    CircularQueue() {
        front = -1;
        rear = -1;
    }
    bool isEmpty() {
        return front == -1 && rear == -1;
    }
    bool isFull() {
        return (rear + 1) % MAX_SIZE == front;
    }
    void enqueue(int value) {
        if (isFull()) {
            std::cout << "Queue Overflow\n";
            return;
        }
        if (isEmpty()) {
            front = 0;
            rear = 0;
        } else {
            rear = (rear + 1) % MAX_SIZE;
        }
        arr[rear] = value;
    }
    void dequeue() {
        if (isEmpty()) {
            std::cout << "Queue Underflow\n";
            return;
        }
        if (front == rear) {
            front = -1;
            rear = -1;
        } else {
            front = (front + 1) % MAX_SIZE;
        }
    }
    int peek() {
        if (isEmpty()) {
            std::cout << "Queue is empty\n";
            return -1;
        }
        return arr[front];
    }
};

int main() {
    CircularQueue queue;

    queue.enqueue(1);
    queue.enqueue(2);
    queue.enqueue(3);

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

    queue.dequeue();
    std::cout << "Front element after dequeue: " << queue.peek() <<
std::endl;

    std::cout << "Is queue empty? " << (queue.isEmpty() ? "Yes" :
"No") << std::endl;

    return 0;
}
```

Output: (screenshot)



```
  cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ 6_circularqueue_array.cpp -o 6_circularqueue_array &
  & "/Users/riyasingh/Downloads/DSA lab file/"6_circularqueue_array
● riyasingh@riyas-MacBook-Air DSA lab file % cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ 6_circula
  rqueue_array.cpp -o 6_circularqueue_array && "/Users/riyasingh/Downloads/DSA lab file/"6_circularqueue_ar
  ray
  Front element: 1
  Front element after dequeue: 2
  Is queue empty? No
○ riyasingh@riyas-MacBook-Air DSA lab file %
```

Test Case: Any two (screenshot)



```
Circular queue operations:
1.Enqueue
2.Dequeue
3.Peek
4.Exit
2
Element 1 is popped from the queue.

Circular queue operations:
1.Enqueue
2.Dequeue
3.Peek
4.Exit
1
Enter element: 6
Element added successfully.

Circular queue operations:
1.Enqueue
2.Dequeue
3.Peek
4.Exit
3
Top element: 2

Circular queue operations:
1.Enqueue
2.Dequeue
3.Peek
4.Exit
1
Queue is full. Cannot add more elements.

Circular queue operations:
1.Enqueue
2.Dequeue
3.Peek
4.Exit
4
Exiting...
```

Conclusion: Therefore, using array, we can implement a circular queue and perform operations like Enqueue, Dequeue and Peek without being constrained by the limitation of the fixed size of the array.

Name of Student: Riya Singh

Roll Number: 26

Experiment No: 7

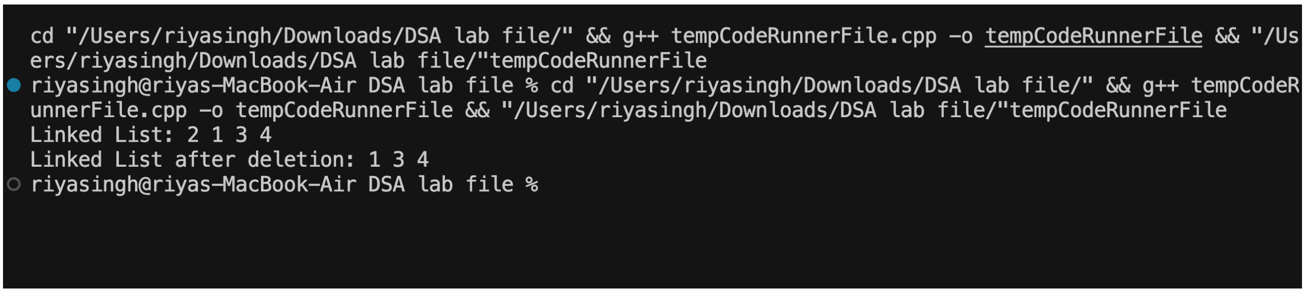
Title: Implement Singly Linked List ADT.

Theory: Linked List is a data type which consists of nodes which contain data and a next pointer which points to the next node in the list. It stores the address of the next node. There is a start pointer in stack memory which points to the first node in the heap memory. It utilises dynamic memory and allocates heap memory to the nodes in the list. The last node's next pointer has NULL value to indicate it's the last node in the list.

Code:

```
//Implement Singly Linked List ADT.
#include <iostream>
class Node {
public:
    int data;
    Node* next;

    Node(int value) {
        data = value;
        next = nullptr;
    }
};
class SinglyLinkedList {
private:
    Node* head;

public:
    SinglyLinkedList() {
        head = nullptr;
    }
    void insertAtBeginning(int value) {
        Node* newNode = new Node(value);
        newNode->next = head;
        head = newNode;
    }
    void insertAtEnd(int value) {
        Node* newNode = new Node(value);
        if (head == nullptr) {
            head = newNode;
            return;
        }
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
    void display() {
        Node* temp = head;
        while (temp != nullptr) {
            std::cout << temp->data << " ";
            temp = temp->next;
        }
        std::cout << std::endl;
    }
    void deleteNode(int value) {
        if (head == nullptr)
            return;
        if (head->data == value) {
            Node* temp = head;
            head = head->next;
            delete temp;
            return;
        }
        Node* prev = head;
        Node* curr = head->next;
        while (curr != nullptr) {
            if (curr->data == value) {
                prev->next = curr->next;
                delete curr;
                return;
            }
            prev = curr;
            curr = curr->next;
        }
    }
};

int main() {
    SinglyLinkedList list;

    list.insertAtBeginning(1);
    list.insertAtBeginning(2);
    list.insertAtEnd(3);
    list.insertAtEnd(4);

    std::cout << "Linked List: ";
    list.display();

    list.deleteNode(2);

    std::cout << "Linked List after deletion: ";
    list.display();

    return 0;
}
```

Output: (screenshot)



