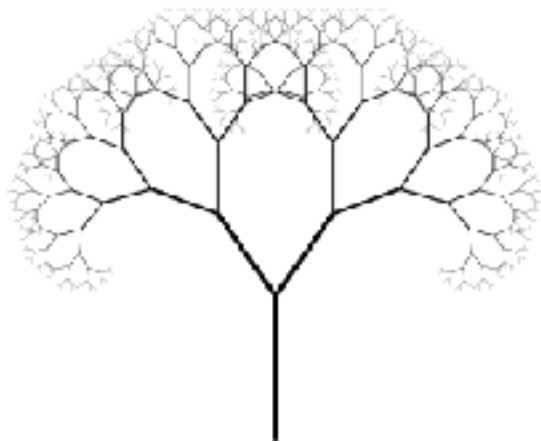


# RECURSION, WRAP UP

---

Problem Solving with Computers-I



# String problems

1. Count the number of vowels in a C string
2. Remove all the spaces from a C++ string

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

```
    double sumRest;
```

```
    sumRest = sumList(head->next);
```

```
    return head->data + sumRest;
```

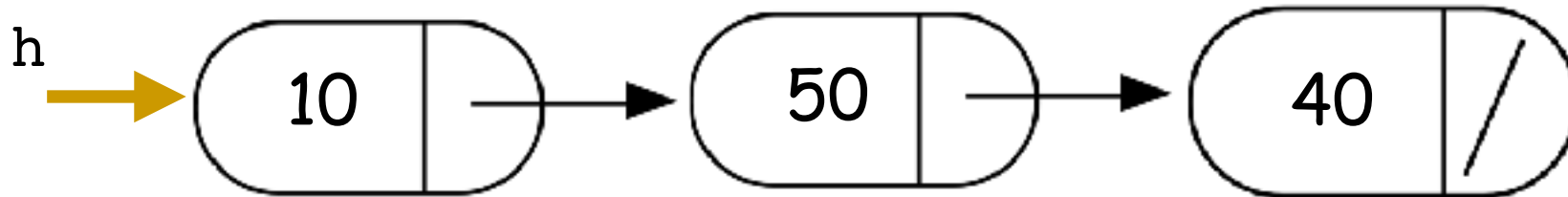
```
}
```



head



sumRest



**Imagine each instance of sumList to be a doll!**

**Calling sumList is like creating a new doll.**

**First call to sumList!**

```
double s=sumList(h);
```



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

```
    double sumRest;
```

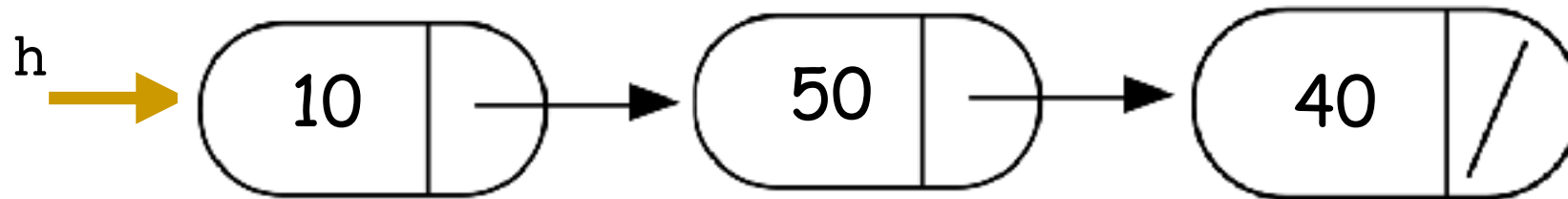


```
    sumRest = sumList(head->next);
```

```
    return head->data + sumRest;
```

```
}
```

The turtle tells us which line of code is going to be executed



head

sumList(1)

First call to sumList!

```
double s=sumList(h);
```

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

```
    double sumRest;
```

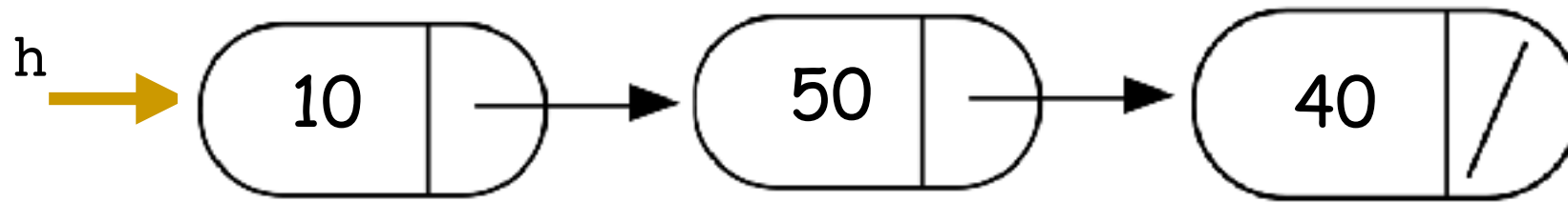
```
    sumRest = sumList(head->next);
```

```
    return head->data + sumRest;
```

```
}
```



Second call to sumList!



head



sumRest

First call to sumList!

```
double s=sumList(h);
```

sumList(1)

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

```
    double sumRest;
```

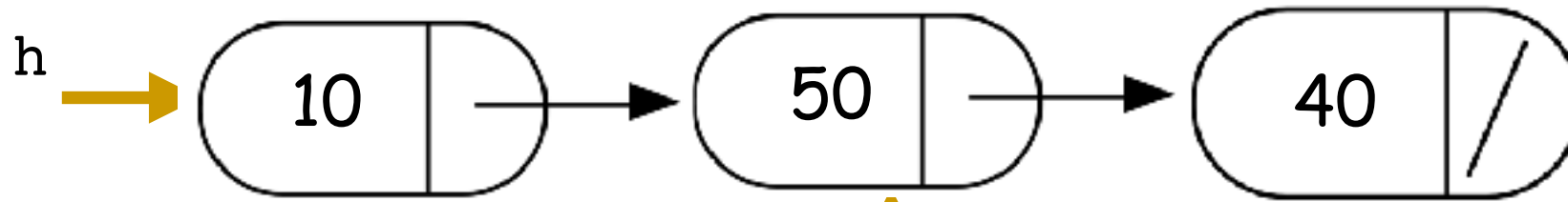


```
    sumRest = sumList(head->next);
```

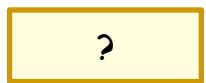
```
    return head->data + sumRest;
```

```
}
```

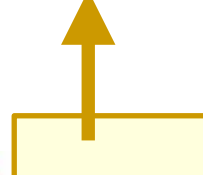
Turtle is going to execute the first line of sumList(2)



head



sumRest



head

**sumRest =  
sumList(head->next);**

sumList(1)

sumList(2)



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

```
    double sumRest;
```

