Building Java Programs

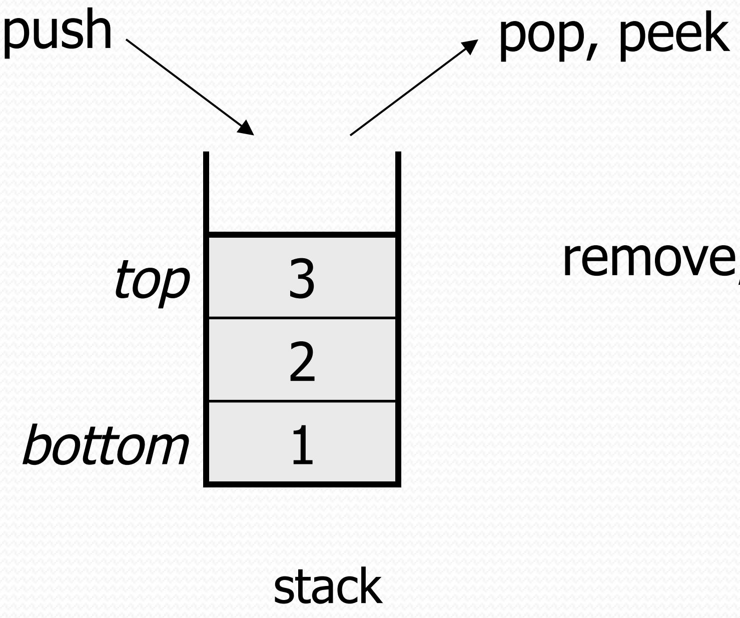
Chapter 16
Linked Nodes

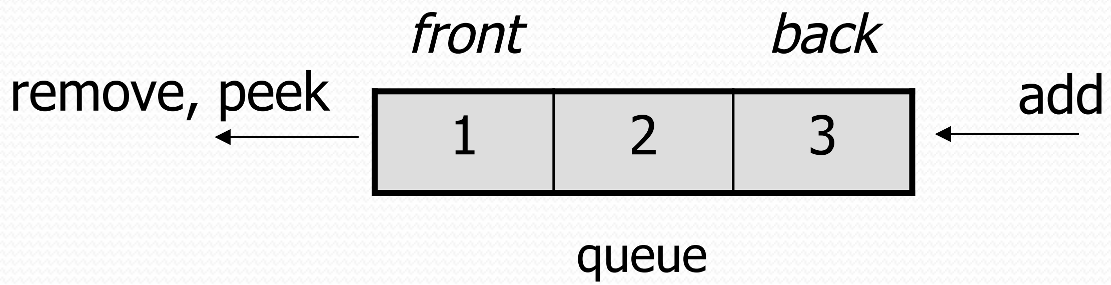
reading: 16.1

LATE NIGHT JIMMY FALLON

Recall: stacks and queues

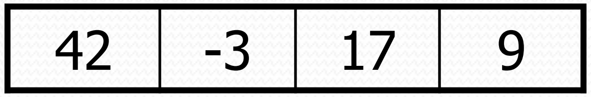
- **stack**: retrieves elements in reverse order as added
- **queue**: retrieves elements in same order as added



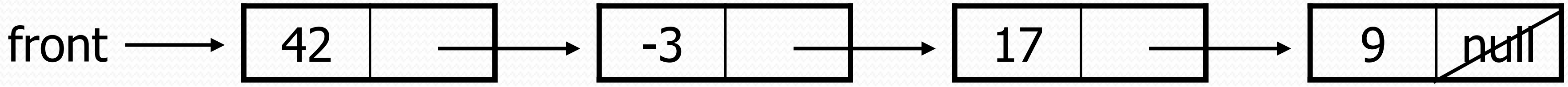


Array vs. linked structure

- All collections in this course use one of the following:
  - an **array** of all elements
    - examples: `ArrayList`, `Stack`, `HashSet`, `HashMap`



  - **linked objects** storing a value and references to other(s)
    - examples: `LinkedList`, `TreeSet`, `TreeMap`



    - First, we will learn how to create a *linked list*.
    - To understand linked lists, we must understand *references*.

