In-lab assignment for week of May 4th

Objectives:

Binary search tree

Data structure:

We'll assume the following structs in this lab:

```
typedef struct{
    int ID;
    float salary;
    int ssn;
}Employee;
typedef struct node{
    Employee* data;
    struct node *left, *right;
}Node;
```

In this lab, programmers need to read data from a file named "employee.csv". The first line in this file indicates how many records it contains below. Each of the remaining lines contains information for one employee in this order: ID, salary, ssn.

Requirements:

In this lab, 5 functions need to be implemented using the following function prototypes:

*1. Employee** readRecord(FILE*);*

This function receives a FILE pointer pointing to the provided csv file. It creates an array of "Employee" struct pointers, and populates the array with the provided information. It returns the array or NULL back to the calling program.

2. int comparison(void,void*);*

This function receives two Employee struct pointers, and compares the "ssn" values in them. It returns 1 if the first ssn is larger, -1 if the second ssn is larger, or 0 if the two values are equal.

3. Node insertBST(Node*, Employee*);*

This function receives the current BST root and an Employee struct pointer. It creates a new Node with the given Employee information and recursively inserts the new node into the

current BST based on its ssn value in descending order. It returns the updated BST back to the calling function.

Note: a) you should use the "comparison" function for comparison. b) the root pointer has been created and initialized to "NULL" before it's passed into this function for the first time.

4. Employee searchBST(Node*,int);*

This function receives the current BST root and an integer number. It recursively searches any Employee struct whose "ssn" member is equal to the given integer number. This function returns the pointer to the struct if such struct exists, or NULL if no such struct can be found.

5. void deleteTree(Node);*

This function receives the current BST root, then frees the memory for the whole BST.

Note: The provided .h header file and csv file should NOT be modified or submitted. If you have any customized functions, you need to write them in your .c header file. Your customized functions should **NOT** be called by other programs, e.g. the main program.

Your .c header file is the ONLY file you need to submit through Canvas. An example of header files can be found as lab 7 solution on Canvas.

Grading Criteria:

readRecord function: 2 points

comparison function: 3 points

insertBST function: 5 points

searchBST function: 5 points

deleteTree function: 3 points

General notes:

1. Command to compile your code in cmd/powershell window: gcc main.c labx.c -Wall -Werror
2. If your code couldn't compile with "-Wall -Werror" flags, you will receive an automatic 0 grade.
3. Changing the given .h header file will lead to an automatic 0 grade.
4. Using any global variables will lead to an automatic 0 grade.
5. Function implementation should include comments describing what it is intended to do and how this function should be called. Examples can be found in lab solutions. Failing to do so will result in a point deduction.