Experiment 3

Student Name: Sabir Ali **UID: 22BET10033**
Branch: BE-IT **Section/Group: 22BET-IOT-701/A**
Semester: 6 **Date of Performance: 05/02/2025**
Subject Name: Advance Programming 2 **Subject Code: 22ITH-359**

1.Aim: Implement the concept of link list for solving the following problems:-Print linked list, Remove duplicates from a sorted list, Reverse a linked list, Delete middle node of a list, Merge two sorted linked lists, Remove duplicates from sorted lists 2, Detect a cycle in a linked list, Reverse linked list 2, rotate a list, Merge k sorted lists, Sort List.

2.Objective: To understand the concept of link list with solving the problem for better understanding of operations and logics of link list.

3.Code:

(A)Print Link List

```
class Solution {

public:

void printList(Node*head) {

Node* temp = head;

while (temp != NULL) {

cout << temp->data << " ";

temp = temp->next;

}

cout << endl;

}

};
```

```
Node* insert(Node* head, int data) {

Node* newNode = new
Node(data);

if (!head) return newNode;

Node* temp = head;

while (temp->next) temp = temp->next;

temp->next = newNode;

return head;

}
```

(B) Remove Duplicates from Sorted List

```
class Solution {

public:

ListNode* deleteDuplicates(ListNode*head) {

ListNode* curr=head;

while(curr!=nullptr && curr->next!=nullptr)

if(curr->val==curr->next->val){

ListNode* temp=curr->next;

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

delete temp;
```

```
}

else{

curr=curr->next;

}

return head;

}

};
```

(C) Reverse Linked List

```
class Solution {

public:

ListNode*
reverseList(ListNode*head) {

ListNode* prev=nullptr;

ListNode* curr=head;

while(curr!=nullptr){

ListNode*nextnode=curr->next;

curr->next=prev;

prev=curr;

curr=nextnode;

}
```

```
return prev;

}

};
```

(D) Delete the Middle Node of a Linked List

```
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;

}

if (prev) prev->next = slow->next;

delete slow;

return head;

}
```

```
};
```

(E) Merge Two Sorted Lists

```
class Solution {

public:

ListNode*

mergeTwoLists(ListNode* list1, ListNode* list2) {

ListNode* dummy = new ListNode(-1);

ListNode* current =dummy;

while (list1 && list2) {

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

current->next = list1;

list1 = list1->next;

} else {

current->next = list2;

list2 = list2->next;

}

current = current->next;

}

if (list1) current->next = list1;

if (list2) current->next = list2;
```

```
ListNode* mergedHead = dummy->next;
delete dummy;
return mergedHead;
}
};
```

(F) Remove Duplicates from Sorted List II

```
class Solution {
public:
ListNode* deleteDuplicates(ListNode* head) {
ListNode* dummy = new ListNode(0, 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;

}

ListNode* newHead = dummy->next;

delete dummy;

return newHead;

}

};
```

(G) Linked List Cycle

```
class Solution {

public:

bool hasCycle(ListNode *head) {

ListNode *slow = head, *fast = head;

while (fast && fast->next) {

slow = slow->next;

fast = fast->next->next;

if (slow == fast) return true;

}
```

```
return false;
}
};
```

(H) Reverse Linked List II

```
class Solution {
public:
ListNode* reverseBetween(ListNode* head, int left, int right) {
if (!head || left == right) return head;
ListNode* dummy = new ListNode(0, head);
ListNode* prev = dummy;
for (int i = 1; i < left; i++) {
prev = prev->next;
}
ListNode* current = prev->next;
ListNode* nextNode = nullptr;
for (int i = 0; i < right - left; i++) {
nextNode = current->next;
current->next = nextNode->next;
nextNode->next = prev->next;
```

```
prev->next = nextNode;
}
return dummy->next;
}
};
```

