CSE 143
Lecture 10

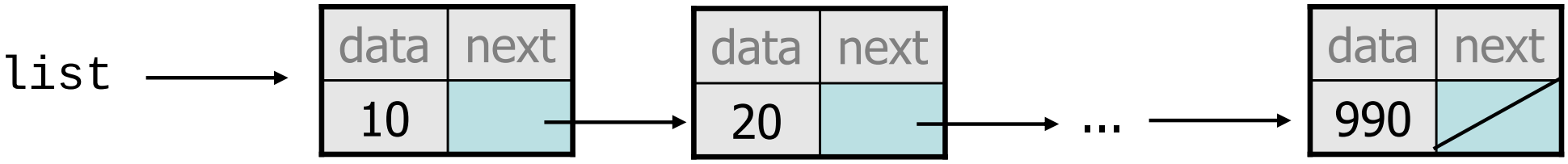
Linked List Basics

reading: 16.1 - 16.2

slides created by Marty Stepp
http://www.cs.washington.edu/143/

Linked node question

- Suppose we have a long chain of list nodes:



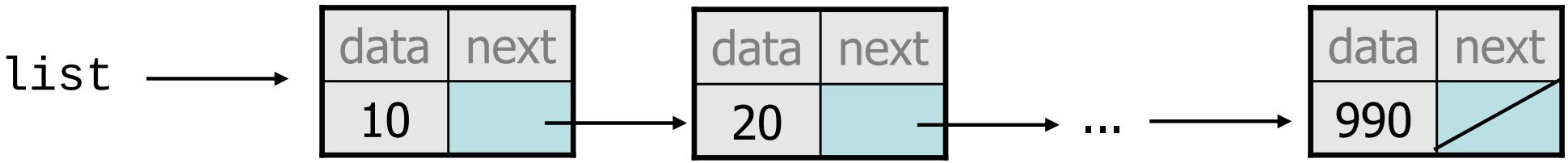
- How would we write code to print the data values in all the nodes in order without redundancy?

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?

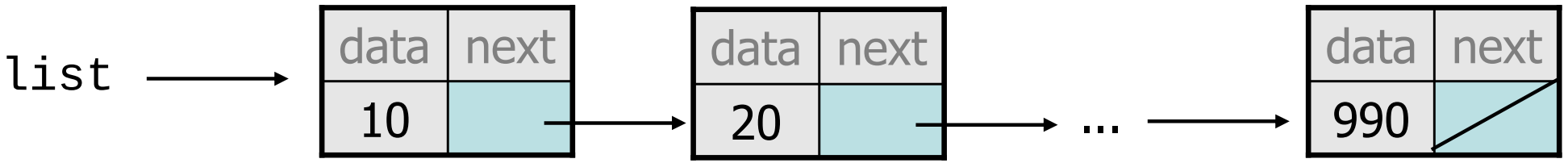
```
list = list.next;   // is this a good idea?
```



A "current" reference

- Important: A `ListNode` variable is NOT a `ListNode` object!

```
ListNode current = list;
```




```
current   ???
```

- Move along a list by advancing a `ListNode` reference.

```
current = current.next;
```

Printing algorithm

- Algorithm to print all list nodes:

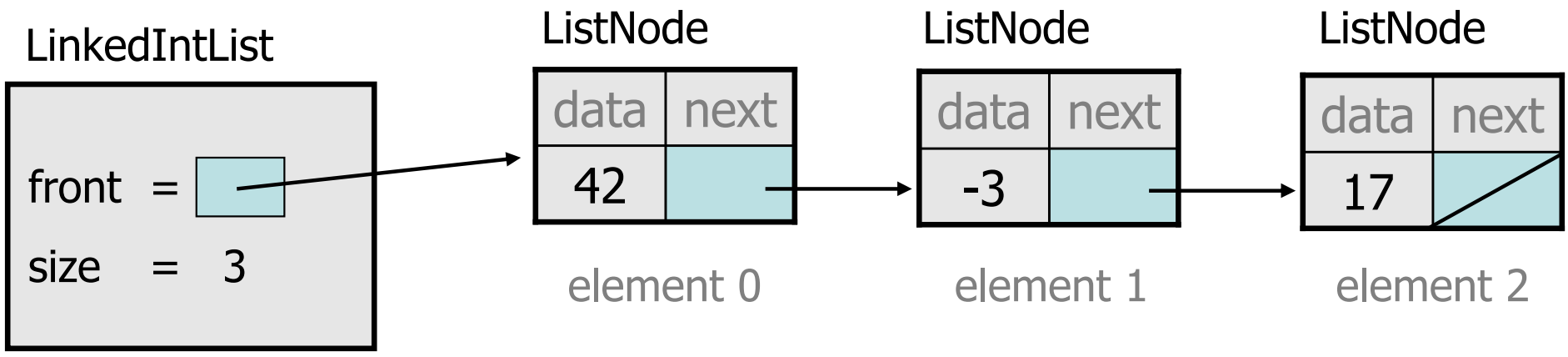
```
ListNode front = ...;

ListNode current = front;
while (current != null) {
    System.out.println(current.data);
    current = current.next;
}
```

