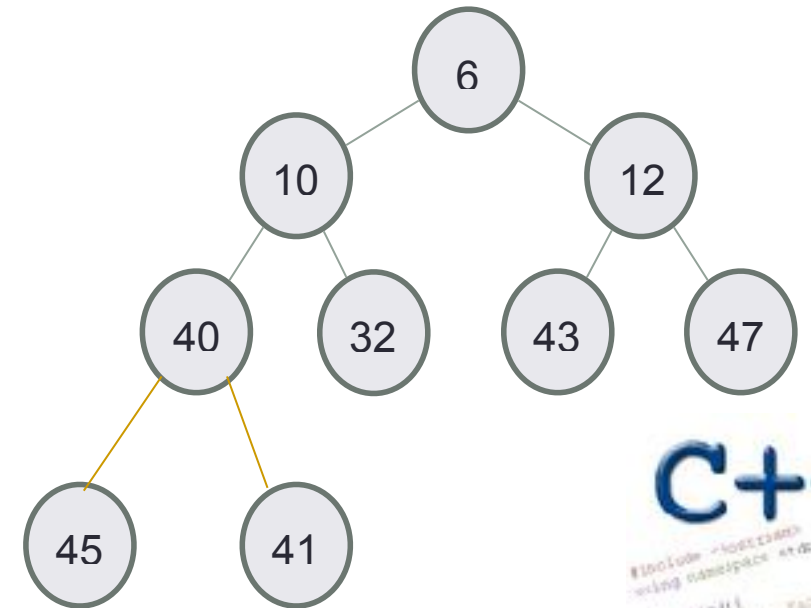


RECURSION

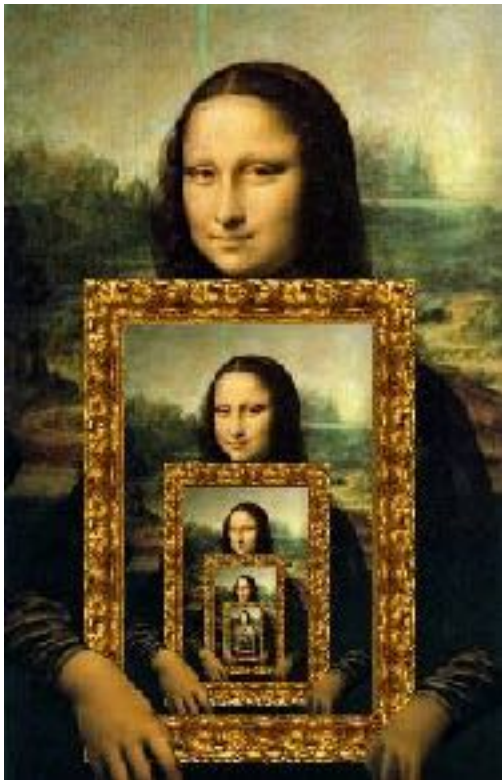


Problem Solving with Computers-I



Let recursion draw you in....

- Identify the “recursive structure” in these pictures by describing them

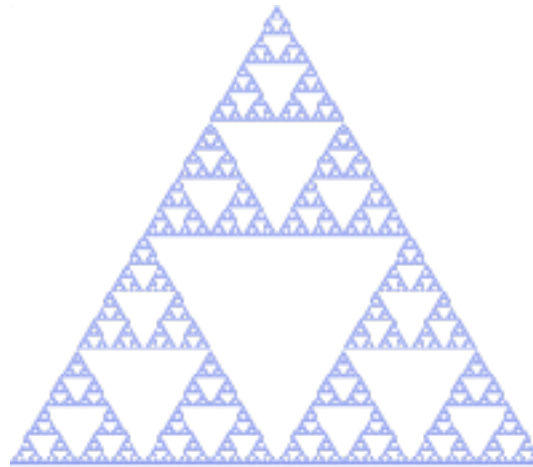
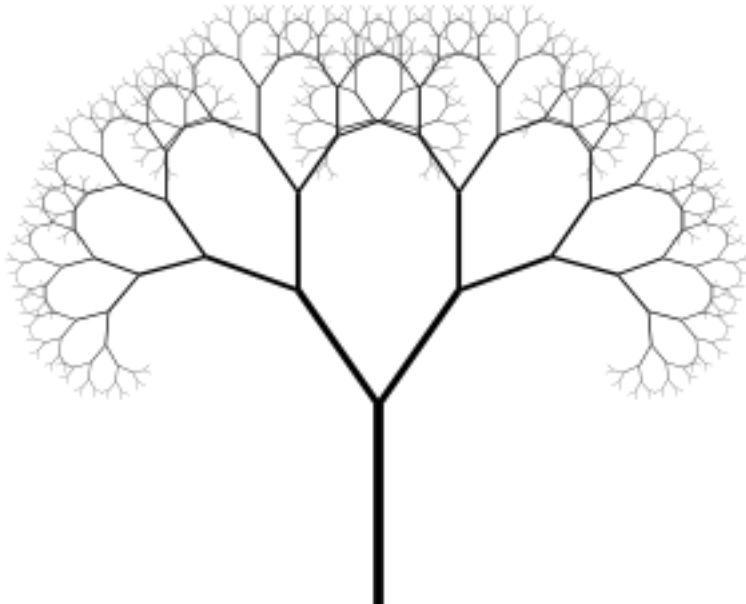


Understanding recursive structures

- **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

Why is recursion important in Computer Science

Tool for solving problems (recursive algorithms)

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:

