CPSC 223: Assignment #8

Due Monday, April 30th

The goal of this assignment is to create an AVLBinSearchTree class. You are required to implement the AVL insertion algorithm (due Monday, April 30th); and you will get bonus to implement the AVL removal algorithm. You need **Hand in** a hard-copy of your code and a design description document; and **submit** your code to the class account on ada (where your code should compile and run).

STEP 1: **AVLBinSearchTree class**, which could be a subclass of BinarySearchTree class. You are not required to do so. The main change (besides insert and remove, see below) is to declare an AVLNode class. You can also modify your Node class in BinarySearchTree if you want to reuse Assignment 7. Compared to Node class in Assignment 7, you should at least add a 'height' field in ALVNode class. However, it is up to you to introduce what kind of data field in each class. You will also need to provide a constructor, destructor, copy constructor, and assignment operator, for AVLBinSearchTree. AVLBinSearchTree must support all methods named in BinarySearchTree(except "Remove"). Additional methods will be added to this class in the following steps.

Each group should provide a design description of your project. For example, how you track the height of nodes and how to maintain the balance. Explain the purpose of every added field in the AVLNode class. Explain your design of the following methods.

STEP 2: **Implement Insert in** AVLBinSearchTree **class**. To implement insertion method in AVLBinSearchTree, you should define a (protected) helper function **insertItem** in AVLBinSearchTree.

- void insert (const ItemType& newItem)

Your insert method in AVLBinSearchTree should simply call the insert helper as below.

```
void AVLBinSearchTree::insert(const ItemType& newItem)
{
  insertItem (mroot, newItem);
}
```

You are free to add additional helper methods to implement the insertion algorithm, but your helper methods should be declared as *protected/private* members of your class.

STEP 3(optional): **Implement remove in** AVLBinSearchTree **class**. The implementation of Remove method in AVLBinSearchTree class follows the same instruction as insertion.

As in STEP 2, you are free to add additional helper methods to implement the removal algorithm, but your helper methods should be declared as *protected/private* members of your class.

STEP 4: **Test** your AVLBinSearchTree. Your tests should demonstrate that your insert (and remove) method(s) work(s) properly. In addition, implement a special print method in your AVLBinSearchTree for testing:

- void printTest()

which should perform a preorder traversal of your AVL tree that prints: (i) each AVLNode item; and (ii) the height of the AVLNode. For example, calling tree.printTest() should output something like this (for a tree containing Key 15 as root and Key 13 as the only leaf node):

```
Key: 15,        height: 2
Key: 13,        height:1
```

Your program should compile using g++;

STEP 5: **Place your files in a hw8 directory, and submit to ADA.** Also, be sure to turn in your design document, hard-copy of your code, and any additional results from running your program.