# AP ASSIGNMENT

## Name: Abhijeet | UID: 22bcs16832 | Section: 612-“B”

**Print linked list – GFG**

```
class Solution {
public:
    // Function to display the elements of a linked list in the same line without trailing space
    void printList(Node *head) {
        Node* temp = head;
        while (temp != nullptr) {
            cout << temp->data;
            if (temp->next != nullptr) {
                cout << " ";
            }
            temp = temp->next;
        }
        cout << endl;  }
};
```

| Time (IST) | Status | Marks | Lang | Test Cases | Code |
|---|---|---|---|---|---|
| 2025-02-14 15:11:36 | Correct | 0 ? | cpp | 1112 / 1112 | View |
| 2025-02-14 15:11:25 | Correct | 1 | cpp | 1112 / 1112 | View |

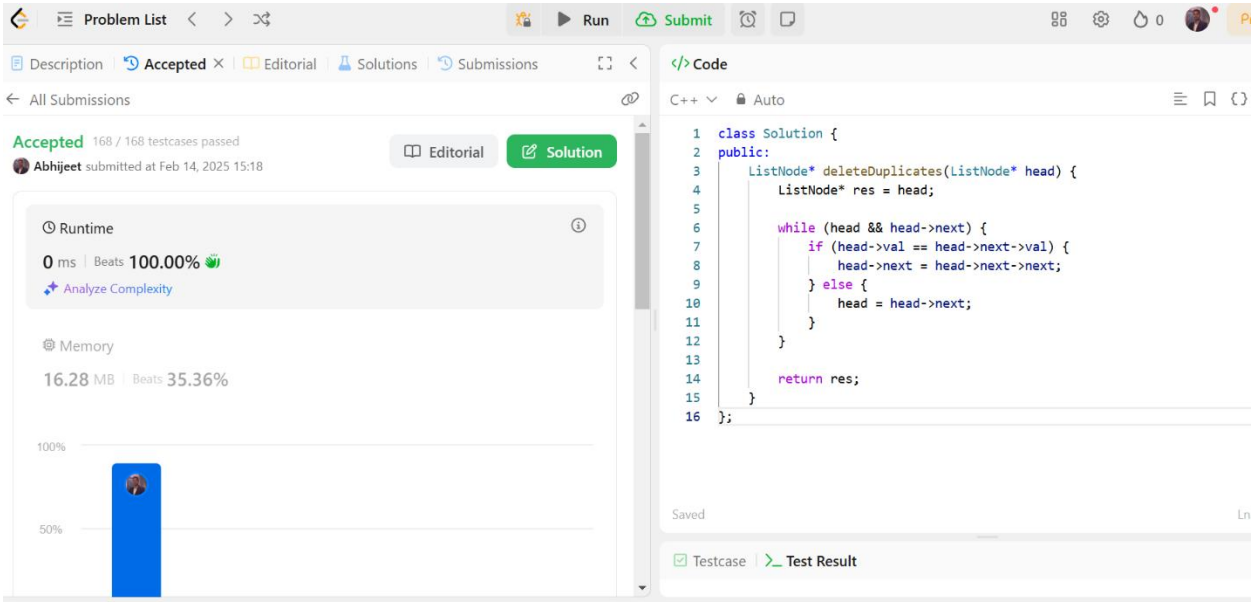
```
        next = nullptr;
    }
};
*/
/*
    Print elements of a linked list on c
    Head pointer input could be NULL as
/*
struct Node {
    int data;
    struct Node* next;

    Node(int x) {
        data = x;
        next = nullptr;
    }
};
*/
/*
    Print elements of a linked list on c
```

