CSE250, Fall 2010 — Prelim II — Dec. 1, 2010

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) ($10 \times 4 = 40$ 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. $999n^2 + 99n + 9 = o(n^3)$.
4. If $f(n) = O(t(n))$ and $g(n) = O(t(n))$, then $f(n) + g(n) = O(t(n))$.
5. If a class `Foo` has a field `Bar* p`, then the destructor `~Foo()` should always include the line `delete p;`
6. If a class `Foo` has a field `vector<Bar*> elements;` then the default assignment operator will copy the vector's pointers, but will not copy the `Bar` objects they point to.
7. If `itr` is a bi-directional iterator (on any data structure that can support it), then the lines

   ```
   *itr++ = x;
   *itr = y;
   return *(--itr);
   ```

   always return the value of `x`.
8. The middle element in an array with an odd number of elements can always be accessed in $O(1)$ time.
9. The middle element in a doubly-linked list with an odd number of elements can always be accessed in $O(1)$ time.
10. The middle element in a `FlexArray` with an odd number of elements can always be accessed in $O(1)$ time.

Problem (2) is overleaf.

(2) (6+9+3+9 = 27 pts.)

(a) A `FlexArray` can pop its rear element by calling its `erase(size_t i)` method with argument `size() - 1`. In particular, the `Deque2Flex` adapter provided for Project 1 translated a call to `popRear()` using the line

```
myImpl->erase(myImpl->size() - 1);
```

Explain, however, why this *fails* to provide an implementation of a deque data structure in which each deque operation can be completed in $O(1)$ time. (This assumes one has coded this version of `erase` using the while-loop through nodes that was exemplified on Assignment 6 for the `at` method; if you coded it differently you may answer based on your code's strategy.)

(b) Show how you would code a `T popRear()` method directly into the `FlexArray` class so that it runs in $O(1)$ time. Here `T` is the template parameter. Write the code in C++, assuming the class has a pointer `endNode` to a dummy end node. In your code you may ignore the requirement to de-allocate a node if its size falls to zero.

(c) Is the $O(1)$ time in (b) always true, even if the pop makes the last node become empty? Or is it true only in an "amortized" sense because in a long sequence of successive pops, the empty-node case happens fairly rarely? Justify your answer briefly.

(d) Now suppose we can use the other version of the `erase` method, namely `iterator erase(iterator me)`, whose argument is an iterator not an index. Write code for `T popRear()` that runs in $O(1)$ time using only the public functionality of `FlexArray`—i.e. without access to the private fields of the class—so that it could go in the `Deque2Flex` adapter (or the `Deque2FlexI` version with iterators).

End of Exam