Comparing the ease of implementation of the positional list in textbook and our linked list, the positional list is more complicated.

First of all, it has an inner class *Position* which initializes the position of a node for the list to access. The *position* is not simply an index like a number but an abstract type (a node reference, according to the textbook). In our linked list, we use simple link to connect each node and put each node in an order with ascending indices, just like the array. However, in the positional list, we don't need to access the position of node in a specific order, but we can directly access it if we know which position (or node) we are looking for. This *Position* class requires more efforts to design and implement.

If we look at the methods in positional list, we can see that the number of methods is greater than that in our linked list. In the implementation of the positional list, it requires a **_validate** method to check the validity of a called position, more accessors such as **before(self, p)**, and more mutators such as **add_after(self,p,e)**, to ensure that most of its implementation are in O(1) and thus achieve its purpose of design. Comparing to our linked list, it's more complicated for the programmer to design and implement.

With respect to the performance implication, our linked list is more similar to the typical array type, which has a few methods with O(n) performance such as **insert**. Nevertheless, because the purpose of the positional list is to eliminate the use of indices and call the *position* directly, it has more methods running in O(1) to make the manipulation more efficient in performance. Methods such as **insert** and **remove** now run in constant time rather than linear time (well, they're a bit different from those in our linked list but they have similar functions and concepts).

In conclusion, comparing to our linked list, the implementation of the positional list sacrifices the ease of implementation since it requires an extra abstract data type and several more mutators and accessors. However, it benefits from it for having a better performance efficiency in some cases. There's not an absolutely better implementation of the linked list. Each of the implementations have some advantages over the other and some drawbacks. We programmers should choose the most suitable one to meet our needs when we considering implementing a linked list.