Harsh Pandey

22BCS17030

IOT-609/B

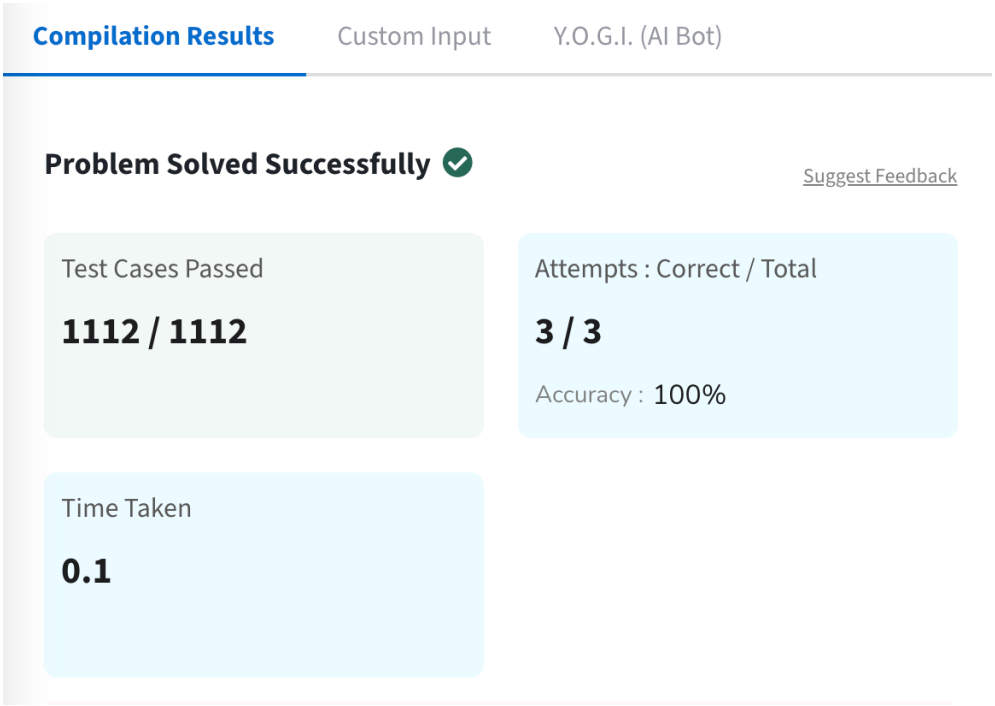
Advance Programming

Assignment-3

1. Print Linked List

CODE:

