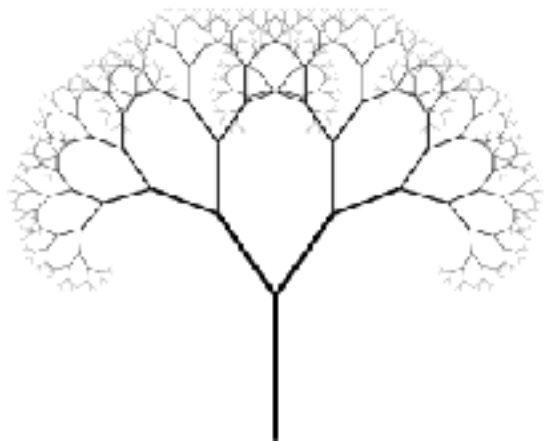


# RECURSION WRAP UP

---

Problem Solving with Computers-I



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

```
    double sumRest;
```

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

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

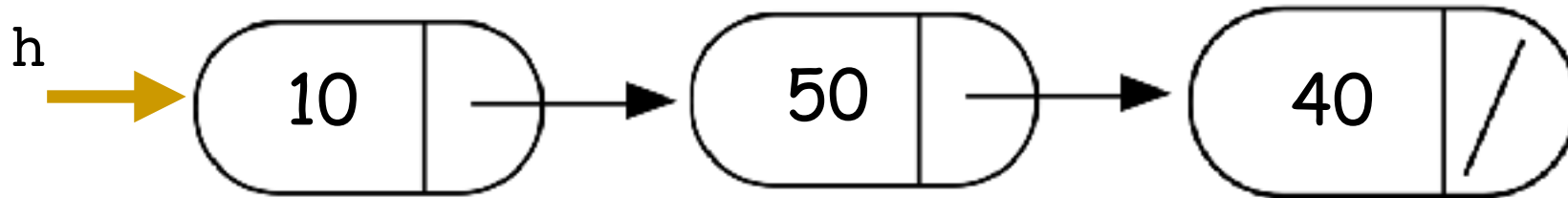
```
}
```



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){
```

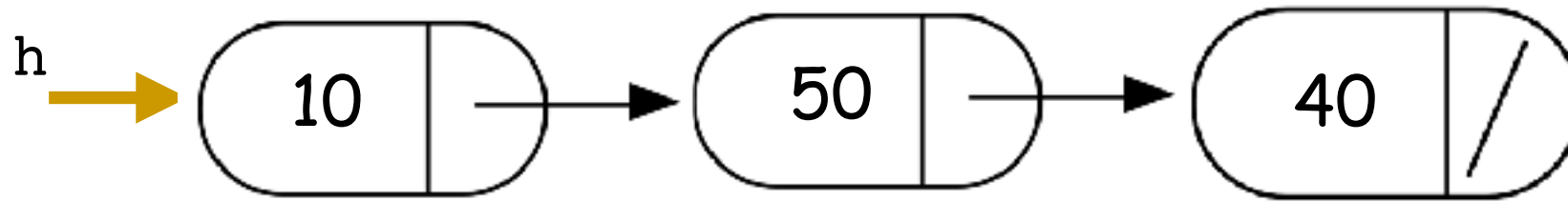
```
    double sumRest;
```



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

```
}
```

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



sumList(1)

First call to sumList!

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

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

```
    double sumRest;
```

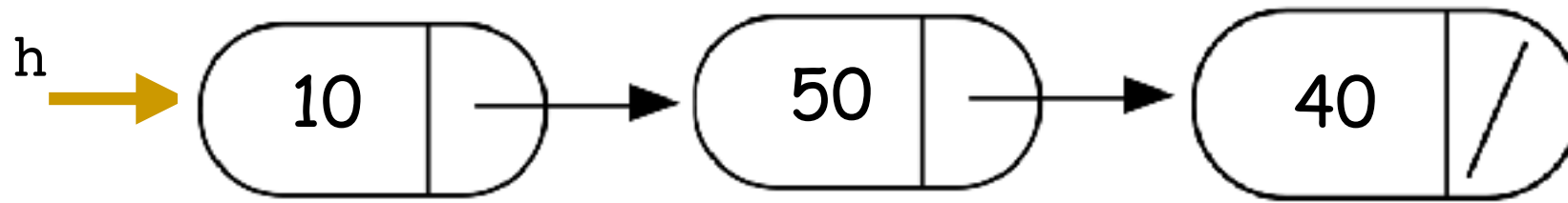
```
    sumRest = sumList(head->next);
```

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

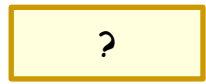
```
}
```



Second call to sumList!



head



sumRest

First call to sumList!

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

sumList(1)

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

