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: Take program #3's concept and implement a binary search tree a meal (by the name of the meal). The key to use with the binary search algorithm should be the meal's name. Use the same data as program #3. Use of external data files is OPTIONAL.

Remember, for each meal you will want to keep track of:

1. Name of the meal
2. Name of the venue that offers this meal
3. The approximate price
4. A rating as to the quality (1-10), 10 being best
5. A review (what you thought of it the last time you ate this food there
6. If it is a cart or restaurant

Implement the following functions:

1. Constructor
2. Destructor and a recursive function to remove all
3. Add a new meal into the BST
4. Search for a particular meal (by name) (supply matching information back to the client program that matches the search key supplied, through the argument list); there may be more than a single match. Suggested arguments (where meal could be a struct or class):

   int search_meal(char * meal_name, meal matches[]);

5. Remove a particular meal at a given restaurant (so match the meal name but only remove it if that meal is offered at a particular restaurant)
6. Display all matches for a meal (so all restaurants that offer that meal)
7. Display all (in sorted order!).
8. **Get_Height** return the height of the tree (refer to Lab #7)
9. **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.

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!! Refer to the mathematics from 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.**

Things you should know...as part of your program:

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 must deallocate all dynamically allocated memory.