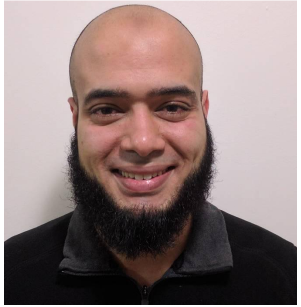
Data Structures

Some Drawing 4 Solutions

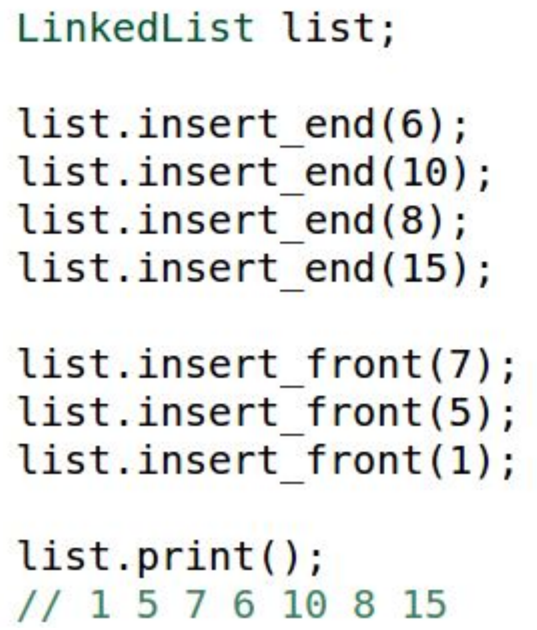
Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



Problem #2: Insert front

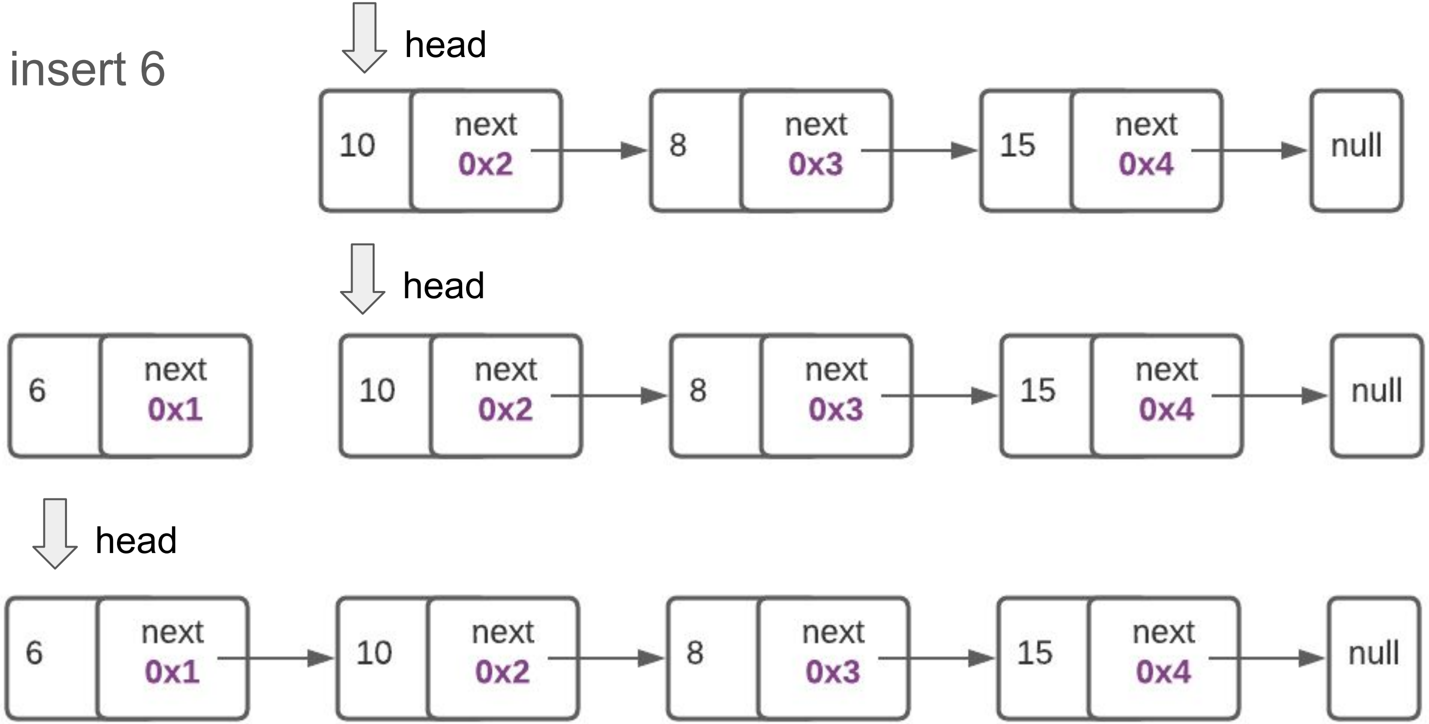
- We implemented insert_end in the lecture
- We want to be able to insert front as following



```
LinkedList list;

list.insert_end(6);
list.insert_end(10);
list.insert_end(8);
list.insert_end(15);

list.insert_front(7);
list.insert_front(5);
list.insert_front(1);

list.print();
// 1 5 7 6 10 8 15
```