- Similar to array code:

```
int[] a = ...;

int i = 0;
while (i < a.length) {
    System.out.println(a[i]);
    i++;
}
```

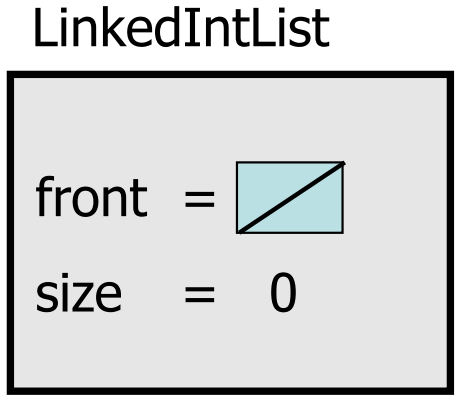
A `LinkedIntList` class

- Let's write a 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



`LinkedIntList` class v1

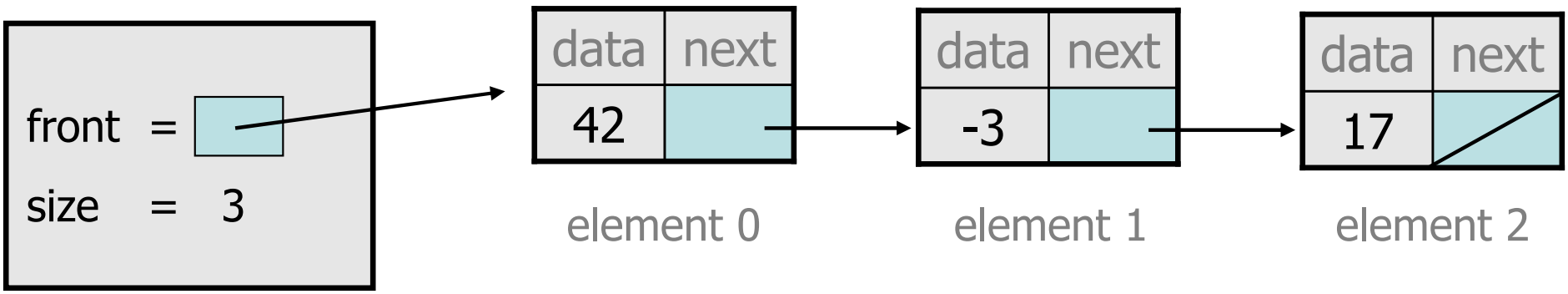
```
public class LinkedIntList {
    private ListNode front;
    private int size;

    public LinkedIntList() {
        front = null;
        size = 0;
    }


    methods go here

}
```



Implementing add

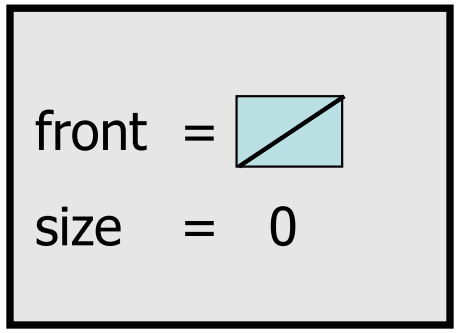
```
// Adds the given value to the end of the list.
public void add(int value) {
    ...
}
```

- 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?

