Binary Search Tree
Cpt S 223 Homework Assignment
Washington State University

In this assignment, you will implement a template class for a binary search tree. This assignment is slightly different from previous assignments because you are given the main.cpp file. This file auto-executes a series of tests on your tree. Do not modify this file.

The grading process for this assignment will involve a similar main.cpp file, only with a different set of tests cases than in the one included in the assignment zip file. This is to ensure you've written correct code that satisfies all requirements listed and not just one specific set of test cases.

Implement the template BST class in Trees223.h

The main.cpp file included in the assignment zip has an include statement for the Trees223.h header file. This is where you will implement the class for the binary search tree. This file must exist or else the code in main.cpp will not compile.

Declare the BST class as a template class with template type T, then implement the following public member functions:

```
BST();

virtual ~BST();

virtual bool Add(
   const T& value,
   std::function<int(const T& first, const T& second)> compare);

int Count() const;

int CountLevels() const;

int GetLevel(std::function<int(const T& existing)> compare) const;

void InOrder(std::function<void(const T& value)> visitCallback) const;

virtual bool Remove(std::function<int(const T& existing)> compare);
```

The constructor and destructor should be self-explanatory. The remaining functions are described below. Before beginning the per-function descriptions, it's worth noting that several functions take an std::function object as a parameter. This is a comparison function that allows whatever type is being used as the template type to be compared in a way that is defined OUTSIDE of the tree class. For example, it may be the case that you want a BST of strings and you want to do standard string comparisons, so that "ABC" would be less than "abc". You may also want it to be the case that the BST does case-insensitive comparisons, so that "ABC" == "abc". With comparison functions passed in as parameters, your tree will be capable of doing either.

Function	**Description**
Add	Adds the value to the tree. Duplicate values are NOT allowed in this BST implementation, so the function must return false if the value is already in the tree. Otherwise the value must be added and true returned. The compare function passed in as the 2nd parameter will return: 0 if first == second > 0 if first > second < 0 if first < second
Count	Returns the number of nodes/values in the tree.
CountLevels	Returns the number of levels in the tree. A tree with no nodes (empty tree) has 0 levels. A tree with only the root node has 1 level. A tree with a root node and only one child of the root has 2 levels, and so on.
GetLevel	Searches for a value in the tree and returns the 0-based index of the level where the value is found. Returns -1 if the value is not found in the tree, 0 if the value is found in the root node, and so on. The value being searched for is not actually passed in as a parameter. It does not need to be. Instead a comparison function is passed in. This function is set up to compare an existing value in the tree against the value being searched for. When called, this function returns: 0 if value being searched for is equal to 'existing' param > 0 if value being searched for is greater than 'existing' param < 0 if value being searched for is less than 'existing' param
InOrder	Does an in-order traversal of the tree, invoking the callback function for each value.
Remove	Removes a value from the tree. Returns true if the value is found in the tree and removed, false otherwise. You MUST use the following removal scheme, which will have been discussed in class: • If a leaf node is being removed, just remove it. • If an internal node with only one child is being removed, then it is effectively replaced by its only child. • If an internal node with two children is being removed, then the minimum value in the right subtree is stored in a temporary variable, recursively removed, then used to replace the actual value being removed. The value being removed for is not actually passed in as a parameter. It does not need to be. Instead a comparison function is passed that is set up to compare an existing value in the tree against the value being removed. When called, this function returns: 0 if value being removed is equal to 'existing' param > 0 if value being removed is greater than 'existing' param < 0 if value being removed is less than 'existing' param

The tests that code in main.cpp executes on your tree will provide further guidance for requirements. You may add additional member functions/variables to the BST class as needed.

Final Notes:

- Like all assignments, if your code does not compile you will be given a 0. This means you must add all the member functions, even if you don't get around to completing the implementation of each one.
- Your code must not crash for any of the included test cases. If your code is crashing, don't turn it in. Fix the crash, even if this means removing certain functionality that is unreliable.
- Do not display anything to the terminal window. This means your tree code must never use std::cout, printf, etc.
- Put comments in your code! Give a brief explanation for each function at a minimum.