Lab 5

Q1:

#include<iostream>

using namespace std;

const int STACKSIZE = 5;

class stack{

private:

int stackArray[STACKSIZE];

int TOP;

public:

stack(){

TOP = -1;

}

bool isEmpty(){

if(TOP == -1){

return true;

}else{

return false;

}

}

bool isFull(){

if (TOP == STACKSIZE - 1)

{

return true;

}else{

return false;

}

}

void Push(int element){

if (isFull() == true)

{

cout<<"Can't insert Element because the stack is already Full";

}else{

TOP++;

stackArray[TOP] = element;

}

}

void Pop(){

if (isEmpty() == true)

{

cout<<"Cannot remove the element Because the stack is already Empty";

}else{

TOP--;

}

}

int size(){

return TOP +1;

}

void DisplayStack(){

if (TOP == 0)

{

cout<<"No element in the Stack";

return;

}else{

for (int i = size()-1; i >=0 ; i--)

{

cout<<stackArray[i]<<endl;

}

}

}

};

int main(){

stack st;

st.Push(4);

st.Push(5);

st.Push(6);

st.DisplayStack();

st.Pop();

cout<<endl;

cout<<endl;

st.DisplayStack();

}

Q2:

#include<iostream>

using namespace std;

const int STACKSIZE = 5;

class stack{

private:

int stackArray[STACKSIZE];

int TOP;

public:

stack(){

TOP = -1;

}

bool isEmpty(){

if(TOP == -1){

return true;

}else{

return false;

}

}

bool isFull(){

if (TOP == STACKSIZE - 1)

{

return true;

}else{

return false;

}

}

void Push(int element){

if (isFull() == true)

{

cout<<"Can't insert Element because the stack is already Full";

}else{

TOP++;

stackArray[TOP] = element;

}

}

void Pop(){

if (isEmpty() == true)

{

cout<<"Cannot remove the element Because the stack is already Empty";

}else{

TOP--;

}

}

int size(){

return TOP +1;

}

void DisplayStack(){

if (TOP == 0)

{

cout<<"No element in the Stack";

return;

}else{

for (int i = size()-1; i >=0 ; i--)

{

cout<<stackArray[i]<<endl;

}

}

}

};

int main(){

stack st;

st.Push(100);

st.Push(200);

st.Push(300);

st.Push(400);

st.Push(500);

