CSSE 220 Day 22

LinkedList Implementation

Mini-project intro

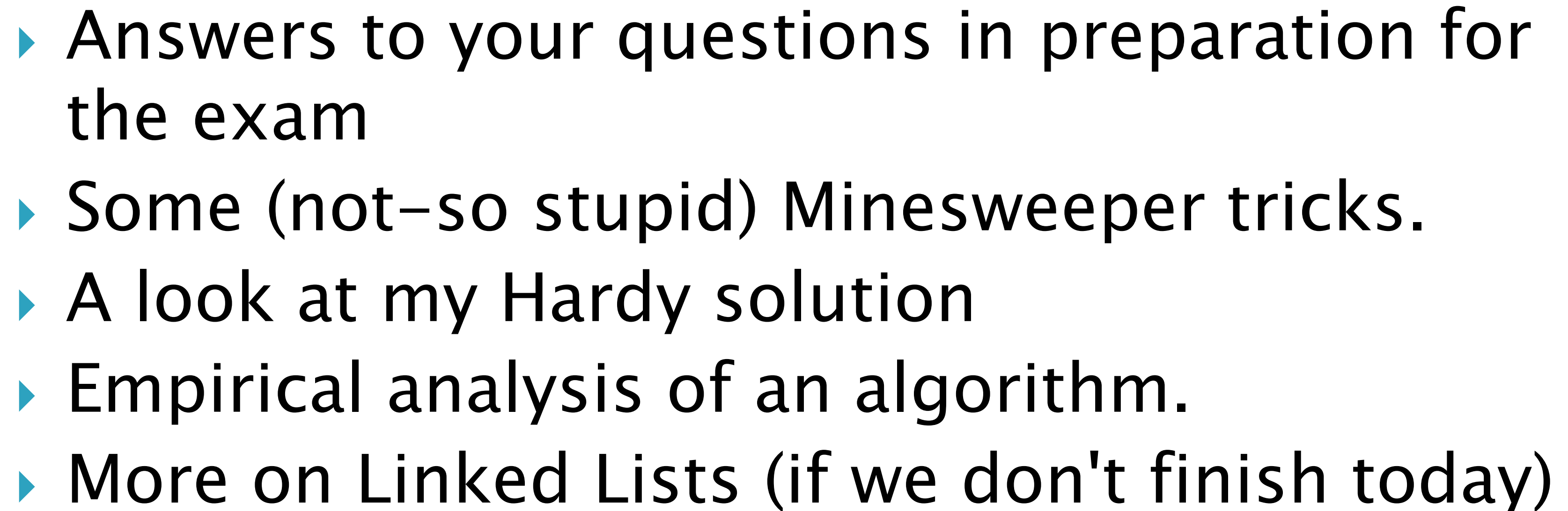
CSSE 220 Day 22

- Turn in your written problems
- Mini-project Partner Survey: Do it by 4:00 today
- Reminder: Exam #2 is this Thursday
  - In order to reduce time pressure, you optionally may take the non-programming part 7:10-7:50 AM.
  - You may bring one piece of paper with notes for the first part.
  - Same resources as last time for the programming part.
- Markov Milestone 2 due Friday
  - http://svn.cs.rose-hulman.edu/repos/220-200820-markovXX
    - where XX is your 2-digit team number.
- Take the Markov Justification quiz on ANGEL now (5 minutes)

```
A. No man is justified in doing evil on the ground of expediency. Theodore Roosevelt
B. No man is justified in doing evil on the ground of expediency.   Theodore Roosevelt
C. No  man  is  justified  in  doing  evil on the ground of expediency.  Theodore Roosevelt
D. No man is justified in doing evil on the ground of expediency.  Theodore Roosevelt
E. No man is justified  in doing evil on   the ground of expediency.  Theodore Roosevelt
F. No  man  is  justified  in  doing   evil  on  the  ground  of  expediency.  Theodore  Roosevelt
G. No man is justified      in doing evil on the ground of expediency.  Theodore Roosevelt
```

A. You must have two spaces after a period
B. It is okay to add a one additional space after the period, for a total of three spaces
C. The places to insert extra spaces are supposed to be randomly chosen, not all placed at the beginning of the line
D: This is the way ist should look if the line length is such that we do not need to add any spaces at all.
E. You must add one space in every location before you are allowed to add a second space anywhere.
F.You must add one space in every location before you are allowed to add a second space anywhere. No additional spaces have been added after the period, but 2 were added after "doing".
G: A blatant case of adding too many spaces in one place.

