

Applied Programming

Assignment 2



November 4, 2022

Muhammad talha Arif

22F-3769

# **Question#1**

Calculator using Stack ADT

## **Code**

#include <iostream>

#include <string>

using namespace std;

#define IS\_DIGIT(c) (c >= '0' && c <= '9')

template <class T>

struct Node

{

T data;

Node<T>\* next;

};

template <class T>

class Stack

{

private:

Node<T>\* Top;

public:

Stack()

{

Top = NULL; // Initialize Top to NULL

}

bool Push(T value)

{

Node<T>\* node = new Node<T>();

if (node == NULL) // If memory alloc() failed.

{

return false;

}

node->data = value;

node->next = Top;

Top = node;

return true;

}

void Pop()

{

if (!IsEmpty()) // if Top == NULL

{

Node<T>\* node = Top;

Top = Top->next;

delete node;

}

}

bool IsEmpty()

{

return (Top == NULL);

}

void Display()

{

Node<T>\* node = Top;

cout << "[ ";

while (node != NULL)

{

cout << node->data << ' ';

node = node->next;

}

cout << "]";

cout << endl;

}

T Peek()

{

return Top->data;

}

};

class Calculator

{

private:

string expr;

int Precedence(char opt) // BODMAS

{

if (opt == '+' || opt == '-')

{

return 1;

}

else if (opt == '\*' || opt == '/')

{

return 2;

}

else if (opt == '^')

{

return 3;

}

else

{

return 0;

}

}

void ConvertToPostfix()

{

Stack<char> stack;

string result;

for (int i = 0; i < expr.length(); i++)

{

if (IS\_DIGIT(expr[i]))

{

result += expr[i];

}

else if (expr[i] == '(')

{

stack.Push('(');

}

else if (expr[i] == ')')

{

while (!stack.IsEmpty() && stack.Peek() != '(')

{

result += stack.Peek();

stack.Pop();

}

if (stack.Peek() == '(')

{

stack.Pop();

}

}

else

{

while (!stack.IsEmpty() && Precedence(expr[i]) <= Precedence(stack.Peek()))

{

result += stack.Peek();

stack.Pop();

}

stack.Push(expr[i]);

}

}

while (!stack.IsEmpty())

{

result += stack.Peek();

stack.Pop();

}

expr = result;

}

int GetResult(int lhs, int rhs, char opt)

{

int result = 0;

if (opt == '+')

result = lhs + rhs;

else if(opt == '-')

result = lhs - rhs;

else if(opt == '\*')

result = lhs \* rhs;

else if(opt == '/')

result = lhs / rhs;

else if(opt == '^')

result = lhs ^ rhs;

return result;

}

public:

Calculator()

{

expr = "";

}

Calculator(string data)

{

this->expr = data;

}

int EvaluteExpression()

{

if (expr == "")

{

cout << "Calculator::EvaluteExpression(), `expr` is empty !!!" << endl;

}

else

{

Stack<int> stack;

ConvertToPostfix();

int lhs, rhs;

for (int i = 0; i < expr.length(); i++)

{

if (IS\_DIGIT(expr[i]))

{

stack.Push(expr[i] - '0');

}

else

{

lhs = stack.Peek();

stack.Pop();

rhs = stack.Peek();

stack.Pop();

stack.Push(GetResult(rhs, lhs, expr[i]));

}

}

int result\_value = stack.Peek();

stack.Pop();

return result\_value;

}

return 0;

}

void SetExpression(string expr)

{

this->expr = expr;

}

string GetExpression()

{

return this->expr;

}

};

int main()

{

Calculator\* calculator = new Calculator();

string Expression = "(7+(6\*3+2)-(6/3))";

calculator->SetExpression(Expression);

int Result = calculator->EvaluteExpression();

