CptS 223 PA #2

For this project, we will be implementing functions of an AVL tree. The tree will act as a BST with integer keys and must maintain the AVL nature as you insert and delete nodes. This archive includes a working AVL tree implementation, but does not do several things:

1) Maintain the heights of nodes after an insert
2) Check heights of subtrees and do rotations as appropriate to keep the tree balanced
3) Search the tree to see if it contains a given node
4) Delete a node when given a key

Your assignment is to add these features to the provided code. There are several notes in AVL.h about where you need to fill in the stubbed functions or replace/update the ones that are there, notably in insert, remove, contains.

That example code compiles and runs on the EECS SSH servers just fine. You can use that as a starting point for you own work. You will need to compile it using the g++ command line option -std=c++0x to let g++ know to use the newer libraries than the defaults:

```
 g++ -Wall -g -std=c++0x main.cpp
```

There is some code in main that will do many random inserts and ⅓ as many deletes. It's just test code, but once you start getting your rotates working the output should start looking much nicer!

Expected Output

Your code must compile on the EECS servers, though you can do your development on other systems and do final tests on the EECS machines if you like. I have included a sample series of inserts that force the tree to do all 4 kinds of rotate situations.

The printTree() function for my included test tree should end up outputting:

```
15
10 20
5 13 18 30
4 19 40
```

Deliverables

You must upload your program through Blackboard no later than midnight on Wednesday, October 26, 2016. The program will be uploaded as a zip file containing:

- Full C++ source code

Grading Criteria

Your assignment will be judged by the following criteria:

- [80] Code operational success. Your code compiles, executes, and generates the CSV files.
- [10] Your code is well documented and generally easy to read.
- [10] Your program intelligently uses classes when appropriate and generally conforms to good OOP design (i.e. everything isn't slapped into main).