CS 2420 Section 2 and Section 3
Spring 2016
Assignment 3: Binary Search Tree
Due: 11:59 p.m. February 19 (Friday), 2016
Total Points: 100 points

Part I [25 points]:
Write down your solution on a piece of paper. You may either turn in your corrected paper right after the class on Feb. 9 to get full credit of 25 points or submit it via Canvas if you did not attend the class on Feb. 9.

1) **[3 points]** Draw a binary search tree after inserting the following values: 20, 5, 15, 30, 40, 25, 10, 12, 8, 28.
2) **[6 points]** Give the pre-order, in-order, and post-order traversal lists of the above binary search tree, respectively.
3) **[1 point]** What is the height of the node containing the value of "30"?
4) **[1 point]** What is the depth of the node containing the value of "30"?
5) **[3 points]** How many comparisons will be performed to decide whether the value of "12" is in the binary search tree generated in Problem 1)? Please list each of the comparisons used to support this decision.
6) **[3 points]** Diagram the changes to the binary search tree generated in Problem 1) after removing 20.
7) **[2 points]** Diagram the changes to the binary search tree in Problem 6) after inserting 20.
8) **[3 points]** What is the final binary search tree after removing 5 from the binary search tree in Problem 7)?
9) **[1 point]** If the number of leaves in a perfect binary tree is 512, what is the height of the tree?
10) **[2 points]** What is the total number of nodes in the perfect binary tree in Problem 9?

Part II [75 points]:

You are required to develop a spell checker using a binary search tree, which makes searching each word in a document to be very fast. You are supplied with a dictionary of about 14,000 words, which are saved in a file "**dictionary.txt**" in alphabetic order.

You must create a binary search tree, which contains the following public methods:

- **[1 point] Constructor**
- **[7 points] Copy constructor**
- **[7 points] Destructor**
- **[7 points] Size**: Return a count of the number of nodes in the tree.
- **[7 points] Height**: Return the height of the tree, which is defined as the length of the path from the root to the deepest node in the tree (i.e., the maximum distance from all leaf nodes to the root).
- **[3 points] Delete**: Delete a word from the binary search tree if a node in the binary search tree contains the desired word. Otherwise, no delete operation is performed. **This delete operation should not increase the height of the tree.** Please refer to slide 19 of

Ch3.BinarySearchTree.pdf for a delete strategy, which does not increase the height of the tree after performing the delete operation. Please copy the code shown in slides 24, 25, and 26 of Ch3.BinarySearchTree.pdf in your implementation to understand how the code works. Revise the section of the code related to "delete a node with two children", which is copied as follows:

```
string successor;
DeleteSuccessor(ptr, successor);
ptr->name = successor;
```

by calling RDelete function. In other words, you will remove DeleteSuccessor function and its implementation and keep RDelete and DelNode functions as private functions which will be called to get the task accomplished.

- **[7 points] Traverse**: Traverse the binary search tree using the **pre-order traversal** and return a string containing all the words, which are stored in the binary search tree, in the descending order. Note: Spaces are used to separate the two words. For example, if "test", "good", "score" are inserted into a binary search tree, the returned string after calling Traverse function is: test score good.
- **[8 points] Insert**: Insert a string into the binary search tree.
- **[8 points] Find**: Return true if the string is found in the tree. Return false otherwise.

You must implement copy constructor, destructor, Size, Height, Delete, and Traverse functions using recursive solutions and the Insert and Find functions using non-recursive solutions.

Write a driver program to do the following tasks:

1. **[5 points]** Read the file "**dictionary.txt**", where each word is on a separate line, and store these words into a binary search tree. **After reading the dictionary into the tree, have your program report the size of the tree (i.e., the number of nodes) and the height of the tree.** Since the dictionary is in the alphabetic order, you cannot sequentially read in each word and insert these words in the tree (i.e., it will essentially be a linked list if doing so). **In other words, you must insert words in a random order to make a tree that can be efficiently searched.**
   Hint: One suggestion is to read the dictionary words into a `vector<string>` and then use the member function `random_shuffle` of the vector library to accomplish the goal. **Refer to Section 16.5 (Introduction to the standard template library) for detailed explanation on the member functions (pages 1006 through 1009 of the seventh edition or pages 1000 through 1013 of the eighth edition of CS1410 textbook).**
2. **[10 points]** Demonstrate the correctness of all the public member functions by **using the first 10 randomized strings** generated from `random_shuffle` function. In other words, use the first 10 randomized strings to create a binary search tree to test all the public member functions.
3. **[5 points]** Read the file "**letter.txt**" which contains some words (with mixed small and capitalized letters) used in a letter without any punctuations. Output to the console all the misspelled words (i.e., any word not found in the dictionary). Keep in mind the grader will use a different file when doing the grading.
   Hint: One suggestion is to read each word in "letter.txt" and convert the word into all small letters before searching the word in the binary search tree. **Refer to pages 809 and 810 of the seventh edition or pages 818 through 820 of the eighth edition of CS1410 textbook for detailed explanation on the string class member functions.**

Below is the BSTree.h file, which contains C++ BSTree class interface, for your reference.

```
#ifndef BSTREE_H
#define BSTREE_H

#include <string>
#include <iostream>
using namespace std ;

class TreeNode
{
public:
        string word ;
        TreeNode *ptrLeft, *ptrRight ;
        TreeNode() ;   // Default constructor.
        TreeNode(string s) ;  // Constructor with one parameter
} ;

typedef TreeNode* TreeNodeptr ;

class BSTree
{
private:
        TreeNodeptr root ;

public:
        BSTree() ;  // Initializes root to NULL.
        BSTree(const BSTree &);  //Copy constructor
        ~BSTree() ;  // Destructor.
        int Size ();  // Return the number of nodes in the tree.
        int Height();    // Return the path length from the root node to a deepest leaf node in the tree.
        void Delete(string s) ; // Delete a string.
        string Traverse() ;  // return a string containing all strings stored in the binary search tree in
                             // the descending order.

        void Insert (string s) ;  // Insert a string into the binary search tree.
        bool Find (string s) ;   // search s in the list.  Return true if s is found; otherwise, return false.


private:
        // You need to add private functions to implement the recursive solutions.
} ;
#endif
```