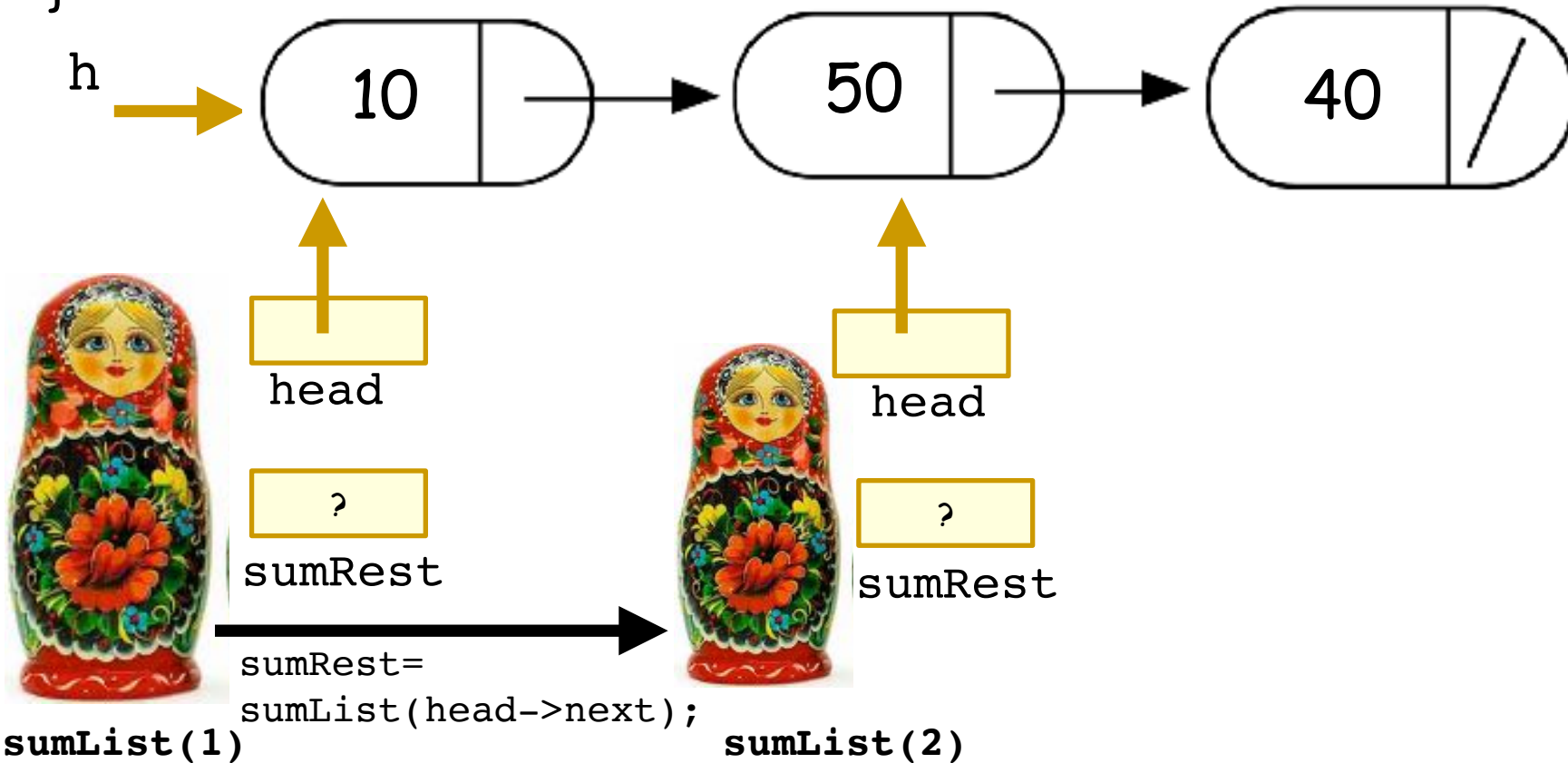
```
    sumRest = sumList(head->next);
```

```
    return head->data + sumRest;
```

```
}
```



Third call to sumList



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

```
    double sumRest;
```

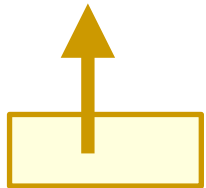
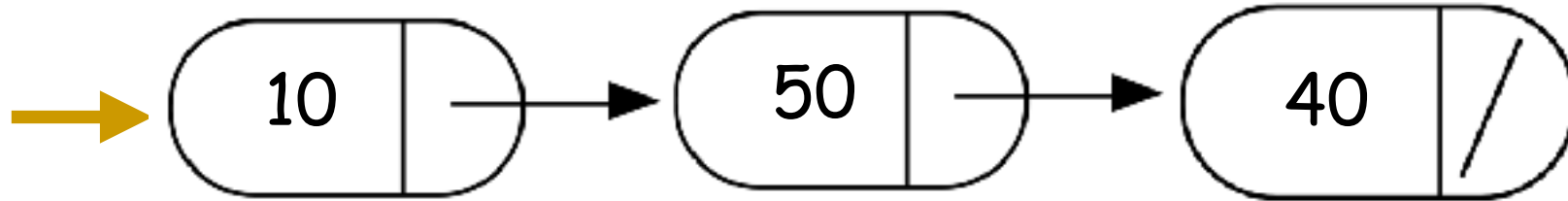


```
    sumRest = sumList(head->next);
```

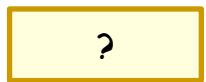
```
    return head->data + sumRest;
```

```
}
```

h



head

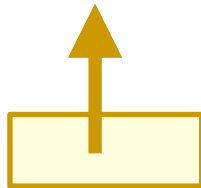


sumRest

sumRest =

sumList(head->next);

sumList(1)



head

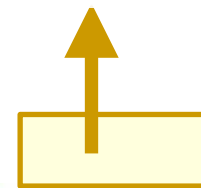


sumRest

sumRest =

sumList(head->next);

sumList(2)



head



sumList(3)