Memory for a List

- Array (contiguous in memory)

42	-3	17	9

- Spread in memory

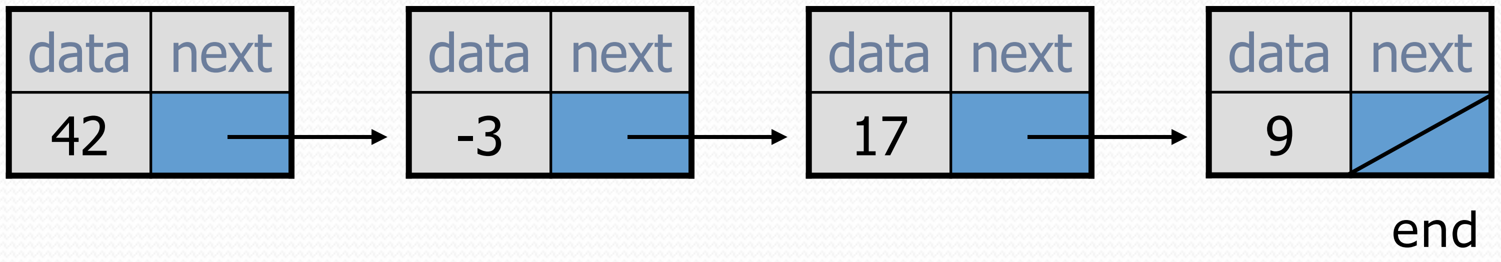
42			9		-3			17

A list node class

```
public class ListNode {
    public int data;
    public ListNode next;
}
```

- Each list node object stores:
  - one piece of integer data
  - a reference to another list node

- `ListNode`s can be "linked" into chains to store a list of values:



References to same type

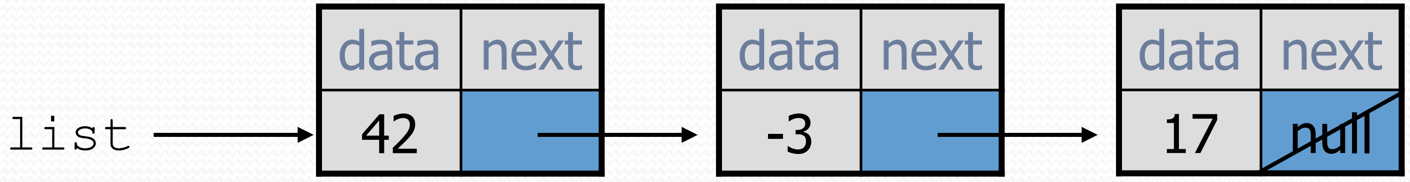
- What would happen if we had a class that declared one of its own type as a field?

```
public class Strange {
    private String name;
    private Strange other;
}
```

  - Will this compile?
    - If so, what is the behavior of the `other` field? What can it do?
    - If not, why not? What is the error and the reasoning behind it?

List node client example

```
public class ConstructList1 {
    public static void main(String[] args) {
        ListNode list = new ListNode();
        list.data = 42;
        list.next = new ListNode();
        list.next.data = -3;
        list.next.next = new ListNode();
        list.next.next.data = 17;
        list.next.next.next = null;
        System.out.println(list.data + " " + list.next.data
                           + " " + list.next.next.data);
        // 42 -3 17
    }
}
```



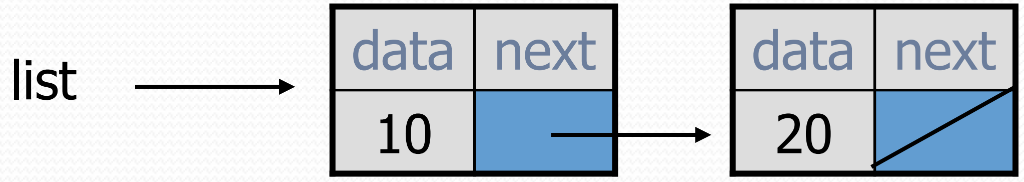
List node w/ constructor

```
public class ListNode {
    int data;
    ListNode next;

    public ListNode(int data) {
        this(data, null);

    }

    public ListNode(int data, ListNode next) {
        this.data = data;
        this.next = next;
    }
}
```

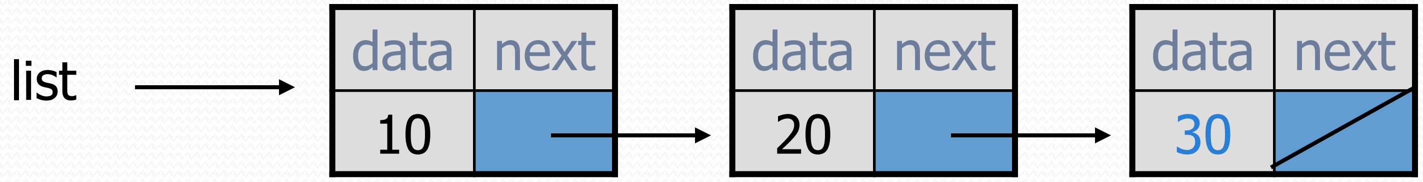
- Exercise: Modify the previous client to use these constructors.

