PA3 Help Session

CS200 RECITATION 8

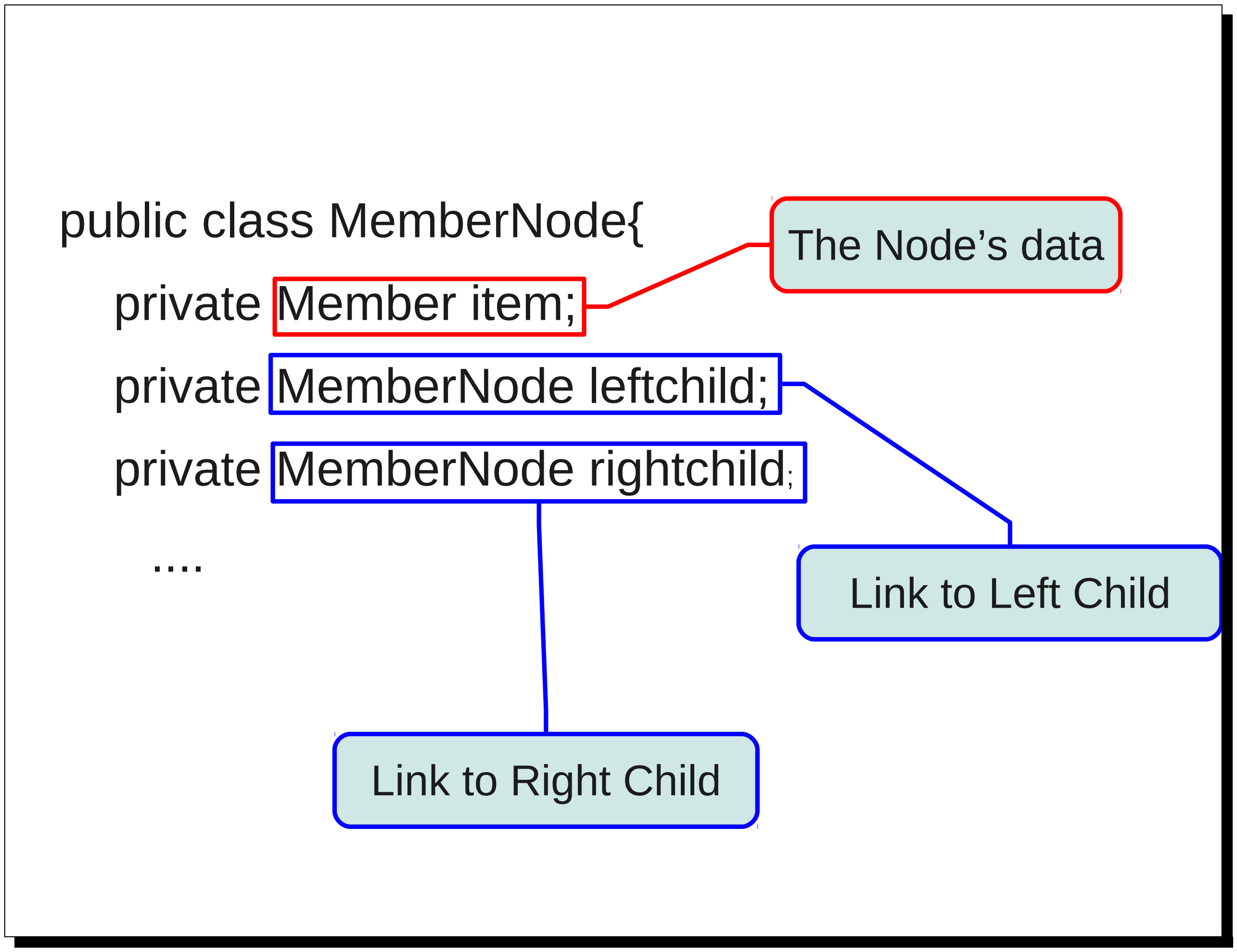
The purpose for today's recitaiton is to help you get started on PA3.

About PA3

PA3 asks you to implement a Binary Search Tree, which can store, sort, and search members of our social networking tool.

- Each Node stores a Member object
- Members compare lexicographically (alphabetic) on their **user ID** fields.
- Remember what a Binary Search Tree is:
  - A tree
  - Left sub-tree only contains nodes whose data is less than it's parent node
  - Right sub-tree only contains nodes whose data is greater that it's parent node

Let's take a look at the MemberNode class, from the skeleton files:



```
public class MemberNode{
    private Member item;
    private MemberNode leftchild;
    private MemberNode rightchild;
      ....
```

This tells us quite a bit about the design of the tree which contains MemberNodes. The node's data is in the form of a link to a Member object. Similarly, it's left and right children are links to other MemberNode objects.