```
    double sumRest;
```

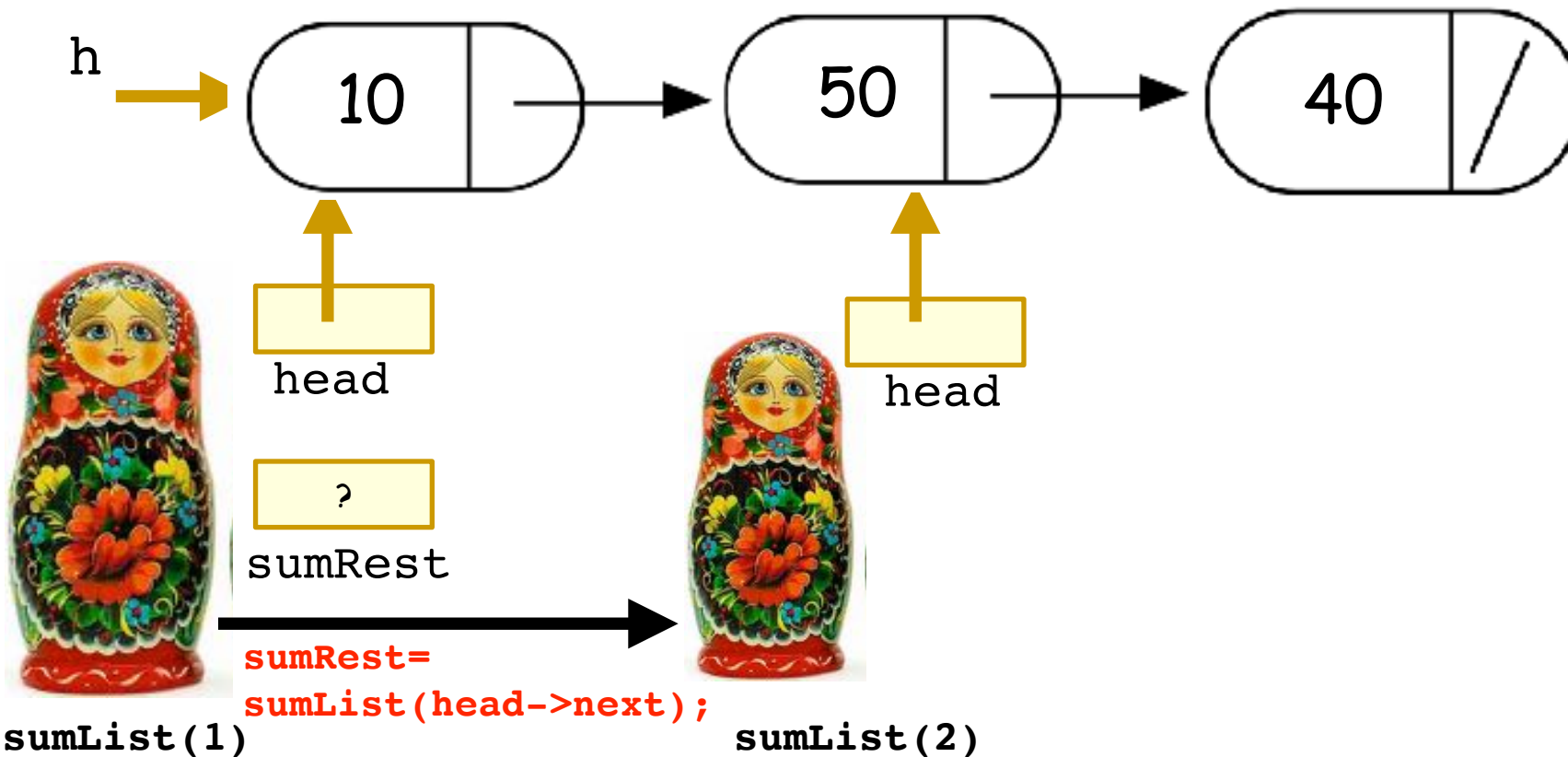


```
    sumRest = sumList(head->next);
```

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

```
}
```

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



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

```
    double sumRest;
```

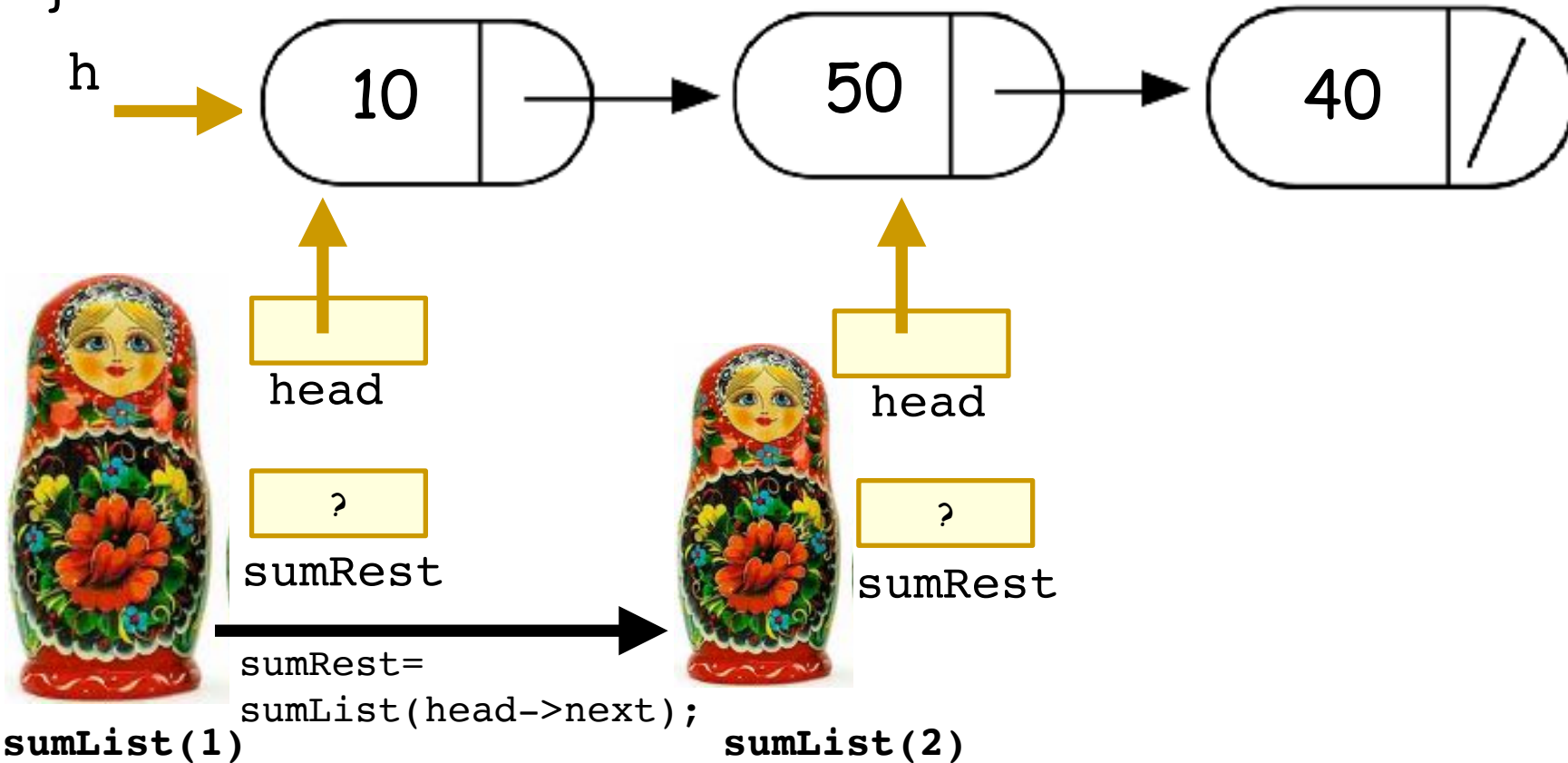
```
    sumRest = sumList(head->next);
```

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