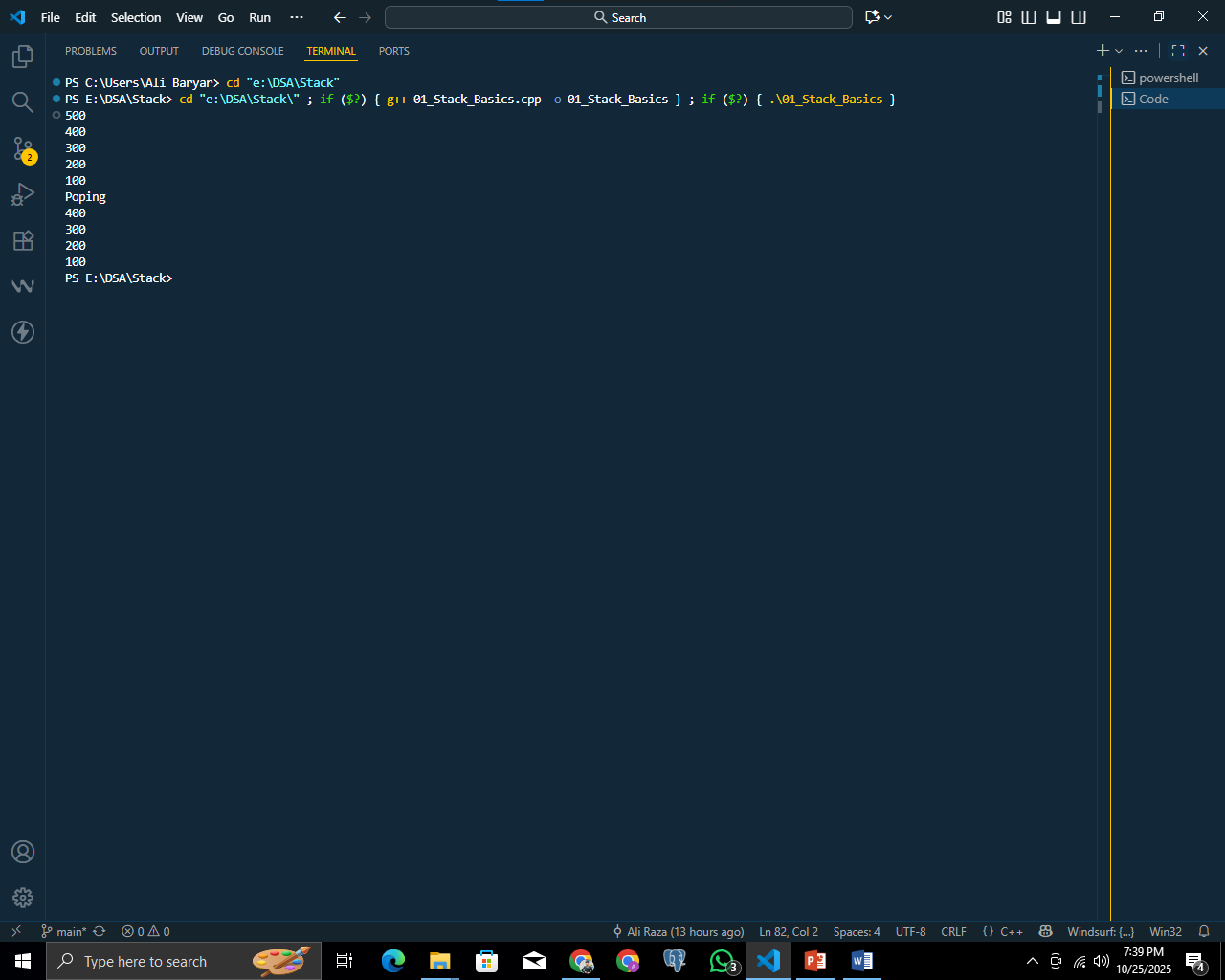
st.DisplayStack();

st.Pop();

cout<<"Poping "<<endl;

st.DisplayStack();

}



Q3:

#include <iostream>

#include <cctype>

using namespace std;

const int STACKSIZE = 100;

class Stack {

private:

char stackArray[STACKSIZE];

int TOP;

public:

Stack() {

TOP = -1;

}

bool isEmpty() {

return (TOP == -1);

}

bool isFull() {

return (TOP == STACKSIZE - 1);

}

void Push(char element) {

if (isFull()) {

cout << "Stack Overflow! Cannot push " << element << endl;

} else {

TOP++;

stackArray[TOP] = element;

}

}

void Pop() {

if (isEmpty()) {

cout << "Stack Underflow! Cannot pop." << endl;

} else {

TOP--;

}

}

char Top() {

if (!isEmpty())

return stackArray[TOP];

else

return '\0';

}

};

int precedence(char op) {

if (op == '^')

return 3;

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

return 2;

else if (op == '+' || op == '-')

return 1;

else

return 0;

}

bool isOperator(char c) {

return (c == '+' || c == '-' || c == '\*' || c == '/' || c == '^');

}

string infixToPostfix(string infix) {

Stack st;

string postfix = "";

for (char c : infix) {

if (isalnum(c)) {

postfix += c;

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

st.Push(c);

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

while (!st.isEmpty() && st.Top() != '(') {

postfix += st.Top();

st.Pop();

}

st.Pop();

} else if (isOperator(c)) {

while (!st.isEmpty() && precedence(st.Top()) >= precedence(c)) {

postfix += st.Top();

st.Pop();

}

st.Push(c);

}

}

while (!st.isEmpty()) {

postfix += st.Top();

st.Pop();

}

return postfix;

}

int main() {

string infix;