```
  cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/Us
  ers/riyasingh/Downloads/DSA lab file/"tempCodeRunnerFile
● riyasingh@riyas-MacBook-Air DSA lab file % cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ tempCodeR
  unnerFile.cpp -o tempCodeRunnerFile && "/Users/riyasingh/Downloads/DSA lab file/"tempCodeRunnerFile
  Linked List: 2 1 3 4
  Linked List after deletion: 1 3 4
○ riyasingh@riyas-MacBook-Air DSA lab file %
```

Test Case: Any two (screenshot):Same as Above

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

Name of Student: Riya Singh

Roll Number: 26

Experiment No: 8

Title: Implement Circular Linked List ADT.

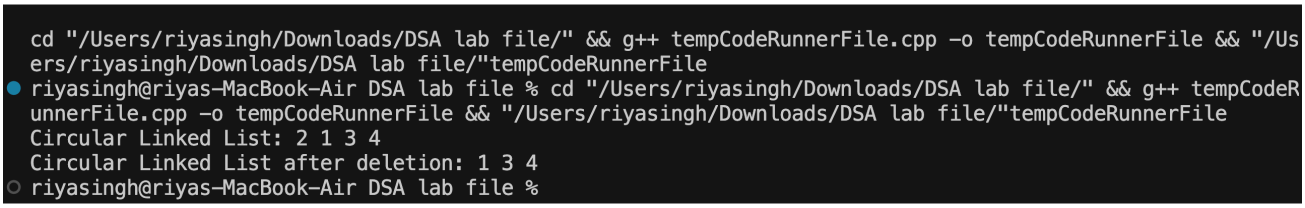
Theory: Linked List is a data type which consists of nodes which contain data and a next pointer which points to the next node in the list. It stores the address

of the next node. There is a start pointer in stack memory which points to the first node in the heap memory. It utilises dynamic memory and allocates heap memory to the nodes in the list. The last node's next pointer has the address of first node, hence it's called circular linked list.

Code:

```cpp
// Implement Circular Linked List ADT.
#include <iostream>
class Node {
public:
    int data;
    Node* next;

    Node(int value) {
        data = value;
        next = nullptr;
    }
};
class CircularLinkedList {
private:
    Node* head;

public:
    CircularLinkedList() {
        head = nullptr;
    }
    void insertAtBeginning(int value) {
        Node* newNode = new Node(value);
        if (head == nullptr) {
            head = newNode;
            head->next = head; // Circular link to itself
            return;
        }
        Node* temp = head;
        while (temp->next != head) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->next = head;
        head = newNode;
    }
    void insertAtEnd(int value) {
        Node* newNode = new Node(value);
        if (head == nullptr) {
            head = newNode;
            head->next = head;
            return;
        }
        Node* temp = head;
        while (temp->next != head) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->next = head;
    }
    void display() {
        if (head == nullptr) {
            std::cout << "List is empty\n";
            return;
        }
        Node* temp = head;
        do {
            std::cout << temp->data << " ";
            temp = temp->next;
        } while (temp != head);
        std::cout << std::endl;
    }
    void deleteNode(int value) {
        if (head == nullptr)
            return;
        Node* temp = head;
        Node* prev = nullptr;
        do {
            if (temp->data == value) {
                if (temp == head) {
                    if (temp->next == head) {
                        delete temp;
                        head = nullptr;
                        return;
                    } else {
                        Node* last = head;
                        while (last->next != head) {
                            last = last->next;
                        }
                        last->next = head->next;
                        head = head->next;
                        delete temp;
                        return;
                    }
                } else {
                    prev->next = temp->next;
                    delete temp;
                    return;
                }
            }
            prev = temp;
            temp = temp->next;
        } while (temp != head);
    }
};

int main() {
    CircularLinkedList list;

    list.insertAtBeginning(1);
    list.insertAtBeginning(2);
    list.insertAtEnd(3);
    list.insertAtEnd(4);

    std::cout << "Circular Linked List: ";
    list.display();

    list.deleteNode(2);

    std::cout << "Circular Linked List after deletion: ";
    list.display();

    return 0;
}
```

Output: (screenshot)



