CSE 143
Lecture 7

Linked List Basics

reading: 16.1 - 16.2

slides adapted from Marty Stepp and Hélène Martin
http://www.cs.washington.edu/143/
Suppose we have a long chain of list nodes:

- 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?

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

![Diagram showing list traversal](image-url)
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 that.
  - A ListNode variable is NOT a ListNode object

ListNode current = list;

- What happens to the picture above when we write:

  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.

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

```
<table>
<thead>
<tr>
<th>data</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td>990</td>
<td></td>
</tr>
</tbody>
</table>
```
• Algorithm to print list values:

```
ListNode front = ...;

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

• Similar to array code:

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

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

- 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:

Before:

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

- data: -3
  - next: data

After:

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

- data: -3
  - next: data

- data: 20
  - next: data

• After:
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);
    }
}
Implementing the `get` method.

```java
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.
/\ 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);
    }
}