Custom

**Remove duplicates from a linkedlist**

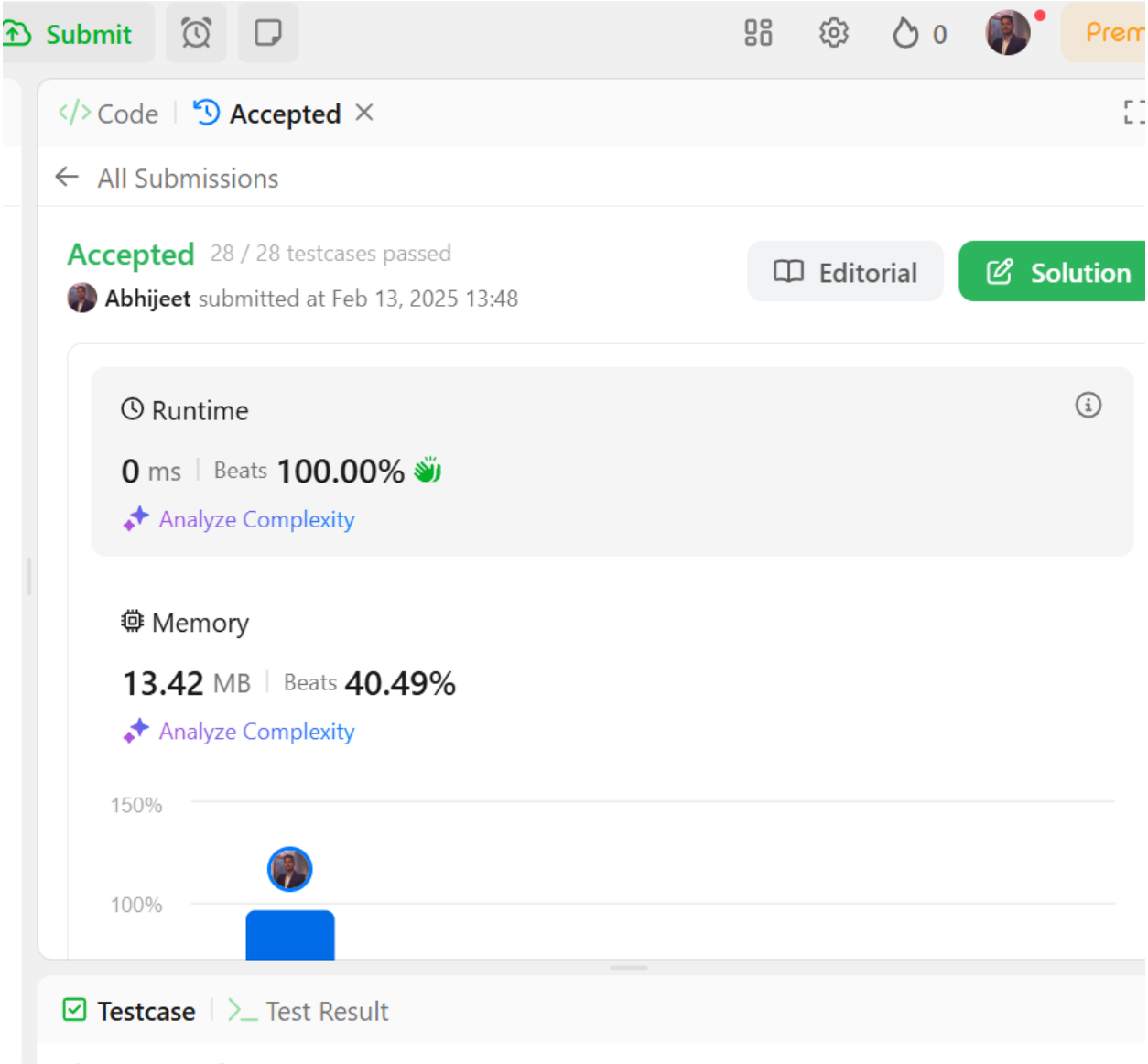
```
class Solution {
```

```
public:
    ListNode* deleteDuplicates(ListNode* head) {
        ListNode* current = head;

        while (current && current->next) {
            if (current->val == current->next->val) {
                current->next = current->next->next;
            } else {
                current = current->next;
            }
        }

        return head;
    }
};
```




