LinkedList Week*

- **Friday 1/18**
  - Intro to ListNodes and Before/After pictures
- **Tuesday 1/22**
  - Practice with Before/After pictures
- **Wednesday 1/23**
  - Intro to LinkedIntList
  - ListNodes and Loops
- **Thursday 1/24**
  - Practice with ListNodes and loops
- **Friday 1/25**
  - Advanced loops with ListNodes
- **Tuesday 1/29**
  - Practice with advanced loops
Linked node problem 3

• What set of statements turns this picture:

```
<table>
<thead>
<tr>
<th>data</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>30</td>
<td>40</td>
</tr>
</tbody>
</table>
```

• Into this?

```
<table>
<thead>
<tr>
<th>data</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
</tbody>
</table>
```
Linked node problem 3

- How many ListNode variables?

- Which variables change?
References vs. objects

```
variable = value;
```

A *variable* (left side of `=`) place to put a reference
(where the phone number goes; where the base of the arrow goes)
A *value* (right side of `=`) is the reference itself
(the phone number; the destination of the arrow)

For the list at right:

- `a.next = value;`
  means to adjust where `a` points

- `variable = a.next;`
  means to make `variable` point at
Linked node question

- Suppose we have a long chain of list nodes:

```
list    data | next
10      |       |
20      |       |
990     |       |
```

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

```
<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>
</tbody>
</table>
```

```
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

```
ListNode current = list;
```

- What happens to the picture above when we write:

```
current = current.next;
```
list1

<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>
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.
Abstract data types (ADTs)

- **abstract data type (ADT):** A specification of a collection of data and the operations that can be performed on it.
  - Describes *what* a collection does, not *how* it does it

- Java's collection framework describes several ADTs:
  - Queue, List, Collection, Deque, List, Map, Set

- An ADT can be implemented in multiple ways:
  - `ArrayList` and `LinkedList` implement `List`
  - `HashSet` and `TreeSet` implement `Set`
  - `LinkedList`, `ArrayDeque`, etc. implement `Queue`

- The **same** external behavior can be implemented in many different ways, each with pros and cons.
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
public class LinkedIntList {
    private ListNode front;

    public LinkedIntList() {
        front = null;
    }

    methods go here
}

LinkedIntList
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++;
  }
  ```

<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=i+1</td>
<td>current = current.next;</td>
</tr>
</tbody>
</table>
Poll Everywhere Workflow

1. Think (1 minute)
   1. Take **45 seconds** to think *on your own* about the problem
   2. Take **15 seconds** to poll in *by yourself*

2. Pair (2 minutes) [TAs will walk around]
   1. Take **1.5 minutes** to *talk with your neighbors* about the problem and compare how you answered
      - If you and your neighbors agree, try to figure out why the other answers might be wrong
      - If you and your neighbors disagree, talk about the material to figure out who is right!
   2. Take **30 seconds** to finish discussion and poll in with your new final answer

3. Share (2 minutes)
   1. Talk as a class about what people were answering in and why
Suppose our list had the contents

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

```java
class ListNode {
    int data;
    ListNode next;
}

class List {
    ListNode front;

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

• Before

front

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

• After

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>

<table>
<thead>
<tr>
<th>data</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td></td>
</tr>
</tbody>
</table>
Implementing `add`

```java
// Adds the given value to the end of the list.
public void add(int value) {
    ...
}
```

- How do we add a new node to the end of a list?
- Does it matter what the list's contents are before the add?
Adding to an empty list

- Before adding 20:
  
  \[
  \begin{array}{l}
  \text{front} = \text{element 0} \\
  \end{array}
  \]

- After:
  
  \[
  \begin{array}{l}
  \text{front} = \begin{array}{c}
  \text{data} \\
  \text{next} \\
  20 \\
  \end{array} \\
  \text{element 0} \\
  \end{array}
  \]

- We must create a new node and attach it to the list.
The `add` method, 1st try

// Adds the given value to the end of the list.
public void add(int value) {
    if (front == null) {
        // adding to an empty list
        front = new ListNode(value);
    } else {
        // adding to the end of an existing list
        ...
    }
}
Adding to non-empty list

- Before adding value 20 to end of list:

  - element 0:
    - front = 42
    - data: 42
    - next:
  - element 1:
    - data: -3
    - next:

- After:

  - element 0:
    - front = 42
    - data: 42
    - next:
  - element 1:
    - data: -3
    - next:
  - element 2:
    - data: 20
    - next:
Don't fall off the edge!

- To add/remove from a list, you must modify the next reference of the node before the place you want to change.

- Where should current be pointing, to add 20 at the end?
- What loop test will stop us at this place in the list?
The `add` method

// Adds the given value to the end of the list.
public void add(int value) {
    if (front == null) {
        // adding to an empty list
        front = new ListNode(value);
    } else {
        // adding to the end of an existing list
        ListNode current = front;
        while (current.next != null) {
            current = current.next;
        }
        current.next = new ListNode(value);
    }
}
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
Implementing `get`

// Returns value in list at given index.
public int get(int index) {
    ...
}

- Exercise: Implement the `get` method.
The get method

// Returns value in list at given index.
// Precondition: 0 <= index < size()
public int get(int index) {
    ListNode current = front;
    for (int i = 0; i < index; i++) {
        current = current.next;
    }
    return current.data;
}
Implementing `add (2)`

// Inserts the given value at the given index.
public void add(int index, int value) {
    ...
}

- Exercise: Implement the two-parameter `add` method.

```
front = [42]
```

```
<table>
<thead>
<tr>
<th>data</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td></td>
</tr>
</tbody>
</table>

```

```
<table>
<thead>
<tr>
<th>data</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td>-3</td>
<td></td>
</tr>
</tbody>
</table>

```

```
<table>
<thead>
<tr>
<th>data</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td></td>
</tr>
</tbody>
</table>
```

```
element 0  element 1  element 2
```
The **add** method (2)

// Inserts the given value at the given index.
// Precondition: 0 <= index <= size()
public void add(int index, int value) {
    if (index == 0) {
        // adding to an empty list
        front = new ListNode(value, front);
    } else {
        // inserting into an existing list
        ListNode current = front;
        for (int i = 0; i < index - 1; i++) {
            current = current.next;
        }
        current.next = new ListNode(value,
                                      current.next);
    }
}