

|  |  |
| --- | --- |
| Name | Shehar Bano |
| Class | BSCS(B) |
| Reg. No | 22-NTU-CS-1222 |
| Semester | 4th |
| Course Name | Data structures and Algorithms |

LAB PLAN 5

**Exercise-1:   
Following is the declaration of Stack class and its member functions. Write the code to implement the definitions of these member functions.**  
  
**Code:**  
  
#include <iostream>

using namespace std;

int const 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;

}

int Size() {

return Top + 1;

}

void Push(int value) {

if(IsFull() == true){

cout << "Can't insert Element because the Stack is Full\n";

}

else{

Top++;

StackArray[Top] = value;

}

}

void Pop() {

if(IsEmpty() == true){

cout << "Can't Pop Element because the Stack is empty\n";

}

else{

Top--;

}

}

void Display() {

if (Top == -1) {

cout << "Stack is empty!" << endl;

return;

}

else{

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

cout << StackArray[i] << " ";

}

cout << endl;

}

}

};  
  
**Exercise-2:   
Test the Stack class with the following main() function:**

int main() {

Stack stk;

stk.Push(100);

stk.Push(200);

stk.Push(300);

stk.Push(400);

stk.Push(500);

stk.Display();

cout << "\nPoping..\n";

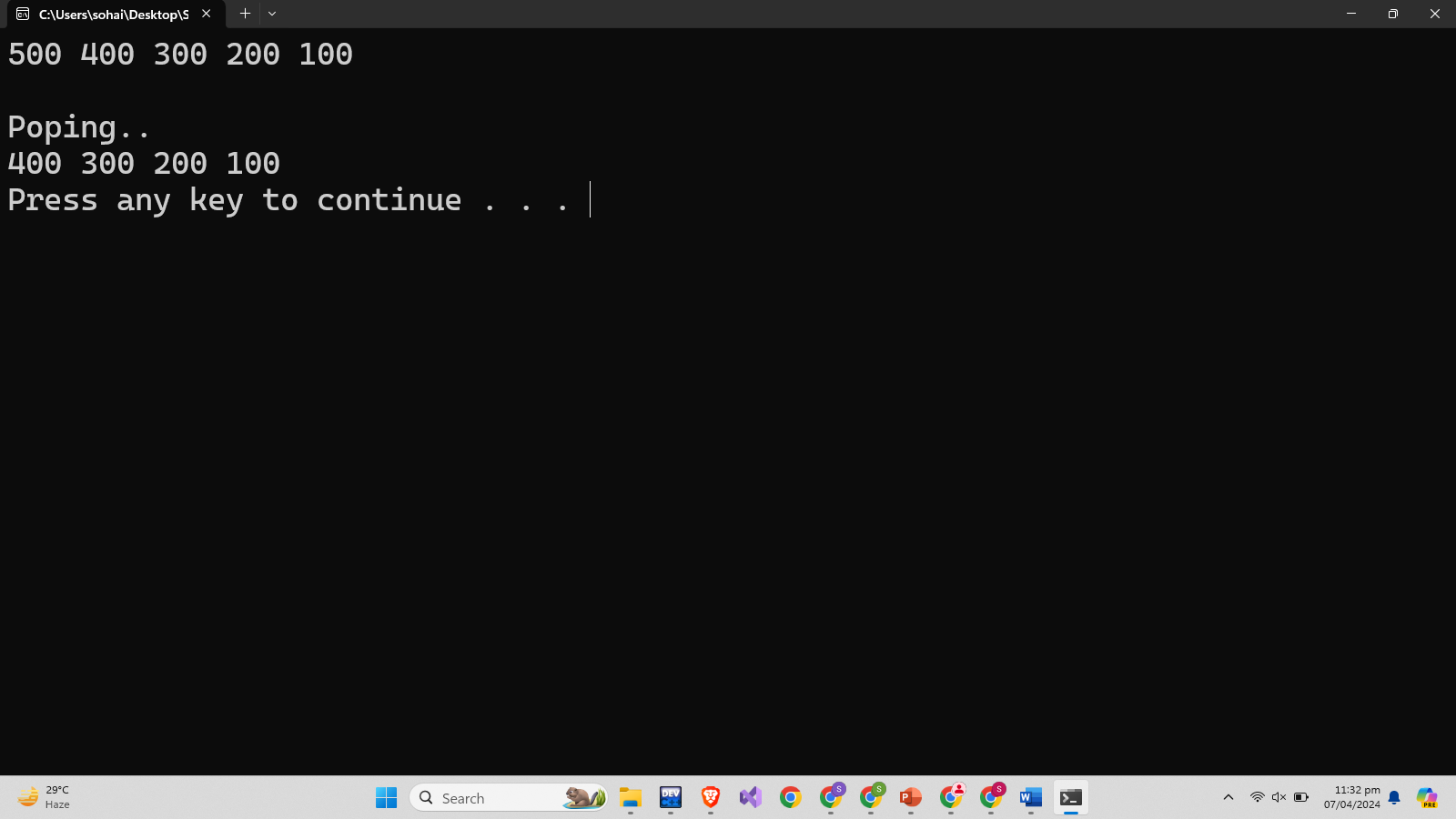
stk.Pop();

