II. Now consider an implementation with just the prev ptr.

The bodies of the ++ ops have prev = prev->next;
The linear-search code for a place to insert is unchanged
until:

itr prev data item next
itr was on this cell...

itr.prev->next = new cell(item, itr, prev->next);
ok with const - under ++ hood?

Since there was just 1 field, no INV to worry about
Since it's OK to change a field prev->next below

However, itr did change from being on the first

dato item (or end) after the while loop, it being on the
new item! And perhaps finishing with "return itr" is not a
totally safe idea. But one can do: return iterator(itr, prev);

Note: now this doesn't work if end is a NULL pointer,

since then itr, curr->prev won't exist! Hence with a
doubly linked list it is common to use a dummy end node.
A dummy head node is not as neat as with a single-linked list,
but without it you still have to treat insertion before begin() as a special case, since itr, curr->prev->next may not exist!

III. Doubly-Linked List Iterator - can use just a single curr pointer. To insert before:

itr curr prev

itr is on the same cell curr points to

itr, curr->prev = itr, curr->prev->next
itr = new Cell(itr, curr, prev, itr, curr);

Although "double assignments" are sometimes frowned upon,
the one used to insert the new cell before itr is an idiom:
itr, curr->prev = itr, curr->prev->next
itr = new Cell(itr, curr, prev, itr, curr);

Consider first initializing the fields in order of the:
constructor should/must use the same order.

In your DOVE class orders fields with

data before prev & next, your simple-setter
constructor should/must use the same order.