# Question

Remove all elements from a linked list of integers that have value ***val***.

**Example:**

**Input:** 1->2->6->3->4->5->6, ***val*** = 6

**Output:** 1->2->3->4->5

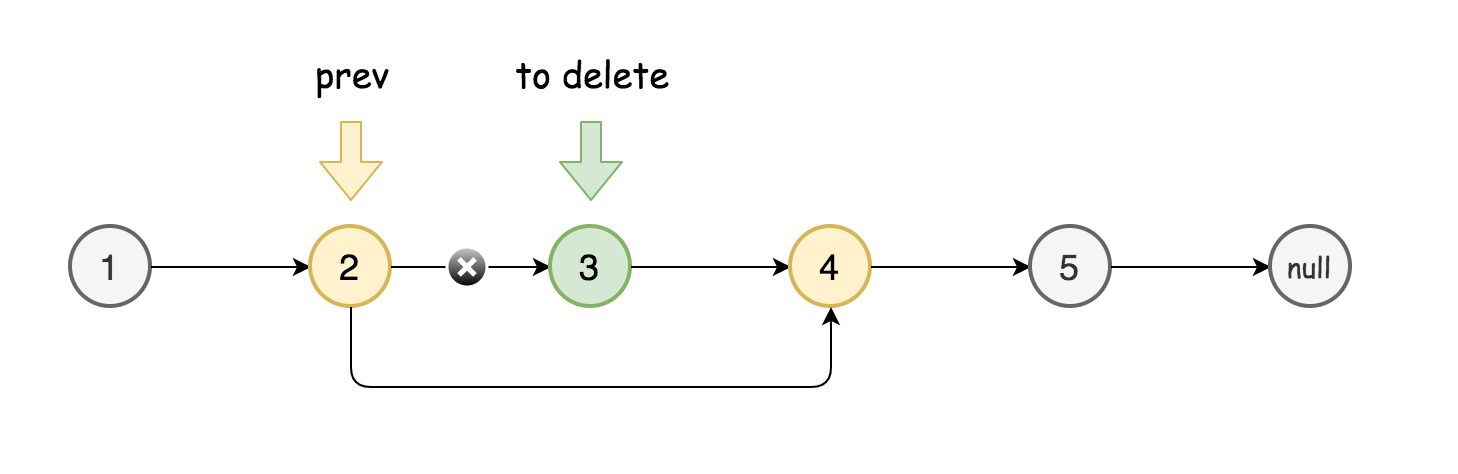
# Solution

#### **Approach 1: Sentinel Node**

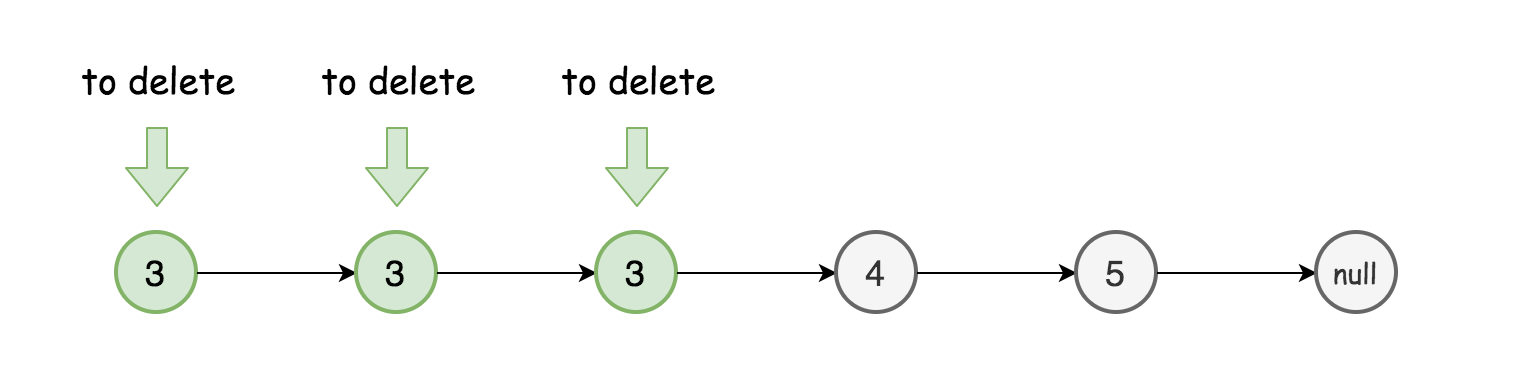
**Intuition**

The problem seems to be very easy if one has to delete a node in the middle:

* Pick the node-predecessor prev of the node to delete.
* Set its next pointer to point to the node next to the one to delete.



The things are more complicated when the node or nodes to delete are in the head of linked list.