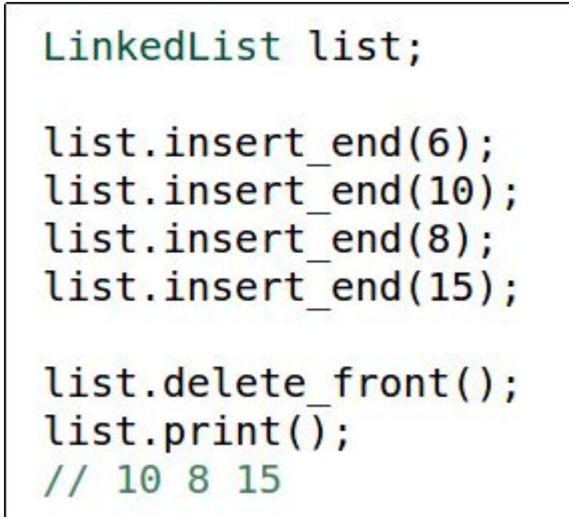
Problem #2: Insert front

- Let’s insert 6



Problem #3: Delete front

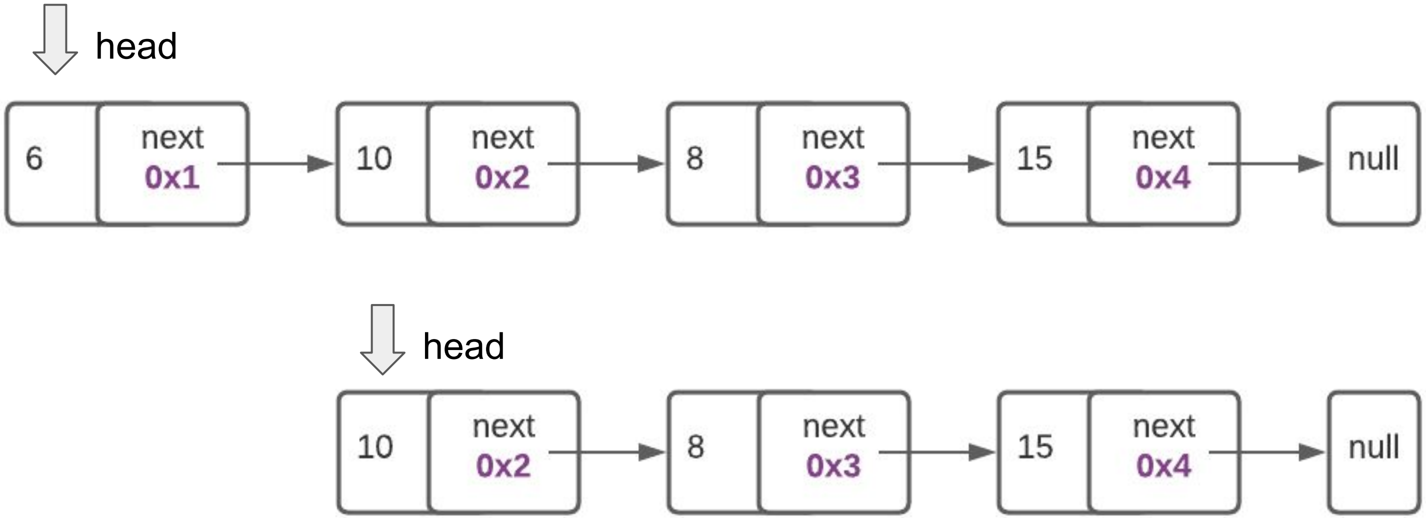
- The opposite of insert front



```
LinkedList list;

list.insert_end(6);
list.insert_end(10);
list.insert_end(8);
list.insert_end(15);

list.delete_front();
list.print();
// 10 8 15
```

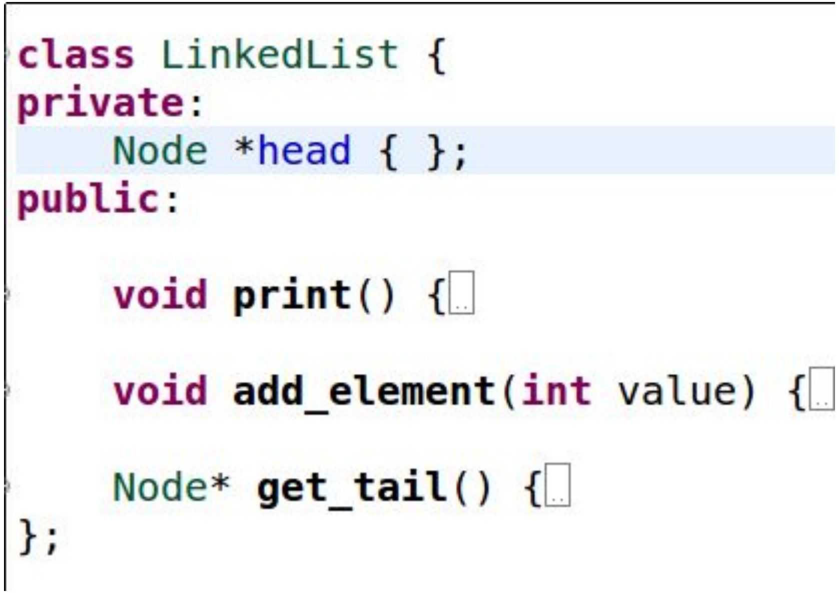
Problem #3: Delete front

- Exactly the opposite
- The head next will be the new head



Problem #6: Linked List without tail/length!