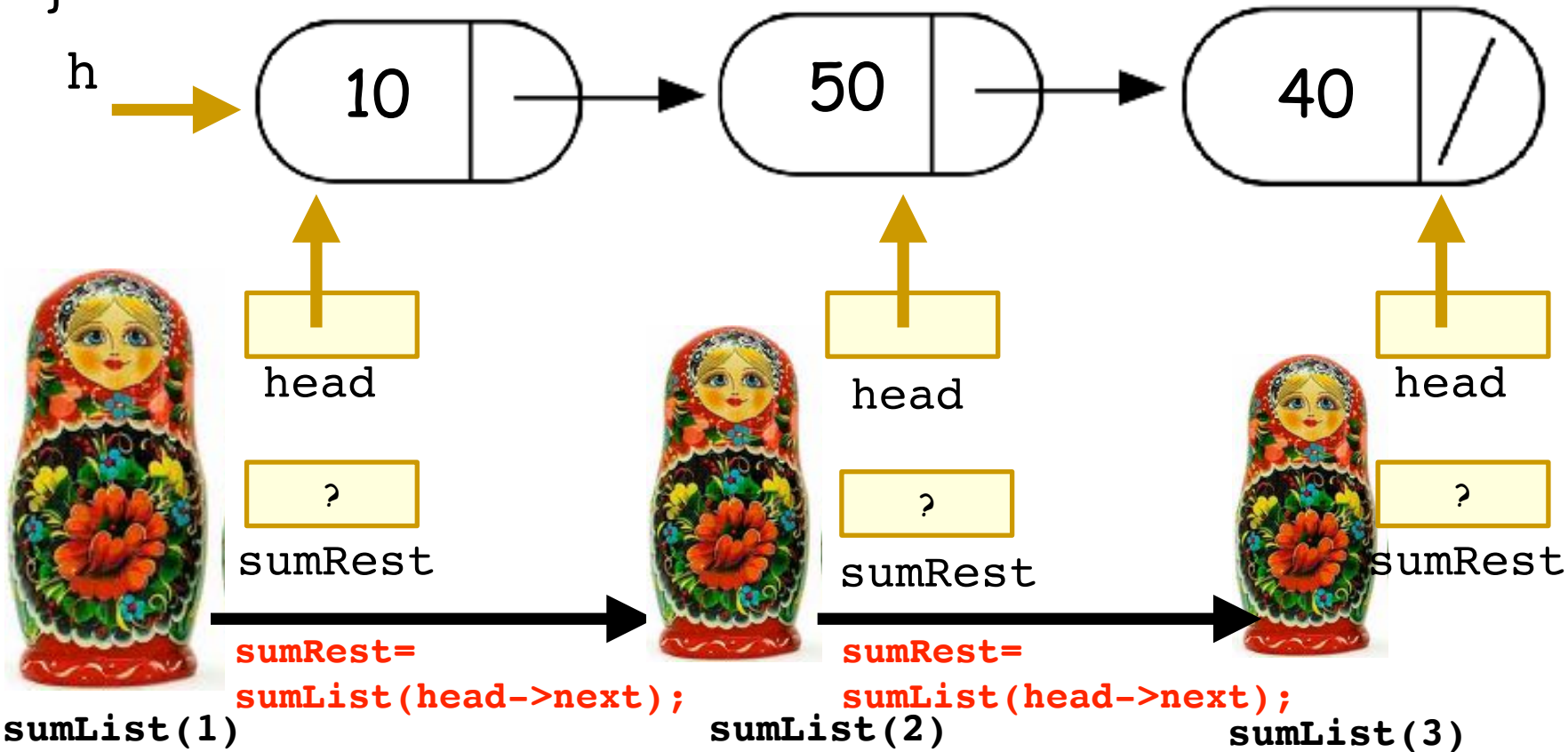
```
double sumList(Node* head){
```

```
    double sumRest;
```


```
    sumRest = sumList(head->next);
```

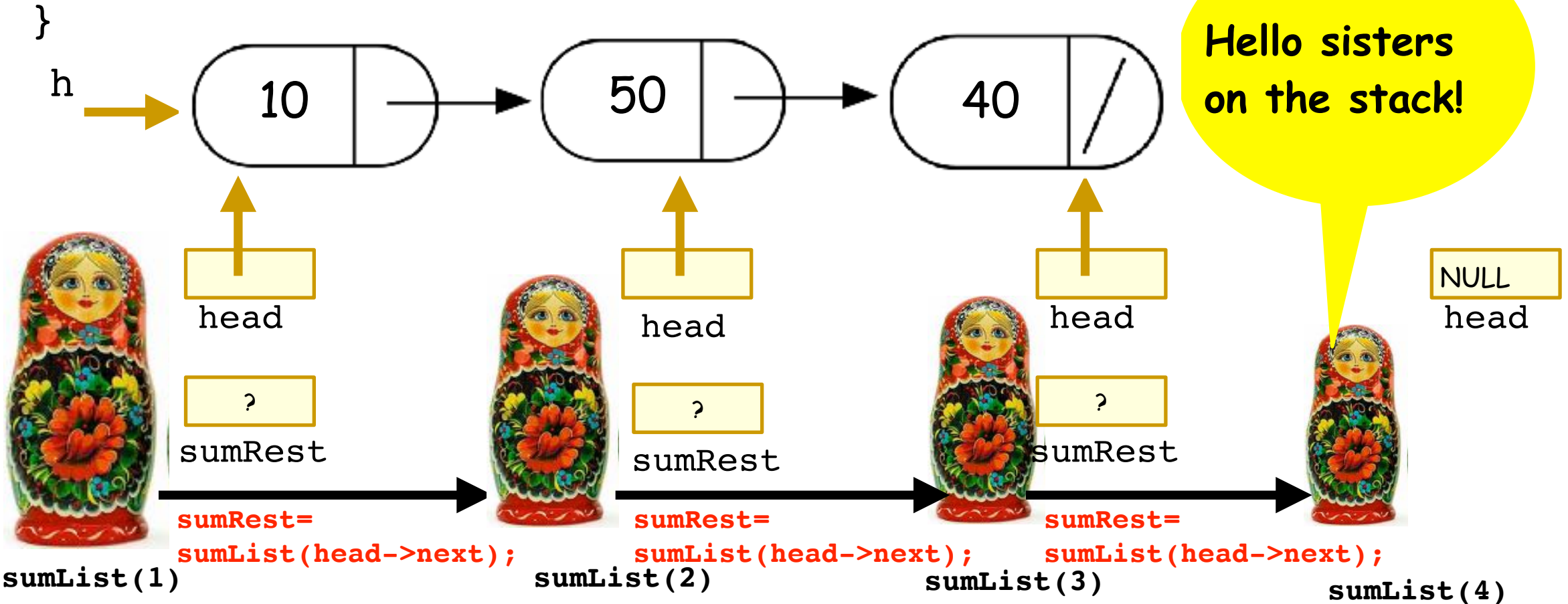
```
    return head->data + sumRest;
```

```
}
```



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

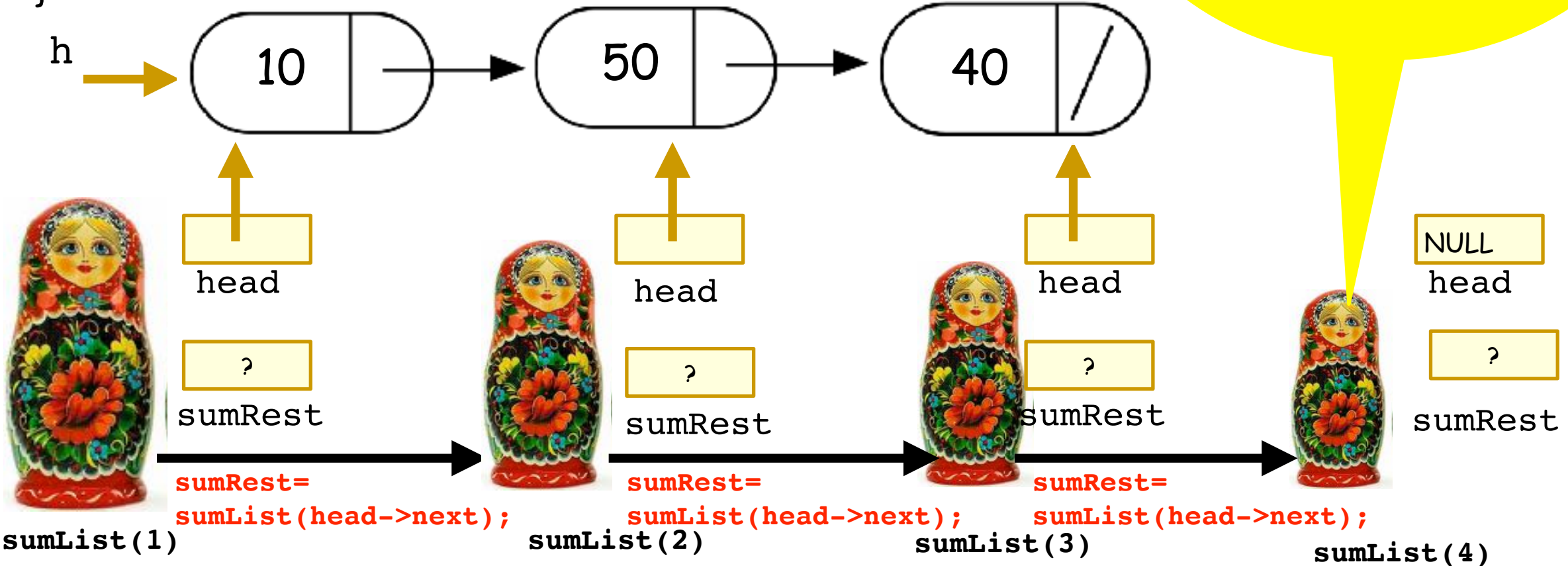
```
    double sumRest;   
    sumRest = sumList(head->next);  
    return head->data + sumRest;  
}
```



```
double sumList(Node* head){
    double sumRest;
    sumRest = sumList(head->next);
    return head->data + sumRest;
}
```



Oops my head is null and turtle is about to dereference it !!!!

