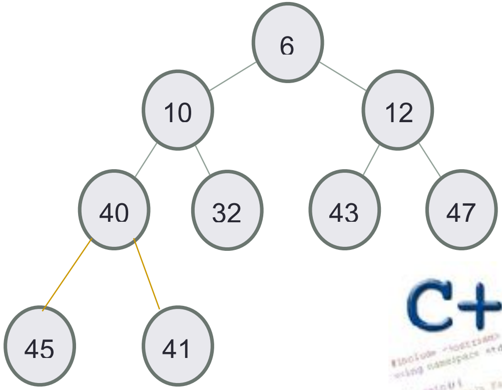
RECURSION

Problem Solving with Computers-I



Let recursion draw you in….

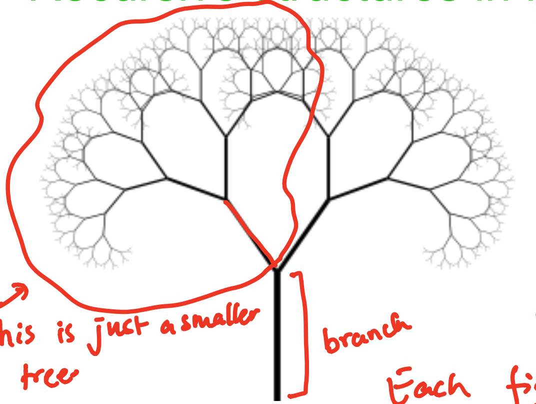
- Identify the “recursive structure” in these pictures by describing them

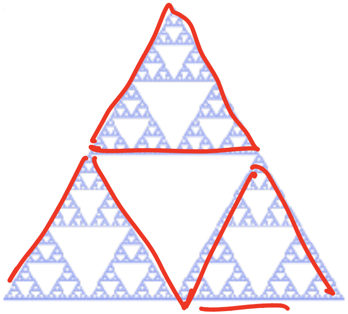
Understanding recursive structures

- 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

Each figure can be described interms of itself

Why is recursion important in Computer Science

Tool for solving problems (recursive algorithms)

Recursive algorithms provide describe the solution to the problem in terms of itself.

The key idea is to use solutions to smaller versions of the problem in the description of the 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!

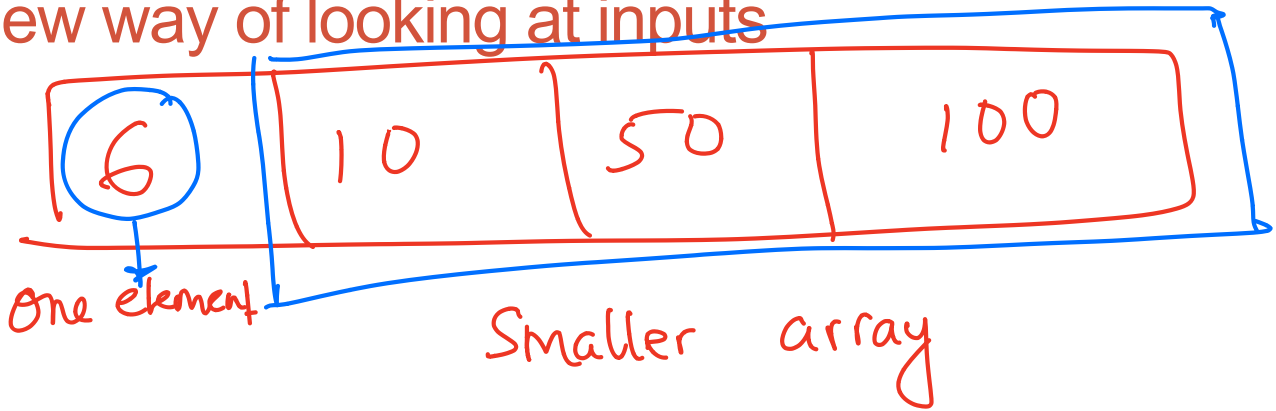
Else:

