Assignment 3: Lists

CS230
Fall 2000

Allen B. Downey
Computer Science Department

Due: Wednesday 4 October

The purpose of this assignment is to implement some of the methods that operate on lists. There are three points to this exercise: practice with manipulating embedded references, thinking about interface design issues, and building infrastructure for future assignments that will use lists to implement other data structures.

Set up List

1. Go to the web page and get the code from:

   `http://rocky.wellesley.edu/cs230/code/hw03`

   Compile and run the program. You should get the following output:

   ```
   (1, 2, 3)
   (3, 2, 1)
   ```

Add more methods to the List class

The following methods are all object methods in the `List` class, but you might want to write some helper methods in the `Node` class as well.

After you write each method, add code to `main` that tests it. Be sure to test the special cases. Try to write each method in a way that makes it obvious that the method is correct.

Also, for each method, identify any preconditions that are necessary for the method to work and add comments that document them. Your comments should also indicate whether each method is a constructor, function, or modifier. These comments are an important part of the assignment. Take as much care to make the comments correct as you take to make the code correct.

1. Write a method named `addNewFirstNode` that takes an integer and puts it at the beginning of the list.

2. Write a method named `removeFirstNode` that removes the first node from a list and returns it.

3. Write a method named `insertAfter` that takes a node named `new` and a node named `predecessor`, and that inserts the new node into the list immediately after the given predecessor.

4. Write a method named `insertBefore` that takes a node named `new` and a node named `successor`, and that inserts the new node into the list immediately before the given successor.

5. Write a method called `copy` that returns a new list that contains all the same cargo as the original list, but it should contain all new `Node` objects. You might want to write a helper method in `Node` to copy the nodes and link up the copies. You might find it natural to write this method recursively.
6. Write a method called `reverse` that reverses the order of the nodes in a list.
7. Write a method named `appendNode` that takes a node and adds it to the end of the list.
8. Write a method named `appendList` that takes a List and appends a *copy* of the nodes from the list onto `this`. You should be able to reuse some code for this one.