San José State University
Department of Computer Engineering

CMPE 180-92

Data Structures and Algorithms in C++

Fall 2016
Instructor: Ron Mak

Assignment #13

100 points

Assigned:	Saturday, December 3
Due:	Friday, December 9 at 11:59 PM
URL:	http://codecheck.it/codecheck/files/1612030756dke3ifjc97pe040p7ft4tyimp
Canvas:	Assignment 13. BST and AVL trees
Points:	100

BST and AVL trees

This assignment will give you practice with binary search trees (BST) and balanced Adelson-Velskii and Landis (AVL) trees.

A tree printer

You are provided a **TreePrinter** class that has a **print()** method that will print any arbitrary binary tree. A template for how it prints a tree:

```
                                                              xx
                                                              /\
                                ------------------------------  ------------------------------
                               /                                                              \
                              xx                                                              xx
                              /\                                                              /\
                --------------  --------------                                  --------------  --------------
               /                              \                                /                              \
              xx                              xx                              xx                              xx
              /\                              /\                              /\                              /\
        ------  ------                  ------  ------                  ------  ------                  ------  ------
       /              \                /              \                /              \                /              \
      xx              xx              xx              xx              xx              xx              xx              xx
      /\              /\              /\              /\              /\              /\              /\              /\
    --  --          --  --          --  --          --  --          --  --          --  --          --  --          --  --
   /      \        /      \        /      \        /      \        /      \        /      \        /      \        /      \
  xx      xx      xx      xx      xx      xx      xx      xx      xx      xx      xx      xx      xx      xx      xx      xx
  /\      /\      /\      /\      /\      /\      /\      /\      /\      /\      /\      /\      /\      /\      /\      /\
 /  \    /  \    /  \    /  \    /  \    /  \    /  \    /  \    /  \    /  \    /  \    /  \    /  \    /  \    /  \    /  \
xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx  xx
```

TreePrinter is able to print trees with height up to 5, i.e., 32 node values on the bottom row. An example of an actual printed tree:

```
                            24
                            /\
              --------------  --------------
             /                              \
            18                              73
            /\                              /\
      ------  ------                  ------  ------
     /              \                /              \
    12              19              38              87
    /                               /\               \
  --                              --  --              --
 /                               /      \               \
10                              37      41              90
                                         \
                                          \
                                          64
```

Part 1

The first part of the assignment makes sure that you can successfully insert nodes into, and delete nodes from, a BST and an AVL tree. **Do this part in CodeCheck.**

First, create BST node by node. You will be provided the sequence of integer values
to insert into the tree. Print the tree after each insertion. The tree will be unbalanced.

Now repeatedly delete the root of the tree. Print the tree after each deletion to verify that
you did the deletion correctly. Stop when the tree becomes empty.

Second, create an AVL tree node by node by inserting the same sequence of integer values. Print the tree after each insertion to verify that you are keeping it balanced. Each time you do a rebalancing, print a message indicating which rotation operation and which node. For example:

```
Inserted node 10:
    --- Single right rotation at node 21
```

As you did with the BST, repeatedly delete the root of your AVL tree. Print the tree after each deletion to verify that you are keeping it balanced.

```
Inserted node 62:

  62

Inserted node 71:

    62
     \
      \
      71

Inserted node 29:

    62
    /\
   /  \
  29  71

Inserted node 88:

        62
        /\
      --  --
     /      \
    29      71
             \
              \
              88
```

A handy AVL tree balance checker:

```
template <class Comparable>
int AvlTree<Comparable>::checkBalance(BinaryNode<Comparable> *ptr)
{
    if (ptr == nullptr) return -1;
    int leftHeight  = checkBalance(ptr->left);
    int rightHeight = checkBalance(ptr->right);
    if ((abs(height(ptr->left) - height(ptr->right)) > 1)
        || (height(ptr->left)  != leftHeight)
        || (height(ptr->right) != rightHeight))
    {
        return -2;       // unbalanced
    }
    return height(ptr);  // balanced
}
```

Expected output for Part 1

See http://www.cs.sjsu.edu/~mak/CMPE180-92/assignments/13/Assignment13-output.txt

Part 2

The second part of the assignment compares the performance of a BST vs. an AVL tree. Part 2 should be a separate program from Part 1. Adjust all counts accordingly for slower machines. **Do this part outside of CodeCheck.**

First, generate n random integers. n is some large number, explained below. Time and print how long it takes to insert the random integers one at a time into an initially empty BST. Do not print the tree after each insertion.

Time and print how long it takes to insert the same random integers one at a time into an initially empty AVL tree. Again, do not print the tree or any rotation messages after each insertion.

Choose values of n large enough to give you consistent timings that you can compare. Try values of n = 10,000 to 100,000 in increments of 10,000.

If $T(n)$ is the time function, graph the growths of $T_{\text{BST}}(n)$ and $T_{\text{AVL}}(n)$.

Second, generate k random integers. k is some large value. Time and print how long it takes to search your 100,000-node BST for all k random integers. It doesn't matter whether or not the search succeeds. Try values of k = 10,000 to 100,000 in increments of 10,000.

Time and print how long it takes to search your n-node AVL tree for the same k random integers.

If $T(n)$ is the time function, graph the growths of $T_{\text{BST}}(n)$ and $T_{\text{AVL}}(n)$.

An alternative (especially for slower machines): Instead of timing the tree insertions and searches, you can count probes and compares, similarly to the way you counted the moves and compares of the sorting algorithms. A probe is whenever you reach a tree node via a pointer, even if you don't do anything with the node other than use its left or right link to go to another node. A compare is a probe where you also do a test of the node's value. Be sure to count probes and compares during AVL tree rotations.

So instead of getting timings, you can count probes and compares, and then graph the growth curves of those counts.

Code

You can use any code from the lectures or from the textbook or from the Web. Be sure to give proper citations (names of books, URLs, etc.) if you use code that you didn't write yourself. Put the citations in your program comments.

What to submit

Submit into Canvas: Assignment #13:

- The signed zip file from CodeCheck. (Only do Part 1 in Canvas.)
- A text copy of the output from Part 1. (CodeCheck truncates the output in its report.)
- A text copy of the timings (or probe and compare counts) and their graphs from Part 2.

Rubrics

Criteria	Maximum points
Part 1	**40**
• The AVL tree remains balanced after each node insertion.	• 20
• The AVL tree remains balanced after each node deletion.	• 20
Part 2	**60**
• BST insertion timings (or probe and compare counts)	• 10
• AVL insertion timings (or probe and compare counts)	• 10
• Graph of insertion timings (or probe and compare counts)	• 10
• BST search timings (or probe and compare counts)	• 10
• AVL search timings (or probe and compare counts)	• 10
• Graph of search timings (or probe and compare counts)	• 10