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

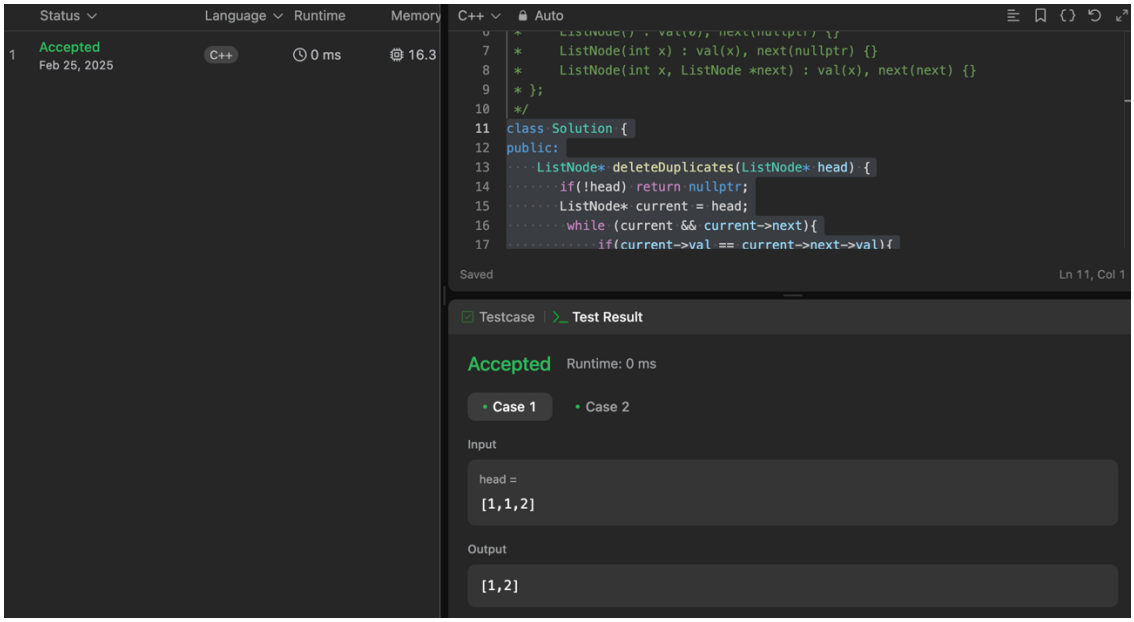


2. Remove duplicates from a sorted list:

CODE:

```
class Solution {
public:
  ListNode* deleteDuplicates(ListNode* head) {
    if(!head) return nullptr;
```

```
        ListNode* current = head;
        while (current && current->next){
            if(current->val == current->next->val){
                ListNode* temp = current->next;
                current->next= current->next->next;
                delete temp;
            }
            else{
                current = current->next;
            }
        }
        return head;
    }
};
```

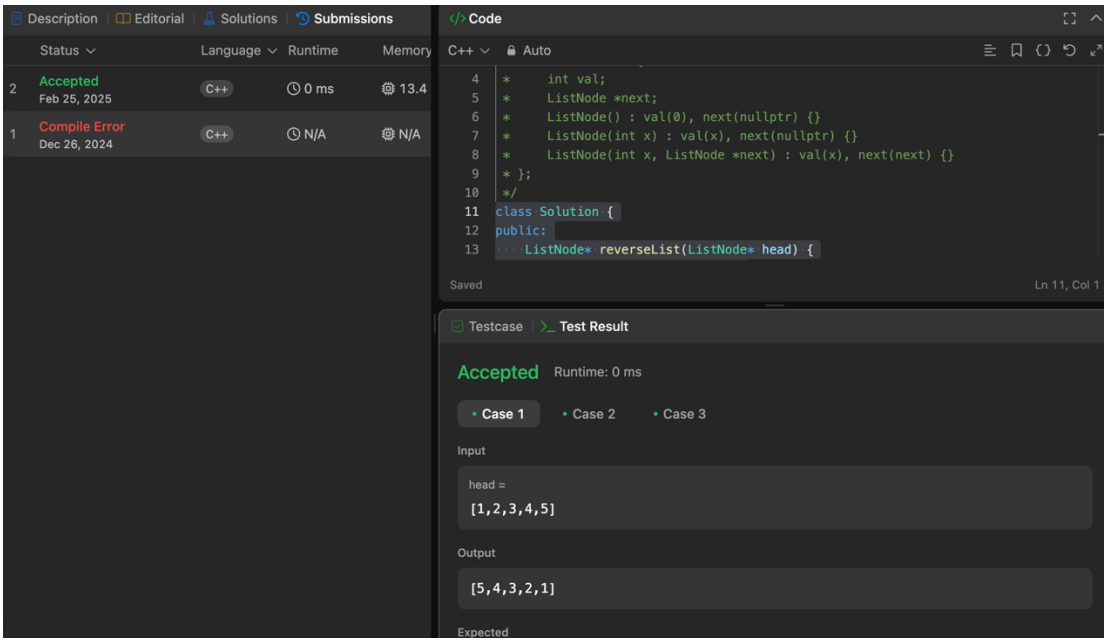


3. Reverse a linked list:

CODE:

```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        if(!head) return nullptr;
        ListNode* current = head;
        ListNode* prev = nullptr;
        while(current){
            ListNode* nextNode= current->next;
            current->next=prev;
            prev= current;
```

