#include<iostream>

#include<stack>

using namespace std;

bool isOperator(char c)

{

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

{

return true;

}

else

{

return false;

}

}

int precedence(char c)

{

if(c == '^')

return 3;

else if(c == '\*'||c=='/')

return 2;

else if(c == '+' || c == '-')

return 1;

else

return -1;

}

string InfixToPostfix(stack<char> 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.top()!='(') &&(!s.empty()))

{

char temp=s.top();

postfix+=temp;

s.pop();

}

if(s.top()=='(')

{

s.pop();

}

}

else if(isOperator(infix[i]))

{

if(s.empty())

{

s.push(infix[i]);

}

else

{

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

{

s.push(infix[i]);

}

else

{

if((precedence(infix[i] )==precedence(s.top()))&&(infix[i]=='^'))

{

s.push(infix[i]);

}

else

{

while((!s.empty())&&( precedence(infix[i])<=precedence(s.top())))

{

postfix+=s.top();

s.pop();

}

s.push(infix[i]);

}

}

}

}

while(!s.empty())

{

postfix+=s.top();

s.pop();

}

return postfix;

}

int main()

{

string infix\_exp, postfix\_exp;

cout<<"Enter a Infix Expression :"<<endl;

cin>>infix\_exp;

stack <char> stack;

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

postfix\_exp = InfixToPostfix(stack, infix\_exp);

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

return 0;

}