Tomorrow



- Answers to your questions in preparation for the exam
- Some (not-so stupid) Minesweeper tricks.
- A look at my Hardy solution
- Empirical analysis of an algorithm.
- More on Linked Lists (if we don't finish today)

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.

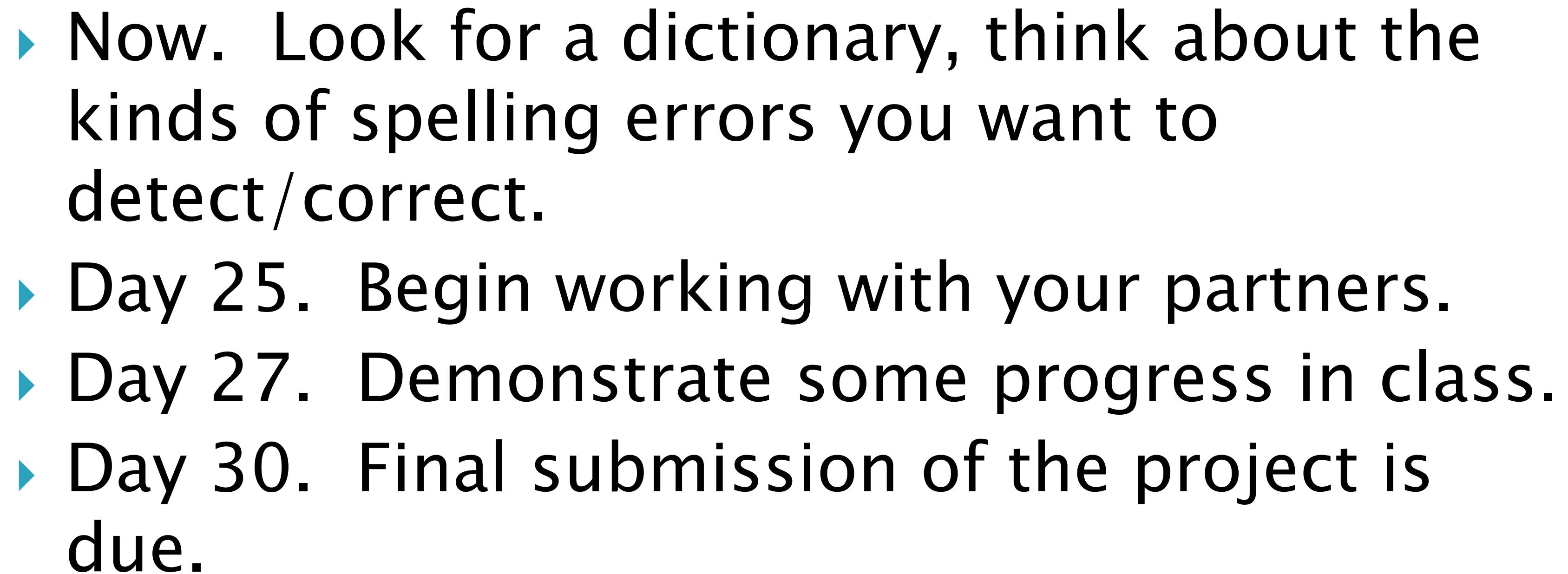
SpellChecker and Suggester

- 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, …