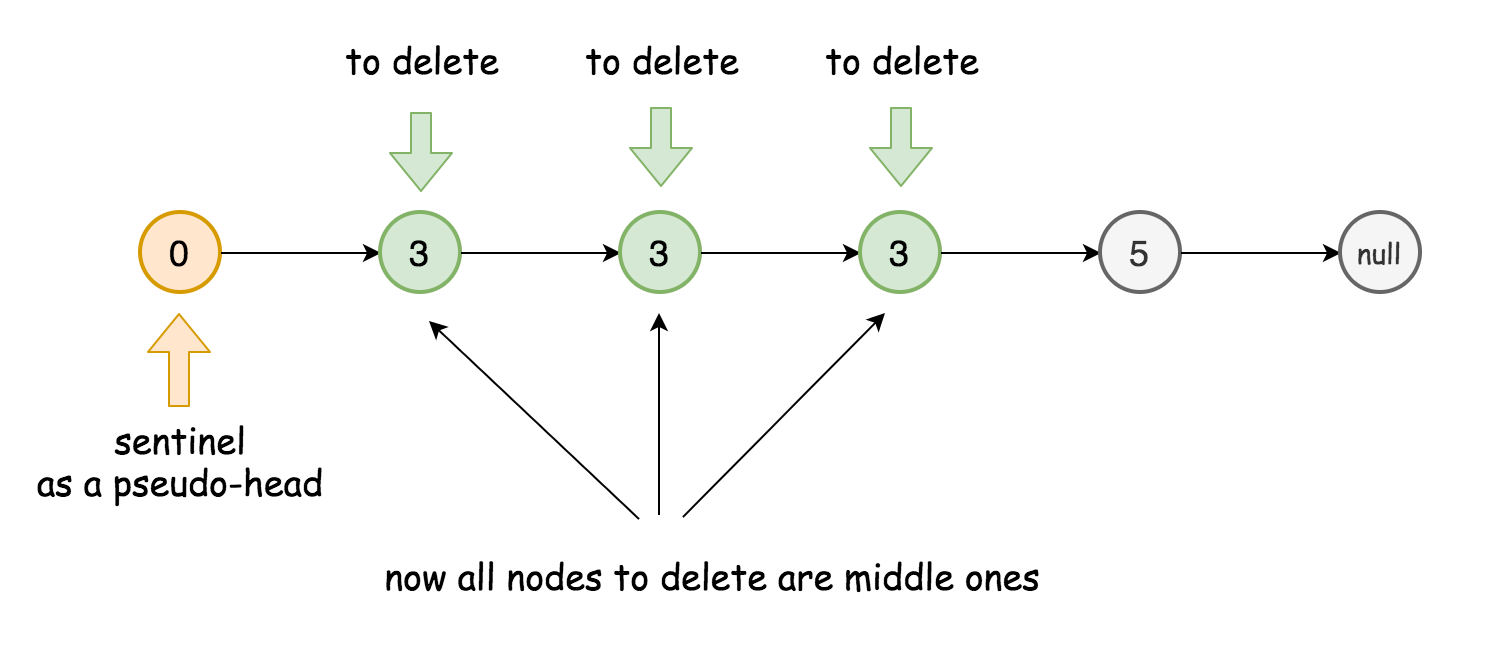


How to deal with that? To reduce the problem to the deletion of middle nodes with the help of [sentinel node](https://en.wikipedia.org/wiki/Sentinel_node).

Sentinel nodes are widely used in trees and linked lists as pseudo-heads, pseudo-tails, markers of level end, etc. They are purely functional, and usually does not hold any data. Their main purpose is to standardize the situation, for example, make linked list to be never empty and never headless and hence simplify insert and delete.

Here are two standard examples:

* [LRU Cache](https://leetcode.com/articles/lru-cache/), sentinel nodes are used as pseudo-head and pseudo-tail.
* [Tree Level Order Traversal](https://leetcode.com/articles/maximum-level-sum-of-a-binary-tree/), sentinel nodes are used to mark level end.



Here sentinel node will be used as pseudo-head.

**Algorithm**

* Initiate sentinel node as ListNode(0) and set it to be the new head: sentinel.next = head.
* Initiate two pointers to track the current node and its predecessor: curr and prev.
* While curr is not a null pointer:
  + Compare the value of the current node with the value to delete.
    - In the values are equal, delete the current node: prev.next = curr.next.
    - Otherwise, set predecessor to be equal to the current node.
  + Move to the next node: curr = curr.next.
* Return sentinel.next.

**Implementation**

|  |
| --- |
| class Solution {  public ListNode removeElements(ListNode head, int val) {  ListNode sentinel = new ListNode(0);  sentinel.next = head;  ListNode prev = sentinel, curr = head;  while (curr != null) {  if (curr.val == val) prev.next = curr.next;  else prev = curr;  curr = curr.next;  }  return sentinel.next;  }  } |

**Complexity Analysis**

* Time complexity: \mathcal{O}(N)O(*N*), it's one pass solution.
* Space complexity: \mathcal{O}(1)O(1), it's a constant space solution.