

# Assignment 3

## Advanced Programming Lab – II

**Submitted By –** Ratnesh Tiwari

**UID –** 22BCS17201

**Section –** 22BCS\_IOT – 609

**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 a dark theme. On the left, a sidebar contains navigation links: Courses, Tutorials, Jobs, Practice, and Contests. The main area is divided into two panels. The left panel, titled 'Output Window', shows 'Compilation Results' for a problem named 'Print Linked List'. It indicates 'Problem Solved Successfully' with a green checkmark. Below this, it shows 'Test Cases Passed: 1112 / 1112', 'Attempts: Correct / Total: 1 / 1', 'Accuracy: 100%', 'Points Scored: 1 / 1', and 'Time Taken: 0.07'. The right panel shows the C++ code for the solution, which includes a linked list structure and a print function. The code is as follows:

```
1 // Driver Code Ends
19
20 /*
21 * struct Node {
22 *   int data;
23 *   struct Node* next;
24 * }
25 * Node(int x) {
26 *   data = x;
27 *   next = nullptr;
28 * }
29 */
30
31 // Print elements of a linked list on console
32 // Head pointer input could be NULL as well for empty list
33
34
35
36 class Solution {
37 public:
38     // Function to display the elements of a linked list in same line
39     void printList(Node* head) {
40         // your code goes here
41         while(head != NULL){
42             cout << head->data << " ";
43             head = head->next;
44         }
45     }
46 };
47
48 // Driver Code Ends
```

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, 2023, using C++, with a runtime of 0 ms and memory of 16.1 MB. The main area displays the C++ code for the 'deleteDuplicates' function. The code uses a while loop to traverse the linked list and remove duplicates by skipping nodes with the same value as the previous node. Below the code, the 'Test Result' section shows 'Accepted' with a runtime of 0 ms. It includes input, output, and expected results for Case 1: Input is 'head = [1,1,2]', Output is '[1,2]', and Expected is '[1,2]'.

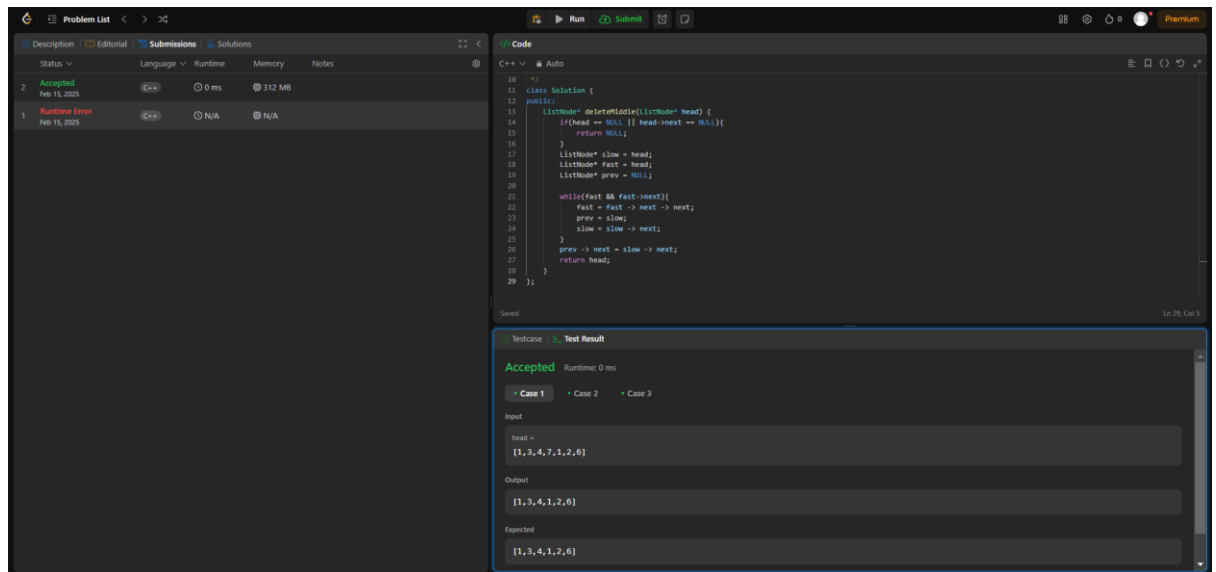
```
25 //
26 class Solution {
27 public:
28     ListNode* deleteDuplicates(ListNode* head) {
29         ListNode* curr = head;
30         while(curr != nullptr && curr->next != nullptr){
31             if(curr->val == curr->next->val){
32                 curr->next = curr->next->next;
33             }
34             else{
35                 curr = curr->next;
36             }
37         }
38         return head;
39     }
40 };
```

