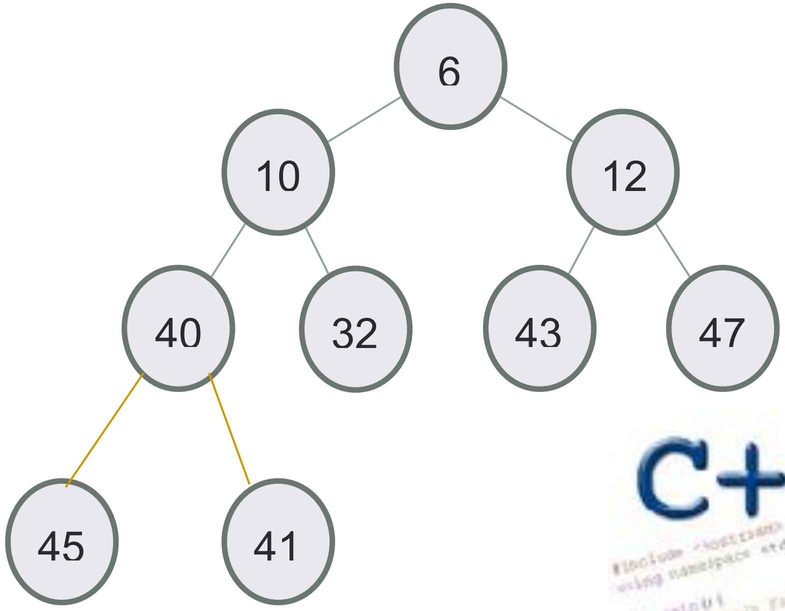
RECURSION

Problem Solving with Computers-I



Let recursion draw you in….

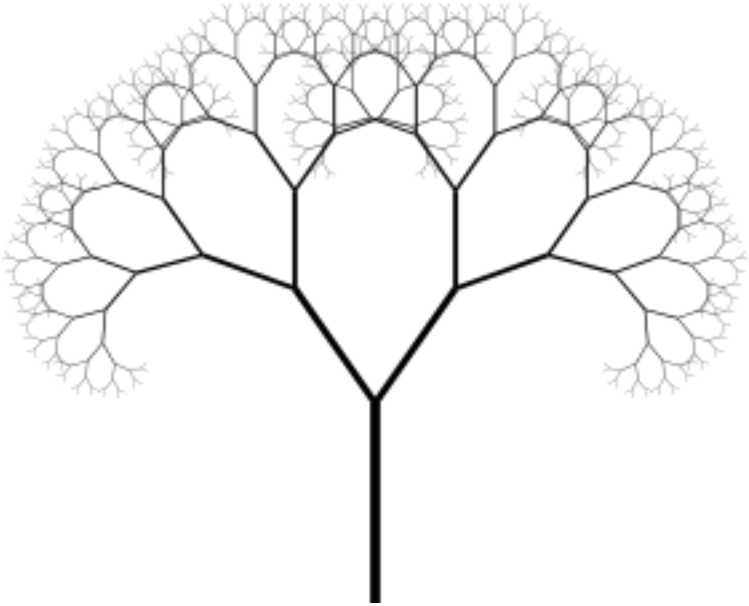
- Many problems in Computer Science have a recursive structure…
- Identify the “recursive structure” in these pictures by describing them

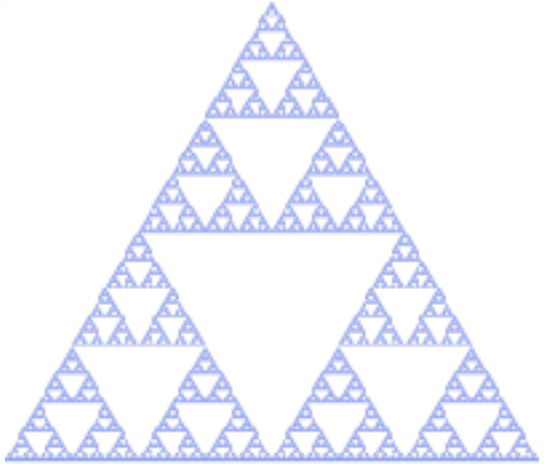
Understanding recursive structures

- Recursive names: The pioneers of open source and free software used clever recursive names

