Infix, Postfix and Prefix notations are three different but equivalent ways of writing expressions. It is easiest to demonstrate the differences by looking at examples of operators that take two operands.

**Infix notation: X + Y**

Operators are written in-between their operands. This is the usual way we write expressions. An expression such as A \* ( B + C ) / D is usually taken to mean something like: "First add B and C together, then multiply the result by A, then divide by D to give the final answer."

**Postfix notation (also known as "Reverse Polish notation"): X Y +**

Operators are written after their operands. The infix expression given “A \* ( B + C ) / D ”is equivalent to “A B C + \* D / ”.

**Prefix notation (also known as "Polish notation"): + X Y**

Operators are written before their operands. The expressions “A \* ( B + C ) / D ” are equivalent to “/ \* A + B C D  ”.

**Source Code:**

#include<bits/stdc++.h>

using namespace std;

int get\_operator\_priority(char c)

{

int p = 0;

switch (c)

{

case '\*':

case '/':

return 2;

case '+':

case '-':

return 1;

default:

return 0;

}

}

class PolishNotation

{

private:

string Expre;

string prefixToInfix();

public:

PolishNotation(string str);

string infixToPostfix();

string infixToprefix();

string prefixToPostfix();

};

PolishNotation::PolishNotation(string str)

{

Expre=str;

}

string PolishNotation::infixToPostfix()

{

stack<char>Stk;

string ret="";

for(int i=0;i<Expre.size();i++)

{

char curr=Expre[i];

if(isalpha(curr))ret+=curr;

else if(curr=='(')Stk.push(curr);

else if(curr==')')

{

char d=Stk.top();

Stk.pop();

while(d!='(')

{

ret+=d;

d=Stk.top();

Stk.pop();

}

}

else

{

while(!Stk.empty() and get\_operator\_priority(Stk.top())>=get\_operator\_priority(curr))

{

ret+=Stk.top();

Stk.pop();

}

Stk.push(curr);

}

}

while(!Stk.empty())

{

ret+=Stk.top();

Stk.pop();

}

return ret;

}

string PolishNotation::infixToprefix()

{

stack<char>Stk;

string ret="";

for(int i=Expre.size()-1;i>=0;i--)

{

char curr=Expre[i];

if(isalpha(curr))ret+=curr;

else if(curr==')')Stk.push(curr);

else if(curr=='(')

{

char d=Stk.top();

Stk.pop();

while(d!=')')

{

ret+=d;

d=Stk.top();

Stk.pop();

}

}

else

{

while(!Stk.empty() and get\_operator\_priority(Stk.top())>=get\_operator\_priority(curr))

{

ret+=Stk.top();

Stk.pop();

}

Stk.push(curr);

}

}

while(!Stk.empty())

{

ret+=Stk.top();

Stk.pop();

}

reverse(ret.begin(),ret.end());

return ret;

}

string PolishNotation::prefixToInfix()

{

stack<string>Stk;

for(int i=Expre.size()-1;i>=0;i--)

{

char curr=Expre[i];

if(isalpha(curr))Stk.push(string("")+curr);

else

{

string A=Stk.top();

Stk.pop();

string B=Stk.top();

Stk.pop();

string C="";

if(curr=='+' or curr=='-')C=A+curr+B;

else if(curr=='/')C="("+A+")"+curr+"("+B+")";

else if(curr=='\*')C="("+A+")"+curr+B;

Stk.push(C);

}

}

return Stk.top();

}

string PolishNotation::prefixToPostfix()

{

Expre=prefixToInfix();

return infixToPostfix();

}

int main()

{

string infix="((a+b-c)\*(d-e)/(f-g+h))";

string prefix="\*+a-bc/-de+-fgh";

PolishNotation Infix(infix);

cout<<"Infix : "<<infix<<" to prefix : "<<Infix.infixToprefix()<<endl<<endl;

cout<<"Infix : "<<infix<<" to postfix : "<<Infix.infixToPostfix()<<endl<<endl;

PolishNotation Prefix("\*+a-bc/-de+-fgh");

cout<<"prefix : "<<prefix<<" to postfix : "<<Prefix.prefixToPostfix()<<endl<<endl;

return 0;

}

Sample Input:

((a+b-c)\*(d-e)/(f-g+h))

\*+a-bc/-de+-fgh

Sample Output:

