CS20 – Section 093 – Fall 2019

Programming Assignment 9 – Due Wednesday December 4 at 11:59PM

Binary search trees (60 points)

You're given a lot of the code for processing a binary search tree:

tree node struct
BST class definition
search function
delete function (named udelete), and internal delete function
print2D function, and internal recursive print2D function

Part 1.

This will test the code you were given and one new function you're adding. Code an insert function to insert nodes into the tree. The function has the prototype

void insert (value)

where value is an integer to insert into the BST. If the node is a duplicate, display an error message and terminate the program.

Either code stubs for the missing functions, or comment out their prototypes in the BST class definition so that the code with only the insert function added will compile and run. Test your code using the supplied main for part 1. Note that this code uses the print2D function to display the tree at various points in the build process.

Part 2.

Add four functions that support the implementation of balance. You will code:

setallBF
setallBFinternal
getLargestBF
getLargestBFInternal

and you're given the height function, and the internal recursive height function.

setallBF is a user function which is called to set all the balance factors in all nodes. It has this prototype:

```
void  setallBF();
```

setallBF calls the internal recursive function setallBFInternal, which traverses the entire tree, setting the balance factor at each node. It has this prototype:

```
void setallBFInternal (node *)
```

setallBF is called for a particular node, and is passed a pointer to the node as the parameter. It calculates the height of the left subtree, then the height of the right subtree, and stores the difference as the node's balance factor. It then calls itself to continue the process for the left subtree, and then the right subtree. It's entirely possible that when setallBFInternal is called, the tree is empty, and the function must be able to handle this case.

getLargestBF is a user function that returns the largest balance factor in the tree. It has this prototype:
egative

```
int getLargestBF ()
```

getLargestBF calls the internal recursive function getLargestBFInternal. This function has the prototype:

```
int getLargestBFInternal (node *)
```

getLargestBFInternal traverses the tree starting at the node passed as a parameter, and returns the largest balance factor found. It calls itself to get the largest balance factor in the node's left subtree, then calls itself again to get the largest from the right subtree. As you traverse the tree, you may encounter balance factors that are positive or negative, and your code must correctly handle this. For example, -3 is smaller than 2 in an algebraic sense, although -3 is "larger" in a tree-balancing sense. getLargestBF passes the root of the tree to getLargestBFInternal to get things started.

Change the node definition to include an integer balance factor in each node.

Change print2D to display the balance factor for each node, along with the node's value, so that the output looks something like this:

```
                55 (0)
        50 (1)
45 (-2)
                42 (0)
        40 (1)
```

Test your code using the supplied main for part 2. For grading, submit part 2 along with the testing main.