CSE 143
Lecture 10

Linked List Basics

reading: 16.1 - 16.2

slides created by Marty Stepp
http://www.cs.washington.edu/143/
variable = value;

a variable (left side of =) is an arrow (the base of an arrow)
a value (right side of =) is an object (a box; what an arrow points at)

- a.next = value;
  means to adjust where 1 points

- variable = a.next;
  means to make variable point at 2
Reassigning references

• when you say:
  
  \[ \text{a.next} = \text{b.next}; \]

• you are saying:
  
  – "Make the \textit{variable} \texttt{a.next} refer to the same \textit{value} as \texttt{b.next}."  
  – Or, "Make \texttt{a.next} point to the same place that \texttt{b.next} points."
Linked node question

• Suppose we have a long chain of list nodes:

```
list → data | next
  → 10      → data | next
  → 20      → ...    → data | next
  →         → 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?

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

- list = [10, next]
- [20, next]
- ...
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

```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

• Algorithm to print list values:

```java
ListNode front = ...;

ListNode current = front;
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++;
}
```
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

}
Implementing `add`

// 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?

```java
data  next
data  next
data  next
42    
   -3
   17
```
Adding to an empty list

• Before adding 20:

  - We must create a new node and attach it to the list.

  \[
  \text{front} = \begin{array}{c}
\end{array}
  \]

  \[
  \begin{array}{c}
  \text{next} \\
  \text{data} \\
  \text{next}
  \end{array}
  \]

  \[
  \begin{array}{c}
  \text{element 0}
  \end{array}
  \]

  \[
  \begin{array}{c}
  \text{front} = \begin{array}{c}
  \end{array}
  \]

  \[
  \begin{array}{c}
  \text{next}
  \end{array}
  \]

  \[
  \begin{array}{c}
  \text{data} \\
  \text{next}
  \end{array}
  \]

  \[
  \begin{array}{c}
  \text{element 0}
  \end{array}
  \]

• After:

  \[
  \text{front} = \begin{array}{c}
  \end{array}
  \]

  \[
  \begin{array}{c}
  \text{next}
  \end{array}
  \]

  \[
  \begin{array}{c}
  \text{data} \\
  \text{next}
  \end{array}
  \]

  \[
  \begin{array}{c}
  \text{element 0}
  \end{array}
  \]
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:

- front = 

  front
  \[ \begin{array}{c}
      \text{data} \\
      42
  \end{array} \]

  \[ \begin{array}{c}
      \text{next} \\
      -3
  \end{array} \]

  element 0

  element 1

• After:

- front = 

  front
  \[ \begin{array}{c}
      \text{data} \\
      42
  \end{array} \]

  \[ \begin{array}{c}
      \text{next} \\
      -3
  \end{array} \]

  element 0

  element 1

  element 2

  \[ \begin{array}{c}
      \text{data} \\
      20
  \end{array} \]
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 `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.
// 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);
    }
}