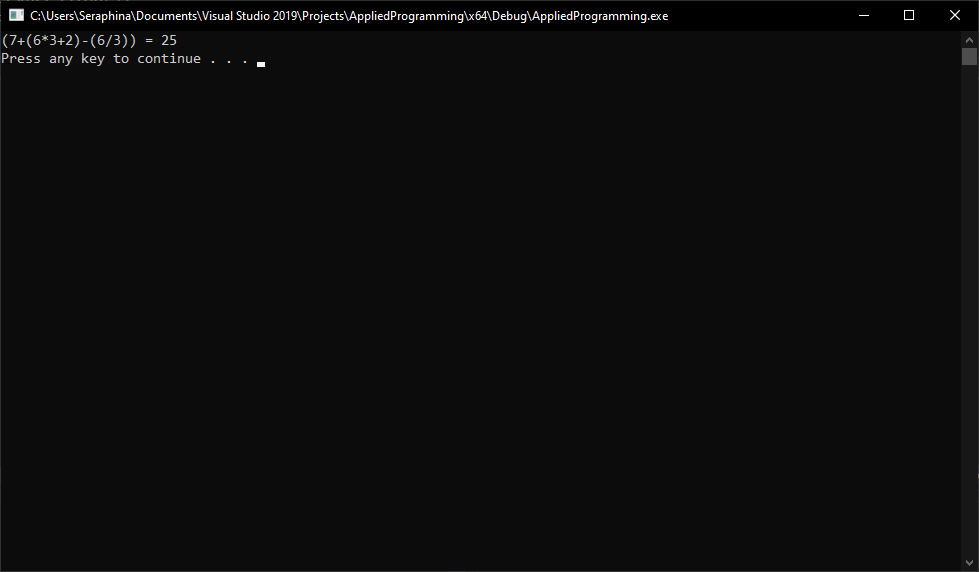
cout << Expression << " = " << Result << endl;

system("PAUSE");

return 0;

}

## **Screenshot**



# **Question#2**

Sequence Mutation

## **Code**

#include <iostream>

#include <string>

using namespace std;

template <class T>

struct Node

{

T data;

Node<T>\* next;

};

template <class T>

class Stack

{

private:

Node<T>\* Top;

public:

Stack()

{

Top = NULL; // Initialize Top to NULL

}

bool Push(T value)

{

Node<T>\* node = new Node<T>();

if (node == NULL) // If memory alloc() failed.

{

return false;

}

node->data = value;

node->next = Top;

Top = node;

return true;

}

void Pop()

{

if (!IsEmpty()) // if !(Top == NULL)

{

Node<T>\* node = Top;

Top = Top->next;

delete node;

}

}

bool IsEmpty()

{

return (Top == NULL);

}

void Display()

{

Node<T>\* node = Top;

cout << "[ ";

while (node != NULL)

{

cout << node->data << ' ';

node = node->next;

}

cout << "]";

cout << endl;

}

T Peek()

{

return Top->data;

}

};

#define LOWER(c) (c >= 'a' && c <= 'z')

#define UPPER(c) (c >= 'A' && c <= 'Z')

#define IS\_ALPHA(c) (LOWER(c) || UPPER(c))

string sequenceMutation(string sequence)

{

Stack<string> stack;

string word = "";

for (int i = 0; sequence[i] != '\0'; i++)

{

if (IS\_ALPHA(sequence[i]))

{

word += sequence[i];

}

else

{

if (word != "")

{

stack.Push(" ");

stack.Push(word);

}

word = "";

}

}

stack.Push(word);

word = "";

while (!stack.IsEmpty())

{

word += stack.Peek();

stack.Pop();

}

return word;

}

int main()

{

string Sequence = "Hurray! Live for assignment.";

// cout << "Enter Something: ";

// getline(cin, Sequence);

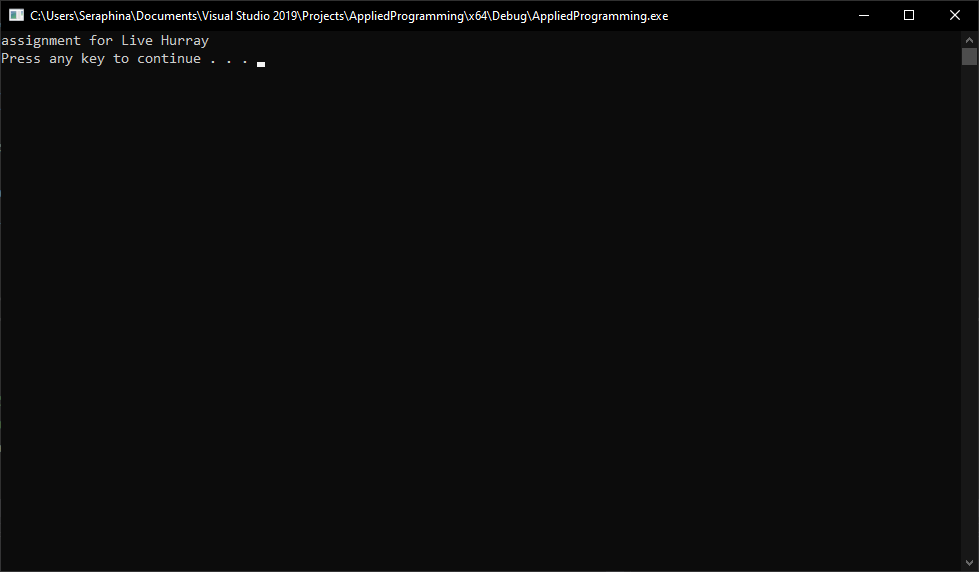
cout << sequenceMutation(Sequence) << endl;

