Recall the method `hmmmm()` from the pre-tutorial exercise. Here is its "signature" (with a better method name) and external comment:

```java
public static ListNode nodeBefore (ListNode first, ListNode n) {
    // Returns a reference to the node before node 'n' in the linked list
    // whose first node is referred to by 'first'.
    // If 'n' itself is the the first node, returns null.
    // If 'n' does not occur in the linked list, returns null.

    public static void whuuut (ListNode first) {
        ListNode temp = first;
        while (temp.link != null) {
            temp = temp.link;
        }

        while (temp != null) {
            System.out.println(temp.data);
            temp = nodeBefore(first, temp);
        }
    }
}
```

1. Trace a call to `whuuut()` with the following linked list:

```
  first  ---->  "a"  ---->  "cat"  ---->  "hi"  ---->  "it"  ---->  "do"  .
```

What would be printed? (Hint: There is no need to trace what happens within the calls to `nodeBefore()`; just assume that it fulfills its contract, as specified in the comments.)
2. At least in this case, the method doesn’t crash and does do something sensible. But does it *always*? Below is a table showing every time we say \( x, y \) in the method. For each one, try to convince yourself that this cannot crash because \( x \) cannot be be null at this moment. If you are sure it cannot crash, write an assertion saying how you know that. If you cannot, you have uncovered a bug; fix it.

<table>
<thead>
<tr>
<th>Occurrence of ( x, y )</th>
<th>Assertion saying why it’s okay OR write “bug”</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{temp.link} \neq \text{null} )</td>
<td></td>
</tr>
<tr>
<td>( \text{temp} = \text{temp.link} );</td>
<td></td>
</tr>
<tr>
<td>( \text{temp} \neq \text{null} )</td>
<td></td>
</tr>
</tbody>
</table>

3. What would be a better name for this method? ____________________________

4. Write an appropriate external comment for the method.

5. **Challenge question:** Write a method that prints out the contents of a linked list in reverse order, using this technique: Go down the list from beginning to end, reversing the pointers as you go. Then follow the pointers from the end back to the beginning, printing out the node contents as you go (and also restoring the pointers to the forwards direction). Suggestion: Try doing this by hand, with a picture, before you write the code.