

Assignment 3

Advanced Programming Lab – II

Submitted By – Lakshyaraj Singh

UID – 22BCS10687

Section – 22BCS_IOT – 610

Submitted To – Pratima Sonali Horo (E18304)

1. Print Linked List: <https://www.geeksforgeeks.org/problems/print-linked-list-elements/0>

The screenshot displays a coding platform interface with the following components:

- Navigation Bar:** Includes links for Courses, Tutorials, Jobs, Practice, and Contests.
- Problem Header:** Shows the problem title "Print Linked List" and a link to the problem page.
- Compilation Results:** A green banner indicates "Problem Solved Successfully".
- Test Cases Passed:** 1112 / 1112.
- Attempts:** 1 / 1.
- Accuracy:** 100%.
- Points Scored:** 1 / 1.
- Time Taken:** 0.07.
- Your Total Score:** 22.
- Solve Next:** A section with buttons for "Count Linked List Nodes", "Delete Alternate Nodes", and "Insert in Middle of Linked List".
- Code Editor:** Displays C++ code for a linked list. The code includes a Node struct, a Node constructor, and a printList function. The code is as follows:

```
1 // Driver Code Ends
2
3 struct Node {
4     int data;
5     struct Node* next;
6 };
7
8 Node(int x) {
9     data = x;
10    next = nullptr;
11 }
12
13 // Print elements of a linked list on console
14 // Head pointer input could be NULL as well for empty list
15
16 class Solution {
17 public:
18     // Function to display the elements of a linked list in same line
19     void printList(Node* head) {
20         // your code goes here
21         while(head != NULL){
22             cout << head->data << " ";
23             head = head->next;
24         }
25     }
26 };
27
28 // Driver Code Ends
```
- Footer:** Includes a "Custom Input" button and a "Submit" button.

2. Remove Duplicates from a Sorted List: <https://leetcode.com/problems/remove-duplicates-from-sorted-list/description/>

The screenshot shows a LeetCode submission interface. On the left, the 'Submissions' tab is active, showing a table with columns: Status, Language, Runtime, Memory, and Notes. The first submission is 'Accepted' on Feb 14, 2025, with a runtime of 0 ms and memory of 16.1 MB. On the right, the 'Code' editor shows a C++ solution for the 'Remove Duplicates from a Sorted List' problem. The code defines a 'ListNode' struct and a 'Solution' class with a 'deleteDuplicates' method. The method uses a while loop to traverse the list and remove duplicates by skipping nodes with the same value as the previous node. Below the code editor, the 'Test Result' section shows 'Accepted' with a runtime of 0 ms. It displays 'Case 1' with an input list [1,1,2] and an output list [1,2].

```
18 //
19 class Solution {
20 public:
21     ListNode* deleteDuplicates(ListNode* head) {
22         ListNode* curr = head;
23         while(curr != nullptr && curr->next != nullptr){
24             if(curr->val == curr->next->val){
25                 curr->next = curr->next->next;
26             }
27             else{
28                 curr = curr->next;
29             }
30         }
31         return head;
32     }
33 };
34
```

Testcase: Test Result
Accepted Runtime: 0 ms
Case 1 Case 2
Input: head = [1,1,2]
Output: [1,2]
Expected: [1,2]