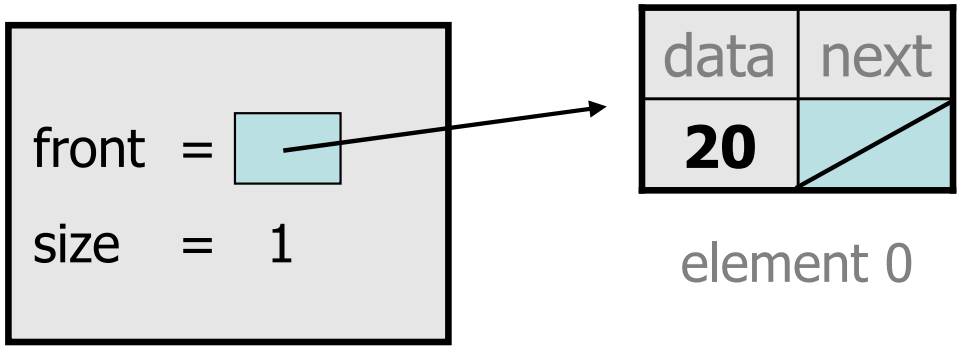


Adding to an empty list

- Before adding 20:



After:

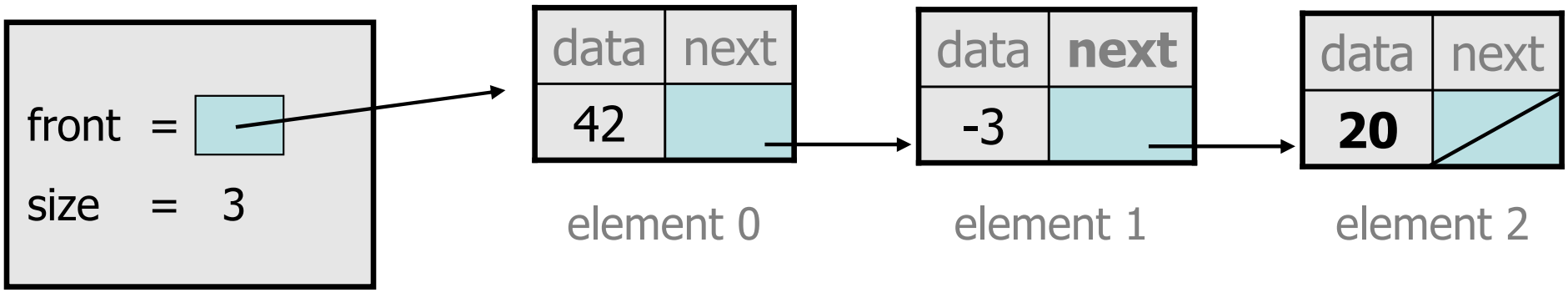


  - We must create a new node and attach it as the front of the list.

Adding to non-empty list

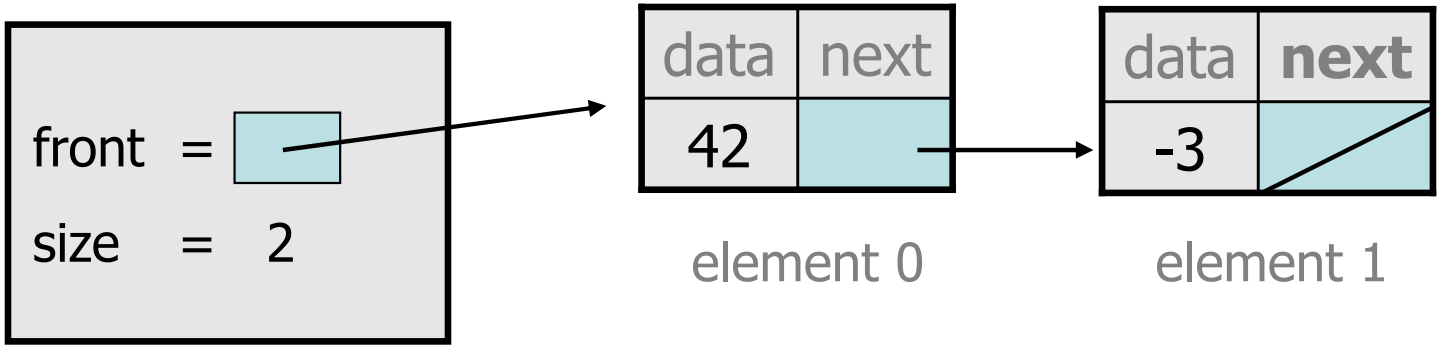
- Before adding value 20 to end of list:

- After:



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?