system("PAUSE");

return 0;

}

## **Screenshot**



# **Question#3**

Organizing plates using Stack ADT

## **Code**

/\*

Code taken from

https://www.codespeedy.com/program-for-tower-of-hanoi-using-stack-in-cpp/#:~:text=C%2B%2B%20Program%3A%20Tower%20of%20Hanoi%20using%20stack&text=We%20have%20used%20the%20inbuilt,for%20the%20number%20of%20disks.

\*/

#include <iostream>

#include <math.h>

#include <string>

using namespace std;

template <class T>

struct Node

{

T data;

Node\* next;

};

template <class T>

class Stack

{

private:

Node<T>\* Top;

public:

Stack()

{

Top = NULL; // Initialize Top to NULL

}

bool Push(T value)

{

Node<T>\* node = new Node<T>();

if (node == NULL) // If memory alloc() failed.

{

return false;

}

node->data = value;

node->next = Top;

Top = node;

return true;

}

T Pop()

{

if (IsEmpty()) // if Top == NULL

{

return NULL;

}

Node<T>\* node = Top;

Top = Top->next;

T result\_value = node->data;

delete node;

return result\_value;

}

bool IsEmpty()

{

return (Top == NULL);

}

void Display()

{

Node<T>\* node = Top;

cout << "[ ";

while (node != NULL)

{

cout << node->data << ' ';

node = node->next;

}

cout << "]";

cout << endl;

}

T Peek()

{

return Top->data;

}

};

int TransferDisk(Stack<int>& stack\_1, Stack<int>& stack\_2)

{

if (stack\_2.IsEmpty())

{

stack\_2.Push(stack\_1.Peek());

stack\_1.Pop();

return 1;

}

else if (stack\_1.IsEmpty())

{

stack\_1.Push(stack\_2.Peek());

stack\_2.Pop();

return 2;

}

else

{

if (stack\_2.Peek() > stack\_1.Peek())

{

stack\_2.Push(stack\_1.Peek());

stack\_1.Pop();

return 1;

}

else {

stack\_1.Push(stack\_2.Peek());

stack\_2.Pop();

return 2;

}

}

}

int main()

{

Stack<int> source, auxiliary, destination;

int input = 0;

cin >> input;

for (int i = input; i >= 1; i--)

{

source.Push(i);

}

int x = pow(2, input) - 1;

int i = 1;

if (input % 2 == 0)

{

while (i <= x)

{

if (i % 3 == 1)

{

int y = TransferDisk(source, auxiliary);

if (y == 1)

{

cout << "Move the disk " << auxiliary.Peek() << " from source to auxiliary" << endl;

}

else

{

cout << "Move the disk " << source.Peek() << " from auxiliary to source" << endl;

}

}

else if (i % 3 == 2) {

int y = TransferDisk(source, destination);

if (y == 1) {

cout << "Move the disk " << destination.Peek() << " from source to destination" << endl;

}

else

cout << "Move the disk " << source.Peek() << " from destination to source" << endl;

}

else

{

int y = TransferDisk(auxiliary, destination);

if (y == 1)

{

cout << "Move the disk " << destination.Peek() << " from auxiliary to destination" << endl;

}

else

{

cout << "Move the disk " << auxiliary.Peek() << " from destination to auxiliary" << endl;

}

}

i++;

}

}

else

{

while (i <= x)

{

if (i % 3 == 1)

{

int y = TransferDisk(source, destination);

if (y == 1)

