```cpp
#include <iostream>
#include <stack>
#include <string>

using namespace std;

class InfixToPostfix {

private:
    string infix;
    string postfix;

    int precedence(char c) {
        if (c == '+' || c == '-') {
            return 1;
        } else if (c == '*' || c == '/') {
            return 2;
        } else {
            return 0;
        }
    }

    double performOperation(char op, double op1, double op2) {
        switch(op) {
        case '+':
            return op1 + op2;
        case '-':
            return op1 - op2;
        case '*':
            return op1 * op2;
        case '/':
            return op1 / op2;
        default:
            return 0;
        }
    }
public:
    InfixToPostfix(string infix) {
        this->infix = infix;
    }

    string convertToPostfix() {
        string postfix = "";
        stack<char> opStack;

        for (int i = 0; i < infix.length(); i++) {
            char c = infix[i];
            if (isalnum(c)) {
                postfix += c;
            } else if (c == '(') {
                opStack.push(c);
            } else if (c == ')') {
                while (!opStack.empty() && opStack.top() != '(') {
                    postfix += opStack.top();
                    opStack.pop();
                }
                opStack.pop();
            } else {
                while (!opStack.empty() && precedence(opStack.top()) >= precedence(c)) {
                    postfix += opStack.top();
                    opStack.pop();
                }
                opStack.push(c);
            }
        }

        while (!opStack.empty()) {
            postfix += opStack.top();
            opStack.pop();
        }

        this->postfix = postfix; // set the member variable to the computed postfix expression

        return postfix;
    }
```

```
    double evaluatePostfix() {
        stack<double> operandStack;

        for (int i = 0; i < postfix.length(); i++) {
            char c = postfix[i];
            if (isdigit(c)) {
                operandStack.push(c - '0');
            } else {
                double op2 = operandStack.top();
                operandStack.pop();
                double op1 = operandStack.top();
                operandStack.pop();
                double result = performOperation(c, op1, op2);
                operandStack.push(result);
            }
        }

        return operandStack.top();
    }

};

int main() {
    string infix;
    cout << "Enter an infix expression: ";
    getline(cin, infix);
    InfixToPostfix converter(infix);
    string postfix = converter.convertToPostfix();
    cout << "Postfix: " << postfix << endl;
    double result = converter.evaluatePostfix();
    cout << "Result: " << result << endl;
    return 0;
}
```