

Assignment-2

Student Name: Armaan Jaswal

Branch: BE- CSE-General

Semester: 6th

UID: 22BCS11756

Section/Group: 22BCS-IOT-606/B

Date of Submission: 06/03/25

Subject Name: Advanced Programming Lab-2

Subject Code: 22CSP-351

Submitted to: Ms. Pratima Sonali Horo(E18304)

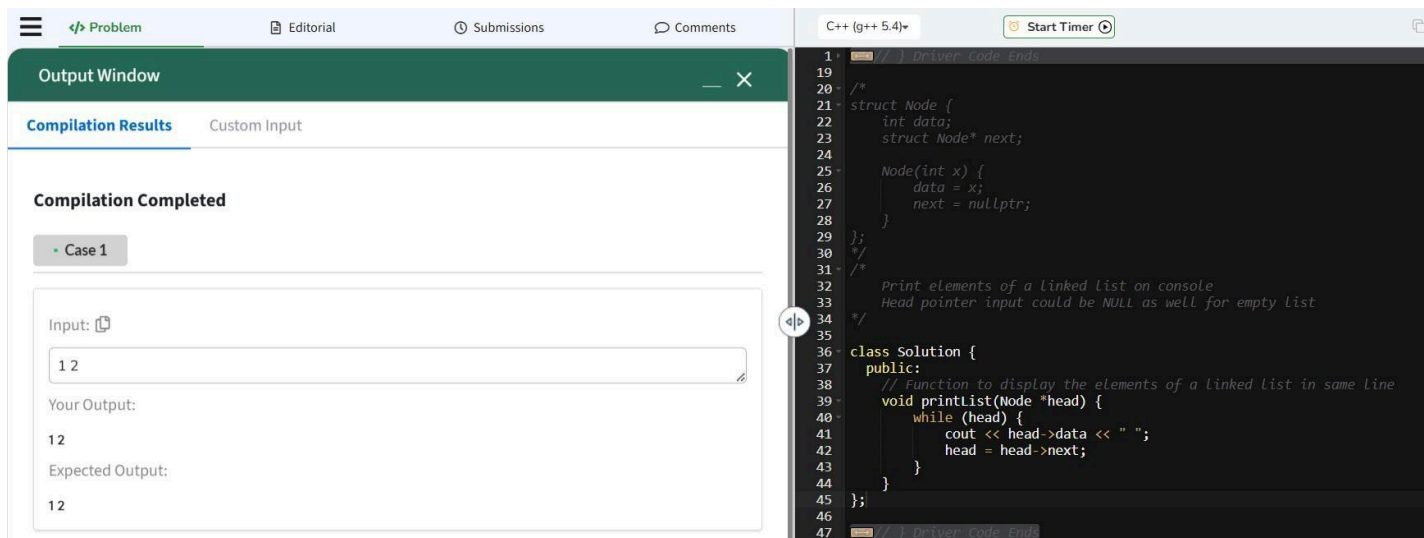
(i): Linked Lists:

Question-1: Print Linked List:

Given a linked list. Print all the elements of the linked list separated by space followed.

Answer:

```
class Solution {
public:
    // Function to display the elements of a linked list in same line
    void printList(Node *head) {
        while (head) {
            cout << head->data << " ";
            head = head->next;
        }
    }
};
```



The screenshot displays a C++ development environment. On the left, the 'Output Window' shows 'Compilation Results' for 'Case 1'. The 'Input' field contains '1 2', and the 'Your Output' field also shows '1 2', matching the 'Expected Output'. On the right, the code editor shows the implementation of the 'printList' function, which iterates through a linked list and prints each node's data followed by a space. The code includes a 'Node' struct and a 'Solution' class with a public 'printList' method.

```
1 // Driver code starts
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         while (head) {
41             cout << head->data << " ";
42             head = head->next;
43         }
44     }
45 };
46
47 // Driver code ends
```



Question-2: Remove Duplicates from Sorted List

Given the head of a sorted linked list, *delete all duplicates such that each element appears only once*. Return *the linked list sorted as well*.

Answer:

```
class Solution {
```

```
public:
```

```
    ListNode* deleteDuplicates(ListNode* head) {
```

```
        if(head==NULL) return head;
```

```
        ListNode* prev = head;
```

```
        ListNode* curr = prev->next;
```

```
        while(curr!=NULL){
```

```
            if(prev->val == curr->val){
```

```
                prev->next = curr->next;
```

```
                delete curr;
```

```
                curr = prev->next;
```

```
            }
```

```
            else{
```

```
                prev = prev->next;
```

```
                curr= curr->next;
```

```
            }
```


```
        }
```

```
        return head;
```

```
    }
```

```
};
```

83. Remove Duplicates from Sorted List

Solved 

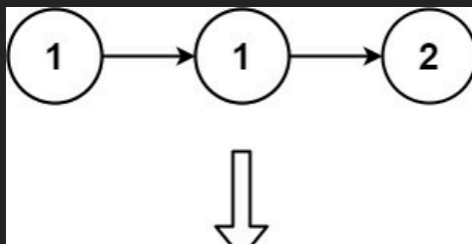
Easy

Topics

Companies

Given the `head` of a sorted linked list, delete all duplicates such that each element appears only once. Return the linked list **sorted** as well.