{

cout << "Move the disk " << destination.Peek() << " from source to destination" << endl;

}

else

{

cout << "Move the disk " << source.Peek() << " from destination to source" << endl;

}

}

else if (i % 3 == 2)

{

int y = TransferDisk(source, auxiliary);

if (y == 1)

{

cout << "Move the disk " << auxiliary.Peek() << " from source to auxiliary" << endl;

}

else

{

cout << "Move the disk " << source.Peek() << " from auxiliary to source" << endl;

}

}

else

{

int y = TransferDisk(auxiliary, destination);

if (y == 1)

{

cout << "Move the disk " << destination.Peek() << " from auxiliary to destination" << endl;

}

else

{

cout << "Move the disk " << auxiliary.Peek() << " from destination to auxiliary" << endl;

}

}

i++;

}

}

while (!destination.IsEmpty())

{

cout << destination.Peek() << endl;

destination.Pop();

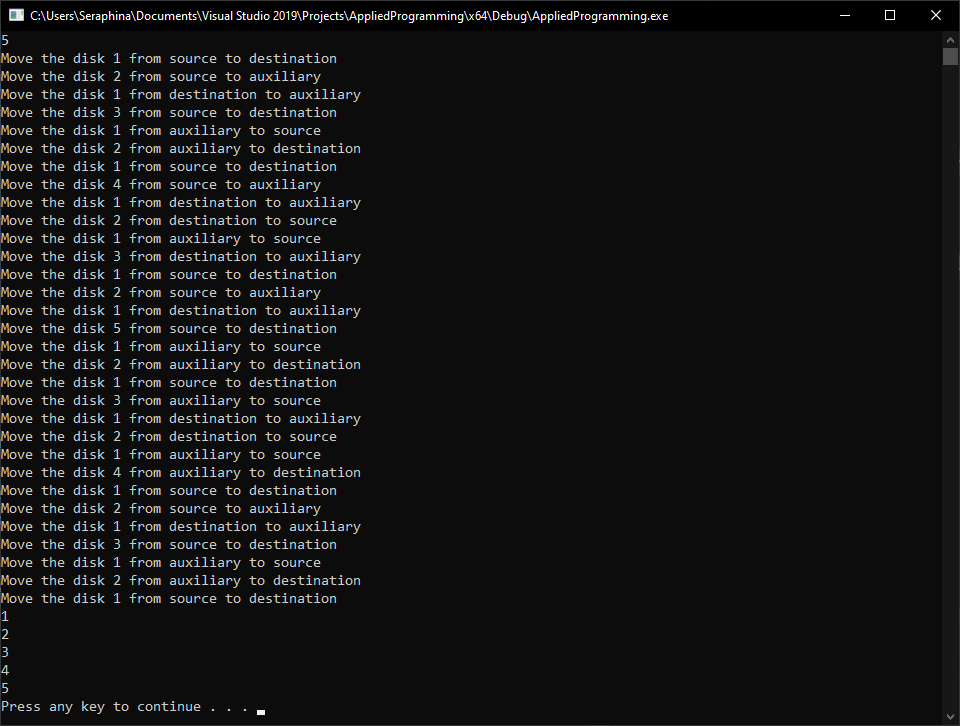
}

system("PAUSE");

return 0;

}

## **Screenshot**



# **Question#4**

Traffic management using DEQUE

## **Code**

#include <iostream>

#include <string>

using namespace std;

struct Node

{

int data;

Node\* next, \*prev;

};

class DoubleEndedQueue

{

private:

Node\* Head, \*Tail;

int Limit, CurrentSize;

bool CanInsert()

{

if (Limit == -1 || CurrentSize < Limit)

return true;

return false;

}

public:

DoubleEndedQueue()

{

Head = Tail = NULL;

Limit = -1; // -1 means no limit

CurrentSize = 0;

}

void SetLimit(int limit)

{

this->Limit = limit;

}

void InsertAtHead(int data)

{

if (!CanInsert())

{

cout << "DoubleEndedQueue::InsertAtHead(), Queue size is full !!!" << endl;

return;

}

Node\* node = new Node();

if (node == NULL)

{

cout << "Unable to allocate() memory for `Node` !!!" << endl;

}

else

{

node->data = data;

if (Head == NULL)

{

node->next = node->prev = NULL;

Head = Tail = node;

