Adam Blank **Lecture 7** **Winter 2015**

CSE 143

Computer Programming II

Linked Lists I

Outline

[0]

1 Get more familiar with LinkedNodes

2 Learn how to run through the values of a LinkedList

3 Learn how LinkedIntList is implemented

4 Learn about the different cases to deal with for LinkedLists

Does That Make Sense?

Quick Note: When I say "does that make sense?"...

- If it does make sense, yell "yes"
- Otherwise, say nothing.

Outline

[0]

1 Get more familiar with LinkedNodes

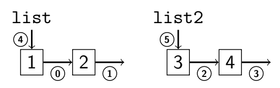
2 Learn how to run through the values of a LinkedList

3 Learn how LinkedIntList is implemented

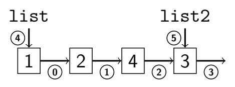
4 Learn about the different cases to deal with for LinkedLists

Another LinkedNode Example

Before:



After:



How many LinkedNodes are there in the before picture?

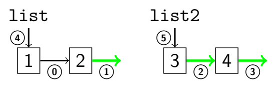
There are FOUR. Each box is a LinkedNode.

How many references to LinkedNodes are there?

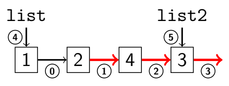
There are SIX. Every arrow is a reference to a LinkedNode.

Another `LinkedNode` Example (Solution)

Before:



After:

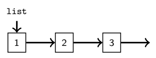


```
list.next.next = list2.next
list2.next.next = list2;
list2.next = null;
```

Outline

[0]

Printing a `LinkedList`

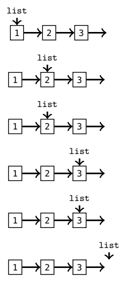


Printing a `LinkedList` Manually

```
System.out.println(list.data);
System.out.println(list.next.data);
System.out.println(list.next.next.data);
```

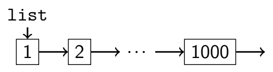
Now, note that we can use a variable to keep track of where we are:

```
System.out.println(list.data);

list = list.next;

System.out.println(list.data);

list = list.next;

System.out.println(list.data);

list = list.next;
```



Printing a `LinkedList`: Better Version

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



Printing a BIG `LinkedList`

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

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

Printing a BIG `LinkedList` Correctly

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

`LinkedList` vs. `ArrayList`

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;`

Outline

[0]

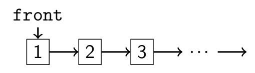
LinkedIntList

- 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!

LinkedIntList Fields

What fields does our LinkedIntList need?

A reference to the front of the list



LinkedIntList v1

```java
public class LinkedIntList {
   private ListNode front;

   public LinkedIntList() {
      front = null;
   }

   ...
}
```

LinkedIntList toString()

Buggy toString()

```java
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()

```java
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 + "]";
}
```

Outline

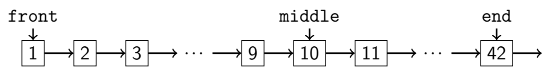
[0]

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

Modifying LinkedLists

Writing a LinkedList Method

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



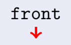
LinkedIntList add() (Empty Case)

Cases to consider:

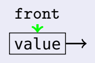
- Add to empty list
- Add to non-empty list

Add To An Empty List

What does an empty list look like?



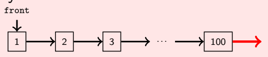
```java
public void add(int value) {
   /* If the list is empty... */
   if (this.front == null) {
      this.front = new ListNode(value);
   }
   /* Other Cases ... */
}
```



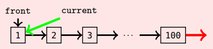
LinkedIntList add() (Non-empty Case)

Add To A Non-Empty List

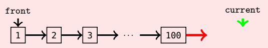
Consider a non-empty list:



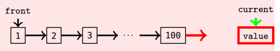
```
/* Idea: We want to change the red arrow.
         Loop until we're at the last node. */
ListNode current = this.front;
```



```
while (current != null) {
   current = current.next;
}
```



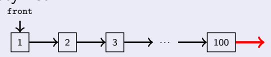
```
current = new ListNode(value);
```



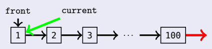
LinkedIntList add() (Non-empty Case)

Add To A Non-Empty List (Fixed)

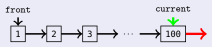
Consider a non-empty list:



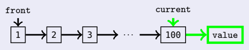
```
/* Idea: We want to change the red arrow.
         Loop until we're at the node before the last node */
ListNode current = this.front;
```



```
while (current.next != null) {
   current = current.next;
}
```



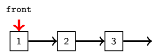
```
current.next = new ListNode(value);
```



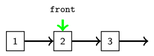
Working with LinkedLists

There are only two ways to modify a LinkedList:

- Change front

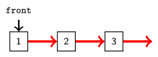


...changing front...

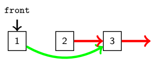


([1,2,3]; ...changing front... [2,3])

- Change current.next for some ListNode, current



...changing .next...

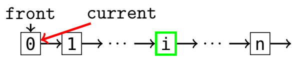


([1,2,3]; ...changing .next... [1,3])

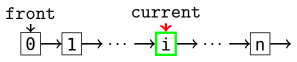
Setting "current" does NOTHING!

LinkedIntList get()

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



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



```
   return current.data;
}
```

Some LinkedList Tips!

- Be able to deal with before-and-after LinkedNode 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 similiarities 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.