#include <iostream>

#include <stack>

#include <cctype> // For isdigit

using namespace std;

// Function to determine precedence of operators

int precedence(char op) {

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

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

return 0;

}

// Function to convert infix to postfix

string infixToPostfix(string infix) {

stack<char> s;

string postfix = "";

for (char ch : infix) {

if (isalnum(ch)) { // If operand, add to postfix

postfix += ch;

} else if (ch == '(') { // If opening parenthesis, push to stack

s.push(ch);

} else if (ch == ')') { // If closing parenthesis, pop until '('

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

postfix += s.top();

s.pop();

}

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

} else { // Operator

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

postfix += s.top();

s.pop();

}

s.push(ch);

}

}

// Pop remaining operators from stack

while (!s.empty()) {

postfix += s.top();

s.pop();

}

return postfix;

}

// Function to evaluate a postfix expression

int evaluatePostfix(string postfix) {

stack<int> s;

for (char ch : postfix) {

if (isdigit(ch)) { // If operand, push to stack

s.push(ch - '0');

} else { // Operator

int b = s.top(); s.pop();

int a = s.top(); s.pop();

switch (ch) {

case '+': s.push(a + b); break;

case '-': s.push(a - b); break;

case '\*': s.push(a \* b); break;

case '/': s.push(a / b); break;

}

}

}

return s.top(); // Final result

}

int main() {

string infix;

cout << "Enter infix expression: ";

cin >> infix;

// Convert infix to postfix

string postfix = infixToPostfix(infix);

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

// Evaluate the postfix expression

int result = evaluatePostfix(postfix);

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

return 0;

}