GNU IS NOT UNIX

- Recursive structures in fractals





Sierpinski triangle

Zooming into a Koch's snowflake

What was common to all these examples

A. Each can be described as smaller versions of itself
B. Each can be described as a collection of very different subparts
C. Each has an infinite instance of itself described within it
D. A and C

Why is recursion important in Computer Science

- Tool for solving problems (recursive algorithms)
- Solution is simply a recursive description of the problem
- Elegant (short and concise) algorithms
- Example of a recursive algorithm:

To wash the dishes in the sink:

Wash the dish on top of the stack

If there are no more dishes

you are done!

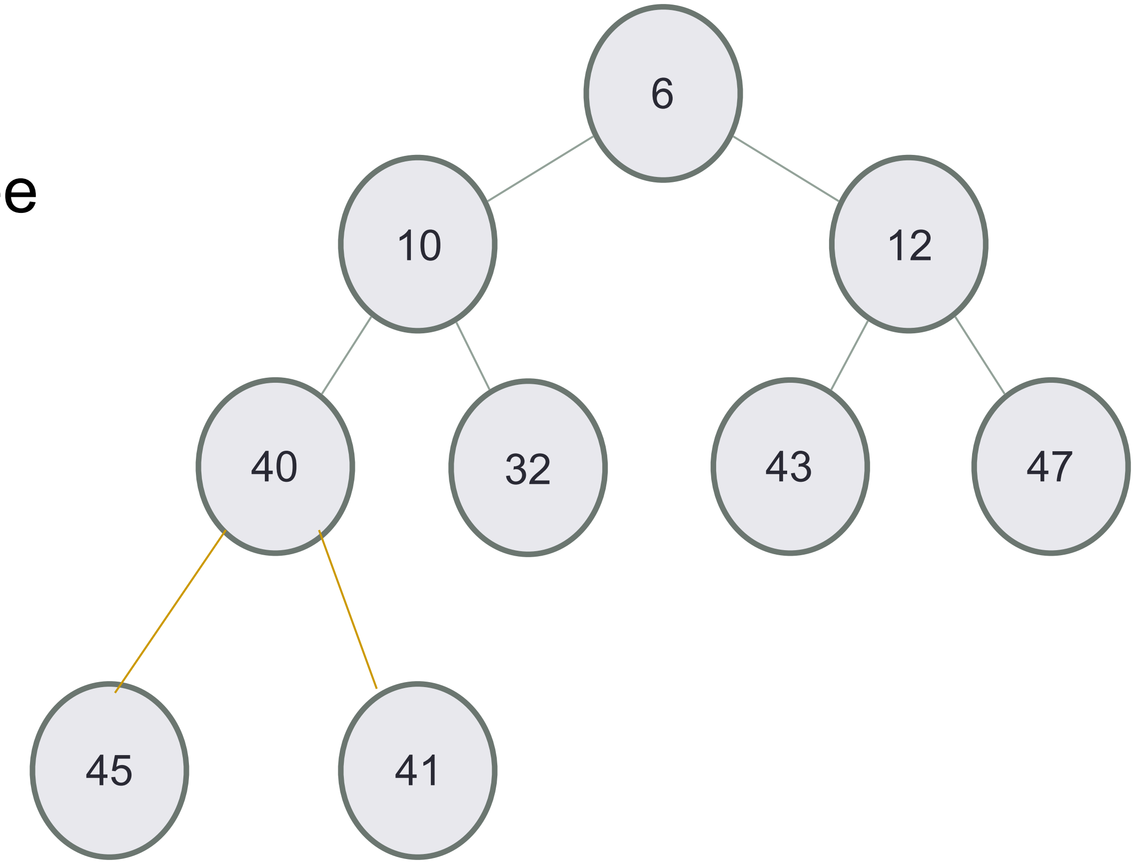
Else:

Wash the *remaining* dishes in the sink

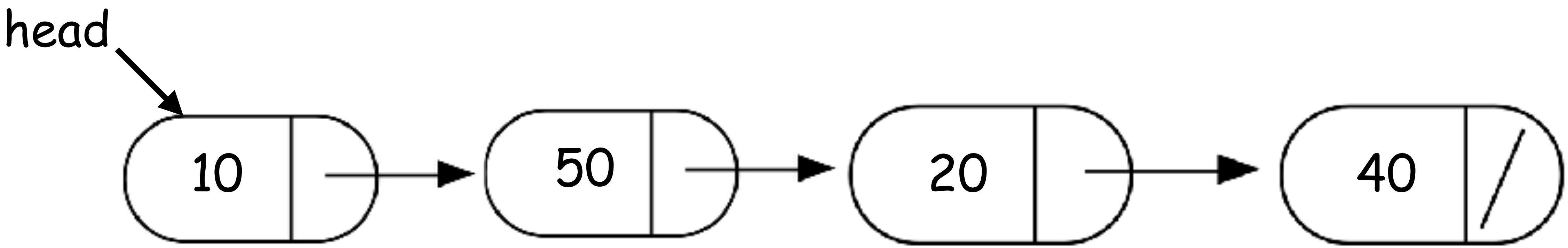
Examples from Computer Science

Ask questions about data structures that have a recursive structure like trees:

- Find the sum of all the elements in this tree
- Print all the elements in the tree
- Count the number of elements in this tree



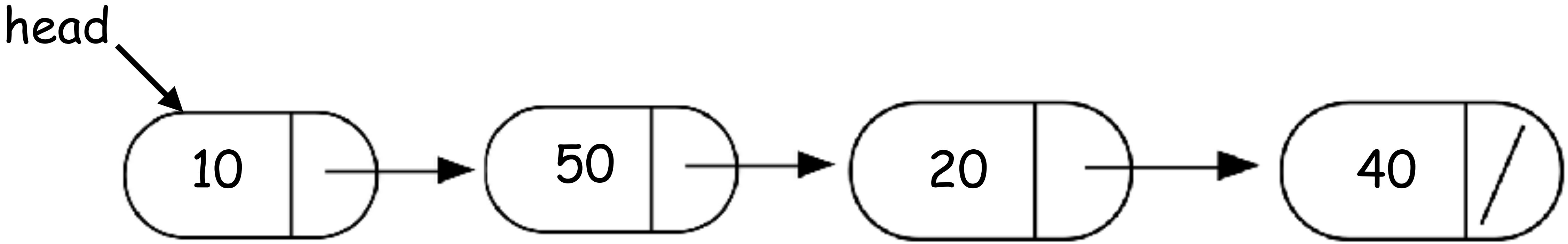
Recursive description of a linked list



- A non-recursive description of the linked list:
  A linked list is a chain of nodes

- A recursive description of a linked-list:
  A linked list is a node, followed by a smaller linked list