Linked node problem 1

- What set of statements turns this picture:

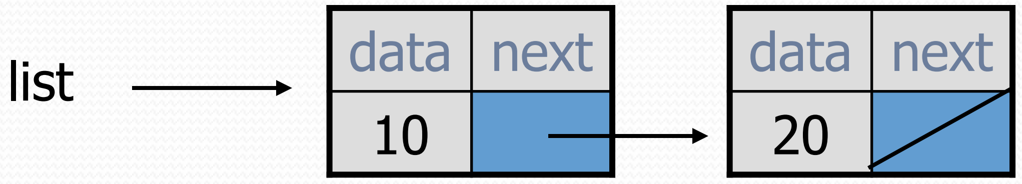


- Into this?

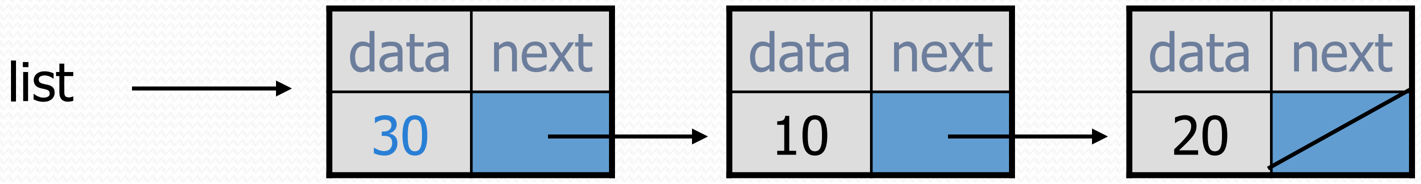


Linked node problem 2

- What set of statements turns this picture:



- Into this?

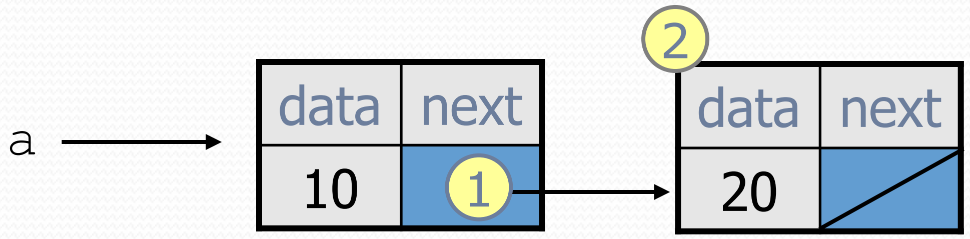


References vs. objects

variable `=` **value**`;`

a *variable* (left side of `=`) is an arrow (the base of an arrow)
a *value* (right side of `=`) is an object (a box; what an arrow points at)

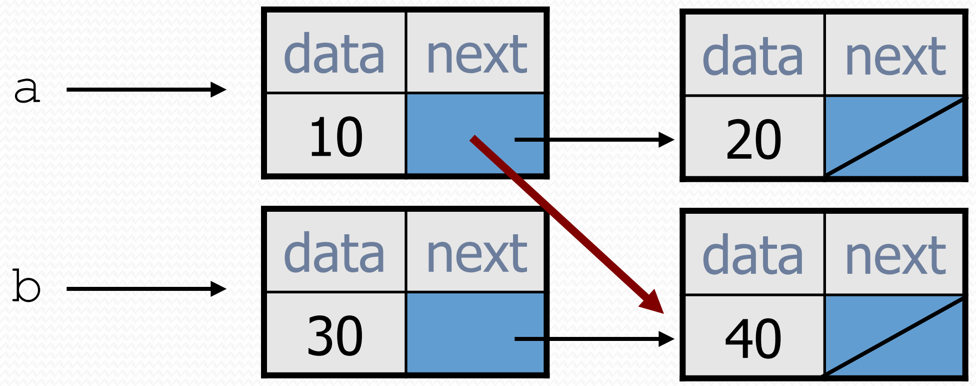
- For the list at right:
  - `a.next =` **value**`;`
    means to adjust where ① points
  - **variable** `= a.next;`
    means to make **variable** point at ②