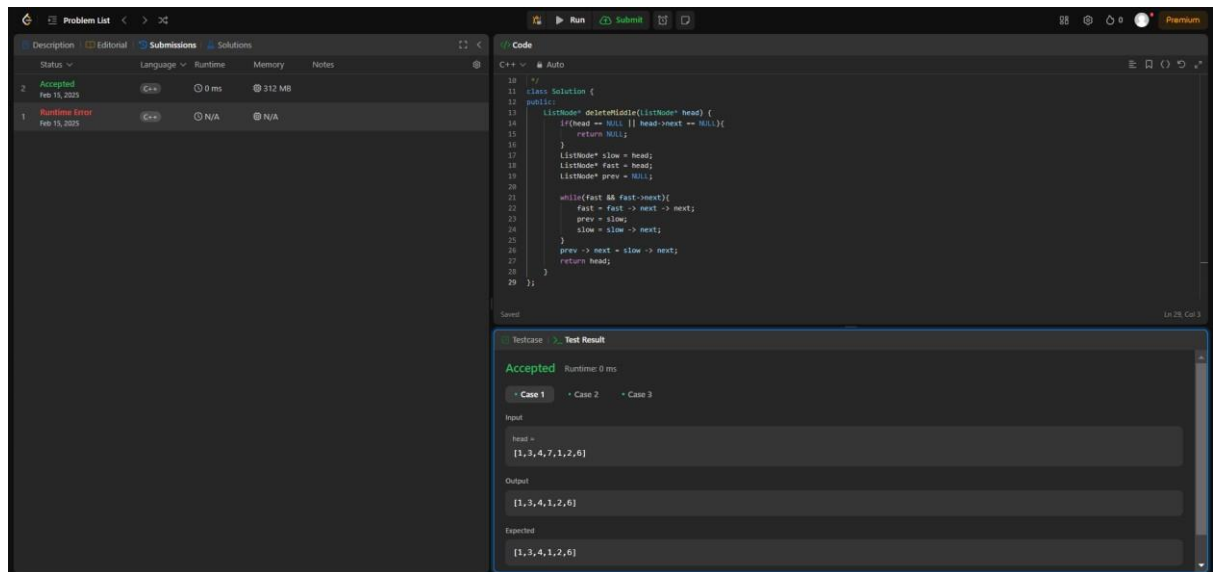
3. Reverse a Linked List: <https://leetcode.com/problems/reverse-linked-list/description/>

The screenshot shows a LeetCode submission interface. On the left, the 'Submissions' tab is active, showing a table with columns: Status, Language, Runtime, Memory, and Notes. The first submission is 'Accepted' on Dec 16, 2024, with a runtime of 0 ms and memory of 13.3 MB. On the right, the 'Code' editor shows a C++ solution for the 'Reverse a Linked List' problem. The code defines a 'ListNode' struct and a 'Solution' class with a 'reverseList' method. The method uses a while loop to traverse the list and reverse the links between nodes. Below the code editor, the 'Test Result' section shows 'Accepted' with a runtime of 0 ms. It displays 'Case 1' with an input list [1,2,3,4,5] and an output list [5,4,3,2,1].

```
19 //
20 class Solution {
21 public:
22     ListNode* reverseList(ListNode* head) {
23         ListNode* currNode = head;
24         ListNode* prevNode = NULL;
25         ListNode* nextNode = NULL;
26         while(currNode){
27             nextNode = currNode->next;
28             currNode->next = prevNode;
29             prevNode = currNode;
30             currNode = nextNode;
31         }
32         return prevNode;
33     }
34 };
35
```

Testcase: Test Result
Accepted Runtime: 0 ms
Case 1 Case 2 Case 3
Input: head = [1,2,3,4,5]
Output: [5,4,3,2,1]
Expected: [5,4,3,2,1]

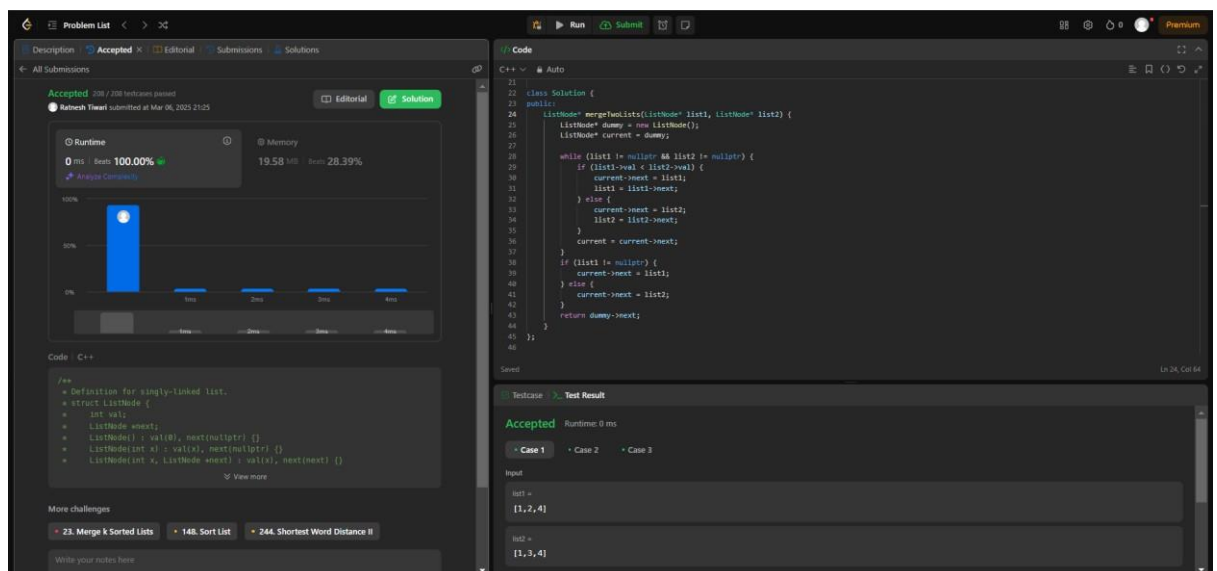
4. Delete middle node of a list: <https://leetcode.com/problems/delete-the-middle-node-of-a-linked-list/description/>



