EECS 2510 – Non-Linear Data Structures and Programming in C++

Lab 1 – Due Tuesday, Feb 14, 2016 at Class Time (8:30 AM)

For this programming lab, you are to implement the seven Dynamic Set Operators using a Binary Search Tree (BST).

The nodes for your tree will contain a `string` (the "key"), an integer (a piece of "Satellite data," to count the number of times the `string` has been seen), and of course, left and right child pointers. You may implement the parent pointer, if you think it will help you implement your solution, but it is not required.

Your program will consist of a loop within `main`, which will accept commands from the keyboard, and execute them one-at-a-time. The commands your program ***must*** support are:

`insert` <string>	If <string> is already IN the set, increment the count (stored in the node containing <string>) If <string> is NOT already in the set, add it to the set (and set the count to 1) In either case, send `<string> <count>` to `cout`
`delete` <string>	If <string> is NOT in the set, output `<string> <-1>`. If <string> IS in the set, and has a count of more than 1, decrement the count corresponding to the string, and output `<string> <nnn>`, where nnn is the new count If <string> IS in the set, and has a count of exactly 1, delete it from the set, and output `<string> <0>`
`search` <string>	If <string> is in the set, output `<string> <nnn>`, where `<nnn>` is the count for the string. If <string> is not in the set, output `<string> <0>`
`min`	Output `<string>`, where `<string>` is the minimum value of the set. If the set is empty, output a blank line
`max`	Just like `min`, except output the maximum value of the set (or a blank line)
`next` <string>	If `<string>` is in the set, output its successor `<string>`. If `<string>` doesn't have a successor, or if it isn't in the set, output a blank line
`prev` <string>	Just like **`next`**, except output the predecessor of <string>
`list`	Does an in-order traversal, listing all of the strings in the tree in ascending order, along with the number of times each appears
`help`	List the commands available to the user
`exit`	Terminate the program

Note: the angle brackets shown above (<>) are not to be used for input or output; they're shown only to indicate that a string <string> or a number (<nnn>) is to be input or output (see sample input/output below). All of your lines of output should end in a new line (<< endl or "\n")

Your input ***commands*** should be case-*in*sensitive, but ***not*** the strings on which the commands operate (i.e., "cat" and "Cat" are two different strings, but "insert cat" and "InSeRt cat" would do the same thing). The strings will be single words, so you don't have to worry about parsing entire lines.

You should implement the BST as a class (perhaps called BST?), using an `#include` file for the interface, and a .cpp file for the implementation (see Savitch pp. 476-488). If you implement the nodes as their own class (a `struct` will suffice, so you don't have to), then you'll need another .h/.cpp pair for that, too.

Some of you may have had occasion to code in C (or in "C-Style" C++) somewhere along the way. You should NOT use the old-style C-Strings; rather, use the new standard `string` class (see chapter 9, Savitch). The new class functions much more like Java's `String` class, and in some ways is even more flexible (for example, you can use the `==` operator to compare the ***contents*** of two strings, rather than their ***references***).

You already have snippets of code (in the lecture slides) for some of the operations you will need, and you have pseudocode for all of the set operators. You may need to code some other methods, but the main ones should be rather straightforward.

All code you write for this assignment must be yours and yours alone (except for what is in the lecture slides and the Savitch text, of course). You may ***not*** use any code from any other source, including the Internet, even as a reference. Using code other than what you write (or are explicitly permitted to use) will be considered academic dishonesty, and will be dealt with in most severe terms.

Sample input and output:

Input	Output
`search cat`	`cat 0`
`min`	`<no output>`
`max`	`<no output>`
`next cat`	`<no output>`
`insert cat`	`cat 1`
`insert cat`	`cat 2`
`search cat`	`cat 2`
`delete cat`	`cat 1`
`insert dog`	`dog 1`
`min`	`cat`
`max`	`dog`
`list`	`cat 1`
	`dog 1`
`next cat`	`dog`
`prev dog`	`cat`

Your program should utilize good programming practices – information hiding, etc. Your public methods should not give away HOW the BST works – `main()` should never see your BST's root pointer, for example. You will have to create some public methods that, in turn, call private methods to get the job done. Your "dynamic set" could just as easily be implemented with arrays, a linked list, or something else. What gives it its behavior is its *interface*; *not its implementation*!

Your code must be well-documented:

- Use block comments at the start of each method, briefly explaining *what* it does, and *how* it does what it does.
- Use line comments liberally to explain what's going on
- Use internal documentation, like descriptive variable names where appropriate
- Make SURE you have a suitable block header on ALL of your source files WITH YOUR NAME, THE COURSE, AND THE DATE!

Some helpful starting points:

- Create your program as a Win32 Console application within VS 2015
- Your program should use the `std` namespace; NOT the `System` namespace
- To get access to `cin` and `cout`, use "`#include <iostream>`"
- To get access to the `string` class and its supporting code, "`#include <string>`"

DO NOT WAIT TO GET STARTED ON THIS – YOU WILL END UP WRITING SEVERAL HUNDRED LINES OF CODE FOR THIS ASSIGNMENT; IT'S TOO BIG TO DO AT THE LAST MINUTE, AND THERE WON'T BE ANY EXTENSIONS GIVEN ON THIS ASSIGNMENT. DON'T LET THE FACT THAT YOU ALREADY HAVE PIECES OF THE PSEUDOCODE LULL YOU INTO A FALSE SENSE OF SECURITY. IF YOU'RE NEW TO C++, YOU STILL HAVE A LEARNING CURVE TO CLIMB! IF YOU'RE ALSO NEW TO VISUAL STUDIO, THEN YOU HAVE TWO LEARNING CURVES TO CLIMB!

IF YOU RUN INTO PROBLEMS, AND HAVE MADE A GOOD ATTEMPT AT SOLVING THEM, E-MAIL ME, RATHER THAN WASTING TIME LOST IN THE WILDERNESS

I strongly suggest your write this code incrementally. You might want to start with just the command-soliciting loop. Next build a shell of the BST class, and finally, implement and test the functionality one method at a time.

To turn in your program, use 7-Zip (www.7-zip.org) to create a compressed archive of your entire Visual Studio workspace (don't just submit your source and/or .exe files). Submit your 7-Zip archive to Blackboard.

As for testing, you may want to create some test scripts, and store them in text files. Assuming you called your program BST, then you can redirect input from a text file (from the command line) using the "<" operator. If TEST1.TXT contains a set of data you want to test your program against, then you can use BST < TEST1.TXT to have the input that *would* have come from the keyboard come *instead* from TEST1.TXT