CSCI 2110 – Data Structures in Java
Lab exercise LabN4

- Deadline: Friday Feb 8th at 5pm
- Please first read the assignment check list available in the following link. If you do not follow the check list, you lose many points unfortunately. https://web.cs.dal.ca/~nourashr/teaching/cs2110/assignments/assignmentchecklist/
- If you have any question regarding the questions, please send an email to the instructor or TAs.
- Please submit your work on the Moodle portal: https://courses.cs.dal.ca/login/index.php
- You probably need to review lecture slides which are available in the following link. Feel free to use the codes of slides. https://web.cs.dal.ca/~nourashr/teaching/cs2110/lectures/
- We follow Dalhousie's policy of plagiarism. http://www.dal.ca/dept/university_secretariat/academic-integrity.html

Q1) (35 points) In the **Towers of Hanoi** puzzle, we are given a platform with three pegs, *a*, *b*, and *c*, sticking out of it. On peg *a* is a stack of *n* disks, each larger than the next, so that the smallest is on the top and the largest is on the bottom. The puzzle is to move all the disks from peg *a* to peg *c*, moving one disk at a time, so that we never place a larger disk on top of a smaller one. Describe a recursive algorithm for solving the **Towers of Hanoi** puzzle for arbitrary *n*. Implement your algorithm as a Java method and test it on $n = \{5, 01, 15, 20\}$. Show the steps of moving disks for each value of *n* as outputs of your code. (Hint: Consider first the sub-problem of moving all but the n^{th} disk from peg *a* to another peg using the third as "temporary storage.") More information about this puzzle is provided in the following link:

https://en.wikipedia.org/wiki/Tower_of_Hanoi

Q2) (35 points) Write a recursive method to compute all permutations of a string. For instance, all permutations of string “abc” are:

abc, bac, bca, acb, cab, cba (the number of permutations is 3!)

To test your method, print all permutations of the following strings:

S1 = “abc”

S2 = “abcd”

S3 = “abcde”

S4 = “abcdee”

S5 = “abcdeff”

*You are free to use any method or class from Java’s library. You may need to use the following classes.

Java.util.Vector (https://docs.oracle.com/javase/7/docs/api/java/util/Vector.html)
Java.lang.String (http://docs.oracle.com/javase/7/docs/api/java/lang/String.html)

Q3) (15 points) Write a short recursive Java method to take a string and determines (true or false) if the string is palindrome, that is, it is equal to its reverse. For example, “race car” is palindrome. Test your code on the following ten strings and output the results. The case of letters is not important and also ignore the spaces. More information about Palindrome is provided in the following Wikipedia page: https://en.wikipedia.org/wiki/Palindrome.

S1 = Was it a car or a cat I saw

S2 = step on no pets

S3 = stack cats

S4 = race car

S5 = stack cats

S6 = In girum imus nocte et consumimur igni

S7 = Do geese see God

S8 = Mr Owl ate my metal worm

S9 = Eva can I stab bat in a cave

S10 = Go hang a salami I am a lasagna hog

Q4) (15 points) In this question you see the difference between recursive programming and iterative programming.

a) (3 points) Write a short Java method to find the minimum value in an array of int values using loops. Print the minimum value. What is the running time?
b) (3 points) Write a short **recursive** Java method to find the minimum value in an array of int values without using any loops. Print the minimum value. What is the running time?
c) (2 points) Test your methods on 5 randomly-generated arrays of length ten.
d) (7 points) Measure the running time of the methods on randomly-generated arrays of the following length. Which method is faster and why?

Length of array	Running time of the recursive method	Running time of the iterative method
10		
100		
1000		
10,000		
100,000		
1,000,000		
10,000,000		

Q5) (20 bonus points) Implement an algorithm to print all valid (e.g., properly opened and closed) combinations of n-pairs of parentheses.

EXAMPLE:

input: 3 (e.g., 3 pairs of parentheses)

output: ()()(), ()(()), (())(), ((()))