**Reverse a linked list**

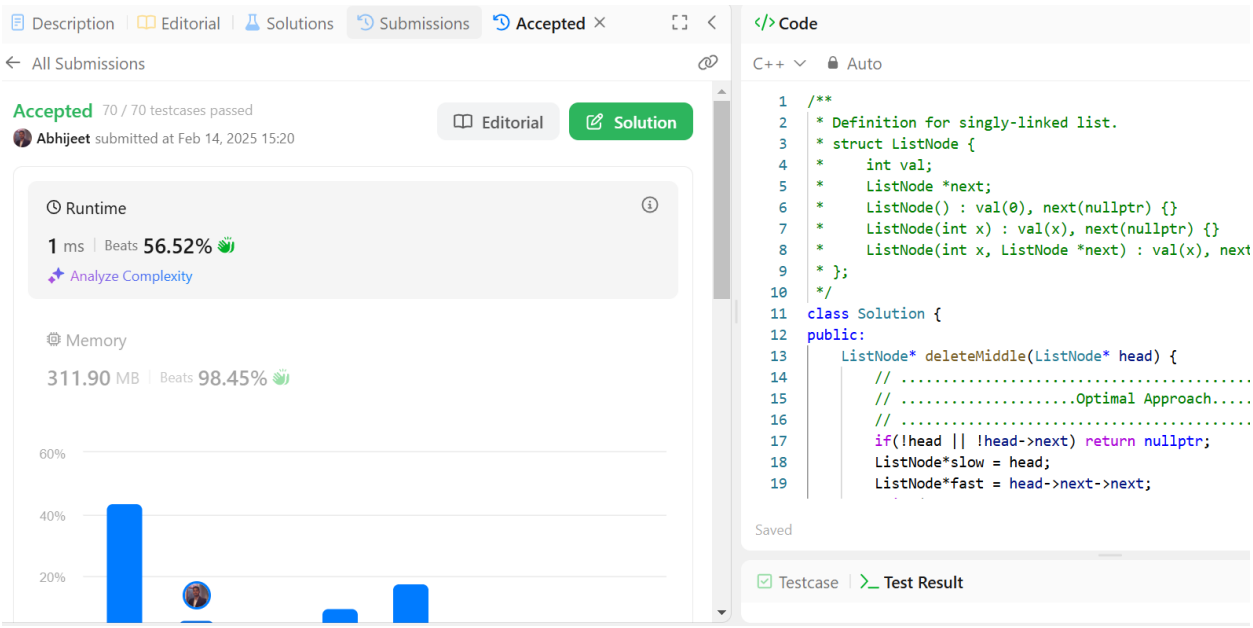
```
class Solution {
public:
   ListNode* reverseList(ListNode* head) {
      ListNode* prev = NULL;
   ListNode* current = head;

   while (current != NULL) {
      ListNode* nextNode = current->next;
      current->next = prev;
      prev = current;
      current = nextNode;
   }
   return prev;
   }
};
```




**Delete middle node of a list**

```
class Solution {
public:
    ListNode* deleteMiddle(ListNode* head) {
        if (!head || !head->next) return nullptr;
        ListNode *slow = head, *fast = head, *prev = nullptr;
        while (fast && fast->next) {
            prev = slow;
            slow = slow->next;
```

