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.

[This is a partial translation of the 2014 C++ version, with `BALBOADLL` in place of what was called `FlexArray` that year. We have not yet covered the idea of re-shaping the data structure with a different pair of (r, m) values when the number m of nodes in an array exceeds a *capacity* limit c.]

The term `BALBOADLL` refers to a data structure consisting of a doubly-linked list of nodes, where each node holds an array of up to some capacity number c of elements, $c = 2^k$ being a power of 2. The data structure has the same public operations as `ArrayBuffer` (including versions of `insert` and `remove` with an index argument), and provides an iterator `Iter` 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. [Spring 2022 note: In 2014, `BALBOA` (then called `FlexArray`) had the policy that whenever the number m of elements in one of the arrays reaches a "capacity" threshold c (which you could assume to be a power of 2 if you want), the array splits into two arrays of size $c/2$ (or one array has one fewer element if c is odd). And the number r of arrays goes up by 1. When elements are removed, however, an array is left alone unless it becomes completely empty, in which case it is removed and r goes down by 1. This year I have kept $r = 1$ and no capacity limit so far—the intent is to talk instead about a `refresh(r)` method that periodically reshapes the whole data structure into r arrays of size about $m = n/r$, "give-or-take" some elements in individual arrays. I have specified `BALBOADLL` here because the way your `BALBOA` uses indexing with native Scala lists technically changes the answers.]

1. In a `BALBOADLL` 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 removals.

3. In a `BALBOADLL` that has just been built with no removals, and with $c \approx \sqrt{n}$ where n is the total number of elements, the next insertion is guaranteed to run in $O(\sqrt{n})$ time, even if it causes a node to split.

4. The middle element in an `ArrayBuffer` with an odd number of elements can always be accessed in $O(1)$ time. [Ah: In C++ this is true; in Scala it may be officially false.]

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

6. The middle element in a binary search tree 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

[Spring 2022 note: The "write code" problem below referred to projects that year, which had to do with how much one word needs to be edited to become another word. Their "Assignment 8" had iterator code similar to your assignment 6. Its setup used the postfix `++` operator, which you may have seen in CSE220, and whose semantics is that when you write `pit = it++`, you call `it.next()` but `pit` keeps the original location of `it`, so that now it is the predecessor of `it`. This can compensate for not having a `prev()` operation, which in C++ lingo is the `--` operator.

Your corresponding problem will use the notation of assignments 4 and 6. Here is an easier substitute problem that covers part of the intended scope:

Suppose you were not guaranteed that the `find` method of a sorted data structure always gives the first element of a range of equal-key elements. Suppose that the iterator associated to that data structure has `prev()`, which behaves like the standard `next()` operation in terms of returning the *current* element, but which moves the iterator to the *previous* position. Note that we don't need a separate `hasPrev` method because `hasNext` really refers to the current location. Sketch how you could code the inner body of the task of Assignment 6 to handle the situation (which actually happened) where the binary search routine didn't give the start of the range. (In fact, it gave the last of an equal-key range. You may suppose also that if `itr` equals `begin`, then `itr.prev()` moves it to the `end` position rather than give an error—this is another reason why data structures are often implemented in a "circular" manner. You need not give all the task details—just show how your `while` loops(s) work.)]

[The rest was from Spring 2014.] 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 [the Spring 2014 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 `BALBOADLL[String]` 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.