Assignment 3 Question 4

#include <iostream>

#include <stack>

#include <string>

using namespace std;

// ----- small helpers (no <cctype>) -----

bool isSpace(char c) { return c == ' ' || c == '\t'; }

bool isLetter(char c) { return (c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z'); }

bool isDigit(char c) { return (c >= '0' && c <= '9'); }

bool isOperand(char c) { return isLetter(c) || isDigit(c); }

// precedence: higher number = higher priority

int precedence(char op) {

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

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

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

return 0;

}

// '^' is right-associative; others are left

bool isRightAssociative(char op) { return op == '^'; }

bool isOperator(char c) {

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

}

string infixToPostfix(const string& infix) {

stack<char> st; // holds operators and '('

string post; // output

for (char c : infix) {

if (isSpace(c)) continue; // ignore spaces/tabs

if (isOperand(c)) // 1) operand → output

post += c;

else if (c == '(') // 2) '(' → push

st.push(c);

else if (c == ')') { // 3) ')' → pop until '('

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

post += st.top(); st.pop();

}

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

}

else if (isOperator(c)) { // 4) operator

while (!st.empty() && isOperator(st.top())) {

int pTop = precedence(st.top());

int pCur = precedence(c);

bool popIt = (pTop > pCur) || (pTop == pCur && !isRightAssociative(c));

if (popIt) { post += st.top(); st.pop(); }

else break;

}

st.push(c);

}

// very-easy version: ignore any other characters

}

// pop remaining operators

while (!st.empty()) { post += st.top(); st.pop(); }

return post;

}

int main() {

string expr;

cout << "Enter infix expression: ";

getline(cin, expr);

cout << "Postfix: " << infixToPostfix(expr) << "\n";

return 0;

}