```
}
```



Third call to sumList



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

```
    double sumRest;
```

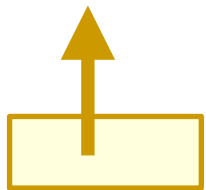
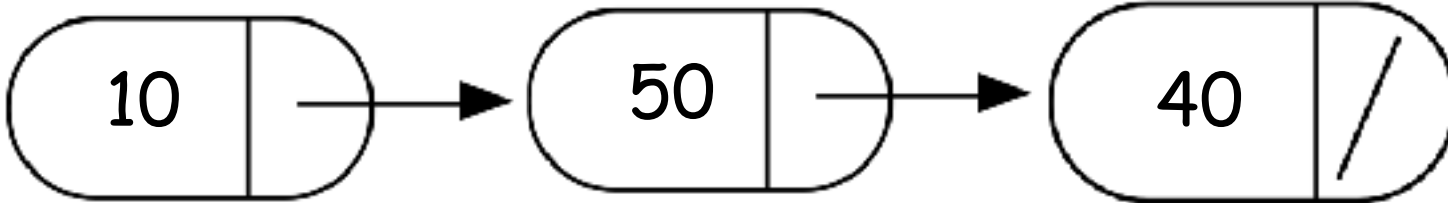


```
    sumRest = sumList(head->next);
```

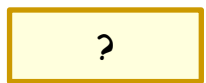
```
    return head->data + sumRest;
```

```
}
```

h



head

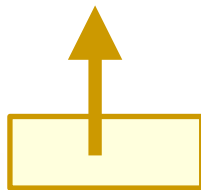


sumRest

sumRest =

sumList(head->next);

sumList(1)



head

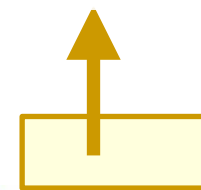


sumRest

sumRest =

sumList(head->next);

sumList(2)



head



sumList(3)

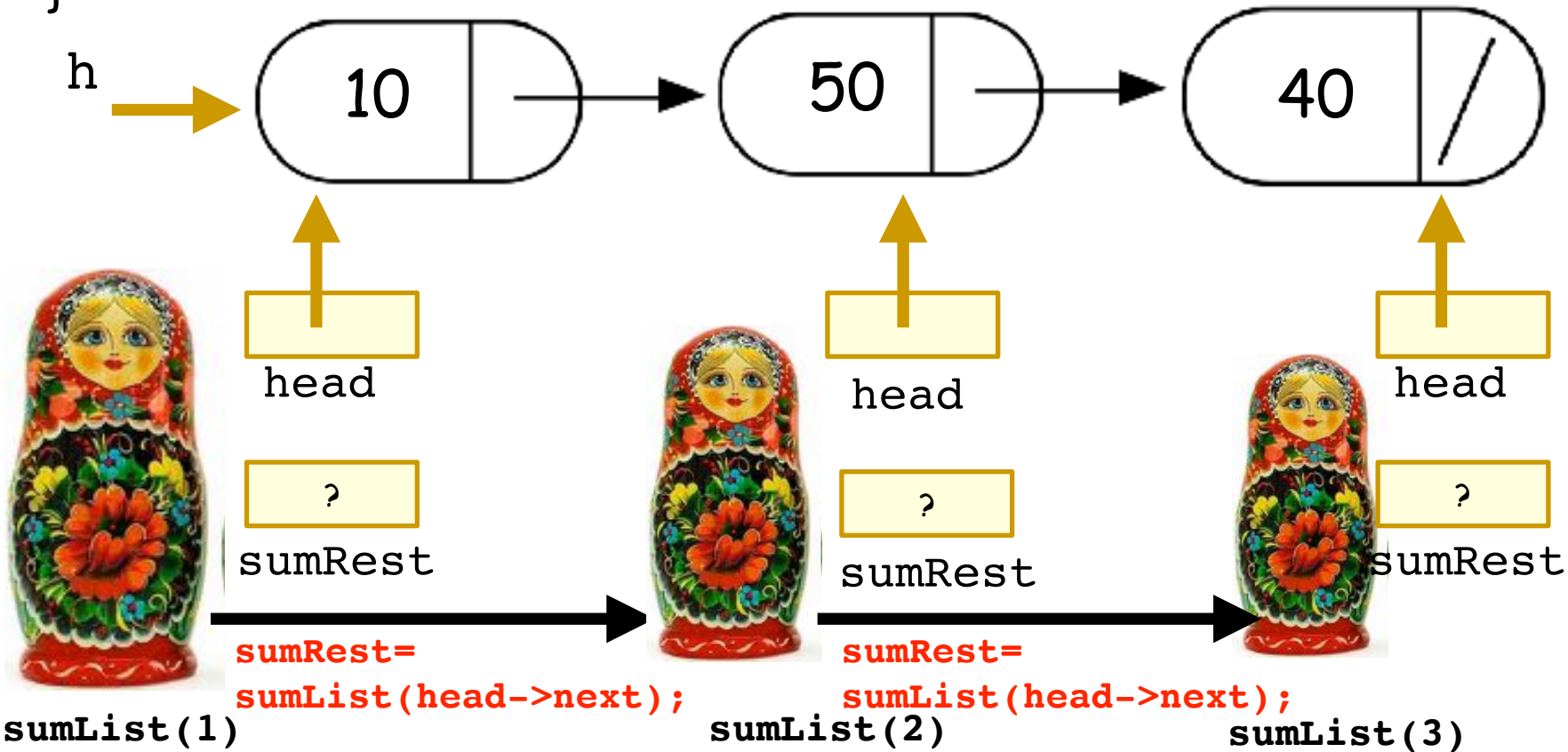
```
double sumList(Node* head){
```

