//Nasirin, Abdul M.

//ccc121,Final\_project.

//Solving Numerical Expression.

#include <iostream>

#include <stack>

using namespace std;

// Function for arithmetic operations

float applyOperation(float a, float b, char op) {

switch (op) {

case '+': return a + b;

case '-': return a - b;

case '\*': return a \* b;

case '/': return a / b;

}

return 0;

}

// Function most priority operators

// Funtion for following MDAS

int typeOP(char op) {

// Check if the operator is '+' or '-'

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

return 1; // these as simple lower priority

}

// Check if the operator is '\*' or '/'

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

return 2; // these return is for the higher priority like MDAS

}

return 0; // Return 0 if the operator is not recognized

}

// Function to check if a character is a digit

bool isDigit(char ch) {

return ch >= '0' && ch <= '9';

}

// Funtion for evaluating expressions

float evaluateExpression(const string &expression) {

stack<float> values; // Stack to store numbers

stack<char> operators; // Stack to store operators

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

// Skip spaces

if (expression[i] == ' ') continue;

// If current character is a digit, analyze the number

if (isDigit(expression[i])) {

float value = 0;

while (i < expression.length() && isDigit(expression[i])) {

value = (value \* 10) + (expression[i] - '0');

i++;

}

if (i < expression.length() && expression[i] == '.') {

// Handle decimal numbers

float decimalFactor = 0.1;

i++;

while (i < expression.length() && isDigit(expression[i])) {

value += (expression[i] - '0') \* decimalFactor;

decimalFactor \*= 0.1;

i++;

}

}

values.push(value);

i--; // Adjust for the outer loop increment

}

else if (expression[i] == '(') {

// Push '(' to operators stack

operators.push(expression[i]);

}

else if (expression[i] == ')') {

// Solve the expression inside and balance the parentheses

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

float b = values.top(); values.pop();

float a = values.top(); values.pop();

char op = operators.top(); operators.pop();

values.push(applyOperation(a, b, op));

}

operators.pop(); // Remove '('

}

else {

// Operator encountered

while (!operators.empty() && typeOP(operators.top()) >= typeOP(expression[i])) {

float b = values.top(); values.pop();

float a = values.top(); values.pop();

char op = operators.top(); operators.pop();

values.push(applyOperation(a, b, op));

}

operators.push(expression[i]);

}

}

// Process remaining operators

while (!operators.empty()) {

float b = values.top(); values.pop();

float a = values.top(); values.pop();

char op = operators.top(); operators.pop();

values.push(applyOperation(a, b, op));

}

return values.top();

}

int main() {

string expression;

cout << "Enter a numerical expression: ";

cin >> expression;

try {

float result = evaluateExpression(expression);

cout << "Result: " << result << endl;

}

catch (const exception &e) {

cout << "Error: " << e.what() << endl;

}

return 0;

}