Reassigning references

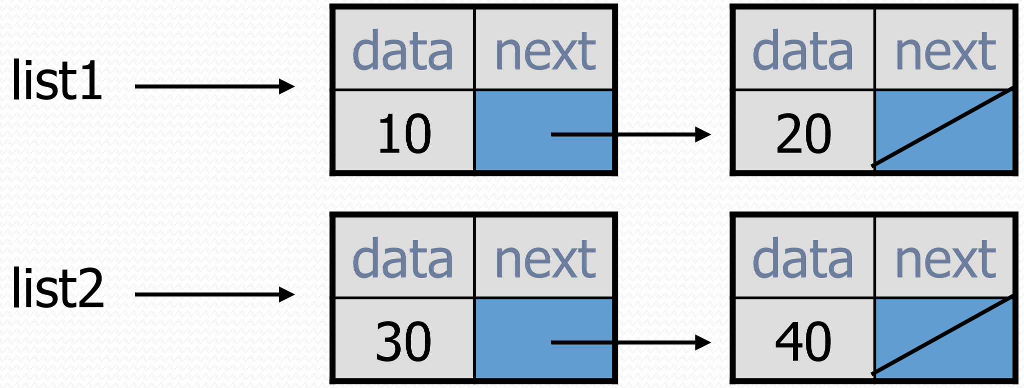
- when you say:
  - `a.next = b.next;`

- you are saying:
  - "Make *variable* `a.next` refer to the same *value* as `b.next`."
  - Or, "Make `a.next` point to the same place that `b.next` points."

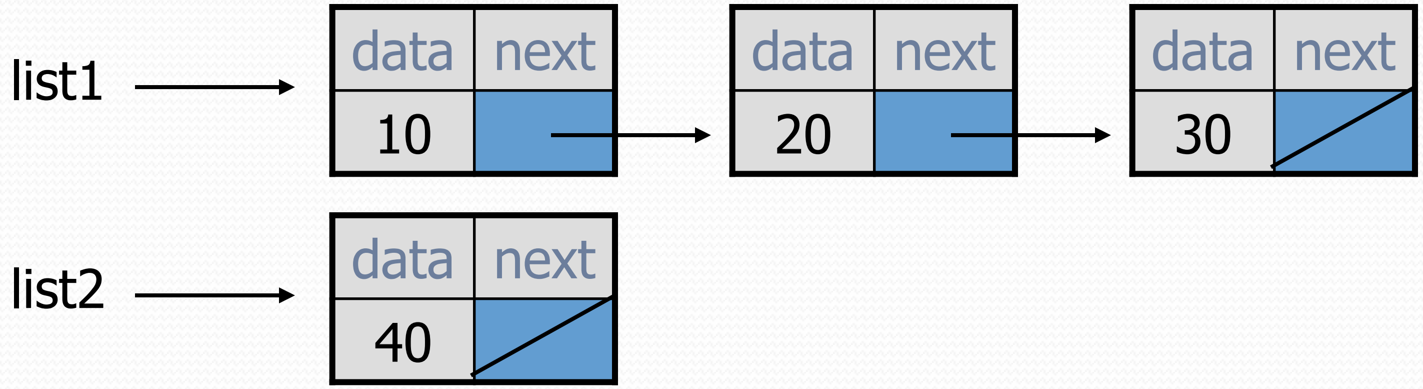


Linked node problem 3

- What set of statements turns this picture:

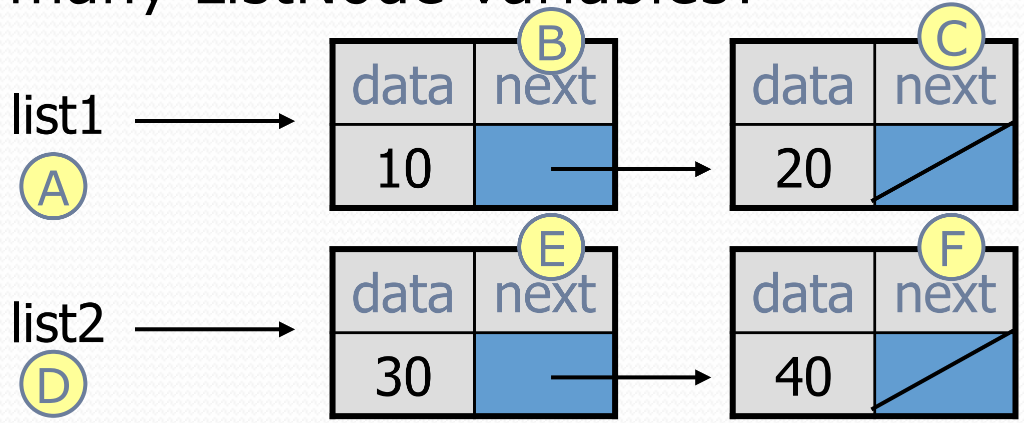


- Into this?