The screenshot shows the LeetCode IDE interface for the problem "Delete the Middle Node of a Linked List". The left sidebar displays the "Submissions" tab with a table showing two submissions: one accepted on Feb 15, 2025, and another with a runtime error on the same date. The main editor shows the C++ code for the solution, which uses a fast and slow pointer technique to find the middle node and delete it. The right sidebar shows the "Test Result" tab, indicating that the solution is "Accepted" with a runtime of 0 ms. The test case shows an input list [1,3,4,7,1,2,6] and an output list [1,3,4,1,2,6], where the middle node (7) has been removed.

```
11 class Solution {
12 public:
13     ListNode* deleteMiddle(ListNode* head) {
14         if(head == NULL || head->next == NULL){
15             return NULL;
16         }
17         ListNode* slow = head;
18         ListNode* fast = head;
19         ListNode* prev = NULL;
20
21         while(fast && fast->next){
22             fast = fast->next;
23             prev = slow;
24             slow = slow->next;
25         }
26         prev->next = slow->next;
27         return head;
28     }
29 }
```

5. Merge two sorted linked lists: <https://leetcode.com/problems/merge-two-sorted-lists/description/>



The screenshot shows the LeetCode IDE interface for the problem "Merge Two Sorted Lists". The left sidebar displays the "Submissions" tab with a table showing one accepted submission on Mar 06, 2025. The main editor shows the C++ code for the solution, which uses a dummy node and a current pointer to merge two sorted linked lists. The right sidebar shows the "Test Result" tab, indicating that the solution is "Accepted" with a runtime of 0 ms. The test case shows two input lists [1,2,4] and [1,3,4], and the output list [1,1,2,3,4], which is the merged result of the two input lists.

```
21 class Solution {
22 public:
23     ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
24         ListNode* dummy = new ListNode();
25         ListNode* current = dummy;
26
27         while(list1 != nullptr && list2 != nullptr) {
28             if(list1->val < list2->val) {
29                 current->next = list1;
30                 list1 = list1->next;
31             } else {
32                 current->next = list2;
33                 list2 = list2->next;
34             }
35             current = current->next;
36         }
37         if(list1 != nullptr) {
38             current->next = list1;
39         } else {
40             current->next = list2;
41         }
42         return dummy->next;
43     }
44 }
```

6. Detect a cycle in a linked list: <https://leetcode.com/problems/linked-list-cycle/description/>

```
1  Accepted
   Dec 21, 2024
   C++
   13 ms
   11.7 MB
```

```
1 //
2 class Solution {
3 public:
4     bool hasCycle(ListNode* head) {
5         if(!head) return false;
6         ListNode* slow = head;
7         ListNode* fast = head;
8         while(fast && fast->next){
9             slow = slow->next;
10            fast = fast->next->next;
11            if(slow == fast){
12                return true;
13            }
14        }
15        return false;
16    }
17 }
```

```
Testcase 1: Test Result
Accepted Runtime: 2 ms
Case 1 Case 2 Case 3
Input:
head = [3,2,0,-4]
pos = 1
Output: true
```

