East West University

Department of CSE

Title : Write program to generate assembly code from prefix code

Course title: Computer Architecture

Course code: CSE360(2)

Semester: FALL 2021

SUBMITTED BY:

Student Name	Student ID
Sazzad Hossen	2019-1-60-063
Sadab Jaowad	2019-1-60-082
Abdul Malek Sarder	2019-1-60-087

SUBMITTED TO :

Dr. Ahmed Wasif Reza

Associate Professor

Department of Computer Science & Engineering

Infix to prefix conversion Process:

- Step 1: Reverse the infix string. Note that while reversing the string you must interchange left and right parentheses.
- Step 2: Obtain the postfix expression of the infix expression Step 1.
- Step 3: Reverse the postfix expression to get the prefix expression

Infix to prefix Code :

```
#include <bits/stdc++.h>
using namespace std;

bool isOperator(char c)
{
    return (!isalpha(c) && !isdigit(c));
}

int getPriority(char C)
{
    if (C == '-' || C == '+')
        return 1;
    else if (C == '*' || C == '/')
        return 2;
    else if (C == '^')
        return 3;
    return 0;
}

string infixToPostfix(string infix)
{
    infix = '(' + infix + ')';
    int l = infix.size();
    stack<char> st;
    string v;

    for (int i = 0; i < l; i++) {

        if (isalpha(infix[i]) || isdigit(infix[i]))
            v += infix[i];
        else if (infix[i] == '(')
            st.push('(');

        else if (infix[i] == ')') {
            while (st.top() != '(') {
                v += st.top();
                st.pop();
            }
            st.pop();
        }

        else {
            if (isOperator(st.top())) {
                while (getPriority(infix[i])
                   <= getPriority(st.top())) {
                    v += st.top();
                    st.pop();
                }
                st.push(infix[i]);
            }
        }
    }
    return v;
}

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);
```