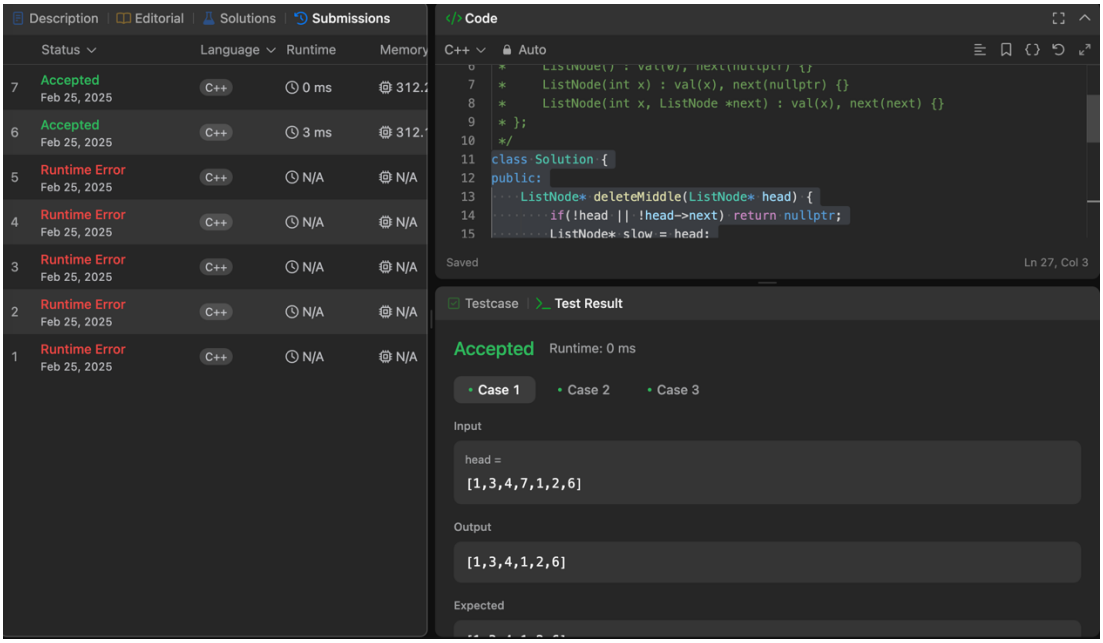
```
        current=nextNode;
      }
      return prev;
    }
};
```



4. **Delete middle node of a list:**

CODE:

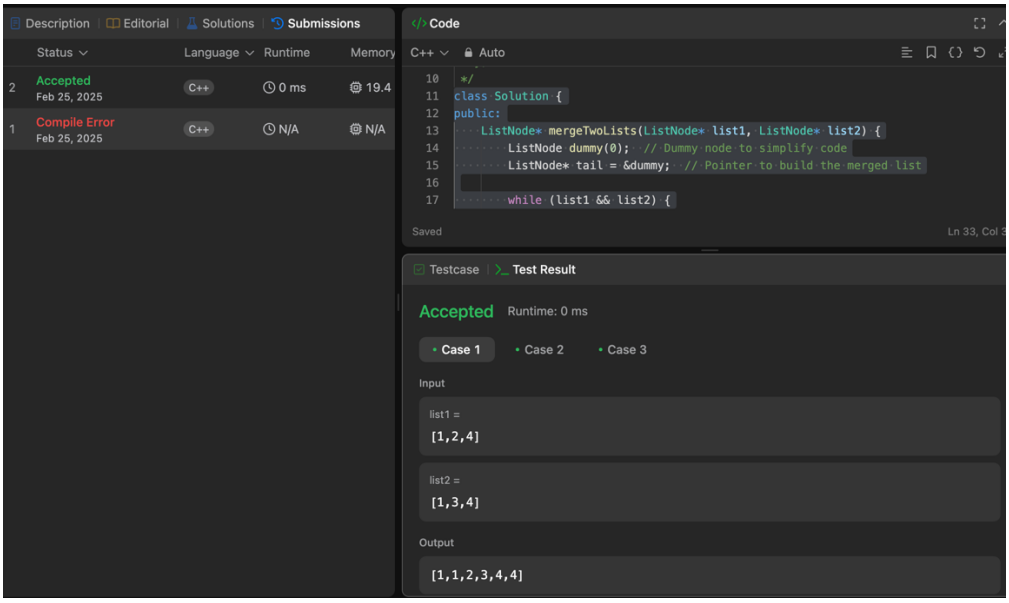
```
class Solution {
public:
  ListNode* deleteMiddle(ListNode* head) {
    if(!head || !head->next) return nullptr;
    ListNode* slow = head;
    ListNode* fast = head;
    ListNode* prev = nullptr;
    while(fast && fast->next){
      prev=slow;
      slow=slow->next;
      fast=fast->next->next;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



5. Merge two sorted linked lists:

CODE:

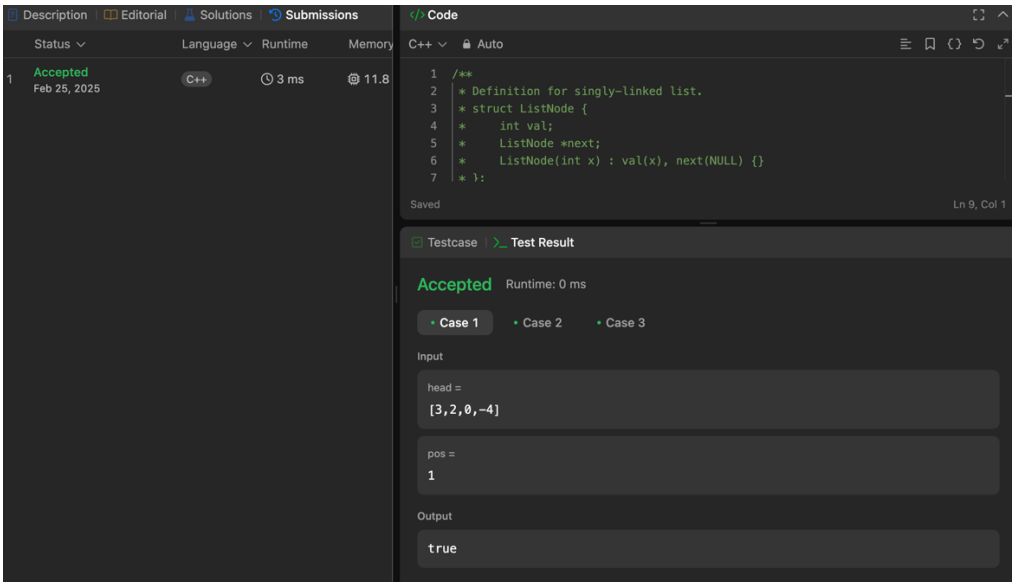
```
class Solution {
public:
  ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
    ListNode dummy(0);  // Dummy node to simplify code
    ListNode* tail = &dummy;  // Pointer to build the merged list