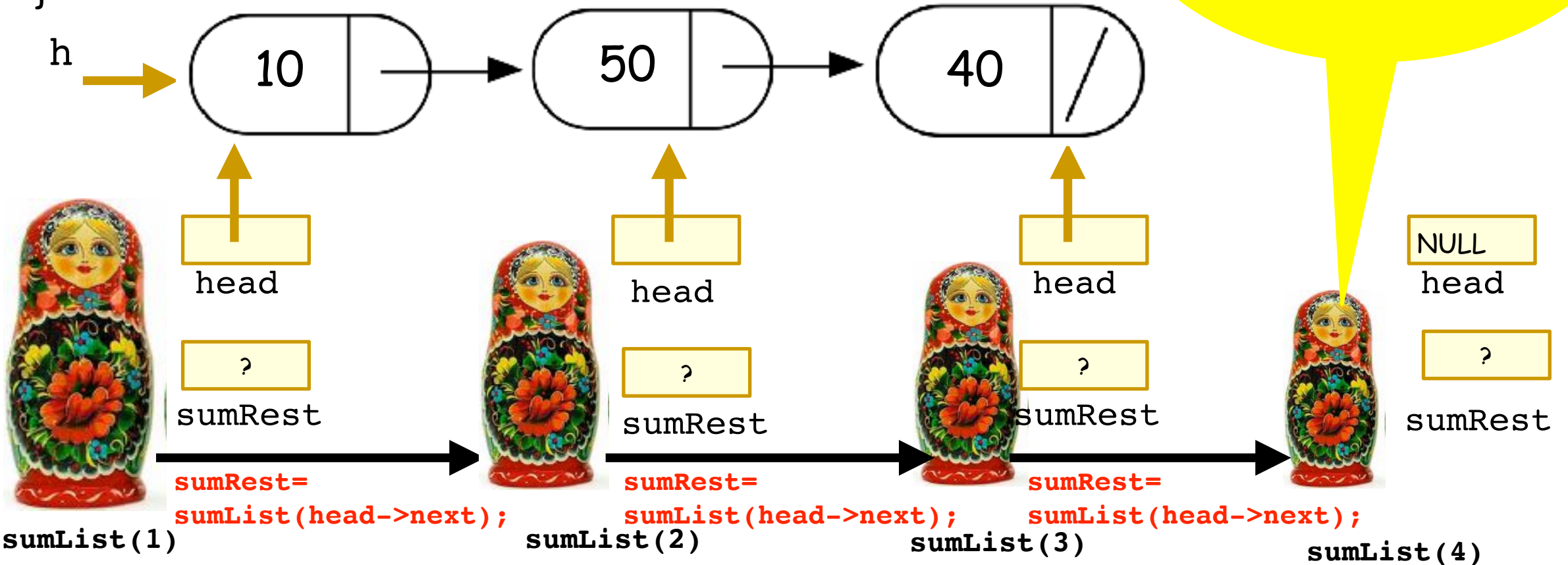


```
double sumList(Node* head){
```

```
    double sumRest;  
    sumRest = sumList(head->next);  
    return head->data + sumRest;  
}
```



No turtle noooooooo !!!!



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

```
double sumRest;
```

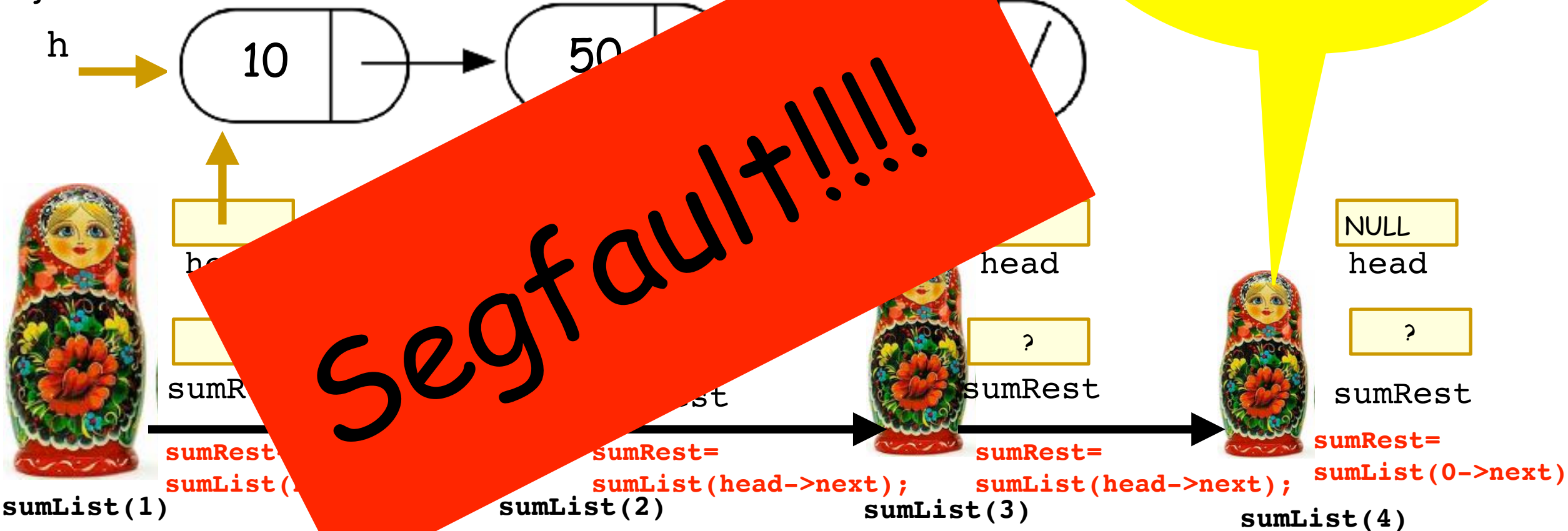
```
sumRest = sumList(head->next);
```

```
return head->data + sumRest;
```

```
}
```

Sorry I just follow instructions!

No turtle noooooo !!!!