```
  cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/Us
  ers/riyasingh/Downloads/DSA lab file/"tempCodeRunnerFile
● riyasingh@riyas-MacBook-Air DSA lab file % cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ tempCodeR
  unnerFile.cpp -o tempCodeRunnerFile && "/Users/riyasingh/Downloads/DSA lab file/"tempCodeRunnerFile
  Circular Linked List: 2 1 3 4
  Circular Linked List after deletion: 1 3 4
○ riyasingh@riyas-MacBook-Air DSA lab file %
```

Test Case: Any two (screenshot)

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

Name of Student: Riya Singh

Roll Number: 26

Experiment No: 9

Title: Implement Stack ADT using Linked List.

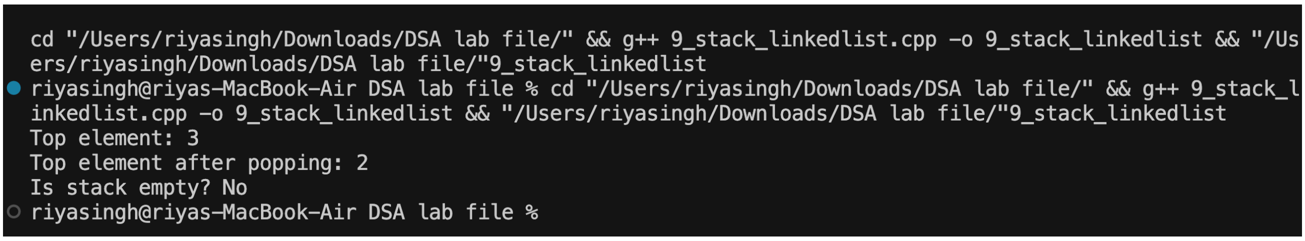
Theory: Stack is an Abstract Data Type which can be implemented using Linked List or Array. It consists of a variable named Top which points to the topmost element of the stack. Stack follows LIFO principle(Last In, First Out) which means that the element which is inserted last will be deleted first. There are three operations in Stack: Push- insertion from top, Pop- deletion from top, Peek- returning the topmost element from the stack. We can implement insertion at beginning, deletion from beginning algorithms to implement Stack using Linked List.

Linked List is a data type which consists of nodes which contain data and a next pointer which points to the next node in the list. It stores the address of the next node. There is a start pointer in stack memory which points to the first node in the heap memory. It utilises dynamic memory and allocates heap memory to the nodes in the list. The last node's next pointer has the address of first node, hence it's called circular linked list.

Code:

```
// Implement Stack ADT using Linked List
#include <iostream>
class Node {
public:
    int data;
    Node* next;

    Node(int value) {
        data = value;
        next = nullptr;
    }
};
class Stack {
private:
    Node* top;

public:
    Stack() {
        top = nullptr;
    }
    bool isEmpty() {
        return top == nullptr;
    }
    void push(int value) {
        Node* newNode = new Node(value);
        newNode->next = top;
        top = newNode;
    }
    void pop() {
        if (isEmpty()) {
            std::cout << "Stack Underflow\n";
            return;
        }
        Node* temp = top;
        top = top->next;
        delete temp;
    }
    int peek() {
        if (isEmpty()) {
            std::cout << "Stack is empty\n";
            return -1;
        }
        return top->data;
    }
};
int main() {
    Stack stack;

    stack.push(1);
    stack.push(2);
    stack.push(3);

    std::cout << "Top element: " << stack.peek() << std::endl;

    stack.pop();
    std::cout << "Top element after popping: " << stack.peek() <<
std::endl;

    std::cout << "Is stack empty? " << (stack.isEmpty() ? "Yes" :
"No") << std::endl;

    return 0;
}
```

Output: (screenshot)



```
  cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ 9_stack_linkedlist.cpp -o 9_stack_linkedlist && "/Us
  ers/riyasingh/Downloads/DSA lab file/"9_stack_linkedlist
● riyasingh@riyas-MacBook-Air DSA lab file % cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ 9_stack_l
  inkedlist.cpp -o 9_stack_linkedlist && "/Users/riyasingh/Downloads/DSA lab file/"9_stack_linkedlist
  Top element: 3
  Top element after popping: 2
  Is stack empty? No
○ riyasingh@riyas-MacBook-Air DSA lab file %
```

Test Case: Any two (screenshot):Same as Above

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

Name of Student: Riya Singh

Roll Number: 26

Experiment No: 10

Title: Implement Linear Queue ADT using Linked List.

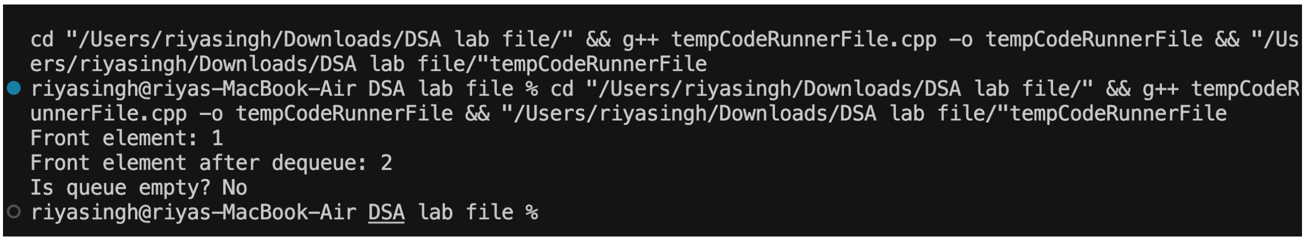
Theory: Queue is an Abstract Data Type which can be implemented using Linked List or Array. It consists of two variables named Front and Rear which point to the first and last elements of the stack, respectively. Queue follows FIFO principle(First In, First Out) which means that the element which is inserted first will be deleted first. There are three operations in Stack: Enqueue- insertion from rear, Dequeue- deletion from front, Peek- returning the frontmost element from the queue. It can be implemented by insertion at end and deletion from beginning algorithms.

Linked List is a data type which consists of nodes which contain data and a next pointer which points to the next node in the list. It stores the address of the next node. There is a start pointer in stack memory which points to the first node in the heap memory. It utilises dynamic memory and allocates heap memory to the nodes in the list. The last node's next pointer has the address of first node, hence it's called circular linked list.

Code:

```
// Implement Linear Queue ADT using Linked List
#include <iostream>
class Node {
public:
    int data;
    Node* next;