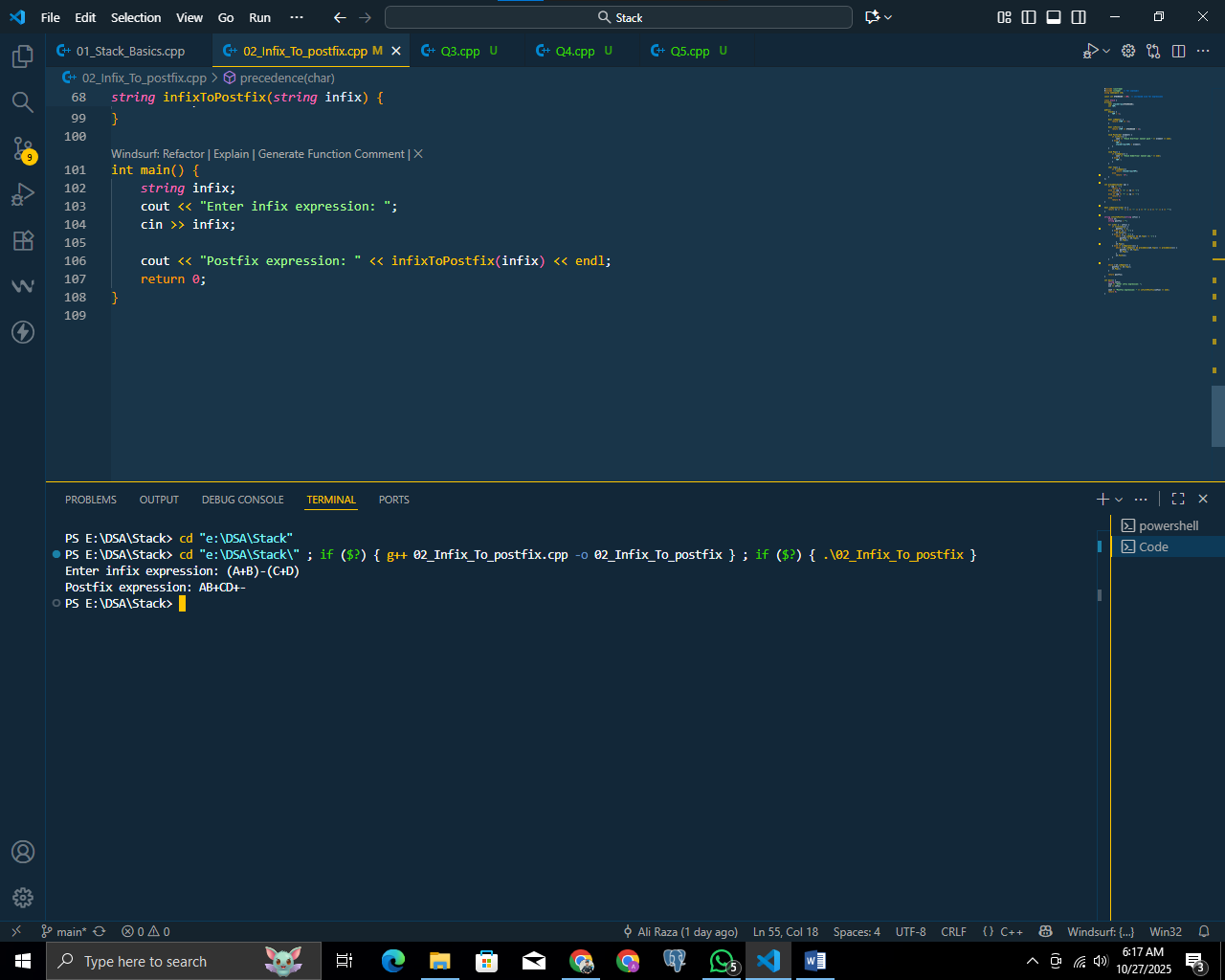
cout << "Enter infix expression: ";

cin >> infix;

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

return 0;

}



Q3:

#include <iostream>

#include <cmath>

#include <cctype>

using namespace std;

#define STACKSIZE 50

class Stack {

private:

char stackArray[STACKSIZE];

int TOP;

public:

Stack() {

TOP = -1;

}

bool isEmpty() {

return (TOP == -1);

}

bool isFull() {

return (TOP == STACKSIZE - 1);

}

void Push(char element) {

if (isFull()) {

cout << "Stack Overflow! Cannot push " << element << endl;

} else {

TOP++;

stackArray[TOP] = element;

}

}

void Pop() {

if (isEmpty()) {

cout << "Stack Underflow! Cannot pop." << endl;

} else {

TOP--;

}

}

char Top() {

if (!isEmpty())

return stackArray[TOP];

else

return '\0';

}

};

int evaluatePostfix(string expression) {

Stack s;

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

char ch = expression[i];

if (isdigit(ch)) {

s.Push(ch);

}

else {

int val2 = s.Top() - '0'; s.Pop();

int val1 = s.Top() - '0'; s.Pop();

int result;

switch (ch) {

case '+': result = val1 + val2; break;

case '-': result = val1 - val2; break;

case '\*': result = val1 \* val2; break;

case '/': result = val1 / val2; break;

case '^': result = pow(val1, val2); break;

default:

cout << "Invalid operator: " << ch << endl;

return -1;

}

s.Push(result + '0');

}

}

return s.Top() - '0';

}

int main() {

string postfix;

cout << "Enter a postfix expression (e.g., 23\*54\*+9-): ";

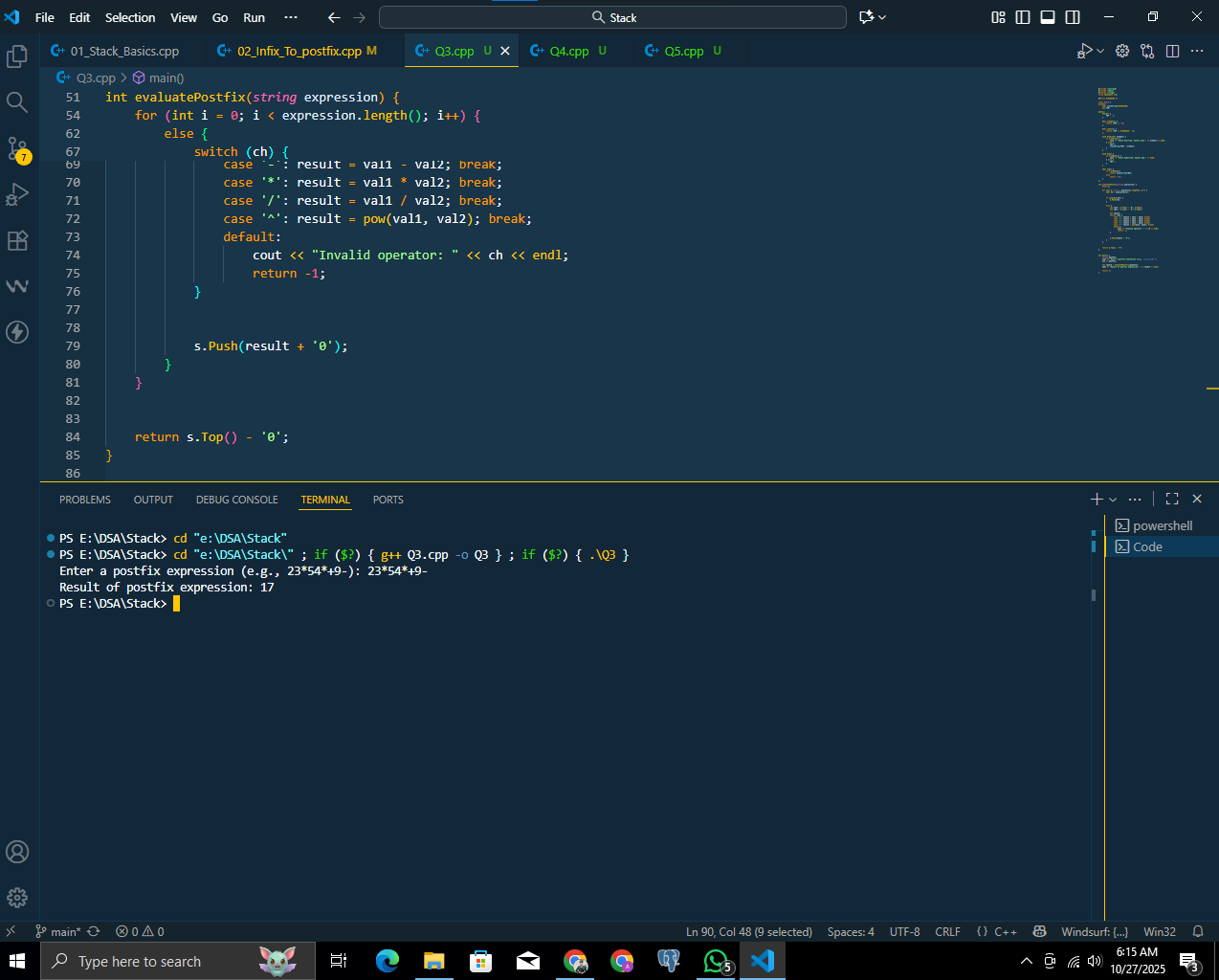
cin >> postfix;