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, using C++, with a runtime of 0 ms and memory of 13.3 MB. The main area displays the C++ code for the 'reverseList' function. The code uses an iterative approach with three pointers: 'prevNode', 'currentNode', and 'nextNode' to reverse the linked list. Below the code, the 'Test Result' section shows 'Accepted' with a runtime of 0 ms. It includes input, output, and expected results for Case 1: Input is 'head = [1,2,3,4,5]', Output is '[5,4,3,2,1]', and Expected is '[5,4,3,2,1]'.

```
13 //
14 class Solution {
15 public:
16     ListNode* reverseList(ListNode* head) {
17         ListNode* prevNode = NULL;
18         ListNode* nextNode = NULL;
19         while(currentNode){
20             nextNode = currentNode->next;
21             currentNode->next = prevNode;
22             prevNode = currentNode;
23             currentNode = nextNode;
24         }
25         return prevNode;
26     }
27 };
```

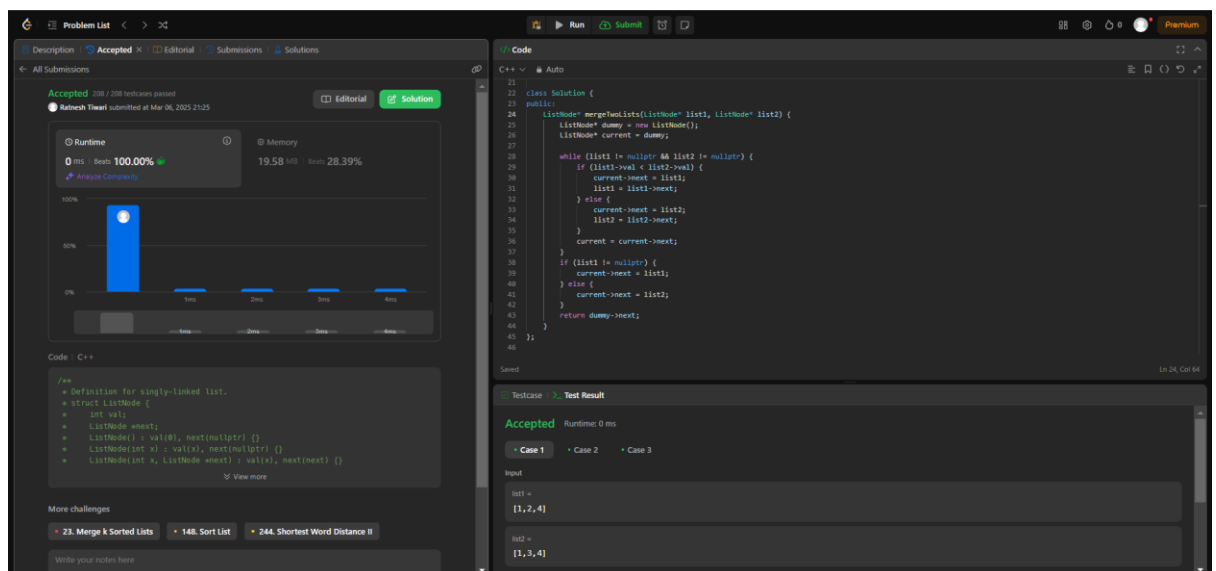
4. Delete middle node of a list: <https://leetcode.com/problems/delete-the-middle-node-of-a-linked-list/description/>