Sum all the elements in a linked list



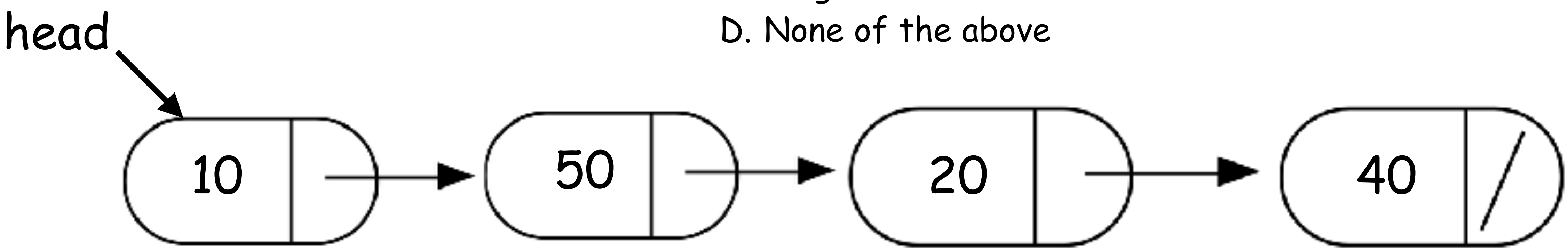
- A recursive description of a linked-list:
  A linked list is a node, followed by a smaller linked list

Sum of all the elements in a linked list is:
Value of the first node +
Sum of the all the elements in the *rest* of the list

Let's code it up

What happens when we execute this code on the example linked list?

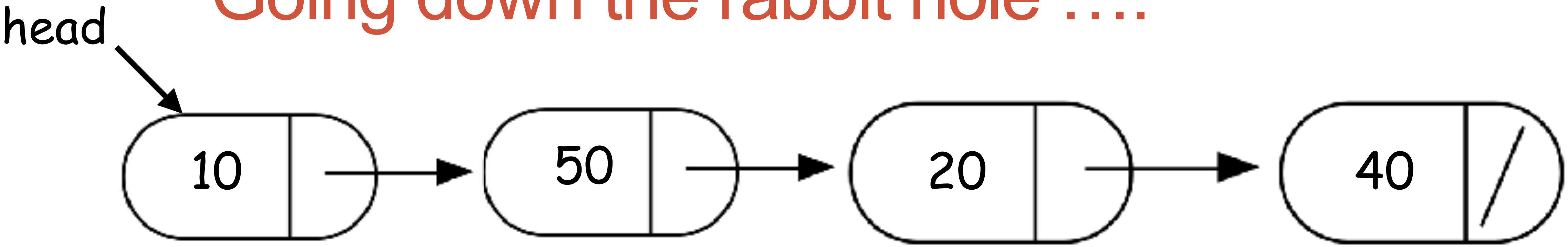
A. Returns the correct sum (120)
B. Program crashes with a segmentation fault
C. Program runs forever
D. None of the above




```
double sumList(Node* head){

    double sum = head->value + sumList(head->next);
    return sum;
}
```

Going down the rabbit hole ….




```
double sumList(Node* head){
    // Solve the smallest version of the problem
    // THE BASE CASE!!
   if(!head) return 0;
    // Go deeper into the rabbit hole!!
    // THE RECURSIVE CASE:
   double sum = head->value + sumList(head->next);
    // Come out of the rabbit hole
   return sum;
  }
```

Find the min element in a linked list

```
double min(Node* head){
    // Assume the linked list has at least one node
    assert(head);
    // Solve the smallest version of the problem
    // Write the BASE CASE



  }
```

Helper functions

- Sometimes your functions takes an input that is not easy to recurse on
- In that case define a new function with appropriate parameters: This is your helper function
- Call the helper function to perform the recursion

