class Solution {

public:

ListNode\* deleteMiddle(ListNode\* head) {

if (head == nullptr || head->next == nullptr) {

return nullptr; // Empty list or single node, nothing to delete

}

ListNode\* slow = head;

ListNode\* fast = head;

ListNode\* prev = nullptr; // Keep track of the node before the middle

while (fast != nullptr && fast->next != nullptr) {

fast = fast->next->next; // Fast moves two steps

prev = slow; // Prev follows slow

slow = slow->next; // Slow moves one step

}

// Now 'slow' points to the middle node (or the second middle if even)

if (prev == nullptr) { // If prev is null, it means the middle node is the head

head = slow->next; // Update head

} else {

prev->next = slow->next; // Bypass the middle node

}

delete slow; // Deallocate memory of the removed node. Important!

return head;

}

};