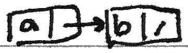
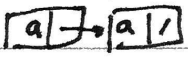
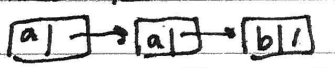
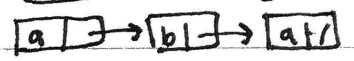
MICP #5

TALK

- check if a LL is a palindrome
- values in LL? any ASCII characters
- null input? throw exception
- return type? boolean
- type of LL? singly-linked
- length of LL? not given

EXAMPLE

INPUT	CLASS	OUTPUT
null	null input	exception
	single node	true
	2-node palindrome F	false
	2-node palindrome T	true
	3-node palindrome F	false
	3-node palindrome T	true

BRUTE FORCE

- ~~have 2 pointers~~
- ~~have~~
- create a copy of the linked list
- reverse the linked list
- loop through both LL at the same time (reversed + original)
  check if the values of the nodes are equal
  return false if not equal, else continue looping

return true at end

OPTIMIZE

- while looping through to reverse the LL, keep a count of # of nodes
- while looping through both LLs, don't need to loop through all of it, stop halfway

WALK THROUGH

Input validation

↓

make a copy of LL

↓

reverse the copy, keep node count

↓

loop through both LLs

↓

check values

↓ True

False → return false

continue looping

↓

return true at end

IMPLEMENTATION

```
isPalindrome(Node head):
    if (head == null)
        throw new Exception();
    copy = copyLL(head);
    nodeCount = 0;

    Node prev = null
    Node curr = copy
    Node next = null
    while (curr != null):
        next = curr.next
        curr.next = prev
        curr = next
        nodeCount++
    copy = prev
    stoppingPoint = floored(nodeCount / 2)
    while stoppingPoint > 0:
        if copy.value != head.value:
            return false
        copy = copy.next
        head = head.next
        stoppingPoint--
    return true
```

prev = curr

copy = copy.next
head = head.next

```
copyLL(Node head):
    node = null
    while head != null
        node
    if head == null:
        return null
    else:
        return Node(head.value,
                    copyLL(head.next))
```

TEST

IN	OUT
null	exception
single node: 'a'	true
single node: '!'	true
2 nodes: a → b	false
2 nodes: a → a	true
3 nodes: a → b → b	false
3 nodes: a → b → a	true