More Notes About Lab 4
Question. Suppose list L contains exactly the data 1 2 3. What would you expect the following code to print?

```java
ListIterator<Integer> iter = L.listIterator();
while (iter.hasNext())
    System.out.print( iter.next() );
while (iter.hasPrevious())
    System.out.print( iter.previous() );
```

A) 1 2 3 2 1  
B) 1 2 3 3 2 1  
C) 3 2 1 2 3   
D) 3 2 1 1 2 3
Answer B:

If L is our list and it currently contains data 1,2,3 the code

```java
ListIterator<Integer> iter = L.listIterator();
while (iter.hasNext())
    System.out.print( iter.next() );
while (iter.hasPrevious())
    System.out.print( iter.previous() );
```

will print

```
1  2  3  3  2  1
```
Note that the third call to next() and the first call to previous() both return 3. In general a call to next() followed by a call to previous() should have both calls returning data from the same node.

To arrange this, remember that we want to think of the *current* marker of the iterator as pointing *between* two elements of the list: the one just visited and the one to come next. We can't really achieve that with a linked list because there is nothing between two nodes to point at. We need to point either at the node to the left of this spot or the node to the right. You can set up the code either way.

In the following pictures I will set the *left* pointer to the box just before the current logical position.
Suppose *left* is pointing at the box with datum 2. We are saying this is to the left of our logical position, so our actual location is between box 2 and box 3. On a call to `next` we move to the right and return 3. Here is code that achieves this:

```python
left = left.next
return left.data
```
Our logical position is now between the 3-node and tail. A call to `previous()` should move `left` to the left and return 3. What code does this?

A

```java
left = left.prev;
return left.data;
```

B

```java
return left.data;
left = left.prev;
```

C

```java
Node p = left;
left = left.prev;
return p.data;
```

D

```java
Node p = left.prev;
return p.data;
left = p;
```
Our logical position now is between the 3-node and the tail. A call to previous() will move *left* to the left and return 3:

Node p = left;
left = left.previous;
return p.data;
So the call to \texttt{next()} and the call to \texttt{previous()} both return 3 and we are back to our original picture:
Note that the codes for `next()` and `previous()` do the same steps in different orders:

**next():**

```java
left = left.next
return left.data
```

**previous():**

```java
Node p = left;
left = left.previous;
return p.data;
```
If you choose to make variable *right* point at the node to the right of the in-between location you will get slightly different code that is just as reasonable. Either way you need to see how your pictures lead to code for hasNext() and hasPrevious() as well as the iterator add() and remove() methods.