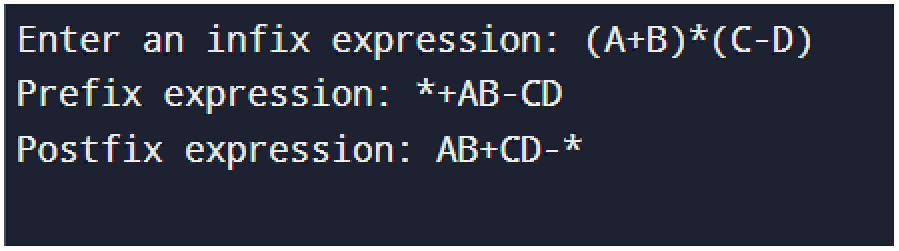
INFIX TO POSTFIX AND PREFIX

```cpp
#include <iostream>
#include <stack>
#include <string>
#include <bits/stdc++.h>    // Includes all standard C++ libraries
using namespace std;

bool isOperator(char c) {
    return c == '+' || c == '-' || c == '*' || c == '/' || c == '^';
}

int precedence(char c) {
    if (c == '^') return 3;
    if (c == '*' || c == '/') return 2;
    if (c == '+' || c == '-') return 1;
    return -1;
}



string infixToPostfix(string infix) {
    stack<char> operators;
    stack<string> operands;

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

        if (c == '(') {
            operators.push(c);
        }
        else if (c == ')') {
            while (operators.top() != '(') {
                string op1 = operands.top();
                operands.pop();
                string op2 = operands.top();
                operands.pop();
                char op = operators.top();
                operators.pop();

                operands.push(op2 + op1 + op);
            }
            operators.pop(); // Remove the '('
        }

        else if (isOperator(c)) {
            while (!operators.empty() && precedence(c) <= precedence(operators.top())) {
                string op1 = operands.top();
                operands.pop();
                string op2 = operands.top();
                operands.pop();
                char op = operators.top();
                operators.pop();

                operands.push(op2 + op1 + op);
            }
            operators.push(c);
        }
         else {
            operands.push(string(1, c));
        }
    }

    while (!operators.empty()) {
        string op1 = operands.top();
        operands.pop();
        string op2 = operands.top();
        operands.pop();
        char op = operators.top();
        operators.pop();

        operands.push(op2 + op1 + op);
    }

    return operands.top();
}


string infixToPrefix(string infix) {

   int l = infix.size();

// Reverse infix
reverse(infix.begin(), infix.end());

 // Replace ( with ) and vice versa
 for (int i = 0; i < l; i++) {

 if (infix[i] == '(') {
            infix[i] = ')';
            }
 else if (infix[i] == ')') {
          infix[i] = '(';
            }
 }

 string prefix = infixToPostfix(infix);

 // Reverse postfix
 reverse(prefix.begin(), prefix.end());

 return prefix;
}




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

    string prefix = infixToPrefix(infix);
    cout << "Prefix expression: " << prefix << endl;


    string postfix = infixToPostfix(infix);
    cout << "Postfix expression: " << postfix << endl;

    return 0;
}

```



```
Enter an infix expression: (A+B)*(C-D)
Prefix expression: *+AB-CD
Postfix expression: AB+CD-*
```