Binary Search Tree (Pt 2):

75 pts, Due Fri, Apr 7 Midnight

For this lab you will be writing methods for a binary search tree, and testing with the process you created in part 1.

A binary search tree, when balanced, allows one to find, insert, and remove data all within O(log n) time. That's not bad!!

In this case we're using a binary search tree to keep track of endangered species. A binary search tree is ideal for keeping track of endangered species because the set of endangered species is constantly changing – animals are, unfortunately, constantly being added to the list. Luckily some are also removed from the list as we protect their habitats or change the factors that were threatening them. Equally we often want to find the species to get up-to-date information about the species, especially concerning their current level of threat.

For this lab, you will be writing the following methods associated with a Binary Search Tree whose data type is Species. The Species class has three fields:

- string name; // for the name of the species (aka animal)
- string status; // for the status of the species (e.g., T1 critically threatened, VU vulnerable, etc.)
- string info; //for information about the animal

Each tree node has the following fields:

- `BSTNode *left;`
- `BSTNode *right;`
- `BSTNode *parent;`
- `Species *animal;`
- `int height;`

The tree will be ordered by the animal field's name field (aka animal->name)

You will be writing the following methods for the binary search tree:

```
/*********You must write ***********************/
BST(string s, string st, string info, bool Xtra);
//4 pts
//Constructor, takes as input 3 strings and a Boolean.  The three
//strings are the species, the status, and the info, and are used to
//create the root node's animal;  The Boolean is true for if you want
//the print methods to print out everything about each animal, or false
//if you just want to see each animal's name and height (easier to test
//with)

bool insert(string n, string s, string info);
// 9 pts
//this is to insert a new node into the tree.  It takes as input 3
```

```
//strings, the name, the status, and the info, which are used to create
//the new node’s animal field.

BSTNode *find(string s);
// 7 pts
//this finds the node who’s animal’s name matches the input string s,
//and returns the node.  If the node is not in the tree, find returns
//NULL

void printTreeIO(BSTNode *n);
// 3 pts
//Prints the tree in order

void printTreePre(BSTNode *n);
// 3 pts
// Prints the tree using a pre-order traversal

void printTreePost(BSTNode *n);
// 3 pts
//Prints the tree using post-order travesal

void updateStatus(string name, string st);
// 4 pts
//This finds a node with the animal's name matching the
//input name parameter and changes its current status to teh
//status st that is being passed in as a parameter

BSTNode *removeNoKids(BSTNode *tmp);
// 5 pts
// the method that removes a node with no children

BSTNode *removeOneKid(BSTNode *tmp, bool leftFlag);
// 7 pts
// the method that removes a node with one child

BSTNode *remove(string s);
// 9 pts
// this method is the general remove method.  If the node to
//be removed has no children, it calls removeNoKids.  If the
//node to be removed has one child, it calls removeOneKid.
//If the node to be removed has 2 children, it finds the
//correct replacement node (left-most of right child) and
//replaces the data to be removed with the data in the left-
//most of the right child.  It then calls either
//removeNoKids or removeOneKid with that left-most of right
//child node

void setHeight(BSTNode *n);
// 9 pts
// this method updates the heights of a node's ancestors
// when a node is added to a tree.  Whenver a node is added
// to a tree, it is possible that every one of its direct
// ancestors' heights might possibly increase by 1, so you
// have to check and possibly update the ancestors' heights

/****************written for you***************/
BST(bool Xtra);
void printTreeIO();
void printTreePre();
void printTreePost();
void clearTree();
void clearTree(BSTNode *tmp);
/************************************************/
```

(12 pts) You must also get the methods working with your test cases (either added to the main method or to the constructor – however you choose). Once you have the test cases working, you can test your data with my larger data set. You can see that your output matches my output run with the interface.