    Node(int value) {
        data = value;
        next = nullptr;
    }
};
class Queue {
private:
    Node* front;
    Node* rear;

public:
    Queue() {
        front = nullptr;
        rear = nullptr;
    }
    bool isEmpty() {
        return front == nullptr;
    }
    void enqueue(int value) {
        Node* newNode = new Node(value);
        if (isEmpty()) {
            front = newNode;
            rear = newNode;
        } else {
            rear->next = newNode;
            rear = newNode;
        }
    }
    void dequeue() {
        if (isEmpty()) {
            std::cout << "Queue Underflow\n";
            return;
        }
        Node* temp = front;
        front = front->next;
        delete temp;
        if (front == nullptr) {
            rear = nullptr;
        }
    }
    int peek() {
        if (isEmpty()) {
            std::cout << "Queue is empty\n";
            return -1;
        }
        return front->data;
    }
};

int main() {
    Queue queue;
    queue.enqueue(1);
    queue.enqueue(2);
    queue.enqueue(3);

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

    queue.dequeue();
    std::cout << "Front element after dequeue: " << queue.peek() <<
std::endl;

    std::cout << "Is queue empty? " << (queue.isEmpty() ? "Yes" :
"No") << std::endl;

    return 0;
}
```

Output: (screenshot)



```
 cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/Us
 ers/riyasingh/Downloads/DSA lab file/"tempCodeRunnerFile
● riyasingh@riyas-MacBook-Air DSA lab file % cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ tempCodeR
 unnerFile.cpp -o tempCodeRunnerFile && "/Users/riyasingh/Downloads/DSA lab file/"tempCodeRunnerFile
 Front element: 1
 Front element after dequeue: 2
 Is queue empty? No
○ riyasingh@riyas-MacBook-Air DSA lab file %
```

Test Case: Any two (screenshot):Same as Above

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

Name of Student: Riya Singh

Roll Number: 26

Experiment No: 11

Title: Implement Binary Search Tree ADT using Linked List.

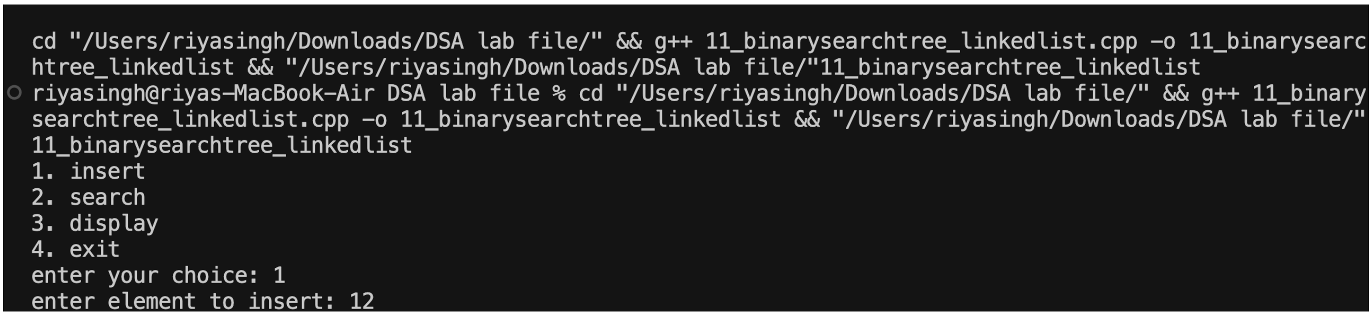
Theory:

A binary tree is a non-linear data structure in which there is a root node and each parent node has 0,1 or 2 child nodes at most. In binary search tree, all the nodes having values less than that of the root node are present in the left subtree of the root node and all the nodes having values greater than or equal to that of the root node are present in the right subtree of the root node.

Code:

```
// Implement Binary Search Tree ADT using Linked List.
#include <iostream>
using namespace std;

struct Node
{
    int data;
    Node* left;
    Node* right;
};

class bst
{
private:
    Node* root;

    Node* createnode(int value)
    {
        Node* newNode = new Node;
        newNode->data = value;
        newNode->left = nullptr;
        newNode->right = nullptr;
        return newNode;
    }

    Node* insert(Node* root, int value)
    {
        if (root == nullptr)
        {
            return createnode(value);
        }

        if (value < root->data)
        {
            root->left = insert(root->left, value);
        }
        else if (value > root->data)
        {
            root->right = insert(root->right, value);
        }
        return root;
    }

    void traversal(Node* root)
    {
        if (root != nullptr)
        {
            traversal(root->left);
            cout << root->data << " ";
            traversal(root->right);
        }
    }

public:
    bst()
    {
        root = nullptr;
    }

    void insert(int value)
    {
        root = insert(root, value);
    }

    bool search(int value)
    {
        Node* current = root;
        while (current != nullptr)
        {
            if (value == current->data)
            {
                return true;
            }
            else if (value < current->data)
            {
                current = current->left;
            }
            else
            {
                current = current->right;
            }
        }
        return false;
    }

    void display()
    {
        if (root == nullptr)
        {
            cout << "binary search tree is empty\n";
        }
        else
        {
            cout << "binary search tree: ";
            traversal(root);
            cout << endl;
        }
    }
};

int main()
{
    bst bst;
    char choice;
    int value;

    do
    {
        cout << "1. insert\n";
        cout << "2. search\n";
        cout << "3. display\n";
        cout << "4. exit\n";
        cout << "enter your choice: ";
        cin >> choice;

        switch(choice)
        {
            case '1':
                cout << "enter element to insert: ";
                cin >> value;
                bst.insert(value);
                break;
            case '2':
                cout << "enter element to search: ";
                cin >> value;
                if (bst.search(value))
                {
                    cout << "found in binary search tree.\n";
                } else {
                    cout << "not found in binary search tree.\n";
                }
                break;
            case '3':
                bst.display();
                break;
            case '4':
                cout << "exited!\n";
                break;
            default:
                cout << "invalid choice\n";
        }
    }
    while(choice != '4');

