CMPT 225 D2
Fall 2020
T.Shermer

Assignment 6
AVL Trees

Due Dec 4 at 23:59

You are to write an AVL Tree class, any associated exception classes, and do some tests of it. As part of the code, you will need to be able to print a binary tree.

We'll simply be storing integers in the AVL tree, so there is no need for an Entry class.

The principal class will be called **AVLTree,** and it should have a nested **Node** class. AVLTree should have operations:

type	name	
Node	find(k)	// if the tree has a Node with key k, return it; // else, return a reference to a special Node **end.**
Node	put(k)	// adds k to the AVLTree (duplicates are allowed), and returns its Node.
void	erase(k)	// erase an entry with the key k.
int	size()	// number of elements in the AVL Tree
bool	empty()	// is size == 0
Node	getEnd()	// returns the special Node **end**

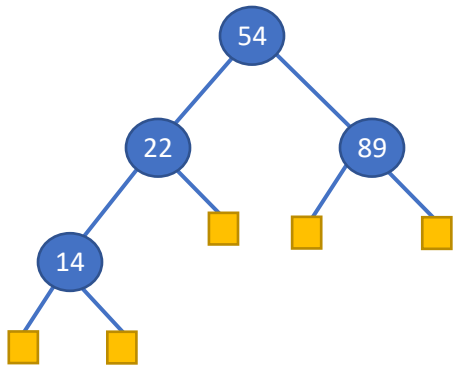
It should also have a constructor and a destructor. The **Node** class should have left and right children, a key, and a height. The height is used in rebalancing. The height of an external node is 0, and the height of any other node is the maximum of its children's heights plus one. When you restructure the tree, you will have to change the height information in nodes accordingly. Each put and erase can cause rebalancing to occur, as shown in lecture.

Note that the functions are given approximately. Add or remove references (&) or pointers (*) where necessary or sensible, and add **const** to arguments and functions if they are const. I also haven't necessarily defined all the data members that the classes should have.

Your code should also include a "print" function for AVL Trees. It can be an overloaded operator<< if you like, or a simple function. It should print each node of the tree, indented, on a separate line using a preorder traversal. (See the "Trees" lecture, slide 18 for an example.) Use a 3 or 4 space indent. To print the node of the tree, print the key at that node, followed by the height of the node in parentheses. If the node is external, print "L" for the key.

For example:

```
54 (3)
   22 (2)
      14 (1)
         L (0)
         L (0)
      L (0)
   89 (1)
     L (0)
     L (0)
```



In your main file, insert 7 entries into an AVL Tree, then print the tree. Delete an entry from the tree, and print it again. Do one find of an element in the tree, printing the result, and one find of an element not in the tree, printing the result.

Finally, generate 15 entries and insert them all into an empty AVL Tree. Print the tree. Delete five of them and print the tree again.

You will be judged on correctness of your code and on code style, so don't forget to keep your code clean as you develop it! (Or at the very least, clean it up before submission. We don't want to see untidy code.)