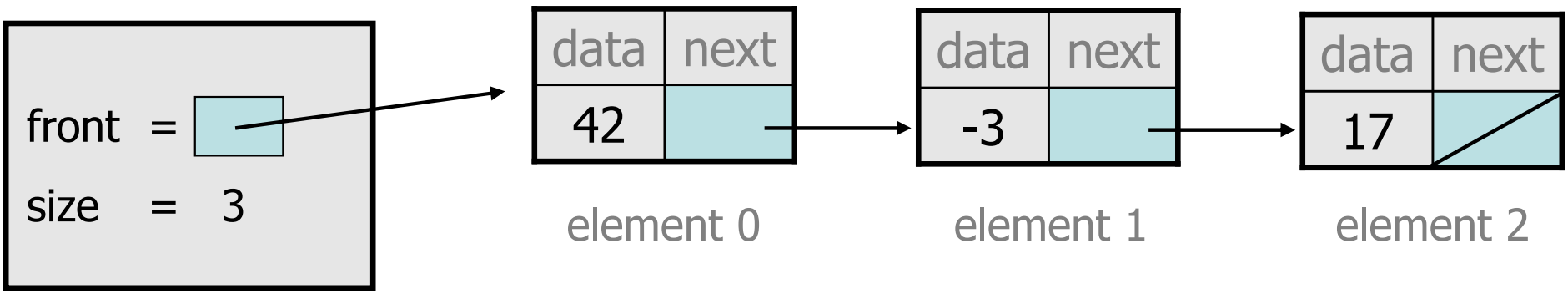
The add method

```
// 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);
    }
    size++;
}
```

Implementing `get`

```
// Returns value in list at given index.
public int get(int index) {
    …
}
```

– Exercise: Implement the `get` method.



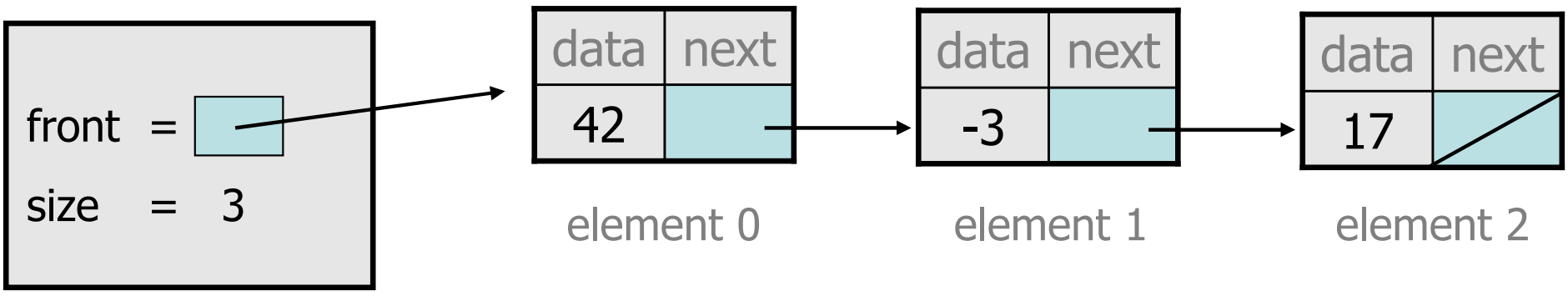
The `get` method

```
// 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.



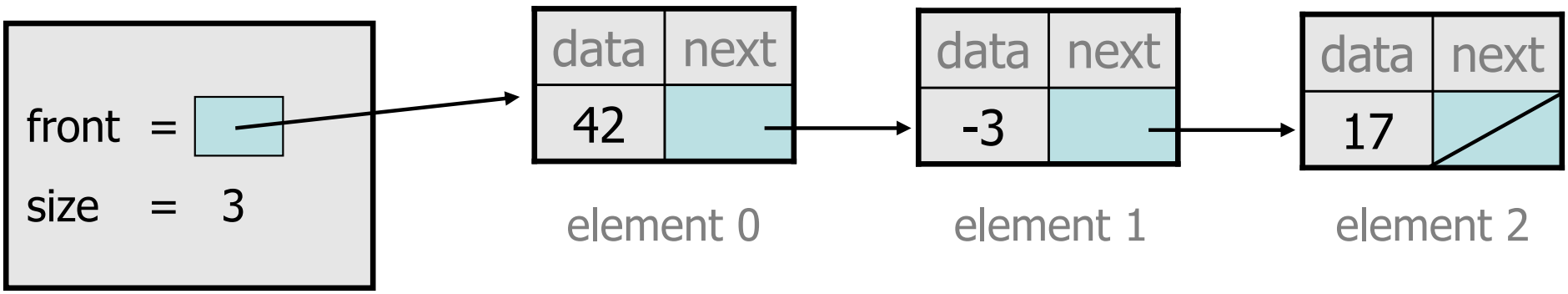
The add method (2)

