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

DATA STRUCTURES & ALGORITHMS
PROGRAMMING LAB

Prepared by:

Name of Student - Anusri Karmokar

Roll No: (150096723003)

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. Anusri Karmokar Roll No. 150096723003 Semester 2 of B.Tech Computer Science & Engineering, ITM Skills University, Kharghar, Navi Mumbai , has completed the term work satisfactorily in subject Data Structures & Algorithms 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: Anusri Karmokar

Roll Number: 150096723003

Experiment No: 1

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

Theory:

This code defines a class Array to handle various operations on an array such as initialization, traversal, insertion, deletion, searching, sorting, and reversal. The main function prompts the user to input the capacity of the array and its elements, then provides a menu-driven interface to perform different operations.

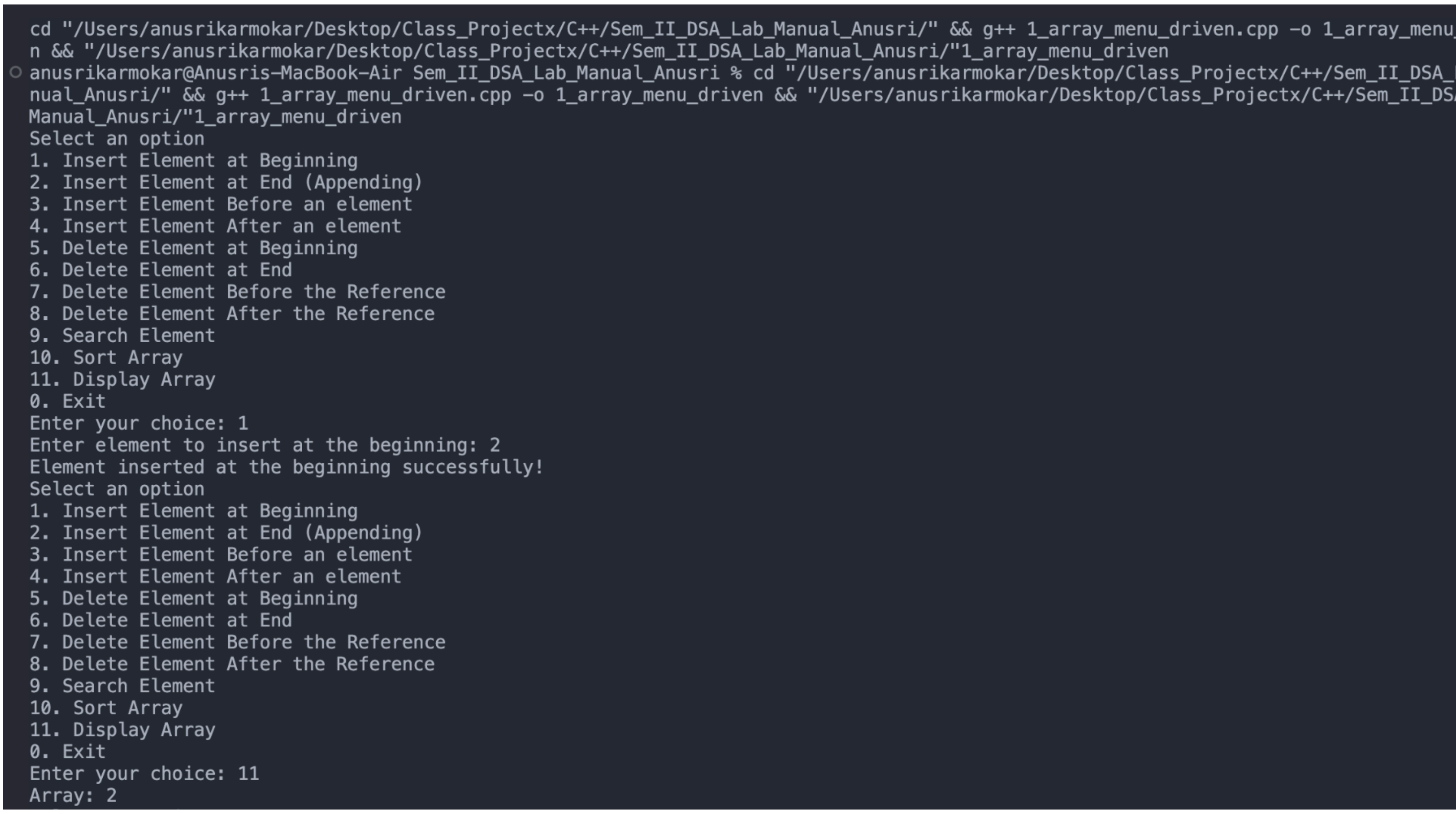
Code:

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

void displayMenu() {
 cout << "Select an option" << endl;
 cout << "1. Insert Element at Beginning" << endl;
 cout << "2. Insert Element at End (Appending)" << endl;
 cout << "3. Insert Element Before an element" << endl;
 cout << "4. Insert Element After an element" << endl;
 cout << "5. Delete Element at Beginning" << endl;
 cout << "6. Delete Element at End" << endl;
 cout << "7. Delete Element Before the Reference" << endl;
 cout << "8. Delete Element After the Reference" << endl;
 cout << "9. Search Element" << endl;
 cout << "10. Sort Array" << endl;
 cout << "11. Display Array" << endl;
 cout << "0. Exit" << endl;
}

void insertElementAtBeginning(int arr[], int &size, int element) {
 if (size == 0) {
   arr[size++] = element;
 } else {
   for (int i = size; i > 0; i--) {
     arr[i] = arr[i - 1];
   }
   arr[0] = element;
   size++;
 }
 cout << "Element inserted at the beginning successfully!" << endl;
}

void insertElementAtEnd(int arr[], int &size, int element) {
 arr[size++] = element;
 cout << "Element inserted at the end successfully!" << endl;
}
```

```
void insertElementBeforeReference(int arr[], int &size, int element,
 int reference) {
 int index = -1;
 for (int i = 0; i < size; i++) {
   if (arr[i] == reference) {
     index = i;
     break;
   }
 }
 if (index != -1) {
   for (int i = size; i > index; i--) {
     arr[i] = arr[i - 1];
   }
   arr[index] = element;
   size++;
   cout << "Element inserted before the reference successfully!" << endl;
 } else {
   cout << "Reference element not found!" << endl;
 }
}

void insertElementAfterReference(int arr[], int &size, int element,
                                int reference) {
 int index = -1;
 for (int i = 0; i < size; i++) {
   if (arr[i] == reference) {
     index = i;
     break;
   }
 }
 if (index != -1) {
   for (int i = size; i > index + 1; i--) {
     arr[i] = arr[i - 1];
   }
   arr[index + 1] = element;
   size++;
   cout << "Element inserted after the reference successfully!" << endl;
 } else {
   cout << "Reference element not found!" << endl;
 }
}

void deleteElementAtBeginning(int arr[], int &size) {
 if (size == 0) {
   cout << "Array is empty, no elements to delete!" << endl;
   return;
 }
 for (int i = 0; i < size - 1; i++) {
   arr[i] = arr[i + 1];
 }
 size--;
 cout << "Element at the beginning deleted successfully!" << endl;
}

void deleteElementAtEnd(int arr[], int &size) {
 if (size == 0) {
   cout << "Array is empty, no elements to delete!" << endl;
   return;
 }
 size--;
 cout << "Element at the end deleted successfully!" << endl;
}

void deleteElementBeforeReference(int arr[], int &size, int reference) {
 int index = -1;
 for (int i = 0; i < size; i++) {
   if (arr[i] == reference) {
     index = i - 1;
     break;
   }
 }
 if (index != -1 && index < size) {
   deleteElementAtBeginning(arr + index, size);
 } else {
   cout << "Reference element not found or it is the first element!" << endl;
 }
}

void deleteElementAfterReference(int arr[], int &size, int reference) {
 int index = -1;
 for (int i = 0; i < size; i++) {
   if (arr[i] == reference) {
     index = i + 1;
     break;
   }
 }
 if (index != -1 && index < size) {
   deleteElementAtBeginning(arr + index - 1, size);
 } else {
   cout << "Reference element not found or it is the last element!" << endl;
 }
}

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

int searchElement(const int arr[], int size, int element) {
 for (int i = 0; i < size; i++) {
   if (arr[i] == element)
     return i;
 }
 return -1;
}

void sortArray(int arr[], int size) {
 for (int i = 0; i < size - 1; i++) {
   for (int j = 0; j < size - i - 1; j++) {
     if (arr[j] > arr[j + 1]) {
       int temp = arr[j];
       arr[j] = arr[j + 1];
       arr[j + 1] = temp;
     }
   }
 }
 cout << "Array sorted successfully!" << endl;
}

int main() {
 const int MAX_SIZE = 100;
 int arr[MAX_SIZE];
 int size = 0;

 int choice;
 do {
   displayMenu();
   cout << "Enter your choice: ";
```

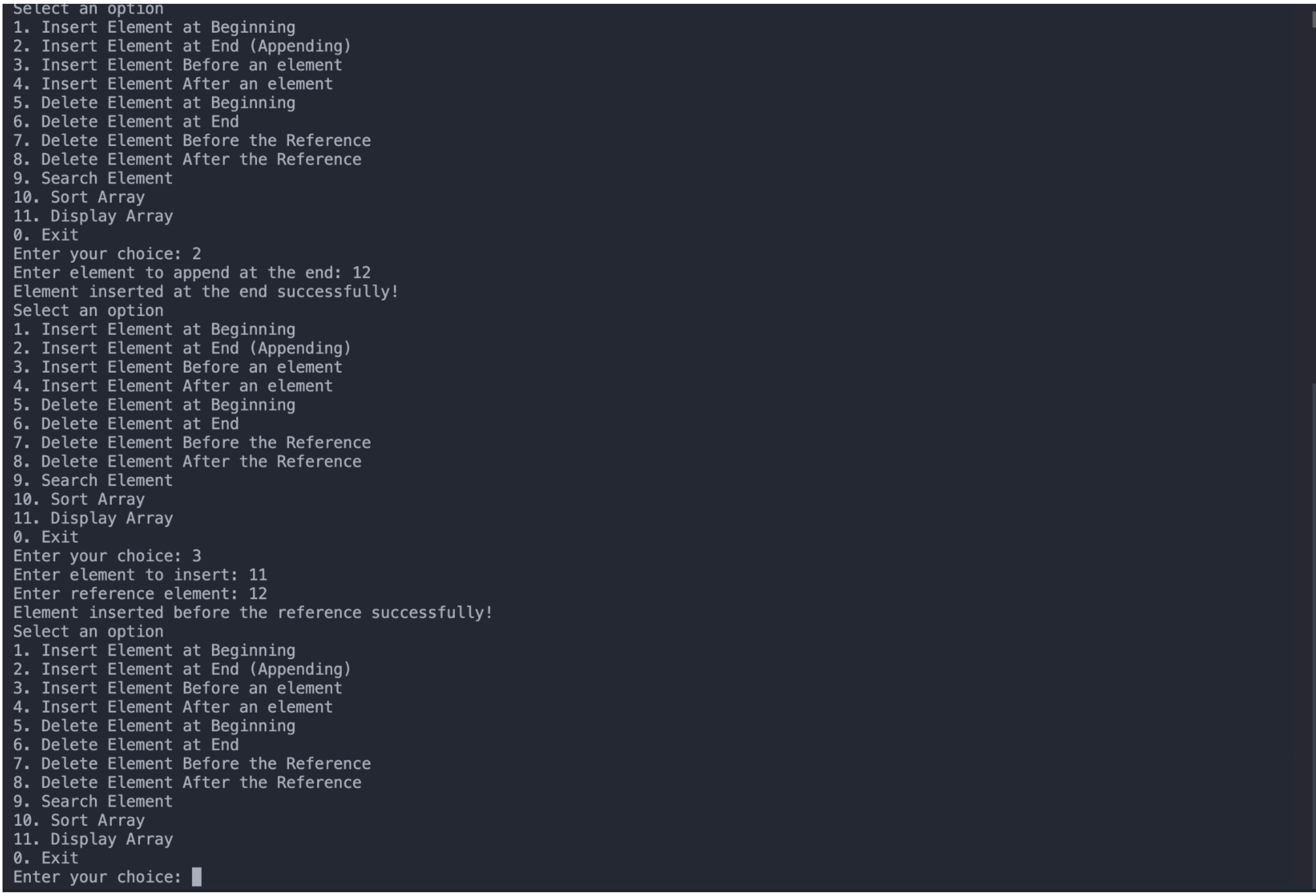
```
    cin >> choice;

    switch (choice) {
    case 1: {
      int element;
      cout << "Enter element to insert at the beginning: ";
      cin >> element;
      insertElementAtBeginning(arr, size, element);
      break;
    }
    case 2: {
      int element;
      cout << "Enter element to append at the end: ";
      cin >> element;
      insertElementAtEnd(arr, size, element);
      break;
    }
    case 3: {
      int element, reference;
      cout << "Enter element to insert: ";
      cin >> element;
      cout << "Enter reference element: ";
      cin >> reference;
      insertElementBeforeReference(arr, size, element, reference);
      break;
    }
    case 4: {
      int element, reference;
      cout << "Enter element to insert: ";
      cin >> element;
      cout << "Enter reference element: ";
      cin >> reference;
      insertElementAfterReference(arr, size, element, reference);
      break;
    }
    case 5: {
      deleteElementAtBeginning(arr, size);
      break;
    }
    case 6: {
      deleteElementAtEnd(arr, size);
      break;
    }
    case 7: {
      int reference;
      cout << "Enter reference element: ";
      cin >> reference;
      deleteElementBeforeReference(arr, size, reference);
      break;
    }
    case 8: {
      int reference;
      cout << "Enter reference element: ";
      cin >> reference;
      deleteElementAfterReference(arr, size, reference);
      break;
    }
    case 9: {
      int element;
      cout << "Enter element to search: ";
      cin >> element;
      int index = searchElement(arr, size, element);
      if (index != -1)
        cout << "Element found at index " << index << endl;
      else
        cout << "Element not found in the array!" << endl;
      break;
    }
    case 10: {
      sortArray(arr, size);
      break;
    }
    case 11: {
      displayArray(arr, size);
      break;
    }
    case 0: {
      cout << "Exiting..." << endl;
      break;
    }
    default:
      cout << "Invalid choice!" << endl;
    }
  } while (choice);

  return 0;
 }
```

Output: (screenshot)



```
cd "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DSA_Lab_Manual_Anusri/" && g++ 1_array_menu_driven.cpp -o 1_array_menu
n && "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DSA_Lab_Manual_Anusri/"1_array_menu_driven
anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri % cd "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DSA_
nual_Anusri/" && g++ 1_array_menu_driven.cpp -o 1_array_menu_driven && "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DS
Manual_Anusri/"1_array_menu_driven
Select an option
1. Insert Element at Beginning
2. Insert Element at End (Appending)
3. Insert Element Before an element
4. Insert Element After an element
5. Delete Element at Beginning
6. Delete Element at End
7. Delete Element Before the Reference
8. Delete Element After the Reference
9. Search Element
10. Sort Array
11. Display Array
0. Exit
Enter your choice: 1
Enter element to insert at the beginning: 2
Element inserted at the beginning successfully!
Select an option
1. Insert Element at Beginning
2. Insert Element at End (Appending)
3. Insert Element Before an element
4. Insert Element After an element
5. Delete Element at Beginning
6. Delete Element at End
7. Delete Element Before the Reference
8. Delete Element After the Reference
9. Search Element
10. Sort Array
11. Display Array
0. Exit
Enter your choice: 11
Array: 2
```

Test Case: Any two (screenshot)



```
Select an option
1. Insert Element at Beginning
2. Insert Element at End (Appending)
3. Insert Element Before an element
4. Insert Element After an element
5. Delete Element at Beginning
6. Delete Element at End
7. Delete Element Before the Reference
8. Delete Element After the Reference
9. Search Element
10. Sort Array
11. Display Array
0. Exit
Enter your choice: 2
Enter element to append at the end: 12
Element inserted at the end successfully!
Select an option
1. Insert Element at Beginning
2. Insert Element at End (Appending)
3. Insert Element Before an element
4. Insert Element After an element
5. Delete Element at Beginning
6. Delete Element at End
7. Delete Element Before the Reference
8. Delete Element After the Reference
9. Search Element
10. Sort Array
11. Display Array
0. Exit
Enter your choice: 3
Enter element to insert: 11
Enter reference element: 12
Element inserted before the reference successfully!
Select an option
1. Insert Element at Beginning
2. Insert Element at End (Appending)
3. Insert Element Before an element
4. Insert Element After an element
5. Delete Element at Beginning
6. Delete Element at End
7. Delete Element Before the Reference
8. Delete Element After the Reference
9. Search Element
10. Sort Array
11. Display Array
0. Exit
Enter your choice: 
```

Conclusion:

The code provides a comprehensive implementation for array manipulation, offering functionalities like insertion, deletion, searching, sorting, and reversal. It offers user-friendly interaction through a menu-driven interface, making it easy to use for array operations.

Name of Student: Anusri Karmokar

Roll Number: 150096723003

Experiment No: 2

Title: Implement Stack ADT using array.

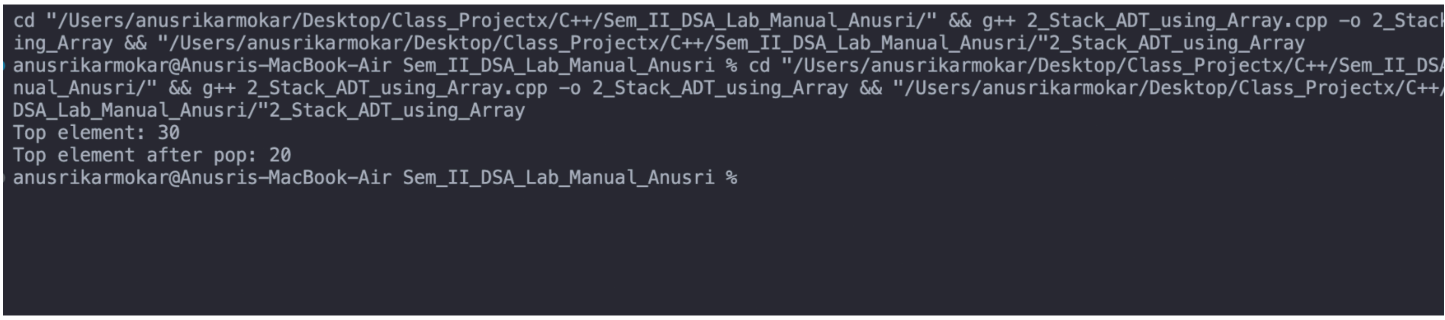
Theory:

This code implements a stack using a class Stack with functionalities like initialization, checking if the stack is empty or full, pushing elements onto the stack, popping elements from the stack, and peeking at the top element. The main function provides menu-driven interface for users to interact with the stack.

Code:

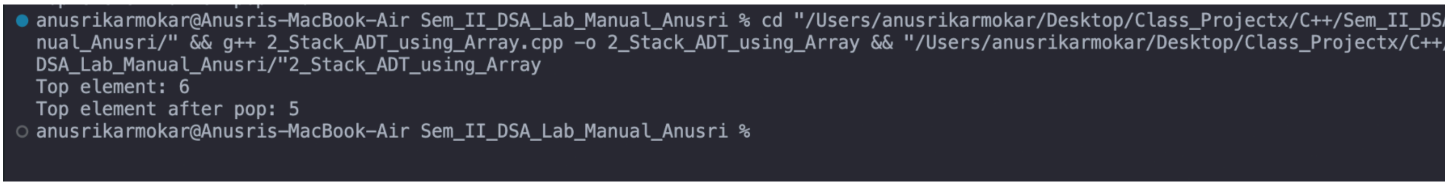
```cpp
#include <iostream>
using namespace std;

const int MAX_SIZE = 100;