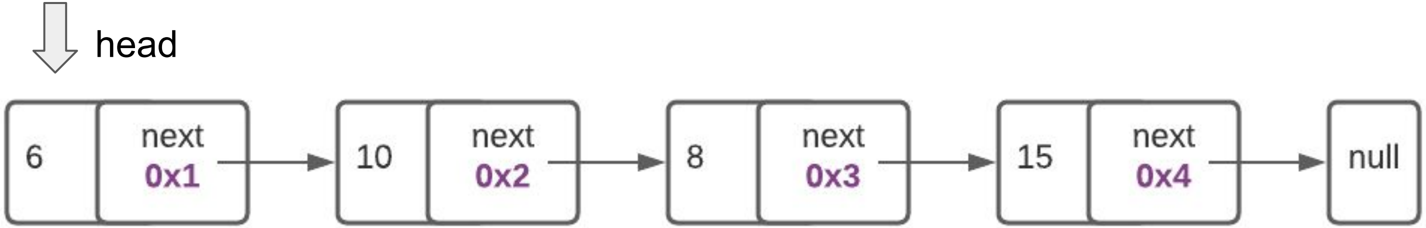
- Assume we will implement our linked list to only have head but no tail
- Implement and test these 3 methods
  - Add element just add element to our current collection of numbers
  - Get tail get the last node



```
class LinkedList {
private:
    Node *head { };
public:

    void print() {

    void add_element(int value) {

    Node* get_tail() {
};
```

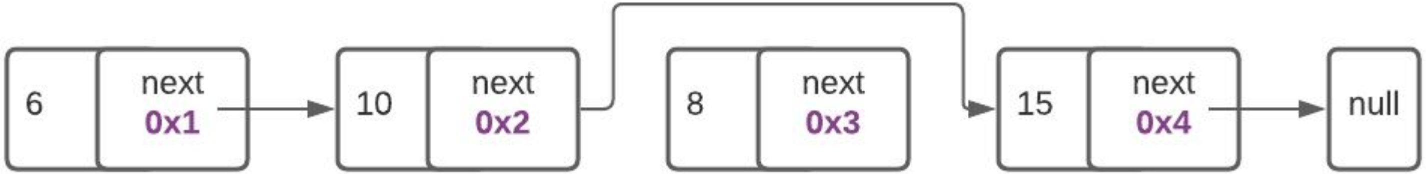
Problem #6: Linked List without tail/length!

- We normally insert_end 6, 10, 6, 8, 15 to get this list
    - We use tail node to add to tail
- Without tail, we must insert them 15, 8, 6, 10
    - We will create new head and link it with previous head



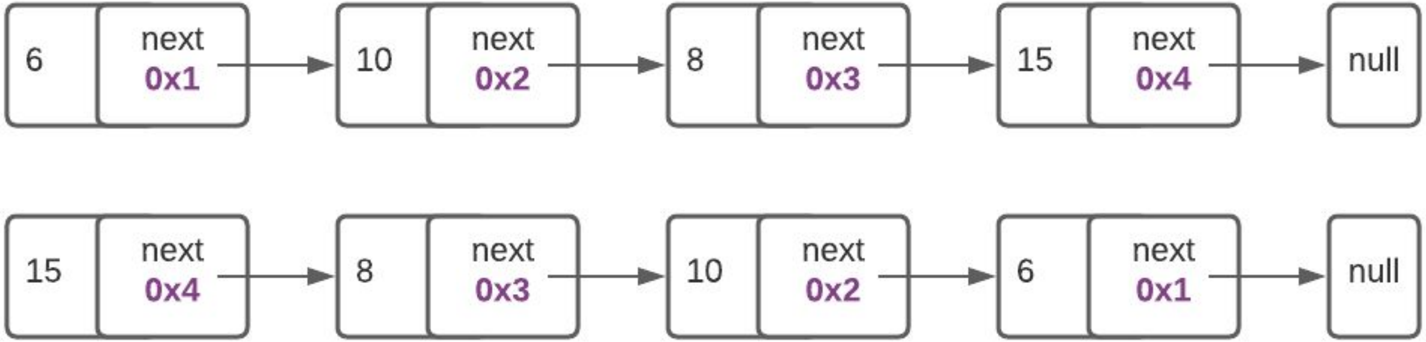
Problem #1: Delete with key

- Given a list, delete the first node with the given key value
- E.g. {1, **2**, 3, 4, **2**, 6}, key = 2 ⇒ {1, 3, 4, 2, 6}
- void delete_node_with_key(int value)
- Same logic as nth node. You need the node before the target key