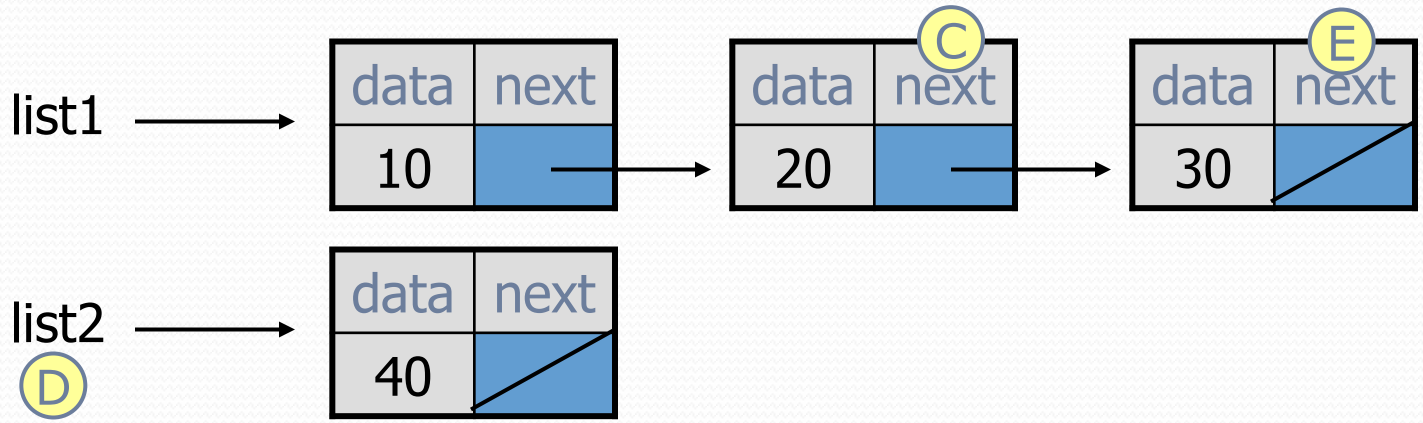


Linked node problem 3

- How many ListNode variables?

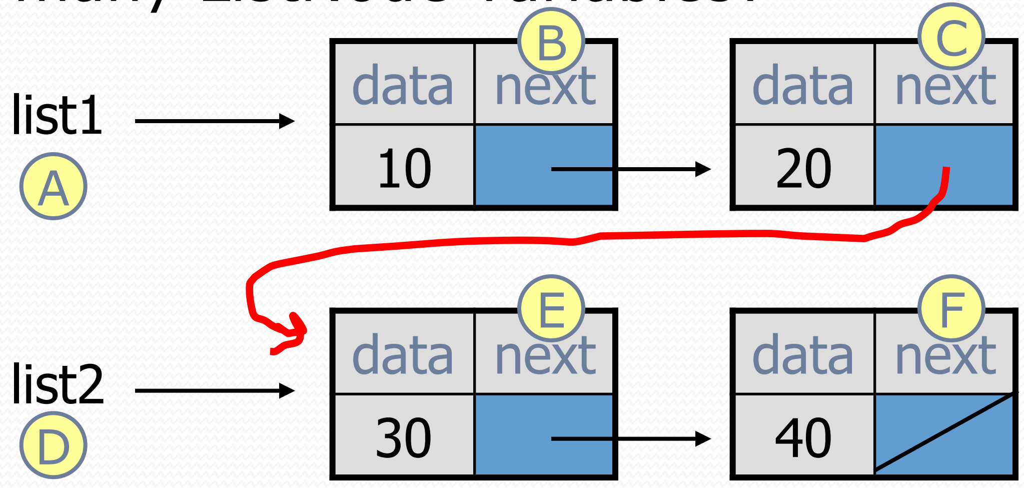


- Which variables change?