```
    double sumRest;
```

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


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

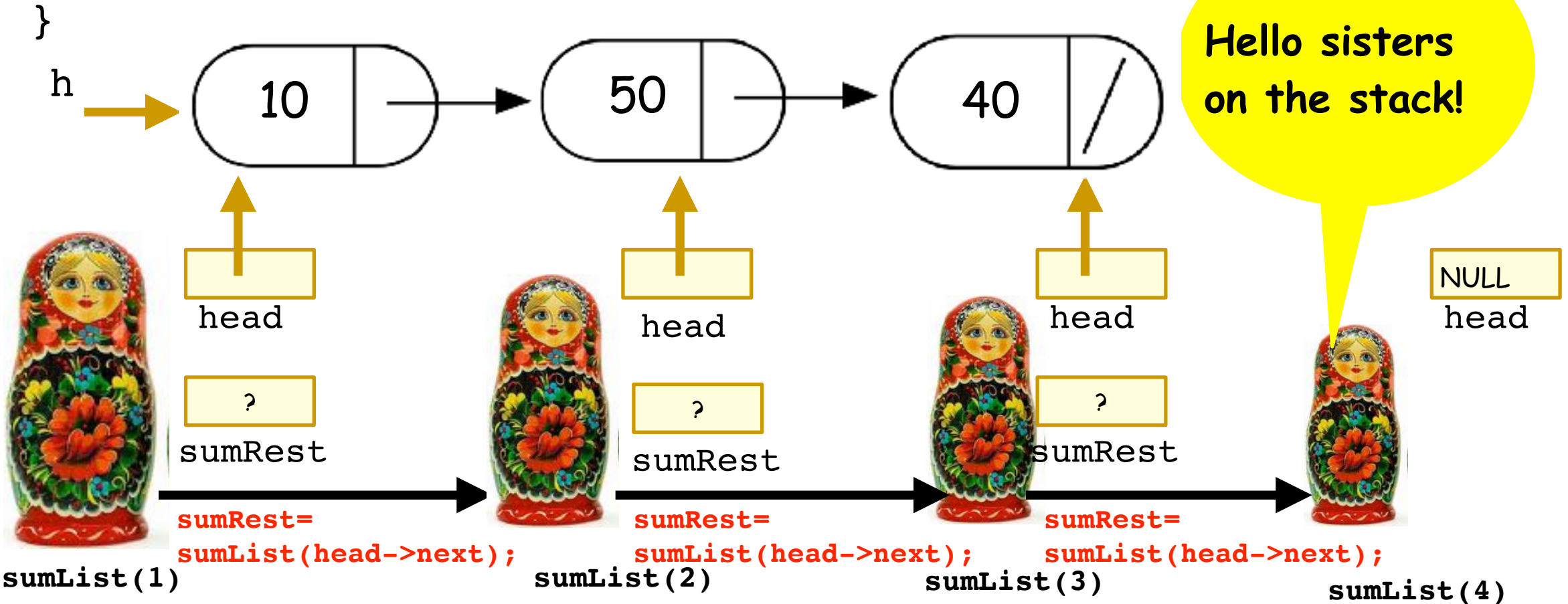
```
}
```





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

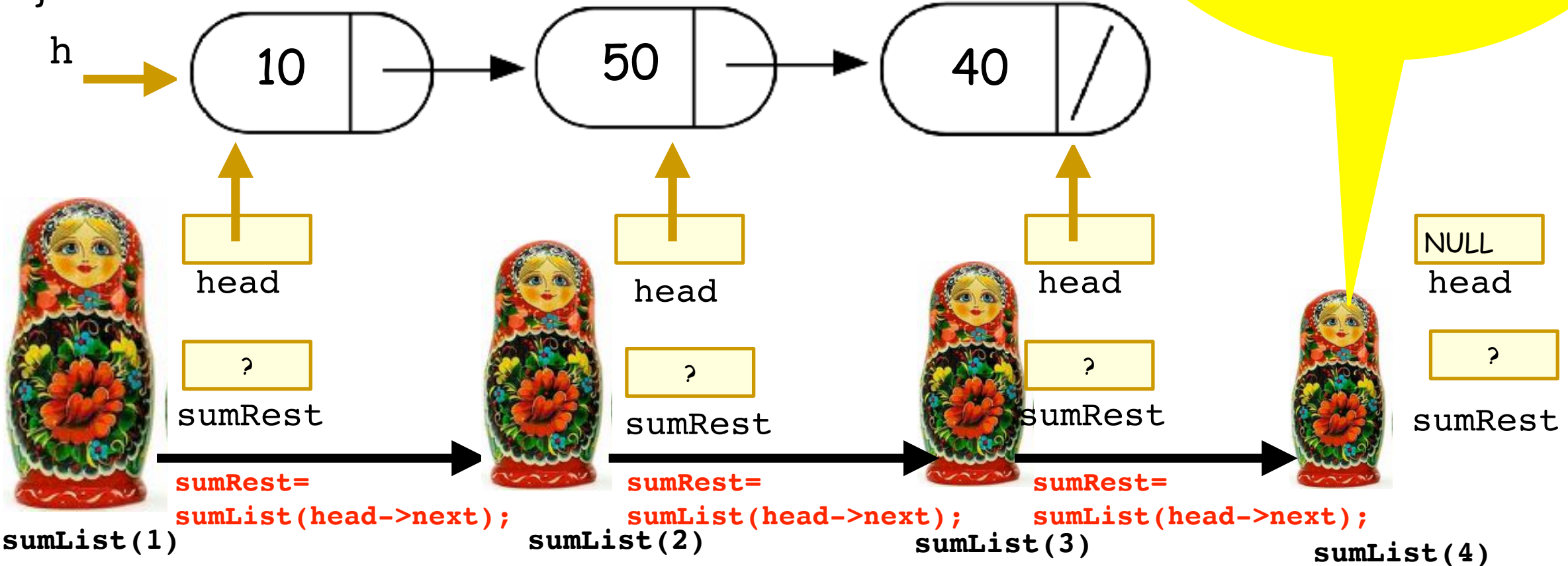
```
    double sumRest;   
    sumRest = sumList(head->next);  
    return head->data + sumRest;  
}
```



```
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 !!!!

