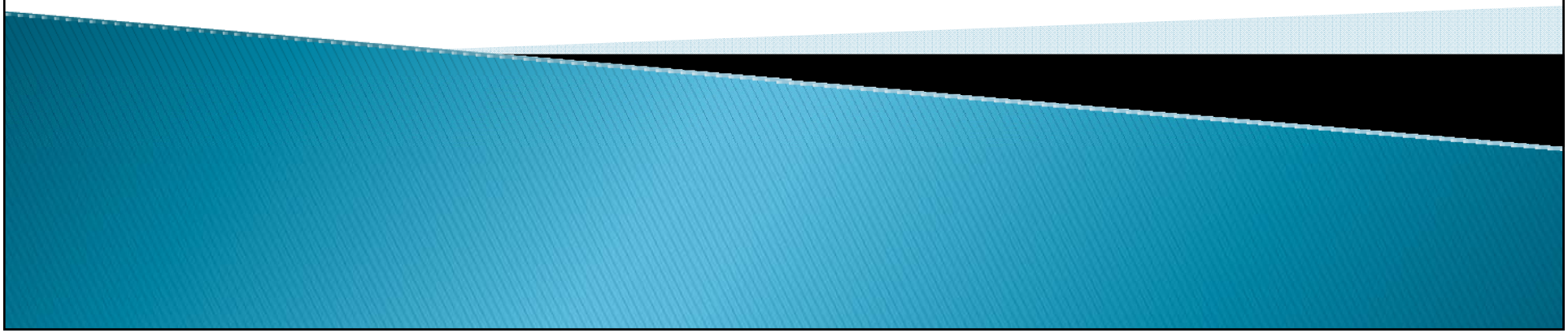
CSSE 220 Day 22

LinkedList Implementation
Recursion



CSSE 220 Day 22

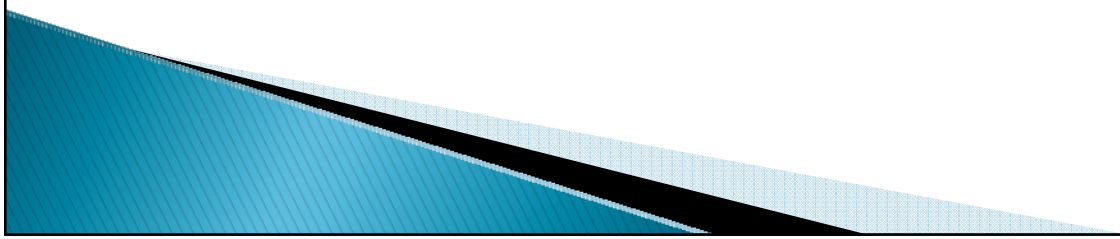
- Turn in your written problems
- Mini-project Partner Survey: Do it by 4:00 today
- Reminder: Exam #2 is this Friday
  - Can start 7:15.
  - You may bring one piece of paper with handwritten notes for the first part.
  - Same resources as last time for the programming part.
- Markov Milestone 2 due Friday, 5:00pm
- Take the Markov Justification quiz on ANGEL now (5 minutes)

Mini-project

- Will be done by teams of 3, Weeks 9-10
- I will pick teams, based on performance of students in the class so far.
  - Rationale for putting people with similar performance together
- There is a survey on ANGEL that lets you tell me the names of up to two people whom you'd prefer NOT to work with.
- Project will be a spell-checker and suggester
- Other projects have been highly-specified. For this one, you have a lot of leeway and can be very creative.

SpellChecker and Suggester

- GUI-based program
- Check the words of a text file for spelling
  - User can browse to file
- Flag words that are not in program's dictionary
- Suggest possible alternate spellings
  - Think of ways misspelling can occur:
    - missing or added letters
    - transposed letters
    - no space between words
    - things you come up with
- An interface that allows user to correct the spelling.
  - change, ignore, ignore all, …



SpellChecker and Suggester

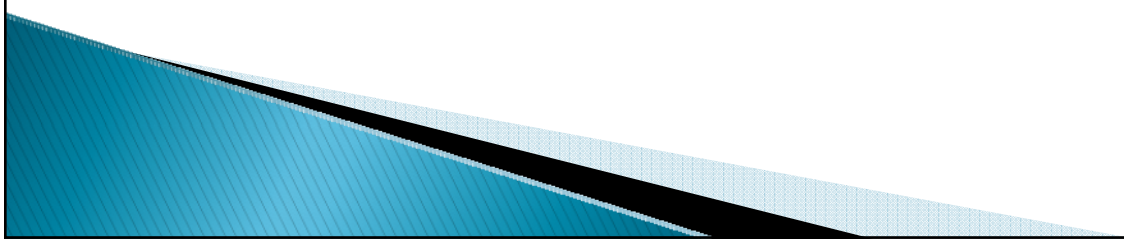
- Some GUI things you'll want to learn how to do
  - Browse to a file and open it
  - Deal with text in a text box
  - Display a list of choices and get user selection
- Some things you can do before Tuesday's kick-off.
  - Look for a dictionary to use (share it!)
  - Look at user interfaces of some spell-checkers
  - Look up various Java classes that may be useful
    - Especially helpful: The Java Swing book from Safari Tech Books online (see course syllabus)

Mini-project timetable

- Now. Look for a dictionary, think about the kinds of spelling errors you want to detect/correct.
- Day 25. Begin working with your partners.
- Day 27. Demonstrate some progress in class.
- Day 30. Final submission of the project is due.