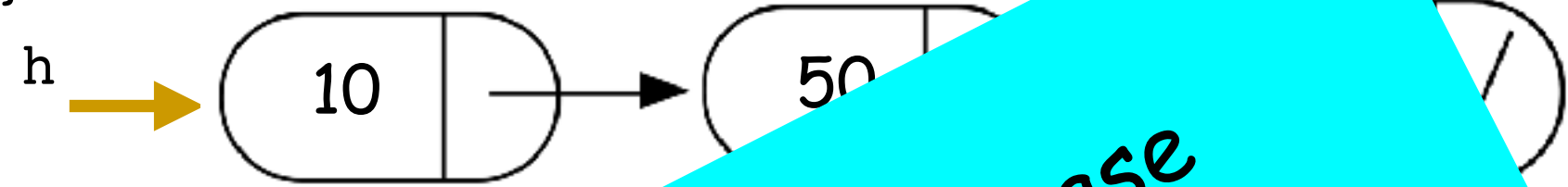


segfault!!!!

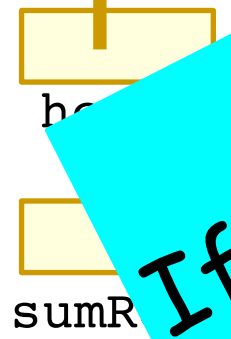
```
double sumList(Node* head){
    double sumRest;
    sumRest = sumList(head->next);
    return head->data + sumRest;
}
```

Sorry I just follow instructions!

No turtle noooooo !!!!



sumList(1)



```
sumRest = sumList(head->next);
```

If only we had a base case....



```
sumRest = sumList(head->next);
```



```
sumRest = sumList(head->next);
```



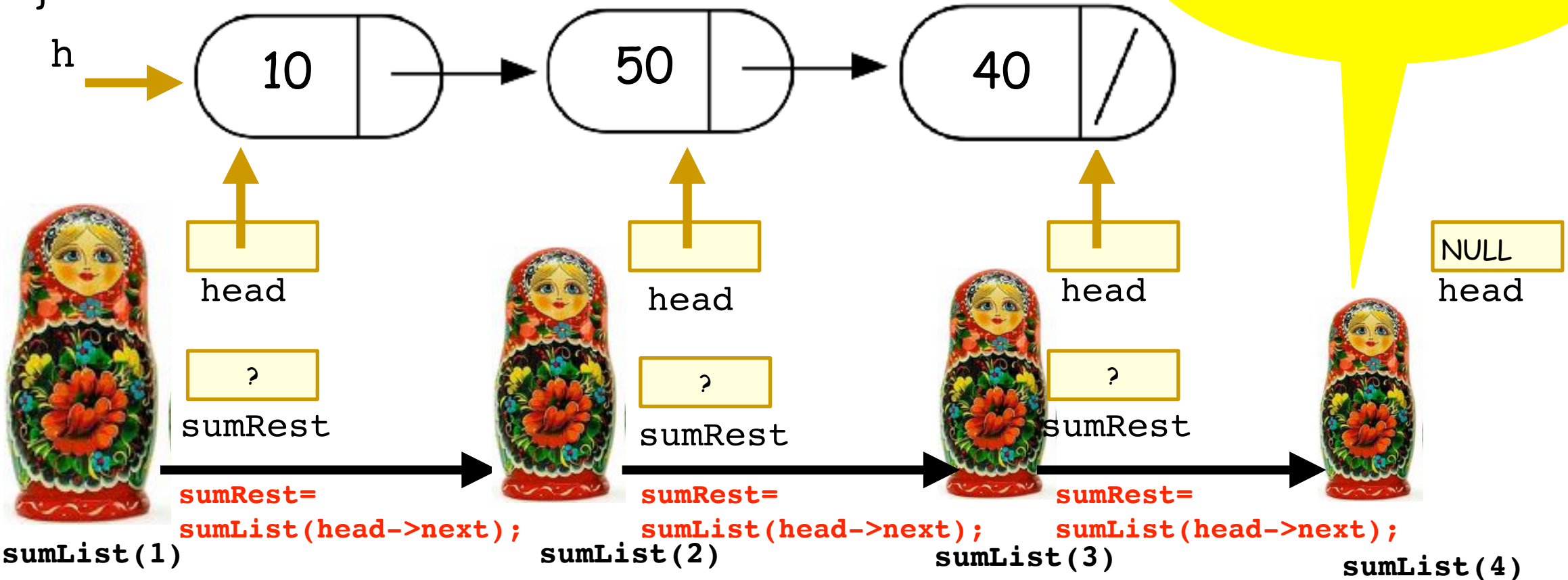
```
sumRest = sumList(0->next);
```

sumList(4)

```
double sumList(Node* head){
    if(head==0) return 0;
    double sumRest;
    sumRest = sumList(head->next);
    return head->data + sumRest;
}
```

I am really well behaved around base cases :)

return 0;

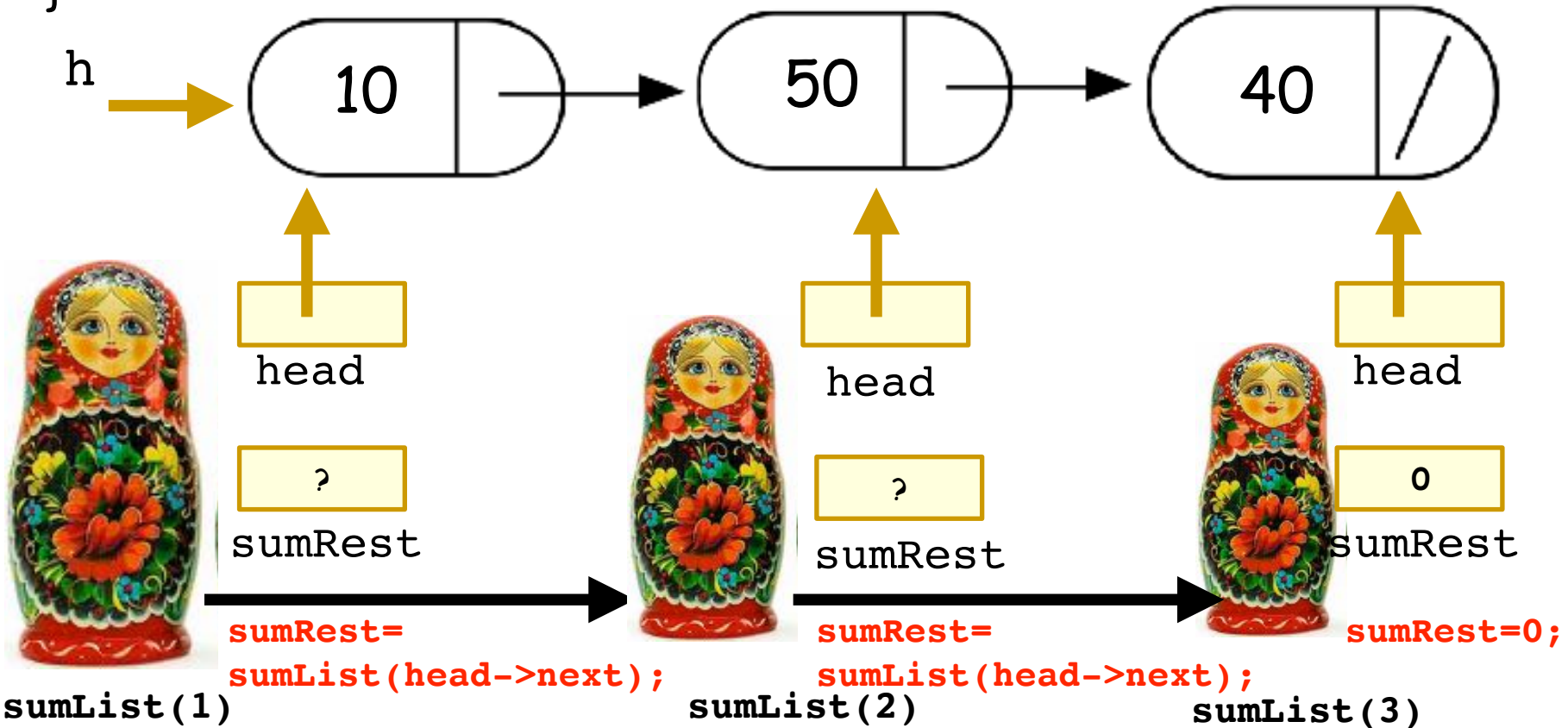


```

double sumList(Node* head){
    if(head==0) return 0;
    double sumRest;
    sumRest = sumList(head->next);
    return head->data + sumRest;
}

```

Hello again sumlist(3)! Your younger sister hit the base case.



