COSC 2336
LAB ASSIGNMENT III
DUE DATE: Thursday, 2 April 2015

Write a Java program which will input sequences of characters representing infix expressions involving the binary operators +, -, *, and / plus parentheses and single digit operands. The program should output the equivalent postfix expressions AND the values of those expressions. For example:

INPUT: **6+9*(5*(3+4))** OUTPUT: **6 9 5 3 4 + * * + = 321**

As discussed in class, Dijkstra's Shunting Algorithm can be employed to convert an infix expression into an equivalent postfix expression using a stack for operators. Furthermore, a postfix expression can be evaluated using a stack for operands. The logical design provided on the next page combines these two procedures into one algorithm.

The Shunting Algorithm requires the use of two methods: *inputPriority* and *stackPriority*. *inputPriority* (with *token* as its actual parameter) returns the priority value of an operator when it is on the input string. *stackPriority* (with top element of operator stack as actual parameter) returns the priority value of an operator on the stack. Use the following table for implementation of these methods:

OPERATOR	**INPUT PRIORITY**	**STACK PRIORITY**
;	**NA**	**0**
+	**2**	**2**
-	**2**	**2**
*****	**3**	**3**
/	**3**	**3**
(	**4**	**1**

Since the algorithm requires two stacks (each is comprised of different element types), an interface (i.e. general specification) for a stack abstract data type should be defined. The implementation of the operator stack class used in your program should reference data elements of type **char**acter; similarly, the implementation of the operand stack class should reference data elements of **int**eger type. Both stack classes should be implemented using the reference-based model.

Be sure to follow the techniques of good programming style and use extensive comments to provide for internal documentation of your source program. For evaluation of this lab assignment, you will be required to submit *listings* of your source program file(s), your input data file, and your output file (these listings should be individually stapled and all clipped together). It is highly preferable that your source file listing be printed in landscape mode. Please submit these on or before the due date.

```
Initialize the operator stack to contain a ';' (bottom of stack operator)
Get the first token
while the end of the expression is not reached
     if the token is an operand then
          Print the token
          Push the operand onto the operand stack
     else if the token is a ')' then
               while the top of the operator stack is not equal to '('
                    Pop the operator stack
                    Print the operator
                    Pop the operand stack twice
                    Apply the designated operation to the two operands
                    Push the operation result onto the operand stack
               end while
               Pop the '(' and discard it
     else
          while inputPriority(token) ≤ stackPriority(top of operator stack)
               Pop the operator stack
               Print the operator
               Pop the operand stack twice
               Apply the designated operation to the two operands
               Push the operation result onto the operand stack
          end while
          Push the token onto the operator stack
     Get the next token
end while

while the top of the operator stack is not equal to ';'
     Pop the operator stack
     Print the operator
     Pop the operand stack twice
     Apply the designated operation to the two operands
     Push the operation result onto the operand stack
end while

Pop the operand stack and print the result
```