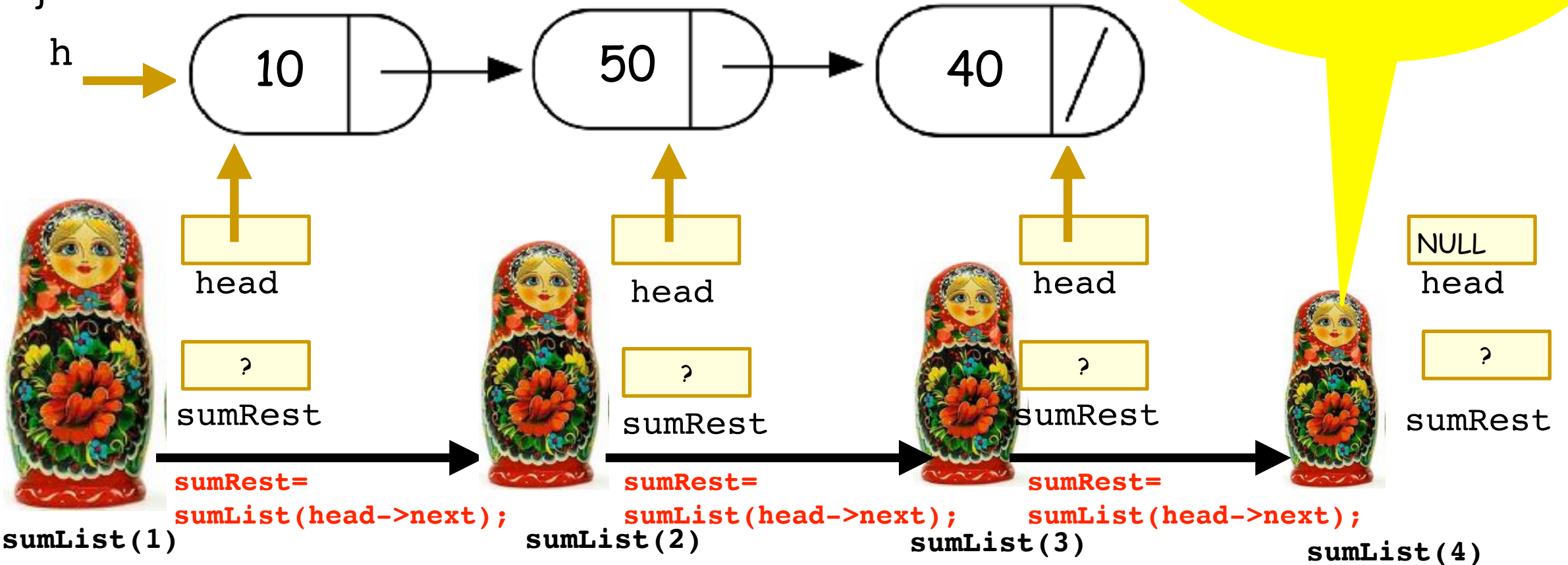


```
double sumList(Node* head){
```

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



No turtle noooooooo !!!!



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

```
    double sumRest;
```

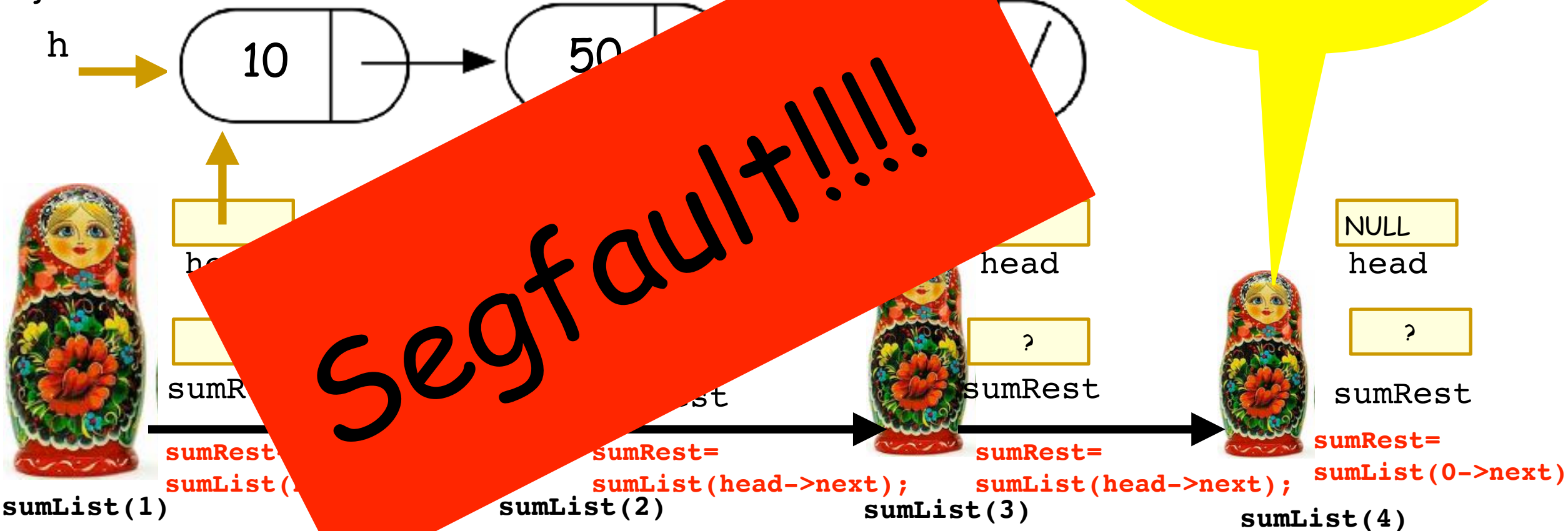
```
    sumRest = sumList(head->next);
```

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

```
}
```

Sorry I just follow instructions!