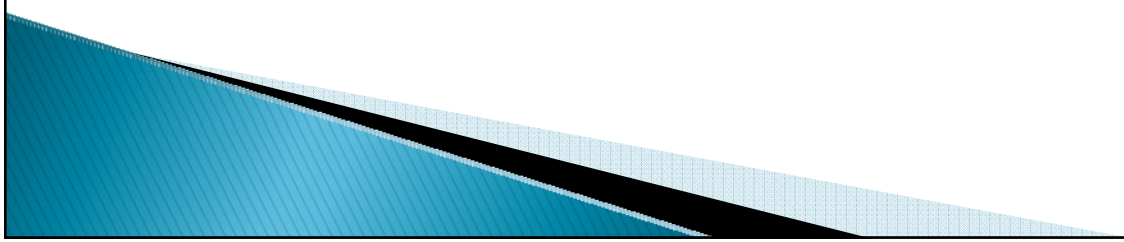
Answers to your questions

- Abstract Data Types and Data Structures
- Collections and Lists
- Markov
- Friday's exam
- Material you have read
- Anything else

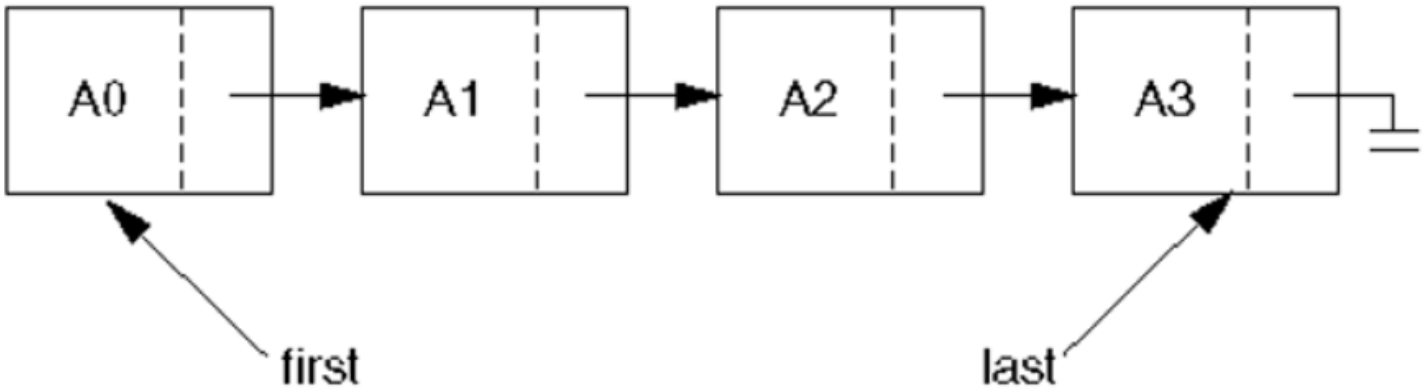


Today's agenda

- LinkedList Implementation part 2
- Recursion



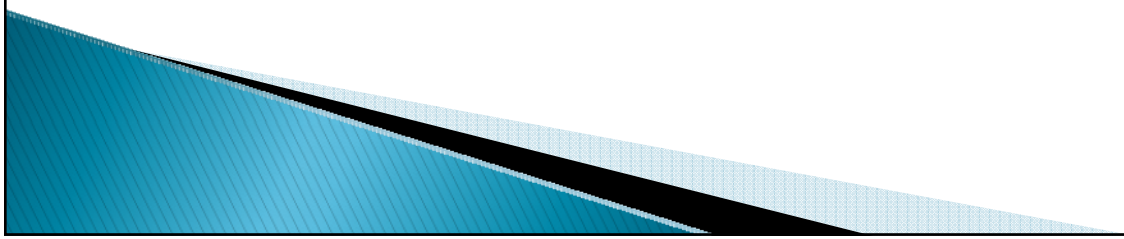
`LinkedList` implementation of the `List` Interface



- Stores items (non-contiguously) in nodes; each contains a reference to the next node.
- Lookup by index is linear time (worst, average).
- Insertion or removal is constant time once we have found the location.
  - show how to insert A4 after A1.
- If `Comparable` list items are kept in sorted order, finding an item still takes linear time.

