When Good Iterators Go Bad... (must!)

CS 2250 Week 7/8

I Consider first a "two-pointer" implementation of a *forward* iterator for a singly-linked list:

```
template <class I> // say "class" if REQ a member method like str()
class SList/Vall/etc {
  ...
  class iterator {
    Cell* prev;
    Cell* curr;
  };
```

CLASS INV: curr = prev->next

← *friend* lines go here, for Cell as well

Both the pre-inc operator++() and the post-inc operator++(int dummy) then have in their bodies:

```
prev = curr;
curr = curr->next;
```

or

```
curr = curr->next;
prev = prev->next;
```

Now the code for inserting an item, keeping sorted, is:

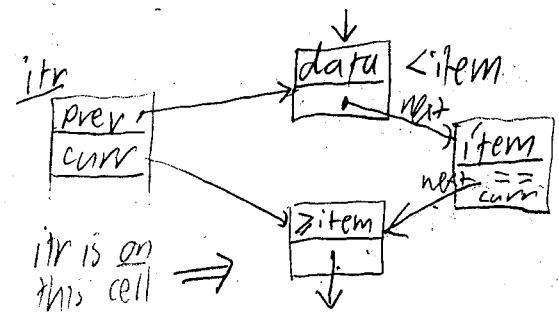
```
iterator itr = begin();
while (itr != end() && *itr < item) {++itr;}
```

//POST: *prev* is on last data that *was* < item.
//So insert new node *after* prev, *before* curr:

```
(*) itr.prev->next = new Cell(item, itr.curr);
```

(≡ ≡ see why?)

// works even if // itr == end(), // ie curr == NULL



PROBLEM 1: curr no longer == prev->next.

PROBLEM 2: If you try to *restore* the CLASS INV by

```
(**) itr.prev = itr.prev->next;
```

and if lines (*), (**) are in your private insert method

```
iterator insert(const iterator& itr, const T& item) {...}
```

you run into the roadblock that you can't do line (**) because itr is *const*. You can't change itr.prev! Unavoidably, in this design, the iterator *itr* that was used to call the private insert becomes *invalid*, because its CLASS INV no longer holds true.

You *can*, however, return a *valid* iterator by

```
(**) return iterator(itr.prev, itr.prev->next);
```

valve return, using the two-parameter constructor that simply sets the two fields.