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

public:
    Stack() {
        top = -1;
    }


    void push(int val) {
        if (top == MAX_SIZE - 1) {
            cout << "Stack Overflow\n";
            return;
        }
        arr[++top] = val;
    }


    void pop() {
        if (top == -1) {
            cout << "Stack Underflow\n";
            return;
        }
        top--;
    }


    int peek() {
        if (top == -1) {
            cout << "Stack is empty\n";
            return -1;
        }
        return arr[top];

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

    }

};

int main() {
    Stack stack;

    stack.push(10);
    stack.push(20);
    stack.push(30);

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

    stack.pop();

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

    return 0;
}
```

Output: (screenshot)



```
cd "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DSA_Lab_Manual_Anusri/" && g++ 2_Stack_ADT_using_Array.cpp -o 2_Stack
ing_Array && "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DSA_Lab_Manual_Anusri/"2_Stack_ADT_using_Array
anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri % cd "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DSA
nual_Anusri/" && g++ 2_Stack_ADT_using_Array.cpp -o 2_Stack_ADT_using_Array && "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/
DSA_Lab_Manual_Anusri/"2_Stack_ADT_using_Array
Top element: 30
Top element after pop: 20
anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri %
```

Test Case: Any two (screenshot)



```
● anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri % cd "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DS
  nual_Anusri/" && g++ 2_Stack_ADT_using_Array.cpp -o 2_Stack_ADT_using_Array && "/Users/anusrikarmokar/Desktop/Class_Projectx/C++
  DSA_Lab_Manual_Anusri/"2_Stack_ADT_using_Array
  Top element: 6
  Top element after pop: 5
○ anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri %
```

Conclusion:

The code offers a basic implementation of a stack data structure with essential operations like push, pop, and peek. allowing users to push elements onto the stack, pop elements from it, and view the top element.

Name of Student: Anusri Karmokar

Roll Number: 150096723003

Experiment No: 3

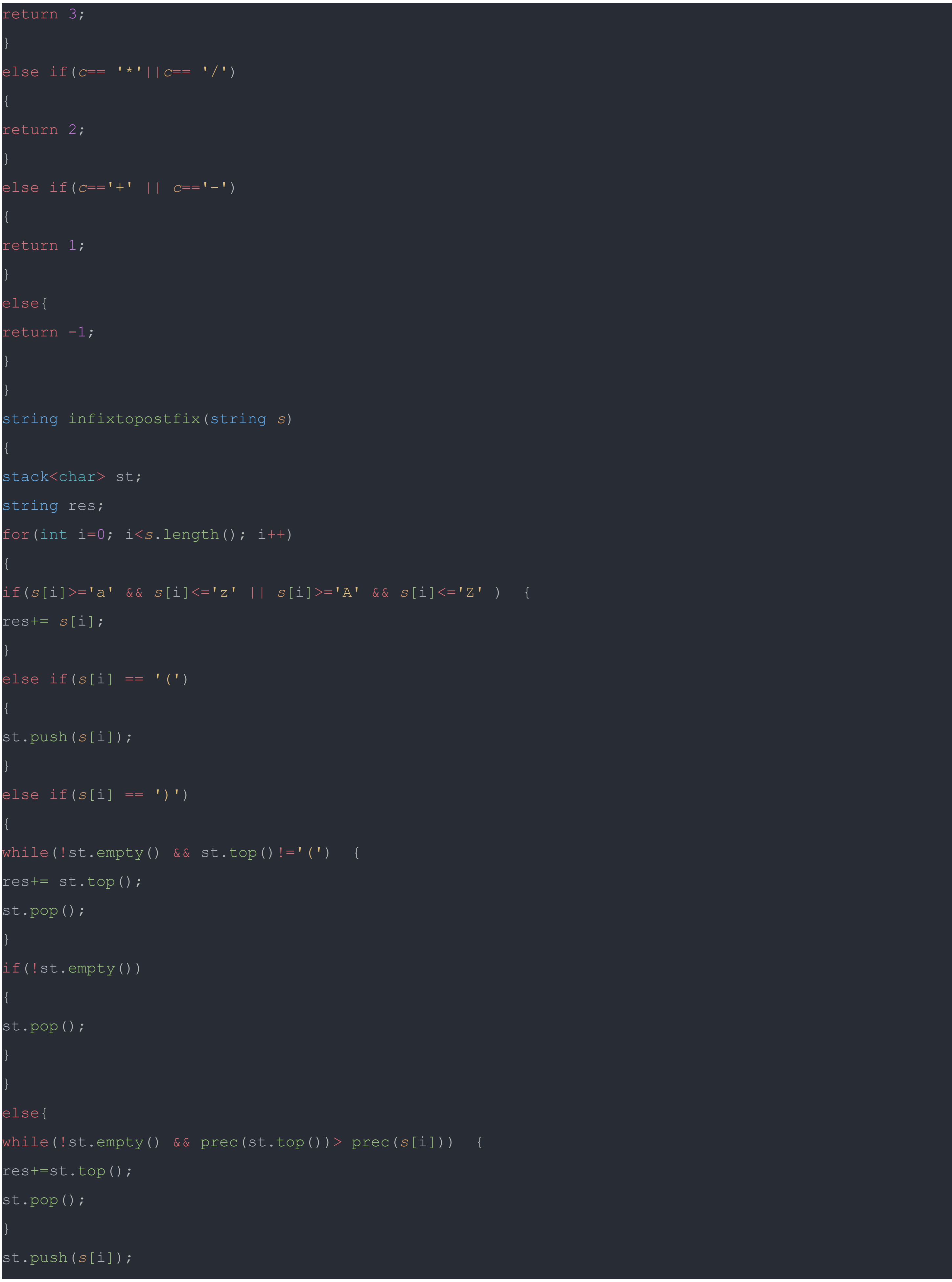
Title: Convert an Infix expression to Postfix expression using stack ADT.

Theory:

This code converts an infix expression to a postfix expression using a stack. It reads an infix expression from the user, iterates through each character, and based on the precedence of operators and parentheses, constructs the corresponding postfix expression.

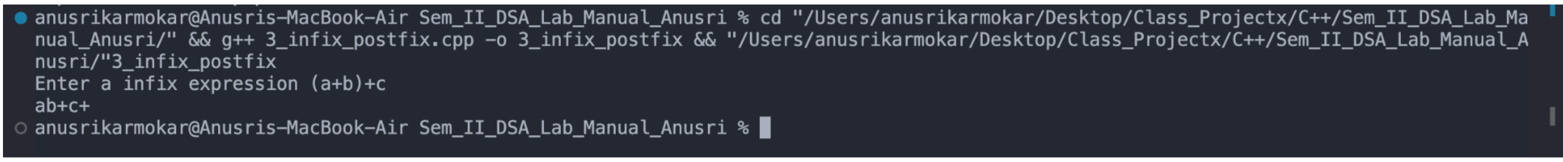
Code:

```
#include<iostream>
#include<stack>
using namespace std;
int prec(char c)
{
if(c=='^')
{
```



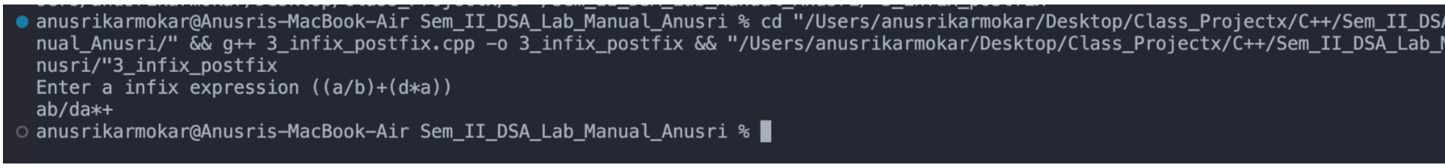
```
return 3;
}
else if(c== '*'||c== '/')
{
return 2;
}
else if(c=='+' || c=='-')
{
return 1;
}
else{
return -1;
}
}
string infixtopostfix(string s)
{
stack<char> st;
string res;
for(int i=0; i<s.length(); i++)
{
if(s[i]>='a' && s[i]<='z' || s[i]>='A' && s[i]<='Z' )  {
res+= s[i];
}
else if(s[i] == '(')
{
st.push(s[i]);
}
else if(s[i] == ')')
{
while(!st.empty() && st.top()!='(')  {
res+= st.top();
st.pop();
}
if(!st.empty())
{
st.pop();
}
}
else{
while(!st.empty() && prec(st.top())> prec(s[i]))  {
res+=st.top();
st.pop();
}
st.push(s[i]);
}
}
while(!st.empty())
{
res+=st.top();
st.pop();
}
return res;
}
int main()
{
string a;
cout<<"Enter a infix expression ";
cin>>a;
cout<<infixtopostfix(a)<<endl;
return 0;
}
```

Output: (screenshot)

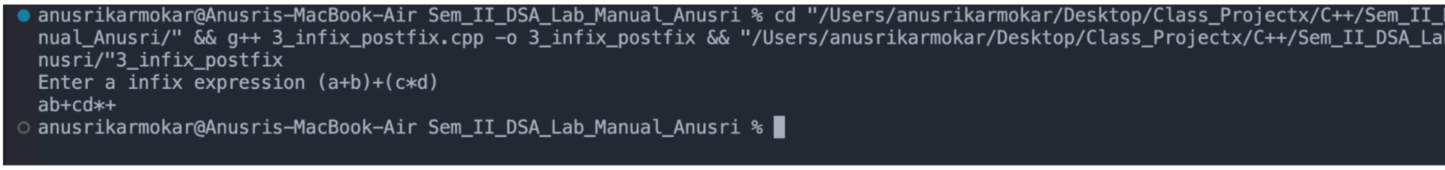


```
● anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri % cd "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DSA_Lab_Ma
  nual_Anusri/" && g++ 3_infix_postfix.cpp -o 3_infix_postfix && "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DSA_Lab_Manual_A
  nusri/"3_infix_postfix
  Enter a infix expression (a+b)+c
  ab+c+
○ anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri %
```

Test Case: Any two (screenshot)



```
● anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri % cd "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DS
  nual_Anusri/" && g++ 3_infix_postfix.cpp -o 3_infix_postfix && "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DSA_Lab_
  nusri/"3_infix_postfix
  Enter a infix expression ((a/b)+(d*a))
  ab/da*+
○ anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri %
```



```
● anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri % cd "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_
  nual_Anusri/" && g++ 3_infix_postfix.cpp -o 3_infix_postfix && "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DSA_La
  nusri/"3_infix_postfix
  Enter a infix expression (a+b)+(c*d)
  ab+cd*+
○ anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri %
```

Conclusion:

The code efficiently converts infix expressions to postfix expressions using a stack-based approach, handling operands, operators, and parentheses while maintaining operator precedence. It provides a straightforward implementation for converting expressions, useful in various parsing and evaluation

algorithms.

Name of Student: **Anusri Karmokar**

Roll Number: **150096723003**

Experiment No: 4

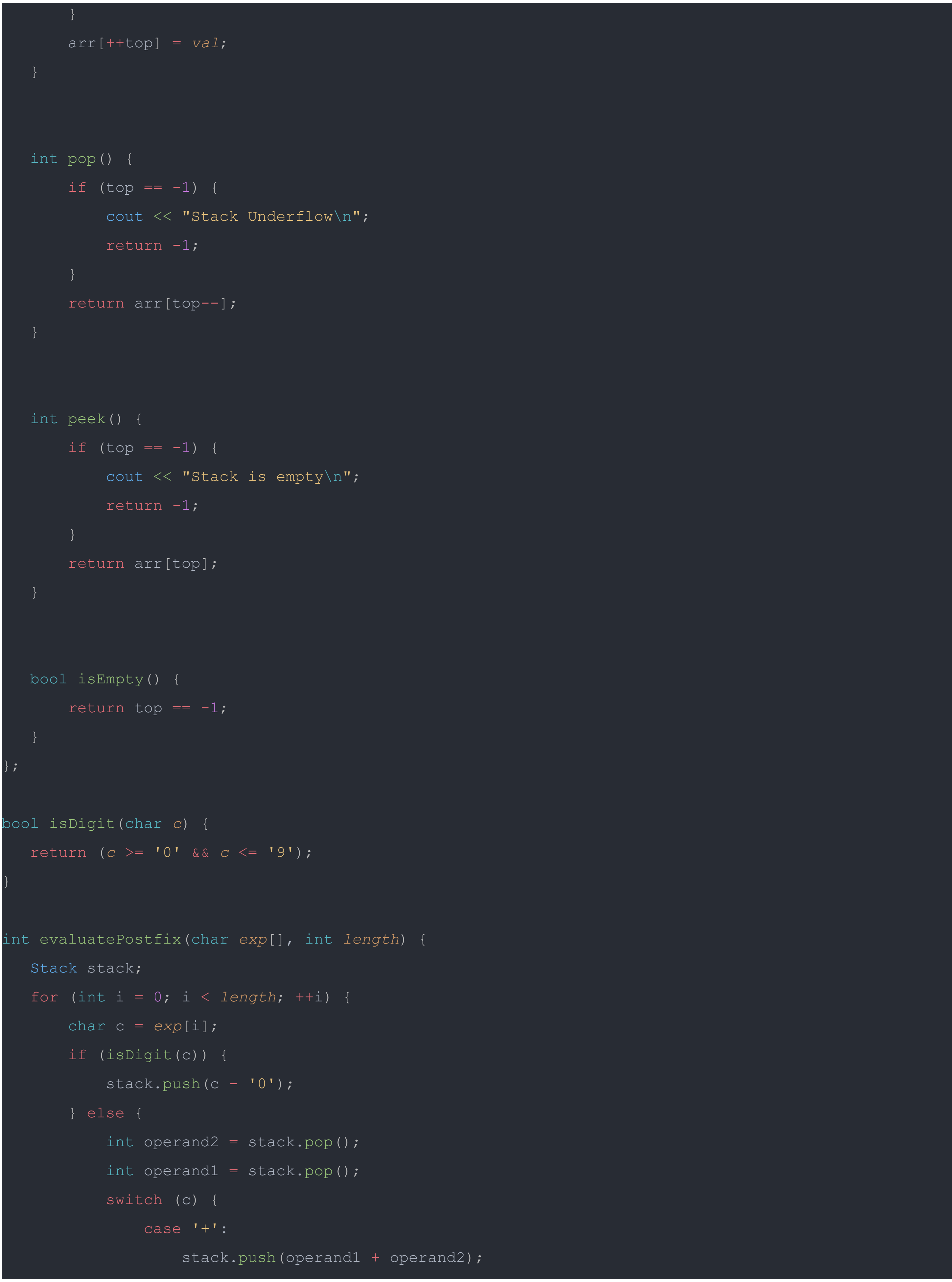
Title: Evaluate Postfix Expression using Stack ADT.

Theory:

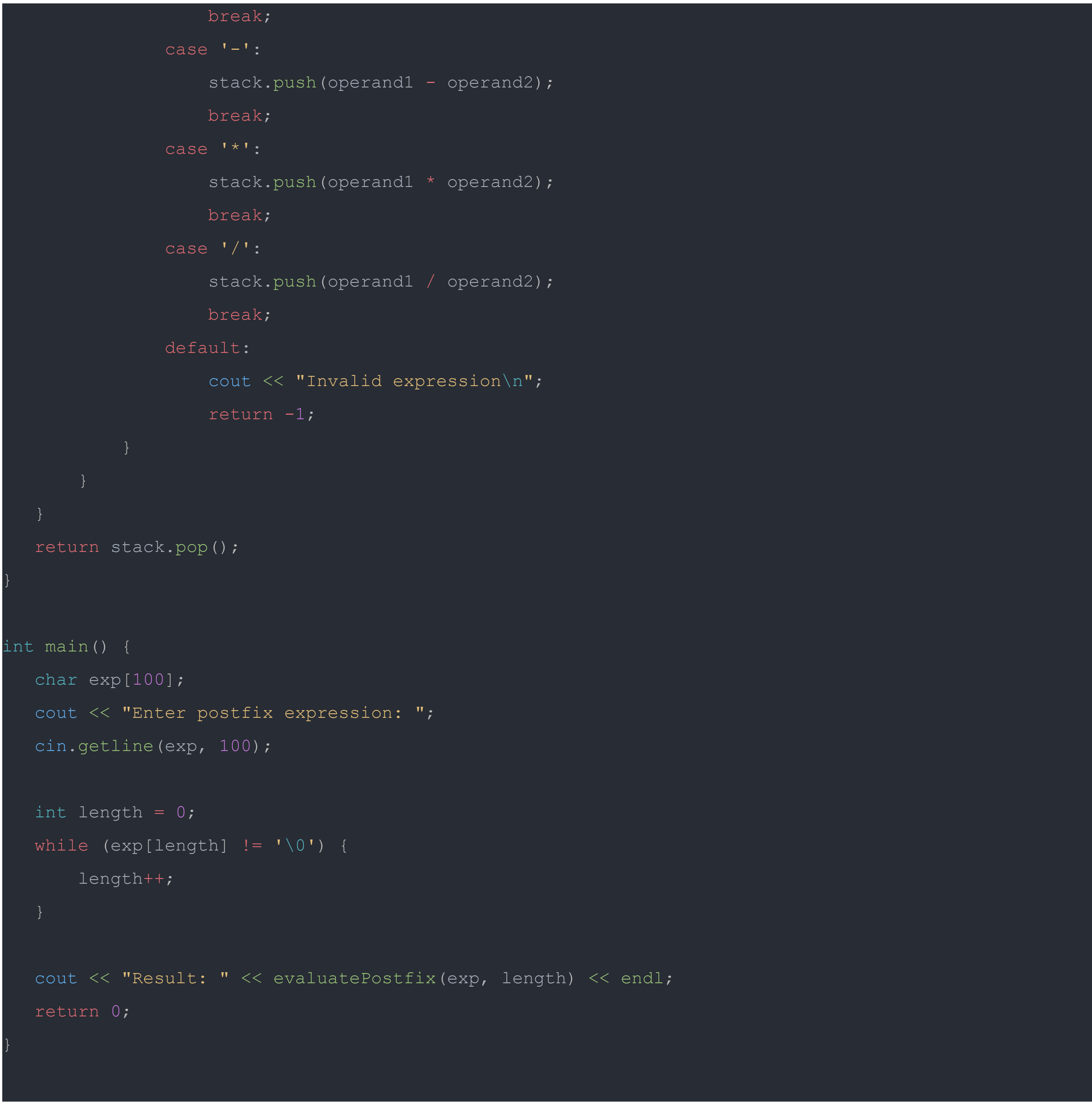
This code evaluates a postfix expression by iterating through each character of the expression and using a stack to perform the necessary arithmetic operations. It pushes operands onto the stack and when encountering an operator, it pops the required number of operands from the stack, performs the operation, and pushes the result back onto the stack.

Code:

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


class Stack {
private:
   int top;
   int arr[100];

public:
   Stack() {
       top = -1;
   }


   void push(int val) {
       if (top == 99) {
           cout << "Stack Overflow\n";
           return;
```

