Name :- Aryan Juneja Roll no. :- 1024030920 Batch 2C64

Q1

#include <iostream>

using namespace std;

class Stack {

int arr[10];

int top;

public:

Stack() {

top = -1;

}

void push(int num) {

if (isFull()) {

cout << "Stack Overflow! Cannot push " << num << "\n";

return;

}

arr[++top] = num;

cout << num << " pushed into stack.\n";

}

void pop() {

if (isEmpty()) {

cout << "Stack Underflow! Cannot pop.\n";

return;

}

cout << arr[top--] << " popped from stack.\n";

}

bool isEmpty() {

return top == -1;

}

bool isFull() {

return top == 9;

}

void display() {

if (isEmpty()) {

cout << "Stack is empty.\n";

return;

}

cout << "<===== Stack Elements =====>\n";

for (int i = top; i >= 0; i--) {

cout << arr[i] << "\n";

}

}

void peek() {

if (isEmpty()) {

cout << "Stack is empty. Nothing to peek.\n";

return;

}

cout << "Top element: " << arr[top] << "\n";

}

};

int main() {

Stack stackk;

int option, num;

bool running = true;

while (running) {

cout << "\nMenu:\n";

cout << "1 - Push\n";

cout << "2 - Pop\n";

cout << "3 - Peek\n";

cout << "4 - Display\n";

cout << "5 - Check if Empty\n";

cout << "6 - Check if Full\n";

cout << "7 - Exit\n";

cout << "Enter choice: ";

cin >> option;

switch (option) {

case 1:

cout << "Enter number to push: ";

cin >> num;

stackk.push(num);

stackk.display();

break;

case 2:

stackk.pop();

break;

case 3:

stackk.peek();

break;

case 4:

stackk.display();

break;

case 5:

cout << (stackk.isEmpty() ? "Stack is empty.\n" : "Stack is not empty.\n");

break;

case 6:

cout << (stackk.isFull() ? "Stack is full.\n" : "Stack is not full.\n");

break;

case 7:

running = false;

cout << "Exiting program.\n";

break;

default:

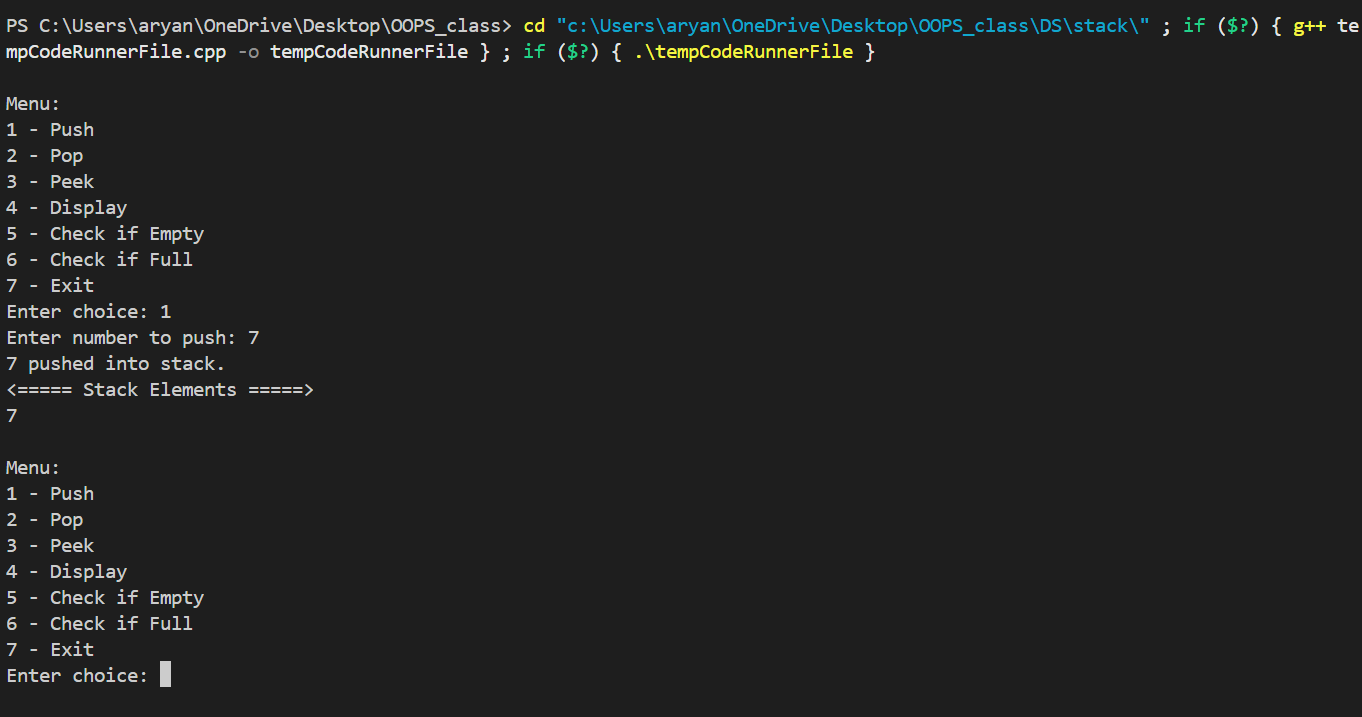
cout << "Invalid choice! Try again.\n";

