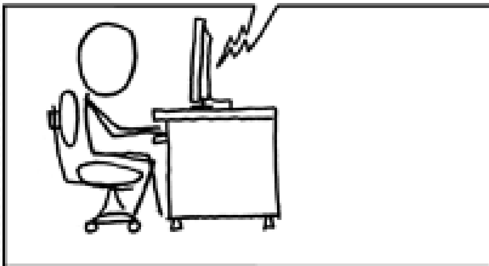
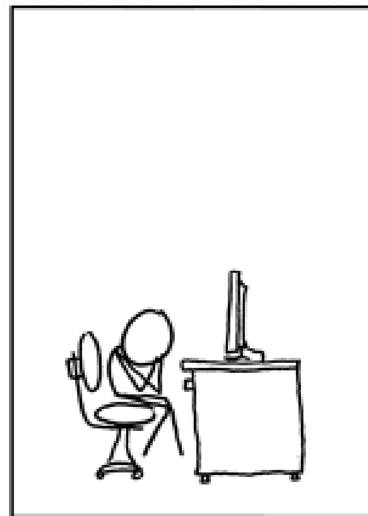
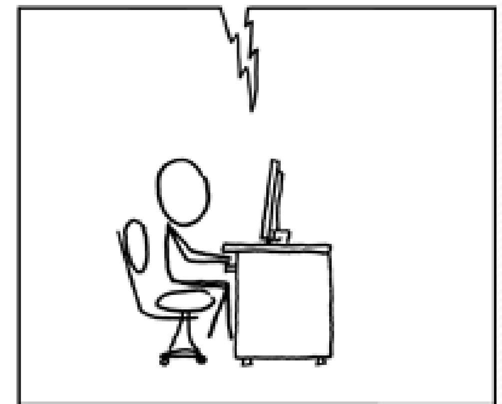
CSE 143

Lecture 8: Complex Linked List Code

reading: 16.2 – 16.3





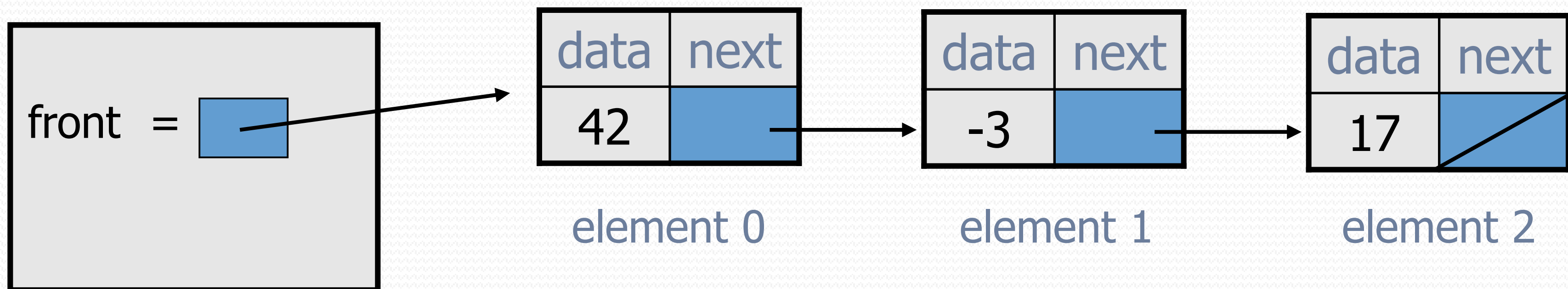


In some languages (C++), -> is used for dereferencing

Implementing `add` (2)

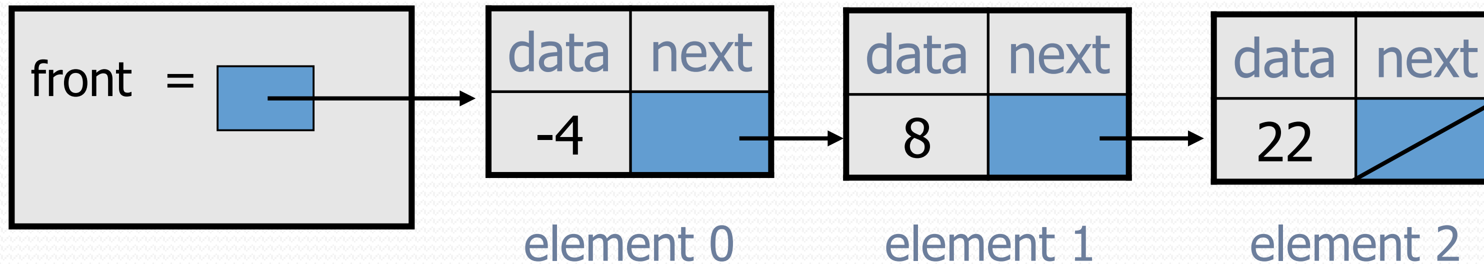
```
// Inserts the given value at the given index.
public void add(int index, int value) {
    ...
}
```

- Exercise: Implement the two-parameter `add` method.