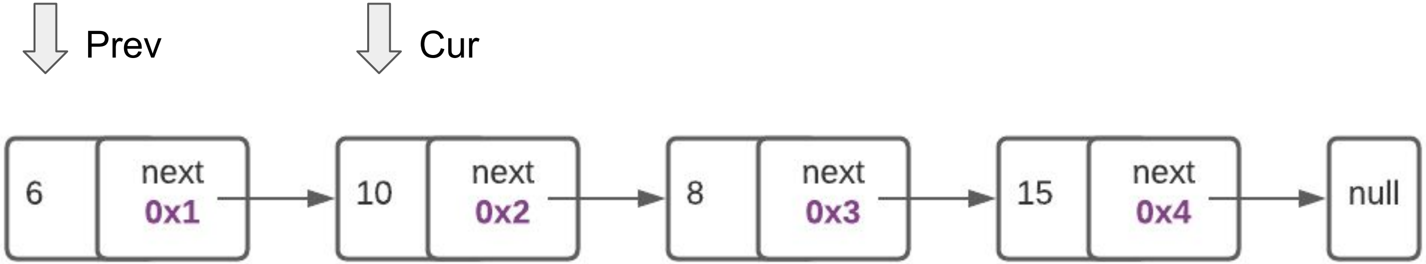
Problem #3: Reverse list nodes

- Given a list, reverse all its nodes (addresses)
- E.g. {1, 2, 3, 4, 5} ⇒ {5, 4, 3, 2, 1}
- void reverse()



