Notes About Lab 4

If you read the lab directions carefully and draw pictures of your structures as you code, most of Lab 4 is straightforward. The place students tend to find confusing is in the interaction between calls to iterator.next() and iterator.previous().

Clicker Question. Suppose list L contains exactly the data 1 2 3
What would you expect the following code to print?

```
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

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

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

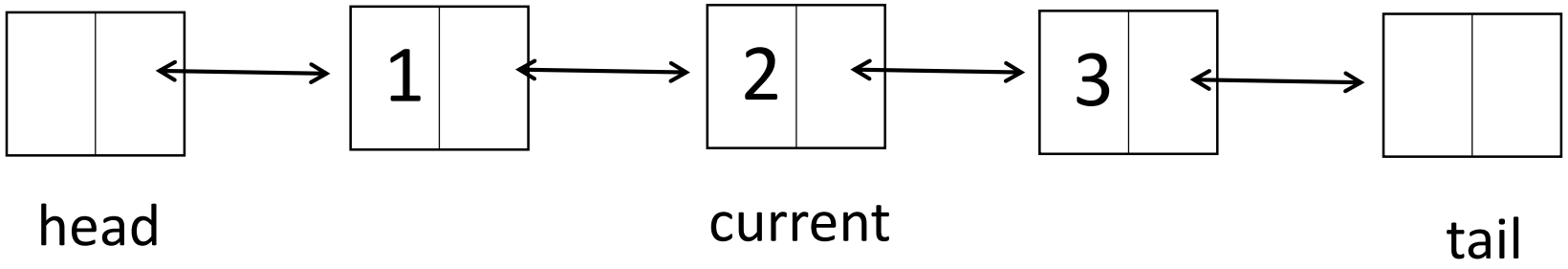
will print

```
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.

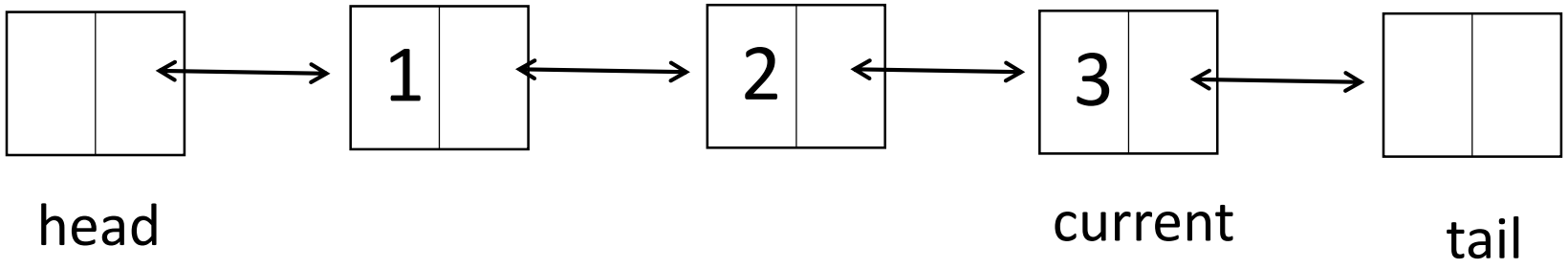
To arrange this, remember that we want to think of the *current* marker of the iterator as pointing *between* two elements of the list: the one just visited and the one to come next. We can't really achieve that with a linked list because there is nothing between two nodes to point at. We need to point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



Suppose *current* 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:

```
current = current.next
return current.data
```



