1. Here again is the code from your pre-tutorial exercise:

```java
public static int mystery(BinaryIntNode root){
    if (root.left == null && root.right == null)
        return 1;
    else if (root.left == null)
        return 1 + mystery(root.right);
    else if (root.right == null)
        return 1 + mystery(root.left);
    else
        return 1 + mystery(root.left) + mystery(root.right);
}
```

Agree with your team on what it is that this method does. Then write a simpler body for it. Hint: make it work even when *root* is null.

2. Write a method that, given a reference to the front node in a linked list, prints the linked list in reverse. Assume you have a *ListNode* class with instance variables called *data* and *link*. 
3. Below is a method from assignment 5, and a simple class that it uses:

// Return a BinaryIntNode that is the root of a tree containing the first n integers in st.
// If n is less than 1, return null. Precondition: n <= the number of integers in st.

private BinaryIntNode buildTree(StringTokenizer st, int n){
    if (n < 1)
        return null;
    else {
        int half = (n-1)/2;
        int remainder = (n-1) - half;

        BinaryIntNode leftSubtree = buildTree(st, half);
        // Build a node and put the next integer from the
        // Stringtokenizer into it.
        BinaryIntNode node = new BinaryIntNode();
        node.key = (new Integer( st.nextToken() )).intValue();
        BinaryIntNode rightSubtree = buildTree(st, remainder);

        node.left = leftSubtree;
        node.right = rightSubtree;
        return node;
    }
}

When calling this method, we can construct a StringTokenizer from a String, such as “10 3 9 8 1 3 4 3 4 -8 1”. Suppose we want buildTree() to build exactly this tree:

```
     8
    / \
   9   1
  / \
 3   2   4   5
```

Exactly what string do we have to pass to it?