int result = evaluatePostfix(postfix);

cout << "Result of postfix expression: " << result << endl;

return 0;

}



Q4:

#include <iostream>

using namespace std;

#define STACKSIZE 50

class Stack {

private:

char stackArray[STACKSIZE];

int TOP;

public:

Stack() {

TOP = -1;

}

bool isEmpty() {

return (TOP == -1);

}

bool isFull() {

return (TOP == STACKSIZE - 1);

}

void Push(char element) {

if (isFull()) {

cout << "Stack Overflow! Cannot push " << element << endl;

} else {

TOP++;

stackArray[TOP] = element;

}

}

void Pop() {

if (isEmpty()) {

cout << "Stack Underflow! Cannot pop." << endl;

} else {

TOP--;

}

}

char Top() {

if (!isEmpty())

return stackArray[TOP];

else

return '\0';

}

};

void decimalToBinary(int decimalNumber) {

Stack s;

if (decimalNumber == 0) {

cout << "Binary: 0" << endl;

return;

}

while (decimalNumber > 0) {

int remainder = decimalNumber % 2;

s.Push(remainder + '0');

decimalNumber = decimalNumber / 2;

}

cout << "Binary: ";

while (!s.isEmpty()) {

cout << s.Top();

s.Pop();

}

cout << endl;

}

int main() {

int number;

