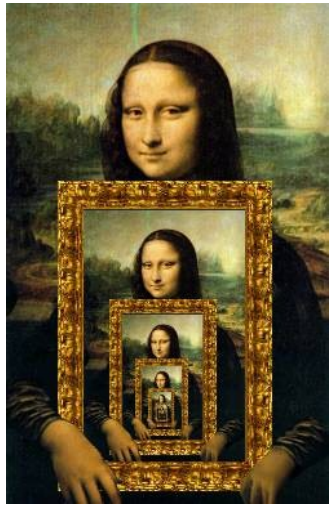
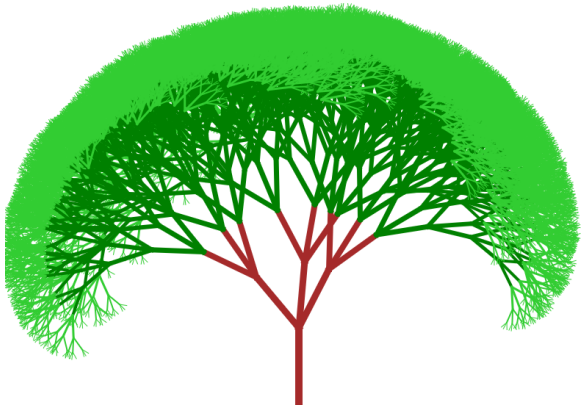
Recursion

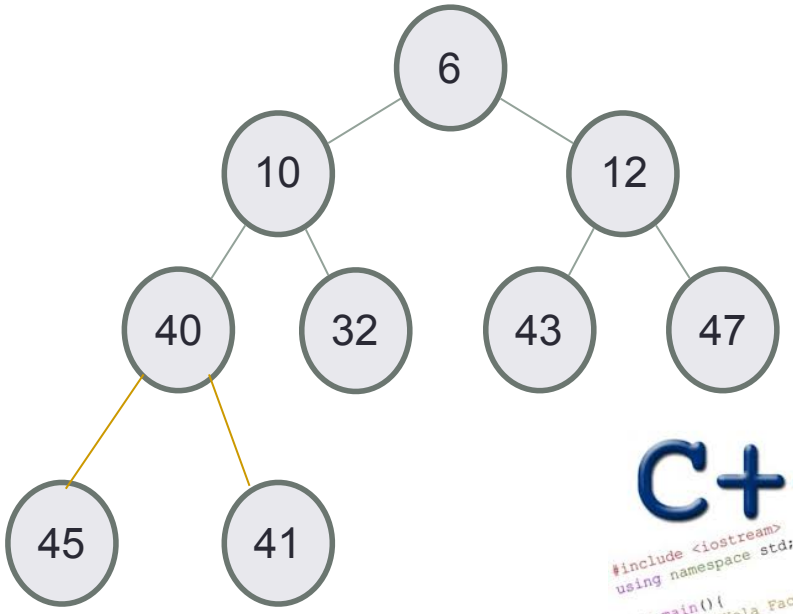
a.k.a., CS's version of mathematical induction



As close as CS gets to magic

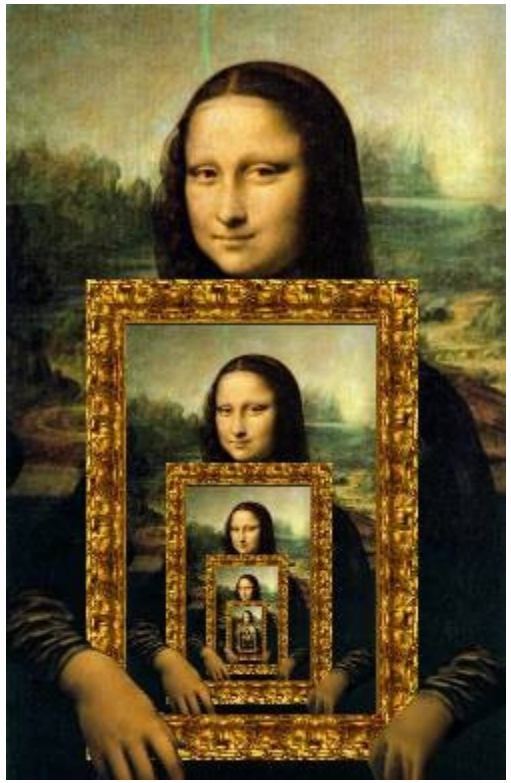
Problem Solving with Computers-I





Let recursion draw you in….

