CSE250, Spring 2014 Prelim II Name:_____
Apr. 25, 2014 St.ID#:_____

Closed book, closed-notes-except-for-1-sheet, closed neighbors, 48 minutes. This question paper has TWO problems. Please do both in the exam books provided. The first problem is True/False **with justifications**. The exam totals 67 pts., subdivided as shown.

The term `FlexArray` refers to a data structure consisting of a doubly-linked list of nodes, where each node holds an array of up to some number c of elements. The data structure has the same public operations as `vector` (plus the extra versions of `insert` and `erase` with an index argument), and provides an iterator that is at least bi-directional. If and when a node's array hits (or exceeds) size c, the node is split into two nodes, each with $c/2$ elements give-or-take one. `Foo` and `Bar` are the usual generic filler names for classes or other types.

(1) (9+3+3 = 15 pts.)

(a) Show the result of inserting the words `bad bed bid bud act beg bet dog` in that order into a binary search tree, using alphabetical order from 'a' at left to 'z' at right/

(b) Now show the result of erasing `bid`. You may use either the predecessor or successor node to move in its place, but should say which.

(c) Show the *preorder* transversal of the tree—which won't be alphabetical order.

(2) ($7 \times 4 = 28$ pts.)

True/False **with justifications**: Write out the word `true` or `false` in full, and then *write a brief but topical justification*. The justification is worth 2 of the 4 pts. for each question.

1. In a `FlexArray` data structure with $n > c$ elements, in which only inserts and no erasures have been performed, each node always has at least $c/2 - 1$ elements.

2. Same question as 1., but now allowing erasures.

3. In a `FlexArray` that has just been built with no erasures, the next insertion is guaranteed to run in $O(\sqrt{n})$ time, even if it causes a node to split.

4. The default copy constructor in a class `Foo` will copy a `Bar` object that is held by the field `Bar* element;`—for example, `Foo` could be the `ChunkNode` class and `Bar` is `vector<T>`.

5. The middle element in a `vector` with an odd number of elements can always be accessed in $O(1)$ time.

6. The middle element in a `FlexArray` with an odd number of elements can usually be accessed in $O(1)$ time.

7. A step in the *postorder* transversal of a binary search tree runs in amortized $O(1)$ time.

(3) 24 pts. total

Suppose we have a word chain in which one word `x` is related to the previous word `w` by a character change (so `hd1(w,x)` holds), and the next word `y` is obtained by inserting or erasing a character, which we'll call `id1(x,y)` for "insertion distance one." Examples are the sequences `rain raid rid` and `rain raid rabid`. We could also have this in reverse with the insertion-or-erasure first, as in `want ant act` or `bake baker biker`. In either case, if you erase the *middle* word `x`, the remaining two words are still related by one-change-plus-one-insert/erase. That is, `ed15(w,y)` still holds, with reference to the "edit-distance 1.5" concept on Assignment 8, and vice-versa: if `ed15(w,y)` holds then we must have one of these two cases.

Write a routine `void reduce(F& chain)` that iterates through the chain and erases such words `x`. What is `F`? It could be `FlexArray<StringWrap>` like on Project 1, but it could be a different container—all you know is that it has an iterator `F::iterator` which obeys the same standard interface for an iterator. For some help, if you have an iterator `nit` on a word `w`, then the lines

```
F::iterator pit = nit++;
F::iterator it = nit++;
```

will leave you with the iterator `it` on word `x`, the iterator `pit` on what you now think of as the previous word `w`, and `nit` has done a post-increment advance twice to end up on the next word `y`—unless `w` was too near the end, that is. One final helpful rule—even if you erase `x` and the word `z` after `y` leaves `w y z` as a sequence where `y` too could be erased, `don't`—you may consider `y` to be the next "previous word `w`" to consider. (This plus the above lines and the standard way `erase(it)` returns an iterator on the next word enable you to avoid having to move an iterator backwards.) Your answer must involve only iterator code—no `peekIndex` or other stuff that would be too specific in the data structure—but of course you may write calls to `ed15(...)` and `hd1,id1` assuming they have already been coded since this is client code. End of Exam.