When Good Iterators Go Bad...

Consider first a 'two-pointer' implementation of a forward iterator for a singly-linked list:

```c++
template <class T> // say 'class' if REAL a
class SList<ValT> etc. { 

    class iterator { ... 

        friend lines
        Cell* prev;
        Cell* curr;  
        [CLASS INV: CURR = PREV->next]

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

    prev = curr;
    curr = curr->next;
    or
    prev = prev->next;
    curr = curr;  

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);
    if (== == see why?)

Problem 1: curr is no longer = prev->next.

Problem 2: If you try to restore the class INV by

(a) itr. prev = itr.prev->next;

and if lines (*) and (**) are in your private insert method
iterator insert(const iterator& itv, const T& item) { -- } 

you run into the roadblock that you can't do line 105
because *itr is const.
You can't change it.r.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, itv.prev->next);

value returned using the two-parameter constructor
that simply sets the two fields.