```
// 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);
    }
    size++;
}
```

Implementing remove

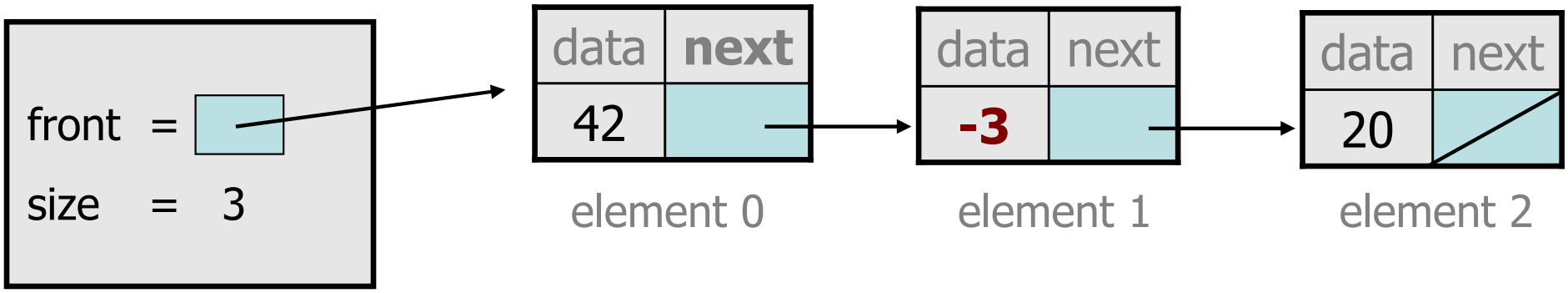
```
// Removes value at given index from list.
public void remove(int index) {
    ...
}
```

- How do we remove a node from a list?
- Does it matter what the list's contents are before the remove?

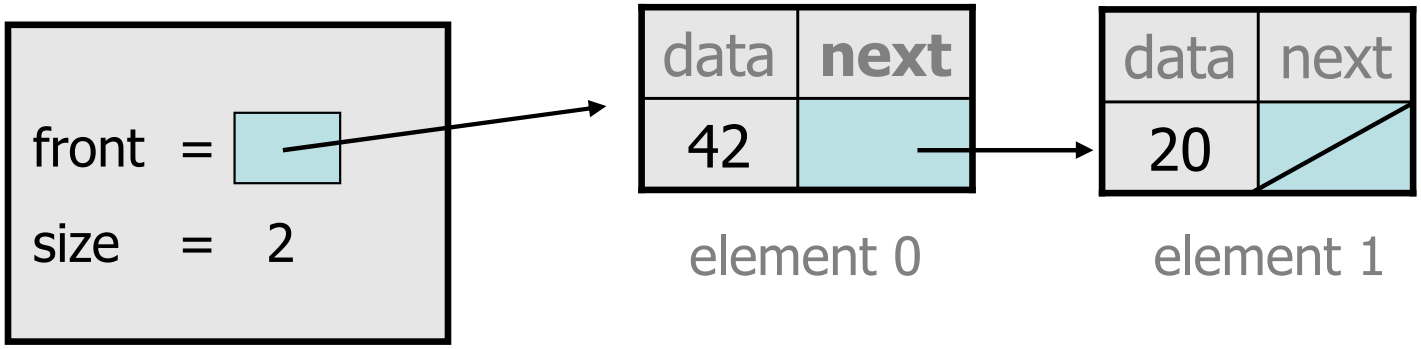


Removing from a list

- Before removing element at index 1:

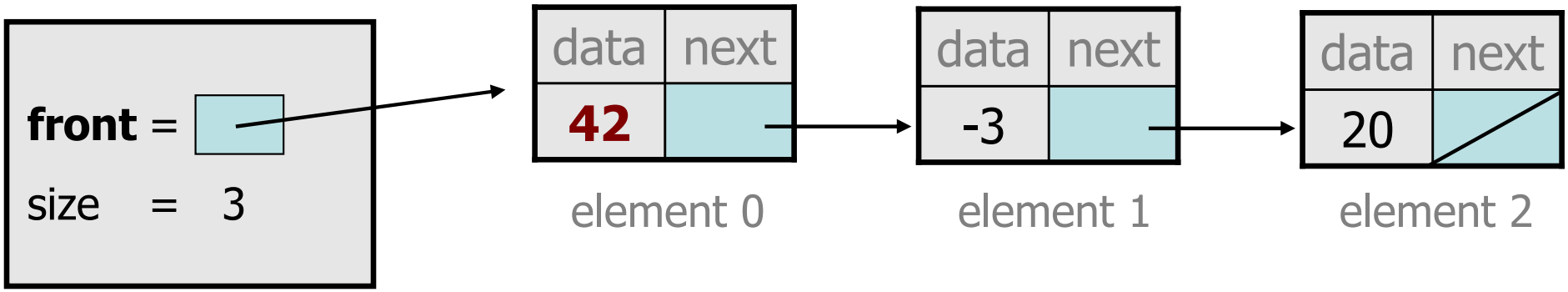


- After:

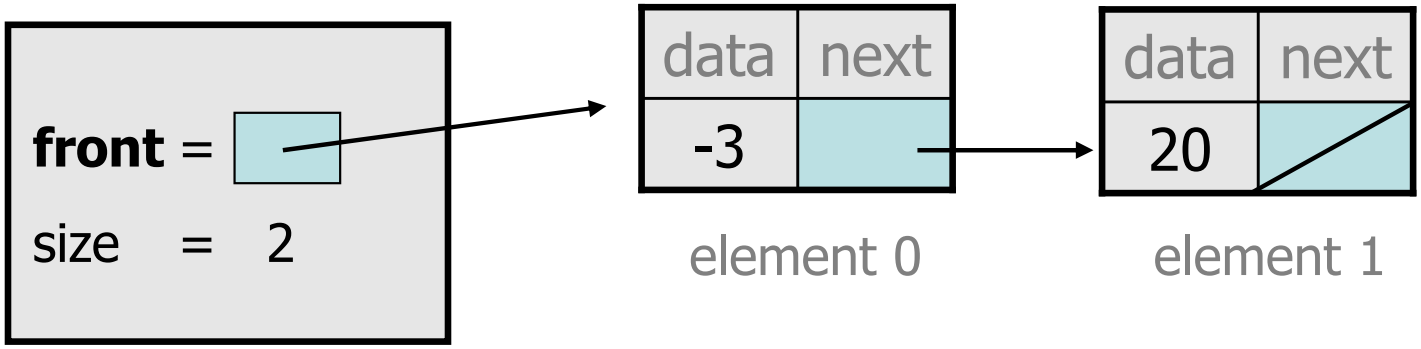


Removing from the front

- Before removing element at index 0:

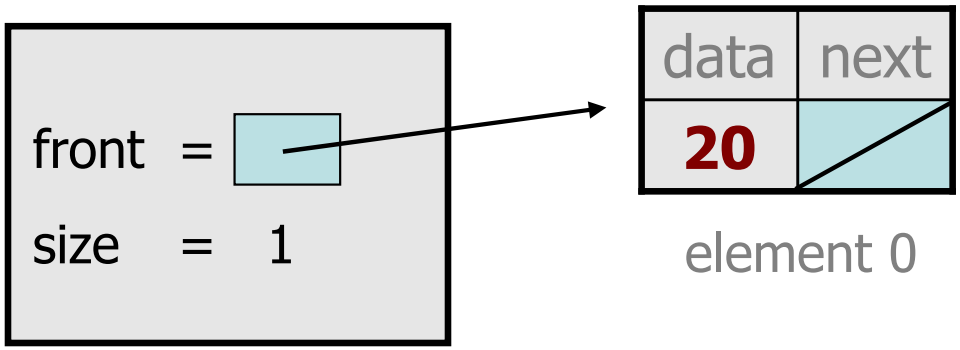


- After:

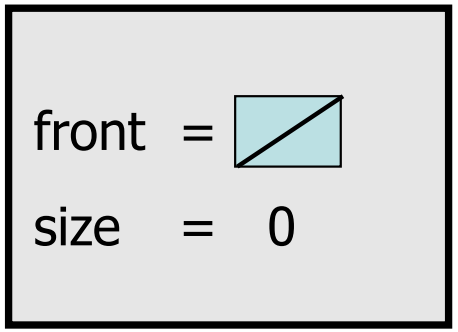


Removing the only element

- Before:



After:



  - We must change the front field to store `null` instead of a node.
  - Do we need a special case to handle this?

The remove method

```
// Removes value at given index from list.
// Precondition: 0 <= index < size()
public void remove(int index) {
    if (index == 0) {
        // special case: removing first element
        front = front.next;
    } else {
        // removing from elsewhere in the list
        ListNode current = front;
        for (int i = 0; i < index - 1; i++) {
            current = current.next;
        }
        current.next = current.next.next;
    }
    size--;
}
```