

LEC 12

CSE 123

Linked Lists with Recursion

Questions during Class?
Raise hand or send here



sli.do #cse123

BEFORE WE START

Talk to your neighbors:

Where is the best boba in Seattle? 🥤

Respond on [sli.do](#)!


Instructor: Miya Natsuhara

TAs:

Arohan	Shiven	Yuntong	Anya
Sreshta	Vrinda	Amiya	Anisha
Rushil	Gavin	Sahana	Trien
Sean B	Shreya	Anirudh	Neal
Chloe	Jonah	Rohan	Evan
Jenny	Renee	Crystal	Rena
Nate	Chris	Eeshani	
Saachi	Ishita	Prakshi	
Hawa	Kuhu	Aidan	
Maggie	Kavya	Cora	
Sean E	Misha	Nhan	

Music: 🌸 [CSE 123 26sp Lecture Tunes](#) 🌸

Lecture Outline: Announcements

- **Announcements** 
- Traversing Linked Lists Recursively
- Modifying Linked Lists Recursively

Announcements

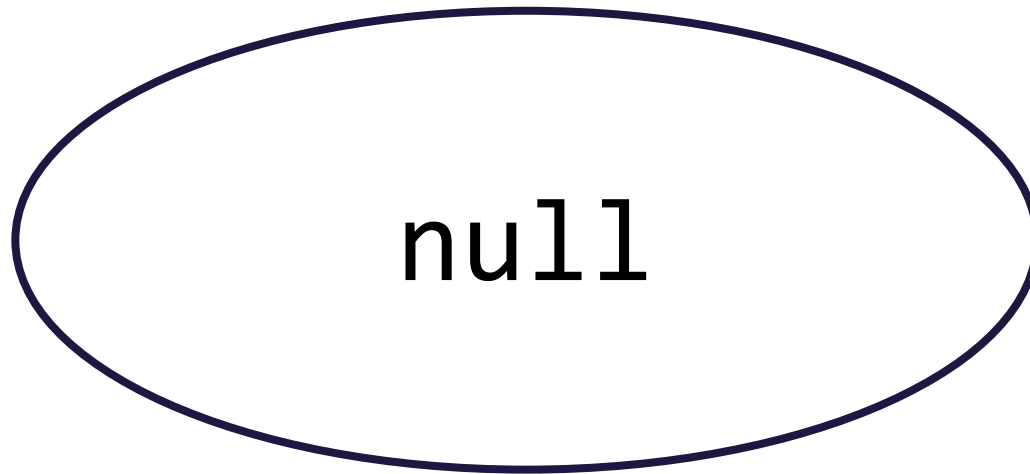
- Creative Project 2 due tonight (May 13) at 11:59pm
- Programming Assignment 2 released tomorrow (May 14)
 - Focused on exhaustive search + recursive backtracking!
- Resubmission Cycle 4 is due Fri (May 15) at 11:59pm
 - C1, P1 eligible

Lecture Outline: Traversing Linked Lists Recursively

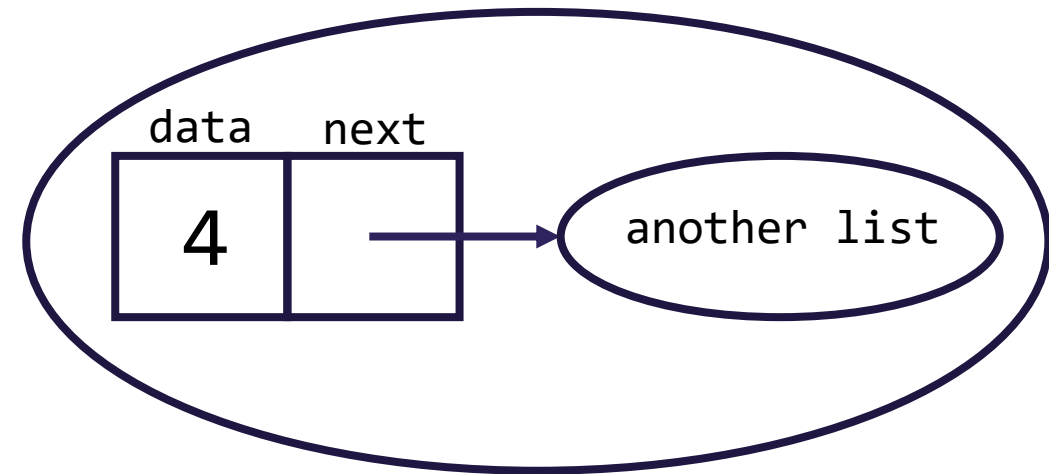
- Announcements
- **Traversing Linked Lists Recursively** ◀
- Modifying Linked Lists Recursively