No turtle noooooo !!!!



sumList(1)

sumList(2)

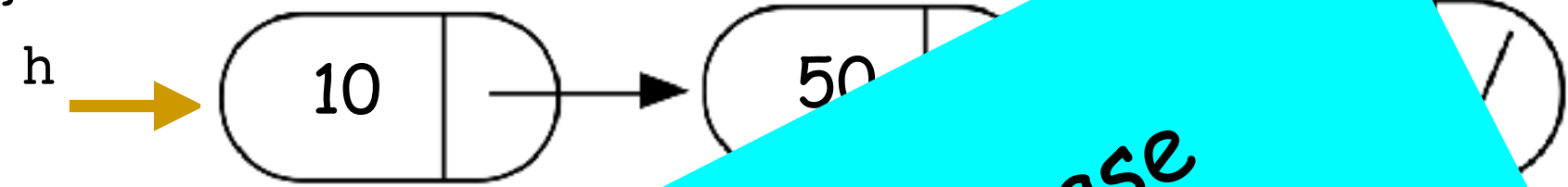
sumList(3)

sumList(4)

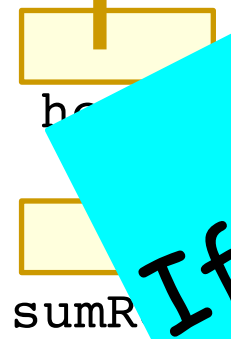
```
double sumList(Node* head){
    double sumRest;
    sumRest = sumList(head->next);
    return head->data + sumRest;
}
```

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);
```



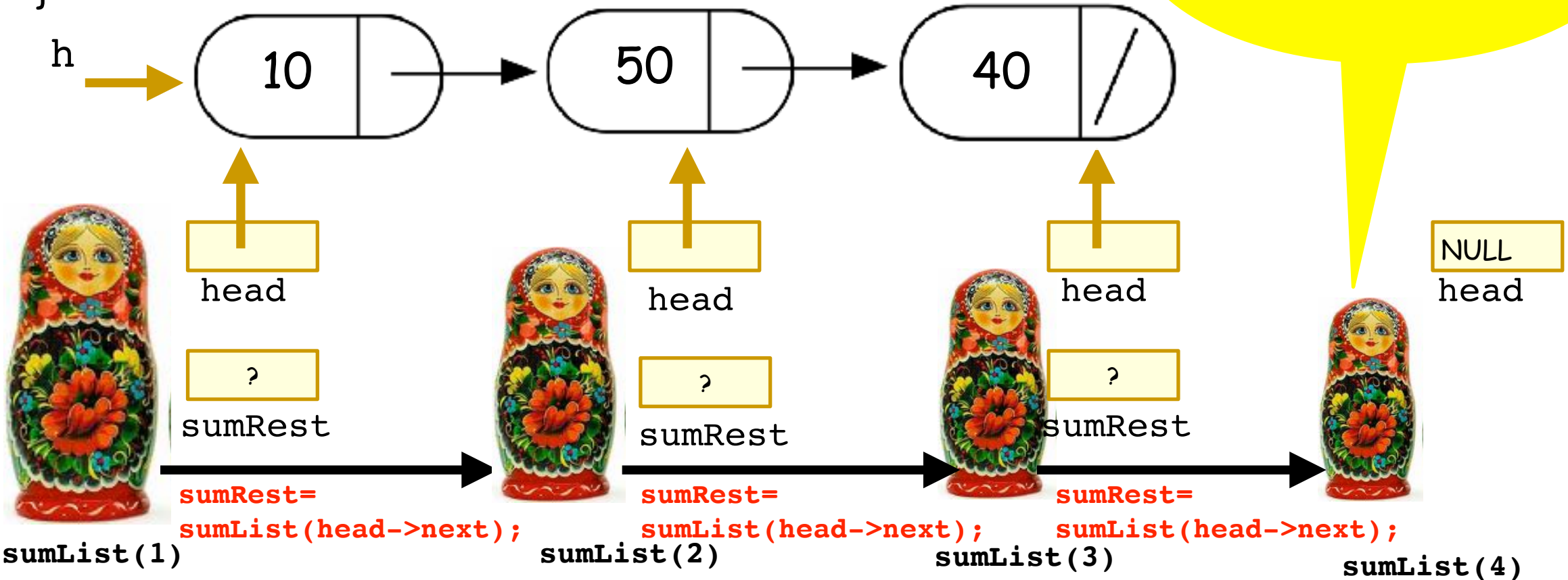
```
sumRest = sumList(0->next);
```

sumList(4)

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

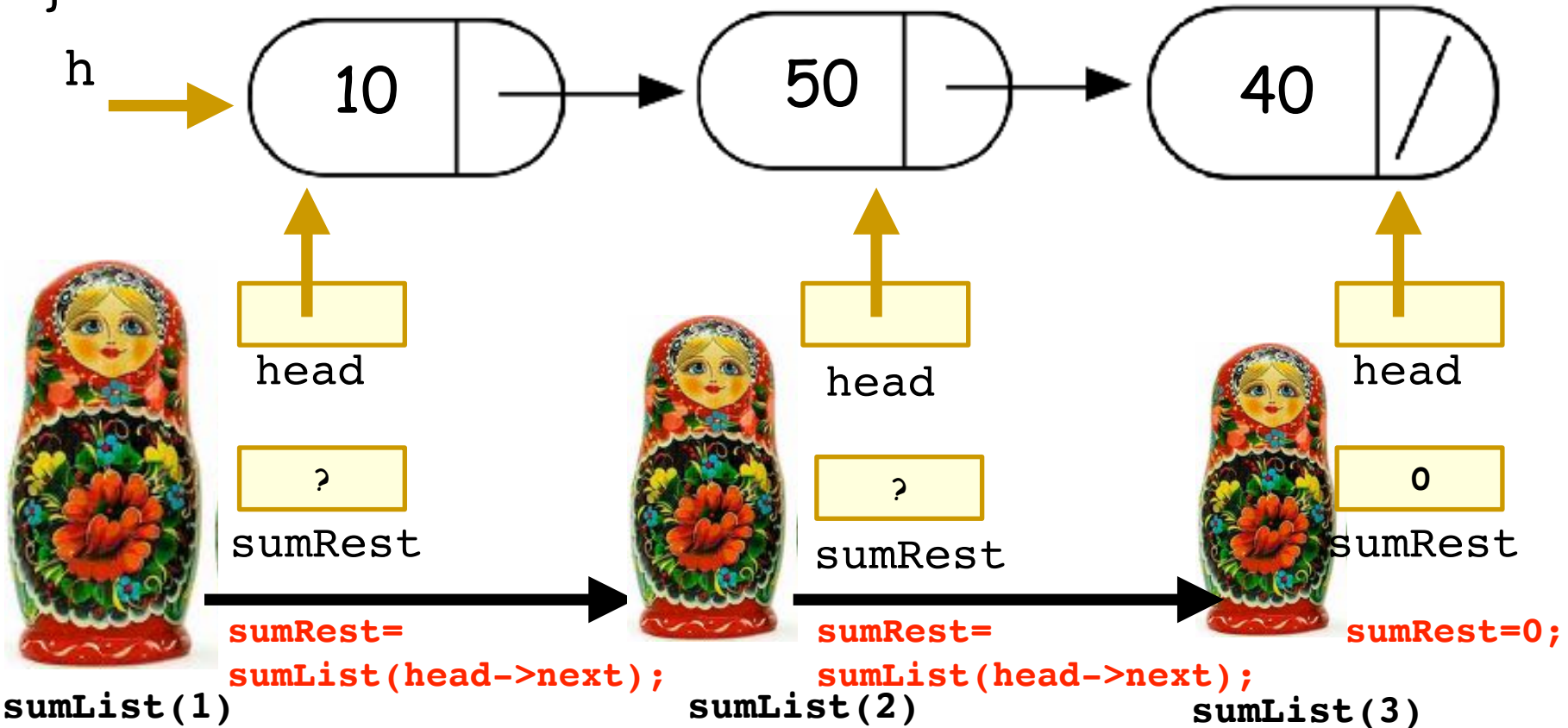
I am really well behaved around base cases :)

return 0;



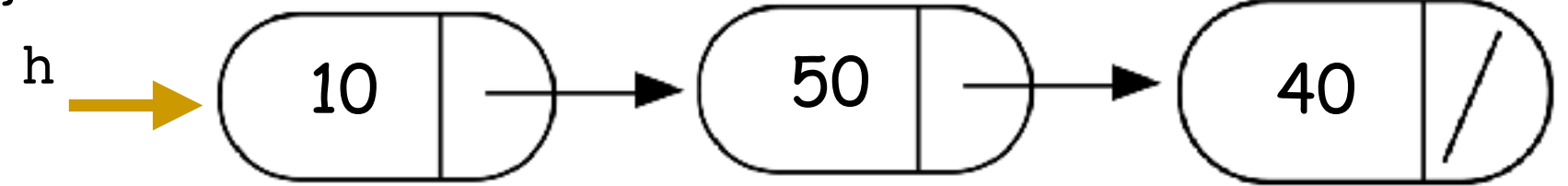
```
double sumList(Node* head){
  if(head==0) return 0;
  double sumRest;
  sumRest = sumList(head->next);
  return head->data + sumRest;
}
```

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

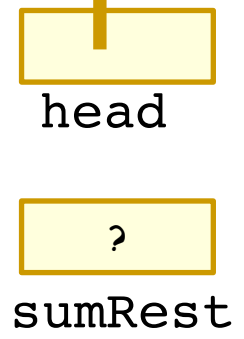