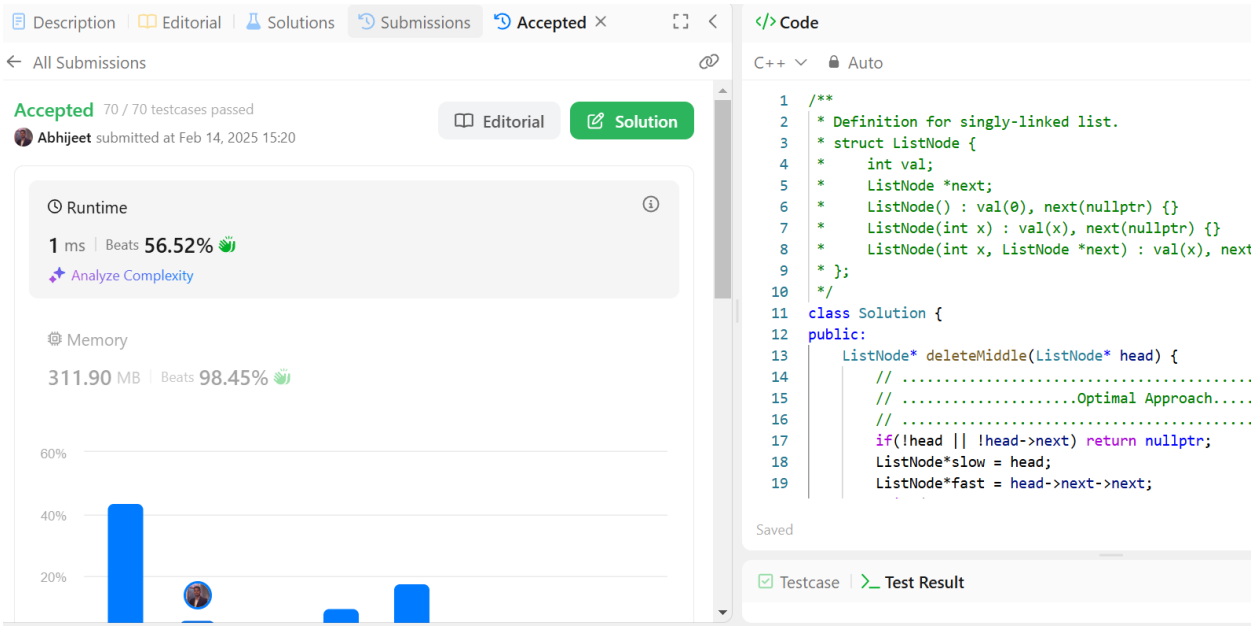
```
        fast = fast->next->next;   }
    prev->next = slow->next;
    return head;
  }
};
```




**<u>Merge two sorted linked lists</u>**

```
class Solution {
public:
  ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
    ListNode dummy(0);
    ListNode* tail = &dummy;

    while (list1 && list2) {
```

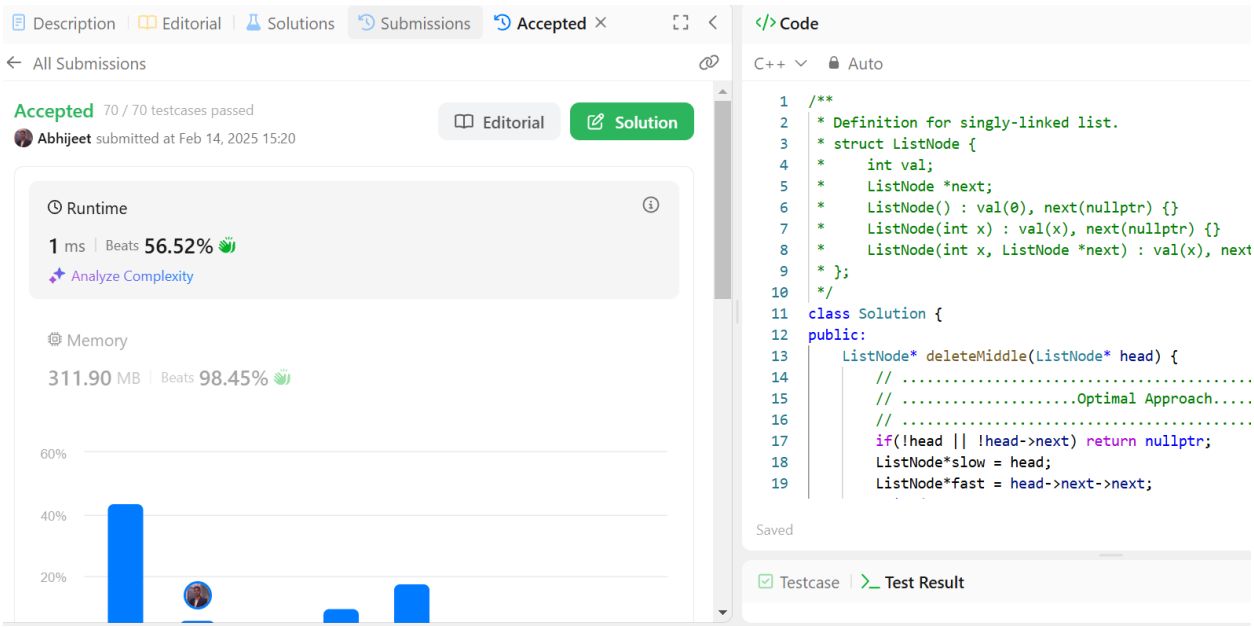
```
        if (list1->val < list2->val) {
            tail->next = list1;
            list1 = list1->next;
        } else {
            tail->next = list2;
            list2 = list2->next;
        }
        tail = tail->next;
    }
    tail->next = list1 ? list1 : list2;
    return dummy.next;
  }
};
```




