CS 1110, LAB 7: RECURSION

Name: _____ **Net-ID**: _____

There is an online version of these instructions at

http://www.cs.cornell.edu/courses/cs1110/2012sp/labs/lab07.php

You may wish to use that version of the instructions.

This lab gives you experience with writing recursive functions. All of the functions in this lab while either be recursive functions on Strings, or recursive functions on integers, just as we saw in class.

Requirements For This Lab. The very first thing that you should do in this lab is to download the file `Recursive.java` from the course web page:

http://www.cs.cornell.edu/courses/cs1110/2012sp/labs/lab07/Recursive.java

You should create a new directory on your hard drive for this file. You will note that all of the functions in this file are stubs, and are not fully implemented. Your entire lab will consist of modifying this file.

To successfully complete this lab, you should **implememnt the first four methods in the file** `Recursive.java`. When you have done this, show your file to a TA. As always, you should try to finish the lab during your section. However, if you do not finish during section, you have **until the beginning of lab next week to finish it**. You should always do your best to finish during lab hours; remember that labs are graded on effort, not correctness.

If you get stuck, do not waste time. Ask the TA or consultant for help.

---

1. Recursive Methods

Remember that creating and understanding a recursive function involves four points:

A precise specification of the function

Without this, you cannot write the function.

Handling the base case properly

The base case involves the "smallest" parameter values, for which the result can be given easily, without the need for recursive calls. For a function that works on a `String`, the base case is a `String` of length 0, or 0 and 1, usually, but it could be something else. For a function that works on the natural numbers `0, 1, 2, ...`, the base case is `0`, or `0` and `1`, usually.

Handling the recursive case properly

> Solve the original problem in terms of the same problem but on a smaller value. For example, if the function involves a `String s`, the solution should be describable in terms of the same problem on some substring of `s`. In writing/understanding a recursive call, understand it in terms of the *specification* of the function being called.

Making progress toward termination

> In keeping with point 3, the arguments of a recursive call must be in some measure smaller than the parameters of the method; this is what ensures termination. Each recursive call has "smaller arguments", so that after some point, the base case will be reached.

In this lab, you are to implement the first four methods from the file `Recursive.java`. These are the ones listed int the table below:

Method	Description
`numberc(char c, String s)`	Yields: number of times c occurs in s.
`numberNotc(char c, String s)`	Yields: number of chars in s that are not c.
`replace(String s, char c, char d)`	Yields: a copy of s but with all occurrences of c replaced by d. **Example**: replace("abeabe", 'e', '$') = "ab$ab$". For the lab, do **not** use the pre-existing String function replace
`removeDups(String s)`	Yields: a copy of s with adjacent duplicates removed. **Example**: for s = "abbcccdeaaa", the answer is "abcdea".

Even though we only ask you to work on the first four functions in lab, you will get greater fluency in recursion if you do them all. So, during the week, every one in a while write one of the functions and test it. You should particularly try some of the integer recursive functions that appear later in `Recursive.java`

---

2. (Optional) Recursion and Memory

This is a completely optional exercise for your own education and enjoyment.

At the very end of `Recursive.java` are several functions called `test()`. In class, we talked about what happens when you forget your base case: the recursive method keeps creating frames until it runs out of memory. How many frames can it create before it runs out of memory? To find out, hit the Reset button and then type these lines in the Interactions pane, one at a time:

```
Recursive.number= 0;
Recursive.test();
Recursive.number
```

Look at the body of test. From it, you can see that the number in static variable number is the number of calls made, and thus frames created, until there was "stack overflow". Write down that number somewhere.

Next, hit the reset button and perform this test:

```
Recursive.number= 0;
Recursive.test(5);
Recursive.number
```

Here, method test(int), which has one parameter but many local variables, is called many times. Compare the number of frames created here with the number created when calling method test(). Why is one bigger than another?

Finally hit the reset button and perform this test:

```
Recursive.number= 0;
Recursive.test(0, 1, 2, 3, 4, 5, 6, 7, 8, 9);
Recursive.number
```

Here, a method with 10 parameters is called many times. Compare the number of frames created this time to the number of frames created the other times. Do the numbers make any sense to you?