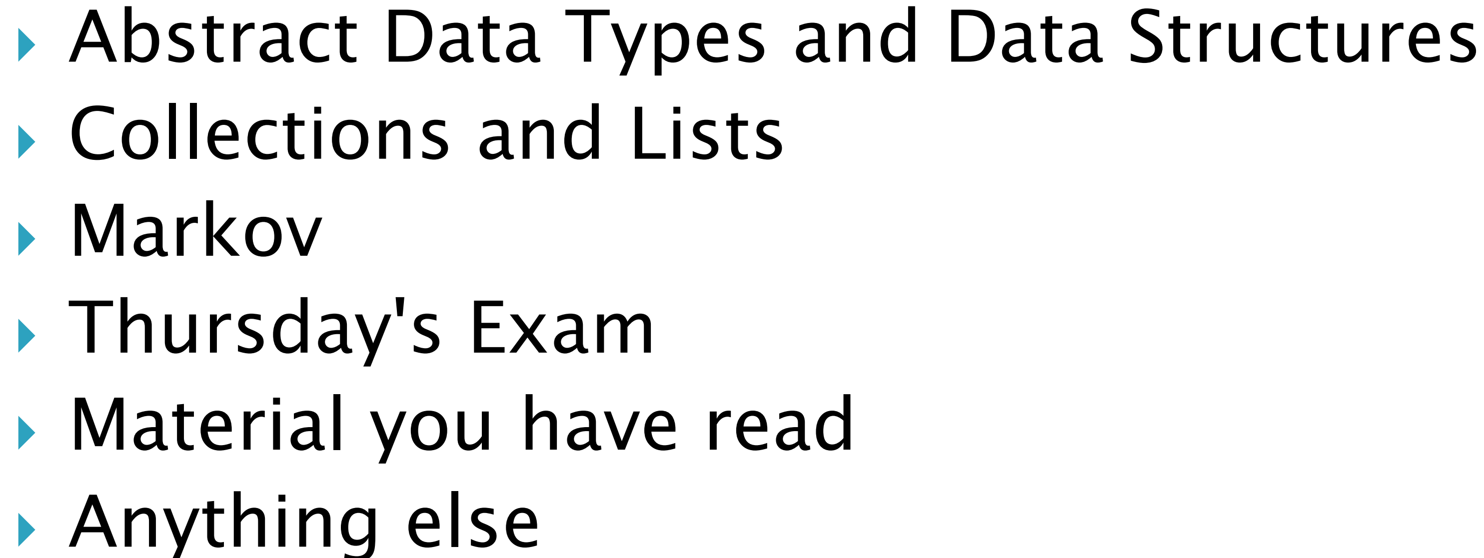
SpellChecker and Suggester

- 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 Monday'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



Answers to your questions

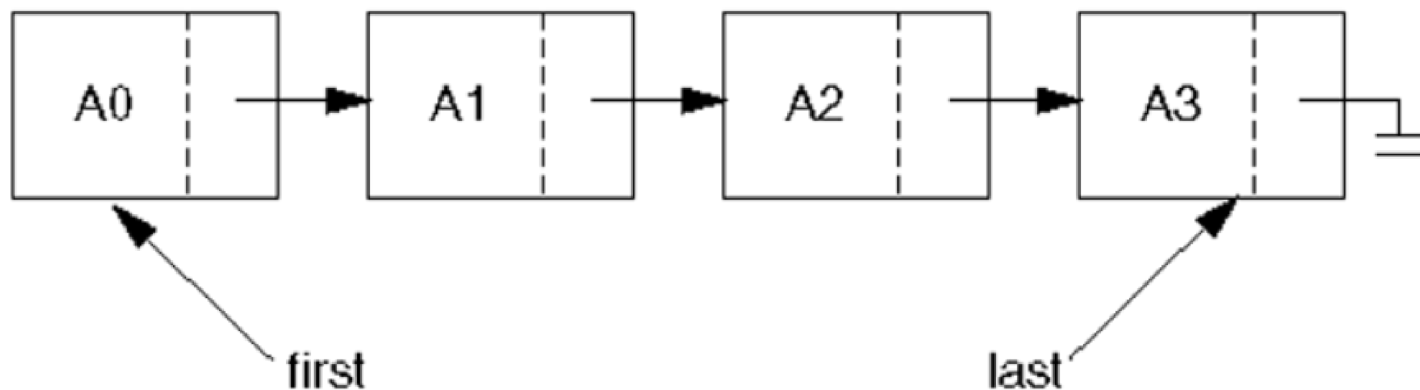


- Abstract Data Types and Data Structures
- Collections and Lists
- Markov
- Thursday's Exam
- Material you have read
- Anything else