Example 1:



Accepted

Runtime: 0 ms

Case 1

Case 2

Input


head =
[1,1,2]

Output

[1,2]

Expected

[1,2]

 Contribute a testcase


Question-3: Reverse a linked list: Given the head of a singly linked list, reverse the list, and return *the reversed list*.

Answer:

```

class Solution {
public:
    ListNode*
    reverseList(ListNode*
    head) {
        if(head==NULL)
            return head;
        ListNode* temp =
            head;
        ListNode* prev =
            NULL;
        while(temp!=NULL){
            ListNode* front =
                temp->next;
            temp->next = prev;
            prev = temp;
            temp = front;
        }
        return prev;
    }
};
  
```

206. Reverse Linked List

Solved 

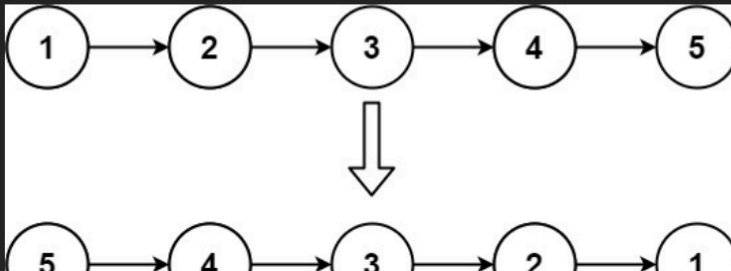
Easy

Topics

Companies

Given the **head** of a singly linked list, reverse the list, and return *the reversed list*.

Example 1:



```

C++
1  ListNode* reverseList(ListNode* head) {
2      if(head == NULL) return head;
3      ListNode* temp = head;
4      ListNode* prev = NULL;
5      while(temp != NULL) {
6          ListNode* front = temp->next;
7          temp->next = prev;
8          prev = temp;
9          temp = front;
10     }
11     return prev;
12 }
  
```

Question-4: Delete middle node of a list:

You are given the head of a linked list. **Delete** the **middle node**, and return *the head of the modified linked list*.


The **middle node** of a linked list of size n is the $\lfloor n / 2 \rfloor^{\text{th}}$ node from the **start** using **0-based indexing**, where $\lfloor x \rfloor$ denotes the largest integer less than or equal to x .

Answer:

```

class Solution {
public:
    ListNode* deleteMiddle(ListNode* head) {
        if(head == NULL || head->next == NULL) return NULL;
        ListNode* slow = head;
        ListNode* fast = head;
        fast = fast->next->next;
        while(fast != NULL && fast->next != NULL) {
            slow = slow->next;
            fast = fast->next->next;
        }
        ListNode* middle = slow->next;
        slow->next = slow->next->next;
        delete middle;
        return head;
    }
};
  
```

2095. Delete the Middle Node of a Linked List

Solved 

Medium

Topics

Companies

Hint

You are given the `head` of a linked list. **Delete the middle node**, and return the `head` of the modified linked list.

The **middle node** of a linked list of size `n` is the $\lfloor n / 2 \rfloor^{\text{th}}$ node from the **start** using **0-based indexing**, where $\lfloor x \rfloor$ denotes the largest integer less than or equal to `x`.

- For `n = 1, 2, 3, 4`, and `5`, the middle nodes are `0, 1, 1, 2`, and `2`, respectively.

```

10  */
11  class Solution {
12  public:
13      ListNode* deleteMiddle(ListNode* head) {
14          if(head == NULL || head->next == NULL) return NULL;
15          ListNode* slow = head;
16          ListNode* fast = head;
17          fast = fast->next->next;
18          while(fast != NULL && fast->next != NULL){
19              slow = slow->next;
20              fast = fast->next->next;
21          }
22          ListNode* middle = slow->next;
23          slow->next = slow->next->next;
24          delete middle;
25          return head;
26      }
27  };

```

Question-5: Merge two sorted linked lists: You are given the heads of two sorted linked lists `list1` and `list2`.

Merge the two lists into one sorted list. The list should be made by splicing together the nodes of the first two lists.

Return *the head of the merged linked list*.