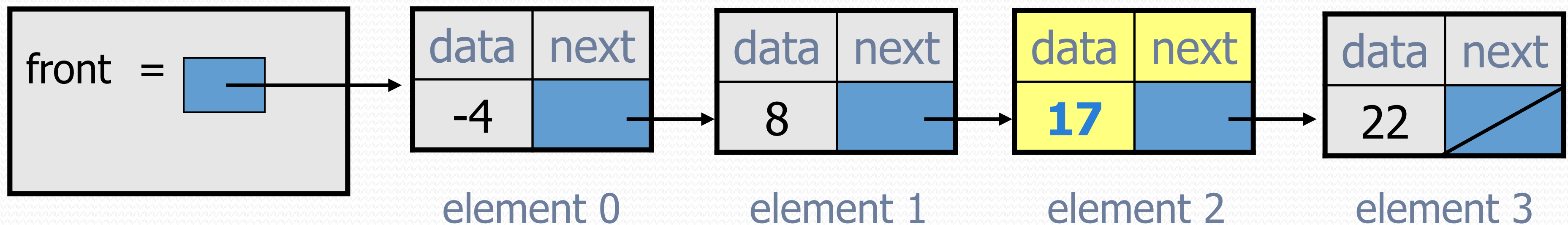


addSorted

- Write a method `addSorted` that accepts an `int` as a parameter and adds it to a sorted list in sorted order.
  - Before `addSorted(17)` :



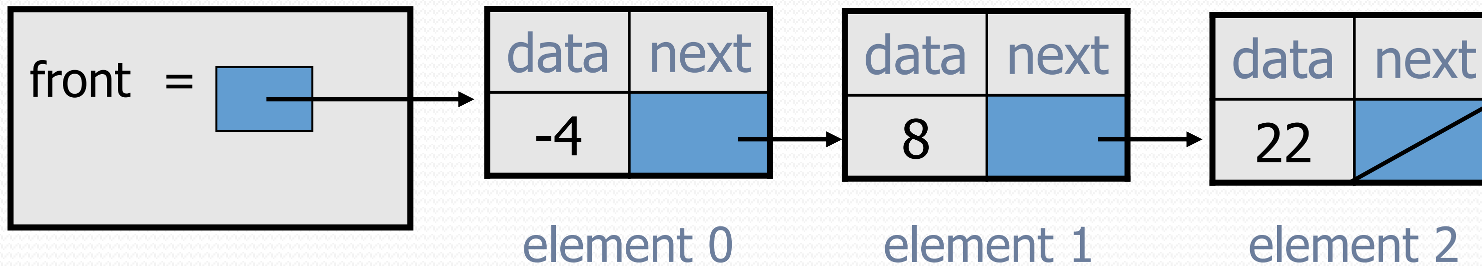
  - After `addSorted(17)` :



The common case

- Adding to the middle of a list:

```
addSorted(17)
```

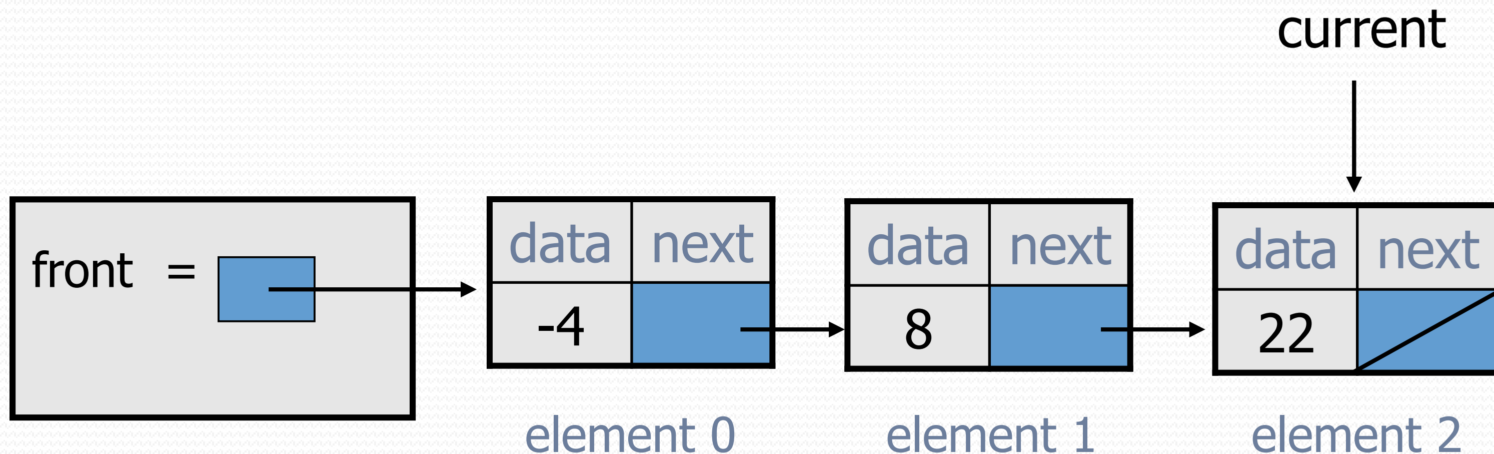


- Which references must be changed?
- What sort of loop do we need?
- When should the loop stop?

