

CSE 143

Computer Programming II

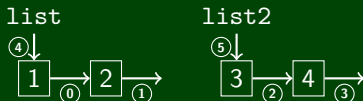
Linked Lists I



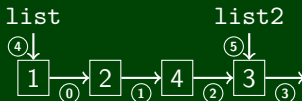
Outline

- 1 Get more familiar with `ListNode`
- 2 Learn how to run through the values of a `LinkedList`
- 3 Learn how `LinkedList` is implemented
- 4 Learn about the different cases to deal with for `LinkedLists`

Before:



After:



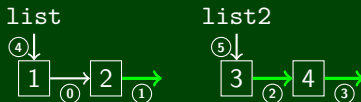
How many ListNodes are there in the before picture?

There are FOUR. Each box is a ListNode.

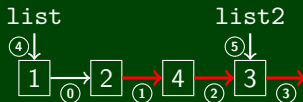
How many references to ListNodes are there?

There are SIX. Every arrow is a reference to a ListNode.

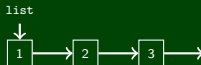
Before:



After:



```
1 list.next.next = list2.next
2 list2.next.next = list2;
3 list2.next = null;
```



Printing a LinkedList Manually

```
1 System.out.println(list.data);  
2 System.out.println(list.next.data);  
3 System.out.println(list.next.next.data);
```

Now, note that we can use a variable to keep track of where we are:

1	<code>System.out.println(list.data);</code>	
2	<code>list = list.next;</code>	
3	<code>System.out.println(list.data);</code>	
4	<code>list = list.next;</code>	
5	<code>System.out.println(list.data);</code>	
6	<code>list = list.next;</code>	

What if our list has 1000 nodes? That would be horrible to write.



Printing a **BIG** LinkedList

```
1 while (list != null) {  
2     System.out.println(list.data);  
3     list = list.next;  
4 }
```

But that destroys the list; so, use a temporary variable instead:

Printing a **BIG** LinkedList Correctly

```
1 ListNode current = list  
2 while (current != null) {  
3     System.out.println(current.data);  
4     current = current.next;  
5 }
```

We can use for loops in a similar way to with ArrayLists to run through LinkedLists!

Traversing an ArrayList

```
for (int i = 0; i < arrayList.size(); i++) {
    System.out.println(arrayList.get(i));
}
```

Traversing an LinkedList

```
for (ListNode current = linkedList; current != null; current = current.next) {
    System.out.println(current.data);
}
```

Description	ArrayList Code	LinkedList Code
Go to front of list	int i = 0;	ListNode current = list;
Test for more elements	i < list.size()	current != null
Current value	list.get(i)	current.data
Go to next element	i++;	current = current.next;

- No generics (only stores ints)
- Fewer methods: `add(value)`, `add(index, value)`, `get(index)`, `set(index, value)`, `size()`, `isEmpty()`, `remove(index)`, `indexOf(value)`, `contains(value)`, `toString()`
- This is the same idea as when we implemented `ArrayIntList`!

What fields does our `LinkedList` need?

A reference to the front of the list



`LinkedList v1`

```

1 public class LinkedList {
2     private ListNode front;
3
4     public LinkedList() {
5
6         front = null;
7     }
8     ...
9 }
    
```

front
↓

Buggy toString()

```
public String toString() {
    String result = "[";

    ListNode current = this.front;
    while (current != null) {
        result += current.data + ", ";
        current = current.next;
    }

    return result + "]";
}
```

Our toString() puts a trailing comma. Fix it by stopping one early:

Fixed toString()

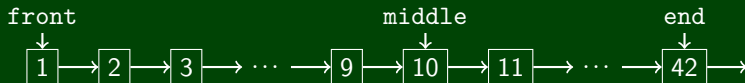
```
public String toString() {
    String result = "[";

    ListNode current = this.front;
    while (current != null && current.next != null) {
        result += current.data + ", ";
        current = current.next;
    }
    if (current != null) {
        result += current.data;
    }

    return result + "]";
}
```

Writing a LinkedList Method

- 1 Identify cases to consider...
 - Front/Empty
 - Middle
 - End
- 2 Draw pictures for each case
- 3 Write each case separately



Cases to consider:

- Add to empty list
- Add to non-empty list

Add To An Empty List

What does an empty list look like?

front



```

1 public void add(int value) {
2     /* If the list is empty... */
3     if (this.front == null) {
4         this.front = new ListNode(value);
5
6     }
7     /* Other Cases ... */
8 }
    
```

front



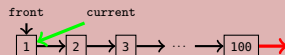
value →

Add To A Non-Empty List

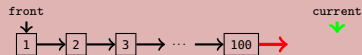
Consider a non-empty list:



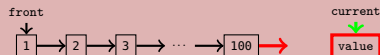
```
1  /* Idea: We want to change the red arrow.
2     Loop until we're at the last node. */
3  ListNode current = this.front;
```



```
4
5  while (current != null) {
6      current = current.next;
7  }
```



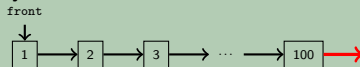
```
8
9  current = new ListNode(value);
```



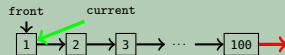
10

Add To A Non-Empty List (Fixed)

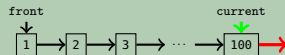
Consider a non-empty list:



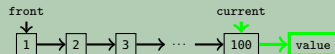
```
1 /* Idea: We want to change the red arrow.
2    Loop until we're at the node before the last node */
3 ListNode current = this.front;
```



```
4
5 while (current.next != null) {
6     current = current.next;
7 }
```

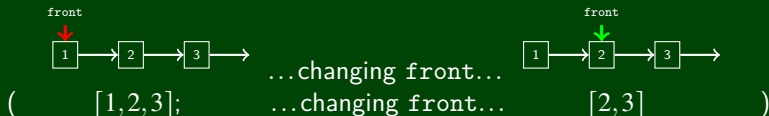


```
8
9 current.next = new ListNode(value);
```

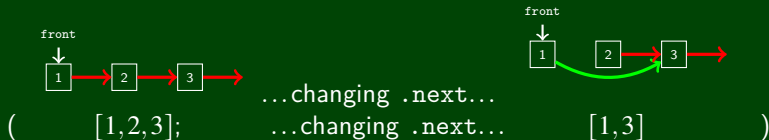


There are only two ways to modify a LinkedList:

- Change front



- Change `current.next` for some `ListNode`, `current`



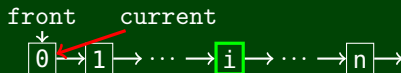
Setting “current” does NOTHING!


```

1 // pre: 0 <= index < size
2 // post: Returns the value in the list at index
3 public int get(int index) {
4     ListNode current = front;

```

5

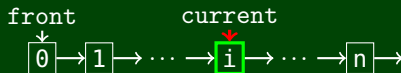


```

6     for (int i = 0; i < index; i++) {
7         current = current.next;
8     }

```

9



```

10    return current.data;
11 }

```

- Be able to deal with before-and-after `ListNode` pictures
- Know how to loop through a `LinkedList`
 - Use a `while` loop.
 - Don't forget to create a `ListNode current` variable so we don't destroy the original list.
 - Don't forget to update the `current` variable.
- Understand differences and similarities between `ArrayList` and `LinkedList`
 - They both have the same functionality (`add`, `remove`, etc.)
 - But they're **implemented** differently (`array` vs. `ListNodes`)
- With `LinkedLists`, you often have to stop **one node before the one you want**.
- DO NOT start coding `LinkedList` problems without drawing pictures first.