Much like the last recitation, the design of this tree has most of the action in the 'tree' class, called MemberTree, while the nodes mostly just store information. Based on this and the skeleton, a simple traversal might look something like this (Keegan's random psudocode):

```
function traverse(MemberNode current):
    if(current == null):    // base case
       return;
    print current    // print the node
    traverse( current.getRightChild() )   // follow right link
    traverse( current.getLeftChild() )    // follow left link
```

This should visit every node, and print them all out. I am assuming that get functions return null if there is no child, which is a good way to handle that case. This could also be done with a loop and without recursion, but it is more difficult. In general, trees are (more or less) naturally recursive data structures. So recursion is generally recommended when working with trees (unless you have a reason not to use it of course). **This is not a BST search**, it is simply a dumb traversal of the whole tree.

Insert and Search

Inserting nodes into a BST is relatively straightforward:

- compare data to be inserted with the current node's data
  - if data to be inserted is less than the node, traverse left
  - if data to be inserted is greater than the node, traverse right
- repeat previous step until you try to traverse into a null child reference. e.g. the current node as a null reference in the direction you are trying to go.
  - make a new node object, and set it's data to the data to be inserted.
  - set the current node to be the new node's parent, by setting the current node's left or right child to the new node you just built.

Searching is very similar:

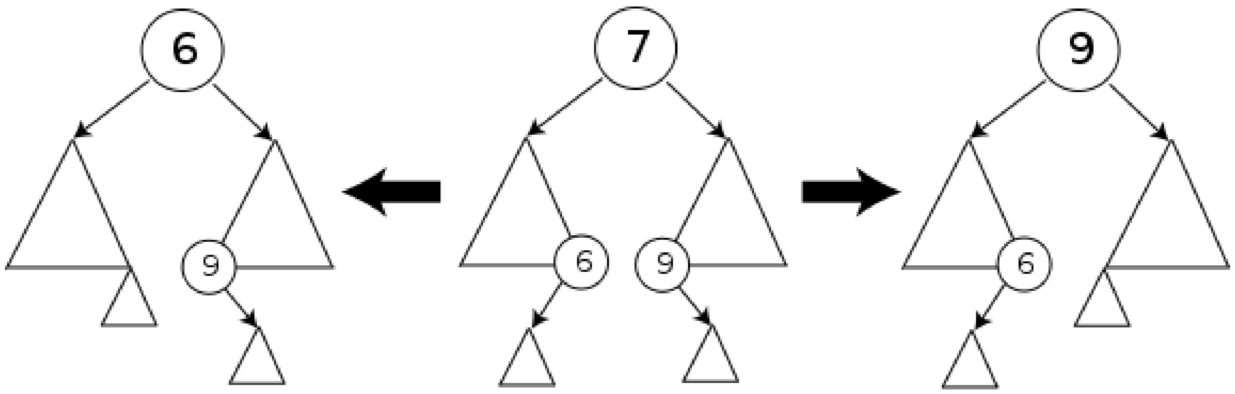
- Compare the search query to the current node's data
  - if the query is less than the node, traverse left
  - if the query is greater than the node, traverse right
  - if the query equals the node, search was successful. Return the node or whatever makes sense.
- repeat previous step until you either find the query or try to traverse into a null child.
  - trying to traverse into a null child node means the query was not found. Return failure to find query.

Deleting nodes

Is a bit more complex, the process differs depending on what type of node you are deleting:

- Leaf node (no children):
  - just delete the node
- Node with one child:
  - delete the node and replace it with it's child
- Node with two children:
  - can't delete this node (call it N), would break the tree.
  - find either it's in-order successor or in-order predecessor node (call this R)
  - swap the data stored in N and R
  - then delete R
    - R will have at most one child, so apply above deletion rules

Wikipedia has a good diagram for this:



Middle is the original tree, we want to delete the node 7. Left tree is with in-order predecessor, right one is with in-order successor. The triangles are sub-trees of arbitrary size.

For the purposes of our course, use the in-order successor node.

Tips for PA3

- Java strings implement the comparable interface. Their compareTo() method should suffice for this assignment.
- Note that tree nodes only keep a reference to their member object, no copying should be necessary, just set the node's data object to be an already existing member object.
- If you were to make the nodes implement the Comparable interface, and then specify their compareTo() method to compare the user ID of their data objects, you would have an easy way to compare nodes for the purpose of a BST search.
- Use of generics for the tree is encouraged, see last week's recitation (Recitation 7) for more on generic types. If you use generics, it would be good to specify that the generic type implement the comparable interface.
- In fact, the generic tree from last recitation could probably be used for PA3 with minor modification.

Exercise:

Work on PA3, ask questions.

Grading

Attendance only, come sign the sheet.