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. 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 depend 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 / - -
```

Some Examples

Infix Expression	Prefix Expression	Postfix Expression
A + B * C + D	+ + A * B C D	A B C * + D +
(A + B) * (C + D)	* + A B + C D	A B + C D + *
A * B + C * D	+ * A B * C D	A B * C D * +
A + B + C + D	+ + + A B C D	A B + C + D +
(A+B)*C-(D-E)*(F+G)	- + A B C * - D E + F G	A B + C * D E – F G + * -
(A+B)*C+D/(E+F*G)-H	- + * + A B C / D + E * F G H	A B + C * D E F G * + / + H -
A-B-C*(D+E/F-G)-H	- - - A B * C - + D / E F G H	A B - C D E F / + G - * - H -
A+((B-C*D)/E)+F-G/H	- + + A / - B * C D E F / G H	A B C D * - E / + F + G H / -

Conversion using Stack

Ex1: Converting Infix to postfix using Stack

A + B *C–D / E

infix	stack	postfix
A + B * C – D / E	#	
+ B * C – D / E	#	A
B * C – D / E	# +	A
* C – D / E	# +	A B
C – D / E	# + *	A B C
– D / E	# + *	A B C * +
D / E	# -	A B C * + D
/ E	# -	A B C * + D
E	# - /	A B C * + D E
	# - /	A B C * + D E
	#	A B C * + D E / -

Ex2: Converting Infix to Postfix: A * B – (C + D)+ E

infix	stack	postfix
A * B – (C + D) + E	#	
* B – (C + D) + E	#	A
B – (C + D) + E	# *	A
– (C + D) + E	# *	A B
(C + D) + E	#	A B *
C + D) + E	# -	A B *
+ D) + E	# - (	A B *
D) + E	# - (	A B * C
) + E	# - (+	A B * C
+ E	# - (+	A B * C D
E	# -	A B * C D +
	#	A B * C D + -
	# +	A B * C D + -
	# +	A B * C D + - E
	#	A B * C D + - E +

Steps to Evaluate Postfix Expression

1. If char read from postfix expression is an operand, push operand to stack.
2. If char read from postfix expression is an operator, pop the first 2 operand in stack and implement the expression using the following operations:
    a. pop(opr1) and pop(opr2)
    b. result = opr2 operator opr1
3. Push the result of the evaluation to stack. Repeat steps 1 to steps 3 until end of postfix expression

    Finally, at the end of the operation, only one value left in the stack. The value is the result of postfix evaluation.

Evaluating Postfix Expression

```
Create Stack
while (not end of postfix notation)
{      ch = getch()
if (ch is operand)
                push (ch)
        else
        {       operan1 = pop()
                operan2 = pop()
                result = operan2 ch operan1
                push(result)
        }
}
result = pop()
```

Ex1: Evaluating Postfix Expression: 2 4 6 +*

postfix	Ch	Opr	Opn1	Opn2	result	stack
2 4 6 + *						
4 6 + *	2					2
6 + *	4					2 4
+ *	6					2 4 6
*	+	+	4	2	6	2 10
	*	*	10	2	20	20

Evaluating Postfix Expression: 2 7 * 18 - 6 +

postfix	Ch	Opr	Opn1	Opn2	result	stack
2 7 * 18 – 6 +						
7 * 18 – 6 +	2					2
* 18 – 6 +	7					2 7
18 – 6 +	*	*	7	2	14	14
– 6 +	18					14 18
6 +	–	–	18	14	-4	-4
+	6					-4 6
	+	+	6	-4	2	2