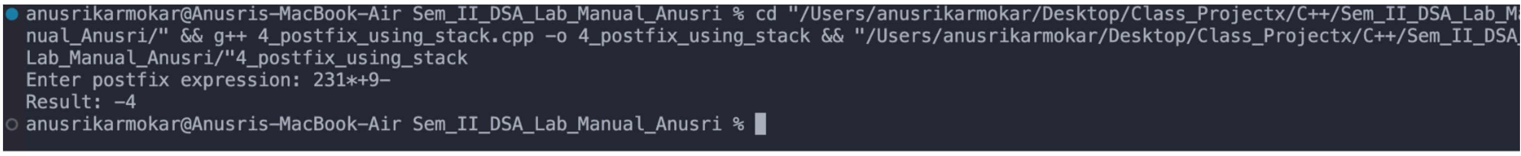


```
        }
        arr[++top] = val;
    }


    int pop() {
        if (top == -1) {
            cout << "Stack Underflow\n";
            return -1;
        }
        return arr[top--];
    }


    int peek() {
        if (top == -1) {
            cout << "Stack is empty\n";
            return -1;
        }
        return arr[top];
    }


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

bool isDigit(char c) {
    return (c >= '0' && c <= '9');
}

int evaluatePostfix(char exp[], int length) {
    Stack stack;
    for (int i = 0; i < length; ++i) {
        char c = exp[i];
        if (isDigit(c)) {
            stack.push(c - '0');
        } else {
            int operand2 = stack.pop();
            int operand1 = stack.pop();
            switch (c) {
                case '+':
                    stack.push(operand1 + operand2);
```



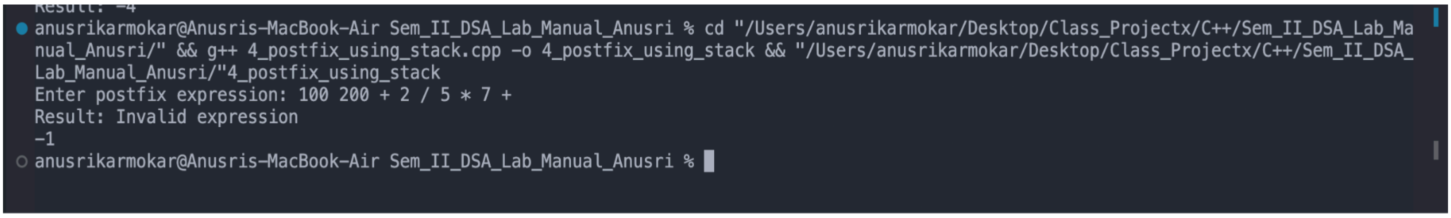
```
                    break;
                case '-':
                    stack.push(operand1 - operand2);
                    break;
                case '*':
                    stack.push(operand1 * operand2);
                    break;
                case '/':
                    stack.push(operand1 / operand2);
                    break;
                default:
                    cout << "Invalid expression\n";
                    return -1;
            }
        }
    }
    return stack.pop();
}

int main() {
    char exp[100];
    cout << "Enter postfix expression: ";
    cin.getline(exp, 100);

    int length = 0;
    while (exp[length] != '\0') {
        length++;
    }

    cout << "Result: " << evaluatePostfix(exp, length) << endl;
    return 0;
}
```

Output: (screenshot)



```
● anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri % cd "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DSA_Lab_Ma
  nual_Anusri/" && g++ 4_postfix_using_stack.cpp -o 4_postfix_using_stack && "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DSA_
  Lab_Manual_Anusri/"4_postfix_using_stack
  Enter postfix expression: 231**9-
  Result: -4
○ anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri % █
```

Test Case: Any two (screenshot)



```
Result: -4
anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri % cd "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DSA_Lab_Ma
nual_Anusri/" && g++ 4_postfix_using_stack.cpp -o 4_postfix_using_stack && "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DSA_
Lab_Manual_Anusri/"4_postfix_using_stack
Enter postfix expression: 100 200 + 2 / 5 * 7 +
Result: Invalid expression
-1
anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri % 
```

Conclusion:

The code efficiently evaluates postfix expressions using a stack-based approach, handling arithmetic operations such as addition, subtraction, multiplication, and division. It provides a straightforward implementation for expression evaluation, useful in various mathematical and computing applications.

Name of Student: Anusri Karmokar

Roll Number: 150096723003

Experiment No: 5

Title: Implement Linear Queue ADT using array.

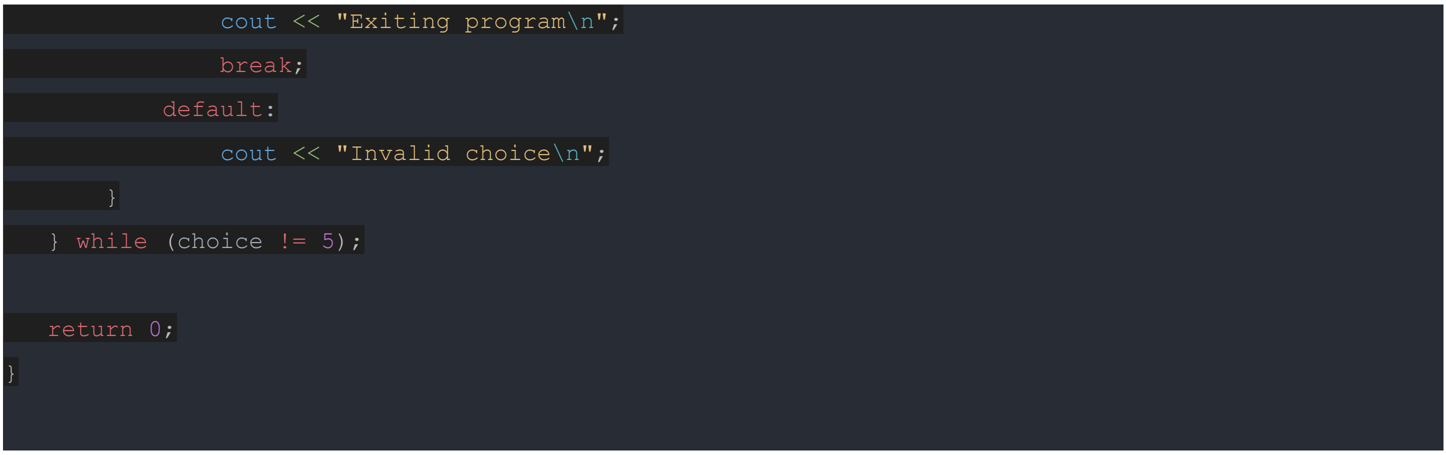
Theory:

This code implements a queue data structure using an array. It provides functionalities to enqueue elements into the queue, dequeue elements from the queue, and display the elements currently in the queue. The main function offers a menu-driven interface for users to interact with the queue.

Code:

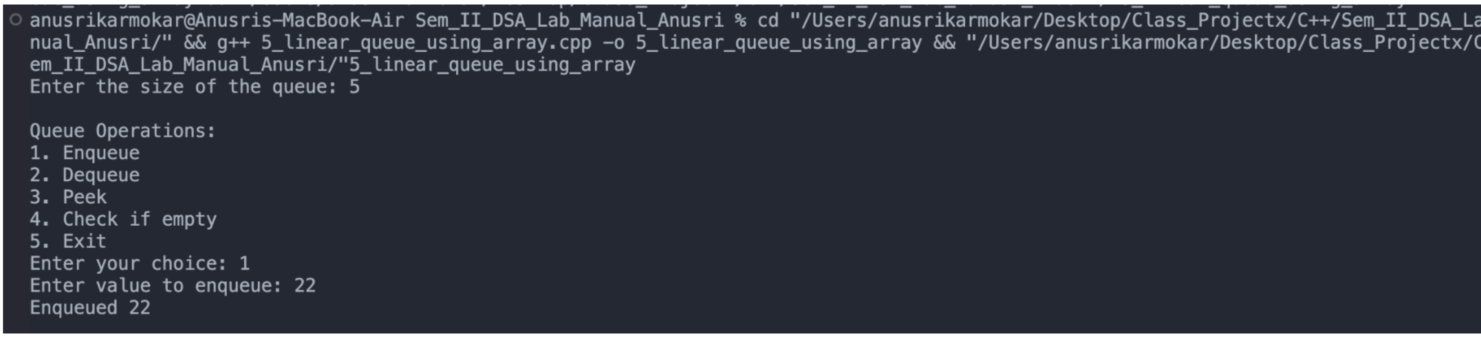
```
#include <iostream>

using namespace std;


class Queue {

private:
```

```cpp
    int front, rear;
    int capacity;
    int* arr;

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

    ~Queue() {
        delete[] arr;
    }

    void enqueue(int item) {
        if (rear == capacity - 1) {
            cout << "Queue Overflow\n";
            return;
        }
        if (front == -1)
            front = 0;
        arr[++rear] = item;
        cout << "Enqueued " << item << endl;
    }

    int dequeue() {
        if (front == -1 || front > rear) {
            cout << "Queue Underflow\n";
            return -1;
        }
        int item = arr[front++];
        cout << "Dequeued " << item << endl;
        return item;
    }

    int peek() {
        if (front == -1 || front > rear) {
            cout << "Queue is empty\n";
            return -1;
        }
        return arr[front];
    }
    bool isEmpty() {
        return front == -1 || front > rear;
    }
};

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

    Queue q(size);

    int choice, value;
    do {
        cout << "\nQueue Operations:\n";
        cout << "1. Enqueue\n";
        cout << "2. Dequeue\n";
        cout << "3. Peek\n";
        cout << "4. Check if empty\n";
        cout << "5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter value to enqueue: ";
                cin >> value;
                q.enqueue(value);
                break;
            case 2:
                q.dequeue();
                break;
            case 3:
                value = q.peek();
                if (value != -1)
                    cout << "Front element: " << value << endl;
                break;
            case 4:
                if (q.isEmpty())
                    cout << "Queue is empty\n";
                else
                    cout << "Queue is not empty\n";
                break;
            case 5:
```



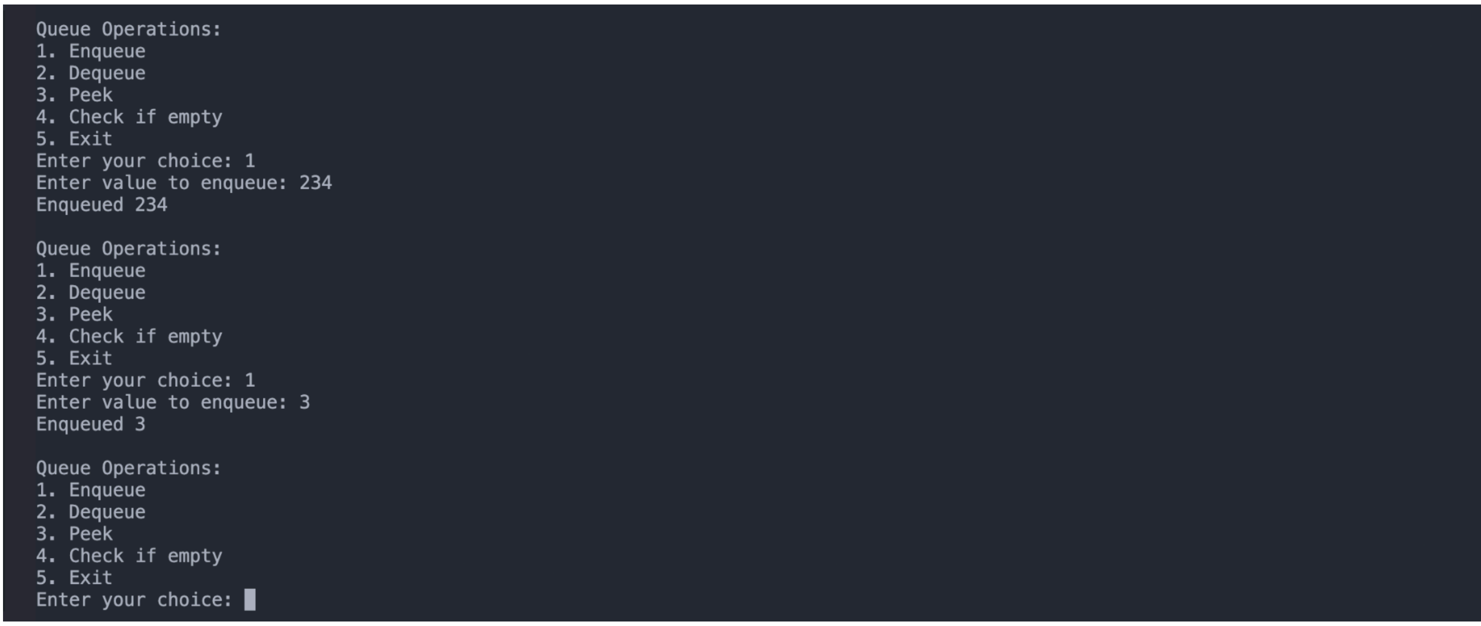
```
                cout << "Exiting program\n";
                break;
            default:
                cout << "Invalid choice\n";
        }
    } while (choice != 5);

    return 0;
}
```

Output: (screenshot)



```
anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri % cd "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DSA_La
nual_Anusri/" && g++ 5_linear_queue_using_array.cpp -o 5_linear_queue_using_array && "/Users/anusrikarmokar/Desktop/Class_Projectx/C
em_II_DSA_Lab_Manual_Anusri/"5_linear_queue_using_array
Enter the size of the queue: 5

Queue Operations:
1. Enqueue
2. Dequeue
3. Peek
4. Check if empty
5. Exit
Enter your choice: 1
Enter value to enqueue: 22
Enqueued 22
```

Test Case: Any two (screenshot)



```
Queue Operations:
1. Enqueue
2. Dequeue
3. Peek
4. Check if empty
5. Exit
Enter your choice: 1
Enter value to enqueue: 234
Enqueued 234

Queue Operations:
1. Enqueue
2. Dequeue
3. Peek
4. Check if empty
5. Exit
Enter your choice: 1
Enter value to enqueue: 3
Enqueued 3

Queue Operations:
1. Enqueue
2. Dequeue
3. Peek
4. Check if empty
5. Exit
Enter your choice: 
```

Conclusion:

The code offers a basic implementation of a queue using an array, providing essential operations such as enqueue, dequeue, and display. The queue, allows users to enqueue elements, dequeue elements, and view the elements currently in the queue.

Name of Student: **Anusri Karmokar**

Roll Number: **150096723003**

Experiment No: 6

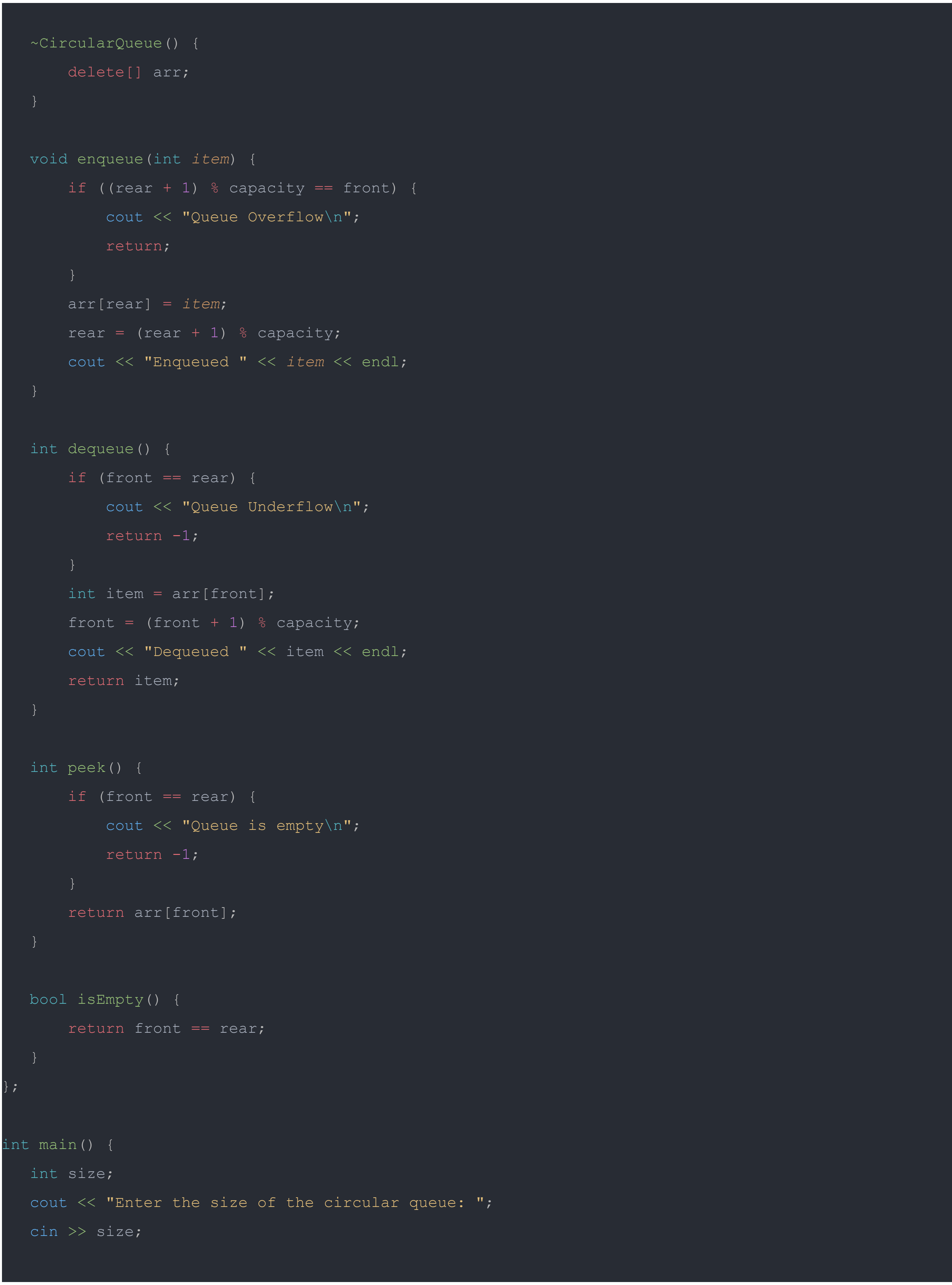
Title: Implement Circular Queue ADT using array.

