ASSIGNMENT WEEK 4

In class we have seen implementations of several methods for a scapegoat tree data structure in Java. For this assignment, you will implement four methods as outlined below in **Java**.

0. Use the code from class / Canvas as a starting point or start yourself from scratch to have a class `ScapegoatTree` with generic key-value nodes.
1. Write `postOrder` and `preOrder` walk methods which take no input and return a **list** of tree nodes in the respective order.
2. Override the `toString` method of the Scapegoat tree class. Use the appropriate form of in-order, pre-order, or post-order walk to form a string of all current **keys** which recursively contains parent(children) pairs. Adapt your method from the methods you implemented in Exercise 1. As an example, if a *parent* has a *left* and a *right* child who in turn have *left* and *right grand*children, your return string would look like
   `parent.key(left.key(lleftgrand.key(…), lrightgrand.key(…)), right.key(…))`.
   If exactly one of the children is `null`, print a space in its place. If both children are `null`, the parent is actually a leaf and no recursion should be executed and no parentheses should be printed.
   For example, the tree in Figure 1 in the Galperin/Rivest 1991 paper should yield the string
   `2(1,6(5(4(3, ), ),15(12(9(7,11(10, )),13( ,14)),16( ,17( ,18)))))`.
   Hint: It is probably easiest to create the return string ***while*** *traversing (instead of after traversing) similar to collecting the nodes in the* `inOrder` *method shown in class. That means you would need a mutable string object. Since Java the* ***String*** *type is immutable, use one of the two mutable Java string types.*
3. Write a public method `successor` which finds the successor of a given key by finding the successor node and returning the key of that node. If a successor does not exist, return the original key. Recall that the successor of a node `n` is either the minimum node in the right subtree of `n` or the first parent of node `n` for which `n` is located in the left parent sub-tree.
4. Write a public method `predecessor` which finds the predecessor of a given key by finding the predecessor node and returning the key of that node. If a predecessor does not exist, return the original key. Recall that the predecessor of a node `n` is either the maximum node in the left subtree of `n` or the first parent of node `n` for which `n` is located in the right parent sub-tree.
5. Test and debug your methods. Provide test runs in form of a main (driver) file in which you create appropriate variables, fill the tree with data from the ***provided JSON file***, and run some cases to show your methods work. Do not forget to upload `Main.java` and `ScapegoatTree.java`!

Each of the problems 1 – 5 will be graded according to the following rubric for a total of 20 points.

SCORE	**4**	**3**	**2**	**1**	**0**
SKILL LEVEL	Response gives evidence of a complete understanding of the problem; is fully developed; is clearly communicated.	Response gives the evidence of a clear understanding of the problem but contains minor errors or is not fully communicated.	Response gives evidence of a reasonable approach but indicates gaps in conceptual understanding. Explanations are incomplete, vague, or muddled.	Response gives some evidence of problem understanding but contains major math or reasoning errors.	No response or response is completely incorrect or irrelevant.