

INFIX TO POSTFIX AND PREFIX

```
1. #include <iostream>
2. #include <stack>
3. #include <string>
4. #include <bits/stdc++.h>    // Includes all standard C++ libraries
5. using namespace std;
6.
7. bool isOperator(char c) {
8.     return c == '+' || c == '-' || c == '*' || c == '/' || c == '^';
9. }
10.
11. int precedence(char c) {
12.     if (c == '^') return 3;
13.     if (c == '*' || c == '/') return 2;
14.     if (c == '+' || c == '-') return 1;
15.     return -1;
16. }
17.
18.
19.
20. string infixToPostfix(string infix) {
21.     stack<char> operators;
22.     stack<string> operands;
23.
24.     for (int i = 0; i < infix.length(); ++i) {
25.         char c = infix[i];
26.
27.         if (c == '(') {
28.             operators.push(c);
29.         }
30.         else if (c == ')') {
31.             while (operators.top() != '(') {
32.                 string op1 = operands.top();
33.                 operands.pop();
34.                 string op2 = operands.top();
35.                 operands.pop();
36.                 char op = operators.top();
37.                 operators.pop();
38.                 operands.push(op2 + op1 + op);
39.             }
40.             operators.pop(); // Remove the '('
41.         }
42.         else if (isOperator(c)) {
43.             while (!operators.empty() && precedence(c) <= precedence(operators.top())) {
44.                 string op1 = operands.top();
45.                 operands.pop();
46.                 string op2 = operands.top();
47.                 operands.pop();
48.                 char op = operators.top();
49.                 operators.pop();
50.                 operands.push(op2 + op1 + op);
51.             }
52.             operators.push(c);
53.         }
54.         else {
55.             operands.push(string(1, c));
56.         }
57.     }
```

```

58.     while (!operators.empty()) {
59.         string op1 = operands.top();
60.         operands.pop();
61.         string op2 = operands.top();
62.         operands.pop();
63.         char op = operators.top();
64.         operators.pop();
65.
66.         operands.push(op2 + op1 + op);
67.     }
68.
69.     return operands.top();
70. }

71. string infixToPrefix(string infix) {
72.     int l = infix.size();
73.
74.     // Reverse infix
75.     reverse(infix.begin(), infix.end());
76.
77.     // Replace ( with ) and vice versa
78.     for (int i = 0; i < l; i++) {
79.
80.         if (infix[i] == '(') {
81.             infix[i] = ')';
82.         }
83.         else if (infix[i] == ')') {
84.             infix[i] = '(';
85.         }
86.     }
87.
88.     string prefix = infixToPostfix(infix);
89.
90.     // Reverse postfix
91.     reverse(prefix.begin(), prefix.end());
92.
93.     return prefix;
94. }
95.

96. int main() {
97.     string infix;
98.     cout << "Enter an infix expression: ";
99.     getline(cin, infix);
100.
101.     string prefix = infixToPrefix(infix);
102.     cout << "Prefix expression: " << prefix << endl;
103.
104.
105.     string postfix = infixToPostfix(infix);
106.     cout << "Postfix expression: " << postfix << endl;
107.
108.     return 0;
109. }
110.

```

Enter an infix expression: $(A+B)*(C-D)$

Prefix expression: $*+AB-CD$

Postfix expression: $AB+CD-*$