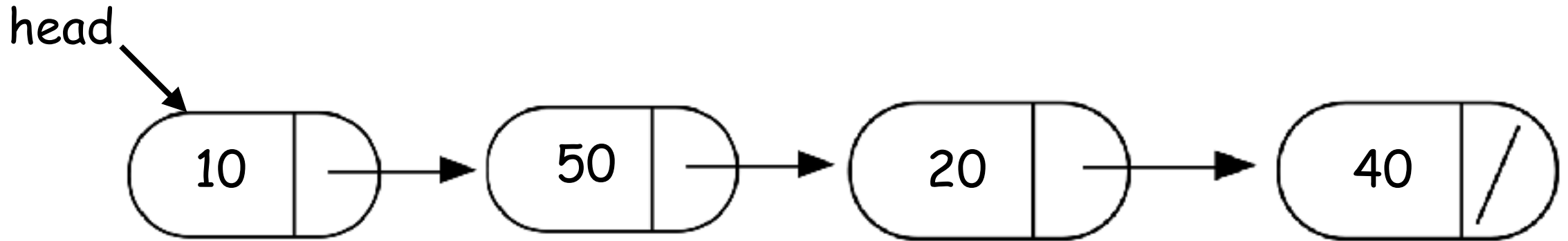
Wash the *remaining* dishes in the sink

A new way of looking at inputs

Arrays:

- Non-recursive description: **a sequence of elements**
- Recursive description: **an element, followed by a smaller array**

Recursive description of a linked list



- Non-recursive description of the linked list: **chain of nodes**
- Recursive description of a linked-list: **a node, followed by a smaller linked list**

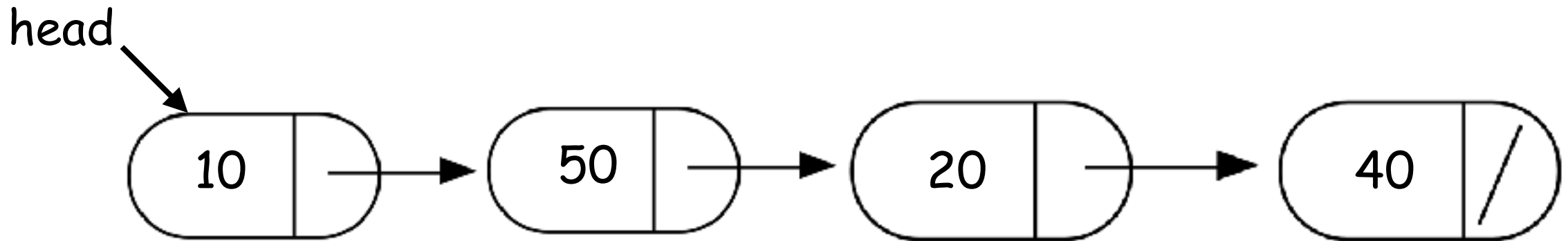
Designing recursive code: print all the elements of an array

Arrays:

- Recursive description: **an element, followed by a smaller array**

Designing recursive code: sum elements in a linked-list

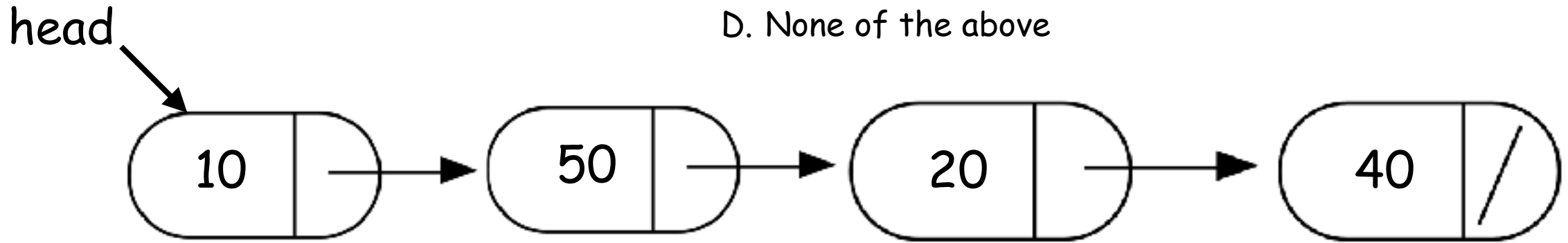
- Recursive description of a linked-list: **a node, followed by a smaller linked list**



What's in a base case?

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

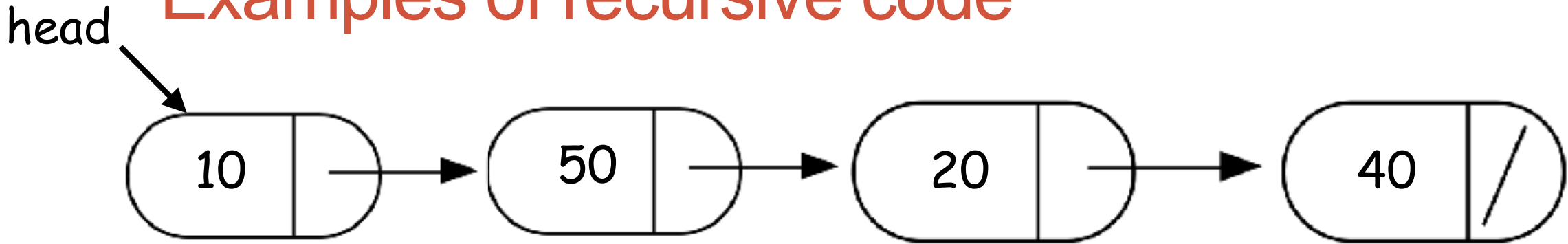


```
double sumList(Node* head) {
```

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

```
}
```

Examples of recursive code



```
double sumList(Node* head){  
    if(!head) return 0;
```

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

```
    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  
  
}
```

See code written in lecture for the complete solution

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 sumLinkedList(LinkedList* list){  
    return sumList(list->head); //sumList is the helper  
    //function that performs the recursion.  
}
```