smaller input

Wash the *remaining* dishes in the sink

→ Recursive step

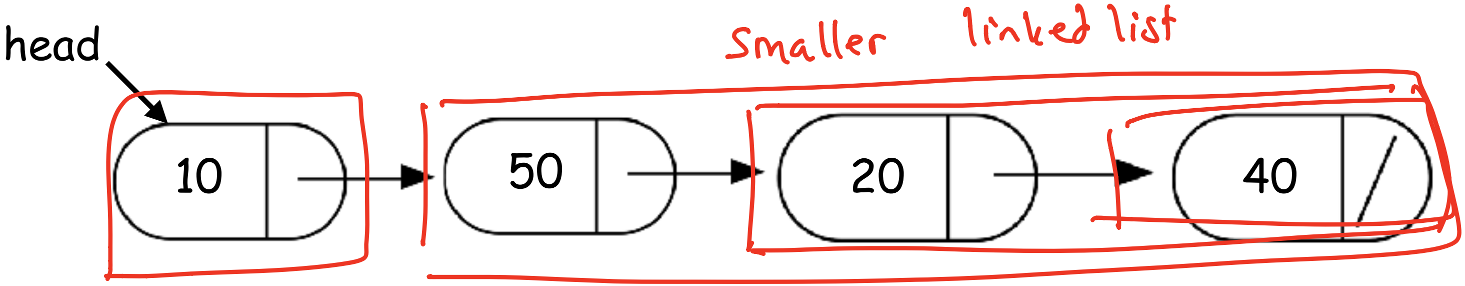
A new way of looking at inputs



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

```
void printArray( int arr[], int len) {
    if (len == 0)
        return;
    // To write the recursive step: Assume the function WORKS
    // for any input smaller than len. This means we can just print
    // one element & call printArray to print the REST of the array.
    cout << arr[0] << " ";
    printArray(arr+1, len-1);
}
```

→ Base case: Solve the problem for the smallest valid input

arr+1, len-1 → passing in the rest of the array

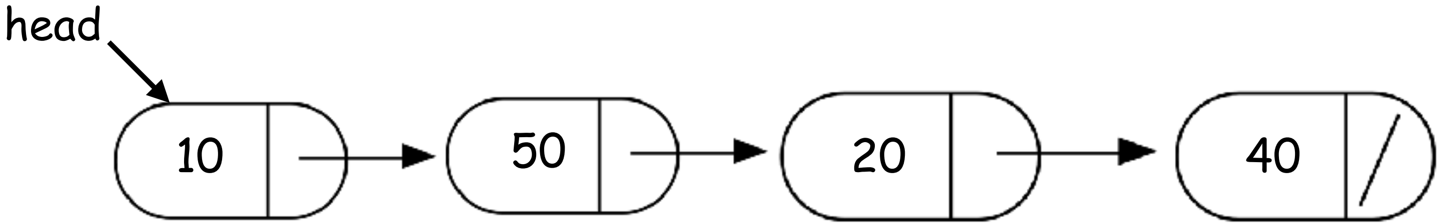
Arrays:

- 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**

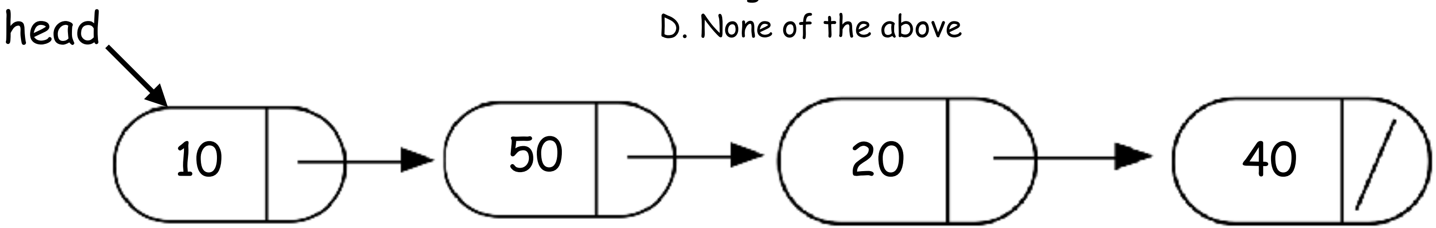
Next lecture



What's in a base case?

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

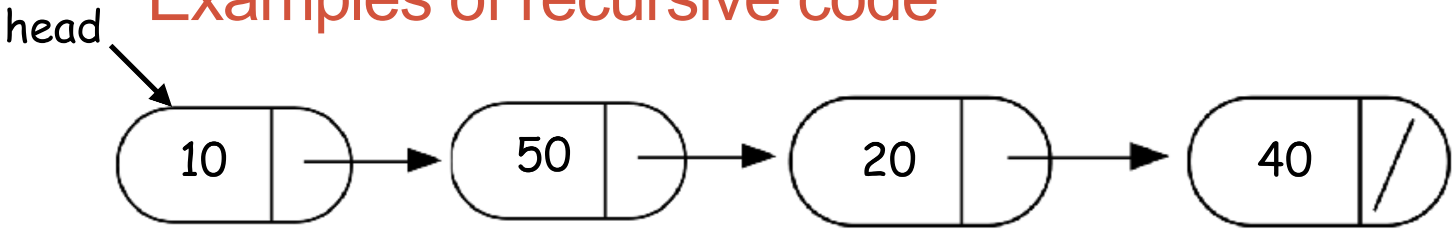
A. Returns the correct sum (120)
B. Program crashes with a segmentation fault
C. Program runs forever
D. None of the above




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

}
```

Next time

- Advanced problems with strings and recursion
- Final practice