CS 474: Object Oriented Programming Languages and Environments

Spring 2014

First C++ project

Due time: 7:00 pm on Saturday 4/19/2013

You are required to implement the all-too-familiar *Set Calculator* one more time for this project. As with the previous projects, you must implement sets as binary search trees (BSTs). However, this time you are required to use the language C++98 or C++03. Also the sets will contain C-style strings (implemented using the char* type). Another difference is that this set calculator will use a command line interface, rather than a graphical user interface. You will maintain two sets at any point in time, S_1 and S_2. The calculator supports the usual operations by responding to commands below.

Recall that BSTs are binary trees subject to the following properties. First, a string value is associated with each node. Second, each node can have at most two children, a left child and a right child. Third, given a node *x*, the values of all nodes in the left subtree of *x* are less than the value of *x*, and the values of all nodes in the right subtree of *x* are greater than the value of *x*. Use lexicographical ordering to compare strings. (You are encouraged to use the ASCII code of each string character to determine the value of that character.) No duplicate values will be allowed in your trees.

As explained in class, you are required to implement the big three for your binary search tree class, namely the copy constructor, the assignment operator and the destructor. The copy constructor and assignment operator must perform a deep copy; the destructor must delete all dynamically-allocated objects used exclusively by the receiver.

Use a command line interface for entering the commands below. Your command line interface will prompt the user for a command, and then execute the command. Here is a list of commands. Make sure not to cause any memory leaks or dangling pointers in the implementation of these commands.

1. e — Erase set. This command allows interactive users to delete the current S_1 set. The previous value stored in S_1 is lost.
2. s — Switch sets. The sets associated with S_1 and S_2 are swapped, meaning that S_1 will receive the previous S_2 set and vice versa.
3. c — Copy set. Set S_1 is deep copied into S_2. The previous content of S_2 is lost. The content of S_1 is not affected. The two sets must not share any data structures, that is, they can be modified independently of each other.
4. l — List set contents. The string values stored in the two sets are displayed on the standard output stream. The two sets are not modified.
5. a — Add element. This function allows a user to add a new string to S_1. The value is obtained through an appropriate prompt with an interactive user. No action is taken if the string in question is already in the set. The insertion should preserve the BST properties of S_1.
6. u — Union. This element takes the set union of S_1 and stores the resulting value in S_1. The previous content of S_1 is lost. S_2 is not modified by this operation.
7. i — Intersection. This command takes the set intersection of S_1 and stores the resulting value in S_1. The previous content of S_1 is lost. S_2 is not modified by this operation.
8. q—Quits the set manager.

You must work alone on this project. Save all your code in a collection of header and code files and submit a zip archive with a (short) readme file containing instructions on how to use your Set Calculator. Submit the archive by clicking on the link provided with this assignment. Your code should compile under the GNU C++ compiler. No late submissions will be accepted.