Linked node problem 3

• What set of statements turns this picture:

```
data  next
10    
```

```
data  next
30    
data  next
40    
```

list1

list2

• Into this?

```
data  next
10    
```

```
data  next
20    
```

```
data  next
30    
```

```
data  next
40    
```

list1

list2
Linked node problem 3

- How many ListNode variables?

- Which variables change?
Linked node question

- Suppose we have a long chain of list nodes:

```
<table>
<thead>
<tr>
<th>data</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>990</td>
<td></td>
</tr>
</tbody>
</table>
```

- We don't know exactly how long the chain is.

- How would we print the data values in all the nodes?
Algorithm pseudocode

Start at the **front** of the list.
While (there are more nodes to print):
    Print the current node's **data**.
    Go to the **next** node.

- How do we walk through the nodes of the list?

```python
list = list.next;  // is this a good idea?
```
Traversing a list?

- One (bad) way to print every value in the list:

  ```java
  while (list != null) {
    System.out.println(list.data);
    list = list.next; // move to next node
  }
  ```

- What's wrong with this approach?
  - (It loses the linked list as it prints it!)
A current reference

- Don't change list. Make another variable, and change it.
  - A ListNode variable is NOT a ListNode object

```java
ListNode current = list;
```

- What happens to the picture above when we write:

```java
current = current.next;
```
Traversing a list correctly

- The correct way to print every value in the list:

```java
ListNode current = list;
while (current != null) {
    System.out.println(current.data);
    current = current.next; // move to next node
}
```

- Changing `current` does not damage the list.
Linked List vs. Array

- Print list values:

```java
ListNode list = ...;

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

- Similar to array code:

```java
int[] a = ...;

int i = 0;
while (i < a.length) {
    System.out.println(a[i]);
    i = i + 1;
}
```

### Description

<table>
<thead>
<tr>
<th>Description</th>
<th>Array Code</th>
<th>Linked List Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Go to front of list</td>
<td>int i = 0;</td>
<td>ListNode current = list;</td>
</tr>
<tr>
<td>Test for more elements</td>
<td>i &lt; size</td>
<td>current != null</td>
</tr>
<tr>
<td>Current value</td>
<td>elementData[i]</td>
<td>current.data</td>
</tr>
<tr>
<td>Go to next element</td>
<td>i=i+1;</td>
<td>current = current.next;</td>
</tr>
</tbody>
</table>
A LinkedIntList class

- Let's write a collection class named LinkedIntList.
  - Has the same methods as ArrayIntList:
    - add, add, get, indexOf, remove, size, toString

- The list is internally implemented as a chain of linked nodes
  - The LinkedIntList keeps a reference to its front as a field
  - null is the end of the list; a null front signifies an empty list

```
LinkedIntList

  front

  add(value)
  add(index, value)
  indexOf(value)
  remove(index)
  size()
  toString()

ListNode

  data | next
  42   |

  element 0

ListNode

  data | next
  -3   |

  element 1

ListNode

  data | next
  17   |

  element 2
```
• Suppose our list had the contents

front

<table>
<thead>
<tr>
<th>data</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>data</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>data</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td></td>
</tr>
</tbody>
</table>

• Practice simulating the code we wrote and tell us what the result will look like when we call `list.add(40);`

```java
public void add(int value) {
    ListNode curr = front;
    while (curr != null) {
        curr = curr.next;
    }
    curr = new ListNode(value);
}
```

Options

- [10, 20, 30]
- [10, 20, 40]
- [10, 20, 40, 30]
- [10, 20, 30, 40]
- Error
Before/After

- **Before**

  - Front

  ![Linked List Diagram Before](image)

- **After**

  - Front

  ![Linked List Diagram After](image)
changing a list

- There are only two ways to change a linked list:
  - Change the value of `front` (modify the front of the list)
  - Change the value of `<node>.next` (modify middle or end of list to point somewhere else)

- Implications:
  - To add in the middle, need a reference to the `previous` node
  - Front is often a special case