CS2401 – Weeks 10 & 11

In this lab assignment, we are going to practice using binary trees and binary search trees. We hope you enjoy this assignment! Let's get started!

What is the scenario? Let's build a family tree!

You have recently become passionate about genealogy and have decided to put your family tree together. You write down all information you have on your family tree starting with you, continuing with your parents, and moving from one family generation (level) to the next filling as many slots as possible per level and as many levels as possible.

You put all this in a txt file where each line contains information about a generation:

- Line 1 is you,
- Line 2 is for your parents,
- Line 3 for your grandparents,
- Line 4 for your great-grandparents,
- etc.

Although you are mostly keeping track of you and your direct ancestors, you are also interested in the number of siblings these direct ancestors had. So you plan to add this information as well in your txt file.

[see example of a txt file of your family history: family.txt and family2.txt]

Once all relevant information has been inputted in your file, you'd like to be able to use it in multiple ways:

- Reading the file and creating your family tree
- Traversing and printing the family tree
- Counting the total number of relatives you have been able to track down
- Exploring a specific branch only
- Knowing how deep is your knowledge of your family, starting at any given ancestor in your tree (i.e., how many ancestry levels of this relative were you able to find)
- etc.

Since you just recently started learning about trees in your CS2 class, you decide that you could put your latest Computer Science skills to use.

What information will we handle? And in which format?

To keep track of all you family information, you decide to represent the information as follows:

- Each person in your family tree will be represented as follows:
  - **FirstName-LastName,**
  - **NumberOfSiblings,**

    - **LocationInTree**: LocationInTree is either “0” if it is you, or a sequence of characters ‘M’ and ‘F’, standing for “Mother line” and “Father line”.
- Ancestors at the same level in your genealogical tree will be entered on the same line of your txt file *[see family.txt for an example]*.
- When you have more than one person at a given level (i.e., in a given line of your txt file), you separate these people by a space.
- Note: at some levels, you may not know all names of your ancestors: it is ok not to have as many ancestors at a level as you could.
    - For instance, you have 2 parents, 4 grandparents, 8 great-grandparents.
    - However, you may not know all of your 8 great-grandparents, and it is ok to not put information about 8 people on the 4th line of your txt file.
    - The same goes for any line in your txt file. However, you cannot leave a line empty if it is not the last line of your file.

Below is an example of the content of a family txt file:

```
John-Doe,2,0
Averell-Dalton,1,F Cassidy-Jones,3,M
Luke-Lucky,4,FF Ma-Dalton,2,FM Emil-Jones,6,MF Myrtille-White,4,MM
Pedro-Lucky,10,FFF Mamy-Mist,0,FFM Pa-Dalton,12,FMF Emilia-Jones,5,MFM Cassis-Black,1,MMM
```

In the above file, we see that John Doe has gathered information about his family across 4 levels (there are 4 lines in the file), including himself (located with “0”).
We also see above that John Doe does not know all of his great-grandparents (4th line), and that is ok.
Let’s look at one of his relatives: “Luke-Lucky,4,FF” on line 3. From this entry, we understand that:

- This relative is at the grandparents’ level.
- This relative’s first name is Luke.
- This relative’s last name is Lucky.
- He is a man: line “FF” for Father of Father (hence a man): he is the paternal grandfather of John Doe.
- He had 4 siblings.

Now let’s look at another one: “Mamy-Mist,0,FFM”, on line 4:

- This is a relative at the great-grandparents’ level.
- First name: Mamy
- Last name: Mist
- This relative is a woman. Indeed, from “FFM”, we know that: she was the mother (M) of the father (F) of the father (F) of John Doe.
- Number of siblings: 0

When the number of siblings is unknown, we will use -1 in place of a 0 or positive number of siblings.

So, how will this work? What will you have to do?

Your goal is to be able to read a file like the one presented above and translate it into a tree, which is binary. We saw in class that you can represent a binary tree as an array or as a linked structure, with left and right links. In this lab assignment, you are only asked to implement the family tree as a linked structure.