Theory:

This code implements a queue data structure using an array with circular buffering to optimize space usage. It provides functionalities to enqueue elements into the queue, dequeue elements from the queue, and display the elements currently in the queue. The circular buffering technique ensures efficient utilization of the array.

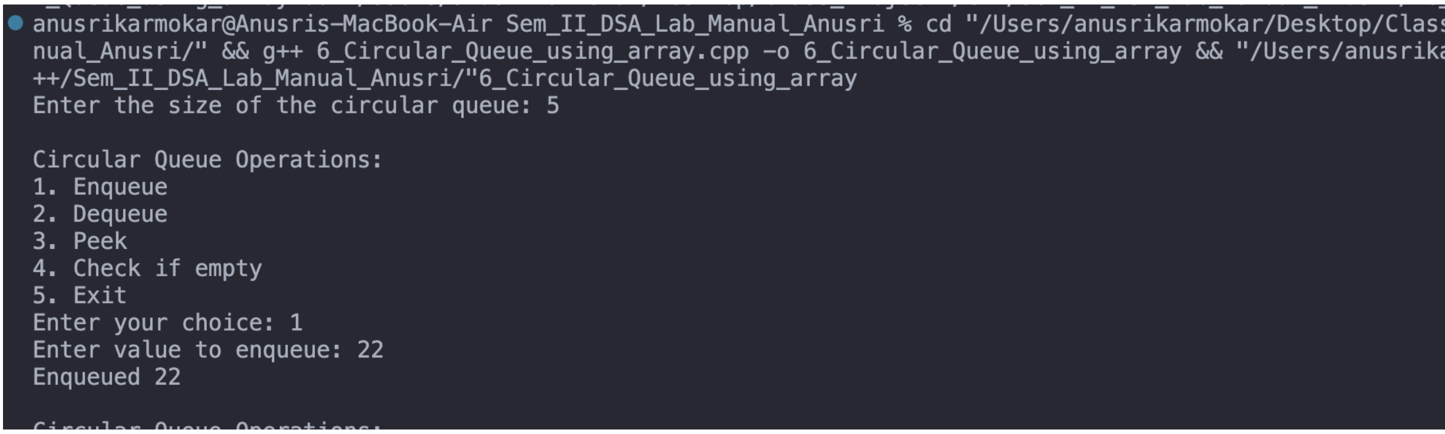
Code:

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


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

public:
   CircularQueue(int size) {
       capacity = size + 1;
       arr = new int[capacity];
       front = rear = 0;
   }
```



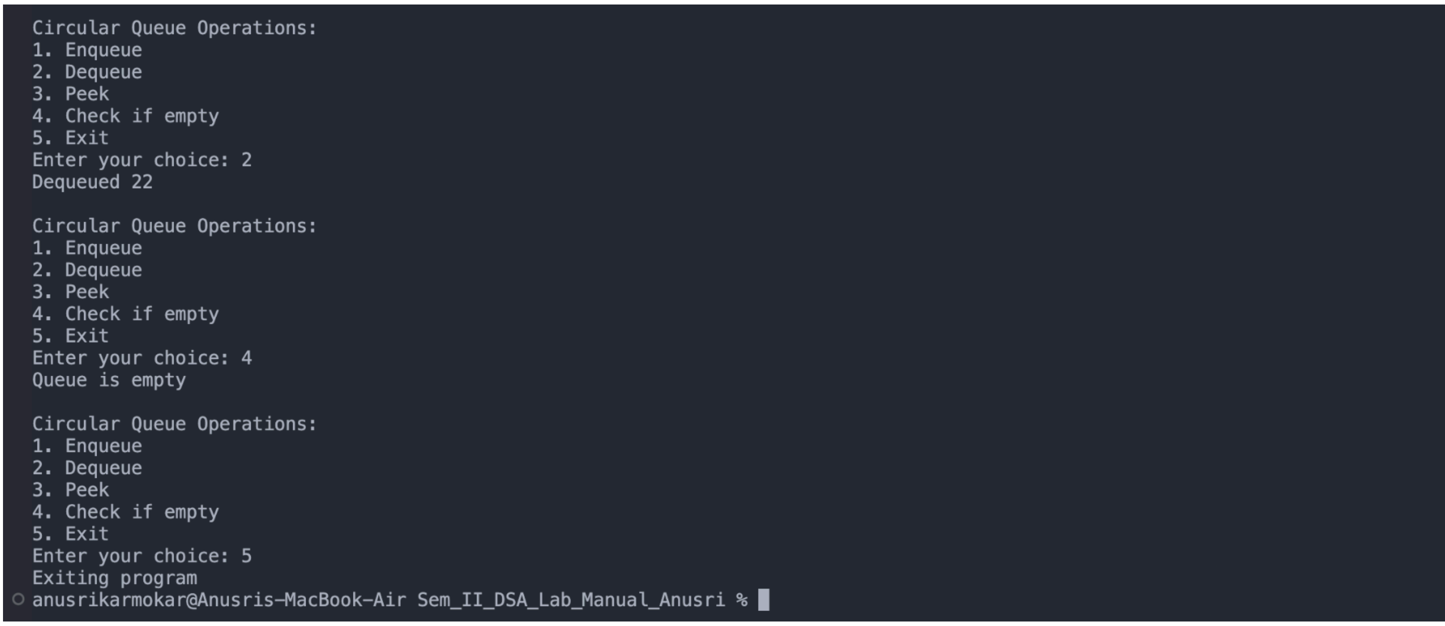
```cpp
    ~CircularQueue() {
        delete[] arr;
    }

    void enqueue(int item) {
        if ((rear + 1) % capacity == front) {
            cout << "Queue Overflow\n";
            return;
        }
        arr[rear] = item;
        rear = (rear + 1) % capacity;
        cout << "Enqueued " << item << endl;
    }

    int dequeue() {
        if (front == rear) {
            cout << "Queue Underflow\n";
            return -1;
        }
        int item = arr[front];
        front = (front + 1) % capacity;
        cout << "Dequeued " << item << endl;
        return item;
    }

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

    bool isEmpty() {
        return front == rear;
    }
};

int main() {
    int size;
    cout << "Enter the size of the circular queue: ";
    cin >> size;
    CircularQueue cq(size);

    int choice, value;
    do {
        cout << "\nCircular Queue Operations:\n";
        cout << "1. Enqueue\n";
        cout << "2. Dequeue\n";
        cout << "3. Peek\n";
        cout << "4. Check if empty\n";
        cout << "5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter value to enqueue: ";
                cin >> value;
                cq.enqueue(value);
                break;
            case 2:
                cq.dequeue();
                break;
            case 3:
                value = cq.peek();
                if (value != -1)
                    cout << "Front element: " << value << endl;
                break;
            case 4:
                if (cq.isEmpty())
                    cout << "Queue is empty\n";
                else
                    cout << "Queue is not empty\n";
                break;
            case 5:
                cout << "Exiting program\n";
                break;
            default:
                cout << "Invalid choice\n";
        }
    } while (choice != 5);

    return 0;
}
```

Output: (screenshot)



```
● anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri % cd "/Users/anusrikarmokar/Desktop/Class
  nual_Anusri/" && g++ 6_Circular_Queue_using_array.cpp -o 6_Circular_Queue_using_array && "/Users/anusrika
  ++/Sem_II_DSA_Lab_Manual_Anusri/"6_Circular_Queue_using_array
  Enter the size of the circular queue: 5

  Circular Queue Operations:
  1. Enqueue
  2. Dequeue
  3. Peek
  4. Check if empty
  5. Exit
  Enter your choice: 1
  Enter value to enqueue: 22
  Enqueued 22

  Circular Queue Operations:
```

Test Case: Any two (screenshot)



```
  Circular Queue Operations:
  1. Enqueue
  2. Dequeue
  3. Peek
  4. Check if empty
  5. Exit
  Enter your choice: 2
  Dequeued 22

  Circular Queue Operations:
  1. Enqueue
  2. Dequeue
  3. Peek
  4. Check if empty
  5. Exit
  Enter your choice: 4
  Queue is empty

  Circular Queue Operations:
  1. Enqueue
  2. Dequeue
  3. Peek
  4. Check if empty
  5. Exit
  Enter your choice: 5
  Exiting program
○ anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri % █
```

Conclusion:

The code offers an optimized implementation of a queue using an array with

circular buffering, providing essential operations such as enqueue, dequeue, and display. It offers a user-friendly menu interface for interacting with the queue, allowing users to enqueue elements, dequeue elements, and view the elements currently in the queue.

Name of Student: Anusri Karmokar

Roll Number: 150096723003

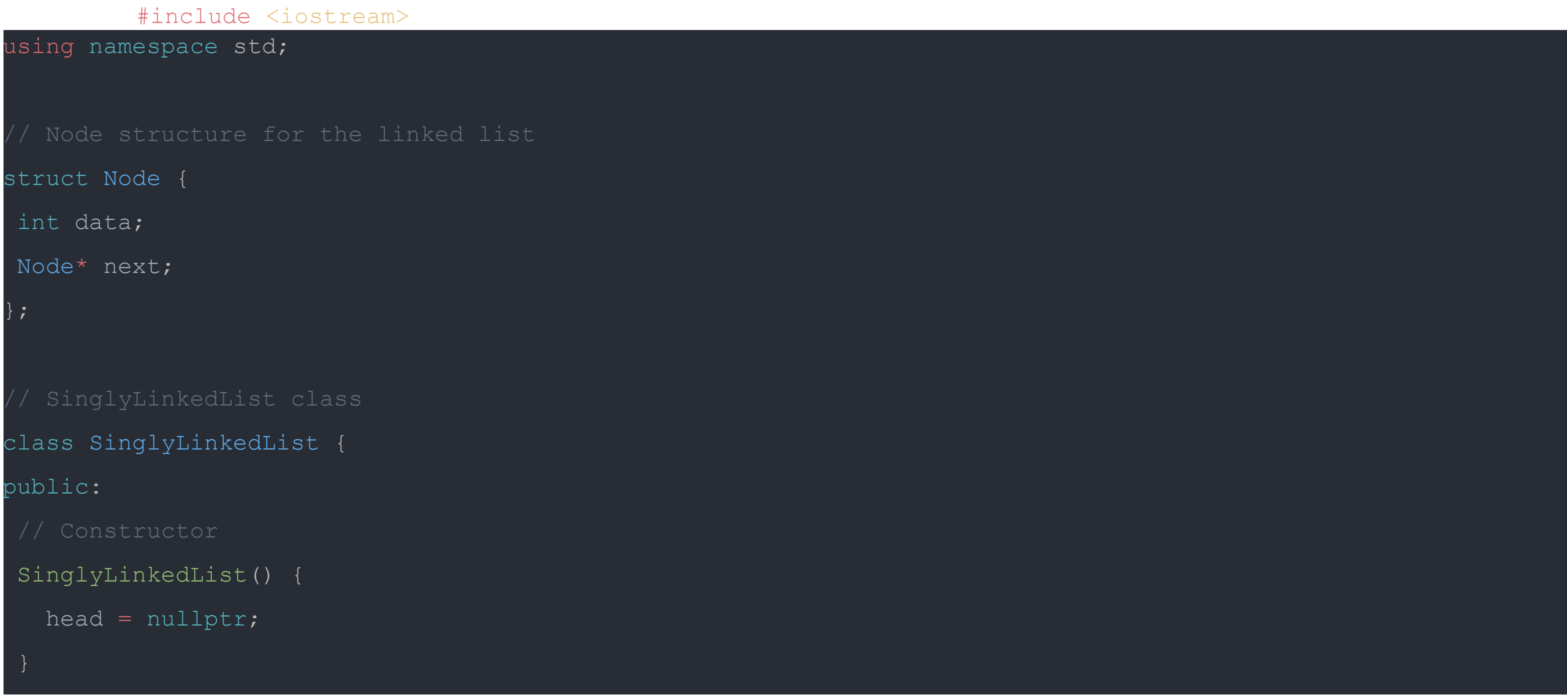
Experiment No: 7

Title: Implement Singly Linked List ADT.

Theory:

This code implements a singly linked list data structure with functionalities to append elements to the list, display the elements in the list, and clear the list. It utilizes a Node class to represent individual elements and a singlylist class to manage the list operations.

Code:

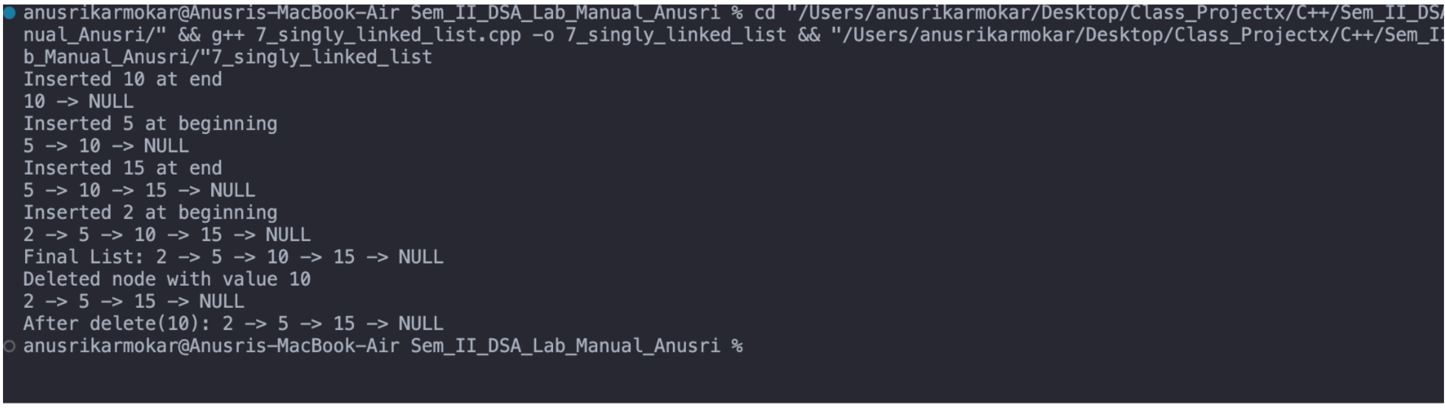


```
#include <iostream>
using namespace std;

// Node structure for the linked list
struct Node {
 int data;
 Node* next;
};

// SinglyLinkedList class
class SinglyLinkedList {
public:
 // Constructor
 SinglyLinkedList() {
   head = nullptr;
 }
```

```
// Function to insert a node at the beginning of the list
void insertAtBeginning(int data) {
  Node* newNode = new Node;
  newNode->data = data;
  newNode->next = head;
  head = newNode;
  printChange("Inserted " + to_string(data) + " at beginning");
}

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

  if (head == nullptr) {
    head = newNode;
    printChange("Inserted " + to_string(data) + " at end");
    return;
  }

  Node* current = head;
  while (current->next != nullptr) {
    current = current->next;
  }
  current->next = newNode;
  printChange("Inserted " + to_string(data) + " at end");
}

// Function to delete a node with a specific value
void deleteNode(int value) {
  if (head == nullptr) {
    return;
  }

  Node* current = head;
  Node* previous = nullptr;

  while (current != nullptr && current->data != value) {
    previous = current;
    current = current->next;
  }
```

```
   if (current == nullptr) {
     // Value not found
     return;
   }


   if (previous == nullptr) {
     // Delete head node
     head = current->next;
   } else {
     previous->next = current->next;
   }

   delete current;
   printChange("Deleted node with value " + to_string(value));
 }

 // Function to print the contents of the list
 void printList() {
   Node* current = head;
   while (current != nullptr) {
     cout << current->data << " -> ";
     current = current->next;
   }
   cout << "NULL" << endl;
 }

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

private:
 Node* head;  // Head pointer of the linked list

 // Function to clear the list (optional)
 void clear() {
   while (head != nullptr) {
     Node* temp = head;
     head = head->next;
     delete temp;
   }
 }
 // Helper function to print change message
 void printChange(const string& message) {
   cout << message << endl;
   printList();
 }
};

int main() {
 SinglyLinkedList list;

 list.insertAtEnd(10);
 list.insertAtBeginning(5);
 list.insertAtEnd(15);
 list.insertAtBeginning(2);

 cout << "Final List: ";
 list.printList();

 list.deleteNode(10);

 cout << "After delete(10): ";
 list.printList();

 return 0;
}
```

Output: (screenshot)



