#include<iostream>

#include<string>

using namespace std;

const int Max=30;

class Stack

{

    public:

    int top;

    char arr[Max];

    Stack()

    {

        top = -1;

    }

    bool Empty()

    {

        if(top==-1)

            return 1;

        else return 0;

    }

    bool Full()

    {

        if(top==Max-1)

            return 1;

        else return 0;

    }

    void Push(char x)

    {

        if(Full())

            cout << "Stack overflow";

        else

            top++;

            arr[top] = x;

    }

    void Pop()

    {

        if(Empty())

            cout << "Stack underflow";

        else

        {

            arr[top] = 0;

            top--;

        }

    }

    char top1()

    {

        return arr[top];

    }

};

bool isOperator(char c)

{

    if(c=='+' || c=='-' || c=='/' || c=='\*' )

        return true;

    else

        return false;

}

int precedence(char c)

{

    if(c=='\*' || c=='/')

    return 2;

    else if(c=='+' || c=='-')

    return 1;

    else

    return -1;

}

string InfixtoPostfix(Stack s, string infix)

{

    string postfix;

    for(int i=0; i<infix.length(); i++)

    {

        if((infix[i]>='a' && infix[i]<='z')

        ||(infix[i]>='A' && infix[i]<='Z'))

        {

            postfix += infix[i];

        }

        else if(infix[i]=='(')

        {

            s.Push(infix[i]);

        }

        else if(infix[i]==')')

        {

            while((s.top1()!='(') && (!s.Empty()))

            {

                postfix += s.top1();

                s.Pop();

            }

            if(s.top1()=='(')

            {

                s.Pop();

            }

        }

        else if ( isOperator(infix[i]) )

        {

            if(s.Empty())

            {

                s.Push(infix[i]);

            }

            else

            {

                if( precedence(infix[i]) > precedence(s.top1()) )

                {

                    s.Push(infix[i]);

                }

                else

                {

                    while (!s.Empty() && (precedence(infix[i]) <= precedence(s.top1())))

                    {

                        postfix += s.top1();

                        s.Pop();

                    }

                    s.Push(infix[i]);

                }

            }

        }

    }

    while(!s.Empty())

    {

        postfix += s.top1();

        s.Pop();

    }

    return postfix;

}

int main()

{

    string infix\_exp,postfix\_exp;

    cout << "Enter infix expression: ";

    getline(cin,infix\_exp);

    Stack st;

    cout << "INFIX EXPRESSION: " << infix\_exp << endl;

    postfix\_exp = InfixtoPostfix(st,infix\_exp);

    cout << endl << "POSTFIX EXPRESSION: " << postfix\_exp;

    return 0;

}

Output

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

INFIX EXPRESSION: A+B\*(C-D)

POSTFIX EXPRESSION: ABCD-\*+