Too many special cases

- What is the main cause?
  - All nodes of the linked list are pointed to by the **next** field of the previous ListNode …
  - … except the first node, which is pointed to by the **first** field of the LinkedList object.
- One solution:
  - Add an extra node at the beginning of the list
  - The "header" node.
  - So a list of n items is represented by n+1 nodes.
  - The first element of the list is in the second node.



List with Header Node

figure 17.4

Using a header node for the linked list

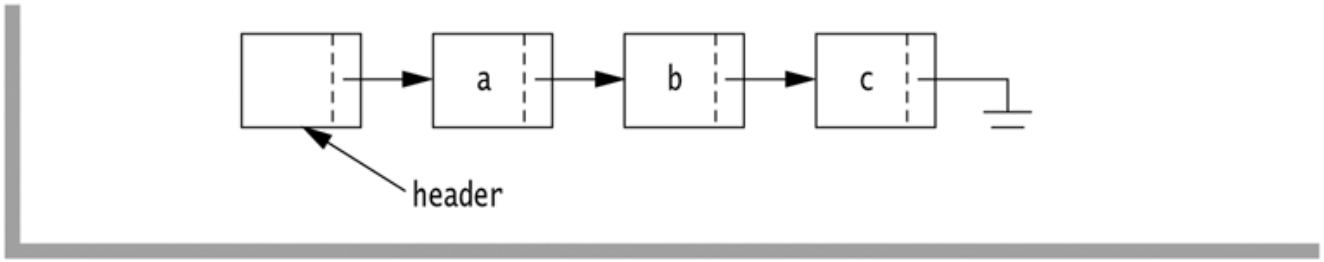
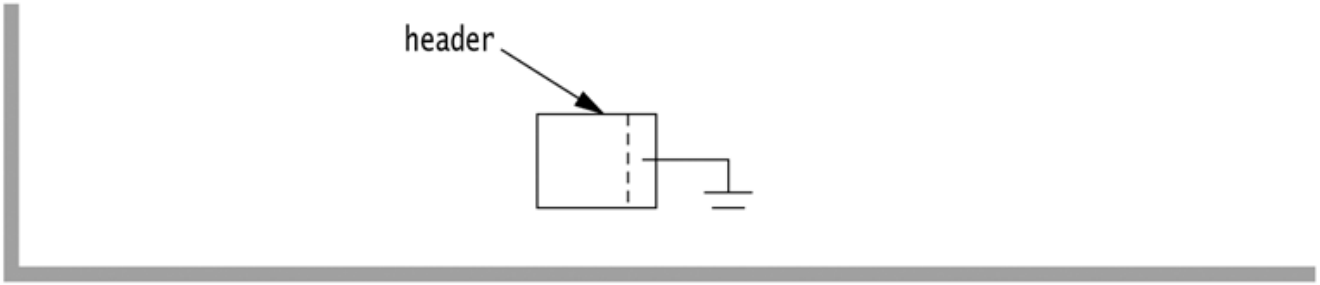


figure 17.5

Empty list when a header node is used



- Change the code to include this node.
- last should point to the last node.
- Write **remove**.

Let's do parts of a `LinkedList` implementation

```
class LinkedList implements List {
  ListNode first;
  ListNode last;
```

Constructors: (a) default (b) single element.

methods:

Attempt these in the order shown here.

public **`boolean add(Object o)`**

Appends the specified element to the end of this list (returns `true`)

`public int size()` Returns the number of elements in this list.

`public void add(int i, Object o)` adds o at index i.

`  throws IndexOutOfBoundsException`

`public boolean contains(Object o)`

Returns true if this list contains the specified element. (2 versions).

`public boolean remove(Object o)`

Removes the first occurrence (in this list) of the specified element.

`public Iterator iterator()` **Can we also write listIterator( ) ?**

Returns an iterator over the elements in this list in proper sequence.

Consider parts of a `LinkedList` implementation

```
class ListNode{
 Object element; // contents of this node
 ListNode next;  // link to next node

 ListNode (Object element,
           ListNode next) {
   this.element = element;
   this.next = next;
 }

 ListNode (Object element) {
   this(element, null);
 }
 ListNode () {
   this(null);
 }
}
```

How to implement LinkedList?

fields?

Constructors?

Methods?

Doubly-linked list

- Each node has two pointers, **prev** and **next.**
- There is one other new node, **tail**, whose **prev** pointer points to the node containing the last element of the list.
- This makes `remove()` easier to write
  - and it also makes an efficient **ListIterator** possible.