Linked Lists

- A linked list is either:



Empty list



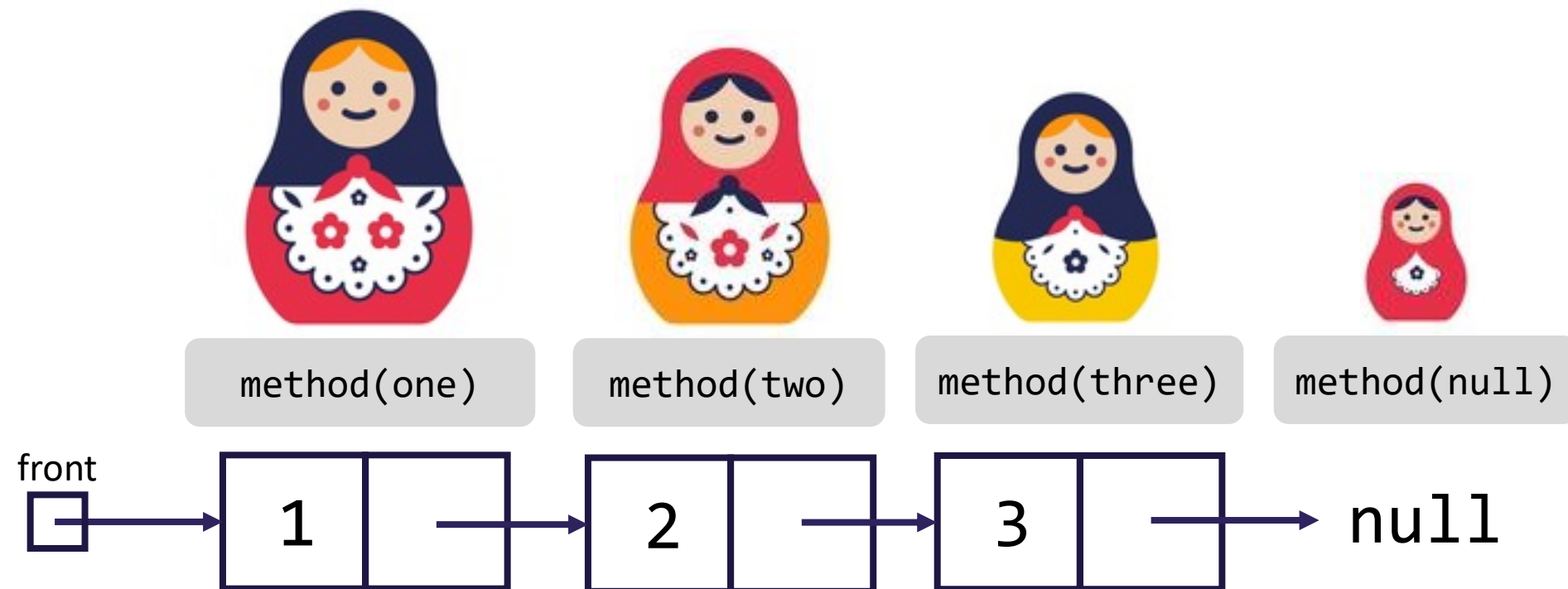
Node w/ another linked list

This is a recursive definition!

A list is either empty or a node with another list!

Recursive Traversals w/ LinkedLists

- Guaranteed base case: empty list
 - Simplest possible input, should immediately know the return
- Guaranteed public / private pair
 - Need to know which sublist you're currently processing (i.e. curr)



