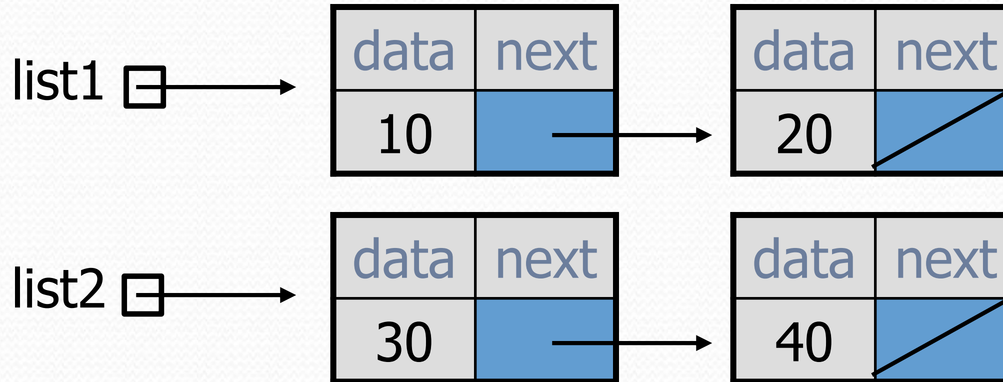
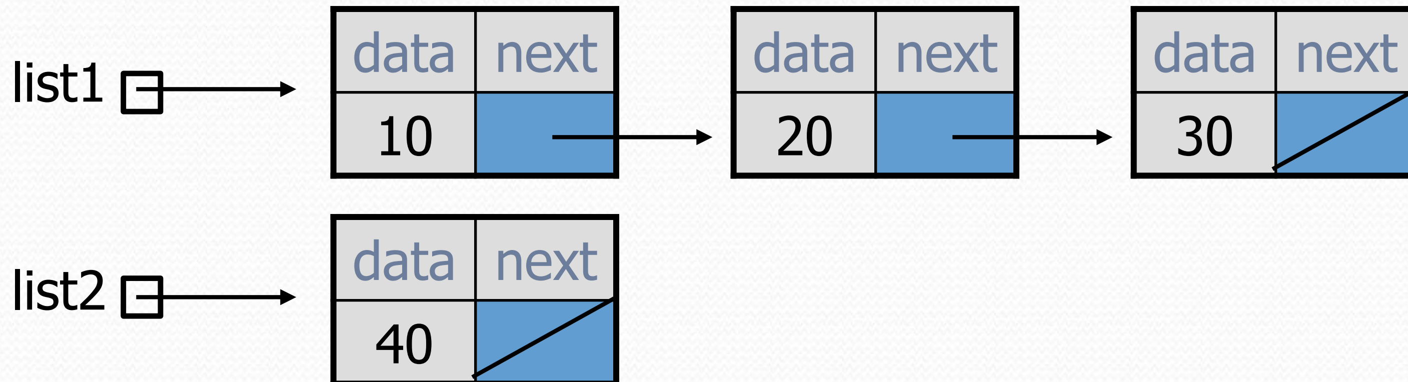
Linked node problem 3

- What set of statements turns this picture:

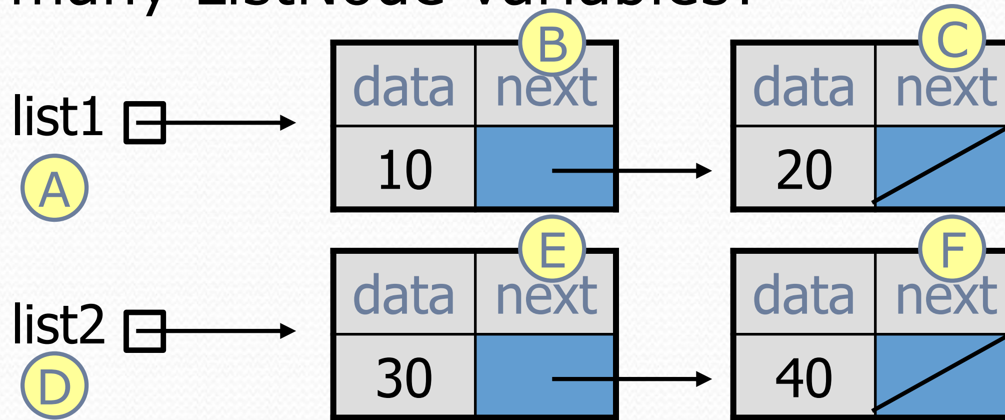


- Into this?