```
double sumList(Node* head){
  if(head==0) return 0;
  double sumRest;
  sumRest = sumList(head->next);
  return head->data + sumRest;
}
```

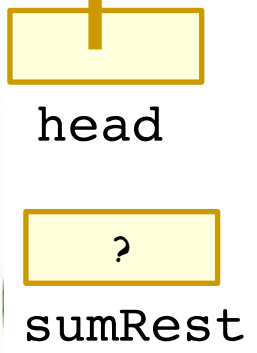
Turtle takes it one line at a time.



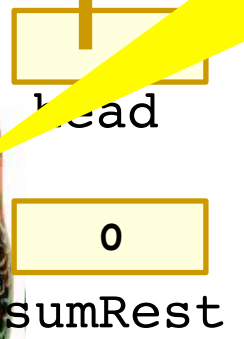
return 40+0;



sumList(1)



sumList(2)



sumList(3)

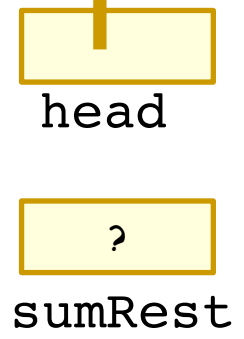
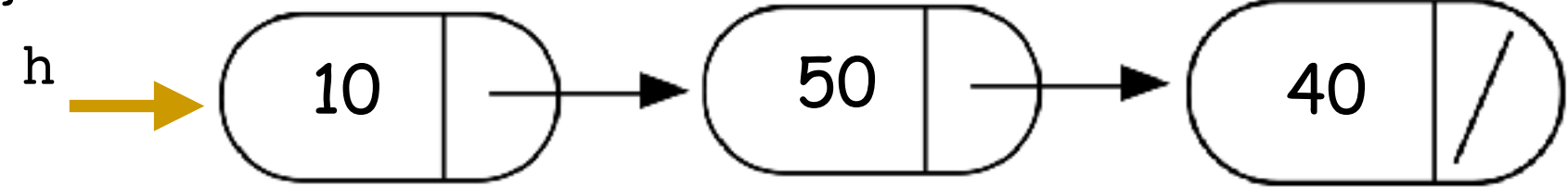
sumRest = sumList(head->next);

sumRest = sumList(head->next);

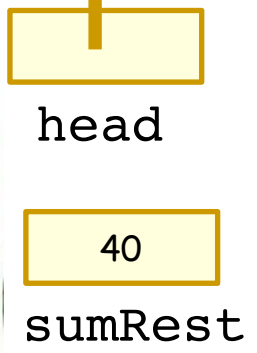