Answer:class Solution { public:

ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {

ListNode dummy(0); // Dummy node to simplify operations

ListNode* tail = &dummy;

while (list1 && list2) {

if (list1->val < list2->val) {

tail->next = list1;

list1 = list1->next;

} else {

tail->next = list2;

list2 = list2->next;

list2->next;

}

tail = tail->next;

}

// Append remaining nodes

tail->next = list1 ? list1 : list2;

return dummy.next;



21. Merge Two Sorted Lists

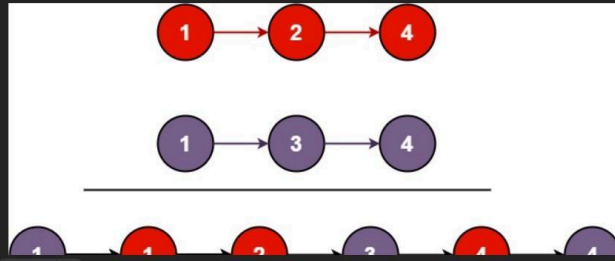
Easy Topics Companies

You are given the heads of two sorted linked lists `list1` and `list2`.

Merge the two lists into one **sorted** list. The list should be made by splicing together the nodes of the first two lists.

Return the head of the merged linked list.

Example 1:



C++

```
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) {}

class Solution {
public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
        ListNode dummy(0); // Dummy node to simplify operations
        ListNode* tail = &dummy;
        while (list1 && list2) {
```

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

list1 =
[1,2,4]

list2 =

Question-6: Detect a cycle in a linked list:

Given head, the head of a linked list, determine if the linked list has a cycle in it.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index of the node that tail's next pointer is connected to. **Note that pos is not passed as a parameter.**

Return true *if there is a cycle in the linked list*. Otherwise, return false.

Answer:

```
class Solution {  
public:  
    bool hasCycle(ListNode *head) {  
        ListNode* slow = head;  
        ListNode* fast = head;  
        while(fast!=NULL && fast->next != NULL){  
            slow = slow->next;  
            fast = fast->next->next;  
            if(slow == fast) return true;  
        }  
        return false;  
    }  
};
```

141. Linked List Cycle

Solved 

Easy Topics Companies

Given head, the head of a linked list, determine if the linked list has a cycle in it.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index of the node that tail's next pointer is connected to. **Note that pos is not passed as a parameter.**

Return true *if there is a cycle in the linked list*. Otherwise, return false.

Example 1:



```
6 *   ListNode(int x) : val(x), next(NULL) {}  
7 *   };  
8 *  
9 class Solution {  
10 public:  
11     bool hasCycle(ListNode *head) {  
12         ListNode* slow = head;  
13         ListNode* fast = head;  
14         while(fast!=NULL && fast->next != NULL){  
15             slow = slow->next;  
16             fast = fast->next->next;  
17             if(slow == fast) return true;  
18         }  
19         return false;  
20     }  
21 };
```



Question-7: Rotate List: Given the head of a linked list, rotate the list to the right by k places.

Answer:

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

        // Step 1: Find length of the linked list
        int len = 1;
        ListNode* tail = head;
        while (tail->next) {
            tail = tail->next;
            len++;
        }

        // Step 2: Make the list circular
        tail->next = head;

        // Step 3: Compute the new tail position
        k = k % len; // Optimize k (rotation beyond length is redundant)
        int stepsToNewHead = len - k; // Find the new head position
        ListNode* newTail = head;

        for (int i = 1; i < stepsToNewHead; i++) {
            newTail = newTail->next;
        }

        // Step 4: Break the cycle and set the new head
        head = newTail->next;
        newTail->next = nullptr;

        return head;
    }
};
```

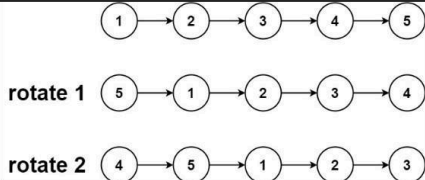

Description | Editorial | Solutions | Submissions

61. Rotate List

Medium Topics Companies

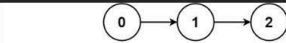
Given the `head` of a linked list, rotate the list to the right by `k` places.

Example 1:



Input: head = [1,2,3,4,5], k = 2
Output: [4,5,1,2,3]

Example 2:



10.2K 104 100 Online

Code

```

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

        // Step 1: Find length of the linked list
        int len = 1;
        ListNode* tail = head;
        while (tail->next) {
            tail = tail->next;
            len++;
        }

        // Step 2: Make the list circular
    }
};

```

Saved Ln 27, Col 17

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

head = [1,2,3,4,5]

k =

Question-8: Sort List: Given the head of a linked list, return *the list after sorting it in ascending order*.

Answer:

```

class Solution {
public:
    // Function to find the middle node of the linked list
    ListNode* getMid(ListNode* head) {
        ListNode* slow = head;
        ListNode* fast = head->next;