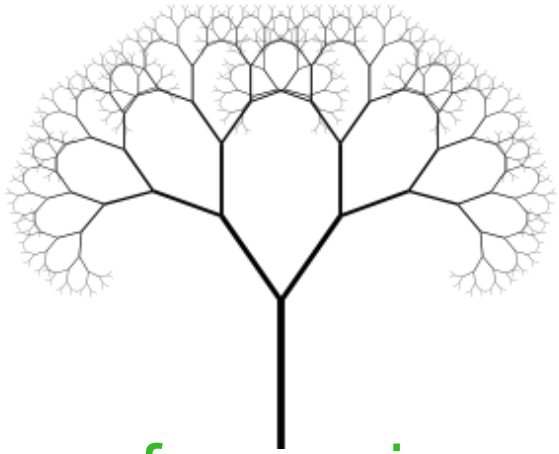
- Recursion occurs when something is described in terms of itself

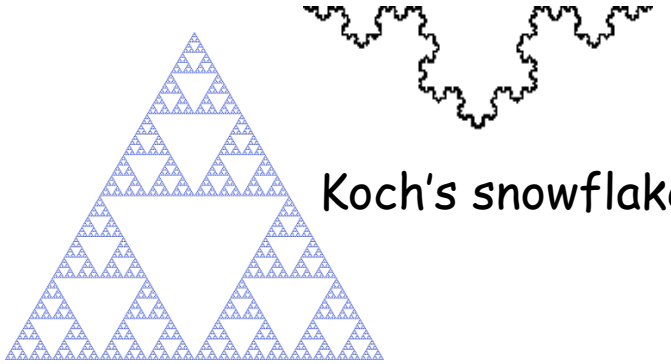


Recursive names

GNU IS NOT UNIX

Fractals





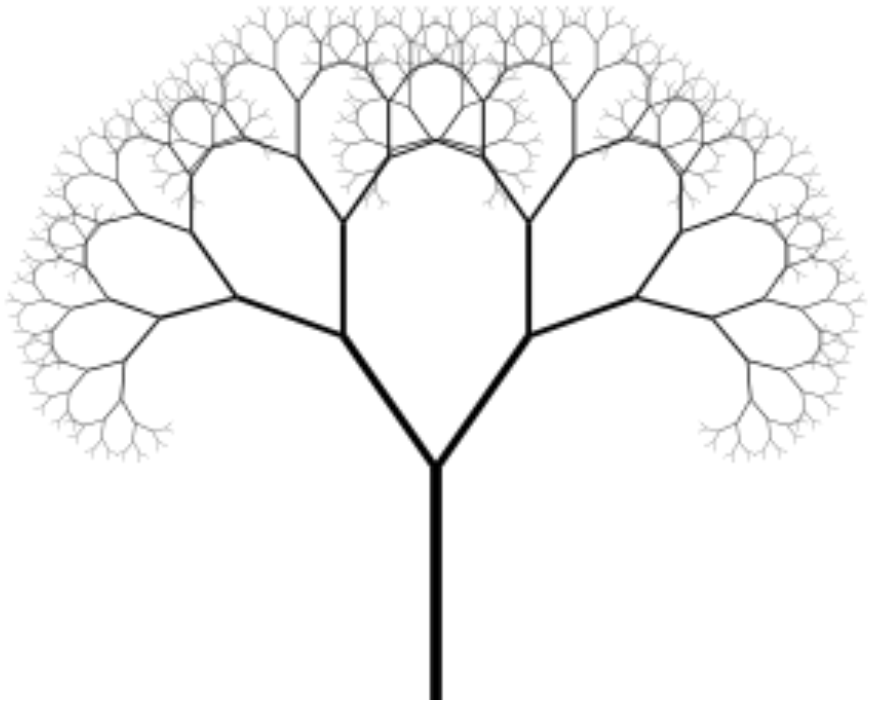
Koch's snowflake

Sierpinski triangle

Visual representations of recursion

Recursion: A way of solving problems in CS

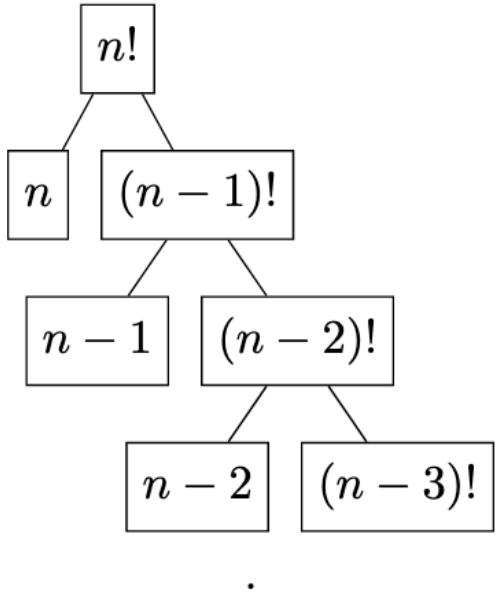
- Solve the simplest case of the problem