}

else

{

node->next = Head;

Head->prev = node;

Head = node;

}

if (Limit != -1)

{

CurrentSize++;

}

}

}

void InsertAtTail(int data)

{

if (!CanInsert())

{

cout << "DoubleEndedQueue::InsertAtTail(), Queue size is full !!!" << endl;

return;

}

Node\* node = new Node();

if (node == NULL)

{

cout << "Unable to allocate() memory for `Node` !!!" << endl;

}

else

{

node->data = data;

if (Tail == NULL)

{

Head = Tail = node;

}

else

{

node->prev = Tail;

Tail->next = node;

Tail = node;

}

if (Limit != -1)

{

CurrentSize++;

}

}

}

void DeleteFromHead()

{

if (IsEmpty())

{

cout << "DoubleEndedQueue::Queue is empty !!!" << endl;

}

else

{

Node\* node = Head;

Head = Head->next;

if (Head == NULL)

{

Tail = NULL;

}

else

{

Head->prev = NULL;

}

delete node;

if (Limit != -1)

{

CurrentSize--;

}

}

}

void DeleteFromTail()

{

if (IsEmpty())

{

cout << "DoubleEndedQueue::Queue is empty !!!" << endl;

}

else

{

Node\* node = Tail;

Tail = Tail->prev;

if (Tail == NULL)

{

Head = NULL;

}

else

{

Tail->next = NULL;

}

delete node;

if (Limit != -1)

{

CurrentSize--;

}

}

}

int GetHead()

{

if (IsEmpty())

return -1;

return Head->data;

}

int GetTail()

{

if (IsEmpty())

return -1;

return Tail->data;

}

int GetSize()

{

return this->CurrentSize;

}

bool IsEmpty()

{

return (Head == NULL);

}

void Display()

{

Node\* temp = Head;

while (temp != NULL)

{

cout << temp->data << " ";

temp = temp->next;

}

}

};

void Menu()

{

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\t1. Insert At Head" << endl;

cout << "\t2. Insert At Tail" << endl;

cout << "\t3. Remove From Head" << endl;

cout << "\t4. Remove From Tail" << endl;

cout << "\t5. Total Cars" << endl;

cout << "\t6. Display" << endl;

cout << "\t7. Exit" << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

}

int main()

{

DoubleEndedQueue\* double\_ended\_queue = new DoubleEndedQueue();

double\_ended\_queue->SetLimit(5);

int choice = 0;

do

{

Menu();

do

{

cout << "Enter Choice Between (1-7): ";

cin >> choice;

} while (choice < 1 || choice > 7);

switch (choice)

{

case 1:

double\_ended\_queue->InsertAtHead(double\_ended\_queue->GetSize());

break;

case 2:

double\_ended\_queue->InsertAtTail(double\_ended\_queue->GetSize());

break;

case 3:

double\_ended\_queue->DeleteFromHead();

break;

case 4:

double\_ended\_queue->DeleteFromTail();

break;

case 5:

cout << double\_ended\_queue->GetSize() << endl;

break;

case 6:

double\_ended\_queue->Display();

cout << endl;

break;

case 7:

cout << "You Exit the program...\n";

break;

