CSC148, Lab #7

This document contains the instructions for lab number 7 in CSC148. To earn your lab mark, you must actively participate in the lab. We mark you in order to ensure a serious attempt at learning, **not** to make careful critical judgments on the results of your work. We will use the same general rules as for the previous labs (including pair programming).

step 0: binary search trees (BSTs)

getting started

- Agree on who will be student **s1** and who will be student **s2** for this lab.
- Read through the entire handout for this lab quickly, so that you have an idea what you will have to do.
- Download the files for this week from the labs page. Read through all of them quickly.

When you are done, get ready to begin pair programming. You should work on steps 1–3. Only work on step 4 if you have time. For each step you will need to:

- Open one of the `BST_rec?.py` files in Idle. Depending on which `BST_rec?.py` you work on, you will need to change the beginning of `BST_test.py` to import that version.
- Read through the code in the file carefully and run `BST_test.py` to see the results. The results are somewhat graphical, so you should calculate **by hand** the expected outputs of `count_less`. Confirm your hand results with your TA before proceeding.

  Make sure you ask questions if there is anything in the code that you don't understand. Do this before you try to write any of your own code.
- Before you move from one step to the next, think about the following questions: do you understand what you just did? Can you describe how your code works and explain why it does the right thing? Or did you manage to get working code by trial-and-error, without really understanding how it works?
- It's OK if you don't fully understand how your code works for now, but keep in mind that you must ask questions (of your partner, of other students, of your TA) to ensure that you do understand by the end of your lab.
- When you are done, run `BST_test.py` again to verify your results.
- Finally, show your work to your TA and switch roles.

step 1: direct recursion

(Student **s1** drives and student **s2** navigates.)

- Implement method `count_less` in class `BST` in file `BST_rec1.py`. For this part, you must do this by writing a nested helper function within `count_less`, and then calling your helper within `count_less`. You are not allowed to add any method to class `_BSTNode` for this part.

  Look over some of the existing code in `BST_rec1.py` for inspiration.

step 2: indirect recursion

(Student **s2** drives and student **s1** navigates.)

- Implement method `count_less` in class `BST` in file `BST_rec2.py`. For this part, you must do this by writing a helper method in class `_BSTNode` and calling the helper within `count_less`. You are not allowed to add any method to class `BST` for this part.

step 3: recursive objects

(Student **s1** drives and student **s2** navigates.)

- Implement method `count_less` in class `BST` in file `BST_rec3.py`. For this part, you must do this by writing a helper method in both classes `_BSTNode` and `_BSTNone` and calling the helpers within `count_less`. You are not allowed to add any method to class `BST` for this part. Notice that the **only** difference between step 3 and step 2 is how `None` is handled — there is no `None` in step 3, but there is a subclass of `_BSTNode` called `BSTNone` that does the work of `None`.

step 4: iteratively (with looping):

(Student **s2** drives and student **s1** navigates.)

- Take the time to review your code from the previous three parts. Compare what you have done with the way that methods `__str__` and `insert` were implemented.
- Now is the time to ask any last questions you have, to make sure that you fully understand how your code works.
- Once you understand all your code, and it all works, think about this: are there ways that you could simplify? (For example, perhaps you have extra base cases that are not necessary because your general case already does the right thing for those base cases).
- Discuss which recursive technique you find easiest to implement. Which one gives the simplest code at the end? Remember that one important goal of programming is to make your programs as easy to understand as possible, both for yourself and for other readers of your code. So it is always a good use of your time to review code after you have written it, in order to clean it up: simplify where you can, remove redundancies, etc.
- **Warning! The rest of this lab is challenging!** It's OK if you don't finish the rest, but you should definitely think about it and get as far into it as you can. It is provided to make you realize that recursion is not your *enemy*, but rather a very useful technique that can save you a lot of trouble — this is why it is so important for you to learn it!

- change `BST_test.py` so that it imports from `BST_iter.py`
- Read through the code in `BST_iter.py` carefully and run `BST_test.py` to see the results. Make sure you ask questions if there is anything in the code that you don't understand. Do this before you try to write any of your own code.
- Implement method `count_less` in class `BST` in file `BST_iter.py`. For this part, you must do this without any form of recursion! But you are allowed to use a separate data structure to manage information...
- When you are done, run `BST_test.py` to verify your results.

If you manage to get this done, show your work to your TA. Then, please stick around to help other students in your lab section!