    return 0;
}
```

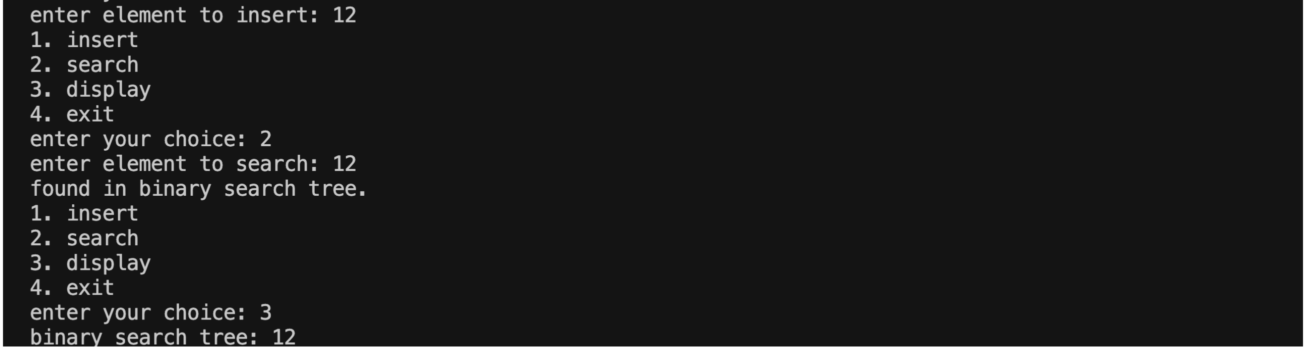
Output: (screenshot



```
cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ 11_binarysearchtree_linkedlist.cpp -o 11_binarysearc
htree_linkedlist && "/Users/riyasingh/Downloads/DSA lab file/"11_binarysearchtree_linkedlist
○ riyasingh@riyas-MacBook-Air DSA lab file % cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ 11_binary
searchtree_linkedlist.cpp -o 11_binarysearchtree_linkedlist && "/Users/riyasingh/Downloads/DSA lab file/"
11_binarysearchtree_linkedlist
1. insert
2. search
3. display
4. exit
enter your choice: 1
enter element to insert: 12
```

Conclusion: Therefore, we can implement Binary Search Tree ADT using Linked List.

Test Case: Any two (screenshot)



```
enter element to insert: 12
1. insert
2. search
3. display
4. exit
enter your choice: 2
enter element to search: 12
found in binary search tree.
1. insert
2. search
3. display
4. exit
enter your choice: 3
binary search tree: 12
```

Conclusion: Therefore, we can implement Binary Search Tree ADT using Linked List.

Name of Student: Riya Singh

Roll Number: 26

Experiment No: 12

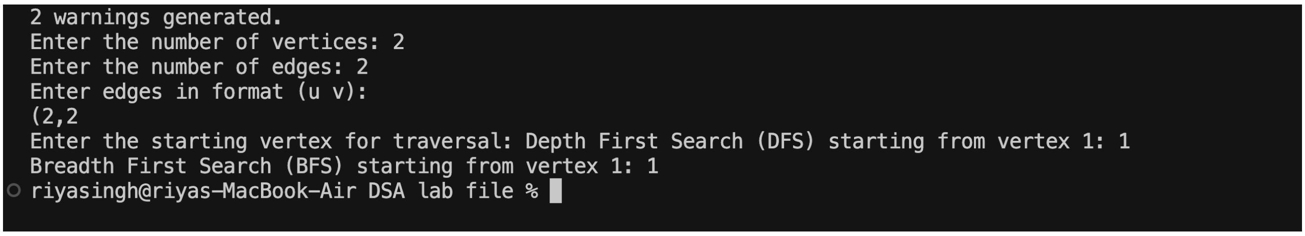
Title: Implement Graph Traversal techniques: a) Depth First Search b) Breadth First Search

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

Code:

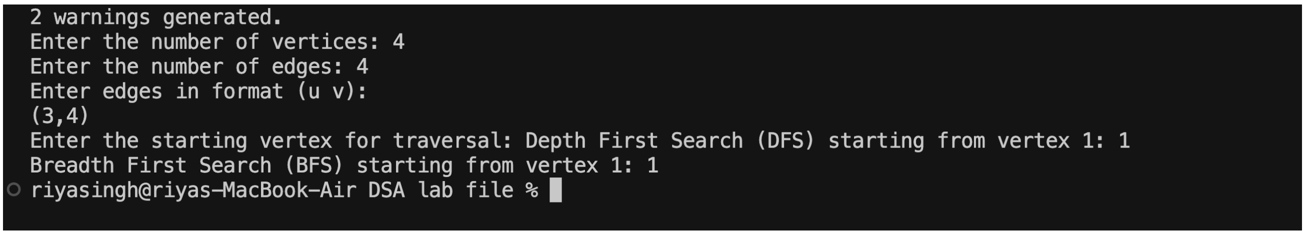
```
// Implement Graph Traversal techniques:
// a) Depth First Search b) Breadth First Search
#include <iostream>
#include <vector>
#include <queue>
#include <stack>

using namespace std;

class Graph {
private:
    int V; // Number of vertices
    vector<vector<int> > adj; // Adjacency list

public:
    Graph(int vertices) : V(vertices) {
        adj.resize(V);
    }

    void addEdge(int u, int v) {
        adj[u].push_back(v);
    }

    void DFSUtil(int v, vector<bool>& visited) {
        visited[v] = true;
        cout << v << " ";

        for (int i : adj[v]) {
            if (!visited[i])
                DFSUtil(i, visited);
        }
    }

    void DFS(int start) {
        vector<bool> visited(V, false);
        DFSUtil(start, visited);
    }

    void BFS(int start) {
        vector<bool> visited(V, false);
        queue<int> q;
        visited[start] = true;
        q.push(start);

