CSE250, Spring 2014 **Assignment 8** **Due Friday 4/18**

`hardcopy only, in-class submission`

Prelim II will be in class period on **Friday, Apr. 25**, covering the text through Chapter 8 and assignments through this one. It will have the same terms as Prelim I: closed book, closed-notes except for 1 notes sheet.

For next week, Finish Chapter 8 and read Chapter 9. The "Phone Directory Case Study" will not be covered in detail but is recommended; the Huffman Code completion may be skipped. Also look ahead to Chapter 10, aspects of which we've already touched on—skip the more esoteric sorting algorithms, namely "Bubble Sort" and "Shell Sort," but notice especially the use of *generic code* with template variables standing for kinds of iterator. Read section 10.11 mainly as an example of issues with indices versus magnitudes and "off-by-1" errors, harking all the way back to early assignments. *Text tweaks:* p518 middle—the text describes just one of the three actual STL `erase` functions for `multiset`. We have been referencing the `erase` that takes an iterator argument and erases just that one element. And regarding pp535–536, the proposed hash-table additions to the STL, such as `unordered_map` (can Google that), will of course provide iterators to traverse the hash tables, but the text is right insofar as the resulting "hash order" is haphazard.

Project 2 will be due on the last day of classes (5/9) and will have teams of 2. On your Problem Set 8 submission, please rank (at least) three possible project partners. Not everyone will get his/her first choice, and preference will be given to partners who have signed in to the same recitation session (or can at least attend it).

(1) Say two strings x and y have "edit distance 1.5" if one can be obtained from the other by one character deletion and one character change. Using iterator-based code (or pointer-based since they have the same syntax), write on paper a function `ed15` that tests whether two strings have edit distance 1.5. You may optionally accept if they have edit distance 1 or less, but note that "transpose" does not necessarily imply "ed1.5." *Your code must run in $O(n)$ time, where n is the length of the longer string.* Be sure to write comments in your code sketch that explain the strategy and prove the running time. (For a hint, since "iterators are cheap" you may freely use multiple copies of them. 18 pts.—this is the "missing piece" of Assignment 5 but updated to use iterators.)

(2) Draw the binary search tree that results from inserting the words of this sentence in the order given, allowing duplicate keys. Use alphabetical order of lowercased words with the lower words at left. Then show the results of deleting all three occurrences of the word "the", one at a time. (It is OK to use either the inorder successor or predecessor for deletion, and putting an equal key left or right, but please show each step separately on the relevant part of the tree—you do not have to re-draw the whole tree each time. 12 + 9 = 21 pts.)

(3)

For each task below labeled 1.–10., say which of the following best describes its running time:

(a) Guaranteed $O(1)$ time.

(b) Amortized $O(1)$ time.

(c) Usually $O(1)$ time.

(d) Guaranteed $O(\log n)$ time.

(e) Usually $O(\log n)$ time.

(f) Guaranteed $O(n)$ time.

In all cases n denotes the number of items currently in the underlying data structure, and any other parameters are stated. The variable `vec` stands for a vector, `dlist` for a doubly-linked list (not necessarily sorted), `valli` for a "`Valli`" data structure, `bst` for a BST—i.e. a general binary search tree, `itr` for an iterator of the appropriate kind, `comp` for a typical comparison function-object, and `item` for a typical item in the data structure. *Justifications* are not required, but might help for partial credit. The tasks are:

1. For a BST iterator `itr`, the call `itr++;`
2. `make_heap(vec.begin(), vec.end(), comp);`
3. `dlist.push_back(item);` (equivalently, `dlist.insert(dlist.end(),item);`)
4. `vec.push_back(item);`
5. `dlist.erase(itr);`
6. `vec.erase(itr);` where `itr` points in the middle of `vec`.
7. For a Valli iterator `itr`, the call `itr++;`
8. `valli.find(item);`, assuming `valli` just refreshed with `ratio` $r = \log_2(n)$.
9. `bst.find(item);`
10. Preorder traversal of a BST.

($10 \times 3 = 30$ pts., for 69 total on the set)