```
anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri % cd "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DSA
nual_Anusri/" && g++ 7_singly_linked_list.cpp -o 7_singly_linked_list && "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II
b_Manual_Anusri/"7_singly_linked_list
Inserted 10 at end
10 -> NULL
Inserted 5 at beginning
5 -> 10 -> NULL
Inserted 15 at end
5 -> 10 -> 15 -> NULL
Inserted 2 at beginning
2 -> 5 -> 10 -> 15 -> NULL
Final List: 2 -> 5 -> 10 -> 15 -> NULL
Deleted node with value 10
2 -> 5 -> 15 -> NULL
After delete(10): 2 -> 5 -> 15 -> NULL
anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri %
```

Conclusion:

The code offers a basic implementation of a singly linked list, allowing users to append elements to the list, display the elements. User can insert after the element, before the element can delete element etc.

Name of Student: Anusri Karmokar

Roll Number: 150096723003

Experiment No: 8

Title: Implement Circular Linked List ADT.

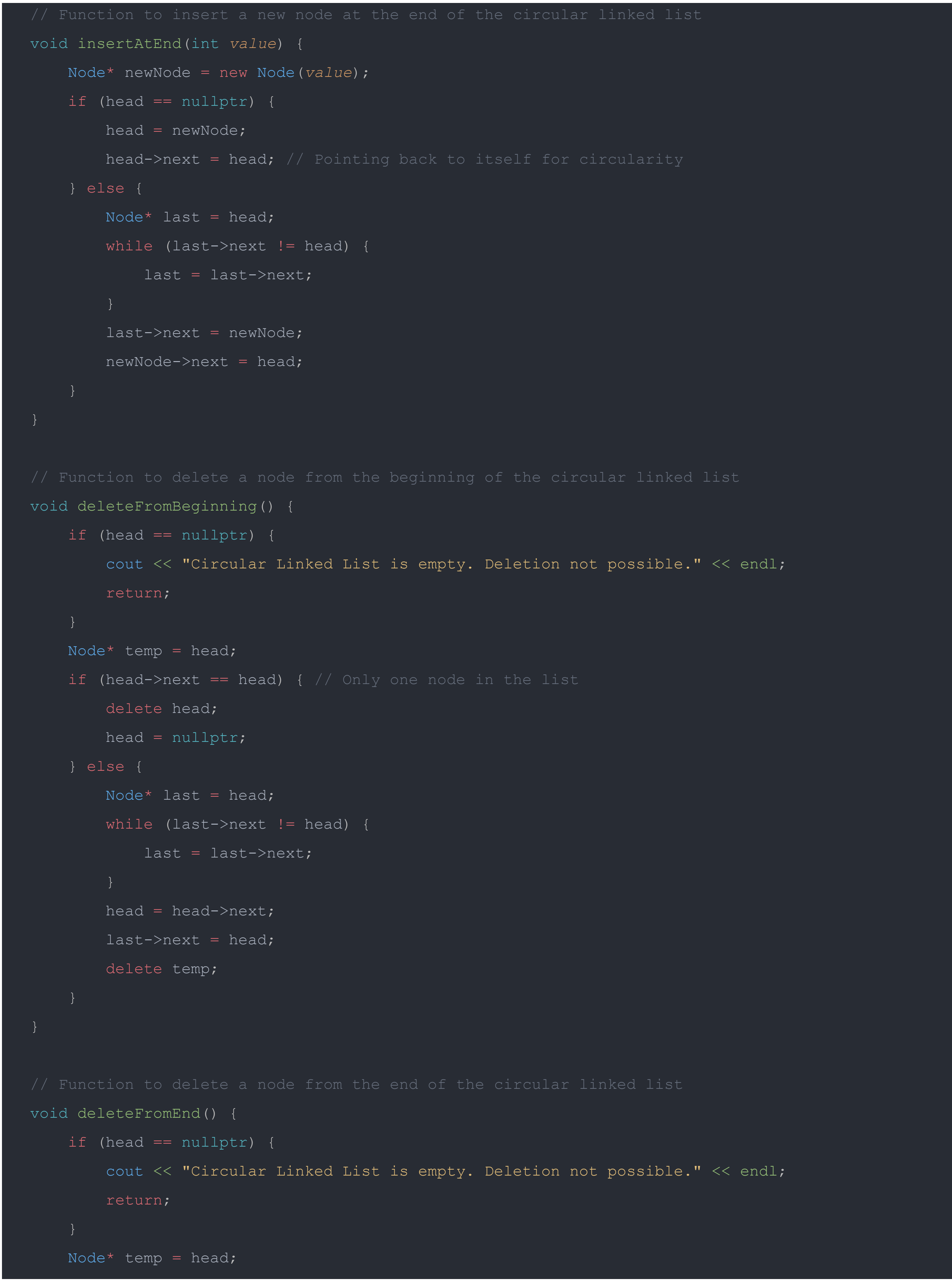
Theory:

This code implements a circular singly linked list data structure with functionalities to append elements to the list, display the elements in the list, and clear the list. It utilizes a Node class to represent individual elements and a circularlist class to manage the list operations.

Code:

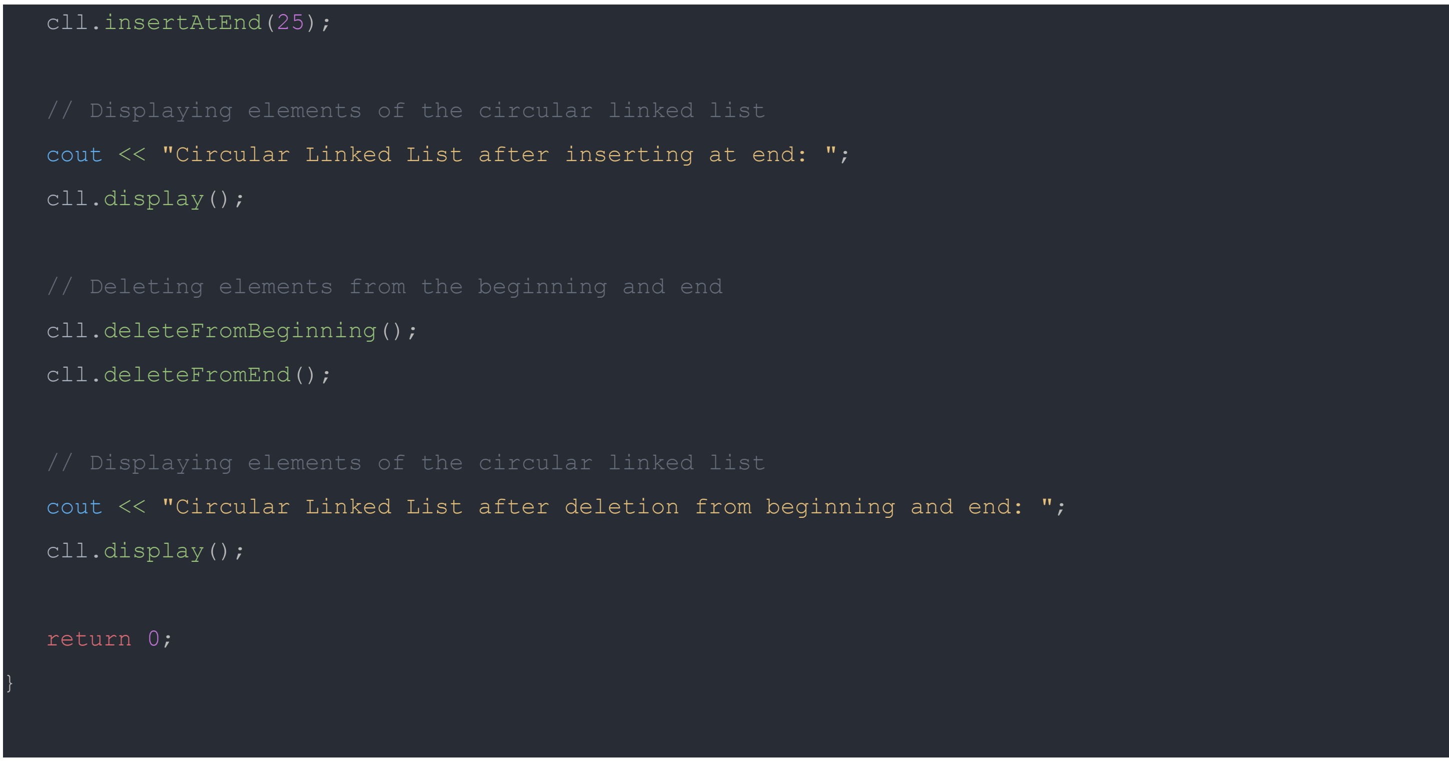
```
#include <iostream>

using namespace std;

// Node class for circular linked list
class Node {
public:
    int data;
    Node* next;

    // Constructor
    Node(int value) {
        data = value;
        next = nullptr;
    }
```

```
};

// Circular Linked List class
class CircularLinkedList {
private:
    Node* head; // Pointer to the head of the circular linked list

public:
    // Constructor
    CircularLinkedList() {
        head = nullptr;
    }

    // Destructor to free memory
    ~CircularLinkedList() {
        if (head == nullptr)
            return;

        Node* current = head->next;
        while (current != head) {
            Node* temp = current;
            current = current->next;
            delete temp;
        }
        delete head;
    }

    // Function to insert a new node at the beginning of the circular linked list
    void insertAtBeginning(int value) {
        Node* newNode = new Node(value);
        if (head == nullptr) {
            head = newNode;
            head->next = head; // Pointing back to itself for circularity
        } else {
            Node* last = head;
            while (last->next != head) {
                last = last->next;
            }
            newNode->next = head;
            last->next = newNode;
            head = newNode;
        }
    }
```



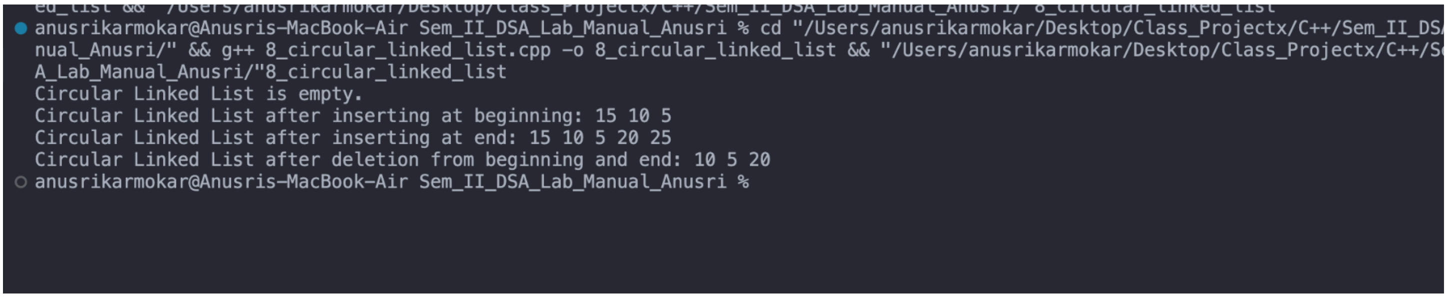
```
// Function to insert a new node at the end of the circular linked list
void insertAtEnd(int value) {
    Node* newNode = new Node(value);
    if (head == nullptr) {
        head = newNode;
        head->next = head; // Pointing back to itself for circularity
    } else {
        Node* last = head;
        while (last->next != head) {
            last = last->next;
        }
        last->next = newNode;
        newNode->next = head;
    }
}

// Function to delete a node from the beginning of the circular linked list
void deleteFromBeginning() {
    if (head == nullptr) {
        cout << "Circular Linked List is empty. Deletion not possible." << endl;
        return;
    }
    Node* temp = head;
    if (head->next == head) { // Only one node in the list
        delete head;
        head = nullptr;
    } else {
        Node* last = head;
        while (last->next != head) {
            last = last->next;
        }
        head = head->next;
        last->next = head;
        delete temp;
    }
}

// Function to delete a node from the end of the circular linked list
void deleteFromEnd() {
    if (head == nullptr) {
        cout << "Circular Linked List is empty. Deletion not possible." << endl;
        return;
    }
    Node* temp = head;
```

```
        if (head->next == head) { // Only one node in the list
            delete head;
            head = nullptr;
        } else {
            Node* last = head;
            while (last->next->next != head) {
                last = last->next;
            }
            Node* toDelete = last->next;
            last->next = head;
            delete toDelete;
        }
    }

    // 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;
    }
};

int main() {
    CircularLinkedList cll;
    cll.display();

    // Inserting elements into the circular linked list
    cll.insertAtBeginning(5);
    cll.insertAtBeginning(10);
    cll.insertAtBeginning(15);

    // Displaying elements of the circular linked list
    cout << "Circular Linked List after inserting at beginning: ";
    cll.display();

    // Inserting elements at the end
    cll.insertAtEnd(20);
```



```
    cll.insertAtEnd(25);

    // Displaying elements of the circular linked list
    cout << "Circular Linked List after inserting at end: ";
    cll.display();

    // Deleting elements from the beginning and end
    cll.deleteFromBeginning();
    cll.deleteFromEnd();

    // Displaying elements of the circular linked list
    cout << "Circular Linked List after deletion from beginning and end: ";
    cll.display();

    return 0;
}
```

Output: (screenshot)



```
  ed_list && "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DSA_Lab_Manual_Anusri/"8_circular_linked_list
● anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri % cd "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DS
  nual_Anusri/" && g++ 8_circular_linked_list.cpp -o 8_circular_linked_list && "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/S
  A_Lab_Manual_Anusri/"8_circular_linked_list
  Circular Linked List is empty.
  Circular Linked List after inserting at beginning: 15 10 5
  Circular Linked List after inserting at end: 15 10 5 20 25
  Circular Linked List after deletion from beginning and end: 10 5 20
○ anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri %
```

Conclusion:

The code offers an implementation of a circular singly linked list, allowing users to append elements to the list, display the elements currently in the list, and clear the list. It utilizes circular linking to ensure that the last node points back to the head, creating a circular structure. The provided menu interface makes it easy for users to perform operations on the list.

Name of Student: **Anusri Karmokar**

Roll Number: **150096723003**

Experiment No: 9

Title: Implement Stack ADT using Linked List

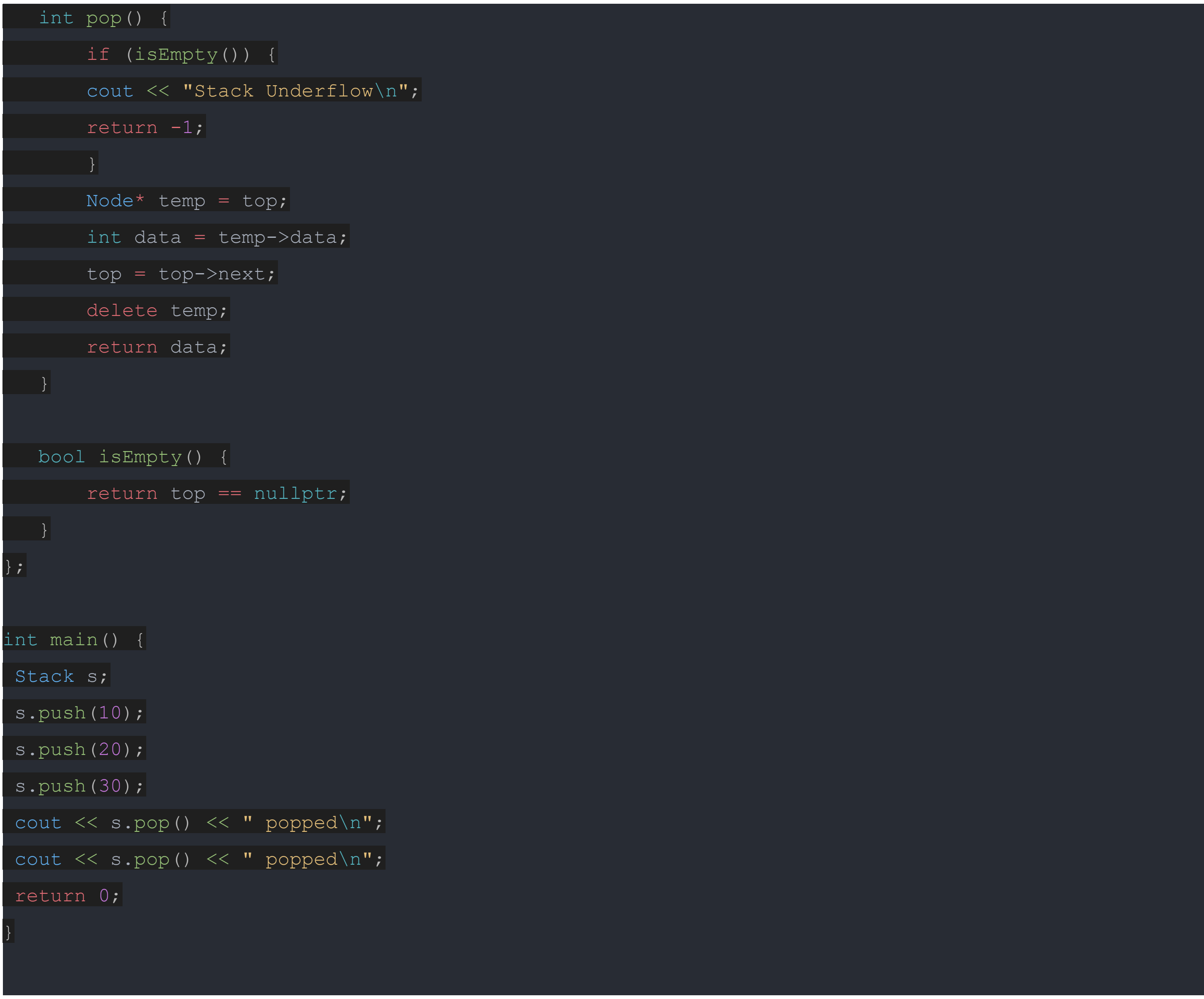
Theory:

Theory:

This code implements a stack data structure using a singly linked list. It provides functionalities to push elements onto the stack, pop elements from the stack, display the elements in the stack, and check if the stack is empty.

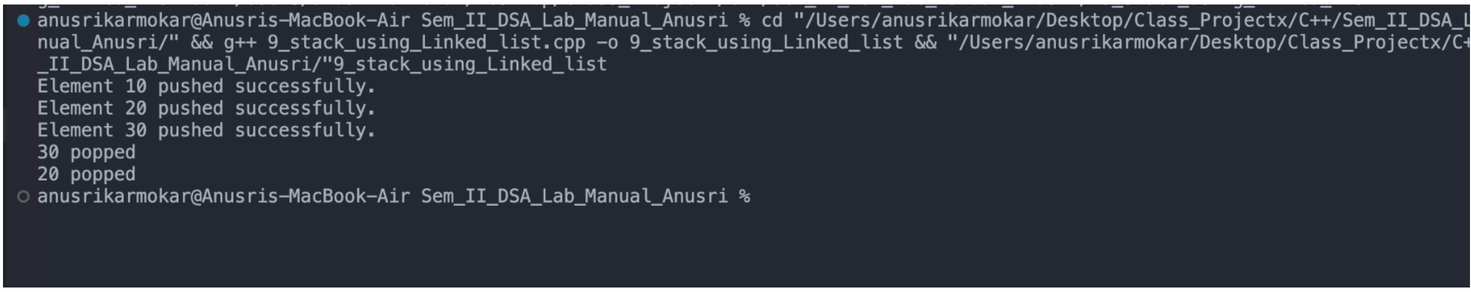
Code:

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

struct Node {
 int data;
 Node* next;
};

class Stack {
   private:
       Node* top;

   public:
   Stack() {
       top = nullptr;
   }

   void push(int data) {
       Node* newNode = new Node;
       newNode->data = data;
       newNode->next = top;
       top = newNode;
       cout << "Element " << data << " pushed successfully." << endl;
   }
```