stk.Display();

system("pause");

return 0;

}  
  
**Output:**



**Exercise-3: Implement the following applications of stack.**

1. **Infix to Postfix Conversion.**

**Code:**

#include <iostream>

#include <stack>

#include <string>

using namespace std;

int precedence(char op){

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

return 1;

}

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

return 2;

}

return 0;

}

string infixToPostfix(string infix){

stack<char> st;

string postfix = "";

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

char e = infix[i];

if(isalnum(e)){

postfix += e;

}

else if(e == '('){

st.push(e);

}

else if(e == ')'){

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

postfix += st.top();

st.pop();

}

st.pop();

}

else{

while(!st.empty() && precedence(e) <= precedence(st.top())){

postfix += st.top();

st.pop();

}

st.push(e);

}

}

while(!st.empty()){

postfix += st.top();

st.pop();

}

return postfix;

}

int main() {

string infix = "a+b\*c+(d\*e+f)\*g";

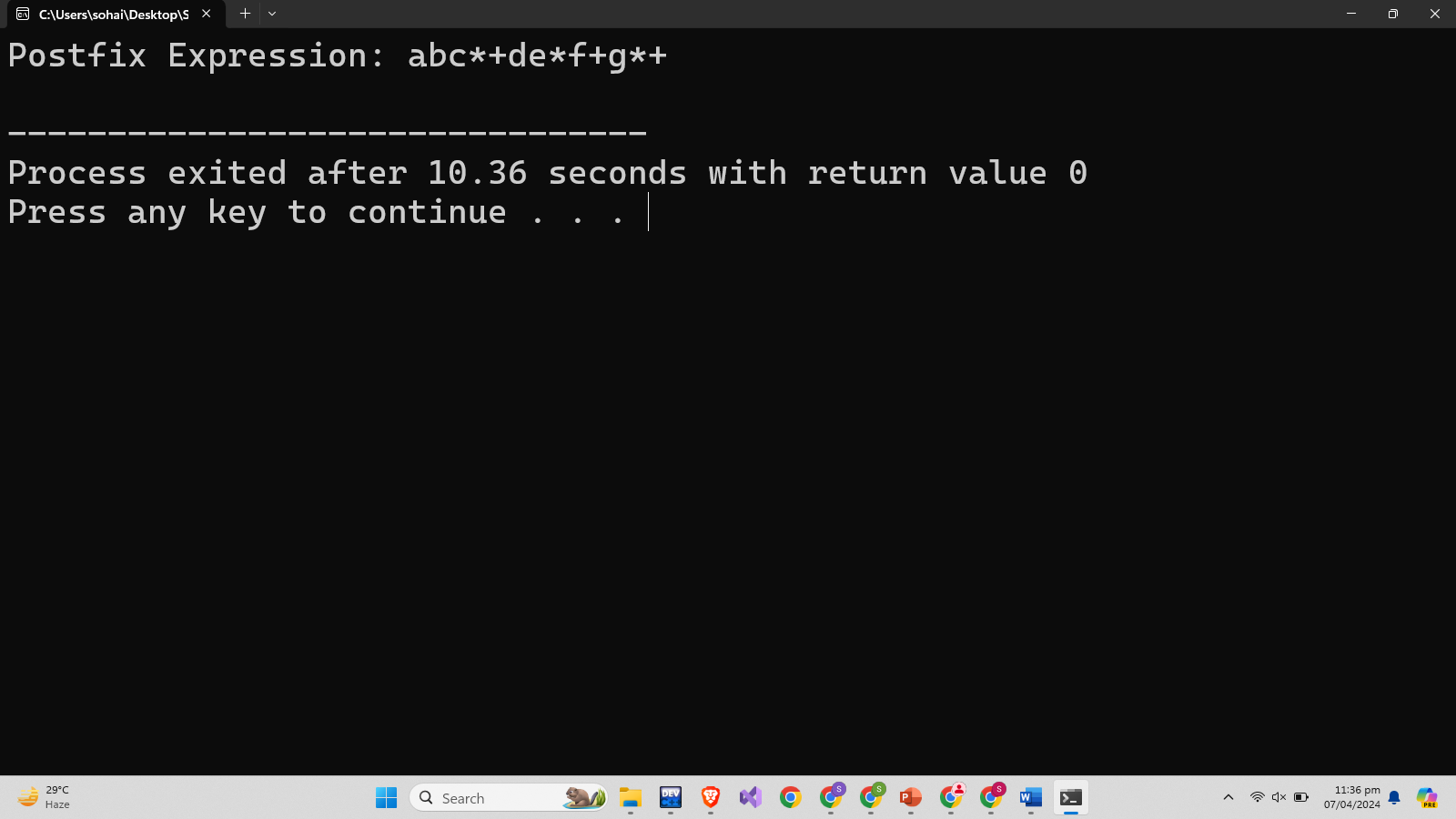
string postfix = infixToPostfix(infix);