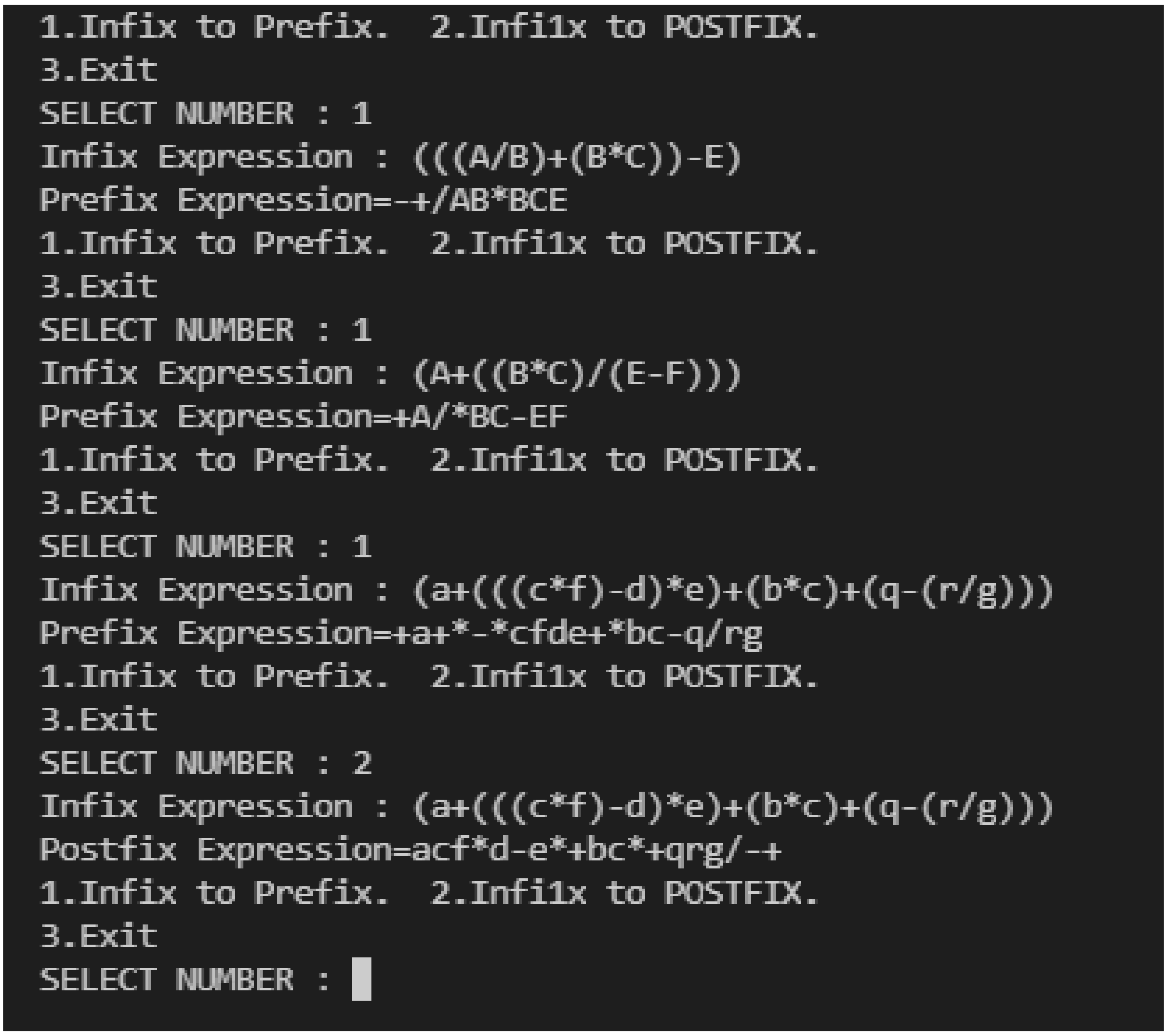
```cpp
    reverse(prefix.begin(), prefix.end());
    return prefix;
}

int main()
{
    while (1)
    {
        printf("1.Infix to Prefix.  2.Infi1x to 
POSTFIX.\n3.Exit\nSELECT NUMBER : ");
        int ch;
        scanf("%d", &ch);
        if (ch == 1)
        {
            string str;
            printf("Infix Expression : ");
            cin >> str;
            cout <<"Prefix Expression="<< infixToPrefix(str) 
<<endl ;

        }
        else if (ch == 2)
        {
            string exp;
            printf("Infix Expression : ");
            cin >> exp;
            cout <<"Postfix Expression="<< infixToPostfix(exp) 
<<endl ;

        }
        else if (ch == 3)
            break;
    }
}
```

OUTPUT :



```
1.Infix to Prefix.  2.Infi1x to POSTFIX.
3.Exit
SELECT NUMBER : 1
Infix Expression : (((A/B)+(B*C))-E)
Prefix Expression=-+/AB*BCE
1.Infix to Prefix.  2.Infi1x to POSTFIX.
3.Exit
SELECT NUMBER : 1
Infix Expression : (A+((B*C)/(E-F)))
Prefix Expression=+A/*BC-EF
1.Infix to Prefix.  2.Infi1x to POSTFIX.
3.Exit
SELECT NUMBER : 1
Infix Expression : (a+(((c*f)-d)*e)+(b*c)+(q-(r/g)))
Prefix Expression=+a+*-*cfde+*bc-q/rg
1.Infix to Prefix.  2.Infi1x to POSTFIX.
3.Exit
SELECT NUMBER : 2
Infix Expression : (a+(((c*f)-d)*e)+(b*c)+(q-(r/g)))
Postfix Expression=acf*d-e*+bc*+qrg/-+
1.Infix to Prefix.  2.Infi1x to POSTFIX.
3.Exit
SELECT NUMBER : 
```

Time Complexity :

Stack operations like push() and pop() are performed in constant time. Since we scan all the characters in the expression once the complexity in linear in time O(n).