PREFIX EVALUATION

```
#include <iostream>
#include <stack>
#include <math.h>
using namespace std;

int prefixNotation(string s)
{
    stack<int> st;
    for (int i = s.length()-1; i >= 0; i--)
    {
        if (s[i]>='0' && s[i] <= '9')
        {                       //// to detect it is an operand
            s[i] = s[i] - '0'; //// ascii value
            st.push(s[i]);
        }
        else
        {
            int opr1 = st.top();
            st.pop();
            int opr2 = st.top();
            st.pop();

            switch (s[i])
            {
            case '+':
                st.push(opr1 + opr2);
                break;
            case '-':
                st.push(opr1 - opr2);
                break;
            case '/':
                st.push(opr1 / opr2);
                break;
            case '*':
                st.push(opr1 * opr2);
                break;
            case '^':
                st.push(pow(opr1, opr2));
                break;


            }
        }
    }
    return st.top();
}
```

```
int main()
{
    cout<<prefixNotation("-+7*45+20")<<endl; //25
    cout<<prefixNotation("+*423")<<endl;     // 11

    return 0;
}
```

POSTFIX EVALUATION

```
#include <iostream>
#include <stack>
#include <math.h>
using namespace std;

int postfixEvaluation(string s)
{
    stack<int> st;
    for (int i=0;i<s.length()-1;i++)
    {
        if (s[i] >= '0' && s[i] <= '9')
        {
            st.push(s[i] - '0');
        }
        else
        {
            int op2 = st.top();
            st.pop();
            int op1 = st.top();
            st.pop();

            switch (s[i])
            {
            case '+':
                st.push(op1 + op2);
                break;
            case '-':
                st.push(op1 - op2);
                break;
            case '/':
                st.push(op1 / op2);
                break;
            case '*':
                st.push(op1 * op2);
                break;
            case '^':
                st.push(pow(op1, op2));
                break;
            }
```

```
        }
    }
    return st.top();
}

int main()
{
    cout << postfixEvaluation("22+2-4*2/2*") << endl;
    cout << postfixEvaluation("22-2+4*2/2*") << endl;

    return 0;
}
```

INFIX TO POSTFIX

```
#include <iostream>
#include <stack>
#include <string>
using namespace std;

// Function to check if a character is an operator or not
bool isOperator(char c) {
    if (c == '+' || c == '-' || c == '*' || c == '/' || c == '^') {
        return true;
    }
    return false;
}

// Function to check precedence of operators
int precedence(char c) {
    if (c == '^') {
        return 3;
    } else if (c == '*' || c == '/') {
        return 2;
    } else if (c == '+' || c == '-') {
        return 1;
    }
    return -1;
}

// Function to convert infix expression to postfix
string infixToPostfix(string infix) {
    stack<char> s;
    string postfix;

    for (int i = 0; i < infix.length(); i++) {
        char c = infix[i];

        // If character is an operand, add it to the output string
        if (isalnum(c)) {
```

```
            postfix += c;
        }
        // If character is a left parenthesis, push it onto the stack
        else if (c == '(') {
            s.push(c);
        }
        // If character is a right parenthesis, pop operators from the stack
        // and add them to the output string until a left parenthesis is
encountered
        else if (c == ')') {
            while (!s.empty() && s.top() != '(') {
                postfix += s.top();
                s.pop();
            }
            if (!s.empty() && s.top() == '(') {
                s.pop();
            }
        }
        // If character is an operator, pop operators from the stack
        // and add them to the output string until an operator of lower
precedence is encountered
        else if (isOperator(c)) {
            while (!s.empty() && precedence(c) <= precedence(s.top())) {
                postfix += s.top();
                s.pop();
            }
            s.push(c);
        }
    }

    // Pop any remaining operators from the stack and add them to the output
string
    while (!s.empty()) {
        postfix += s.top();
        s.pop();
    }

    return postfix;
}

int main() {
    string infix = "(a-b/c)*(a/k-l)";

    cout << "Infix expression: " << infix << endl;
    cout << "Postfix expression: " << infixToPostfix(infix) << endl;

    return 0;
}
```