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

return 0;

}

**Output:**

****

1. **Evaluation of Postfix Expressions.**

**Code:**

#include <iostream>

#include <cstring>

#include <cctype>

using namespace std;

const int Size = 100;

class Stack {

private:

int stackArray[Size];

int top;

public:

Stack() : top(-1) {}

bool isEmpty() {

return top == -1;

}

void push(int item) {

if (top < Size - 1) {

stackArray[++top] = item;

} else {

cout << "Stack Overflow\n";

}

}

int pop() {

if (!isEmpty()) {

return stackArray[top--];

} else {

cout << "Stack Underflow\n";

return INT\_MIN;

}

}

};

class PostfixEvaluator {

public:

int evaluatePostfix(const char\* exp) {

Stack stack;

for (int i = 0; exp[i]; i++) {

if (isdigit(exp[i])) {

stack.push(exp[i] - '0');

} else {

int val1 = stack.pop();

int val2 = stack.pop();

switch (exp[i]) {

case '+': stack.push(val2 + val1); break;

case '-': stack.push(val2 - val1); break;

case '\*': stack.push(val2 \* val1); break;

case '/': stack.push(val2 / val1); break;

}

}

}

return stack.pop();

}

};

int main() {

PostfixEvaluator evaluator;

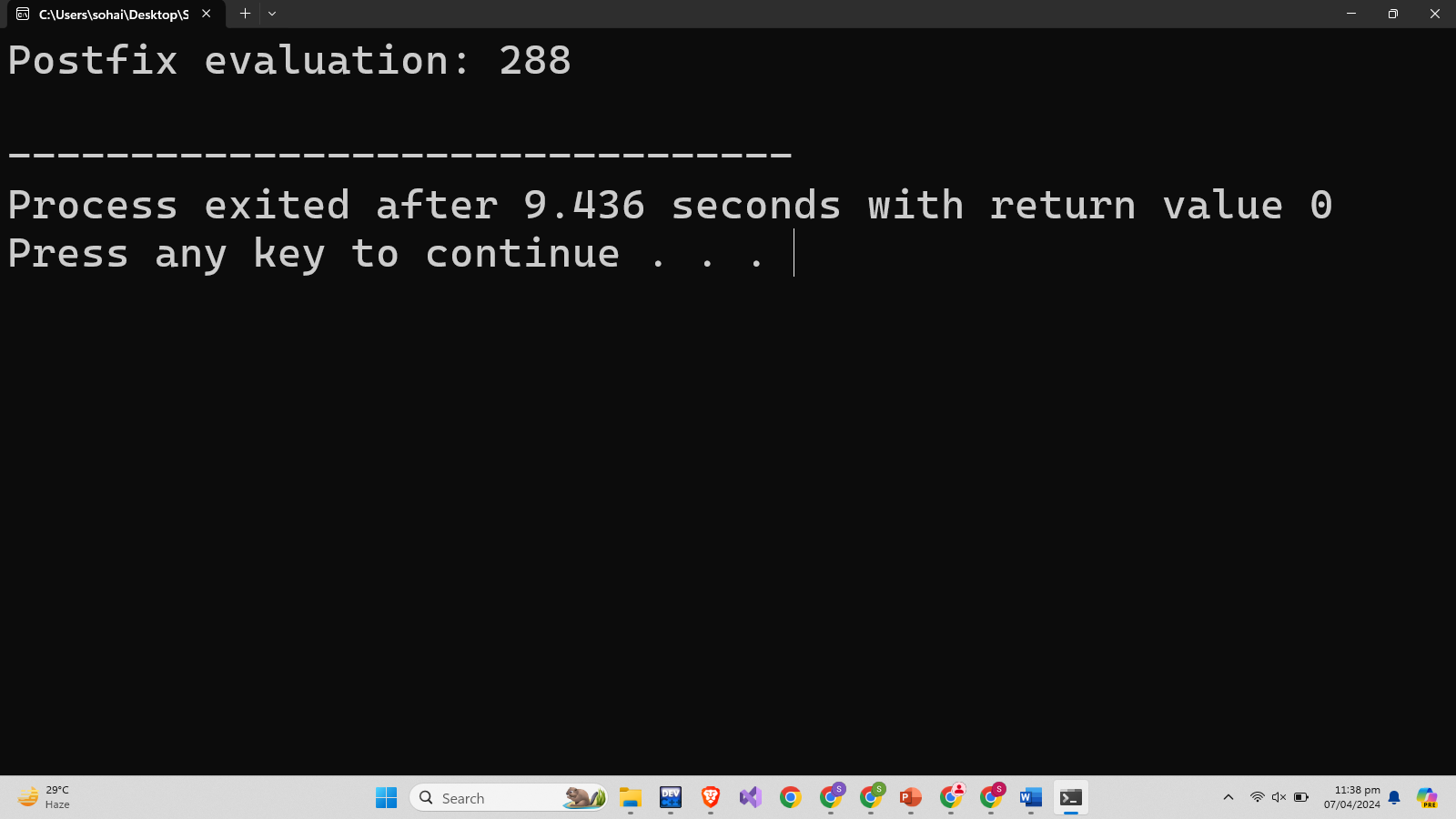