**Remove duplicates from sorted lists 2**

```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
```

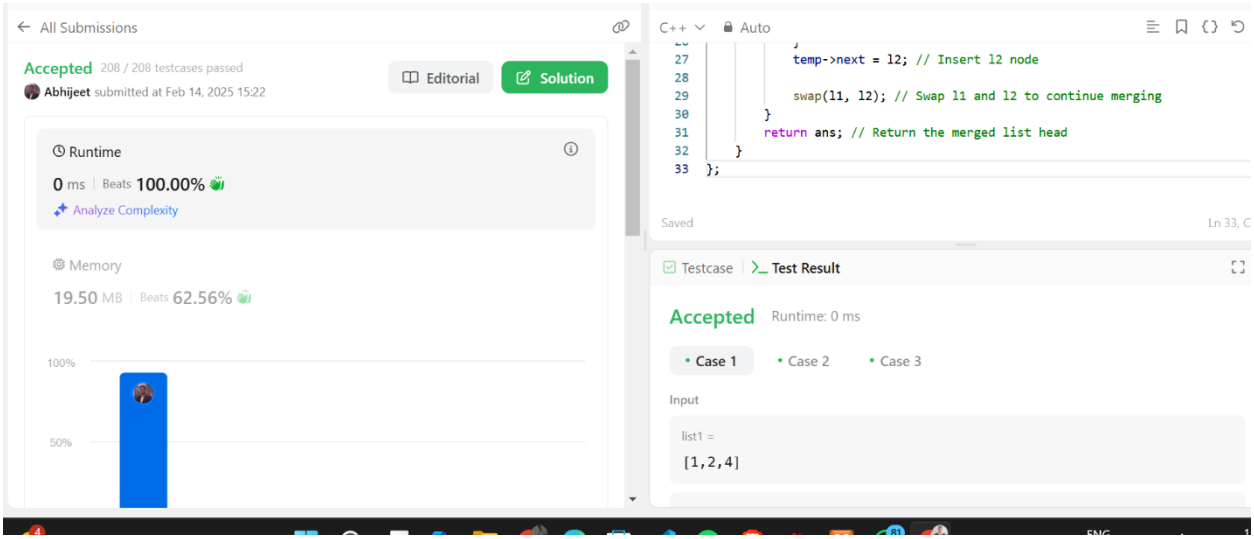
```
        if (!head || !head->next) return head;

        ListNode dummy(0);
        dummy.next = head;
        ListNode* prev = &dummy;

        while (head) {

            if (head->next && head->val == head->next->val) {

                while (head->next && head->val == head->next->val) {
                    head = head->next;
                }
                prev->next = head->next;
            } else {
                prev = prev->next;
            }
            head = head->next;
        }

        return dummy.next;

    }
};
```




**Detect a cycle in a linked list**

```
class Solution {
public:
    bool hasCycle(ListNode *head) {
        if (!head || !head->next) return false;

        ListNode *slow = head;
        ListNode *fast = head;

        while (fast && fast->next) {
            slow = slow->next;
```