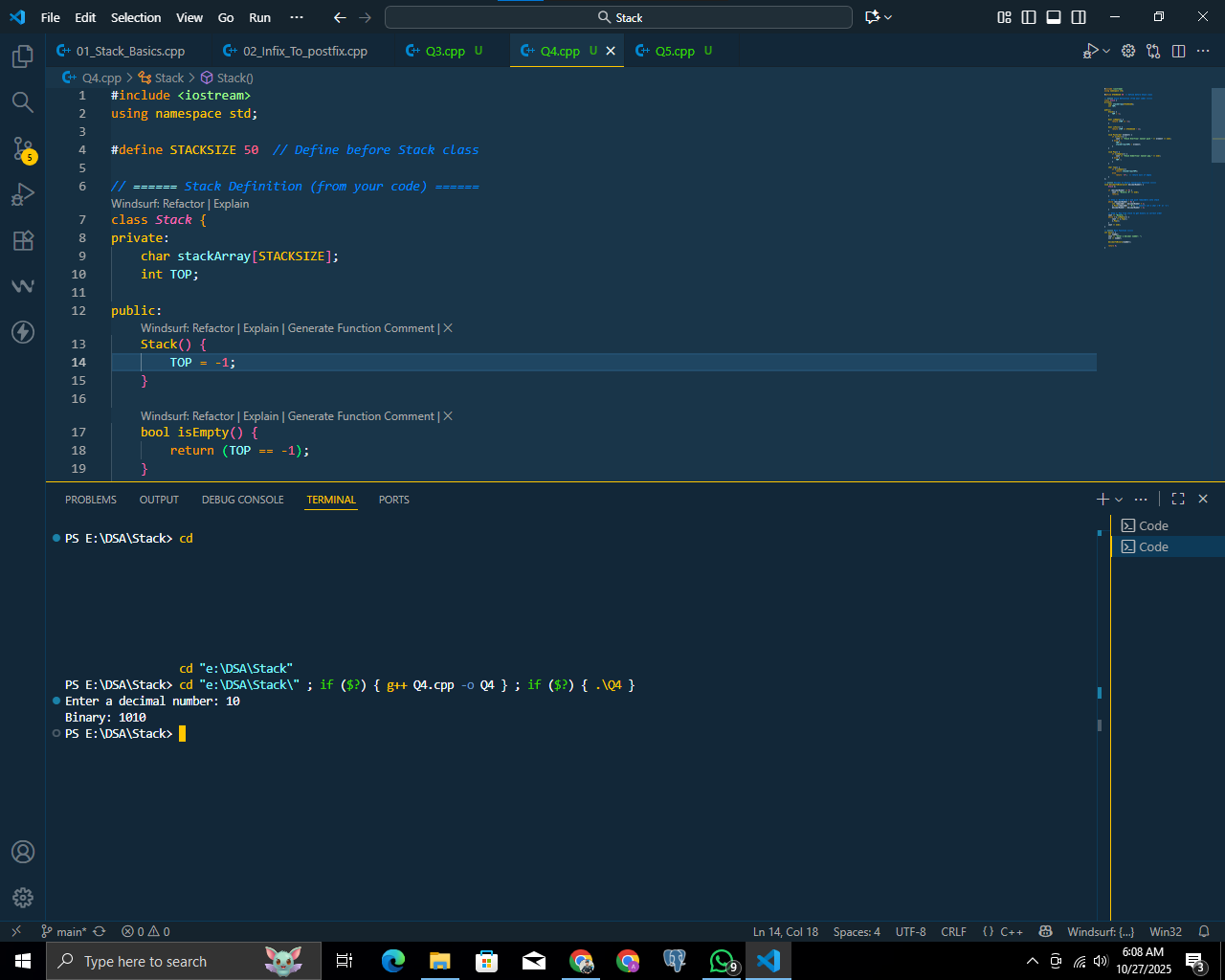
cout << "Enter a decimal number: ";

cin >> number;

decimalToBinary(number);

return 0;

}



Q5:  
#include <iostream>

using namespace std;

#define STACKSIZE 100

class Stack {

private:

char stackArray[STACKSIZE];

int TOP;

public:

Stack() {

TOP = -1;

}

bool isEmpty() {

return (TOP == -1);

}

bool isFull() {

return (TOP == STACKSIZE - 1);

}

void Push(char element) {

if (isFull()) {

cout << "Stack Overflow! Cannot push " << element << endl;

} else {

TOP++;

stackArray[TOP] = element;

}

}

void Pop() {

if (isEmpty()) {

cout << "Stack Underflow! Cannot pop." << endl;

} else {

TOP--;

}

}

char Top() {

if (!isEmpty())

return stackArray[TOP];

else

return '\0';

}

};

bool isMatchingPair(char open, char close) {

return ((open == '(' && close == ')') ||

(open == '{' && close == '}') ||

(open == '[' && close == ']'));

}

bool areSymbolsBalanced(const string& expr) {

Stack s;

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

char ch = expr[i];

if (ch == '(' || ch == '{' || ch == '[') {

s.Push(ch);

}

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

if (s.isEmpty()) {

return false;

} else if (!isMatchingPair(s.Top(), ch)) {

return false;

} else {

s.Pop();

}

}

}

return s.isEmpty();

}

int main() {

string expression;

cout << "Enter an expression with symbols: ";

cin >> expression;

if (areSymbolsBalanced(expression))

cout << "Symbols are balanced!" << endl;

else

cout << "Symbols are NOT balanced!" << endl;

return 0;

}

