

NAME : PREM SAH

PRN : 123B1B234

DIV : D

### ASSIGNMENT NO.8

Write a program to convert infix expression to postfix, infix expression to prefix and evaluation of postfix and prefix expression.

Code :

```
#include <iostream>
#include <stack>
#include <algorithm>
using namespace std;
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 == '^');
}
// Infix to Postfix
string infixToPostfix(const string& infix) {
    stack<char> st;
    string postfix;
    for (char c : infix) {
        if (isalnum(c)) postfix += c;
        else if (c == '(') st.push(c);
        else if (c == ')') {
            while (st.top() != '(') { postfix += st.top(); st.pop(); }
            st.pop();
        } else {
            while (!st.empty() && precedence(c) <= precedence(st.top())) {
                postfix += st.top(); st.pop();
            }
            st.push(c);
        }
    }
    while (!st.empty()) { postfix += st.top(); st.pop(); }
    return postfix;
}
// Infix to Prefix
string infixToPrefix(string infix) {
    reverse(infix.begin(), infix.end());
    for (char &c : infix) if (c == '(') c = ')'; else if (c == ')') c = '(';
    string postfix = infixToPostfix(infix);
    reverse(postfix.begin(), postfix.end());
    return postfix;
}
```

```

}
// Evaluate Postfix
int evaluatePostfix(const string& postfix) {
    stack<int> st;
    for (char c : postfix) {
        if (isdigit(c)) st.push(c - '0');
        else {
            int op2 = st.top(); st.pop();
            int op1 = st.top(); st.pop();
            switch (c) {
                case '+': st.push(op1 + op2); break;
                case '-': st.push(op1 - op2); break;
                case '*': st.push(op1 * op2); break;
                case '/': st.push(op1 / op2); break;
            }
        }
    }
    return st.top();
}

// Evaluate Prefix
int evaluatePrefix(const string& prefix) {
    stack<int> st;
    for (int i = prefix.size() - 1; i >= 0; i--) {
        char c = prefix[i];
        if (isdigit(c)) st.push(c - '0');
        else {
            int op1 = st.top(); st.pop();
            int op2 = st.top(); st.pop();
            switch (c) {
                case '+': st.push(op1 + op2); break;
                case '-': st.push(op1 - op2); break;
                case '*': st.push(op1 * op2); break;
                case '/': st.push(op1 / op2); break;
            }
        }
    }
    return st.top();
}

int main() {
    string infix = "(5+3)*2";
    string postfix = infixToPostfix(infix);
    string prefix = infixToPrefix(infix);
    cout << "Postfix: " << postfix << endl;
    cout << "Prefix: " << prefix << endl;
    cout << "Postfix Evaluation: " << evaluatePostfix(postfix) << endl;
    cout << "Prefix Evaluation: " << evaluatePrefix(prefix) << endl;
    return 0;
}

```

Output :

Postfix: 53+2\*

Prefix: \*+532

Postfix Evaluation: 16

Prefix Evaluation: 16

main.cpp

```
1 #include <iostream>
2 #include <stack>
3 #include <algorithm>
4 using namespace std;
5 int precedence(char op) {
6     if (op == '+' || op == '-') return 1;
7     if (op == '*' || op == '/') return 2;
8     if (op == '^') return 3;
9     return 0;
10 }
11 bool isOperator(char c) {
12     return (c == '+' || c == '-' || c == '*' || c == '/' || c == '^');
13 }
14 // Infix to Postfix
15 string infixToPostfix(const string& infix) {
16     stack<char> st;
17     string postfix;
18     for (char c : infix) {
19         if (isdigit(c)) postfix += c;
20         else if (c == '(') st.push(c);
21         else if (c == ')') {
22             while (st.top() != '(') { postfix += st.top(); st.pop(); }
23             st.pop();
24         } else {
25             while (!st.empty() && precedence(c) <= precedence(st.top())) {
26                 postfix += st.top(); st.pop();
27             }
28             st.push(c);
29         }
30     }
31     while (!st.empty()) { postfix += st.top(); st.pop(); }
32     return postfix;
33 }
34 // Infix to Prefix
35 string infixToPrefix(string infix) {
36     reverse(infix.begin(), infix.end());
37     for (char &c : infix) if (c == '(') c = ')'; else if (c == ')') c = '(';
38     string postfix = infixToPostfix(infix);
39     reverse(postfix.begin(), postfix.end());
40     return postfix;
41 }
42 // Evaluate Postfix
43 int evaluatePostfix(const string& postfix) {
44     stack<int> st;
45     for (char c : postfix) {
46         if (isdigit(c)) st.push(c - '0');
47         else {
48             int op2 = st.top(); st.pop();
49             int op1 = st.top(); st.pop();
50             switch (c) {
```

```

50         case '+': st.push(op1 + op2); break;
51         case '-': st.push(op1 - op2); break;
52         case '*': st.push(op1 * op2); break;
53         case '/': st.push(op1 / op2); break;
54     }
55 }
56 }
57 return st.top();
58 } // Evaluate Prefix
59 int evaluatePrefix(const string& prefix) {
60     stack<int> st;
61     for (int i = prefix.size() - 1; i >= 0; i--) {
62         char c = prefix[i];
63         if (isdigit(c)) st.push(c - '0');
64         else {
65             int op1 = st.top(); st.pop();
66             int op2 = st.top(); st.pop();
67             switch (c) {
68                 case '+': st.push(op1 + op2); break;
69                 case '-': st.push(op1 - op2); break;
70                 case '*': st.push(op1 * op2); break;
71                 case '/': st.push(op1 / op2); break;
72             }
73         }
74     }
75     return st.top();
76 }
77 int main() {
78     string infix = "(5+3)*2";
79     string postfix = infixToPostfix(infix);
80     string prefix = infixToPrefix(infix);
81     cout << "Postfix: " << postfix << endl;
82     cout << "Prefix: " << prefix << endl;
83     cout << "Postfix Evaluation: " << evaluatePostfix(postfix) << endl;
84     cout << "Prefix Evaluation: " << evaluatePrefix(prefix) << endl;
85     return 0;
86 }
87

```

## Output

```

/tmp/U14CfyU1EZ.o
Postfix: 53+2*
Prefix: *(53)2
Postfix Evaluation: 16
Prefix Evaluation: 16

```

=== Code Execution Successful ===