11-0: More on linked lists

Recall that a linked list can be thought of as a chain of ListItems.
- Each ListItem points to the next member in the list.
- The LinkedList class keeps a pointer to the head of the list.

11-1: ListItems

```java
public class ListItem {
    public String data;
    public ListItem next;
    public ListItem(String d) {
        data = d;
        next = null;
    }
}
```

11-2: LinkedList

```java
public class LinkedList {
    public ListItem head = null;
    public void insert(String o) {
        ...}
    public void insertAt(String o, int index) {
        ...}
    public Object get(int index) {
        ...}
    public Object remove(int index) {
        ...}
    public int find(String o) {
        ...}
}
```

11-3: Adding elements

To add an element at the front, we point the new element’s “next” pointer to whatever head is pointing to, then point head to point to the new element.

```java
public void insert(String o) {
    ListItem l = new ListItem(o);
    l.next = head;
    head = l;
}
```

11-4: Exercise

So how do we iterate through a list and print out all the ListItems?
11-5: Exercise

So how do we iterate through a list and create a string representing all the list contents? ListItems?

```java
public String toString() {
    String result = "";
    ListItem current = head;
    while (current != null) {
        result += current.data;
        current = current.next;
    }
    return result;
}
```

11-6: Adding elements

What if we want to add an element in the middle of the list?

First, we find where to put it, then we insert as if this was the front.

11-7: Adding elements by position

The easiest way to add is by position.

If we want an element to be in the nth position, we find the n-1th element, and insert after that.

```java
public void insertAt(String o, int index) {
    ListItem newItem = new ListItem(o);
    ListItem current = head;
    for (int i = 0; i < index; i++) {
        current = current.next;
    }
    newItem.next = current.next;
    current.next = newItem;
}
```

11-8: Adding elements by position

Is there any way that this code could fail?

11-9: Adding elements by position

Is there any way that this code could fail?

What if index is larger than the number of elements in the list?

Need to also make sure current is not null.

```java
for (int i = 0; i < n && current != null; i++){
    ...
}
```
11-11: Adding by order

- What if I want to add based on order?
  - Say we want to keep the list alphabetized?
- Look through until I find the right place.
- But: I need to be careful - I can’t “back up” in the list.

11-12: Adding by order

```java
public void insertAlpha(String name) {
    ListItem newItem = new ListItem(name);
    ListItem current = head;
    if (newItem.data.compareTo(current.data) <= 0) {
        insert(name);
    }
    while (current.next.data.compareTo(newItem.data) < 0) {
        current = current.next;
    }
}
```

11-13: Adding by order

- Wrinkles:
  - What about running off the end of the list?
  - What if the list is empty?
  - What if there’s one element in the list?

11-14: Removing an element

- Removing is similar to finding by order.
- Conceptually, we need to link the item before the one to be removed to the item after the one to be removed.
- Remember: we can only go in one direction!

```java
public String remove(String name) {
    ListItem current = head;
    if (current.data.equals(name)) {
        head = head.next;
        return current.data;
    } else {
        while ((current.next != null) && !(current.next.data.equals(name))) {
            current = current.next;
        }
        String rval = current.next.data;
        current.next = current.next.next;
        return rval;
    }
}
```

11-15: Removing an element

11-16: Exercise

- Write a ‘main’ function that can insert people’s names.
- Add a ‘findBobs’ method that will iterate over your list and find all people named Bob.
- Modify insertAlpha to insert names in reverse alphabetical order.
- Write a method called removeAt(int index) that removes the element at position index.