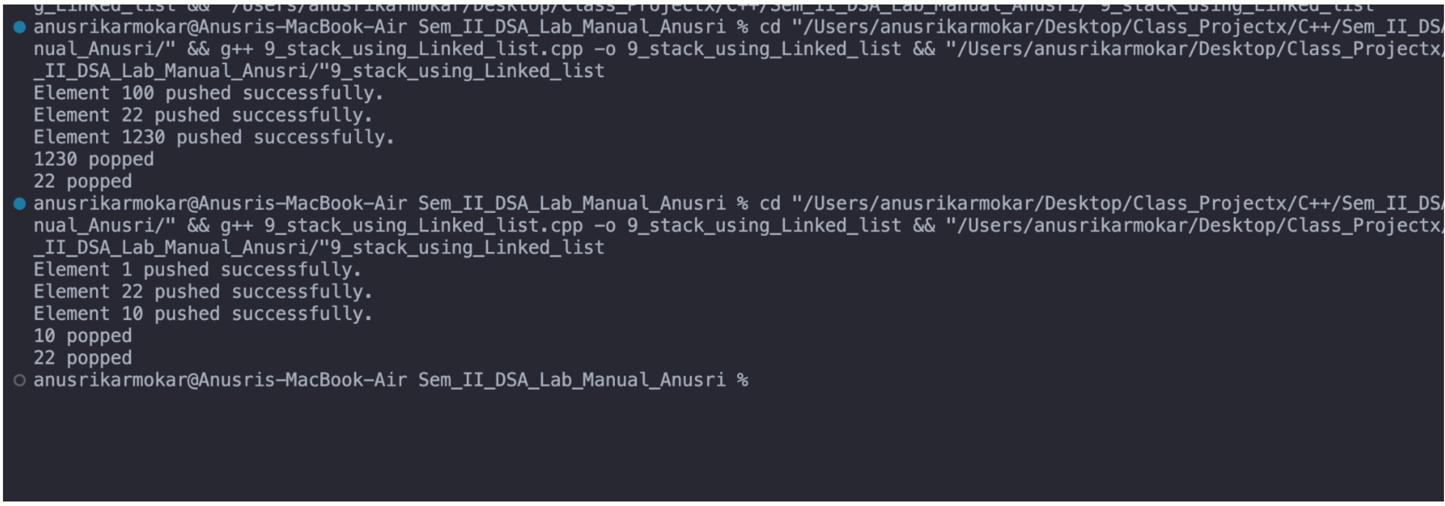
```
    int pop() {
        if (isEmpty()) {
        cout << "Stack Underflow\n";
        return -1;
        }
        Node* temp = top;
        int data = temp->data;
        top = top->next;
        delete temp;
        return data;
    }

    bool isEmpty() {
        return top == nullptr;
    }
};

int main() {
 Stack s;
 s.push(10);
 s.push(20);
 s.push(30);
 cout << s.pop() << " popped\n";
 cout << s.pop() << " popped\n";
 return 0;
}
```

Output: (screenshot)



```
● anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri % cd "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DSA_L
  nual_Anusri/" && g++ 9_stack_using_Linked_list.cpp -o 9_stack_using_Linked_list && "/Users/anusrikarmokar/Desktop/Class_Projectx/C+
  _II_DSA_Lab_Manual_Anusri/"9_stack_using_Linked_list
  Element 10 pushed successfully.
  Element 20 pushed successfully.
  Element 30 pushed successfully.
  30 popped
  20 popped
○ anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri %
```

Test Case: Any two (screenshot)



```
  g_Linked_list && "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DSA_Lab_Manual_Anusri/"9_stack_using_Linked_list
● anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri % cd "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DS
  nual_Anusri/" && g++ 9_stack_using_Linked_list.cpp -o 9_stack_using_Linked_list && "/Users/anusrikarmokar/Desktop/Class_Projectx
  _II_DSA_Lab_Manual_Anusri/"9_stack_using_Linked_list
  Element 100 pushed successfully.
  Element 22 pushed successfully.
  Element 1230 pushed successfully.
  1230 popped
  22 popped
● anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri % cd "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DS
  nual_Anusri/" && g++ 9_stack_using_Linked_list.cpp -o 9_stack_using_Linked_list && "/Users/anusrikarmokar/Desktop/Class_Projectx
  _II_DSA_Lab_Manual_Anusri/"9_stack_using_Linked_list
  Element 1 pushed successfully.
  Element 22 pushed successfully.
  Element 10 pushed successfully.
  10 popped
  22 popped
○ anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri %
```

Conclusion:

The code offers a flexible implementation of a stack using a singly linked list, providing essential operations such as push, pop, and display. It utilizes dynamic memory allocation to manage nodes, allowing for efficient memory usage. The provided menu interface makes it easy for users to perform operations on the stack.

Name of Student: Anusri Karmokar

Roll Number: 150096723003

Experiment No: 10

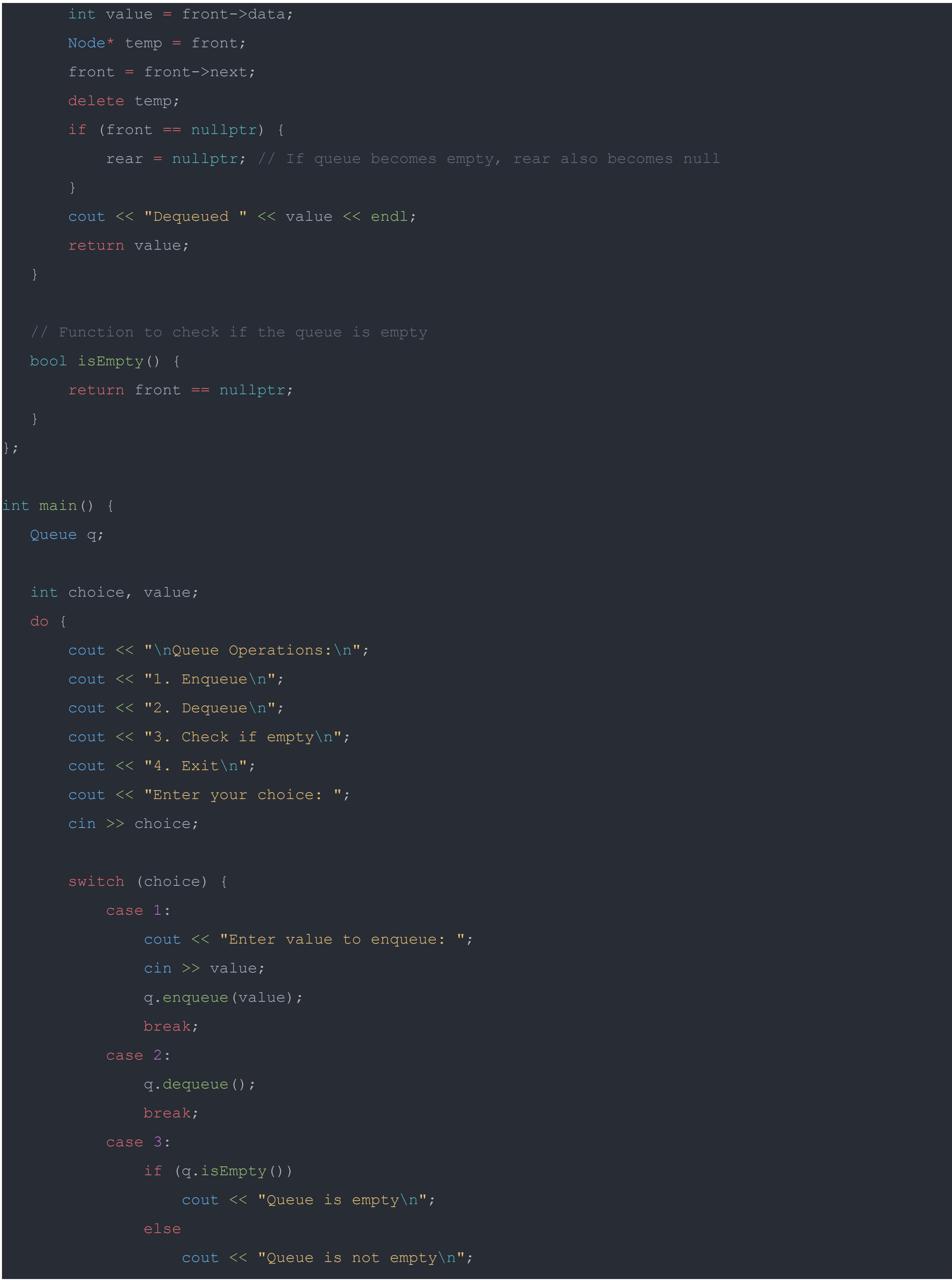
Title: Implement Linear Queue ADT using Linked List

Theory:

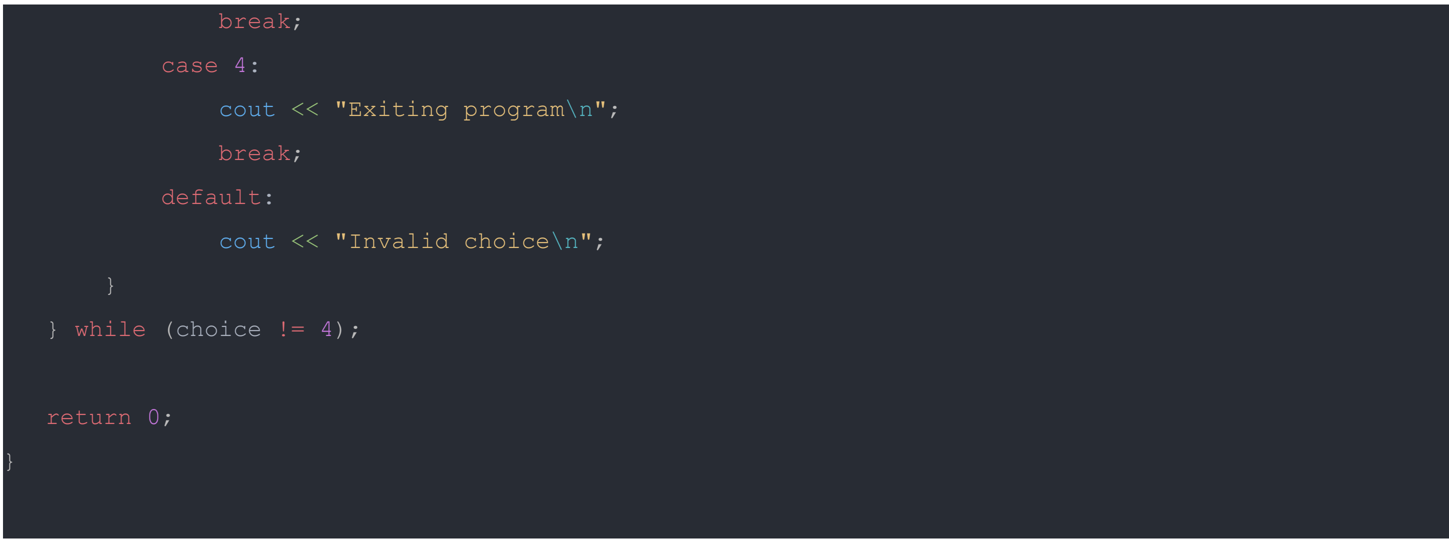
This code implements a queue data structure using a singly linked list. It provides functionalities to enqueue elements into the queue, dequeue elements from the queue, display the elements in the queue, and check if the queue is empty.

Code:

```
#include <iostream>
using namespace std;
// Node structure for the linked list
struct Node {
    int data;
    Node* next;
    Node(int value) : data(value), next(nullptr) {}
};

class Queue {
private:
    Node* front;
    Node* rear;

public:
    Queue() : front(nullptr), rear(nullptr) {}

    ~Queue() {
        while (!isEmpty()) {
            dequeue(); // Dequeue all elements and deallocate memory
        }
    }

    // Function to enqueue an element
    void enqueue(int value) {
        Node* newNode = new Node(value);
        if (isEmpty()) {
            front = rear = newNode;
        } else {
            rear->next = newNode;
            rear = newNode;
        }
        cout << "Enqueued " << value << endl;
    }

    // Function to dequeue an element
    int dequeue() {
        if (isEmpty()) {
            cout << "Queue Underflow\n";
            return -1;
        }
```

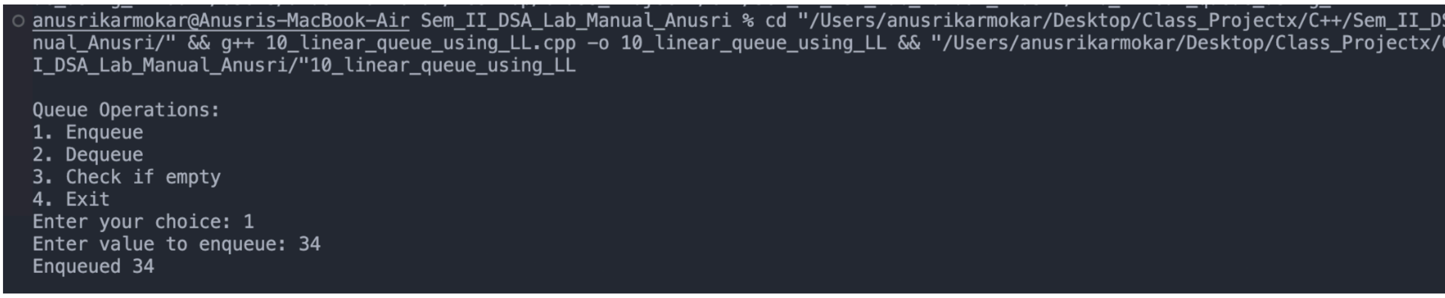


```cpp
        int value = front->data;
        Node* temp = front;
        front = front->next;
        delete temp;
        if (front == nullptr) {
            rear = nullptr; // If queue becomes empty, rear also becomes null
        }
        cout << "Dequeued " << value << endl;
        return value;
    }

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

int main() {
    Queue q;

    int choice, value;
    do {
        cout << "\nQueue Operations:\n";
        cout << "1. Enqueue\n";
        cout << "2. Dequeue\n";
        cout << "3. Check if empty\n";
        cout << "4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter value to enqueue: ";
                cin >> value;
                q.enqueue(value);
                break;
            case 2:
                q.dequeue();
                break;
            case 3:
                if (q.isEmpty())
                    cout << "Queue is empty\n";
                else
                    cout << "Queue is not empty\n";
```



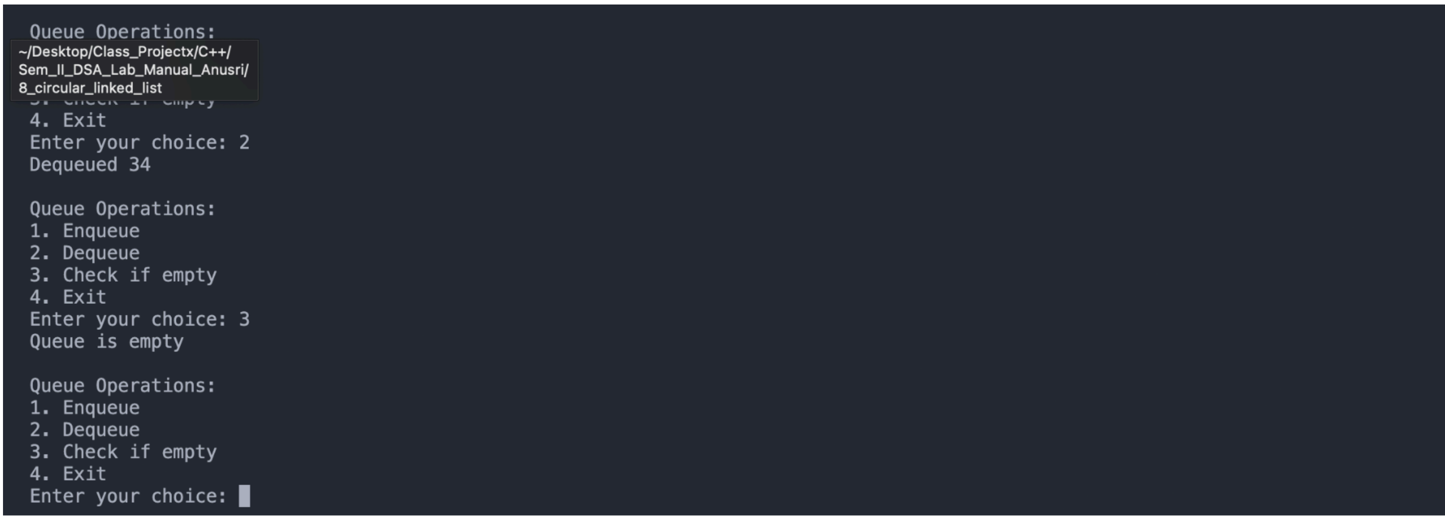
```
                break;
            case 4:
                cout << "Exiting program\n";
                break;
            default:
                cout << "Invalid choice\n";
        }
    } while (choice != 4);

    return 0;
}
```

Output: (screenshot)



```
○ anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri % cd "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DS
  nual_Anusri/" && g++ 10_linear_queue_using_LL.cpp -o 10_linear_queue_using_LL && "/Users/anusrikarmokar/Desktop/Class_Projectx/C
  I_DSA_Lab_Manual_Anusri/"10_linear_queue_using_LL

  Queue Operations:
  1. Enqueue
  2. Dequeue
  3. Check if empty
  4. Exit
  Enter your choice: 1
  Enter value to enqueue: 34
  Enqueued 34
```

Test Case: Any two (screenshot)



```
  Queue Operations:
  3. Check if empty
  4. Exit
  Enter your choice: 2
  Dequeued 34

  Queue Operations:
  1. Enqueue
  2. Dequeue
  3. Check if empty
  4. Exit
  Enter your choice: 3
  Queue is empty

  Queue Operations:
  1. Enqueue
  2. Dequeue
  3. Check if empty
  4. Exit
  Enter your choice: █
```

Conclusion:

The code offers a flexible implementation of a queue using a singly linked list, providing essential operations such as enqueue, dequeue, and display. It utilizes dynamic memory allocation to manage nodes, allowing for efficient memory usage.