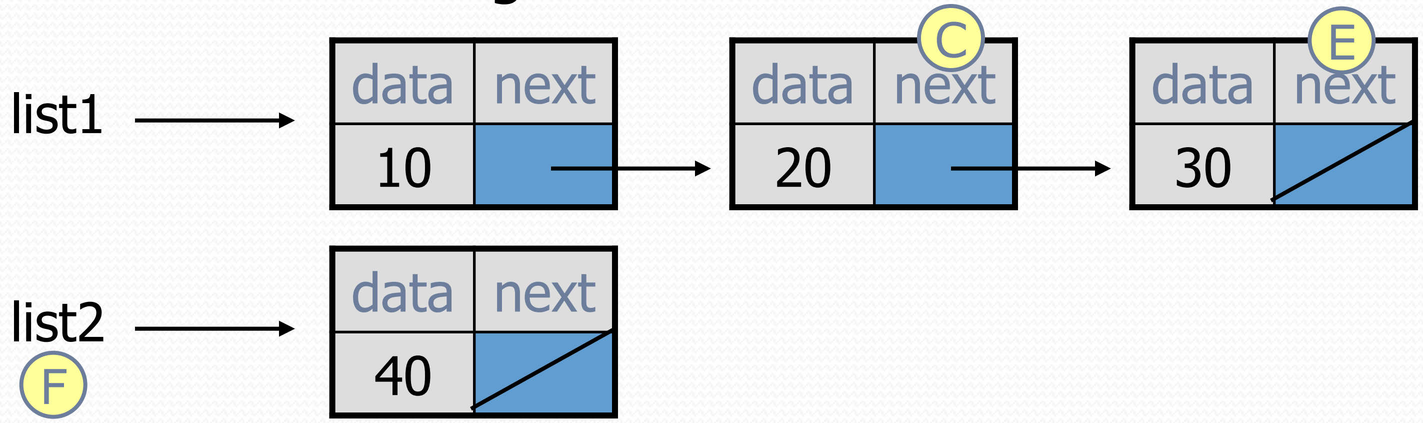
Linked node problem 3

- How many ListNode variables?



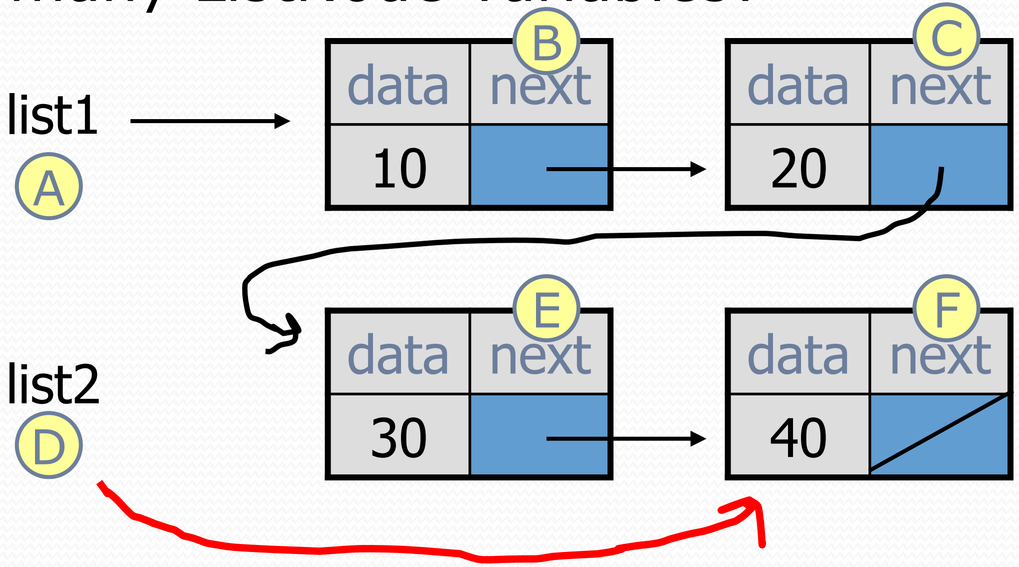
list1.next.next = list2

- Which variables change?



Linked node problem 3

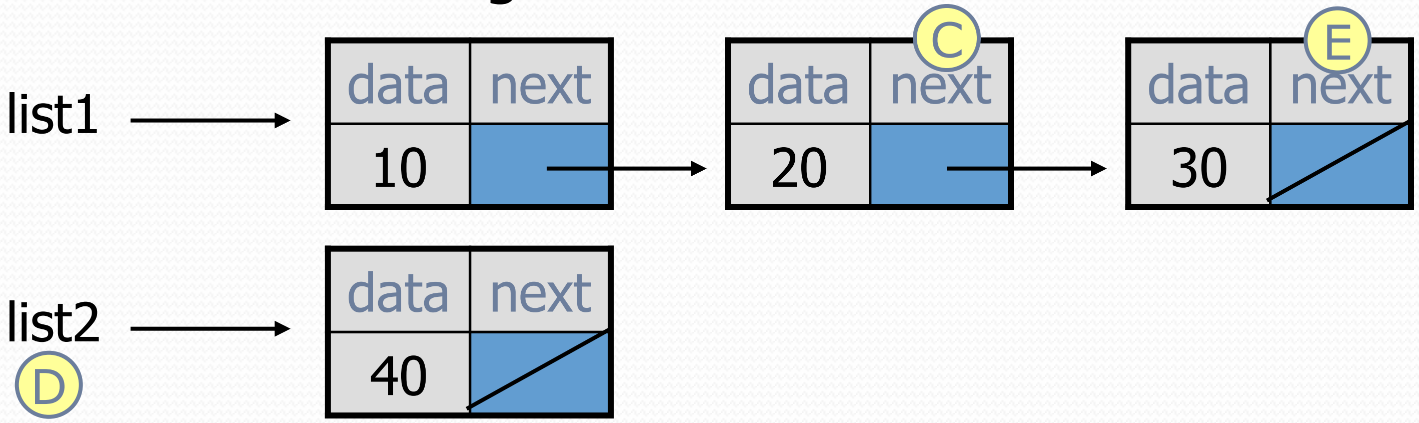
- How many ListNode variables?