(I)Rotate List

```
class Solution {
public:
ListNode* rotateRight(ListNode* head, int k) {
if (!head || !head->next || k == 0) return head;
int length = 1;
ListNode* tail = head;
while (tail->next) {
tail = tail->next;
length++;
}
tail->next = head;
k = k % length;
int stepsToNewHead = length - k;
ListNode* newTail = head;
```

```
for (int i = 1; i < stepsToNewHead; i++) {

newTail = newTail->next;

}

ListNode* newHead = newTail->next;

newTail->next = nullptr;

return newHead;

}

};
```

(J) Merge k Sorted Lists

```
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 (ListNode* list : lists) {
if (list) minHeap.push(list);
}
ListNode* dummy = new ListNode(0);
ListNode* tail = dummy;
while (!minHeap.empty()) {
ListNode* top = minHeap.top();
minHeap.pop();
tail->next = top;
tail = tail->next;
if (top->next) minHeap.push(top->next);
}
return dummy->next;
}
};
```

(K) Sort List

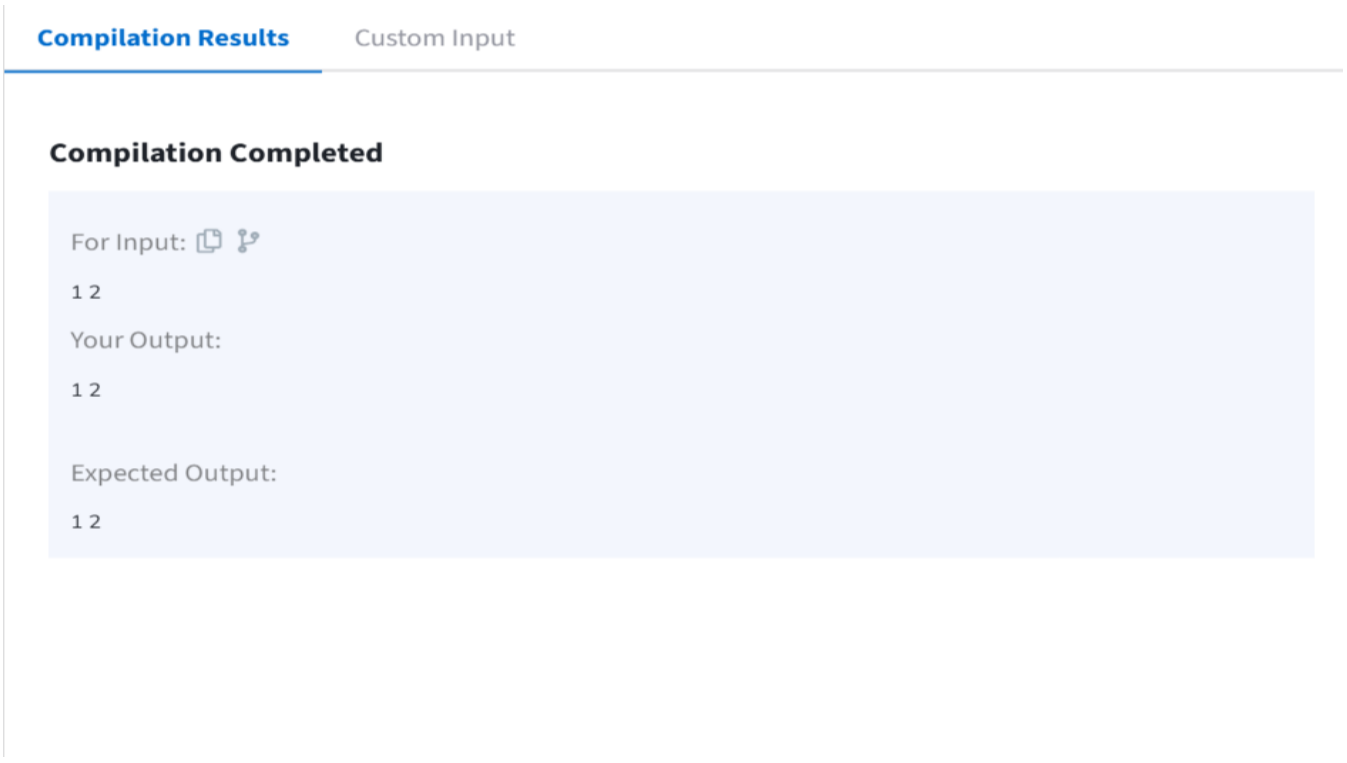
```
class Solution {
public:
ListNode* sortList(ListNode* head) {
```

```
if (!head || !head->next) return head;

ListNode* slow = head, *fast = head, *prev = nullptr;

while (fast && fast->next) {

prev = slow;

slow = slow->next;

fast = fast->next->next;

}

prev->next = nullptr;

ListNode* left = sortList(head);

ListNode* right = sortList(slow);

return merge(left, right);

}

private:

ListNode* merge(ListNode* l1, ListNode* l2) {

ListNode* dummy = new ListNode(0);

ListNode* current = dummy;

while (l1 && l2) {

if (l1->val < l2->val) {

current->next = l1;

l1 = l1->next;
```