Today's agenda

- LinkedList Implementation

`LinkedList` implementation of the `List` Interface

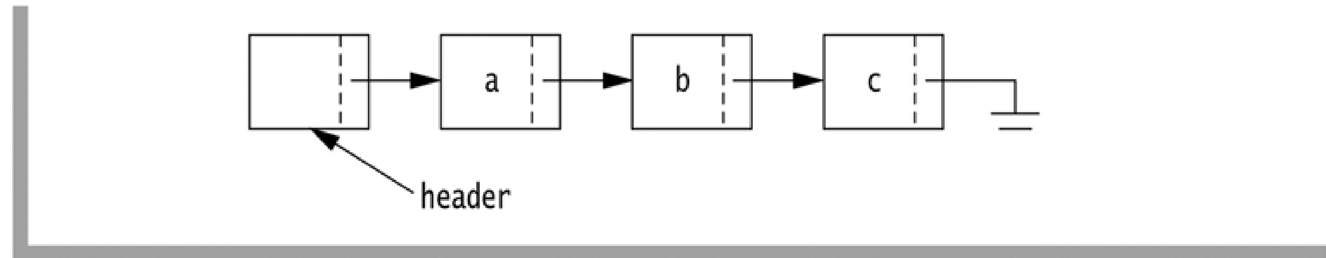


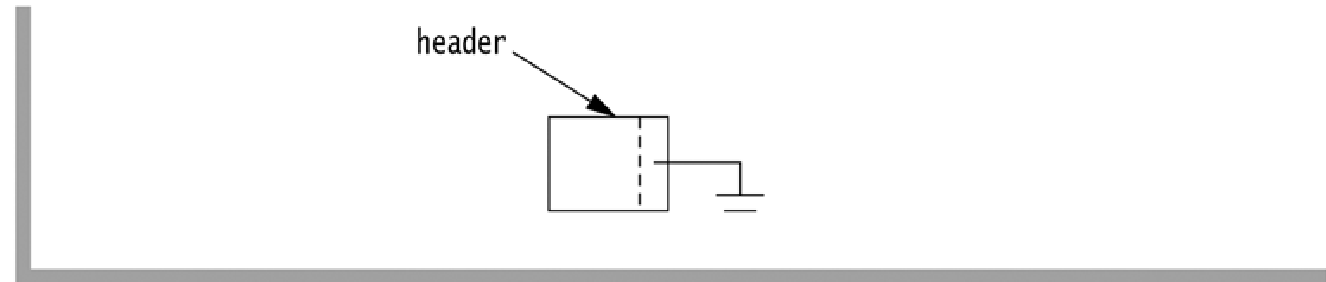
- Stores items (non-contiguously) in nodes; each contains a reference to the next node.
- Lookup by index is linear time (worst, average).
- Insertion or removal is constant time once we have found the location.
  - show how to insert A4 after A1.
- If `Comparable` list items are kept in sorted order,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





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

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?

What's an iterator?

- More specifically, what is a `java.util.Iterator`?
  - It's an interface:
  - **`interface java.util.Iterator<E>`**
  - with the following methods:

`boolean`	`hasNext()` Returns `true` if the iteration has more elements.
`E`	`next()` Returns the next element in the iteration.
`void`	`remove()` Removes from the underlying collection the last element returned by the iterator (optional operation).

An extension, `ListIterator`, adds:

`boolean`	`hasPrevious()` Returns `true` if this list iterator has more elements when traversing the list in the reverse direction.
`int`	`nextIndex()` Returns the index of the element that would be returned by a subsequent call to `next`.
`Object`	`previous()` Returns the previous element in the list.
`int`	`previousIndex()` Returns the index of the element that would be returned by a subsequent call to `previous`.
`void`	`set(Object o)` Replaces the last element returned by `next` or `previous` with the specified element (optional operation).

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.

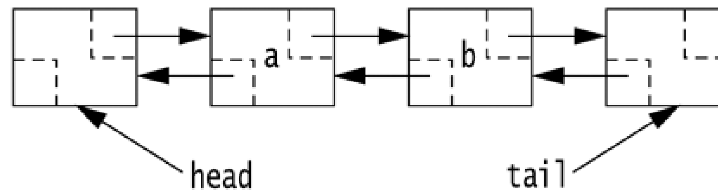


figure 17.15

A doubly linked list