Name of Student: Anusri Karmokar

Roll Number: 150096723003

Experiment No: 11

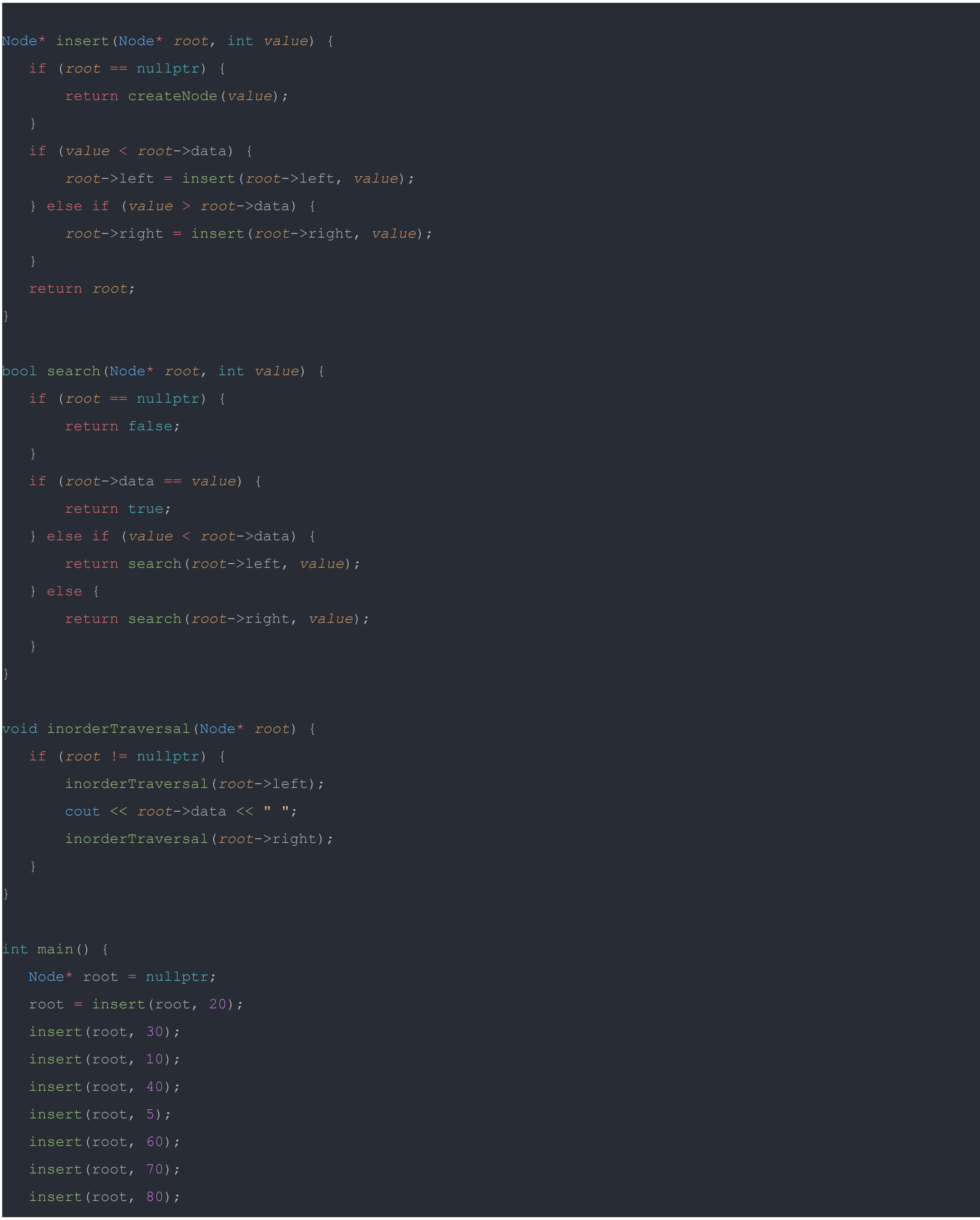
Title: Implement Binary Search Tree ADT using Linked List.

Theory:

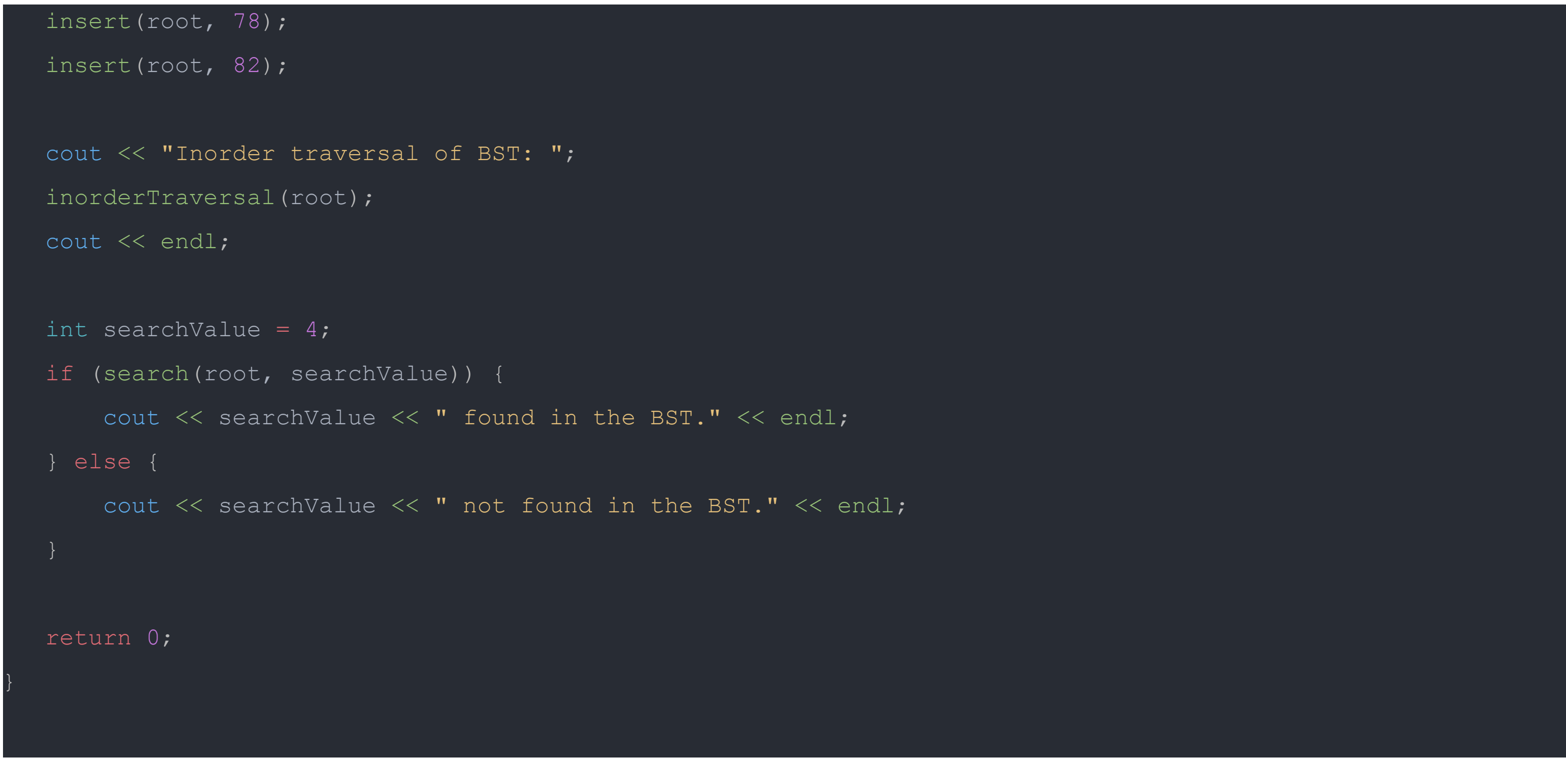
A binary tree is a non-linear data structure in which there is a root node and each parent node has 0,1 or 2 child nodes at most. In binary search tree, all the nodes having values less than that of the root node are present in the left subtree of the root node and all the nodes having values greater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

Code:

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

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

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

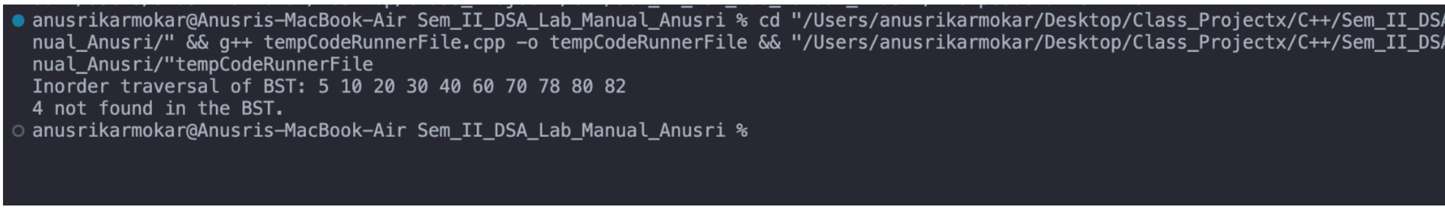


```cpp
Node* insert(Node* root, int value) {
   if (root == nullptr) {
       return createNode(value);
   }
   if (value < root->data) {
       root->left = insert(root->left, value);
   } else if (value > root->data) {
       root->right = insert(root->right, value);
   }
   return root;
}

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

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

int main() {
   Node* root = nullptr;
   root = insert(root, 20);
   insert(root, 30);
   insert(root, 10);
   insert(root, 40);
   insert(root, 5);
   insert(root, 60);
   insert(root, 70);
   insert(root, 80);
```



```
    insert(root, 78);

    insert(root, 82);


    cout << "Inorder traversal of BST: ";

    inorderTraversal(root);

    cout << endl;


    int searchValue = 4;

    if (search(root, searchValue)) {

        cout << searchValue << " found in the BST." << endl;

    } else {

        cout << searchValue << " not found in the BST." << endl;

    }


    return 0;

}
```

Output: (screenshot)



```
● anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri % cd "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DS/
  nual_Anusri/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DS/
  nual_Anusri/"tempCodeRunnerFile
  Inorder traversal of BST: 5 10 20 30 40 60 70 78 80 82
  4 not found in the BST.
○ anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri %
```

Conclusion:

The code provides a flexible implementation of a binary search tree, allowing users to insert elements, search for elements, and display the elements in sorted order. It utilizes a recursive approach for insertion and traversal, ensuring efficient operations on the tree.

Name of Student: **Anusri Karmokar**

Roll Number: **150096723003**

Experiment No: 12

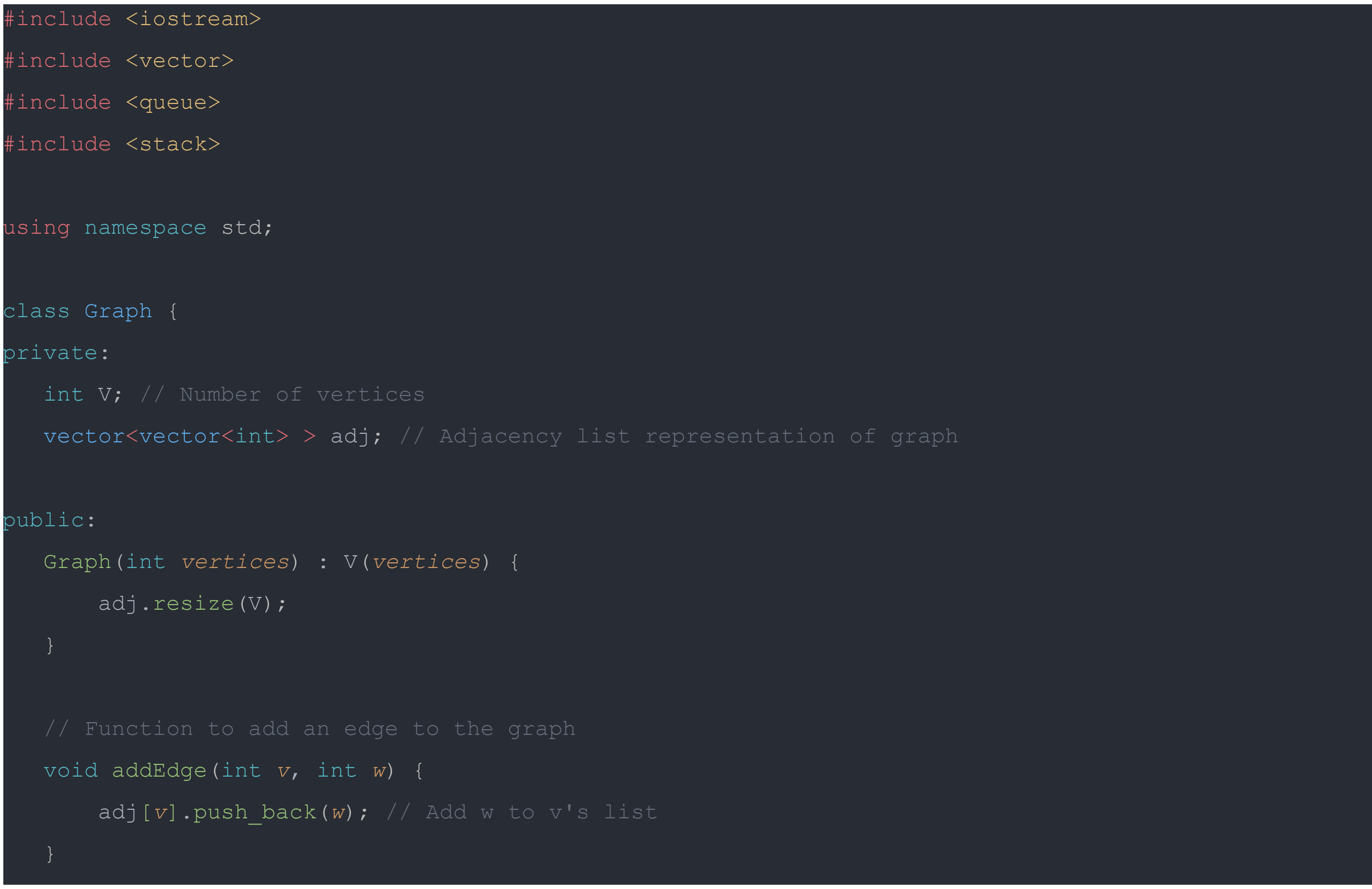
Title: Implement Graph Traversal techniques:

a) Depth First Search b) Breadth First Search

Theory:

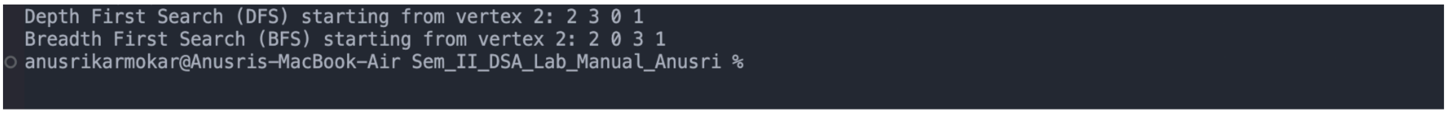
A Graph is a non-linear data structure which can have parent-child as well as other complex relationships between the nodes. It is a set of edges and vertices, where vertices are the nodes, and the edges are the links connecting the nodes. We can implement a graph using adjacency matrix or adjacency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

Code:



```
#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 representation of graph

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

    // Function to add an edge to the graph
    void addEdge(int v, int w) {
        adj[v].push_back(w); // Add w to v's list
    }
    // Depth First Search traversal starting from a given vertex
    void DFS(int start) {
        // Mark all the vertices as not visited
        vector<bool> visited(V, false);

        // Create a stack for DFS
        stack<int> stack;

        // Push the current source node
        stack.push(start);

        while (!stack.empty()) {
            // Pop a vertex from stack and print it
            int current = stack.top();
            stack.pop();

            // Stack may contain same vertex twice. So, we need to print the popped item only if it is not visited.
            if (!visited[current]) {
                cout << current << " ";
                visited[current] = true; // Mark the current node as visited

                // Get all adjacent vertices of the popped vertex and push the adjacent vertices to the stack if not already visited
                for (auto it = adj[current].begin(); it != adj[current].end(); ++it) {
                    if (!visited[*it]) {
                        stack.push(*it);
                    }
                }
            }
        }
    }

    // Breadth First Search traversal starting from a given vertex
    void BFS(int start) {
        // Mark all the vertices as not visited
        vector<bool> visited(V, false);

        // Create a queue for BFS
        queue<int> queue;

        // Mark the current node as visited and enqueue it
        visited[start] = true;
        queue.push(start);

        while (!queue.empty()) {
            // Dequeue a vertex from queue and print it
            int current = queue.front();
            cout << current << " ";
            queue.pop();

            // Get all adjacent vertices of the dequeued vertex current. If an adjacent vertex has not been visited, then mark it visited and enqueue it
            for (auto it = adj[current].begin(); it != adj[current].end(); ++it) {
                if (!visited[*it]) {
                    visited[*it] = true;
                    queue.push(*it);
                }
            }
        }
    }
};

int main() {
    // Create a graph given in the above diagram
    Graph g(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);

    cout << "Depth First Search (DFS) starting from vertex 2: ";
    g.DFS(2);
    cout << endl;

    cout << "Breadth First Search (BFS) starting from vertex 2: ";
    g.BFS(2);
    cout << endl;

    return 0;
}
```

Output: (screenshot)



```
Depth First Search (DFS) starting from vertex 2: 2 3 0 1
Breadth First Search (BFS) starting from vertex 2: 2 0 3 1
○ anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri %
```