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 a successful submission (Accepted) on Feb 15, 2025, with a runtime of 0 ms and memory usage of 312 MB. The main editor shows the C++ code for the solution, which uses a fast-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 input is [1, 3, 4, 7, 1, 2, 6] and the output is [1, 3, 4, 1, 2, 6].

```
18 //
19 class Solution {
20 public:
21     ListNode* deleteMiddle(ListNode* head) {
22         if(head == NULL || head->next == NULL){
23             return NULL;
24         }
25         ListNode* slow = head;
26         ListNode* fast = head;
27         ListNode* prev = NULL;
28
29         while(fast && fast->next){
30             fast = fast->next->next;
31             prev = slow;
32             slow = slow->next;
33         }
34         prev->next = slow->next;
35         return head;
36     }
37 };
38
```

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 a successful submission (Accepted) on Mar 06, 2023, with a runtime of 0 ms and memory usage of 19.58 MB. The main editor shows the C++ code for the solution, which uses a dummy node and a while loop to merge the two sorted lists. The right sidebar shows the "Test Result" tab, indicating that the solution is "Accepted" with a runtime of 0 ms. The test case input is list1 = [1, 2, 4] and list2 = [1, 3, 4].

```
21 //
22 class Solution {
23 public:
24     ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
25         ListNode* dummy = new ListNode();
26         ListNode* current = dummy;
27
28         while (list1 != nullptr && list2 != nullptr) {
29             if (list1->val < list2->val) {
30                 current->next = list1;
31                 list1 = list1->next;
32             } else {
33                 current->next = list2;
34                 list2 = list2->next;
35             }
36             current = current->next;
37         }
38         if (list1 != nullptr) {
39             current->next = list1;
40         } else {
41             current->next = list2;
42         }
43         return dummy->next;
44     }
45 };
46
```

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

```
1  Description 2  Editorial 3  Submissions 4  Solutions
Status
Accepted
Dec 21, 2024
Language
Runtime
Memory
Notes
C++
13 ms
11.7 MB

Code
C++
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27
//
class Solution {
public:
    bool hasCycle(ListNode* head) {
        if(head == NULL){
            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;
    }
};

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  Description 2  Editorial 3  Submissions 4  Solutions
Status
Accepted
Mar 04, 2025
Language
Runtime
Memory
Notes
C++
0 ms
16.5 MB

Code
C++
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40
class Solution {
public:
    ListNode* rotateRight(ListNode* head, int k) {
        if(!head || !head->next){
            return head;
        }
        ListNode* tail = head;
        int length = 1;
        while(tail->next){
            tail = tail->next;
            length++;
        }
        tail->next = head;
        k = k % length;
        if(k == 0){
            tail->next = nullptr;
            return head;
        }
        ListNode* new_tail = head;
        for(int i = 0; i < length - k - 1; i++){
            new_tail = new_tail->next;
        }
        ListNode* new_head = new_tail->next;
        new_tail->next = nullptr;
        return new_head;
    }
};

Testcase 1 Test Result
Accepted Runtime: 0 ms
Case 1 Case 2
Input
head =
[1,2,3,4,5]
Output
[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 20ms, which is 59.72% of the total allowed time, and the memory usage is 58.12 MB, which is 68.55% of the total allowed memory. The code implements a recursive merge function. The test case shows two input lists: [4, 2, 1, 3] and [1, 2, 3, 4], resulting in the merged output [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;
        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 3ms, which is 64.88% of the total allowed time, and the memory usage is 18.61 MB, which is 40.31% of the total allowed memory. The code implements a priority queue-based solution to merge k sorted lists. The test case shows three input lists: [1, 4, 5], [1, 3, 4], and [2, 6], resulting in the merged output [1, 1, 2, 3, 4, 4, 5, 6].

```

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

class Solution {
public:
    vector<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;
    }
};

```