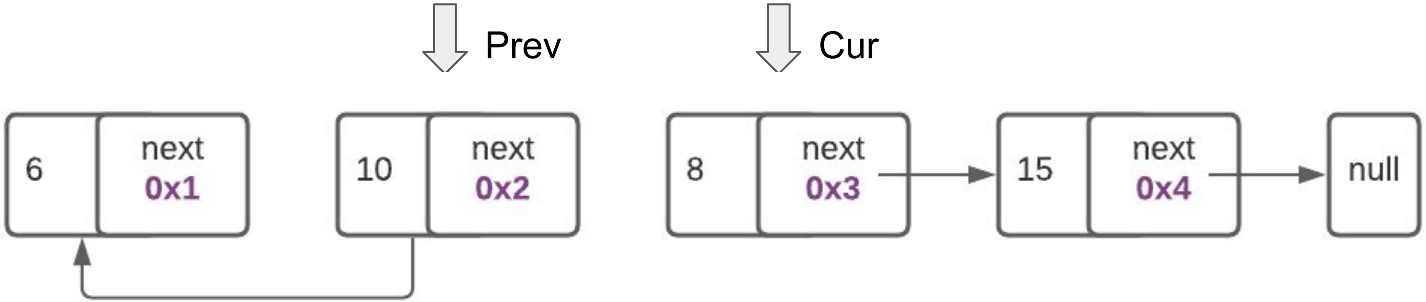
Problem #3: Reverse list nodes

- Start from head, and reverse links one by one
- Assign new head/tail



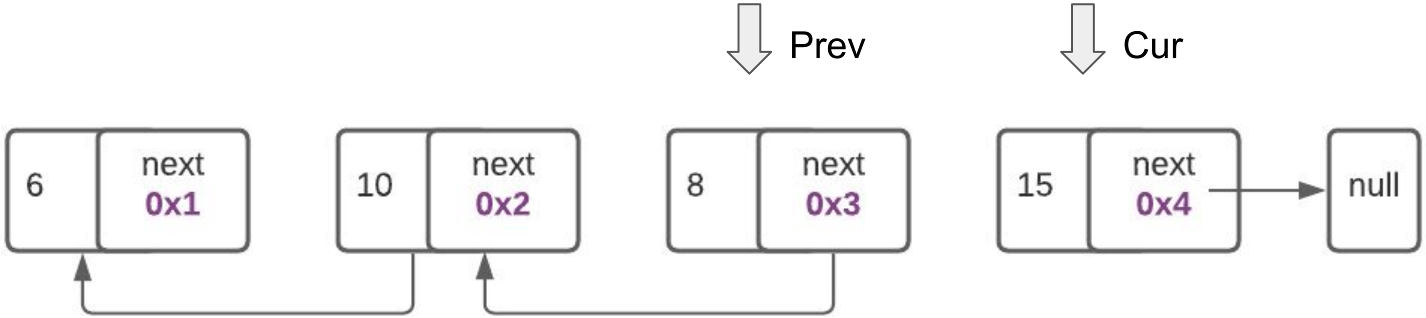
Problem #3: Reverse list nodes

- Swap and move



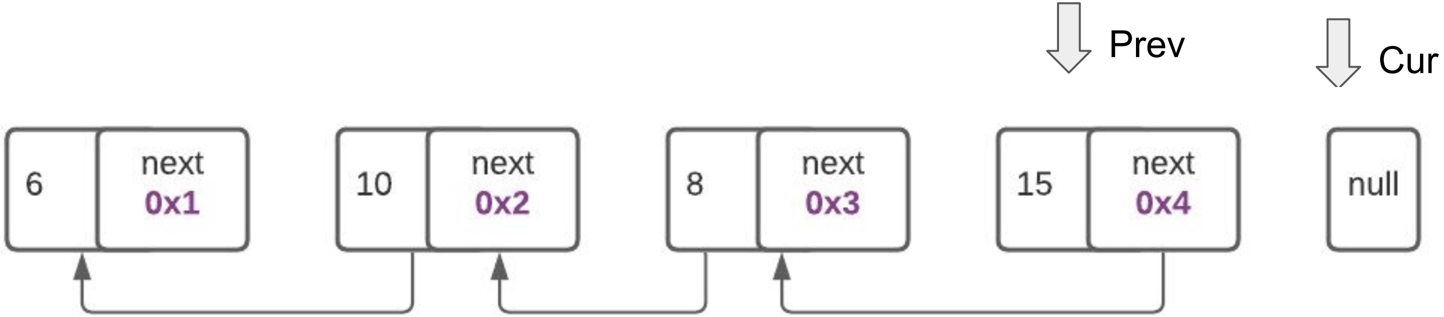
Problem #3: Reverse list nodes

- Swap and move



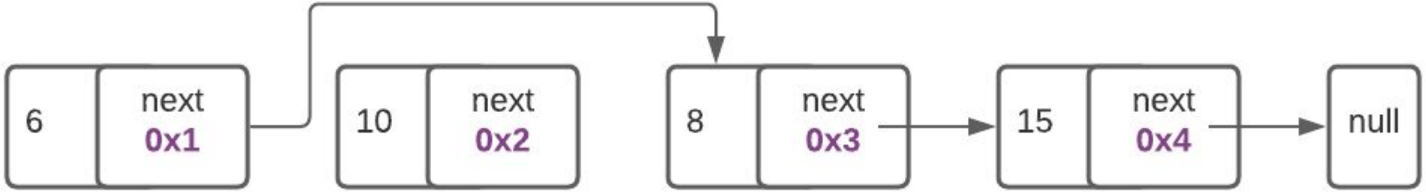
Problem #3: Reverse list nodes

- Swap and move
- Set 15 as head and 6 as tail during the process



Problem #4: Delete even positions

- Given a list, delete all nodes at even positions (2, 4, 6, etc)
- E.g. {1, 2, 3, 4, 5} ⇒ {1, 3, 5}
- E.g. {1, 2, 3, 4, 5, 6} ⇒ {1, 3, 5}
- void delete_even_positions()

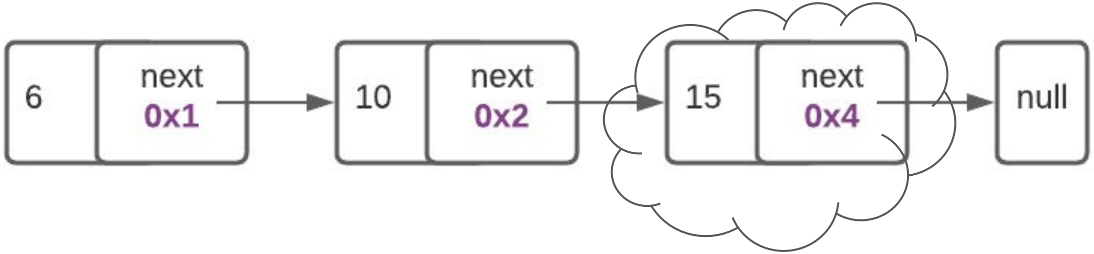


Problem #5: Insert to be sorted

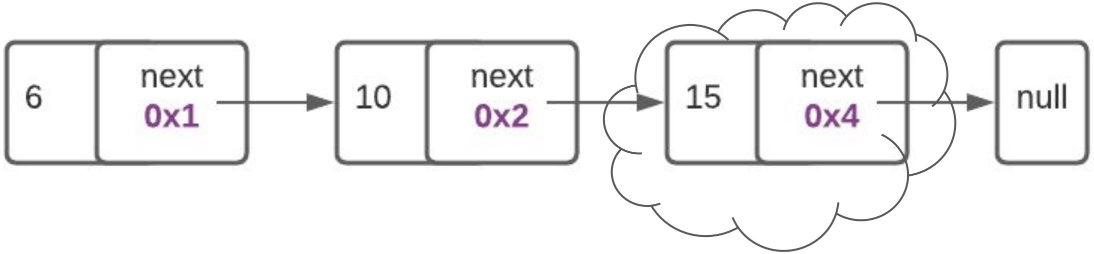
- Implement: void insert_sorted(int value)
- It will always insert the value in position so that list is sorted
- Let’s insert values: 10 2 30 4 1
- insert(10) ⇒ {10}
- insert(2) ⇒ {2, 10}
- insert(30) ⇒ {2, 10, 30}
- insert(4) ⇒ {2, 4, 10, 30}
- insert(1) ⇒ {1, 2, 4, 10, 30}

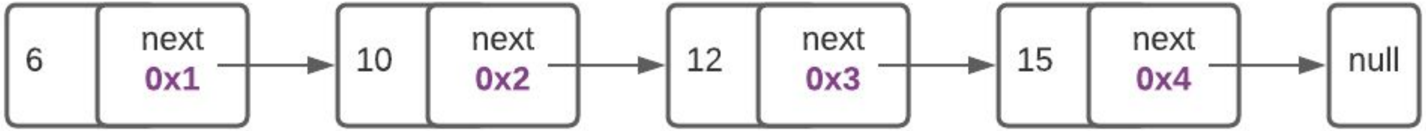
Problem #5: Insert to be sorted

- Let’s assume the current list is sorted: 6, 10, 15
- Now we want insert 12 and we want the list to still be sorted
- We need to find the first node where 12 <= its value
- Then we insert before it