In order for you to complete this lab assignment, we provide you with several files to support your work. We are describing them below:

- **family.txt, family2.txt**: are two examples of family txt files, which follow the format outlined above. *[You do not have anything to do about this file.]*
- **FamilyMember.java**: In this lab assignment, you are going to manipulate family members about whom you want to know the first name, last name, and number of siblings. We are providing you with the java type FamilyMember. *[You do not have anything to do about this file.]*
- **BTNode.java**: Since you are going to have to handle binary trees as linked lists, we are providing you with the type BTNode, which allows you to create and handle generic nodes that have two links, called left and right. Most of the code is provided to you in this file. However, you are **tasked with completing 3 methods** in this file:
  - **sizeBelow**
  - **hasLeft**
  - **hasRight**
- **BTree.java**: This file provides you with a generic type for binary trees implemented as a structure that handles a root BTNode and a size. Most of the code is provided to you. However, in this file, you are **tasked with completing 2 methods** and offered **a bonus activity**:
  - **print**
  - **insertDataAtLocation**
  - **inOrderTraversal (Bonus)**
- **Execute.java**: This file contains the main method and a few helper methods for your convenience if you elect to use them. You must not modify the code of the main method.
  However, there is **1 method** whose code you need to complete:
  - **readFamilyIntoTree**

BONUS ACTIVITY:

- Complete the file BSTree.java (Binary Search Tree) that extends the type BTree (Binary Tree) by completing the code of its method: insert.
- In Execute.java, create a new method called readFamilyIntoBST: call this method from the main method and print the resulting tree. This method prompts you to insert family members, not anymore by family relationship (parents above grandparents, above great-grandparents, etc.) but rather in the order of the number of siblings, starting at the first family member read, which is you. So for instance, in the case of the file below:

```
John-Doe,2,0
Averell-Dalton,1,F Cassidy-Jones,3,M
Luke-Lucky,4,FF Ma-Dalton,2,FM Emil-Jones,6,MF Myrtille-White,4,MM
Pedro-Lucky,10,FFF Mamy-Mist,0,FFM Pa-Dalton,12,FMF Emilia-Jones,5,MFM Cassis-Black,1,MMM
```

John Doe will be the root of your BST since it is read first.
Then Averell Dalton will be on the left of John Doe because he has fewer siblings (only 1) than John Doe.
Cassidy Jones will be to the right of John Doe because she has more siblings (3) than John Doe.
Luke Lucky will be to the right of Cassidy Jones because he has more siblings than John Doe and more siblings than Cassidy Jones.
Ma Dalton will be to the right of Averell Dalton because she has the same number of siblings as John Doe (so she goes to its left) and more siblings than Averell Dalton (so she goes to its right).
Etc.
Note: this part of the lab is not signaled in the code by a BONUS or TODO. Make sure (if you want to complete this bonus activity) not to forget to do it.

General Note:
All throughout your lab, you are expected to maintain a high quality of code (consistent indentation, line breaks, variable naming, commenting, methods are grouped by category: setters, getters, etc.). Your code quality will be part of your grade on this lab.

Grading: over 100 pts (+30 bonus pts)

20 pts	Code is deemed to be of high quality
BTNode.java: 30 pts	
10 pts	The method sizeBelow performs as expected and is RECURSIVE.
10 pts	Method hasRight performs as expected.
10 pts	Method hasLeft performs as expected.
BTree.java: 30 pts + 5 bonus	
20 pts	The method print performs as expected and is RECURSIVE.
10 pts	The method insertDataAtLocation performs as expected.
+5 pts	BONUS: The method inOrderTraversal performs as expected and is RECURSIVE.
Execute.java: 20 pts	
20 pts	The method readFamilyIntoTree performs as expected.

BONUS (+25 pts):
+10pts BSTree.java: The method insert performs as expected and is RECURSIVE.
+10pts Execute.java: The method readFamilyIntoBST performs as expected.
+ 5pts Execute.java: The main method properly calls readFamilyIntoBST and correctly prints the resulting tree.

Due date: November 9th at 11:59pm

How to submit?
Check with your own TA for submission guidelines.

Failing to follow submission instructions and guidelines will result in up to 15 points off your overall grade in this lab. So please pay attention.