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

Name of Student: Anusri Karmokar

Roll Number: 150096723003

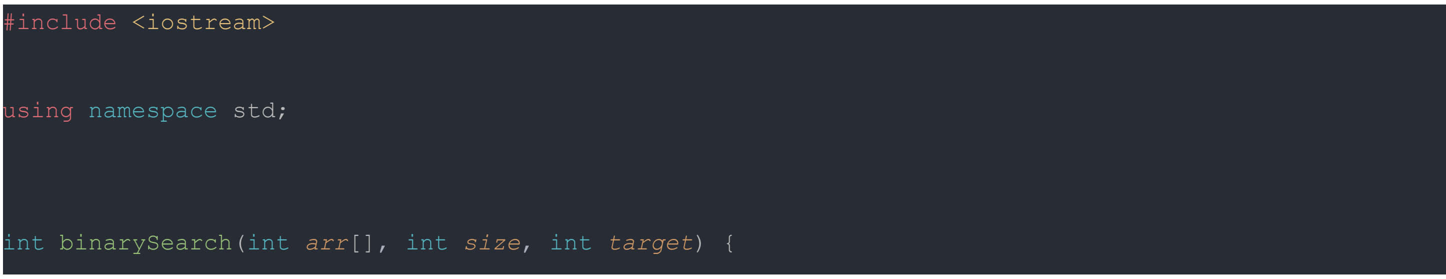
Experiment No: 13

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

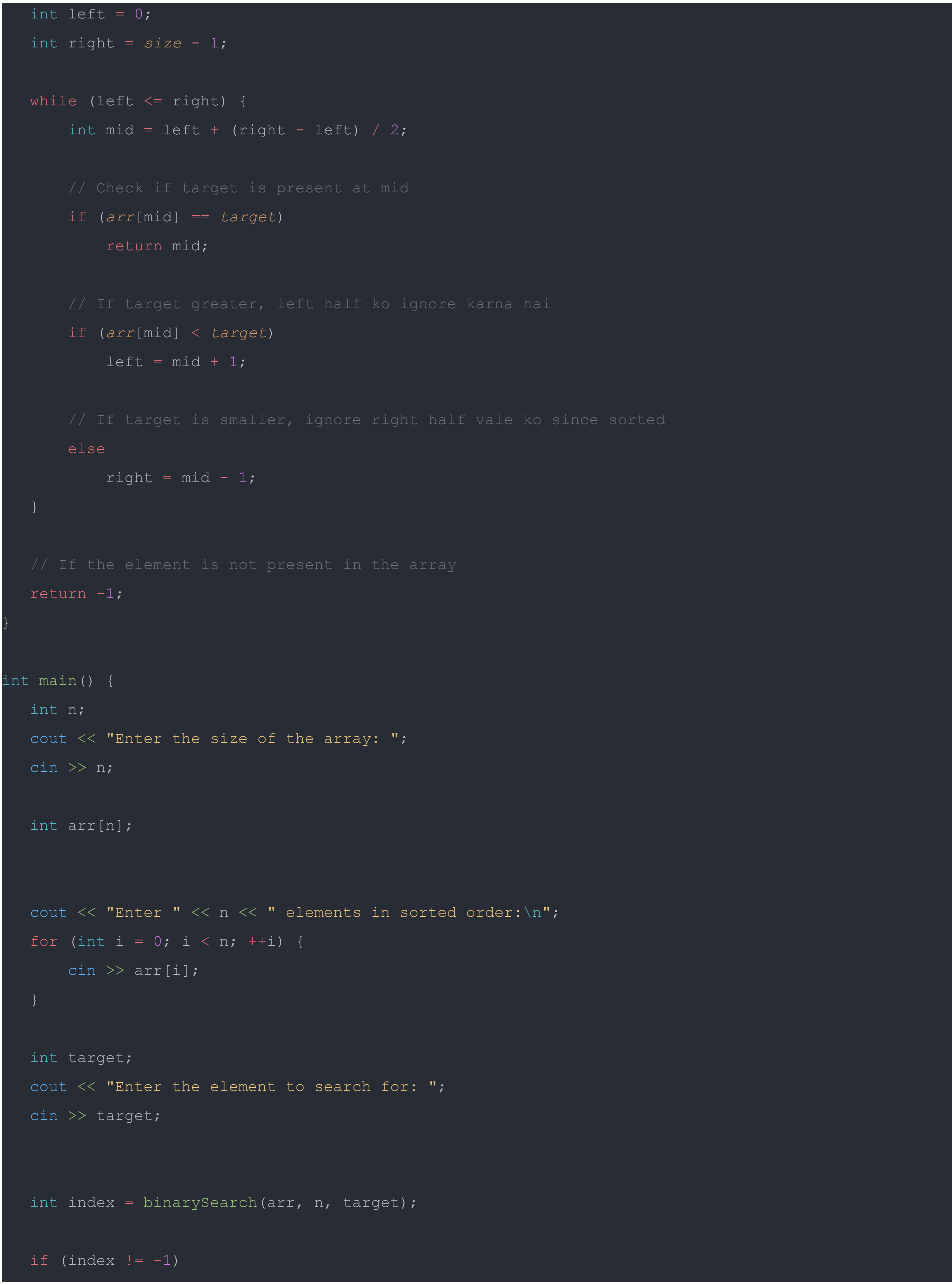
Theory:

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:



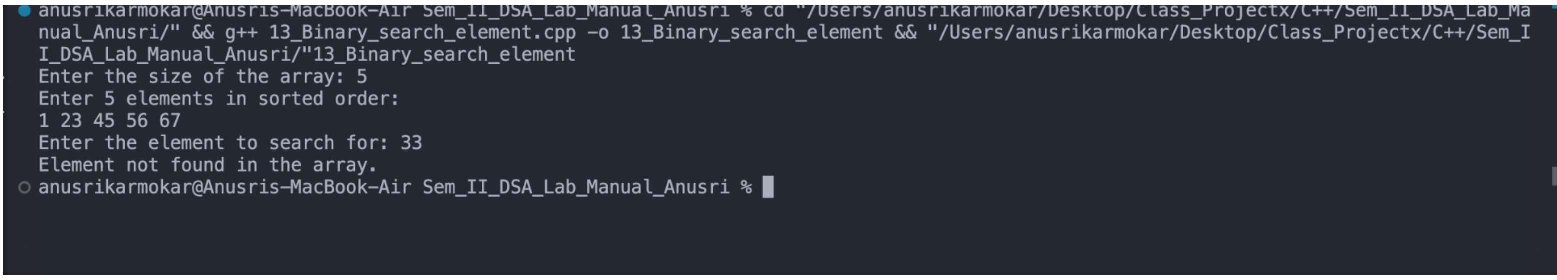
```
#include <iostream>


using namespace std;



int binarySearch(int arr[], int size, int target) {
```



```
    int left = 0;
    int right = size - 1;

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

        // Check if target is present at mid
        if (arr[mid] == target)
            return mid;

        // If target greater, left half ko ignore karna hai
        if (arr[mid] < target)
            left = mid + 1;

        // If target is smaller, ignore right half vale ko since sorted
        else
            right = mid - 1;
    }

    // If the element is not present in the array
    return -1;
}

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

    int arr[n];



    cout << "Enter " << n << " elements in sorted order:\n";
    for (int i = 0; i < n; ++i) {
        cin >> arr[i];
    }

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



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

    if (index != -1)
```

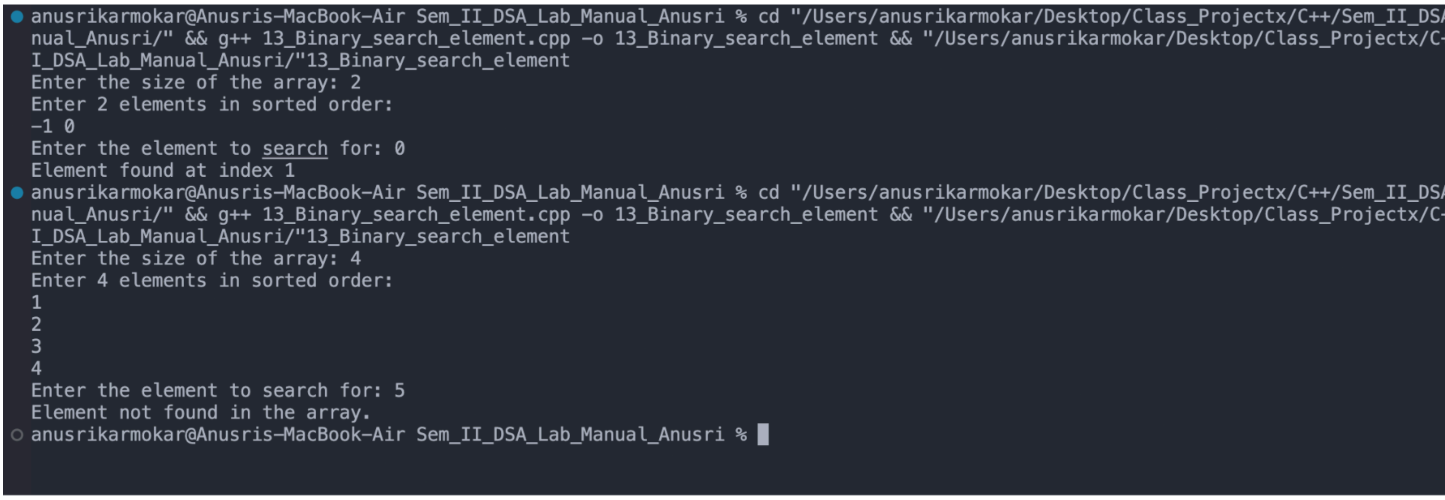
```
        cout << "Element found at index " << index << endl;
    else
        cout << "Element not found in the array." << endl;

    return 0;
}
```

Output: (screenshot)



```
● anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri % cd "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DSA_Lab_Ma
  nual_Anusri/" && g++ 13_Binary_search_element.cpp -o 13_Binary_search_element && "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_I
  I_DSA_Lab_Manual_Anusri/"13_Binary_search_element
  Enter the size of the array: 5
  Enter 5 elements in sorted order:
  1 23 45 56 67
  Enter the element to search for: 33
  Element not found in the array.
○ anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri % █
```

Test Case: Any two (screenshot)



```
● anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri % cd "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DS
  nual_Anusri/" && g++ 13_Binary_search_element.cpp -o 13_Binary_search_element && "/Users/anusrikarmokar/Desktop/Class_Projectx/C-
  I_DSA_Lab_Manual_Anusri/"13_Binary_search_element
  Enter the size of the array: 2
  Enter 2 elements in sorted order:
  -1 0
  Enter the element to search for: 0
  Element found at index 1
● anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri % cd "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DS
  nual_Anusri/" && g++ 13_Binary_search_element.cpp -o 13_Binary_search_element && "/Users/anusrikarmokar/Desktop/Class_Projectx/C-
  I_DSA_Lab_Manual_Anusri/"13_Binary_search_element
  Enter the size of the array: 4
  Enter 4 elements in sorted order:
  1
  2
  3
  4
  Enter the element to search for: 5
  Element not found in the array.
○ anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri % █
```

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: Anusri Karmokar

Roll Number: 150096723003

Experiment No: 14

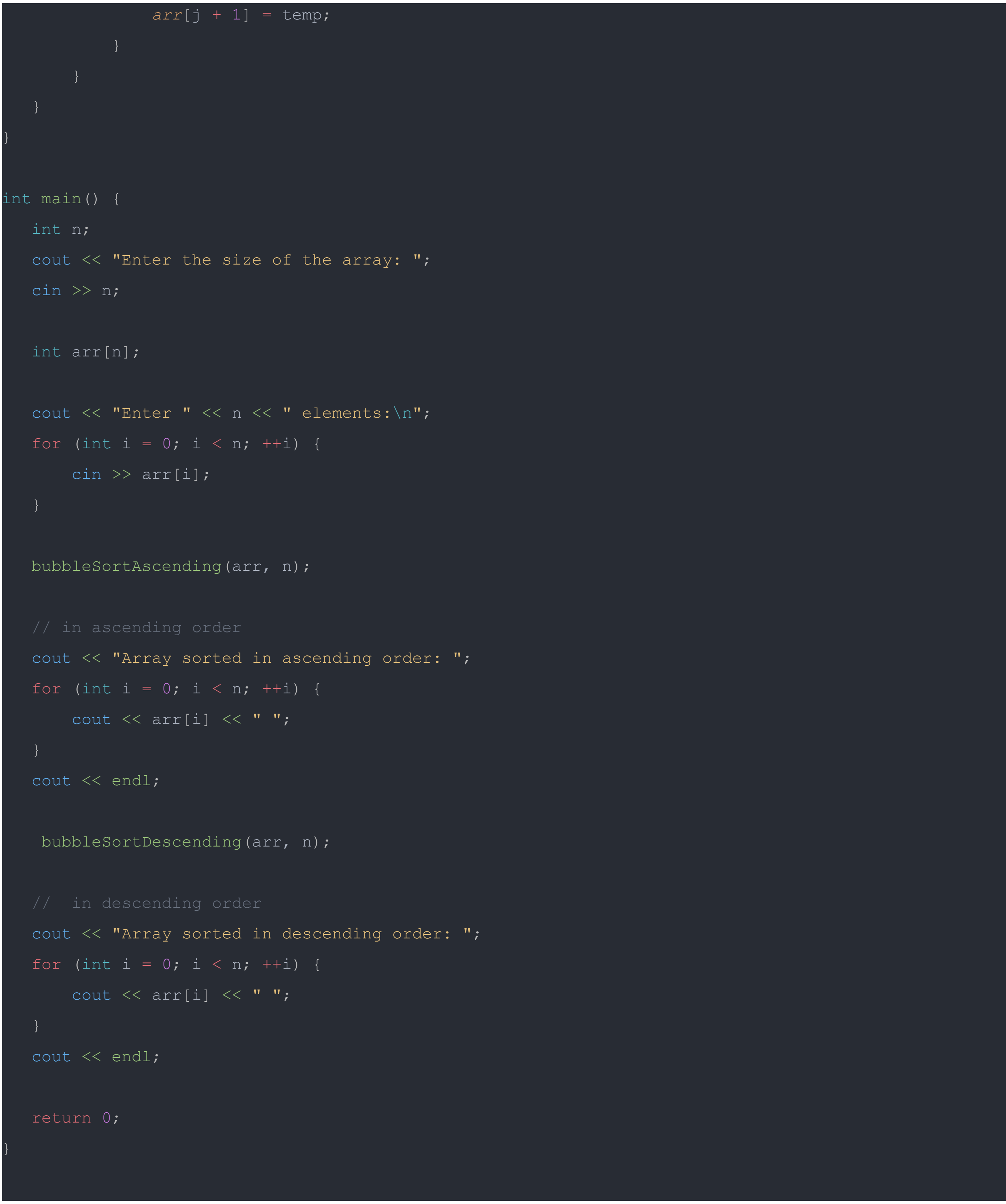
Title: Implement Bubble sort algorithm to sort elements of an array in ascending and descending order

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.

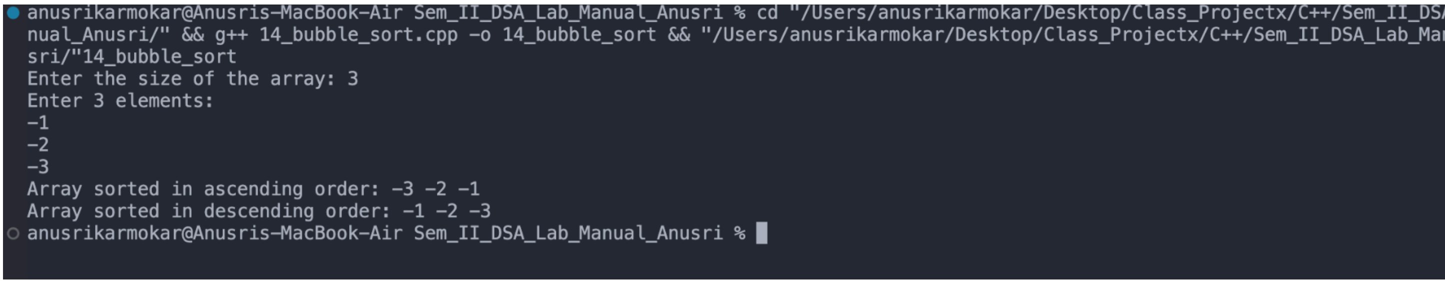
Code:

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

//  Bubble Sort in ascending order
void bubbleSortAscending(int arr[], int size) {
   for (int i = 0; i < size - 1; ++i) {
       for (int j = 0; j < size - 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;
           }
       }
   }
}

//Bubble Sort in descending order
void bubbleSortDescending(int arr[], int size) {
   for (int i = 0; i < size - 1; ++i) {
       for (int j = 0; j < size - 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];
```



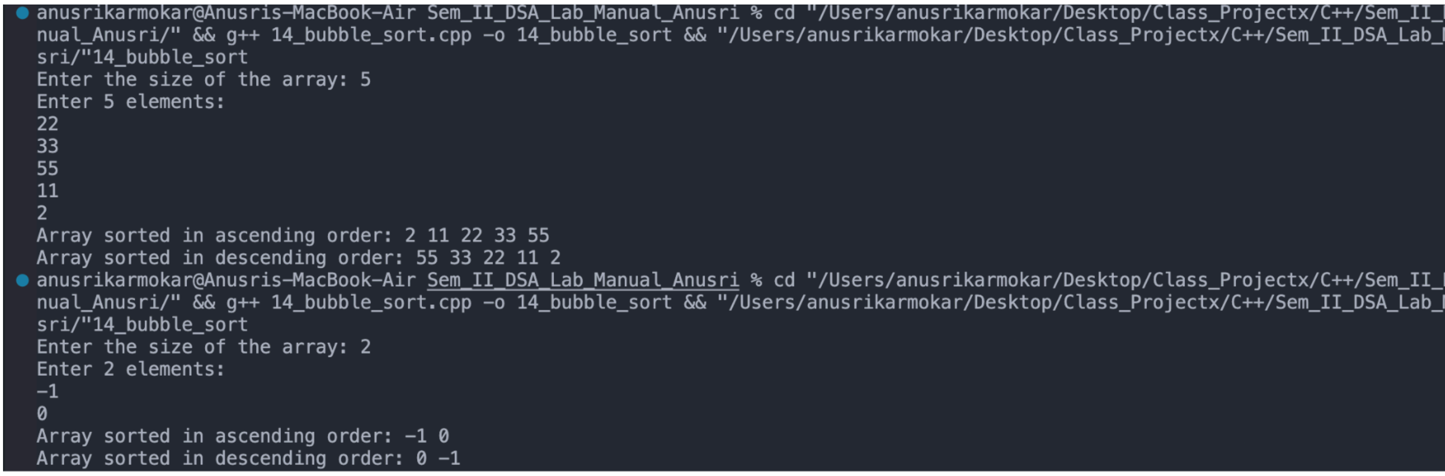
```cpp
                arr[j + 1] = temp;
            }
        }
    }
}

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

    int arr[n];

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

    bubbleSortAscending(arr, n);

    // in ascending order
    cout << "Array sorted in ascending order: ";
    for (int i = 0; i < n; ++i) {
        cout << arr[i] << " ";
    }
    cout << endl;

     bubbleSortDescending(arr, n);

    //  in descending order
    cout << "Array sorted in descending order: ";
    for (int i = 0; i < n; ++i) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```

Output: (screenshot)



```
anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri % cd "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DS
nual_Anusri/" && g++ 14_bubble_sort.cpp -o 14_bubble_sort && "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DSA_Lab_Ma
sri/"14_bubble_sort
Enter the size of the array: 3
Enter 3 elements:
-1
-2
-3
Array sorted in ascending order: -3 -2 -1
Array sorted in descending order: -1 -2 -3
anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri %
```

Test Case: Any two (screenshot)



```
anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri % cd "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_
nual_Anusri/" && g++ 14_bubble_sort.cpp -o 14_bubble_sort && "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DSA_Lab_
sri/"14_bubble_sort
Enter the size of the array: 5
Enter 5 elements:
22
33
55
11
2
Array sorted in ascending order: 2 11 22 33 55
Array sorted in descending order: 55 33 22 11 2
anusrikarmokar@Anusris-MacBook-Air Sem_II_DSA_Lab_Manual_Anusri % cd "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_
nual_Anusri/" && g++ 14_bubble_sort.cpp -o 14_bubble_sort && "/Users/anusrikarmokar/Desktop/Class_Projectx/C++/Sem_II_DSA_Lab_
sri/"14_bubble_sort
Enter the size of the array: 2
Enter 2 elements:
-1
0
Array sorted in ascending order: -1 0
Array sorted in descending order: 0 -1
```

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.