#include <iostream>

using namespace std;

#define MAX 100

class Stack {

    char arr[MAX];

    int top;

public:

    Stack() { top = -1; }

    void push(char ch) {

        if (top == MAX - 1) {

            cout << "Stack Overflow" << endl;

            return;

        }

        arr[++top] = ch;

    }

    char pop() {

        if (top == -1) {

            cout << "Stack Underflow" << endl;

            return -1;

        }

        return arr[top--];

    }

    char peek() {

        if (top == -1) return -1;

        return arr[top];

    }

    bool isEmpty() {

        return top == -1;

    }

};

class IntStack {

    int arr[MAX];

    int top;

public:

    IntStack() { top = -1; }

    void push(int val) {

        if (top == MAX - 1) {

            cout << "Stack Overflow" << endl;

            return;

        }

        arr[++top] = val;

    }

    int pop() {

        if (top == -1) {

            cout << "Stack Underflow" << endl;

            return -1;

        }

        return arr[top--];

    }

    bool isEmpty() {

        return top == -1;

    }

};

int precedence(char op) {

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

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

    if (op == '^') return 3;

    return 0;

}

bool isOperator(char c) {

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

}

string infixToPostfix(const string& infix) {

    Stack s;

    string postfix;

    for (int i = 0; i < infix.length(); i++) {

        char c = infix[i];

        if (isalnum(c)) {

            postfix += c;

        } else if (c == '(') {

            s.push(c);

        } else if (c == ')') {

            while (!s.isEmpty() && s.peek() != '(') {

                postfix += s.pop();

            }

            s.pop();

        } else if (isOperator(c)) {

            while (!s.isEmpty() && precedence(s.peek()) >= precedence(c)) {

                postfix += s.pop();

            }

            s.push(c);

        }

    }

    while (!s.isEmpty()) {

        postfix += s.pop();

    }

    return postfix;

}

void reverseString(string& str) {

    int len = str.length();

    for (int i = 0; i < len / 2; i++) {

        swap(str[i], str[len - i - 1]);

    }

}

string infixToPrefix(string infix) {

    reverseString(infix);

    for (int i = 0; i < infix.length(); i++) {

        if (infix[i] == '(') {

            infix[i] = ')';

        } else if (infix[i] == ')') {

            infix[i] = '(';

        }

    }

    string postfix = infixToPostfix(infix);

    reverseString(postfix);

    return postfix;

}

int performOperation(int operand1, int operand2, char op) {

    switch (op) {

        case '+': return operand1 + operand2;

        case '-': return operand1 - operand2;

        case '\*': return operand1 \* operand2;

        case '/': return operand1 / operand2;

        case '^': {

            int result = 1;

            for (int i = 0; i < operand2; i++)

                result \*= operand1;

            return result;

        }

        default: return 0;

    }

}

int evaluatePostfix(const string& postfix) {

    IntStack s;

    for (int i = 0; i < postfix.length(); i++) {

        char c = postfix[i];

        if (isalnum(c)) {

            int value;

            cout << "Enter value for " << c << ": ";

            cin >> value;

            s.push(value); // Use the variable value directly

        } else if (isOperator(c)) {

            int operand2 = s.pop();

            int operand1 = s.pop();

            int result = performOperation(operand1, operand2, c);

            s.push(result);

        }

    }

    return s.pop();

}

int evaluatePrefix(const string& prefix) {

    IntStack s;

    for (int i = prefix.length() - 1; i >= 0; i--) {

        char c = prefix[i];

        if (isalnum(c)) {

            int value;

            cout << "Enter value for " << c << ": ";

            cin >> value;

            s.push(value); // Use the variable value directly

        } else if (isOperator(c)) {

            int operand1 = s.pop();

            int operand2 = s.pop();

            int result = performOperation(operand1, operand2, c);

            s.push(result);

        }

    }

    return s.pop();

}

int main() {

    string infix;

    cout << "Enter infix expression: ";

    cin >> infix;

    string postfix = infixToPostfix(infix);

    string prefix = infixToPrefix(infix);

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

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

    cout << "Postfix Evaluation: " << evaluatePostfix(postfix) << endl;

    cout << "Prefix Evaluation: " << evaluatePrefix(prefix) << endl;

    return 0;

}