}

}

return 0;

}



Q2

#include <iostream>

#include <string>

#include <stack>

using namespace std;

void Reversed\_String(string &ch)

{

int n = ch.length();

stack<char> c;

for (int i = 0; i < n; i++)

{

c.push(ch[i]);

}

while (!c.empty())

{

cout << c.top();

c.pop();

}

}

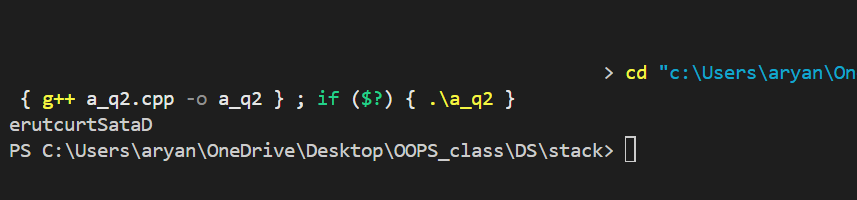
int main()

{

string ch = "DataStructure";

Reversed\_String(ch);

}



Q3

#include <iostream>

#include <stack>

#include <string>

using namespace std;

bool is\_pair(char opening, char closing)

{

if (opening == '(' && closing == ')')

{

return true;

}

else if (opening == '[' && closing == ']')

{

return true;

}

else if (opening == '{' && closing == '}')

{

return true;

}

else

{

return false;

}

}

bool is\_parentheses(string &exp)

{

stack<char> ch;

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

{

if (exp[i] == '(' || exp[i] == '{' || exp[i] == '[')

{

ch.push(exp[i]);

}

else if (exp[i] == ')' || exp[i] == '}' || exp[i] == ']')

{

if (ch.empty() || (!is\_pair(ch.top(), exp[i])))

{

return false;

}

else

{

ch.pop();

}

}

}

return ch.empty();

}

int main()

{

string expr = "{([}])";

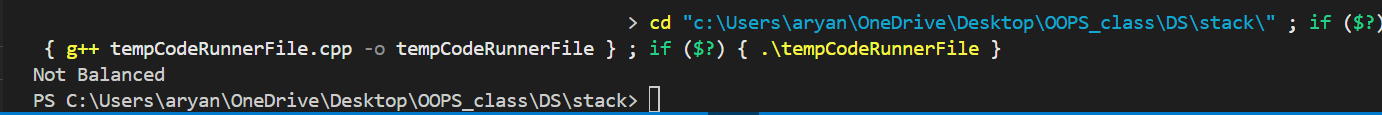
if (is\_parentheses(expr))

cout << "Balanced\n";

else

cout << "Not Balanced\n";

}



Q4

#include <iostream>

#include <stack>

#include <string>

using namespace std;

int precedence(char op) {

if (op == '^') return 3;

if (op == '\*' || op == '/') return 2;

if (op == '+' || op == '-') return 1;

return 0;

}

bool isOperator(char ch) {

return ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == '^';

}

string infixToPostfix(string infix) {

stack<char> st;

string postfix;

for (char ch : infix) {

if (isalnum(ch)) {

// Operand: directly add to output

postfix += ch;

} else if (ch == '(') {

st.push(ch);

} else if (ch == ')') {

// Pop until '('

while (!st.empty() && st.top() != '(') {

postfix += st.top();

st.pop();

}

if (!st.empty()) st.pop(); // Remove '('

} else if (isOperator(ch)) {

// Pop operators with higher or equal precedence

while (!st.empty() && precedence(st.top()) >= precedence(ch)) {

postfix += st.top();

st.pop();

}

st.push(ch);

}

}

while (!st.empty()) {

postfix += st.top();

st.pop();

}

return postfix;

}

int main() {

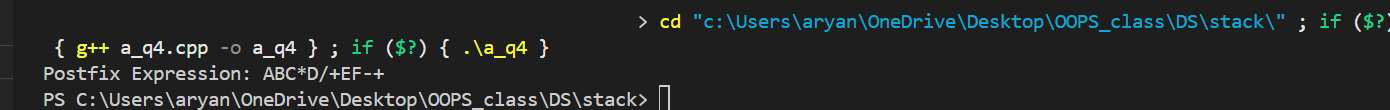
string infix = "A+B\*C/D+(E-F)";

string postfix = infixToPostfix(infix);

cout << "Postfix Expression: " << postfix << endl;

return 0;

}



Q5

#include <iostream>

#include <stack>

#include <cmath>

using namespace std;

int evaluatePostfix(string &s)

{

stack<int> st;

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

{

char c = s[i];

if (isdigit(c))

{

st.push(c - '0');

}

else

{

int num1 = st.top();

st.pop();

int num2 = st.top();

st.pop();

switch (c)

{

case '+':

st.push(num1 + num2);

break;

case '-':

st.push(num1 - num2);

break;

case '\*':

st.push(num1 \* num2);

break;

case '/':

st.push(num1 / num2);

break;

case '^':

st.push(pow(num1, num2));

break;

}

}

}

return st.top();

}

int main()

{

string postfix = "23+5\*3+";

cout << "Postfix Evaluation: " << evaluatePostfix(postfix) << endl;

return 0;

}

