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


struct Node {
    char data;
    Node* next;
};


class Stack {
private:
    Node* top;

public:
    Stack() : top(nullptr) {}

    void push(char val) {
        Node* newNode = new Node();
        newNode->data = val;
        newNode->next = top;
        top = newNode;
    }

    char pop() {
        if (isEmpty()) {
            throw runtime_error("Stack Underflow");
        }
        Node* tmp = top;
        char value = top->data;
        top = top->next;
        delete tmp;
        return value;
    }

    char peek() {
        if (isEmpty()) {
            throw runtime_error("Stack is Empty");
        }
        return top->data;
    }

    bool isEmpty() {
        return top == nullptr;
    }
};


int precedence(char op) {
    if (op == '+' || op == '-') return 1;
```

```
    if (op == '*' || op == '/') return 2;
    if (op == '^') return 3;
    return 0;
}


string infixToPostfix(string infix) {
    Stack stack;
    string postfix = "";
    for (char ch : infix) {
        if (isalnum(ch)) {
            postfix += ch;
        } else if (ch == '(') {
            stack.push(ch);
        } else if (ch == ')') {
            while (!stack.isEmpty() && stack.peek() != '(') {
                postfix += stack.pop();
            }
            stack.pop();
        } else {
            while (!stack.isEmpty() && precedence(stack.peek()) >=
precedence(ch)) {
                postfix += stack.pop();
            }
            stack.push(ch);
        }
    }
    while (!stack.isEmpty()) {
        postfix += stack.pop();
    }
    return postfix;
}


string infixToPrefix(string infix) {
    reverse(infix.begin(), infix.end());

    for (char& ch : infix) {
        if (ch == '(') ch = ')';
        else if (ch == ')') ch = '(';
    }

    string postfix = infixToPostfix(infix);
    reverse(postfix.begin(), postfix.end());
    return postfix;
}


int evaluatePostfix(string postfix) {
    stack<int> evalStack;
    for (char ch : postfix) {
        if (isdigit(ch)) {
            evalStack.push(ch - '0');
        } else {
```

```
            int b = evalStack.top(); evalStack.pop();
            int a = evalStack.top(); evalStack.pop();
            switch (ch) {
                case '+': evalStack.push(a + b); break;
                case '-': evalStack.push(a - b); break;
                case '*': evalStack.push(a * b); break;
                case '/': evalStack.push(a / b); break;
                case '^': evalStack.push(pow(a, b)); break;
            }
        }
    }
    return evalStack.top();
}


int evaluatePrefix(string prefix) {
    stack<int> evalStack;
    reverse(prefix.begin(), prefix.end());

    for (char ch : prefix) {
        if (isdigit(ch)) {
            evalStack.push(ch - '0');
        } else {
            int a = evalStack.top(); evalStack.pop();
            int b = evalStack.top(); evalStack.pop();
            switch (ch) {
                case '+': evalStack.push(a + b); break;
                case '-': evalStack.push(a - b); break;
                case '*': evalStack.push(a * b); break;
                case '/': evalStack.push(a / b); break;
                case '^': evalStack.push(pow(a, b)); break;
            }
        }
    }
    return evalStack.top();
}


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

    string postfix = infixToPostfix(infix);
    string prefix = infixToPrefix(infix);

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

    cout << "Evaluating Postfix Expression: " << evaluatePostfix(postfix)
<< endl;
    cout << "Evaluating Prefix Expression: " << evaluatePrefix(prefix) <<
endl;
```

```
    return 0;
}
```