}

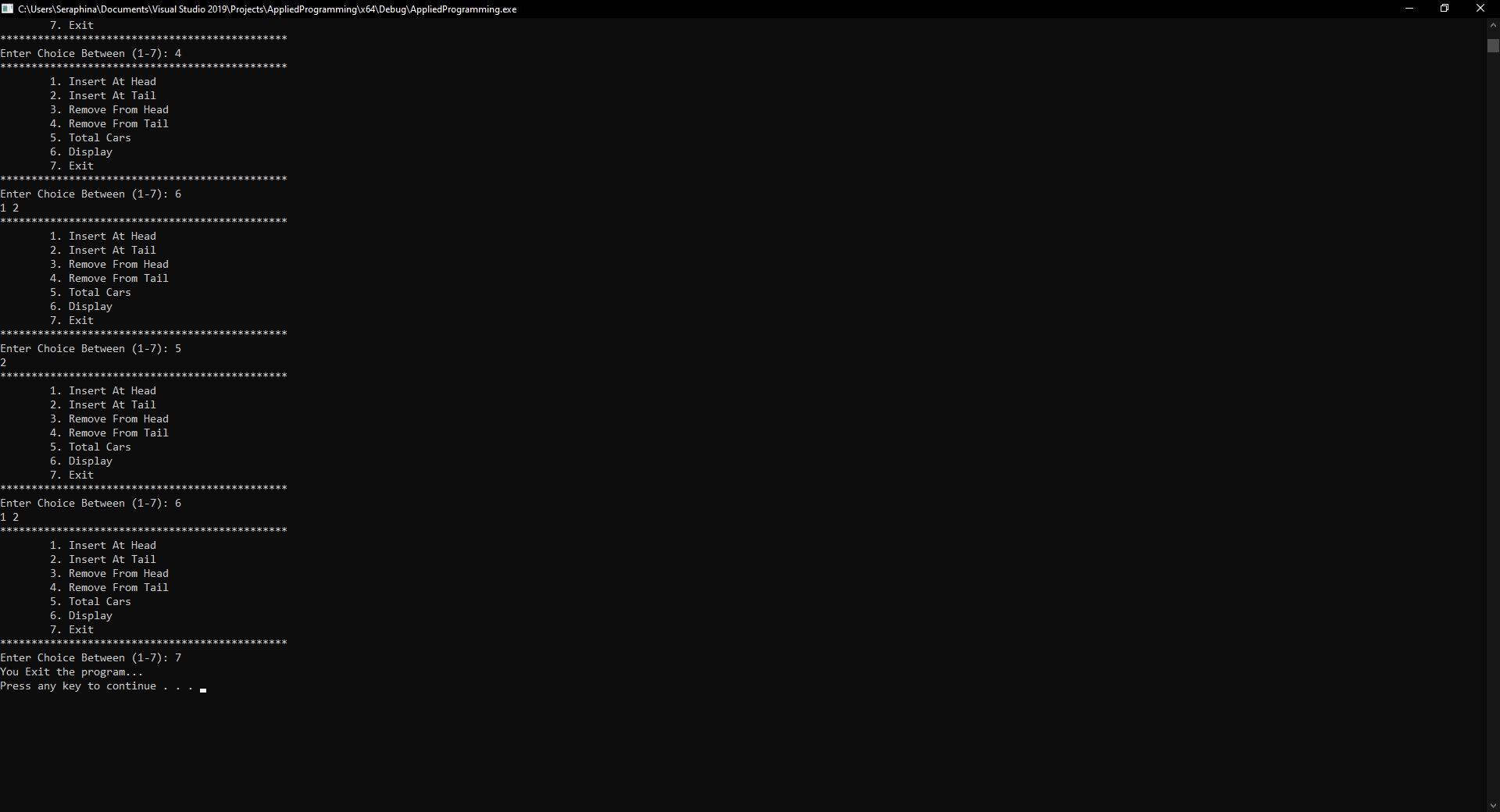
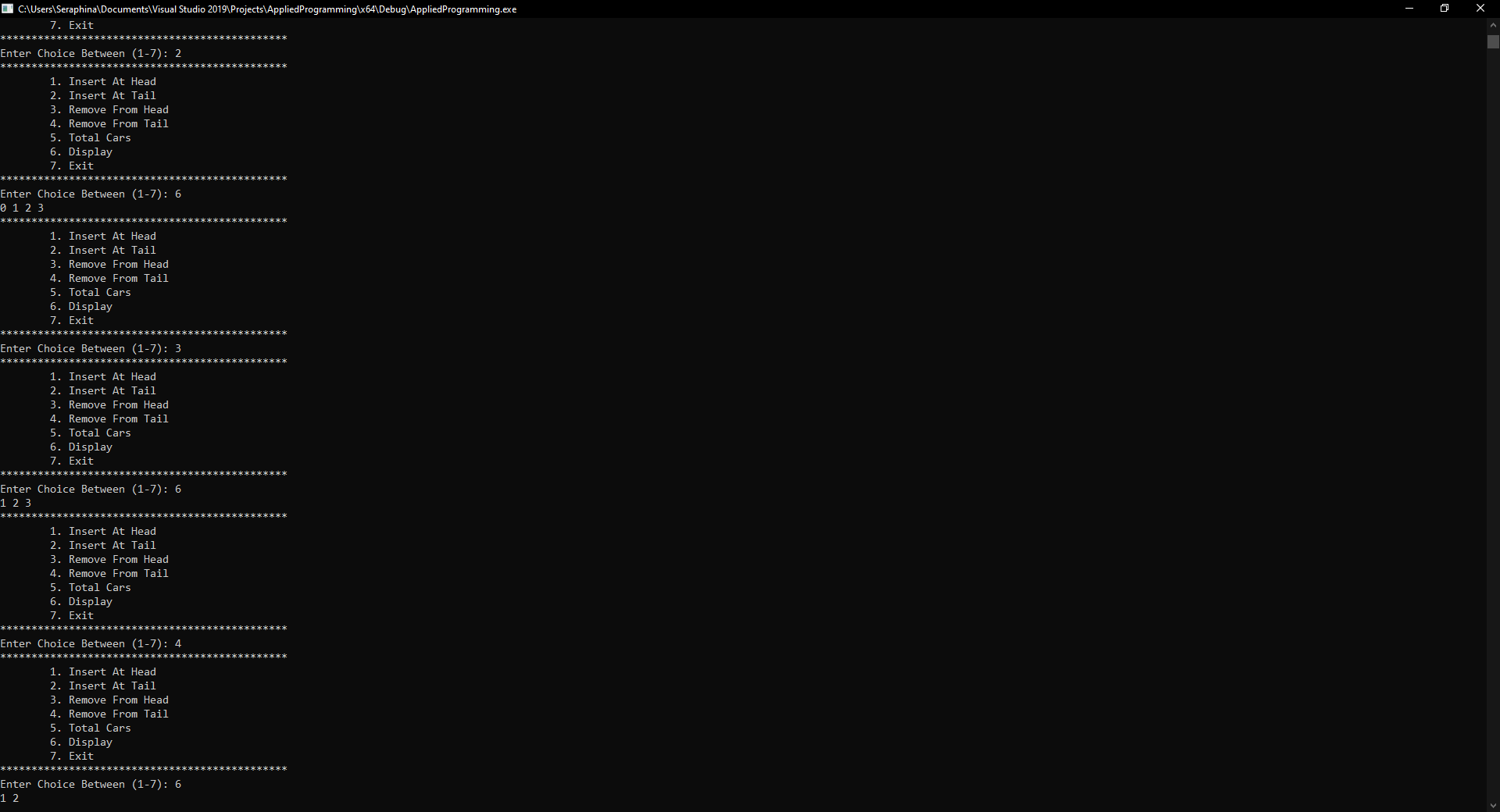
