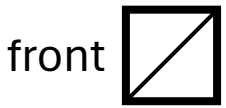
Linked Lists and Recursion

Linked Lists are defined recursively!

- A linked is is either:
  - Empty (null reference)



  - A ListNode with a reference to a linked list

Reading a Linked List (Recursively)

- Public-private pair:
  - Public method:
    - Call private method with argument `front`
  - Private method (recursive):
    - If the current node is null, you've reached the end!
      - Just return (base case)
    - If the current node is not null, there's more list!
      - "Read" the current node
      - Keep going! Recursive call with argument `curr.next`

```
public int readThing(){
    return readThing(front);
}

private int readThing(ListNode curr){
    if (curr == null){
        return 0;
    } else{
        return 1 + readThing(curr.next);
    }
}
```

Modifying a Linked List (Recursively)

- Public method:
  - Call private method with argument `front`
  - Assign return value to `front`
- Private method (recursive):
  - If the current node is null, you've reached the end!
    - End/Last case!
  - If the current node is not null, there's more list!
    - "modify" at the current node
    - Keep going! Recursive call with on `curr.next`
    - Assign return value to proper place

```
public void changeList(){
    front = changeList(front);
}
private ListNode changeList(ListNode curr){
    // The previous node will link to what we return
    if (curr == null){
        // End/Last case.
        // Do we need to add a node here?
        return new ListNode(0); //if so, return it!
    } else{
        // Middle Case
        // Our jobs:
        // 1) Modify the list at curr (e.g. add a node)
        // 2) Do a recursive call, get link to the node returned
        // 3) return what the previous node should link to
            curr.next = changeList(curr.next);
            return curr;
        }
}
```

`x = change(x)`

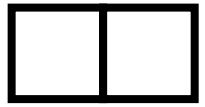
- Pattern used to modify a linked data structure
  - E.g. linked lists and trees (soon!)

- `x` is a reference to the first node in the data structure
- `change` is a method that modifies a data structure, starting from the node `x`
  - It returns the “new” first thing

Modifying a Linked List (Recursively)

A chain of nodes, already modified, it will link to what we return

A chain of nodes, not yet modified, we will link to what this returns



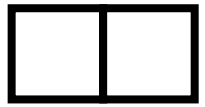
`curr`

What do we do with `curr`?

1) Check for base case
2) Modify the “neighborhood” of `curr`
3) Do a recursive call
4) Link things up
5) Return node previous should link to

A chain of nodes, already modified, it will link to what we return

A chain of nodes, not yet modified, we will link to what this returns



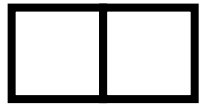
curr

```
public void changeList(){
    front = changeList(front);
}
private ListNode changeList(ListNode curr){
    if (curr == null){
        return new ListNode(0);
    } else{
            curr.next = changeList(curr.next);
            return curr;
        }
    }
```

`removeAll()` – data doesn't match value

A chain of nodes, already modified, it will link to what we return

A chain of nodes, not yet modified, we will link to what this returns

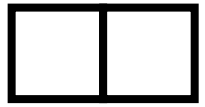


`curr`

`removeAll()` – data matches value

A chain of nodes, already modified, it will link to what we return

A chain of nodes, not yet modified, we will link to what this returns

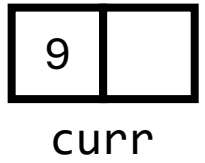


`curr`

duplicateEvens()

A chain of nodes, already modified, it will link to what we return

A chain of nodes, not yet modified, we will link to what this returns



duplicateEvens()

A chain of nodes, already modified, it will link to what we return

A chain of nodes, not yet modified, we will link to what this returns