Lecture Outline: Modifying Linked Lists Recursively

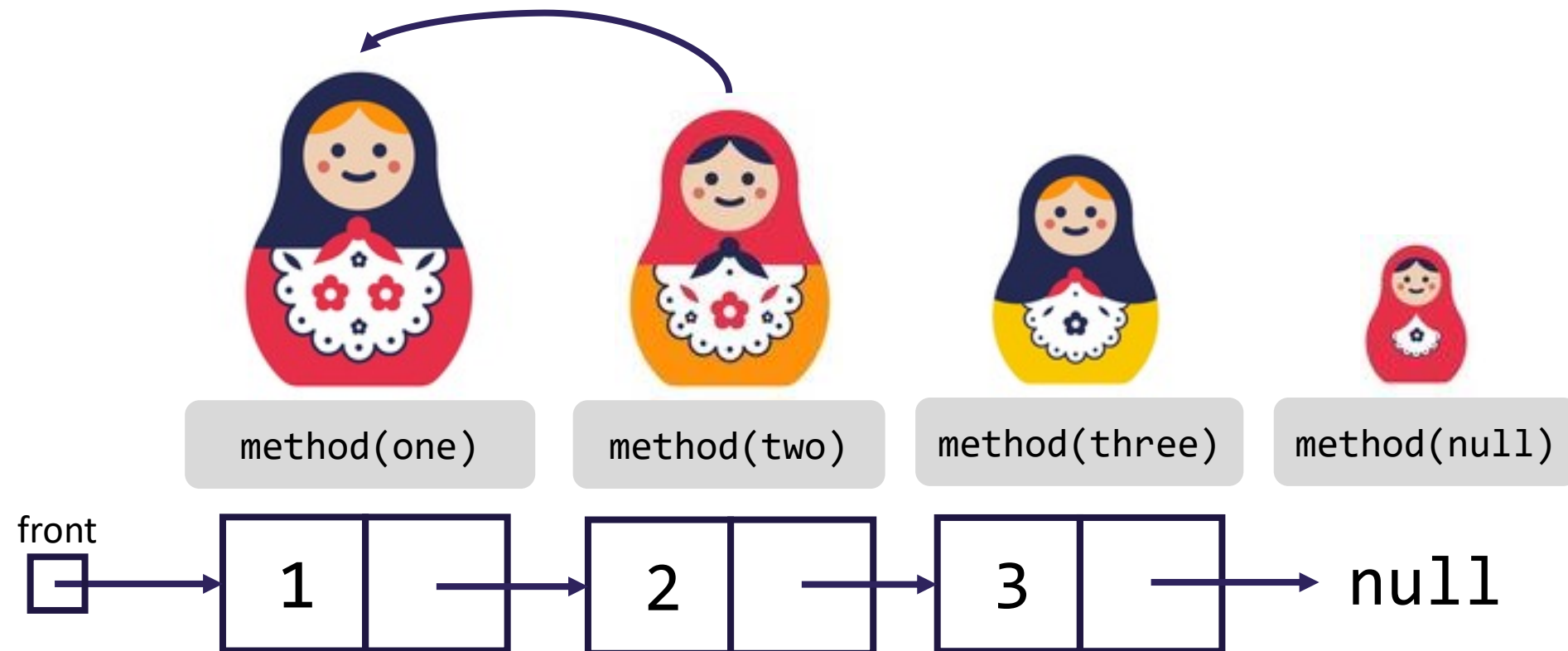
- Announcements
- Traversing Linked Lists Recursively
- **Modifying Linked Lists Recursively** ◀

Modifying LinkedLists [Review]

- Remember: using a `curr` variable to iterate over nodes
- Does changing `curr` actually update our chain?
 - What will? Changing `curr.next`, changing `front`
 - Need to **stop one early** to make changes
- Often a number of cases to watch out for:
 - M(iddle) – Modifying node in the middle of the list (general)
 - F(ront) – Modifying the first node
 - E(mpty) – What if the list is empty?
 - E(nd) – Rare, do we need to do something with the end of the list?

Modifying LinkedLists Recursively

- Much easier than iterative solutions!
- No longer need to stop one early
 - Can go right to the point you'd like to make the change



Modifying LinkedLists Recursively: $x = \text{change}(x)$

- Much easier than iterative solutions!
- No longer need to stop one early
 - Can go right to the point you'd like to make the change
- How? Return the updated change and catch it!
 - Private pair returns `ListNode` type
 - `curr.next = change(curr.next) / front = change(front)`
 - Resulting solutions much cleaner than iterative cases
- We call this pattern $x = \text{change}(x)$

removeAll Walkthrough



```
public ListNode removeAll(int value ) {
    front = removeAll(value, front);
}

private ListNode removeAll(int value, ListNode node) {
    if (node == null) {
        return node;
    } else if (node.data == value) {
        return removeAll(value, node.next);
    } else {
        // x = change(x)
        node.next = removeAll(value, node.next);
        return node;
    }
}
```