February 12, 2015 **Computer Science 282** Due: March 3/4, 2015

Programming Assignments #2

In addition to requiring you to get AVL trees to work, I also thought it would be useful if you learned some basic things about Java Strings. For example, you will need to use compareTo to compare two Strings. Also, I know some of you do not like having to work around my code. But again, I insist you do it so that you can demonstrate that you understand and can apply what we're studying in class.

Like last time, add whatever is necessary so that the two classes below behave correctly. Leave all variables, method names, and code fragments exactly as they appear. Only add code to make the methods work. Do not add a setVal() method.

Get started on insert immediately. It is not hard, but it is time consuming getting all the balancing code in place. Start by writing a simple test program so you can quickly see how your insert (and delete) is behaving. Delete is a little trickier and we will have to talk some more about it in class as we work on this program.

You will, of course, need a toString method to be able to test your program. Try to write a simple one. You might also try using Nir's. I have not tried it so I can't say much about it. I will also provide one in a day or so.

You should email me your Java classes in a single file named prog2.java. Do not email me your output, do not email me the test program, and be sure the file has exactly two classes in it. The subject of your email should be “Bobby Fischer – prog 2 – XX” if your name is Bobby Fischer and XX is either MW of TT according to what section you are in. The usual warning applies about following all instructions.

```
class StringAVLNode {

   private String val;
   private int balance;
   private StringAVLNode left, right;

   // I believe we have all agreed that one constructor should suffice
   public StringAVLNode(String str)

   public int getBalance()

   public void setBalance(int bal)

   public String getVal()

   public StringAVLNode getLeft()

   public void setLeft(StringAVLNode pt)

   public StringAVLNode getRight()

   public void setRight(StringAVLNode pt)

} // StringAVLNode


class StringAVLTree {

   private StringAVLNode root;

   // the one and only constructor
   public StringAVLTree()


   // this is here to make it easier for me to write a test
   // program - you would never do this in real life!
   public StringAVLNode getRoot()
   // Rotate the node to the right
   public StringAVLNode rotateRight(StringAVLNode t)

   // Rotate the node to the left
   public StringAVLNode rotateLeft(StringAVLNode t)

// For these next four, be sure not to use any global variables

   // Return the height of the tree - not to be used anywhere in insert or delete
   public int height()

   // Return the number of leaves in the tree
   public int leafCt()

   // Return the number of nodes with exactly one non-null child
   public int stickCt()

   // Return the inorder successor or null if there is none or num is not in the tree
   public StringAVLNode successor(String str) {

   public void insert(String str) {

   private StringAVLNode insert(String str, StringAVLNode t) {

      if (t == null)  // easiest case - inserted node goes here

      else if (t.getVal() == str)  // already in the tree - do nothing

      else if ... {  // str is smaller than this node - go left

         // get the old balance of the left child (where the insertion
         // is taking place)
         if (t.getLeft() == null)
            oldBalance =
         else
            oldBalance = t.getLeft().getBalance();
         t.setLeft(insert(str, t.getLeft()));
         newBalance = t.getLeft().getBalance();
         if ...  // did the height increase?
            // fix the balance value

            if (t.getBalance == -2)  // out of balance - must rotate
               if (t.getLeft().getBalance() == -1) //  single rotation
                                                   //  and balance update
               else  {                 //  double rotation
                                       //  and balance update
                    //  once you get it right here, basically the
                    //  same code can be used in other places,
                    //  including delete
               }
      }
      else { // str is bigger than this node

      }
      return t;
   }

   public void delete(String str) {

   private StringAVLNode delete(StringAVLNode t, String str) {

      if (t == null)
          // Do nothing if it is not in the tree
      else if (str.compareTo(t.getItem()) == -1) {

            // get the old balance.

            if (t.getLeft() == null)
               // still must deal with this special case in case
               // the element to be deleted is not in the tree
            else
            t.setLeft(delete(t.getLeft(), str));

            // get the new balance
            if (t.getLeft() == null)

            else

            if ...    // did the height decrease?

               // correct the balance

               if   // need to rotate?

                  // there are now actually 3 cases because t.getRight.getBalance()
                  // could be -1, 0, or 1.
                  if (t.getRight().getBalance() == 1) {

                  }
                  else if ...

                  else {  // double rotation case

      else if (d.compareTo(t.getItem()) == 1)

      else  // t is the node to be deleted

         if           // one of the easy cases

         else if ...  // the other easy case

         else {

         OldBalance =  // get the old balance

         t = replace(t, null, t.getLeft());   // find the replacement node and
                                              // move it up
         // get the new balance

         // just like before, see if height decrease and if so
         //  check to see if only need to change balance values or rotate


   // The code to find and replace the node being deleted must be recursive
   // so that we have easy access to the nodes that might have balance changes
   private StringAVLNode replace(StringAVLNode t, StringAVLNode prev,
                                          StringAVLNode replacement) {

      if (replacement.getRight() == null) {  // at the node that will replace
                                             // the deleted node
         if (prev != null) {     // if replacement node is NOT the child
                                 // ...  a special case
         }
         //  move the replacement node - Recall there is no setVal
      }
      else {

         // find the old balance

         t = replace(t, replacement, replacement.getLeft());

         // find the new balance

         // update balance and rotate if needed
   }

   // who are you? Put your name here!
    public static String myName() {
        return "Alfred E. Newman";
    }

}  // end of StringAVLTree class
```