list1.next.next = list2

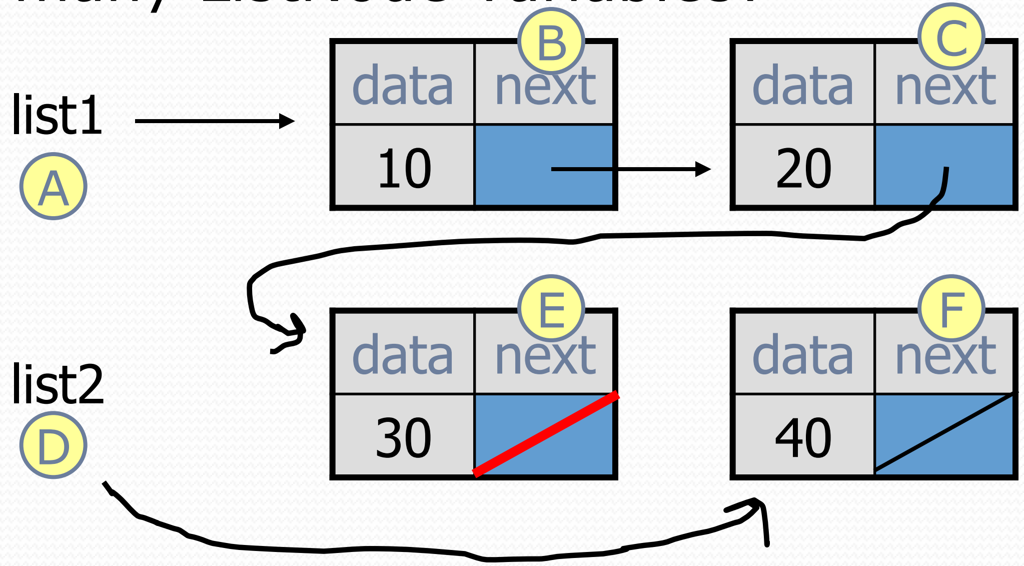
list2 = list2.next

- Which variables change?



Linked node problem 3

- How many ListNode variables?

