EECS 233 Homework 4

General requirements:

- Due at 11:00 PM on the posted due date.
- Include your name and network ID as a comment at the top of all of your programs.
- Create a typed document (.docx or .pdf) for the report with your name and network ID at the top.
- Upload your document and all .java files as a .zip file to Canvas. Do not use other formats such as .rar.
- All work should be your own, as explained in the Academic Integrity policy from the syllabus.

Instructions: This assignment requires you to use the following programs from the textbook:

- ArrayStack.java
- LinkedStack.java
- Node.java
- IsBalancedDemonstration.java
- EvaluateDemonstration.java

The following additional programs are provided and discussed in lecture:

- ArrayStackDemo.java
- LinkedStackDemo.java

1. This problem is intended to better understand the use of the Stack class. Write a program that checks an expression only for balanced parentheses (not curly or square brackets) without using an array or stack. Hint: count the parenthesis level using an integer that increases or decreases, depending on how many unmatched parentheses have appeared. The program should follow the same principles as IsBalancedDemonstration.java from the textbook, except that you will not include curly or square brackets. Answer the following questions in your report.
   a. What is the Big O time complexity of the algorithm in IsBalancedDemonstration.java that uses Stack?
   b. Does the new "counting" algorithm have the same time complexity as the algorithm that uses Stack? Explain why or why not in 2 – 3 sentences. Note that this is similar to the analysis in the textbook for evaluating arithmetic expressions (p 325).
   c. Can this "counting" algorithm be used to evaluate expressions with all three kinds of parentheses and brackets, including curly and square, without using an array or stack? Explain why or why not in 2 – 3 sentences. NOTE: You do not need to write a program for this part.

2. Using the algorithm in EvaluateDemonstration.java from the textbook, write (in your report) the contents of both stacks before and after each right parenthesis is evaluated in a fully parenthesized expression. For convenience, you may write the stack values horizontally. An example analysis is provided below for the expression ((6+9)/3):

Character	*Numbers*	*Operations*
(		
(		
6	*6*	
+	*6*	*+*
9	*6, 9*	*+*
)	*15*	
/	*15*	*/*
3	*15, 3*	*/*
)	*5*	

   a. `(((1+2)+3)+4)`
   b. `(1+(2+(3+4)))`
   c. `(((2*(3+4))-5)/3)`

3. The following tasks require ArrayStack.java, LinkedStack.java, and EvaluateDemonstration.java from the textbook.
   a. Modify the ArrayStack and LinkedStack classes to have a "print" method that prints all of the stack values in a row, separated by spaces.
   b. Modify EvaluateDemonstration.java to use ArrayStack for storing numbers and LinkedStack for storing operations. Note that the results should not change, even though the stacks use different implementations.
   c. Use your new "print" methods from (a) to check your answers from problem #2 by printing the stack values after each character in the expression. Provide your program output in your report. Below is an example of partial output for the expression ((6+9)/3). Not all output is shown.

```
Character:(  Numbers=  Operators=
Character:(  Numbers=  Operators=
Character:6  Numbers=6  Operators=
Character:+  Numbers=6  Operators=+
Character:9  Numbers=6,9  Operators=+
```

4. Write the following postfix expressions in equivalent <u>fully parenthesized infix</u> form. Do <u>not</u> calculate a numerical result.

   a. 5 8 3 7 – / – = (   )
   b. 5 8 – 3 7 / – = (   )

5. Write the following expressions in equivalent <u>postfix</u> form. Do <u>not</u> calculate a numerical result.

   a. (4 – ((6 – 2) / 8)) =
   b. ((4 – (6 / 2)) – 8) =

6. Write a program that asks the user for a <u>postfix</u> expression and computes the result using the built-in Stack class. Note that pseudocode is provided in the textbook (Figure 6.8 on p 347). Two output examples are given below (user input in **bold**):

`Enter expression: `**`2 3 + 4 *`**
`Result = 20.0`
	`Enter expression: `**`6 5 2 3 + 8 * + 3 + *`**
	`Result = 288.0`

Tip (full parenthesization): Notice that IsBalancedDemonstration.java does not check for a fully parenthesized expression. Therefore, your solution for #1 does not need to check this either. For example, ()() is considered valid for #1. However, full parenthesization <u>is</u> required for EvaluateDemonstration.java (and #2 and #3).

Tip (printing stacks bottom-to-top and left-to-right): For #3, the example output shows the stacks printed horizontally, and the bottom should be displayed on the left. This is fairly easy for ArrayStack because you can start printing from element #0. For LinkedStack, you can begin at the "top" and build a new String in reverse order. That is, start with an empty String, and add the data for each node to the <u>front</u> of the String. When all the node data has been combined into a single String, just print it.

Tip (when and where to print): For #3, each of the two stack classes should be able to print its own stack values, as explained in part (a). In part (b), you'll modify EvaluateDemonstration to print a whole row of information for each parenthesis, number, and operation in the expression String. For each row of output, print the character and call the "print()" methods for each of the two stacks.

Tip (postfix user input): For #6, the user input code can remain the same as EvaluateDemonstration.java (except the user instructions, which aren't necessary). Assume the user correctly types space between the numbers so that the Scanner class can know where numbers start and stop. You should keep whatever error checking is in the original version, such as valid characters.

Tip (evaluateStackTops): In EvaluateDemonstration, the evaluate() method calls another method named evaluateStackTops() that performs the actual math operations. For example: `numbers.push(operand1 + operand2)`. This was useful because all operators were pushed onto their own stack, and the math operations were only performed when a right parenthesis was encountered. For #6, however, the math should be performed every time an operator character is encountered. You may find that it makes more sense to eliminate evaluateStackTops() and do the math operations directly in evaluate().

Rubric:

Item	**Points**
General programming requirements	5
1. Using original IsBalancedDemonstration.java	5
1. General counting algorithm (basic idea gets full credit)	5
1. Correctness of counting algorithm	5
1. (a), (b), and (c): 4 points each	12
2. (a), (b), and (c): 4 points each	12
3. Using original three .java programs	5
3(a). Adding print(). 5 points for each stack class	10
3(b). Modifying main() method for printing	5
3(c). Including output in report (full credit for any output)	4
4. (a) and (b): 3 points each	6
5. (a) and (b): 3 points each	6
6. Using original EvaluateDemonstration.java	5
6. Correct use of push() and pop(), even if logic is wrong	5
6. Performing math operations, even if incorrect location	5
6. Correct algorithm	5
Total	100