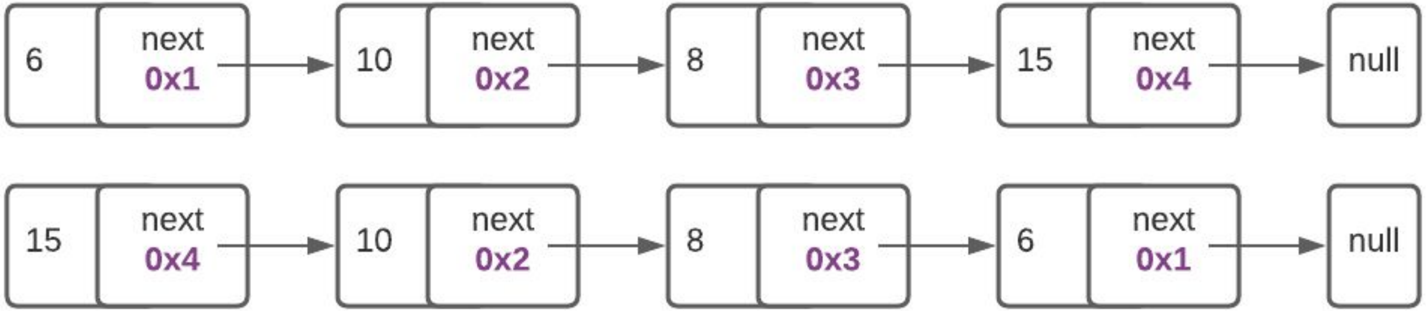
Problem #5: Insert to be sorted





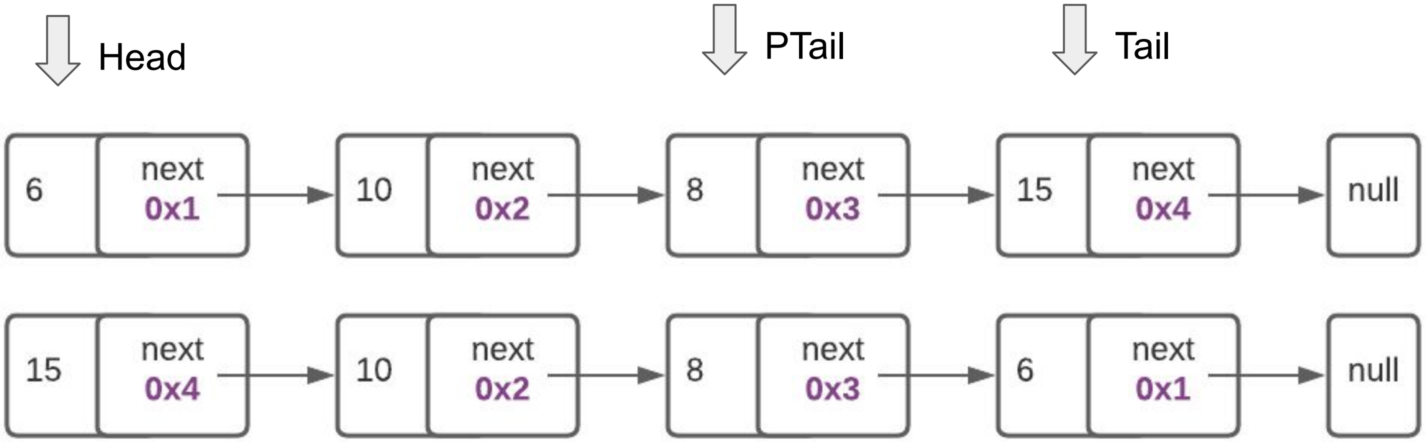
Problem #1: Swap head and tail

- Given a list, we would like swap the head node with the tail now
  - **NODES swap** (addresses) not just values swap
  - See the before and after before. Observe the addresses
- Tip: Draw step by step your procedure. This will save a lot of your time
- Make sure to print after the swap: once the values and also the addreses



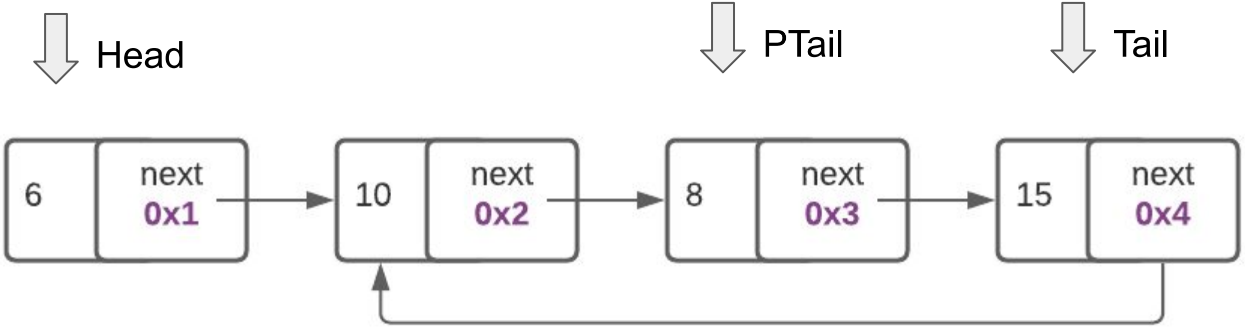
Problem #1: Swap head and tail

- To achieve the target, we need to connect the tail to the 2nd node
- And the node before tail to the head and cancel the head
- In other words, swapping actual nodes