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

            for (int i : adj[v]) {
                if (!visited[i]) {
                    visited[i] = true;
                    q.push(i);
                }
            }
        }
    }
};

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

    Graph g(vertices);

    int edges;
    cout << "Enter the number of edges: ";
    cin >> edges;

    cout << "Enter edges in format (u v):" << endl;
    for (int i = 0; i < edges; ++i) {
        int u, v;
        cin >> u >> v;
        g.addEdge(u, v);
    }

    int start_vertex;
    cout << "Enter the starting vertex for traversal: ";
    cin >> start_vertex;

    cout << "Depth First Search (DFS) starting from vertex " <<
start_vertex << ": ";
    g.DFS(start_vertex);
    cout << endl;

    cout << "Breadth First Search (BFS) starting from vertex " << start_vertex << ": ";
    g.BFS(start_vertex);
    cout << endl;

    return 0;
}
```

Output: (screenshot)

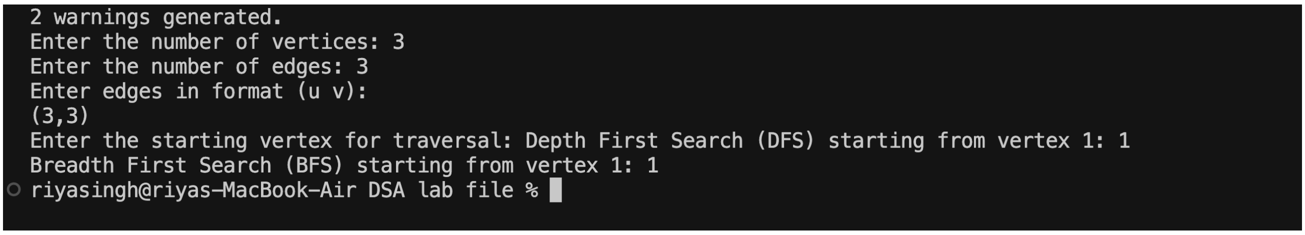


```
2 warnings generated.
Enter the number of vertices: 2
Enter the number of edges: 2
Enter edges in format (u v):
(2,2
Enter the starting vertex for traversal: Depth First Search (DFS) starting from vertex 1: 1
Breadth First Search (BFS) starting from vertex 1: 1
○ riyasingh@riyas-MacBook-Air DSA lab file % █
```

Test Case: Any two (screenshot)



```
2 warnings generated.
Enter the number of vertices: 4
Enter the number of edges: 4
Enter edges in format (u v):
(3,4)
Enter the starting vertex for traversal: Depth First Search (DFS) starting from vertex 1: 1
Breadth First Search (BFS) starting from vertex 1: 1
○ riyasingh@riyas-MacBook-Air DSA lab file % █
```



```
2 warnings generated.
Enter the number of vertices: 3
Enter the number of edges: 3
Enter edges in format (u v):
(3,3)
Enter the starting vertex for traversal: Depth First Search (DFS) starting from vertex 1: 1
Breadth First Search (BFS) starting from vertex 1: 1
○ riyasingh@riyas-MacBook-Air DSA lab file % █
```

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

Name of Student: Riya Singh

Roll Number: 26

Experiment No: 13

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

Theory:

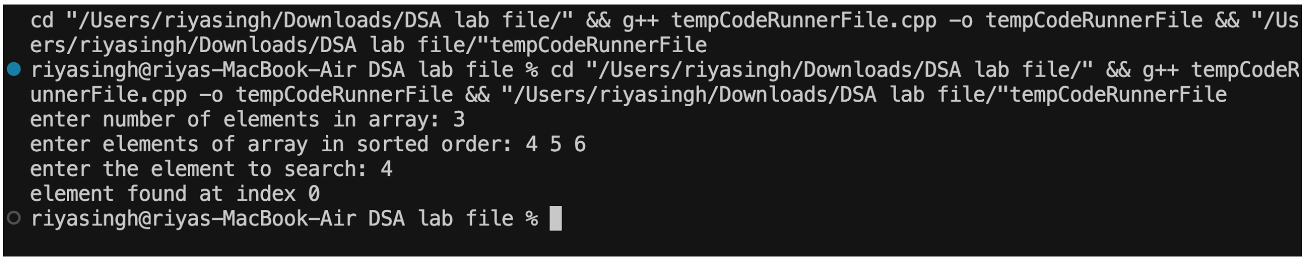
Binary Search is a searching algorithm which is used in a sorted array by repeatedly dividing the search interval in half. The idea of binary search is to use the information that the array is sorted and reduce the time complexity to O(log N).

Code:

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

int bst(int arr[], int left, int right, int target)
{
    while (left <= right)
    {
        int mid = left + (right - left) / 2;

        if (arr[mid] == target)
            return mid;

        if (arr[mid] < target)
            left = mid + 1;

        else
            right = mid - 1;
    }

    return -1;
}

int main()
{
    int n;
    cout << "enter number of elements in array: ";
    cin >> n;

    int arr[n];
    cout << "enter elements of array in sorted order: ";

    for (int i = 0; i < n; ++i)
        cin >> arr[i];
```