```
double sumList(Node* head){
    if(head==0) return 0;
    double sumRest;
    sumRest = sumList(head->next);
    return head->data + sumRest;
}
```

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



sumList(1)



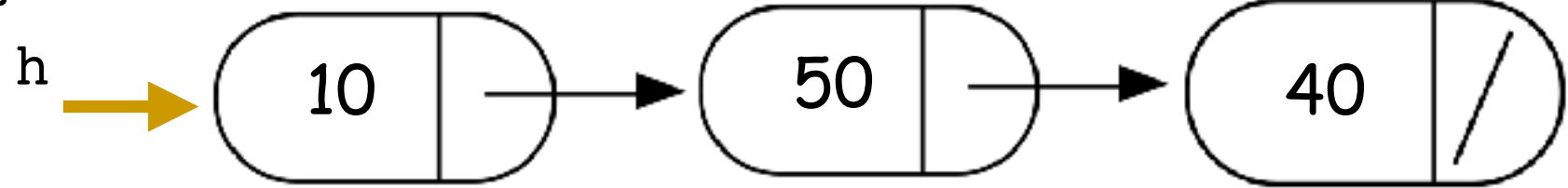
sumList(2)

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

```
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

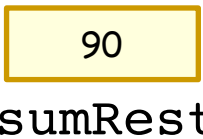
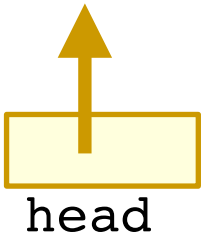
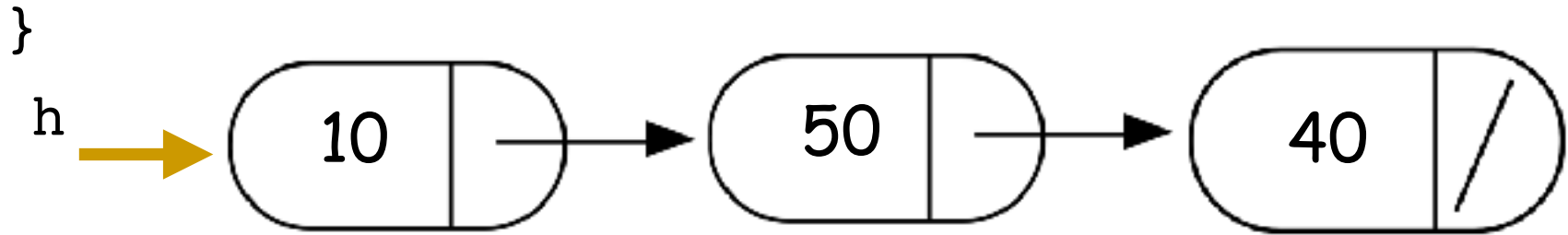
sumList(2)

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!

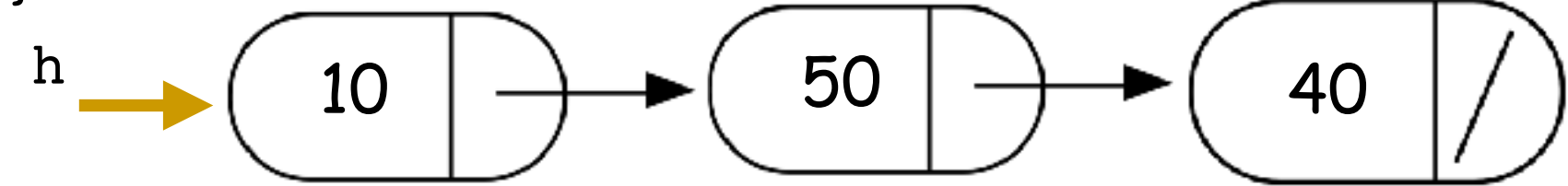


```
return head->data + sumRest;
```

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.



head

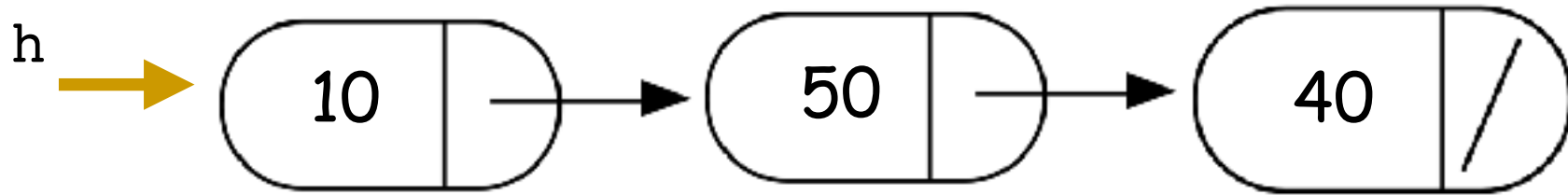


sumRest

**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!**

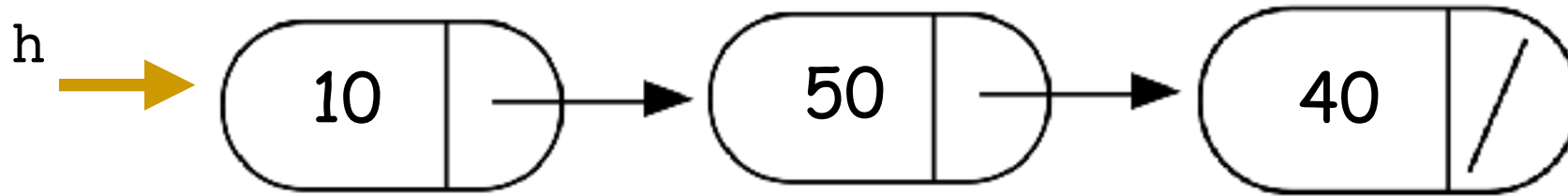
```
double s=sumList(h);
```



```
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);
```