7. Rotate a list: <https://leetcode.com/problems/rotate-list/description/>

```
1  Accepted
   Mar 04, 2025
   C++
   0 ms
   16.5 MB
```

```
1 //
2 class Solution {
3 public:
4     ListNode* rotateRight(ListNode* head, int k) {
5         if(!head || !head->next) return head;
6         ListNode* tail = head;
7         int length = 1;
8         while(tail->next){
9             tail = tail->next;
10            length++;
11        }
12        tail->next = head;
13        k = k % length;
14        if(k == 0) return head;
15        tail->next = nullptr;
16        return head;
17    }
18
19     ListNode* new_tail = head;
20     for(int i = 0; i < length - k - 1; i++){
21         new_tail = new_tail->next;
22     }
23     ListNode* new_head = new_tail->next;
24     new_tail->next = nullptr;
25     return new_head;
26 }
```

```
Testcase 1: Test Result
Accepted Runtime: 0 ms
Case 1 Case 2
Input:
head = [1,2,3,4,5]
```

8. Sort List: <https://leetcode.com/problems/sort-list/description/>

```
class Solution {
public:
    ListNode* sortList(ListNode* head) {
        if (!head || !head->next) {
            return head;
        }

        ListNode* mid = findMiddle(head);
        ListNode* left = sortList(head);
        ListNode* right = sortList(mid);

        return merge(left, right);
    }

private:
    ListNode* findMiddle(ListNode* head) {
        ListNode* slow = head;
        ListNode* fast = head;

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

        ListNode* mid = slow;
        ListNode* prev = nullptr;

        while (head != mid) {
            prev = head;
            head = head->next;
        }

        if (prev) {
            prev->next = nullptr;
        }

        return mid;
    }

    ListNode* merge(ListNode* left, ListNode* right) {
        if (!left) return right;
        if (!right) return left;
```

```

if (left->val < right->val) {
    left->next = merge(left->next, right);
    return left;
} else {
    right->next = merge(left, right->next);
    return right;
}
}
};

```

The screenshot shows a LeetCode submission for the problem "Merge Two Sorted Lists". The submission is in C++ and has been accepted. The runtime is 20 ms, which is 59.72% faster than the average, and the memory usage is 58.12 MB, which is 68.55% less than the average. The code implements a recursive merge function. The test case shows two input lists: [4, 2, 1, 3] and [1, 2, 3, 4], which are merged into the output list [1, 2, 3, 4].

```

class Solution {
public:
    ListNode* sortList(ListNode* head) {
        if (!head || !head->next) {
            return head;
        }

        ListNode* mid = findMiddle(head);
        ListNode* left = sortList(head);
        ListNode* right = sortList(mid);
        return merge(left, right);
    }

private:
    ListNode* findMiddle(ListNode* head) {
        ListNode* slow = head;
        ListNode* fast = head;

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

        ListNode* mid = slow;
        ListNode* next = mid->next;
        mid->next = nullptr;
        return mid;
    }
};

```

9. Merge k sorted lists: <https://leetcode.com/problems/merge-k-sorted-lists/description/>

The screenshot shows a LeetCode submission for the problem "Merge k Sorted Lists". The submission is in C++ and has been accepted. The runtime is 3 ms, which is 64.88% faster than the average, and the memory usage is 18.61 MB, which is 40.31% less than the average. The code uses a priority queue to merge k sorted lists. The test case shows k input lists: [[1, 4, 5], [1, 3, 4], [2, 6]], which are merged into the output list [1, 1, 2, 3, 4, 4, 5, 6].

```

class Solution {
public:
    ListNode* mergeKLists(vector<ListNode*> lists) {
        priority_queue<ListNode*, vector<ListNode*&, function<bool(ListNode*, ListNode*)>> pq(([]<ListNode* a, ListNode* b) {
            return a->val < b->val;
        }));

        for (auto list : lists) {
            if (list) pq.push(list);
        }

        ListNode* dummy = new ListNode(0);
        ListNode* curr = dummy;

        while (!pq.empty()) {
            ListNode* node = pq.top();
            pq.pop();
            curr->next = node;
            curr = curr->next;
            if (node->next) pq.push(node->next);
        }

        return dummy->next;
    }
};

```