First attempt

- An incorrect loop:

```
ListNode current = front;
while (current.data < value) {
    current = current.next;
}
```

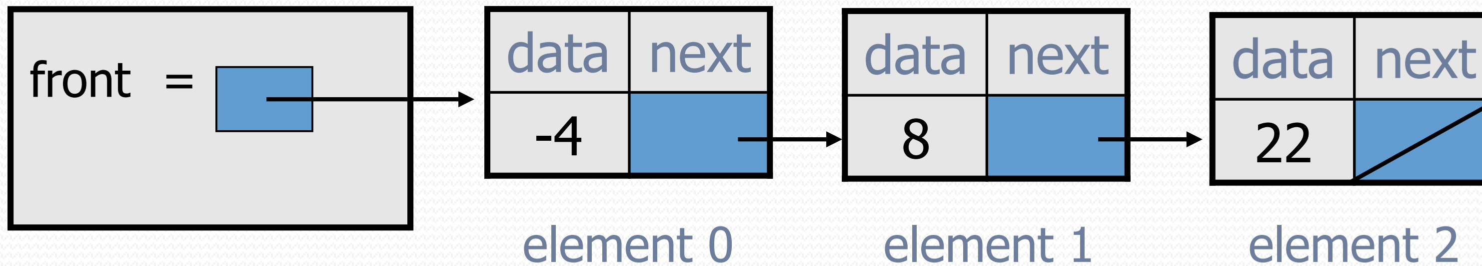


- What is wrong with this code?
  - The loop stops too late to affect the list in the right way.

Another case to handle

- Adding to the end of a list:

```
addSorted(42)
```



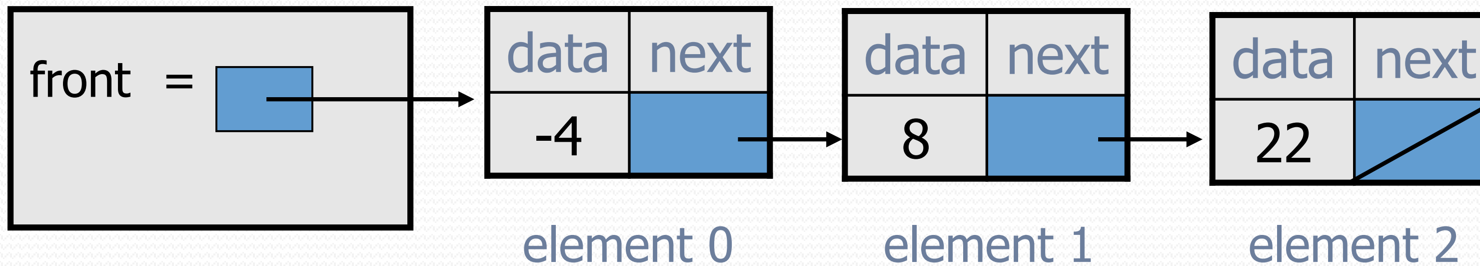

```
Exception in thread "main": java.lang.NullPointerException
```

  - Why does our code crash?
  - What can we change to fix this case?

Third case to handle

- Adding to the front of a list:

```
addSorted(-10)
```

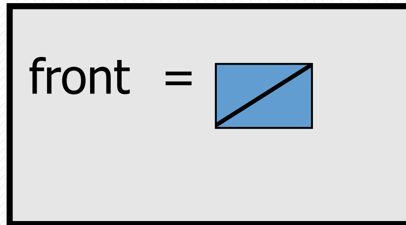


- What will our code do in this case?
- What can we change to fix it?

Fourth case to handle

- Adding to (the front of) an empty list:

```
addSorted(42)
```



  - What will our code do in this case?
  - What can we change to fix it?

Common cases

- **middle**: "typical" case in the middle of an existing list
- **back**: special case at the back of an existing list
- **front**: special case at the front of an existing list
- **empty**: special case of an empty list

Other list features

- Add the following methods to the `LinkedIntList`:
  - `size`
  - `isEmpty`
  - `clear`
  - `toString`
  - `indexOf`
  - `contains`
  - `remove`

- Add preconditions and exception tests to appropriate methods.

Interfaces

- **interface**: A list of methods that a class can promise to implement.
  - Inheritance gives you an is-a relationship *and* code sharing.
    - A `Lawyer` can be treated as an `Employee` and inherits its code.
  - Interfaces give you an is-a relationship *without* code sharing.
    - A `Rectangle` object can be treated as a `Shape` but inherits no code.
  - Always declare variables using the ***interface*** type.

```
List<String> list = new ArrayList<String>();
```