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

```cpp
#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

```
/tmp/Ul4CfyU1EZ.o
Postfix: 53+2*
Prefix: *+532
Postfix Evaluation: 16
Prefix Evaluation: 16


=== Code Execution Successful ===
```