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

A

```
current = current.previous;
return currrent.data;
```

B

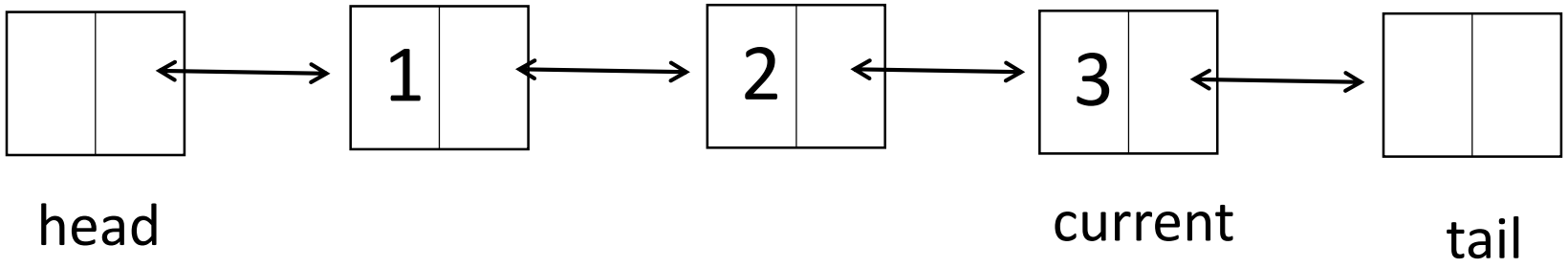
```
return current.data;
current = current.previous
```

C

```
Node p = current;
current = current.previous;
return p.data;
```

D

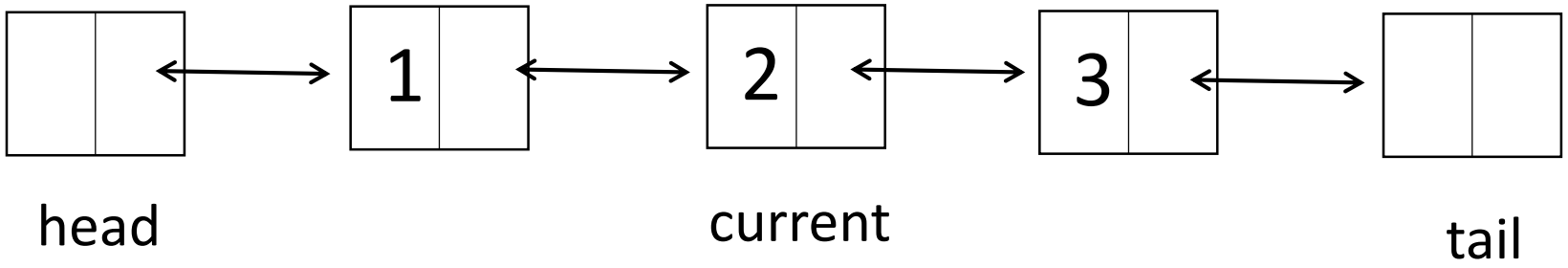
```
Node p = current.previous;
return p.data;
current = p;
```

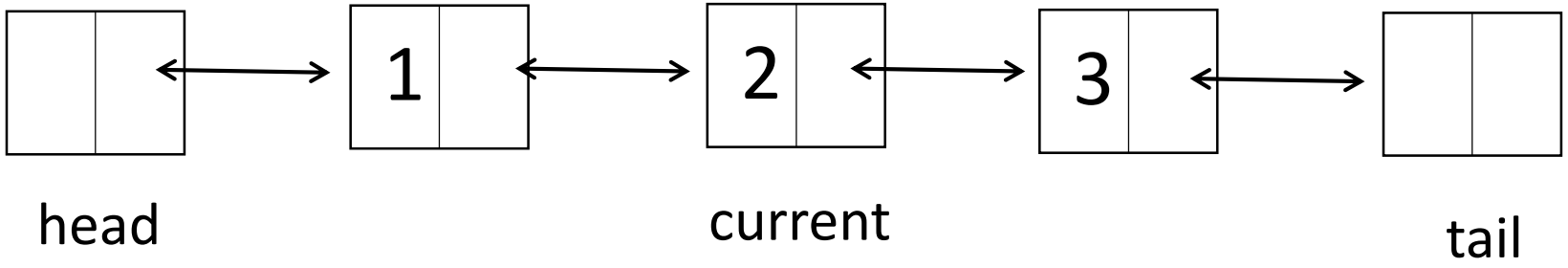


Our logical position now is between the 3-node and the tail. A call to previous() will move *current* to the left and return 3:

```
Node p = current;
current = current.previous;
return p.data;
```

So the call to next() and the call to previous() both return 3 and we are back to our original picture:





Note that the codes for head() and previous() do the same steps in different orders:

```
next():
current = current.next
return current.data
```

```
previous:
Node p = current;
current = current.previous;
return p.data;
```

If you choose to make *current* 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.