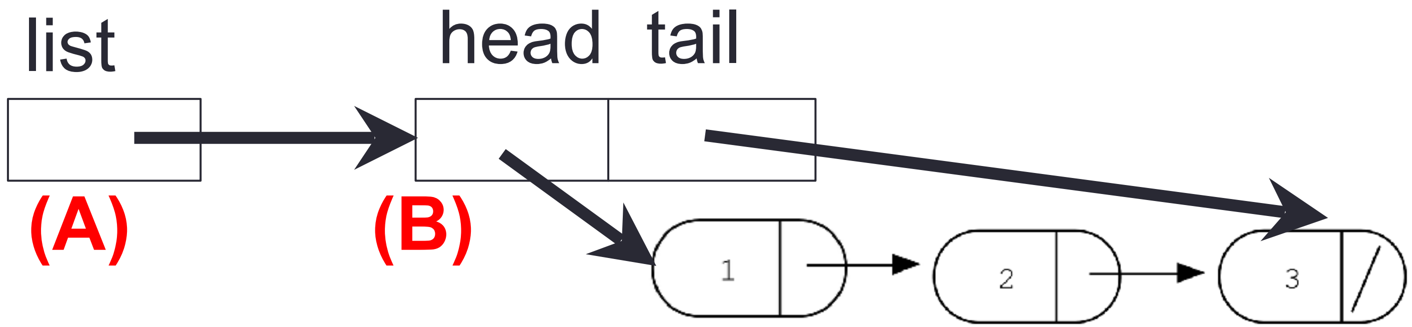
For example

```
double sumList(LinkedList* list){
    // How would you change the following code
    // for the new input?
    double sum = head->value + sumList(head->next);
    return sum;
}
```

Deleting the list

```
int deleteList(LinkedList * list){
   delete list;
}
```

Which data objects are deleted when the above function is called on the linked list shown below:

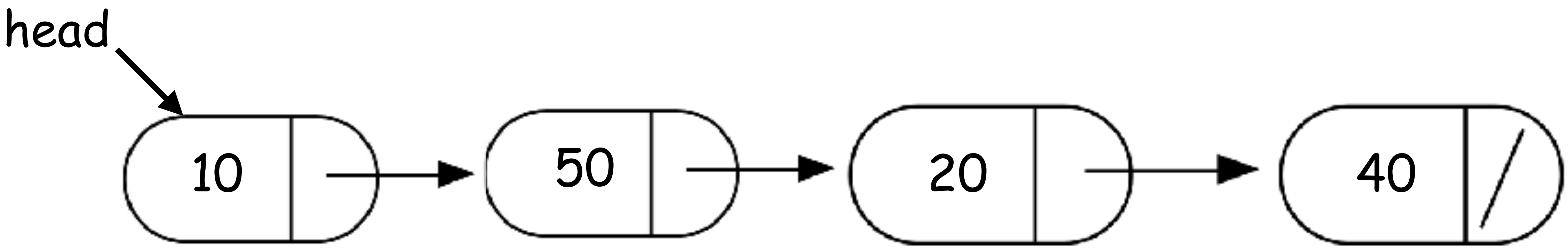


(C) All nodes of the linked list

(D) B and C
(E) All of the above

Does this result in a memory leak?

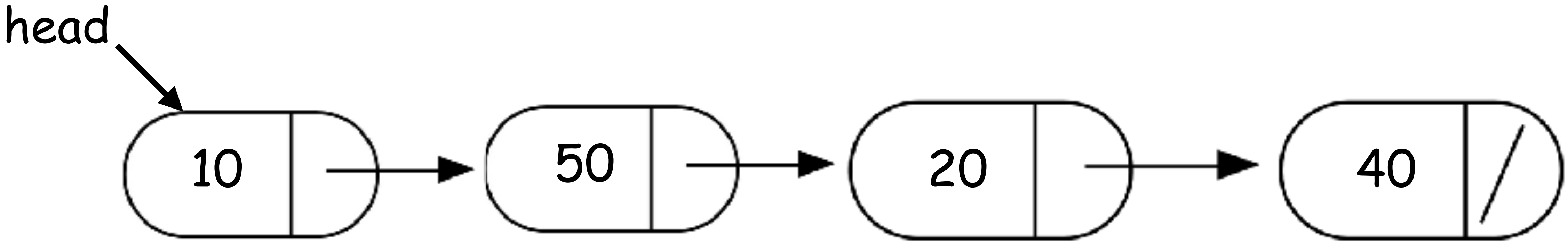
Delete a node in a linked list



Given: a pointer to the first node
: a value to delete from the list

Write code to iteratively delete the node

Delete a node recursively



Given: a pointer to the first node

: a value to delete from the list

Next time

- Final review and wrap up