    while (list1 && list2) {
      if (list1->val <= list2->val) {
        tail->next = list1;
        list1 = list1->next;
      } else {
        tail->next = list2;
        list2 = list2->next;
      }
      tail = tail->next;  // Move the tail pointer
    }

    // Attach the remaining elements of the non-empty list
    tail->next = list1 ? list1 : list2;

    return dummy.next;  // Return the merged list (skip dummy node)
  }
};
```



6. Detect a cycle in a linked list:

CODE:

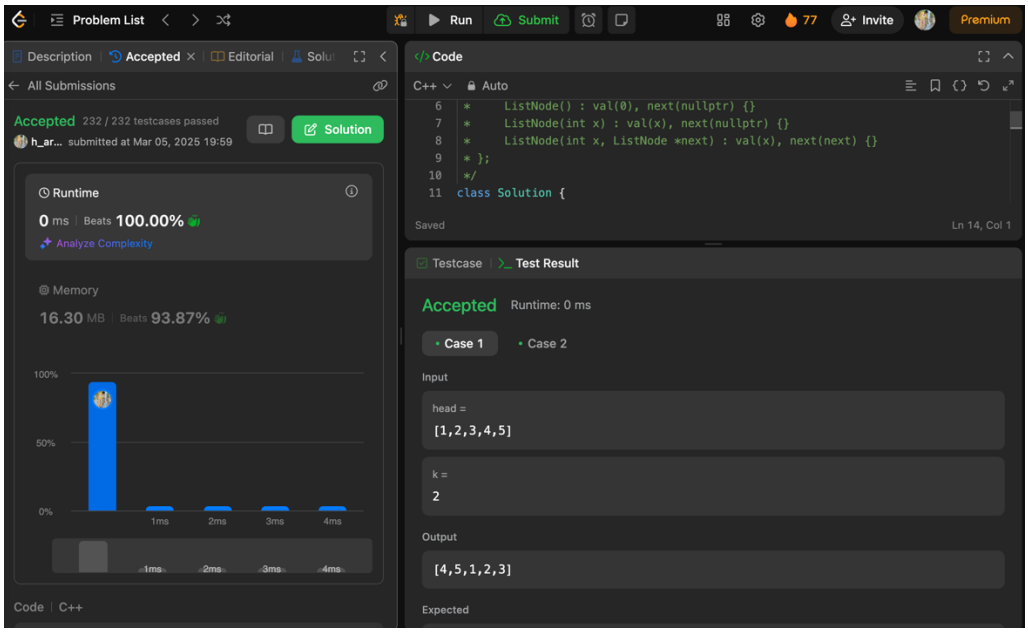
```
class Solution {
public:
  bool hasCycle(ListNode *head) {
    if(!head) return false;
    ListNode* slow = head;
    ListNode* fast = head;
    while(fast && fast->next){
      slow=slow->next;
      fast=fast->next->next;
      if (slow == fast) return true;      }
   return false;       }    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



