

## 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 (s)	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

```
26     next = nullptr;
27 }
28 };
29 /*
30 */
31 Print elements of a linked list on c
32 Head pointer input could be NULL as
33 */
34 struct Node {
35     int data;
36     struct Node* next;
37 }
38 Node(int x) {
39     data = x;
40     next = nullptr;
41 }
42 };
43 /*
44 */
45 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;
```

```
}
```

```
};
```

The screenshot shows a code editor interface with a C++ solution for the "Delete Duplicates from Sorted List" problem. The code is as follows:






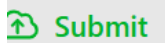

```
1 class Solution {
2 public:
3     ListNode* deleteDuplicates(ListNode* head) {
4         ListNode* res = head;
5
6         while (head && head->next) {
7             if (head->val == head->next->val) {
8                 head->next = head->next->next;
9             } else {
10                 head = head->next;
11             }
12         }
13
14         return res;
15     }
16 };
```

The solution is accepted, with 168 / 168 testcases passed. The runtime is 0 ms, and the memory usage is 16.28 MB. The code is saved and ready for submission.

**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;
    }
};
```

0Prem

</> Code | Accepted ×


← All Submissions


**Accepted** 28 / 28 testcases passed


 **Abhijeet** submitted at Feb 13, 2025 13:48


Editorial

**Solution**


 Runtime

0 ms | Beats 100.00% 

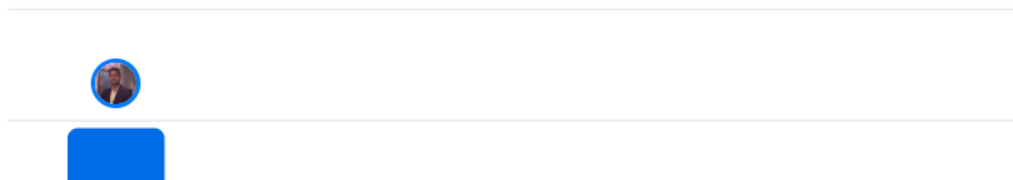
 [Analyze Complexity](#)

 Memory

13.42 MB | Beats 40.49%

 [Analyze Complexity](#)

150%



100%

☒ Testcase

 | 

> Test Result

### 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;

}

};

```

Accepted 70 / 70 testcases passed

Abhijeet submitted at Feb 14, 2025 15:20

Editorial Solution


Runtime

1 ms | Beats 56.52%

Analyze Complexity

Memory

311.90 MB | Beats 98.45%



```

1 /**
2  * Definition for singly-linked list.
3  * struct ListNode {
4  *     int val;
5  *     ListNode *next;
6  *     ListNode() : val(0), next(nullptr) {}
7  *     ListNode(int x) : val(x), next(nullptr) {}
8  *     ListNode(int x, ListNode *next) : val(x), next
9  * };
10 */
11 class Solution {
12 public:
13     ListNode* deleteMiddle(ListNode* head) {
14         // .....
15         // .....Optimal Approach.....
16         // .....
17         if(!head || !head->next) return nullptr;
18         ListNode*slow = head;
19         ListNode*fast = head->next->next;

```

Saved

Testcase Test Result

## Merge two sorted linked lists

```

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;
}
};

```

Accepted 70 / 70 testcases passed  
Abhijeet submitted at Feb 14, 2025 15:20

Runtime  
1 ms | Beats 56.52% 🌱  
[Analyze Complexity](#)

Memory  
311.90 MB | Beats 98.45% 🌱

```

1 /**
2  * Definition for singly-linked list.
3  * struct ListNode {
4  *     int val;
5  *     ListNode *next;
6  *     ListNode() : val(0), next(nullptr) {}
7  *     ListNode(int x) : val(x), next(nullptr) {}
8  *     ListNode(int x, ListNode *next) : val(x), next
9  * };
10 */
11 class Solution {
12 public:
13     ListNode* deleteMiddle(ListNode* head) {
14         // .....
15         // .....Optimal Approach.....
16         // .....
17         if(!head || !head->next) return nullptr;
18         ListNode*slow = head;
19         ListNode*fast = head->next->next;

```

Saved

Testcase Test Result

## 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;

}

};

```

[Description](#)
[Editorial](#)
[Solutions](#)
[Submissions](#)
[Accepted](#)

Accepted

70 / 70 testcases passed

Abhijeet

submitted at Feb 14, 2025 15:20

Editorial

Solution

Runtime

1 ms | Beats 56.52%

Analyze Complexity

Memory

311.90 MB | Beats 98.45%

Code

```

1  /**
2  * Definition for singly-linked list.
3  * struct ListNode {
4  *     int val;
5  *     ListNode *next;
6  *     ListNode() : val(0), next(nullptr) {}
7  *     ListNode(int x) : val(x), next(nullptr) {}
8  *     ListNode(int x, ListNode *next) : val(x), next
9  * };
10 */
11 class Solution {
12 public:
13     ListNode* deleteMiddle(ListNode* head) {
14         // .....
15         // .....Optimal Approach.....
16         // .....
17         if(!head || !head->next) return nullptr;
18         ListNode*slow = head;
19         ListNode*fast = head->next->next;

```

Saved

Testcase

Test Result

## 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;
}

};

```

← All Submissions

Accepted 208 / 208 testcases passed  
Abhijeet submitted at Feb 14, 2025 15:22

Editorial Solution

Runtime  
0 ms | Beats 100.00% 🏆  
[Analyze Complexity](#)

Memory  
19.50 MB | Beats 62.56% 🏆

100%  
50%

C++ Auto

```

27         temp->next = 12; // Insert 12 node
28
29         swap(11, 12); // Swap 11 and 12 to continue merging
30     }
31     return ans; // Return the merged list head
32 }
33 };

```

Saved Ln 33, C

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

list1 =  
[1, 2, 4]

## Reverse linked list 2

```

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;

}

};

```

The screenshot displays a coding platform interface. On the left, the 'Accepted' status is shown with a green checkmark and the text '166 / 166 testcases passed'. Below this, the runtime is '0 ms' and memory is '15.72 MB'. A bar chart shows the memory usage relative to other solutions. On the right, the C++ code is displayed in a split view. The code is as follows:

```

29     if (flag) {
30         prev->next = temp;
31     }
32
33     return dummy->next;
34 }
35 };

```

Below the code, the 'Testcase' and 'Test Result' sections are visible. The test result is 'Accepted' with a runtime of '0 ms'. The input is 'head = [1, 2, 3, 3, 4, 4, 5]'.

## rotate a list

```

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	C++	8 ms	11.7 MB	1	/**
Feb 13, 2025				2	* [
Compile Error	C++	N/A	N/A	3	* ;
Feb 13, 2025				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 <b>Accepted</b> Feb 13, 2025	C++	0 ms	11.2 MB	

C++ Auto

```

1 /**
2  * Definition for singly-linked list.
3  * struct ListNode {
4  *     int val;
5  *     ListNode *next;
6  *     ListNode() : val(0), next(nullptr) {}
7  *     ListNode(int x) : val(x), next(nullptr) {}
8  *     ListNode(int x, ListNode *next) : val(x), next(next) {}

```

Saved

Testcase

> Test Result

Case 1

Case 2

+

head =

[1,2,3,4,5]