**Projects Stacks**

Transforming an Infix Expression to a Postfix Expression

In a postfix expression, a binary operator follows its two operands. Here are a few examples of infix expressions and their corresponding postfix forms:

**Infix Postfix**

a + b a b +

(a + b) \* c a b + c \*

a + b \* c a b c \* +

Notice that the order of the operands a, b and c in an infix expression is the same in the corresponding postfix expression. However, the order of the operators might change. This order depends on the precedence of the operators and the existence of parentheses. As we mentioned, parentheses do not appear in a postfix expression.

**The basics of a conversion algorithm.** To convert an infix expression to postfix form, we scan the infix expression from left to right. When we encounter an operand, we place it at the end of the new expression that we are creating. Recall that operands in an infix expression remain in the same order in the corresponding postfix expression. When we encounter an operator, we must save it until we determine where in the output expression it belongs. For example, to convert the infix expression a + b, we append a to the initially empty output expression, save +, and append b to the output expression. We now need to retrieve the + and put it at the end of the output expression to get the postfix expression a b +. Retrieving the operator saved most recently is easy if we have saved it in a stack.

**Converting the infix expression a + b \* c to postfix form**

Next Character in Postfix Form Operator Stack

Infix Expression

a a

+ a +

b a b +

\* a b + \*

c a b c + \*

a b c \* +

a b c \* +

Figure 1.

**Successive operators with the same precedence.** What if two successive operators have the same precedence ? We need to distinguish between operators that have a left-to-right association – namely +, -, \* and / - and exponentiation, which has a right-to-left association. For example, consider the expression a- b + c. When we encounter the +, the stack will contain the operator – and the incomplete postfix expression will be ab. The subtraction operator belongs to the operands a and b, so we pop the stack and append – to the end of the expression ab. Since the stack is empty, we push the + onto the stack. We then append c to the result, and finally we pop the stack and append the +. The result is a b – c +.

Converting an infix expression to postfix form:

1. a – b + c ; (b) a ^ b ^ c

Next Character in Postfix Form Operator Stack

Infix Expression (bottom to top)

a a

* a -

b a b -

+ a b – +

c a b – c +

a b – c +

Figure 2.

Consider the expression a^b^c. By the time we encounter the second exponentiation operator, the stack contains ^, and the result so far is ab. As before, the current operator has the same precedence as the top entry of the stack. But since a^b^c means a^(b^c), we must push the second ^ onto the stack.

1. a – b + c ; (b) a ^ b ^ c

Next Character in Postfix Form Operator Stack

Infix Expression (bottom to top)

a a

^ a ^

b a b ^

^ a b ^^

c a b c ^^

a b c ^ ^

a b c ^^

Figure 3.

**Parentheses** Parentheses override the rules of operator precedence. We will push open parenthesis onto the stack. Once it is in the stack, we treat an open parenthesis as an operator with the lowest precedence. Any subsequent operator will get pushed on the stack. When we encounter a close parenthesis, we will pop operators from the stack and append them to the forming postfix expression until we pop an open parenthesis.

To convert an infix expression to postfix form, you take the following actions:

* Operand Append each operand to the end of the output expression.
* Operator^ Push ^ onto the stack.
* Operator +, -, \* or / Pop operators from the stack, appending them to the output expression, until the stack is empty or its top entry has a lower precedence then the new operator. Then push the new operator onto the stack.
* Open parenthesis Push ( onto the stack.
* Close parenthesis Pop operators from the stack and append them to the output expression until an open parenthesis is popped. Discard both parentheses.

Requirements:

Define a class Postfix that includes the static methods converttoPostfix and evaluatePostfix. These methods should implement the infix-to-postfix algorithm and the evaluation algorithm.

Assume that the given algebraic expressions are syntactically correct. The standard class StringBuilder, which is in the JAVA Class Library will be helpful.