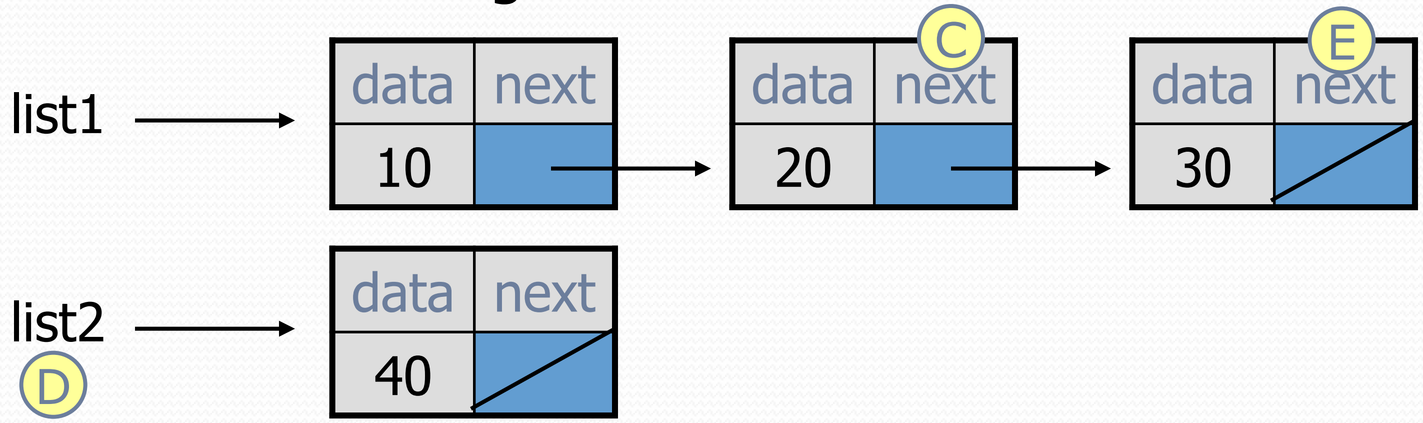


list1.next.next = list2

list2 = list2.next

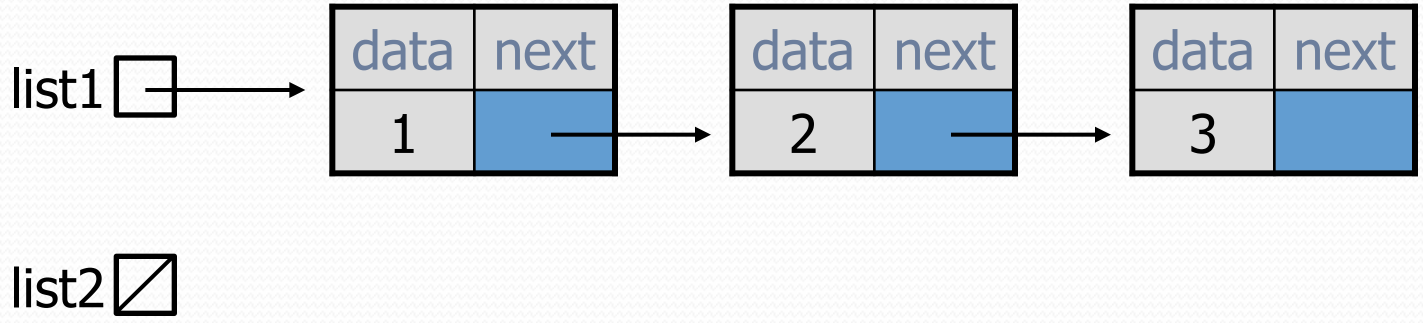
list1.next.next.next = null

- Which variables change?

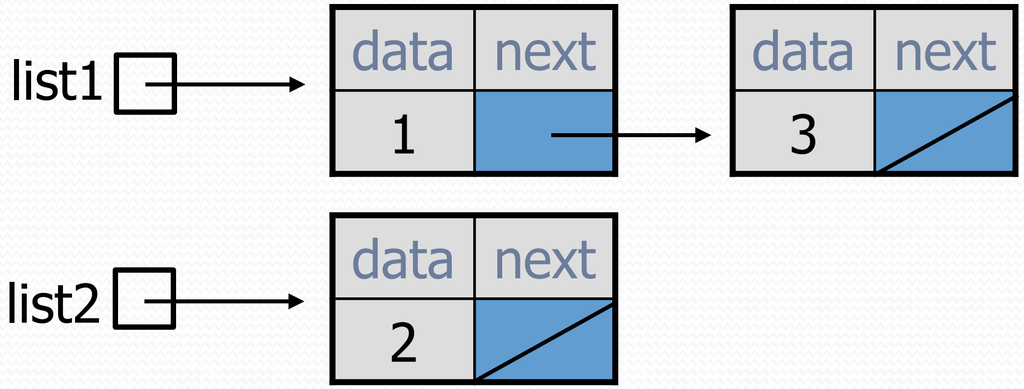


Linked node problem 4

- What set of statements turns this picture:



- Into this?

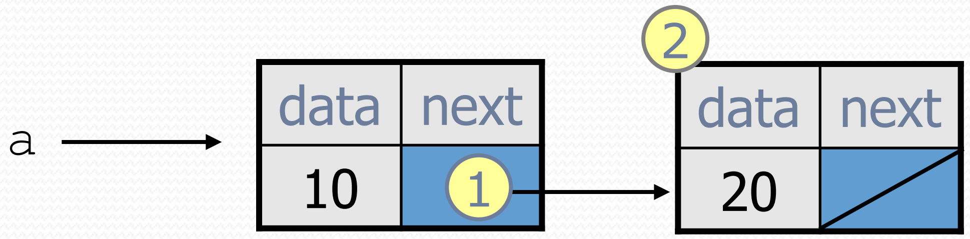


References vs. objects

variable `=` **value**`;`

a *variable* (left side of `=`) is an arrow (the base of an arrow)
a *value* (right side of `=`) is an object (a box; what an arrow points at)

- For the list at right:



  - `a.next =` **value**`;`
    means to adjust where 1 points
  - **variable** `= a.next;`
    means to make **variable** point at 2