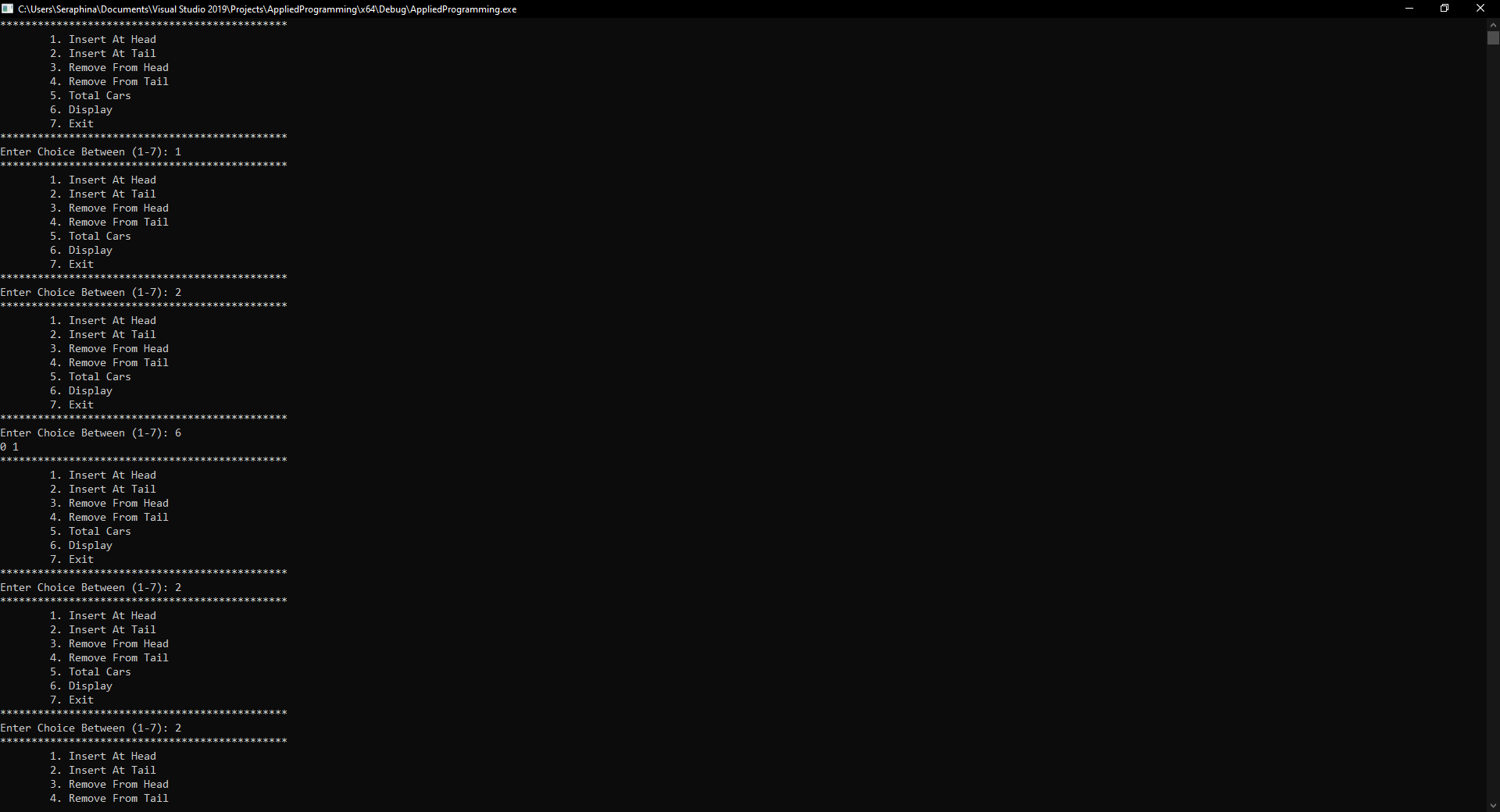
} while (choice != 7);