`sumList(1)`

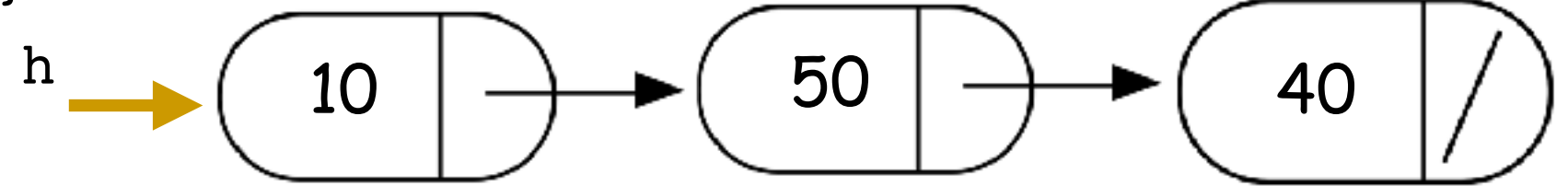
`sumList(2)`

`sumList(3)`

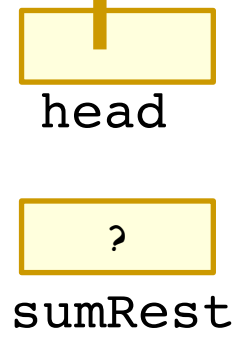


```
double sumList(Node* head){
  if(head==0) return 0;
  double sumRest;
  sumRest = sumList(head->next);
  return head->data + sumRest;
}
```

Turtle takes it one line at a time.

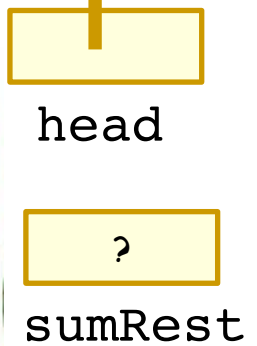


return 40+0;



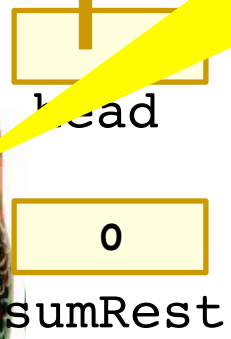
sumRest = sumList(head->next);

sumList(1)



sumRest = sumList(head->next);

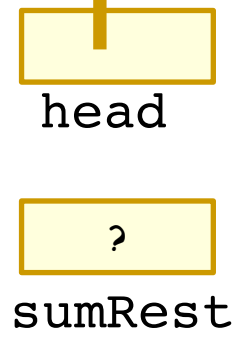
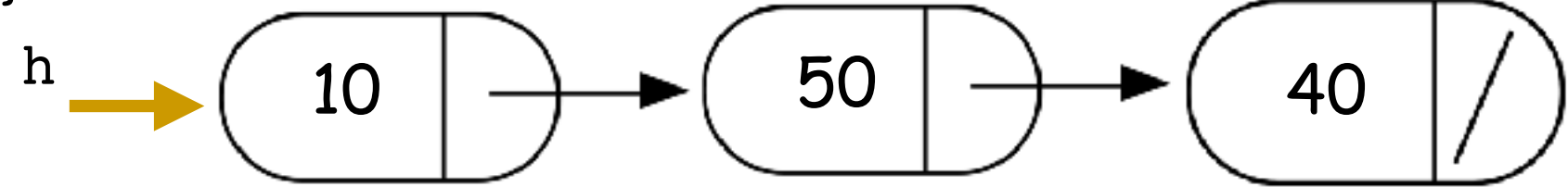
sumList(2)



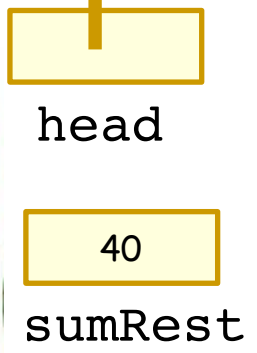
sumList(3)

```
double sumList(Node* head){
  if(head==0) return 0;
  double sumRest;
  sumRest = sumList(head->next);
  return head->data + sumRest;
}
```

Hello again Sumlist(2)! Your younger sister returned 40.



sumList(1)

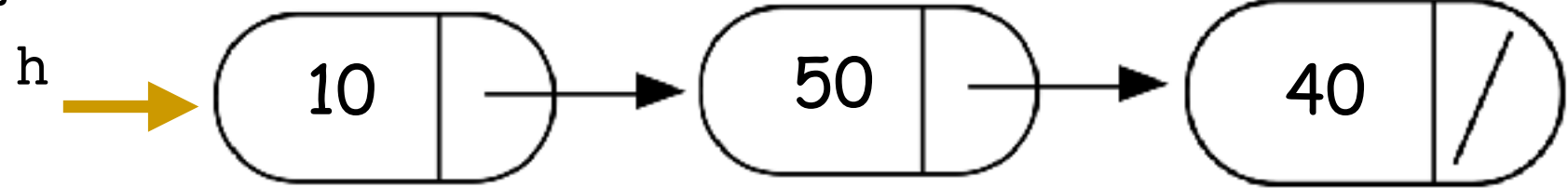


sumList(2)

```
return head->data + sumRest;
```

```
double sumList(Node* head){
  if(head==0) return 0;
  double sumRest;
  sumRest = sumList(head->next);
  return head->data + sumRest;
}
```

Any last words, sumList(2)?



head  
sumRest

sumList(1)



head  
sumRest

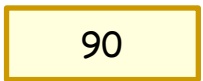
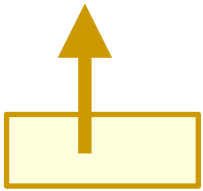
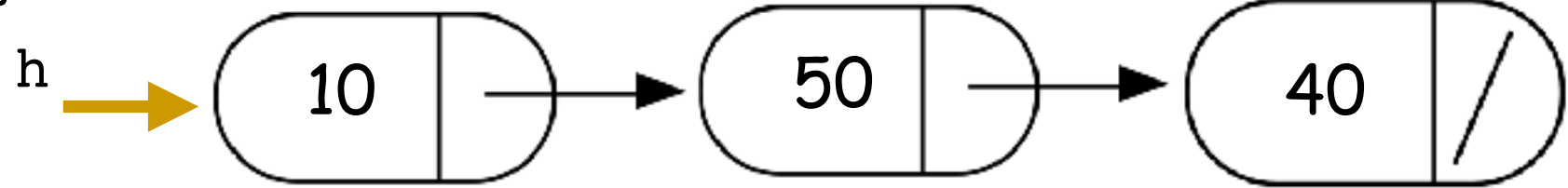
sumList(2)

return 50+40;

```
return head->data + sumRest;
```

```
double sumList(Node* head){
  if(head==0) return 0;
  double sumRest;
  sumRest = sumList(head->next);
  return head->data + sumRest;
}
```