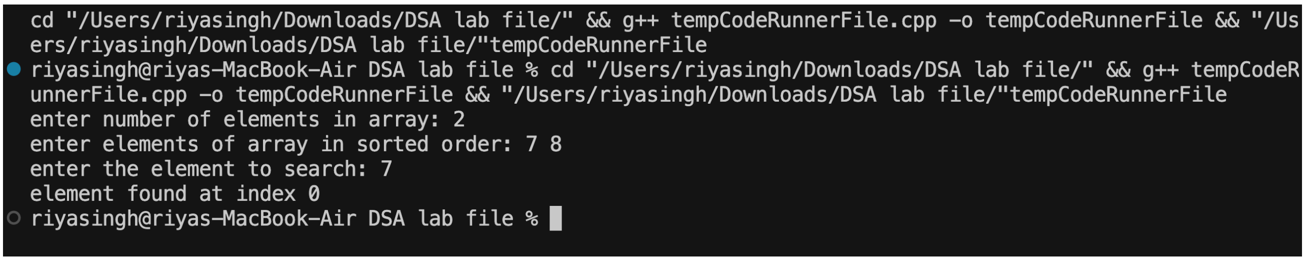
```cpp
    int target;
    cout << "enter the element to search: ";
    cin >> target;

    int index = bst(arr, 0, n - 1, target);

    if (index != -1)
        cout << "element found at index " << index << endl;
    else
        cout << "element not found" << endl;

    return 0;
}
```

Output: (screenshot)

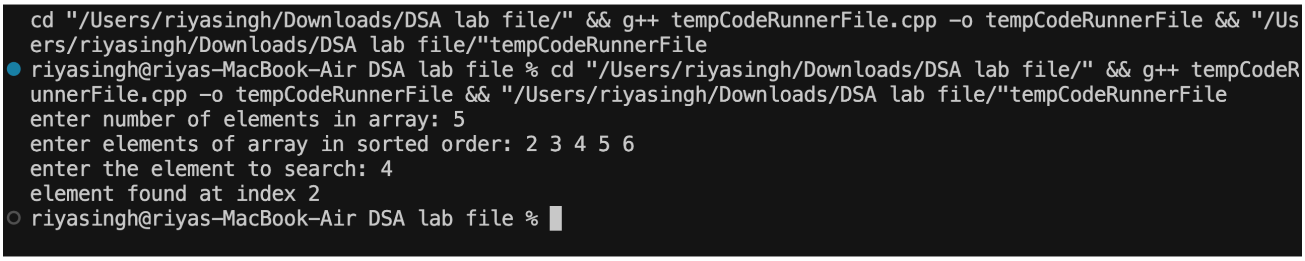


```
cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/Users/riyasingh/Downloads/DSA lab file/"tempCodeRunnerFile
● riyasingh@riyas-MacBook-Air DSA lab file % cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/Users/riyasingh/Downloads/DSA lab file/"tempCodeRunnerFile
enter number of elements in array: 3
enter elements of array in sorted order: 4 5 6
enter the element to search: 4
element found at index 0
○ riyasingh@riyas-MacBook-Air DSA lab file % █
```

Test Case: Any two (screenshot)



```
cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/Users/riyasingh/Downloads/DSA lab file/"tempCodeRunnerFile
● riyasingh@riyas-MacBook-Air DSA lab file % cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/Users/riyasingh/Downloads/DSA lab file/"tempCodeRunnerFile
enter number of elements in array: 2
enter elements of array in sorted order: 7 8
enter the element to search: 7
element found at index 0
○ riyasingh@riyas-MacBook-Air DSA lab file % █
```



```
cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/Users/riyasingh/Downloads/DSA lab file/"tempCodeRunnerFile
● riyasingh@riyas-MacBook-Air DSA lab file % cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/Users/riyasingh/Downloads/DSA lab file/"tempCodeRunnerFile
enter number of elements in array: 5
enter elements of array in sorted order: 2 3 4 5 6
enter the element to search: 4
element found at index 2
○ riyasingh@riyas-MacBook-Air DSA lab file % █
```

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

Name of Student: Riya Singh

Roll Number: 26

Experiment No: 14

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

Theory:

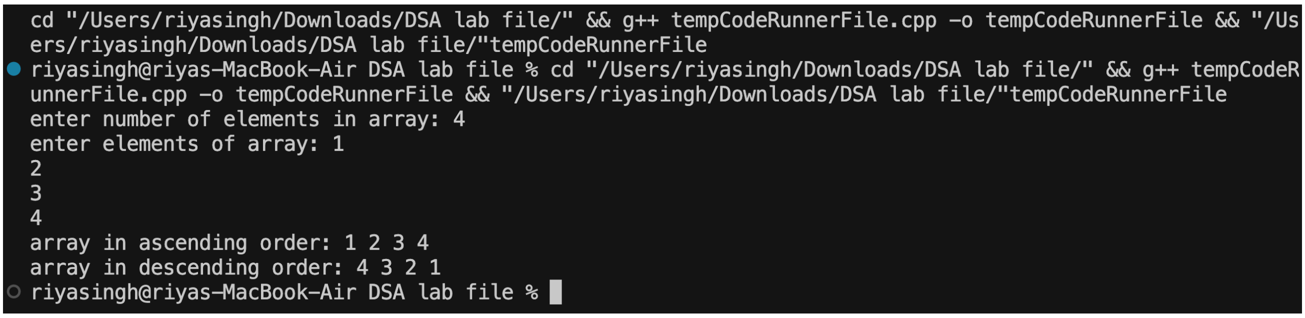
In Bubble Sort algorithm, we traverse from left and compare adjacent elements and the higher one is placed at right side. In this way, the largest element is moved to the rightmost end at first. This process is then continued to find the second largest and place it and so on until the data is sorted.