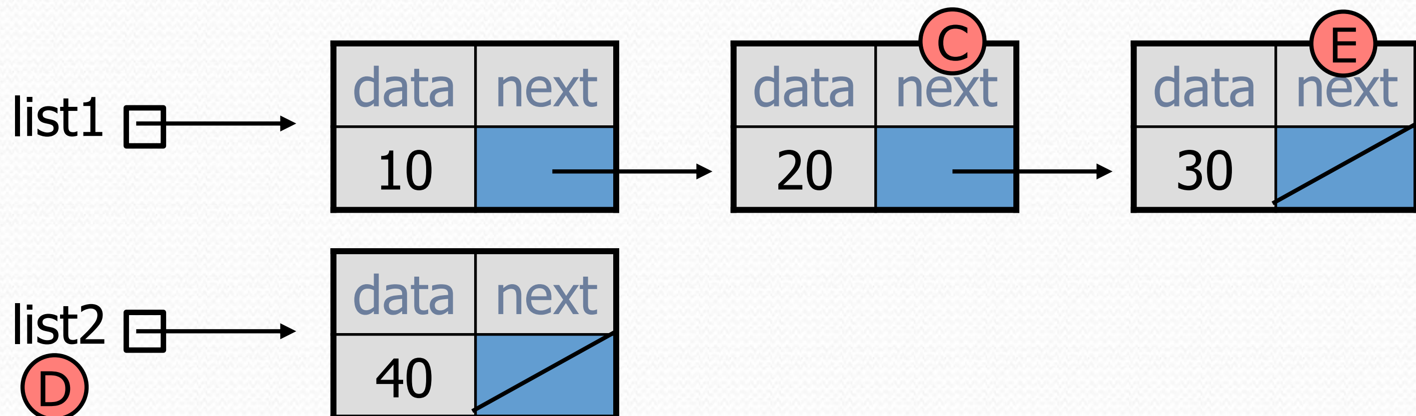


Linked node problem 3

- How many ListNode variables?

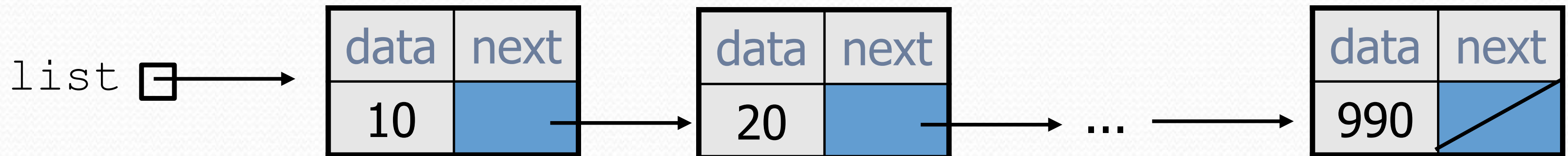


- Which variables change?



Linked node question

- Suppose we have a long chain of list nodes:



  - We don't know exactly how long the chain is.

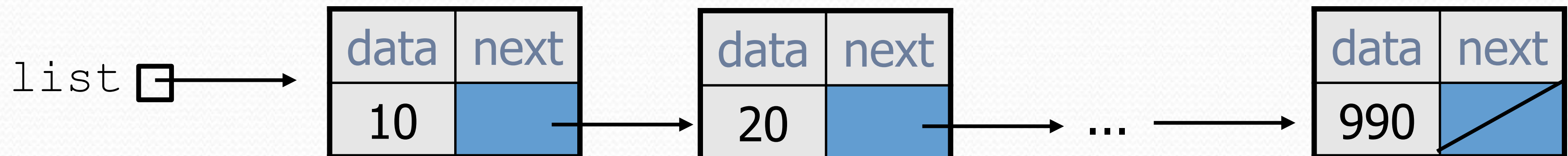
- How would we print the data values in all the nodes?

