Experiment Number 10

Aim: Intermediate code generation – Postfix, Prefix.

Algorithm:

Step 1: Start

Step 2: Read infix input.

Step 3: Convert to postfix.

Step 4: Convert it to prefix.

Step 5: Print respective output.

Step 6: Stop.

Code:

```
#include <bits/stdc++.h>
using namespace std;
int prec(char c)
{
    if (c == '^')
        return 3;
    else if (c == '/' || c == '*')
        return 2;
    else if (c == '+' || c == '-')
        return 1;
    return -1;
}
string infixToPostfix(string s)
{
    stack<char> st;
    string result;
    for (int i = 0; i < s.length(); i++) {
        char c = s[i];
        if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')|| (c >= '0' && c <= '9'))
            result += c;
        else if (c == '(')
            st.push('(');
        else if (c == ')') {
            while (st.top() != '(') {
                result += st.top();
                st.pop();
            }
            st.pop();
        }
```

```
        else {
            while (!st.empty()
                   && prec(s[i]) <= prec(st.top())) {
                result += st.top();
                st.pop();
            }
            st.push(c);
        }
    }
    while (!st.empty()) {
        result += st.top();
        st.pop();
    }
    return result;
}
bool isOperator(char x)
{
    switch (x) {
    case '+':
    case '-':
    case '/':
    case '*':
        return true;
    }
    return false;
}
string infixToPrefix(string infix)
{
    int l = infix.size();
    reverse(infix.begin(), infix.end());
    for (int i = 0; i < l; i++) {

        if (infix[i] == '(') {
            infix[i] = ')';
        }
        else if (infix[i] == ')') {
            infix[i] = '(';
        }
    }
    string prefix = infixToPostfix(infix);
    reverse(prefix.begin(), prefix.end());
    return prefix;
}
string postToPre(string post_exp)
{
    stack<string> s;
```

```
    int length = post_exp.size();
    for (int i = 0; i < length; i++) {
        if (isOperator(post_exp[i])) {
            string op1 = s.top();
            s.pop();
            string op2 = s.top();
            s.pop();
            string temp = post_exp[i] + op2 + op1;
            s.push(temp);
        }
        else {
             s.push(string(1, post_exp[i]));
        }
    }
    string ans = "";
    while (!s.empty()) {
        ans += s.top();
        s.pop();
    }
    return ans;
}
int main(){
    string e;
    cout<<"Enter the infix expression: ";
    cin>>e;
    string pre=infixToPrefix(e);
    string post=infixToPostfix(e);
    cout<<"Prefix: "<<pre<<"\nPostfix: "<<post;
    cout<<"\nPostfix to Prefix: "<<postToPre(post);
    return 0;
}
```

Output:

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

Result: Thus, Intermediate code generation conversion to Postfix and Prefix implemented successfully.