Programming Assignment #4
CS 163 Data Structures

Submit your assignment to the D2L Dropbox (sign on via d2l.pdx.edu)

Programming – Goal: The goal of this program is to create a binary search tree (BST) and to implement **ALL** of the BST algorithms **recursively.**

Background: The advantage of a binary search tree is the ability to retrieve our data using a logarithmic performance assuming that the tree is relatively balanced and be able to search for a range of information and obtain our data in sorted order. In program #4 we will experience all of these characteristics. Although we will be implementing a "Table ADT" (which means we are working with the "value" of the data), we are not implementing a hash table. Instead we are implementing a BST as our data structure.

Specifics: There are many terms used in Computer Science that are unique to this industry and have specialized meanings. Many of these words are hard to understand for other majors and are even harder to translate into other languages! For Program #4, your job is to create a BST to build a tree of CS concept or term that you have read from the Carrano reading. Focus first on the two tree chapters and then on hash tables and finally on graphs. You are welcome to use an external data file for the majority of your data, but you are not required to.

The data you are working with should have the following fields:

1. Name of the CS concept or term (e.g., Collision Resolution)
2. Description of the concept
3. Which data structure this relates to (e.g., Hash Tables)

The **BST's ADT operations that must be performed on this data are:**

1) **Constructor** – initialize all data members and load the data from the external data file
2) **Destructor** – deallocate (release) all dynamic memory and reset the data members to their zero equivalent value, use post order traversal
3) **Insert** a new CS concept or term and its definition from the client program; the name should be the key used to organize your BST. Data that is "less" goes to the left. Data that is "greater than or equal" will go to the right.
4) **Remove** a CS concept or term, based on the name (e.g., "Tail recursion")
5) **Retrieve** the definition for a particular CS concept or term by their name
    a. Make sure the keyword being searched for is properly capitalized
    b. Remember, retrieve is NOT a display function and should supply the matching information back to the calling routine through the

argument list

6) **Get_Height** return the height of the tree (refer to Lab #7)
7) **Is_Efficient** using the mathematics from Topic #9 to determine how close to balanced the tree is and return true if it is a reasonably balanced tree.
8) **Display all** (in sorted order by name)
9) **Display range** - display all concepts where their names are within a range (staring first letter of the keyword to ending first letter of the keyword (e.g. 'K' through 'M'). This function will need two arguments.

Data Structures: Write a C++ program that implements **a binary search tree**. The binary search tree should be a non-linear implementation (using left and right pointers). The underlying data may be stored as a struct or class.

Preparing for the Efficiency Writeup: Evaluate the performance of storing and retrieving items from this tree. Monitor the height of the tree and determine how that relates to the number of items. If the number of items is 100 and the height is 90, we know that we do not have a relatively balanced tree!! Remember the mathematics we learned in Topic #9. Also, use the information from the Carrano reading to assist in determine if we have a reasonable tree, or not. **Your efficiency write up must discuss what you have discovered.**

<u>Things you should know...as part of your program:</u>

1) Do not use statically allocated arrays in your classes or structures. All memory must be dynamically allocated and kept to a minimum!
2) All data members in a class must be private
3) Never perform input operations from your class in CS163
4) Global variables are not allowed in CS163
5) Do not use the String class! (use arrays of characters instead and the cstring library!)
6) Use modular design, separating the .h files from the .cpp files. Remember, .h files should contain the class header and any necessary prototypes. The .cpp files should contain function definitions. You must have at least 1 .h file and 2 .cpp files. **Never "#include" .cpp files!**
7) Use the iostream library for all I/O; do not use stdio.h.
8) Make sure to define a constructor and destructor for your class. Your destructor <u>must</u> deallocate all dynamically allocated memory.