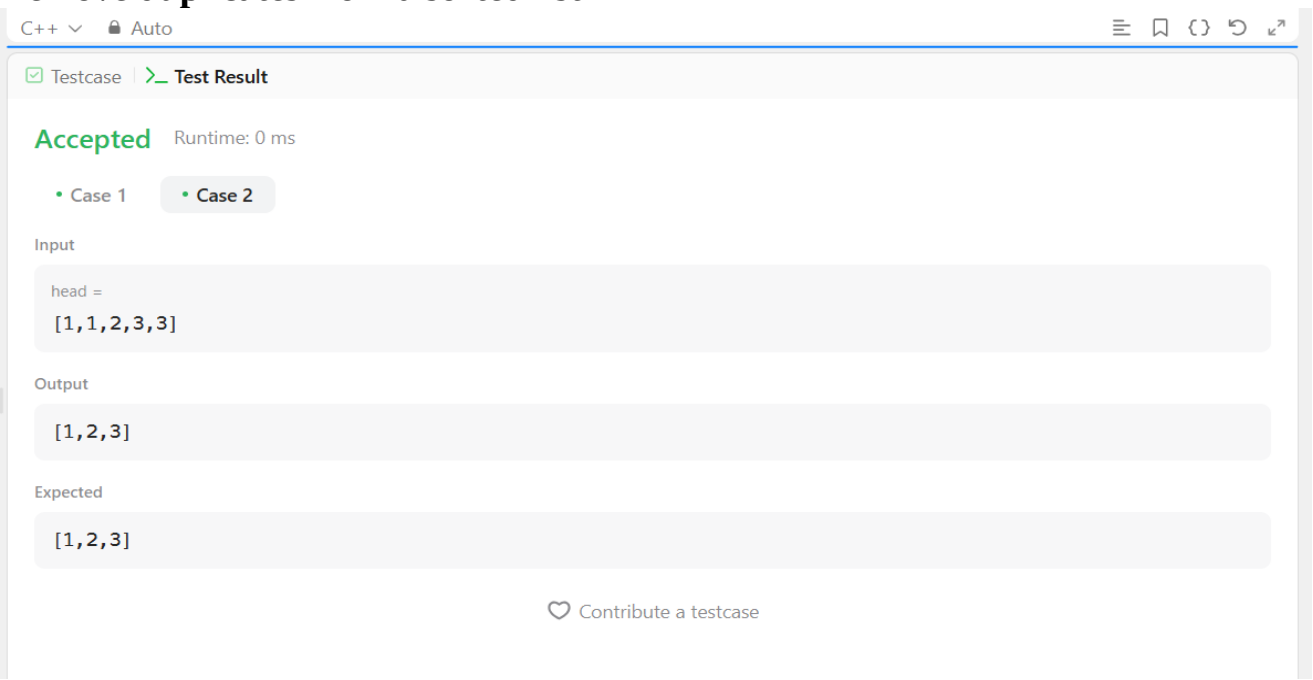
```
} else {

current->next = l2;

l2 = l2->next;

}

current = current->next;

}

current->next = l1 ? l1 : l2;

return dummy->next;

}

};
```