system("PAUSE");

return 0;

}

## **Screenshot**



# **Question#5**

Senate win prediction using Queue ADT

## **Code**

#include<iostream>

#include <string>

using namespace std;

struct Node

{

char value;

Node\* next;

};

class Queue

{

private:

Node \*Head, \*Tail;

public:

Queue()

{

Head = Tail = NULL;

}

bool IsEmpty()

{

return (Head == NULL);

}

void Push(char value)

{

Node\* node = new Node;

node->value = value;

node->next = NULL;

if (IsEmpty())

{

Head = Tail = node;

}

else

{

Tail->next = node;

Tail = node;

}

}

void Pop()

{

if (IsEmpty())

{

cout << "Queue is Empty!!!\n";

return;

}

Node\* node = Head;

Head = Head->next;

delete node;

}

char Top()

{

return Head->value;

}

};

int main()

{

Queue queue;

int total\_senators = 0;

cout << "Number of Senators: " << endl;

cin >> total\_senators;

cin.ignore();

string str;

cout << "Enter your Input: " << endl;

getline(cin, str);

if (str.length() != total\_senators)

{

cout << "Error!!!\n" << endl;

return 0;

}

for (int i = 0; i < total\_senators; i++)

{

queue.Push(str[i]);

}

char c;

int oppositions = 0, governments = 0;

for (int i = 0; i < total\_senators; i++)

{

c = queue.Top();

if (c == 'G' || c == 'g')

{

governments++;

}

else if(c == 'O' || c == 'o')

{

oppositions++;

}

queue.Pop();

}

if (governments > oppositions)

{

cout << "Government\n";

cout << "Because the number of votes of governmnet are greater then the number of votes of opposition.\n";

cout << "Or the first vote is from government " << endl;

}

else

{

cout << "Opposition\n" << endl;

cout << "Because the number of votes of opposition are greater then the number of votes of government.\n";

cout << "Or the first vote is from oppostion\n";

}

system("PAUSE");

return 0;

}

## **Screenshot**



# **Question#6**

Student Facilitation center management system using Queue ADT

## **Code**

## **Screenshot**

# **Question#7**

Bank queue management using Queue ADT

## **Code**

## **Screenshot**