7. Rotate a list:

CODE:

```
class Solution {
public:
  ListNode* rotateRight(ListNode* head, int k) {

    if (!head || !head->next || k == 0) return head;
    ListNode* temp = head;
    int length = 1;
    while (temp->next) {
      temp = temp->next;
      length++;
    }
    temp->next = head;
    k = k % length;
    int stepsToNewHead = length - k;
    ListNode* newTail = head;
    for (int i = 1; i < stepsToNewHead; i++) {
      newTail = newTail->next;
    }
    head = newTail->next;
    newTail->next = nullptr;
    return head;
  }
};
```

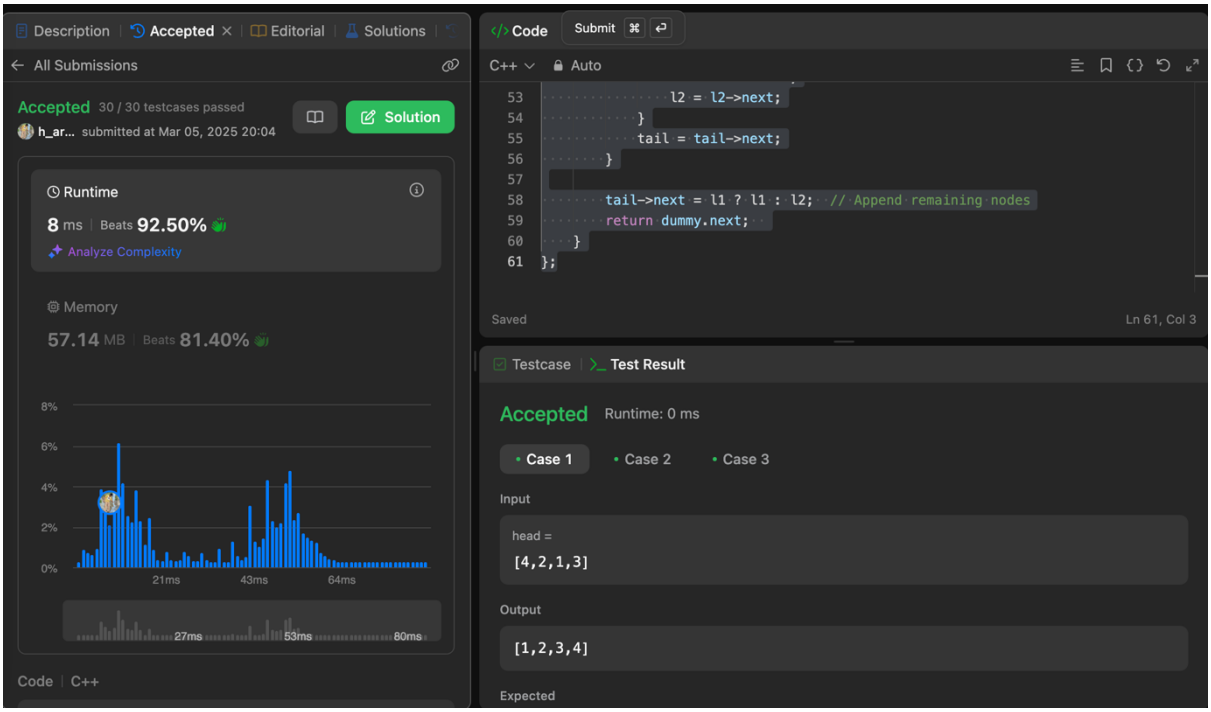


8. **Sort List:**

CODE:

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

    // Step 1: Split the list into two halves
    ListNode* mid = getMid(head);
    ListNode* left = head;
    ListNode* right = mid->next;
    mid->next = nullptr;  // Break the list into two halves

    // Step 2: Recursively sort each half
    left = sortList(left);
    right = sortList(right);

    // Step 3: Merge sorted halves
    return merge(left, right);
  }

private:
  // Function to find the middle node using the slow-fast pointer approach
  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 linked 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;
    }

    tail->next = l1 ? l1 : l2;  // Append remaining nodes
    return dummy.next;
  }
};
```



9. Merge k sorted lists:

CODE:

```
class Solution {
public:
    struct Compare {
        bool operator()(ListNode* a, ListNode* b) {
            return a->val > b->val;
        }
    };
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        priority_queue<ListNode*, vector<ListNode*>, Compare> minHeap;
```

```
        for (auto list : lists) {
            if (list) minHeap.push(list);
        }
        ListNode dummy;
        ListNode* tail = &dummy;
        while (!minHeap.empty()) {
            ListNode* smallest = minHeap.top();
            minHeap.pop();
            tail->next = smallest;
            tail = tail->next;
            if (smallest->next) {
                minHeap.push(smallest->next);
            }
        }
        return dummy.next;
    }
};
```