Hello again sumList(1)!  
Your sisters are no longer on the stack,  
here is your 90, store it safely!



sumRest

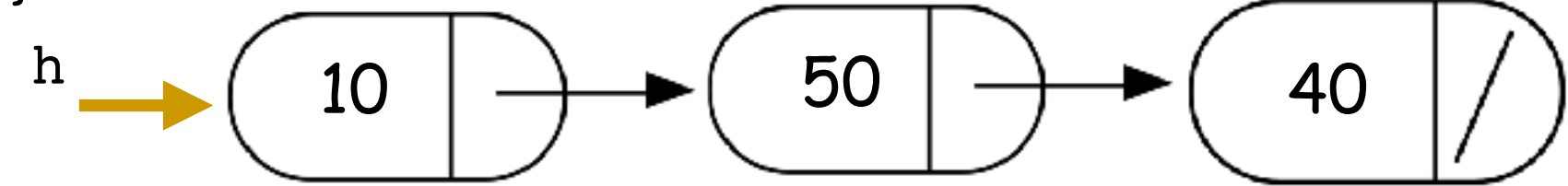


```
return head->data + sumRest;
```

sumList(1)

```
double sumList(Node* head){
  if(head==0) return 0;
  double sumRest;
  sumRest = sumList(head->next);
  return head->data + sumRest;
}
```

How did I get into this line of work, so many goodbyes make me want to cry.

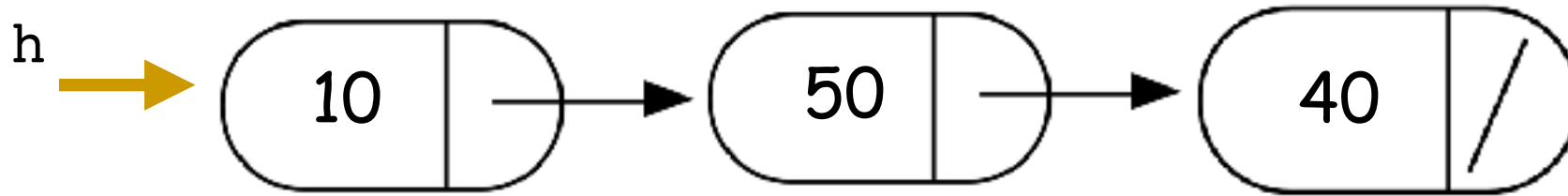


sumRest

**return 10+90;**  
B'bye and thanks for all the computation!

sumList(1)

```
double sumList(Node* head){
    if(head==0) return 0;
    double sumRest;
    sumRest = sumList(head->next);
    return head->data + sumRest;
}
```



**You have  
no idea how many calls I had  
to make to sum this list!**

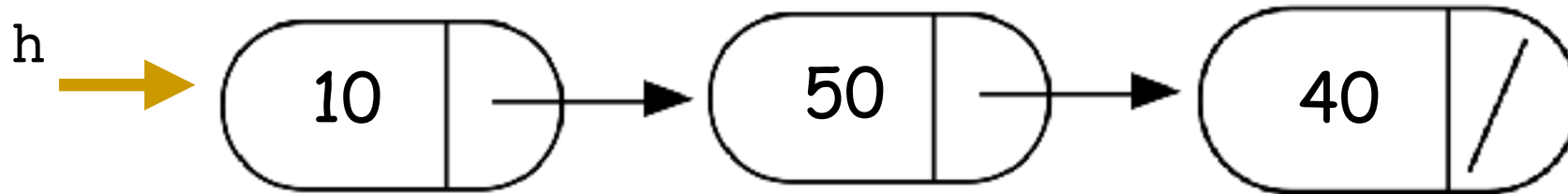
```
double s=sumList(h);
```



```
double sumList(Node* head){
    if(head==0) return 0;
    double sumRest;
    sumRest = sumList(head);
    return head->data + sumRest;
}
```

What happens when we call sumList on the example linked list?

- A. Returns the correct sum (100)
- B. Program crashes with a segmentation fault
- C. Program runs for a while, then crashes**
- D. None of the above



```
double s=sumList(h);
```



# More Recursion Examples

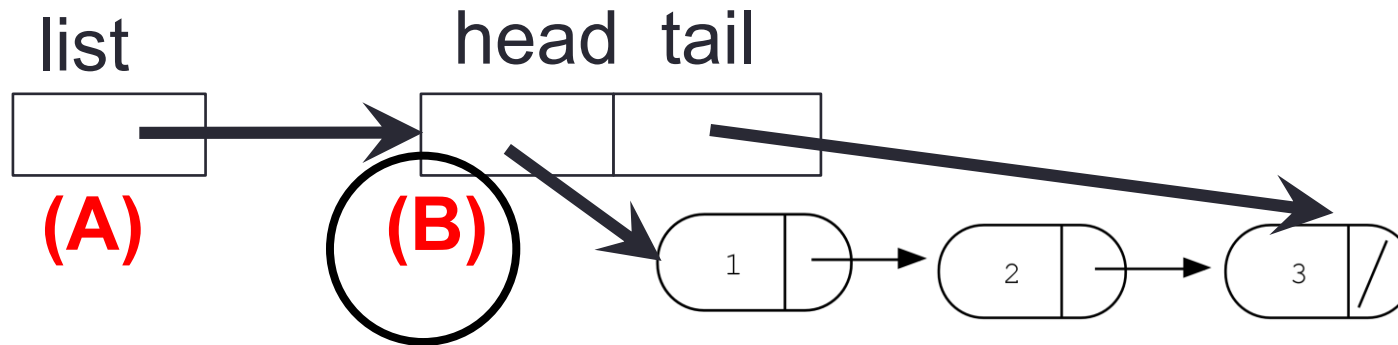
- Calculate the nth element of the Fibonacci sequence
  - Why is the recursive solution inefficient?
- Calculate  $a^b$  when b is a non-negative int
- Binary tree sum of nodes
  - This is a more complicated example involving a data structure that we don't cover in this class. I won't test you on this!



# Deleting the list

```
int freeLinkedList(LinkedList * list){...}
```

Which data objects are deleted by the statement: **delete list;**



**(C)** All nodes of the linked list

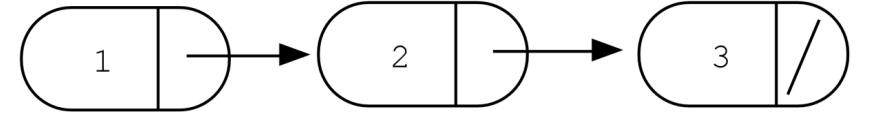
**(D)** B and C

**(E)** All of the above

Does this result in a memory leak?

# Recursion on lists: delete a value recursively

```
void deleteNodeRecursive(LinkedList *list, int value)
```



```
Node* deleteNodeRecursiveHelper(Node *head, int value)
```



why does this need to return a Node\*?

