Intro to Programming II
Linked Lists

Chris Brooks

Department of Computer Science
University of San Francisco
Previously we talked about using arrays to store sequences of data.

Advantages:
△ Everything is stored in sequential memory locations.
△ Fast lookup of elements.

Disadvantages:
△ Hard to resize
△ Removing or adding an element in the middle is costly.
Linked lists have the opposite advantages and disadvantages

**Advantages:**
- Easy to insert and remove
- Easy to resize

**Disadvantages:**
- Elements are not stored sequentially
- Finding the nth element is slower.
The general idea:

Each element of the list will “know” who the next element is.

Let’s try this as a class.
So how do we do this in Java?

Our Element class needs to have two components:
- The data that we want to store in the list.
- A pointer to the next element in the list.
public class ListItem {
    public Object data;
    public ListItem next;

    public ListItem(Object d) {
        data = d;
        next = null;
    }
}
11-5: Arranging ListItems

6 ListItems hook together like a chain.
6 All we need to do is keep track of the beginning of the chain.
6 No need to allocate everything ahead of time.
The LinkedList class will be responsible for hanging onto the 'head' of the list and providing methods for working with the list.

- Insert()
- InsertAt(index)
- get(index)
- remove(index)
- find(object)
public class LinkedList {
    public ListItem head = null;
    public void insert(Object o) { ... }
    public void insertAt(Object o, int index) { ... }
    public Object get(int index) { ... }
    public Object remove(int index) { ... }
    public int find(Object o) { ... }
}
So how do we add something to the front of a linked list?

- Have the new thing point to the currently-first ListItem
- Point 'head’ to our new thing.

```java
public void insert(Object o) {
    ListItem l = new ListItem(o);
    l.next = head;
    head = l;
}
```
What about adding something into the middle of a list?

If we want an item to go between current elements 5 and 6, then we need our new item to point to 6, and 5 to point to the new element.

How do we code this?
public void insertAt(Object o, int index) {
    // first find the place to insert it.
    ListItem pointer = head;
    ListItem l = new ListItem(o);
    for (int i = 0; i < index - 1; i++) {
        pointer = pointer.next;
    }
    l.next = pointer;
    pointer = l;
}
Are there any special cases we need to worry about?
Are there any special cases we need to worry about?
- What if the list is empty?
- What if it only has one element?
- What if we’re adding at the end?
11-13: Exercise

6 Code the ListItem and LinkedList classes as described above.
6 Write a main method that creates a list and prompts the user for strings, which are always added to the front of the list.
6 Add a method to the LinkedList class called toString().
6 This should walk the list and print out each of the strings in order.