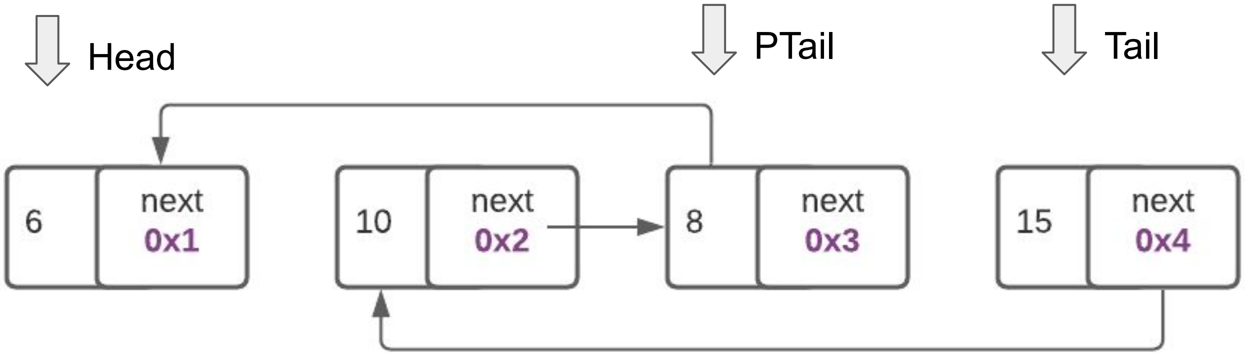
Problem #1: Swap head and tail

- Compute the node before the tail
- Create circle: connect Tail to Head’s next



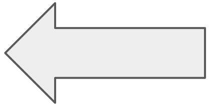
Problem #1: Swap head and tail

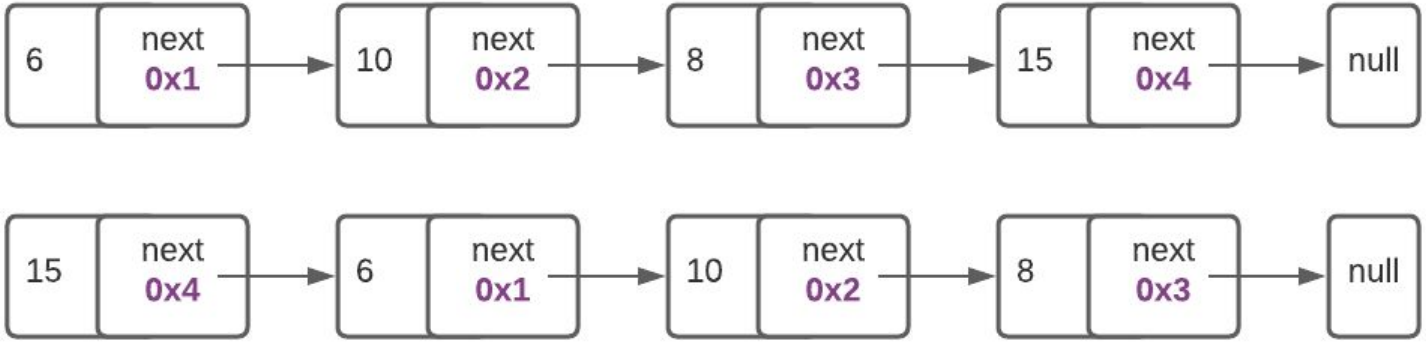
- Connect Ptail with head and cancel head’s next
- Now items are swapped



Problem #2: Left Rotate

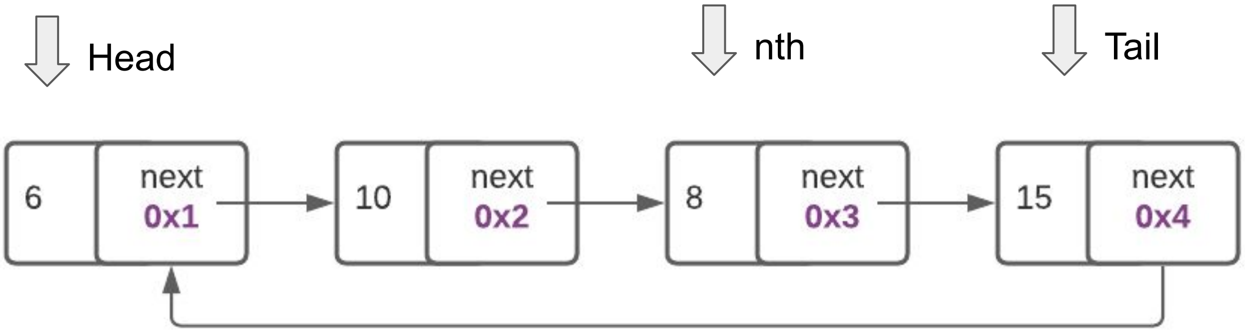
- Given a list, we would like left rotate it k steps (k up to 200000000)
    - Takes the first k elements and shift to back
- void left_rotate(int k): Your code should be O(n) time
- Below list rotated with k = 3   (nodes 6 10 8 shifted back)
    - If k = 1 ⇒ {10, 8, 15, 6}
    - If k = 2 ⇒ {8, 15, 6, 10}