Code:

```
// 14. Implement Bubble sort algorithm to sort elements of
// an array in ascending and descending order.
#include <iostream>
using namespace std;

void ascendingsort(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])
            {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

void descendingsort(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])
```

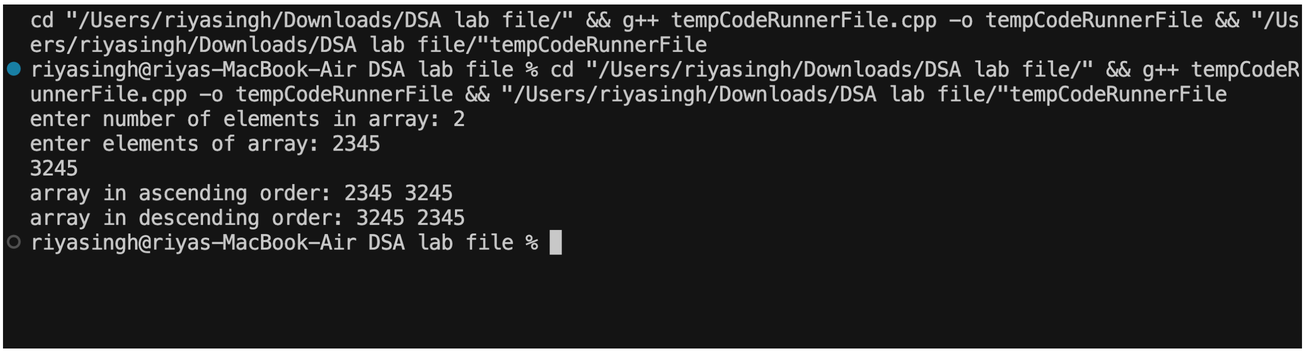
```cpp
            {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

int main()
{
    int n;
    cout << "enter number of elements in array: ";
    cin >> n;

    int arr[n];
    cout << "enter elements of array: ";

    for (int i = 0; i < n; ++i)
        cin >> arr[i];

    ascendingsort(arr, n);
    cout << "array in ascending order: ";

    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";
    cout << endl;

    descendingsort(arr, n);
    cout << "array in descending order: ";

    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";
    cout << endl;

    return 0;
}
```

Output: (screenshot)

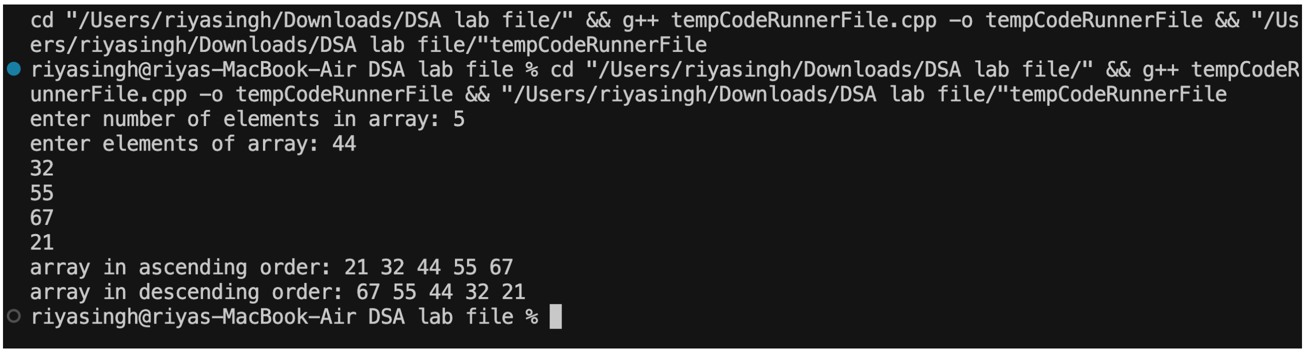


```
cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/Us
ers/riyasingh/Downloads/DSA lab file/"tempCodeRunnerFile
riyasingh@riyas-MacBook-Air DSA lab file % cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ tempCodeR
unnerFile.cpp -o tempCodeRunnerFile && "/Users/riyasingh/Downloads/DSA lab file/"tempCodeRunnerFile
enter number of elements in array: 4
enter elements of array: 1
2
3
4
array in ascending order: 1 2 3 4
array in descending order: 4 3 2 1
riyasingh@riyas-MacBook-Air DSA lab file %
```

Test Case: Any two (screenshot)



```
cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/Us
ers/riyasingh/Downloads/DSA lab file/"tempCodeRunnerFile
● riyasingh@riyas-MacBook-Air DSA lab file % cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ tempCodeR
unnerFile.cpp -o tempCodeRunnerFile && "/Users/riyasingh/Downloads/DSA lab file/"tempCodeRunnerFile
enter number of elements in array: 2
enter elements of array: 2345
3245
array in ascending order: 2345 3245
array in descending order: 3245 2345
○ riyasingh@riyas-MacBook-Air DSA lab file % █
```



```
cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/Us
ers/riyasingh/Downloads/DSA lab file/"tempCodeRunnerFile
● riyasingh@riyas-MacBook-Air DSA lab file % cd "/Users/riyasingh/Downloads/DSA lab file/" && g++ tempCodeR
unnerFile.cpp -o tempCodeRunnerFile && "/Users/riyasingh/Downloads/DSA lab file/"tempCodeRunnerFile
enter number of elements in array: 5
enter elements of array: 44
32
55
67
21
array in ascending order: 21 32 44 55 67
array in descending order: 67 55 44 32 21
○ riyasingh@riyas-MacBook-Air DSA lab file % █
```

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