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?
```

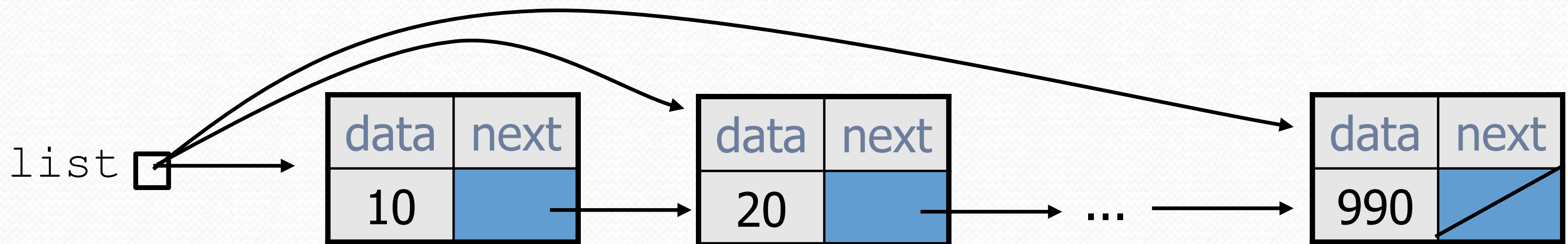


Traversing a list?

- One (bad) way to print every value in the list:

```
while (list != null) {
    System.out.println(list.data);
    list = list.next;     // move to next node
}
```

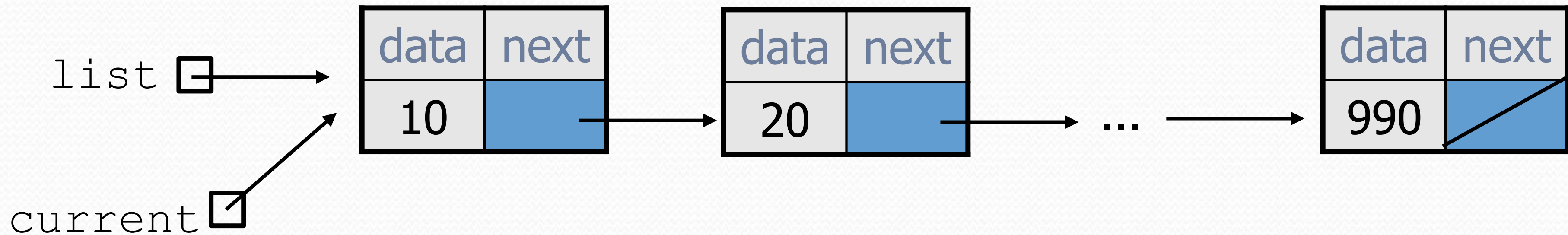
  - What's wrong with this approach?
    - (It loses the linked list as it prints it!)



A `current` reference

- Don't change `list`. Make another variable, and change it.
  - A `ListNode` variable is NOT a `ListNode` object

```
ListNode current = list;
```



- What happens to the picture above when we write:

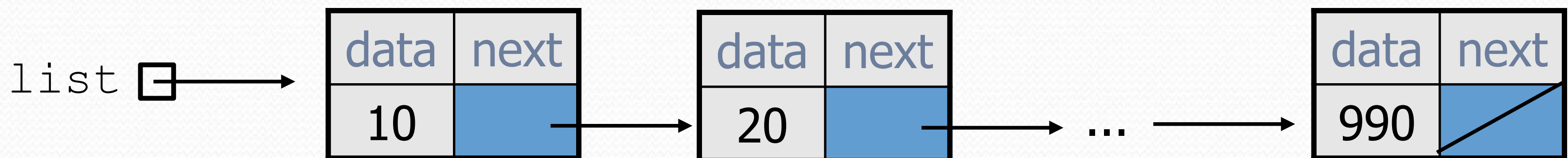
```
current = current.next;
```

Traversing a list correctly

- The correct way to print every value in the list:

```
ListNode current = list;
while (current != null) {
    System.out.println(current.data);
    current = current.next;  // move to next node
}
```

  - Changing `current` does not damage the list.



Linked List vs. Array

- Print list values:

```
ListNode list= ...;

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

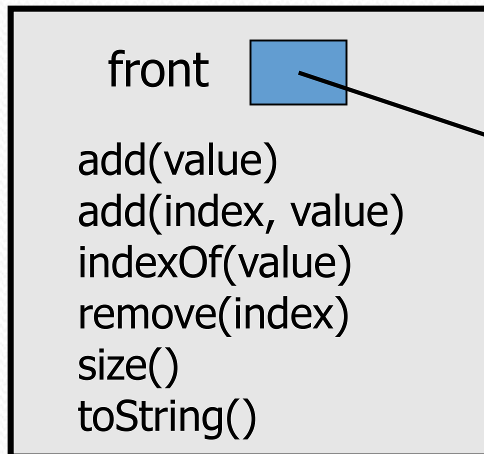
- Similar to array code:

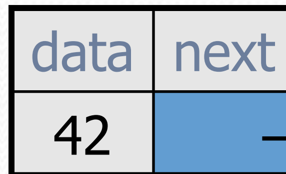
```
int[] a = ...;

int i = 0;
while (i < a.length) {
    System.out.println(a[i]);
    i = i + 1;
}
```

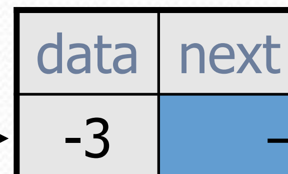
Description	Array Code	Linked List Code
Go to front of list	`int i = 0;`	`ListNode current = list;`
Test for more elements	`i < size`	`current != null`
Current value	`elementData[i]`	`current.data`
Go to next element	`i=i+1;`	`current = current.next;`

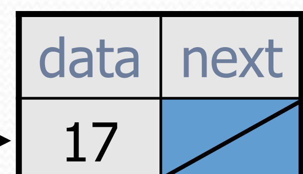
A `LinkedIntList` class

- Let's write a collection 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



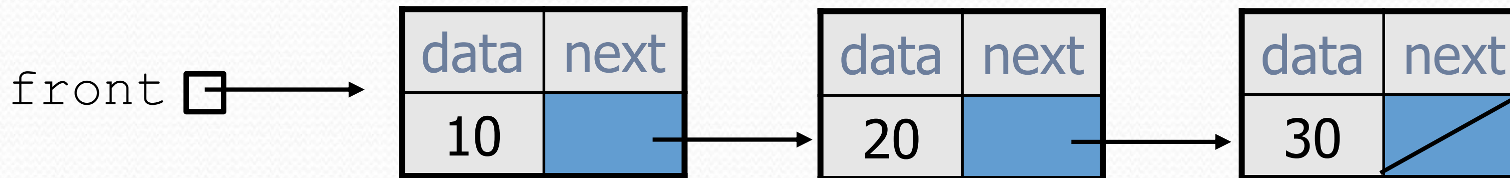






Poll Everywhere pollev.com/cse143

- Suppose our list had the contents



- Practice simulating the code we wrote and tell us what the result will look like when we call `list.add(40);`

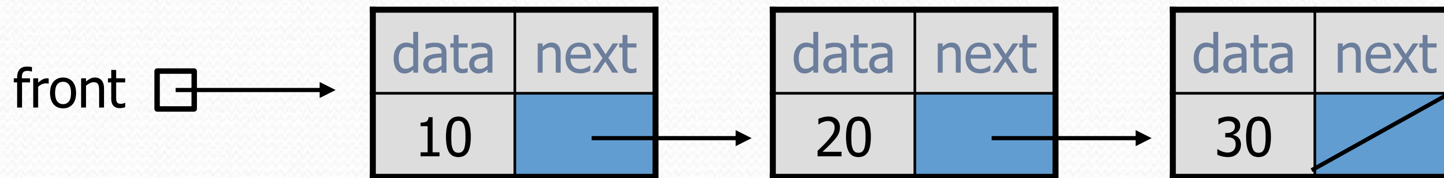
```
public void add(int value) {
    ListNode curr = front;
    while (curr != null) {
        curr = curr.next;
    }
    curr = new ListNode(value);
}
```

Options

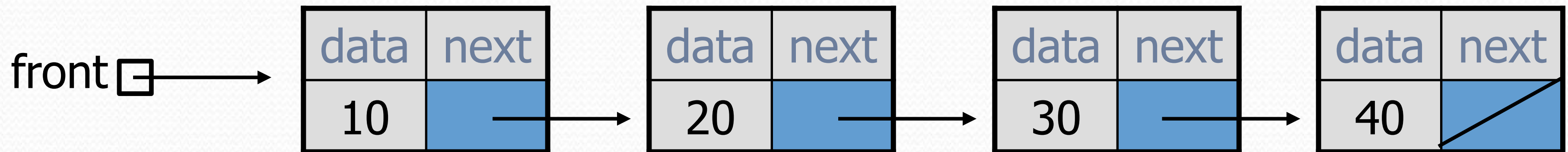
- [10, 20, 30]
- [10, 20, 40]
- [10, 20, 40, 30]
- [10, 20, 30, 40]
- Error

Before/After

- Before



- After



changing a list

- There are only two ways to change a linked list:
  - Change the value of `front` (modify the front of the list)
  - Change the value of `<node>.next` (modify middle or end of list to point somewhere else)

- Implications:
  - To add in the middle, need a reference to the *previous* node
  - Front is often a special case