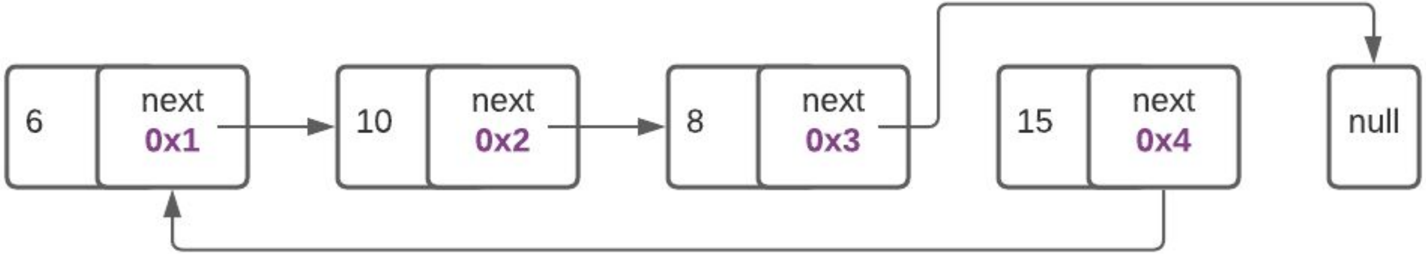
Problem #2: Left Rotate

- Assume n = 3: Find nth node
- Connect Circle



Problem #2: Left Rotate

- Now, prepare the nth node to be the new tail and after it the new head
- What about huge K? Just use k %= length to remove useless cycles
    - Think like clock: every 12 hours are not useless

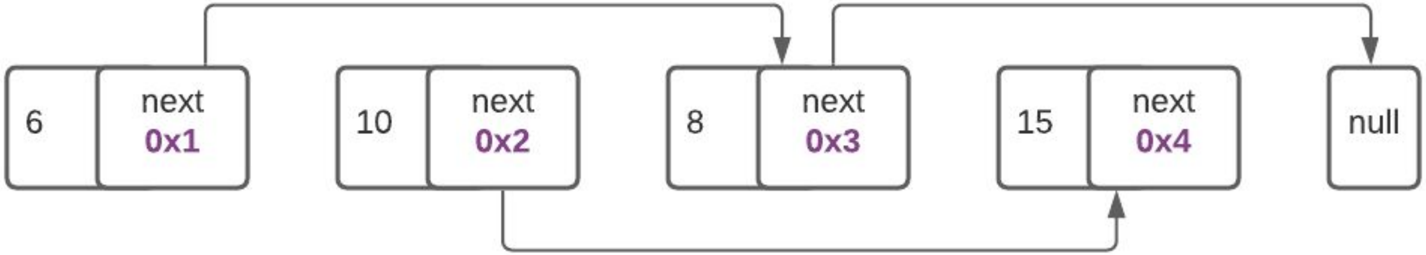


Problem #1: Arrange odd & even nodes

- This problem is not about nodes values, but their order (odd & even)
- Rearrange the nodes so that, odd nodes comes first and even nodes comes last
- E.g. if list is 10, 20, 3, 7, 15: Nodes (10, 3, 15) are at odd positions
- 1, 2, 3, 4 ⇒ 1, 3, 2, 4
- 1, 2, 3 ⇒ 1, 3, 2
- 1, 2, 3, 4, 5, 6, 7 ⇒ 1 3 5 7 2 4 6
- 11 33 55 4 50 17 8 ⇒ 11 55 50 8 33 4 17

Problem #1: Arrange odd & even nodes

- Create 2 chains: one for odd positions and one for even
- Then connect the end of the odd with the first of the even
- Consider: for odd length sequence, tail is the last even!

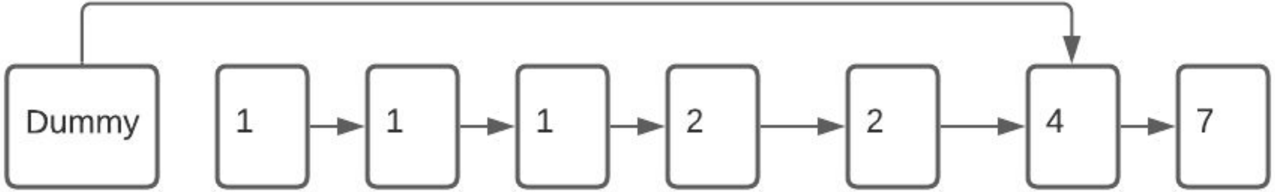


Problem #4: Remove all repeated

- Given linked list of **sorted** integers, keep only nodes that **never repeated** and remove everything else (duplicate nodes)
- Input: 1, 1, 5, 2, 2, 2, 3 ⇒ {5, 3}   both 1 and 2 are repeated
- Input: 1, 1 ⇒ {}
- Input: 1, 1, 2, 2, 2 ⇒ {}
- Input: 1, 1, 2, 2, 2, 5 ⇒ {5}
- Input: 1, 2, 2, 2, 3 ⇒ {1, 3}
- Caution: Coding this problem may drain your time
  - Think about several test cases
  - Draw & verify!

Problem #4: Remove all repeated

- Assume the list is: 1 1 1 2 2 4 7 ⇒ 4 7
- As many first nodes can be removed, let’s do simple coding trick
  - Add a dummy head node and later remove it
- Keep removing blocks of node of equal values
- Once not found, connect the previous node (initially dummy) with this found
- Keep doing so
- Maintain the new tail during the removal
- Actual head is the next of the dummy head



“Acquire knowledge and impart it to the people.”

“Seek knowledge from the Cradle to the Grave.”