Background

The code we will review in this problem works the following a linked list type. For simplicity of notation, we have left the fields public.

```
public class Node {
  public final int value;
  public final Node next;

  public Node(int value, Node next) {
    this.value = value;
    this.next = next;
  }
}
```

An empty listed is represented by `null`, while a `Node` object represents a list that starts with `value` and is followed by the list `next`. For example, the list [1,2, 3] would be created like this:

```
new Node(1, new Node(2, new Node(3, null)));
```

The code below returns a new list that is the reversal of the given one. To explain how it *should* work, we define a mathematical function, rev, that describes what it is supposed to do:

rev([]) = []
rev([v] + L) = rev(L) + [v] *for any* integer v and *any* list L

This mathematical function is defined in terms of what its argument *looks like*:

- When the argument is an empty list, the first rule says that rev returns an empty list.
- When the argument is not an empty list, then it is a list with at least one element, which we describe as the concatenation ("+") of the first value, v, and the rest of the list, L. The second rule says that it returns the list formed by calling rev recursively on L (to reverse that list) and then concatenating that with the list [v]. The resulting list has v at the end.

This definition is short and simple. However, if we directly translated it into code, we would get a reversal algorithm that runs in $\Theta(n^2)$ time, where n is the length of the list, because each list concatenation ("+") requires O(n) time. That is unnecessarily slow.

Our goal below is to check the correctness of a reversal method that runs in $\Theta(n)$ time. In this case, correctness means that it should return the same result as the rev defined above.

Hints:

- If L has type Node, then you have either L = [] or L = [L.value] + L.next. If L is `null`, then it is the empty list, i.e., L = []. Otherwise, it is a Node object, which means it is the list containing L.value followed by the items in L.next, i.e., L = [L.value] + L.next.
- You may use without proof the fact that A = rev(B) iff rev(A) = B for any lists A and B.

Verifying Correctness

Fill in the missing assertions by reasoning in the direction indicated by the arrows. In each place where two assertions appear next to each other with no code in between (where "?"s appear), provide an explanation of why the top assertion implies the bottom one.

Note: use math notation in your assertions, e.g., [] rather than "null".

```
Node reverse(Node A) {
 {{ }}
↓ Node R = A;
 {{ _____ }}
↓ Node L = null;
 {{ _____ }}

?

 {{ Inv: A = rev(L) + R }}
 while (R != null) {
↓
  {{ _____ }}

?

  {{ _____ }}
↑  L = new Node(R.value, L);
  {{ _____ }}
↑  R = R.next;
  {{ _____ }}
↑
 }
↓
 {{ _____ }}

?

 {{ Post: L = rev(A) }}
 return L;
}
```

Code Review

We have now verified that the code is correct. However, a code review can give other kinds of feedback beyond just spotting bugs. In this case, we can provide feedback about how the code could be written to make it easier for others to understand why it is correct.

The code above includes no comments other than the description of the loop invariant. In your opinion, what other comments do you think the author should have provided? What should they have explained in those comments (if anything) to make it easy for you to check its correctness? (More than one answer is reasonable here.)