const char exp[] = "6523+8\*+3+\*";

cout << "Postfix evaluation: " << evaluator.evaluatePostfix(exp) << endl;

return 0;

}

**Output:**

****

1. **Decimal to Binary Conversion.**

**Code:**

#include <iostream>

using namespace std;

int const STACKSIZE = 20;

class Stack {

private:

int StackArray[STACKSIZE];

int Top;

public:

Stack() {

Top = -1;

}

bool IsEmpty() {

return Top == -1;

}

bool IsFull() {

return Top == STACKSIZE - 1;

}

int Size() {

return Top + 1;

}

void Push(int value) {

if(IsFull()) {

cout << "Stack is Full\n";

} else {

StackArray[++Top] = value;

}

}

void Pop() {

if(IsEmpty()) {

cout << "Stack is empty\n";

} else {

--Top;

}

}

int TopElement() {

if(IsEmpty()) {

cout << "Stack is empty\n";

return -1;

} else {

return StackArray[Top];

}

}

void Display() {

if (IsEmpty()) {

cout << "Stack is empty!" << endl;

return;

} else {

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

cout << StackArray[i];

}

cout << endl;

}

}

};

void integer(int number) {

Stack stk;

while(number > 0) {

int rem = number % 2;

number = number / 2;

stk.Push(rem);

}

stk.Display();

}

int main() {

int num;

cout << "Enter a number: ";