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

        return slow;
    }

    // Function to merge two sorted lists
    ListNode* merge(ListNode* l1, ListNode* l2) {
        ListNode dummy(0);
        ListNode* tail = &dummy;
    }
};

```



```
while (l1 && l2) {
    if (l1->val < l2->val) {
        tail->next = l1;
        l1 = l1->next;
    } else {
        tail->next = l2;
        l2 = l2->next;
    }
    tail = tail->next;
}

if (l1) tail->next = l1;
if (l2) tail->next = l2;

return dummy.next;
}

// Function to sort the linked list using Merge Sort
ListNode* sortList(ListNode* head) {
    if (!head || !head->next) return head;

    ListNode* mid = getMid(head);
    ListNode* rightHead = mid->next;
    mid->next = nullptr;

    ListNode* left = sortList(head);
    ListNode* right = sortList(rightHead);

    return merge(left, right);
}
};
```

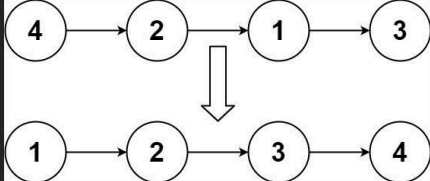
Description | Editorial | Solutions | Submissions

148. Sort List

Medium Topics Companies


Given the `head` of a linked list, return the list after sorting it in **ascending order**.

Example 1:



Input: head = [4,2,1,3]
Output: [1,2,3,4]

Example 2:



Code

```

11 class Solution {
12 public:
13     // Function to find the middle node of the linked list
14     ListNode* getMid(ListNode* head) {
15         ListNode* slow = head;
16         ListNode* fast = head->next;
17
18         while (fast && fast->next) {
19             slow = slow->next;
20             fast = fast->next->next;
21         }
22
23         return slow;
24     }

```

Saved Ln 12, Col 8

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

head = [4,2,1,3]

Question-9: Merge k sorted lists:

You are given an array of k linked-lists lists, each linked-list is sorted in ascending order.

Merge all the linked-lists into one sorted linked-list and return it.

Answer:

```

class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
        if (!l1) return l2;
        if (!l2) return l1;

        if (l1->val < l2->val) {
            l1->next = mergeTwoLists(l1->next, l2);
            return l1;
        } else {
            l2->next = mergeTwoLists(l1, l2->next);
            return l2;
        }
    }

    ListNode* mergeKLists(vector<ListNode*>& lists) {
        if (lists.empty()) return nullptr;
        int n = lists.size();

```



```
while (n > 1) {int
```

```
j=0;
```

```
for (int i = 0; i < n / 2; i++) {
```

```
    lists[i] = mergeTwoLists(lists[i], lists[n - i - 1]);
```

```
}
```

```
n = (n + 1) / 2; // Reduce the number of lists
```

```
}
```

```
return lists[0];
```

```
}
```

```
};
```

The screenshot displays a coding platform interface with two main panels. The left panel, titled '23. Merge k Sorted Lists', contains the problem description: 'You are given an array of k linked-lists lists, each linked-list is sorted in ascending order. Merge all the linked-lists into one sorted linked-list and return it.' It includes two examples: Example 1 with input lists = [[1,4,5],[1,3,4],[2,6]] and output [1,1,2,3,4,4,5,6], and Example 2 with input lists = [] and output []. The right panel shows a C++ code editor with a solution. The code defines a class Solution with a public method mergeTwoLists that takes two ListNode* pointers and returns a merged sorted list. The code uses a recursive approach to merge the lists. Below the code editor, the 'Testcase' and 'Test Result' sections are visible, showing 'Accepted' status with a runtime of 0 ms. The input field contains the same list of lists as in Example 1, and the output field is empty.

23. Merge k Sorted Lists

Hard Topics Companies

You are given an array of k linked-lists `lists`, each linked-list is sorted in ascending order.

Merge all the linked-lists into one sorted linked-list and return it.

Example 1:

Input: `lists = [[1,4,5],[1,3,4],[2,6]]`
Output: `[1,1,2,3,4,4,5,6]`
Explanation: The linked-lists are:
[
1->4->5,
1->3->4,
2->6
]
merging them into one sorted list:
1->1->2->3->4->4->5->6

Example 2:

Input: `lists = []`
Output: `[]`

```
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
        if (!l1) return l2;
        if (!l2) return l1;

        if (l1->val < l2->val) {
            l1->next = mergeTwoLists(l1->next, l2);
            return l1;
        } else {
            l2->next = mergeTwoLists(l1, l2->next);
            return l2;
        }
    }
};
```

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

lists =

[1,4,5], [1,3,4], [2,6]

Output