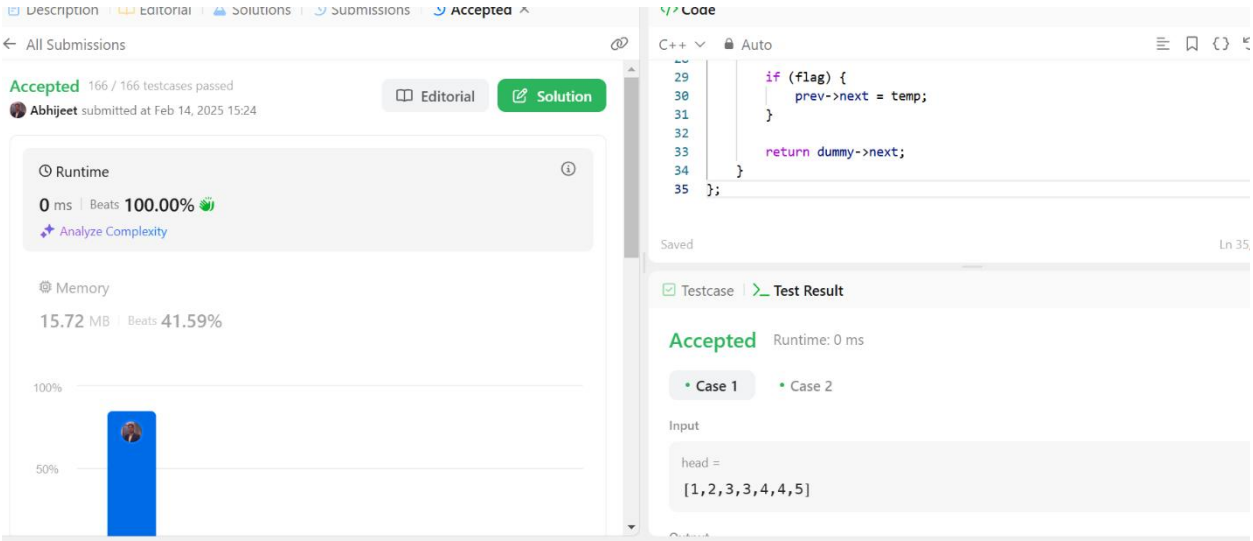
```
            fast = fast->next->next;
            if (slow == fast) return true;
        }
        return false;
    }
};
```




**<u>Reverse linked list 2</u>**

```
class Solution {
public:
    ListNode* reverseBetween(ListNode* head, int left, int right) {
        if (!head || left == right) return head;

        ListNode dummy(0);
        dummy.next = head;
        ListNode* prev = &dummy;

        for (int i = 1; i < left; i++) {
            prev = prev->next;
        }
```

```
        ListNode* curr = prev->next;
        ListNode* nextNode;
        for (int i = 0; i < right - left; i++) {
            nextNode = curr->next;
            curr->next = nextNode->next;
            nextNode->next = prev->next;
            prev->next = nextNode;
        }

        return dummy.next;

    }
};
```




**<u>rotate a list</u>**

```
class Solution {
public:
    ListNode* rotateRight(ListNode* head, int k) {
        if (!head || !head->next || k == 0) return head;
```

```
        ListNode* tail = head;
        int length = 1;
        while (tail->next) {
            tail = tail->next;
            length++;
        }
        k = k % length;
        if (k == 0) return head;
        tail->next = head;
        for (int i = 0; i < length - k - 1; i++) {
            head = head->next;
        }
        ListNode* newHead = head->next;
        head->next = nullptr;
        return newHead;
    }
};
```

| | | | |
|---|---|---|---|
| Accepted<br>Feb 13, 2025 | C++ | 8 ms | 11.7 MB |
| Compile Error<br>Feb 13, 2025 | C++ | N/A | N/A |

```
1  /**
2   * [
3   * s
4   *
```

**Detect a cycle in a linked list 2**

```
class Solution {
public:
    ListNode *detectCycle(ListNode *head) {
        ListNode *slow = head, *fast = head;
```

```
        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;

            if (slow == fast) {
                slow = head;
                while (slow != fast) {
                    slow = slow->next;
                    fast = fast->next;
                }
                return slow;
            }
        }
        return nullptr;
    }
};
```

| | Status | Language | Runtime | Memory | Notes |
|---|---|---|---|---|---|
| 1 | Accepted<br>Feb 13, 2025 | C++ | 0 ms | 11.2 MB | |

C++ Auto

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
```

Saved

Testcase | Test Result

Case 1 Case 2 +

head =

```
[1,2,3,4,5]
```