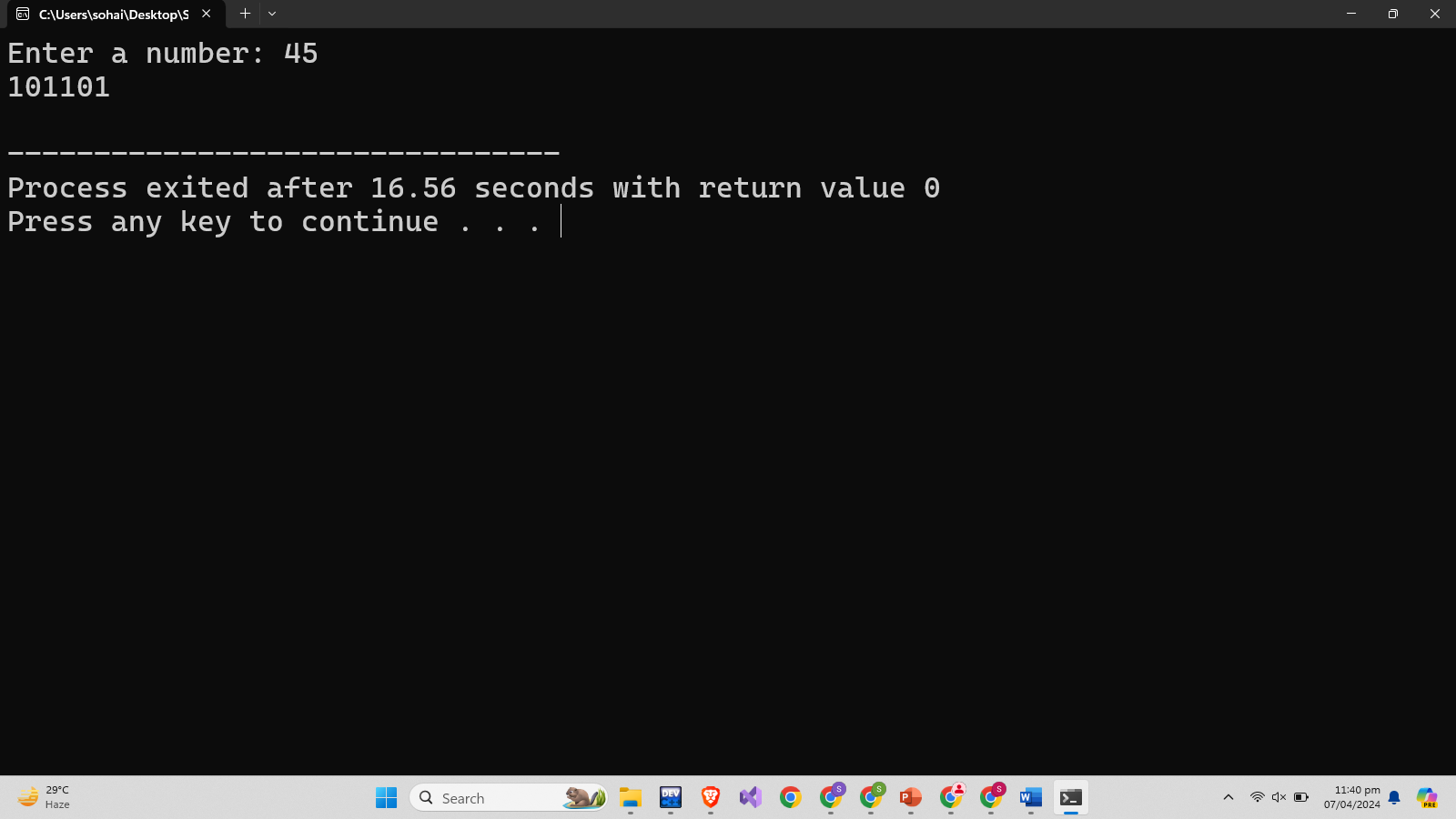
cin >> num;

integer(num);

return 0;

}

**Output:**

****

1. **Checking Balanced Symbols.**

**Code:**

#include <iostream>

#include <string>

using namespace std;

const int STACKSIZE = 100;

class Stack {

private:

char StackArray[STACKSIZE];

int Top;

public:

Stack() : Top(-1) {}

bool IsEmpty() const {

return Top == -1;

}

bool IsFull() const {

return Top == STACKSIZE - 1;

}

int Size() const {

return Top + 1;

}

void Push(char value) {

if (!IsFull()) {

Top++;

StackArray[Top] = value;

} else {

cout << "Stack Overflow!" << endl;

}

}

void Pop() {

if (!IsEmpty()) {

Top--;

} else {

cout << "Stack Underflow!" << endl;

}

}

char TopElement() const {

if (!IsEmpty()) {

return StackArray[Top];

} else {

return '\0';

}

}

};

bool isBalanced(const string& expression) {

Stack s;

for (size\_t i = 0; i < expression.length(); ++i) {

char ch = expression[i];

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

s.Push(ch);

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

if (s.IsEmpty())

return false;

char top = s.TopElement();

if ((ch == ')' && top == '(') ||

(ch == ']' && top == '[') ||

(ch == '}' && top == '{')) {

s.Pop();

} else {

return false;

}

}

}

return s.IsEmpty();

}

int main() {

string expression;

cout << "Enter an expression: ";

getline(cin, expression);

if (isBalanced(expression)) {

cout << "The expression is balanced." << endl;

} else {

cout << "The expression is not balanced." << endl;

}

return 0;

}

**Output:**

**A screenshot of a computer

Description automatically generated**