- Solve the general case by describing the problem in terms of a smaller version of itself

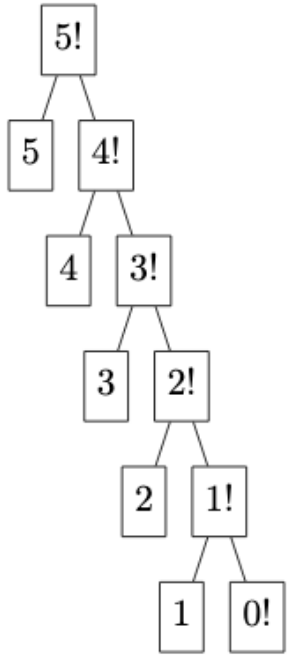


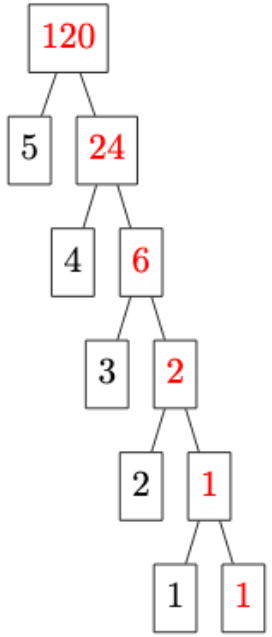
Thinking *recursively*

```
N!  =  N * (N-1)! , if  N > 1
    =  1, if N <= 1
```

Recursion == ***self***-reference!







Computing a recursive function

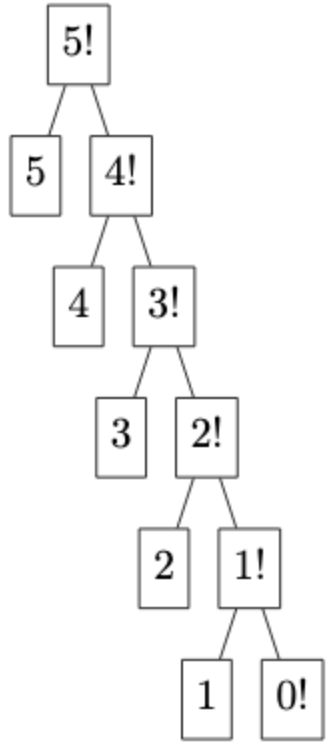
Designing Recursive Functions

```
int fac(int N){
    if(N <= 1){
        return 1;
    }


}
```

Base case:

Solution to inputs where the answer is simple to solve

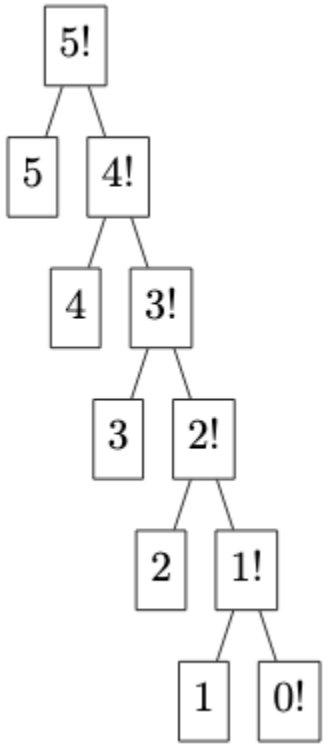


Designing Recursive Functions

```
int fac(int N){
    if(N <= 1){
        return 1;
    }else{
        double rest= fac(N-1);
        return N* rest;
    }
}
```

Base case

Recursive case



Human: Base case and 1 step

Computer: Everything else

Warning: *this is legal!*

```
int fac(int N){
    return N* fac(N-1);
}
```

legal **!=** *recommended*

```
int fac(int N){
    return N* fac(N-1);
}
```

No ***base case*** -- the calls to **`fac`** will never stop!

Make sure you have a **base case**, *then* worry about the recursion...

Print the numbers 1 to N recursively

```
void printInorder(int N){

                                    //Base case

}
```

Select the appropriate base case:

A.
```
cout<<N<<endl;
```

B.
```
if (N == 1){
    cout<<N<<endl;
    return;
}
```

C.
```
if (N > 1){
    cout<<N<<endl;
    return;
}
```

D. All of the above are correct

Print the numbers 1 to N recursively

```
void printInorder(int N){
        if (N == 1){
             cout<<N<<endl; //Base case
             return;
        }

        ____________________________(A)
        printInorder(N-1);
        ____________________________(B)
}
```

Choose the correct location of this statement:

```
cout<<N<<endl;
```

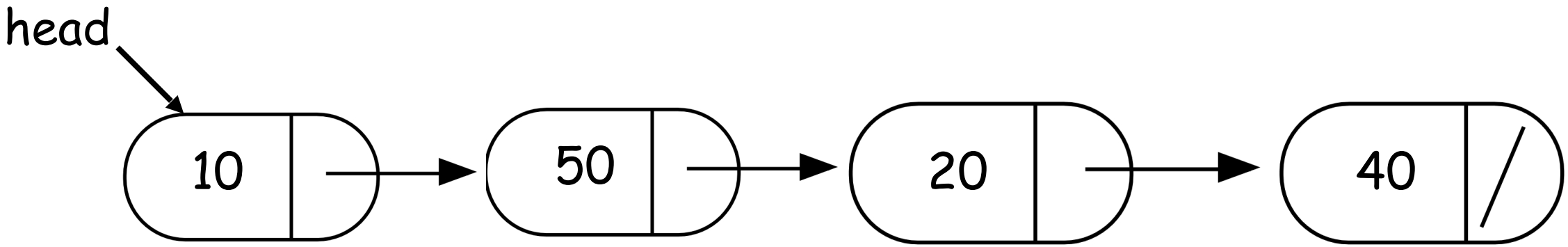
Print all the elements of an array in order

```
void printArray(int arr[], int len){
        if(len <=0) return;
        cout<<arr[0]<<endl;
        printArray(_______, _______);
}
```

Select the arguments to the call to printArray:

A. (arr, len)
B. (arr - 1, len - 1)
C. (arr + 1, len - 1)
D. (arr + 1, len)
E. (arr - 1, len)

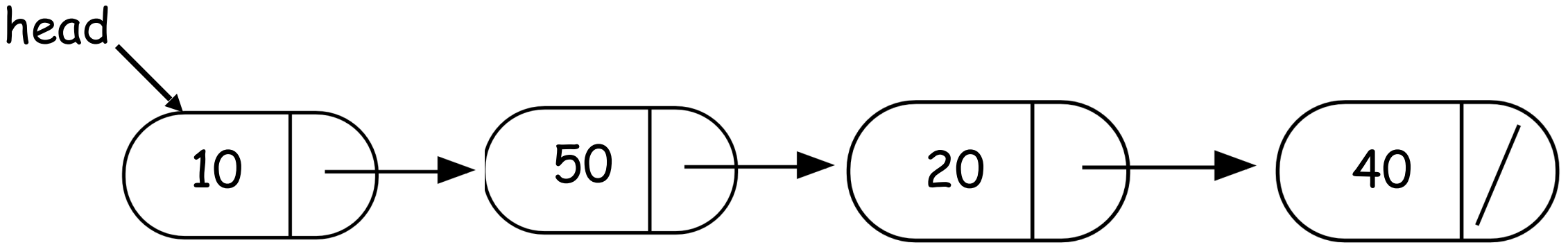
Recursive description of a linked list



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

Recursion to solve problems involving linked-lists

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



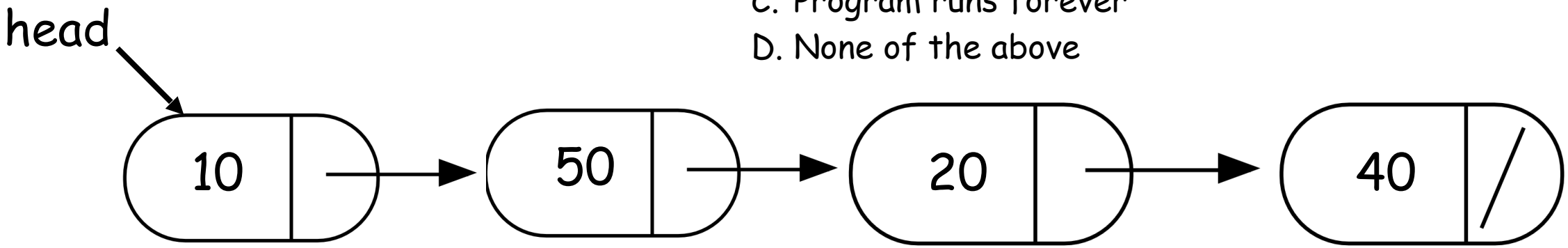
Small group activity (10 minutes)

1. **Write a recursive function to return the sum of the values stored in a linked list**
2. **Share your code with the person sitting next to you and discuss**

What's in a base case?

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

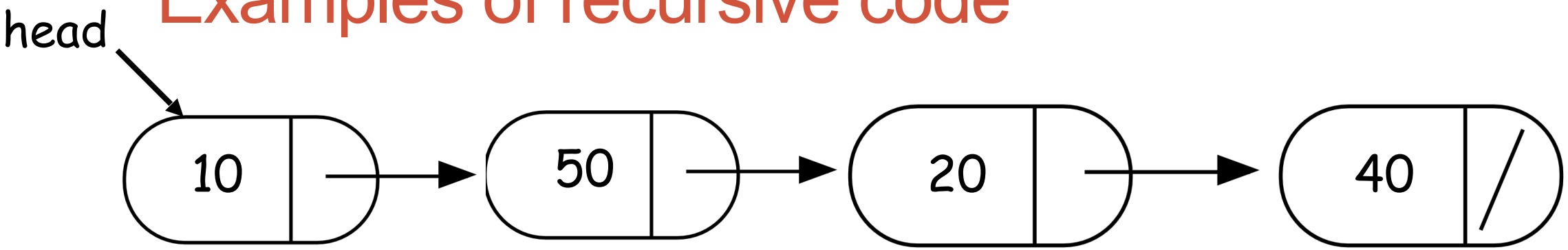
A. Returns the correct sum (120)
B. Program crashes with a segmentation fault
C. Program runs forever
D. None of the above




```
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









}
```

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

}
```

Next time

- More practice with recursion
- Final practice