PreFix, InFix, PostFix Conversion

Conversion Calculators

http://www.mathblog.dk/tools/infix-postfix-converter/

http://scanftree.com/Data_Structure/prefix-postfix-infix-online-converter

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."
>
> Infix notation needs extra information to make the order of evaluation of the operators clear: rules built into the language about operator precedence and associativity, and brackets `( )` to allow users to override these rules. For example, the usual rules for associativity say that we perform operations from left to right, so the multiplication by A is assumed to come before the division by D.

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

> Operators are written after their operands. The infix expression given above is equivalent to `A B C + * D /`
> The order of evaluation of operators is always left-to-right, and brackets cannot be used to change this order. Because the "+" is to the left of the "*" in the example above, the addition must be performed before the multiplication.
> Operators act on values immediately to the left of them. For example, the "+" above uses the "B" and "C". We can add (totally unnecessary) brackets to make this explicit:
> `( (A (B C +) *) D /)`
> Thus, the "*" uses the two values immediately preceding: "A", and the result of the addition. Similarly, the "/" uses the result of the multiplication and the "D".

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

> Operators are written before their operands. The expressions given above are equivalent to `/ * A + B C D`
> As for Postfix, operators are evaluated left-to-right and brackets are superfluous.
> Operators act on the two nearest values on the right. I have again added (totally

unnecessary) brackets to make this clear:

```
(/ (* A (+ B C) ) D)
```

Although Prefix "operators are evaluated left-to-right", they use values to their right, and if these values themselves involve computations then this changes the order that the operators have to be evaluated in. In the example above, although the division is the first operator on the left, it acts on the result of the multiplication, and so the multiplication has to happen before the division (and similarly the addition has to happen before the multiplication).

Because Postfix operators use values to their left, any values involving computations will already have been calculated as we go left-to-right, and so the order of evaluation of the operators is not disrupted in the same way as in Prefix expressions.

In all three versions, the operands occur in the same order, and just the operators have to be moved to keep the meaning correct. (This is particularly important for asymmetric operators like subtraction and division: `A - B` does not mean the same as `B - A`; the former is equivalent to `A B -` or `- A B`, the latter to `B A -` or `- B A`).

Find the prefix and postfix notation for the following infix expression:

(a + b – c) * (e / f) – (g – h/i)

There is a quick way to convert from one notation to another. You start by inserting all the ***implicit*** brackets/parentheses that determine the order of evaluation, regardless of what notation the expression is already in. This is called **fully parenthesizing** an expression. So, taking the expression above and adding these implicit brackets gives:

```
This is before:
(a + b - c) * (e / f) - ( g - h/i)

This is after adding the implicit parentheses:
( ( ( (a + b) - c) * (e / f)) - ( g - (h/i)))
```

Now, in order to convert from one notation to another using the bracketed form, you would move the operator in each bracketed expression. Where you move the operator depends on the notation that you want to convert to.

Postfix vs. Prefix Notation

If you want to convert to postfix notation, you would move the operator to the end of the bracketed expression, right before the closing brace. To convert to prefix notation, you would move the operator to the beginning of the bracketed expression, right after the opening brace. So, (h/i) in postfix notation would look like (h i /), and in prefix notation would look like (/ h i). Do this for every operator in a bracket. So, converting the expression above to prefix notation will give you:

Moving all the operators to the beginning of the bracketed expression for prefix notation gives us:

```
( - ( *  (  - ( + a b) c)  ( / e f))  ( - g ( / h i ) ) )
```

And finally, removing all the parentheses gives us our final prefix notation:

```
- * - + a b c / e f - g / h i
```

Try figuring out how to get the postfix notation on your own using the rules given above. You should get this as your answer:

```
a b + c - e f / * g h i / - -
```

Another Example: convert from infix to postfix

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

Step 1: Fully Parenthesis

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

Step 2: Move operators after operands within parentheses

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

Step 3: Remove parentheses

```
a b / c + d e f - * g / -
```

Convert from postfix to infix

a b / c + d e f - * g / -

Step 1: Fully Parenthesis

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

Step 2: Move operators between operands within parentheses

(((a / b) + c) – ((d * (e – f)) / g))

Step 3: Keep parentheses to retain order of operation

(((a / b) + c) – ((d * (e – f)) / g))