Lab #12

CS-2050 - Section D

Week of May 3, 2021

1 Requirements

In this lab, you will write a set of functions for maintaining **a binary search tree**. The structure of the tree is *not clearly defined*, and you must design the tree structure to meet the performance constraints outlined in the functions below. Your implementation of the required functions should match the performance requirement *exactly*.

```
> lab.h
typedef struct BST BST;
    ...
typedef struct {
        float squareFeet;
        int baths;
        int houseNumber;
} House;

> lab.c
struct BST {
    // What goes here is up to you
};
```

Info: In this lab, you will be graded for your implementation of the Tree structure. You may choose to include as many or as few struct members as you like, and you may define additional struct types to complete this assignment. The names of struct types and members should be **clear and accurate** as to their purpose, and you should **leave comments** describing your implementation. In case it is not clear, you are ***intended*** to use helper functions to achieve recursion.

1.1 Support Functions

```
// Complexity: O(1)
BST* initBST();
// Complexity: O(n)
size_t BSTSize(BST *tree);
// Complexity: O(n)
void freeBST(BST *tree);
```

Info: These functions are required for grading purposes, but are not part of the testing for this lab. You are expected to implement these functions to support your implementation, but they are being counted as a single "function group" and will not be a significant part of your grade for this lab.

1.2 insertHouse

```
// Complexity: O(log(n))
int insertHouse(BST *tree, House *house);
```

Info: This function **recursively** inserts the given struct pointer onto the tree using the same compare function as with before. It should return 1 on success or 0 on failure.

1.3 searchHouses

```
// Complexity: O(log(n))
int searchHouses(BST *tree, House *query);
```

Info: This function performs a *recursive binary search* on the given tree. This function will return 1 if the struct is found or 0 if not.

1.4 postOrderPrintHouses

```
// Complexity: O(n)
void postOrderPrintHouses(BST *tree);
```

Info: This function prints the tree **post order** using *recursion*.

2 Notice

Grading: Total 25 points

1. Write required *support* functions
   * 7 points
2. Write required *insert* function
   * 6 points
3. Write required *search* function
   * 3 points
4. Write required *print* function
   * 3 points
5. Tree structure is properly formatted and supports performance requirements
   * 6 points

Notice:

1. All of your lab submissions must compile under GCC using the $-Wall$ and $-Werror$ flags to be considered for a grade.
2. You are expected to provide proper documentation in every lab submission, in the form of code comments. For an example of proper lab documentation and a clear description of our expectations, see the lab policy document.