4.Output:

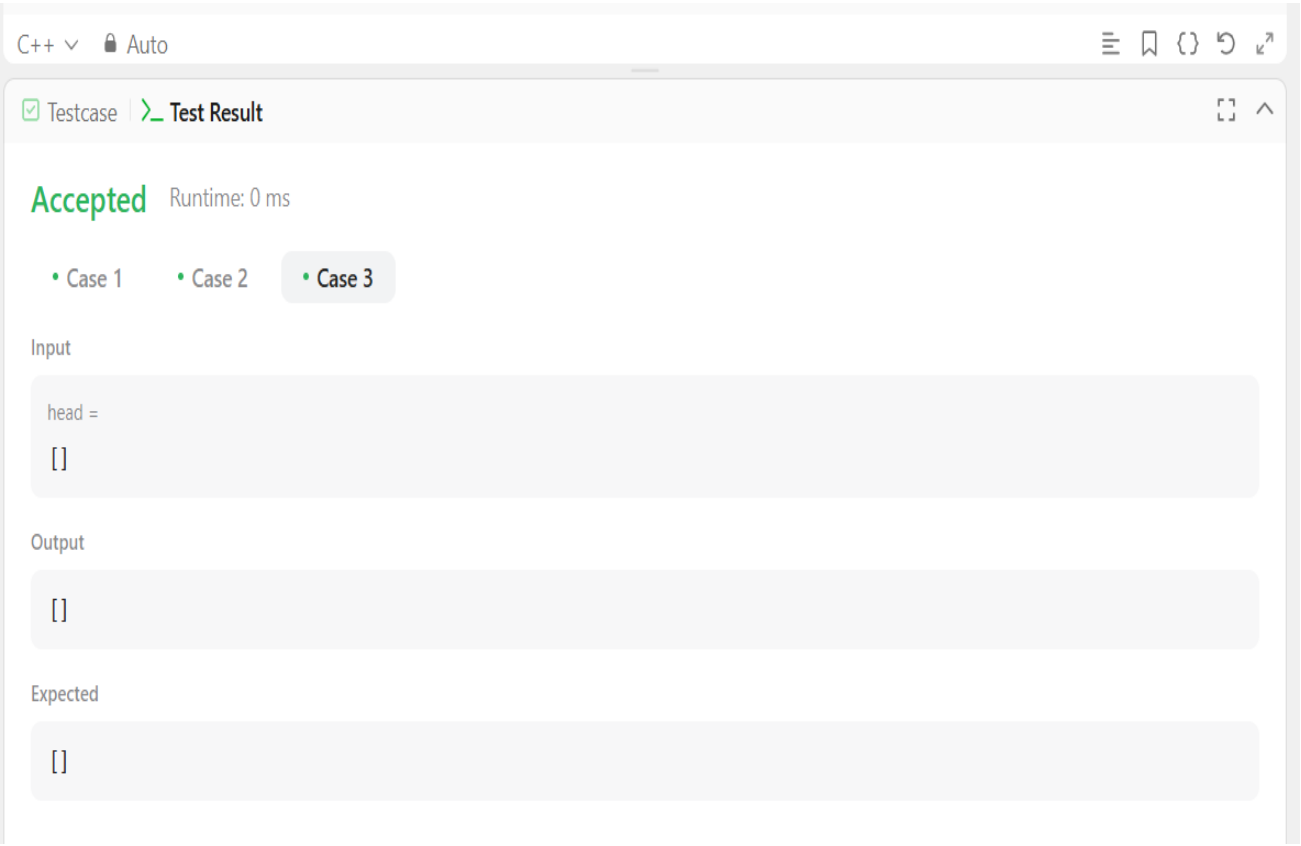
Print linked list



Remove duplicates