Recall the steps towards a recursive solution

The remaining slides are about concepts that I will not test you on. You'll need to know them for CS 24, though!

# Searching for a value in a sorted array

10	20	30	40	50	60	70	80
0	1	2	3	4	5	6	7

Binary search!

Start in the middle (make a guess), and keep halving your search space as you update your guess with new information.

# Find the square root of a number

Same concept!

max of 1  
and the

Guess somewhere between 0 and the number, and keep on updating your guess (and halving the search space). You will eventually approach the answer.

How do you check if your guess was too big or too small?

# Binary/Hex Arithmetic

- Calculate  $101_2 + 001_2$
- Calculate  $10101101_2 + 10001111_2$ 
  - What if our ints were only 8 bits long?
- Calculate  $2A_{16} + B8_{16}$
- What about negative numbers?!

# Bitwise operations

- and ( & )
- or ( | )
- not ( ~ )
- exclusive or (xor) ( ^ )
- shift left and right ( << and >> )

## Bitwise AND `&` [\[edit\]](#)

bit a	bit b	a & b (a AND b)
0	0	0
0	1	0
1	0	0
1	1	1

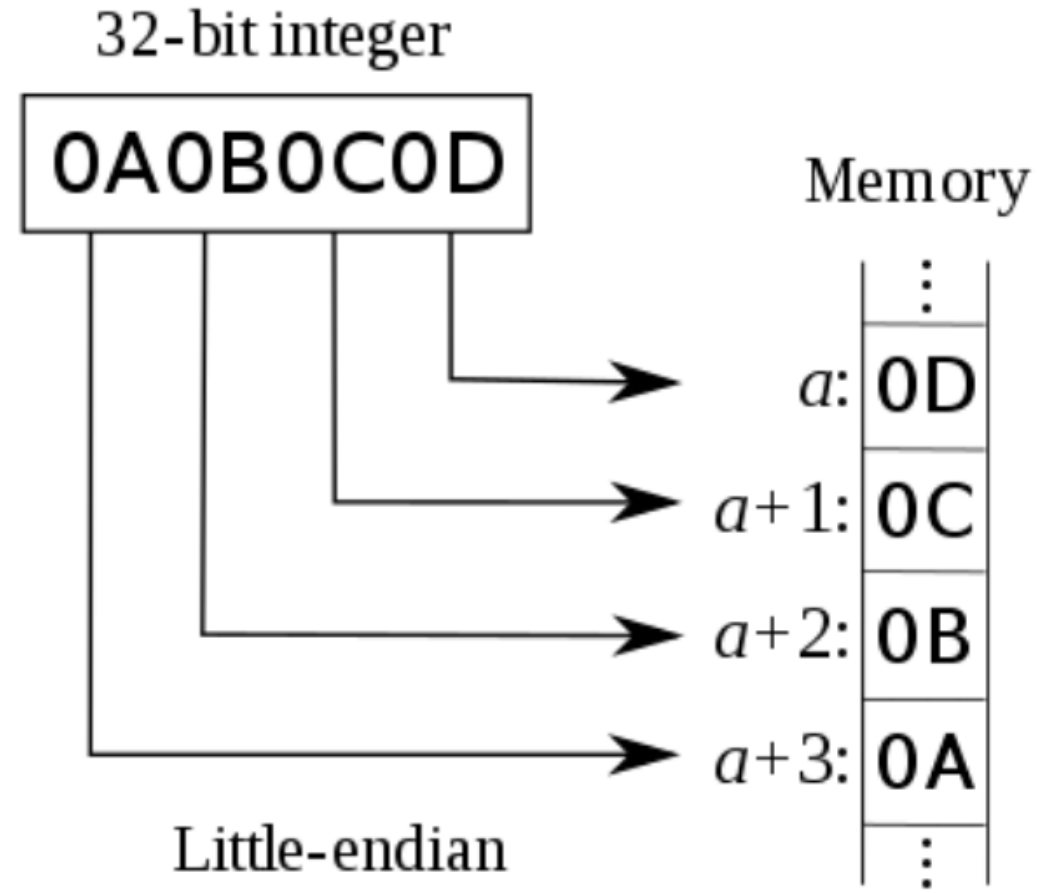
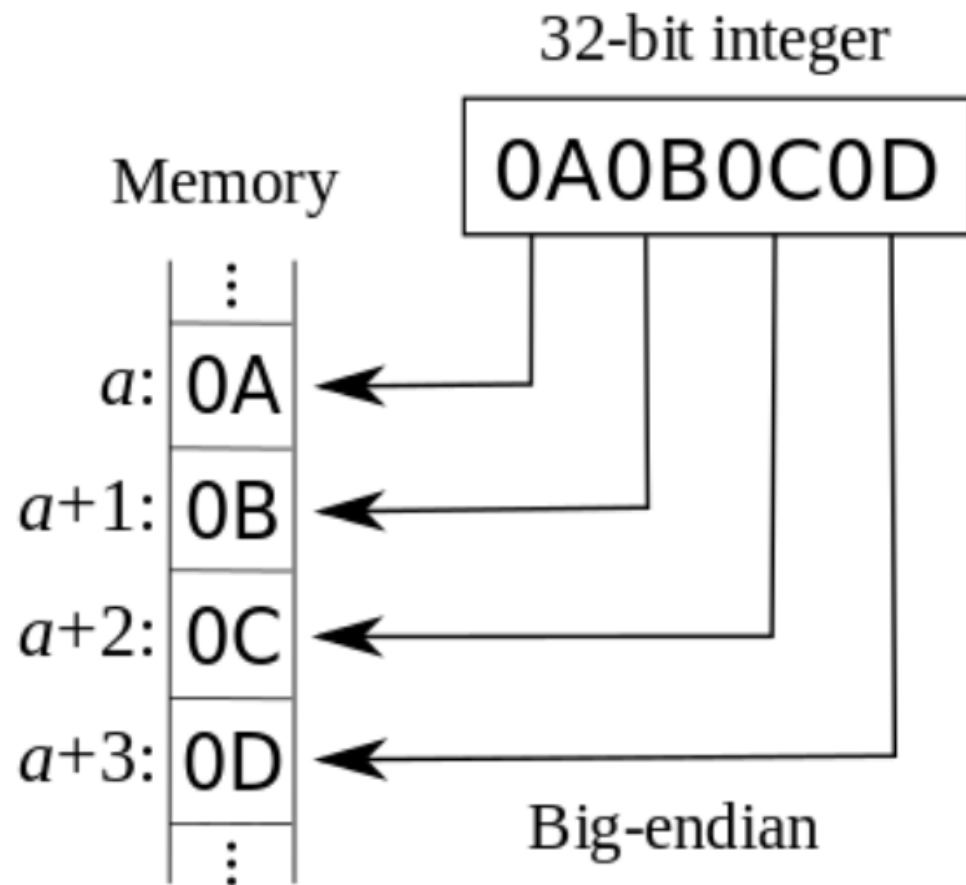
## Bitwise OR `|` [\[edit\]](#)

bit a	bit b	a   b (a OR b)
0	0	0
0	1	1
1	0	1
1	1	1

## Bitwise XOR `^` [\[edit\]](#)

bit a	bit b	a ^ b (a XOR b)
0	0	0
0	1	1
1	0	1
1	1	0

# Endianness





## Some comic relief...

	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

[HTTP://XKCD.COM/1296/](http://xkcd.com/1296/)