Infix to postfix

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

// Function to check operator precedence
int precedence(char op) {
    if (op == '+' || op == '-') return 1;
    if (op == '*' || op == '/') return 2;
    return 0;
}

// Function to convert infix to postfix
std::string infixToPostfix(const std::string& infix) {
    std::stack<char> s;
    std::string postfix;

    for (char ch : infix) {
        if (std::isalnum(ch)) {
            postfix += ch;
        } else if (ch == '(') {
            s.push(ch);
        } else if (ch == ')') {
            while (!s.empty() && s.top() != '(') {
                postfix += s.top();
                s.pop();
            }
            s.pop();
        } else {
            while (!s.empty() && precedence(s.top()) >= precedence(ch)) {
                postfix += s.top();
                s.pop();
            }
            s.push(ch);
        }
    }

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

    return postfix;
}
```

```
// Function to evaluate postfix expression
int evaluatePostfix(const std::string& postfix) {
    std::stack<int> s;

    for (char ch : postfix) {
        if (std::isdigit(ch)) {
            s.push(ch - '0');
        } else {
            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();
}

int main() {
    std::string infix;
    std::cout << "Enter an infix expression: ";
    std::cin >> infix;

    std::string postfix = infixToPostfix(infix);
    std::cout << "Postfix expression: " << postfix << "\n";

    int result = evaluatePostfix(postfix);
    std::cout << "Result: " << result << "\n";

    return 0;
}
```