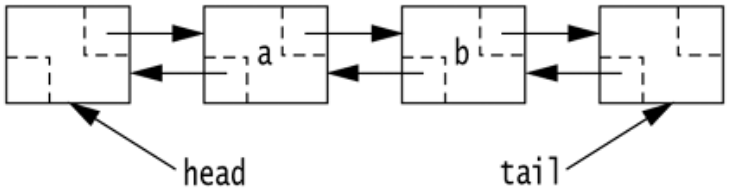
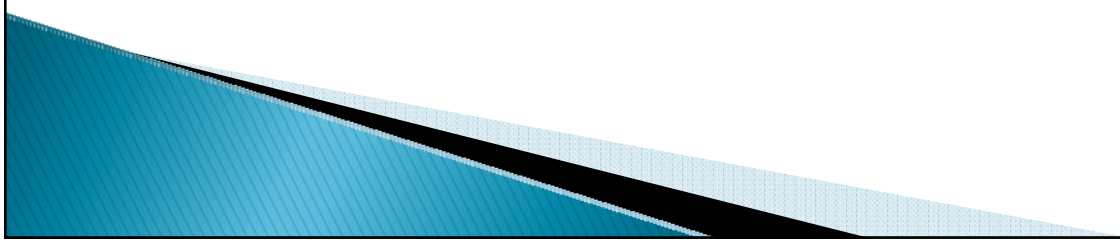


figure 17.15
A doubly linked list

Recursion

- What is a recursive method?
  - A method that calls itself, but on a simpler problem
- Used for any situation where parts of a whole look like mini versions of the whole:
  - Folders within folders on computers
  - Some computer languages (Scheme)
  - Trees in general
- Cons: Takes more space (but time can be roughly equal)
- Pros: Can gives code that's very easy to understand



Recursion template

- For a method that calculates a value:

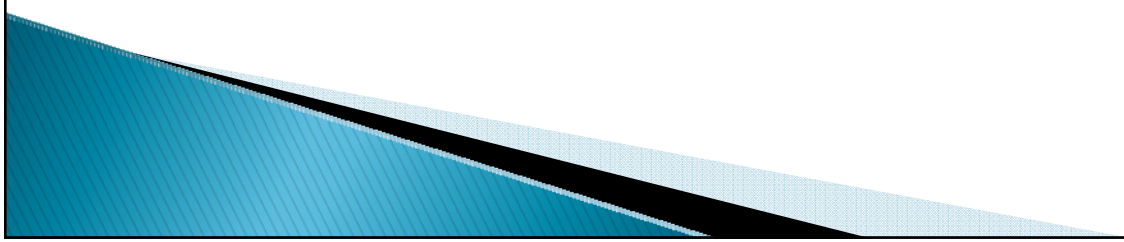
```
int foo(int n) {
  if (n <=1) {      //Base case
    return (some easy expression);
  } else {
    return (some expr. with foo(n-1); //not just
      foo(n)) so progress
  }
```

- Of course, you can write void recursive methods, and ones that recurse on values other than n-1

- Example we've seen: factorial. Look at debugger.

Weiss' Four Rules of Recursion

1. **Base case**
   - You need at least 1 base case that can be solved without recursing
2. **Progress**
   - You can only recurse on a simpler problem
3. **"You gotta believe"**
   - Otherwise, you'll try to solve the problem both recursively and non-recursively. This is bad.
4. **Compound interest rule**
   - Efficiency: Don't duplicate work by solving the same instance of the problem in separate recursive calls
   - Later

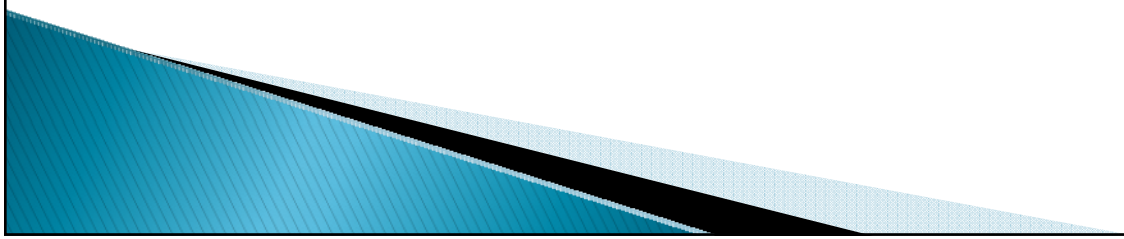


Break

If you don't have a base case for your recursion, it can become a nightmare!

Another Example

- Euclid's algorithm for calculating gcd(a,b)
- gcd(a,b): //assumes a > b
  - if a is a multiple of b, return b
  - Otherwise, return gcd(b, a % b) (guaranteed to be smaller)
- Check it out now
- Watch it in the debugger.



What else?

- Could you write a recursive size() method for linked lists?

- Helper function:

```
int size() {
  if (this.header.next == null) return 0;
  return this.header.next.size();
}
```

- We now need to write the ListNode's size function.