from a sorted list

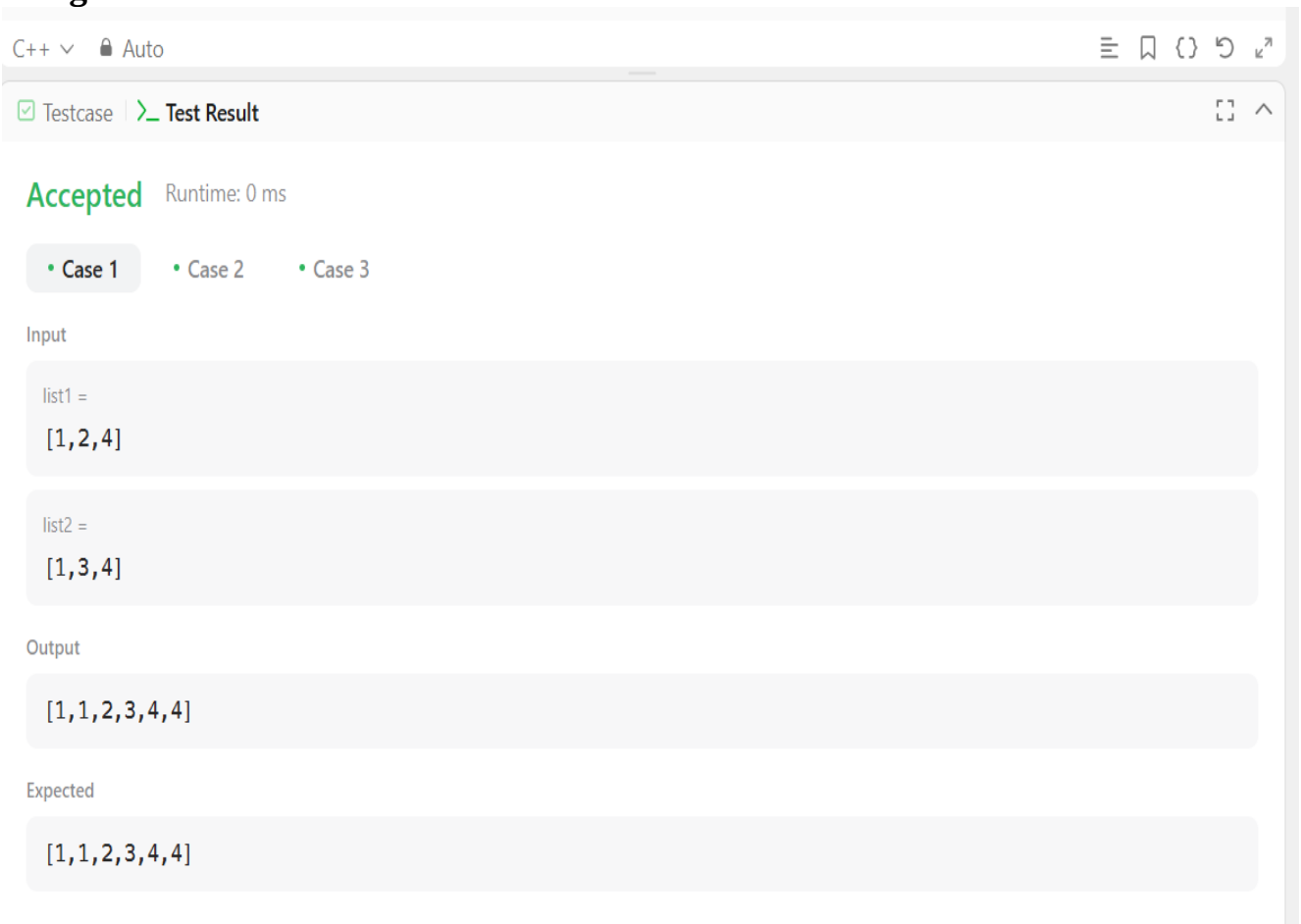


Reverse a linked list

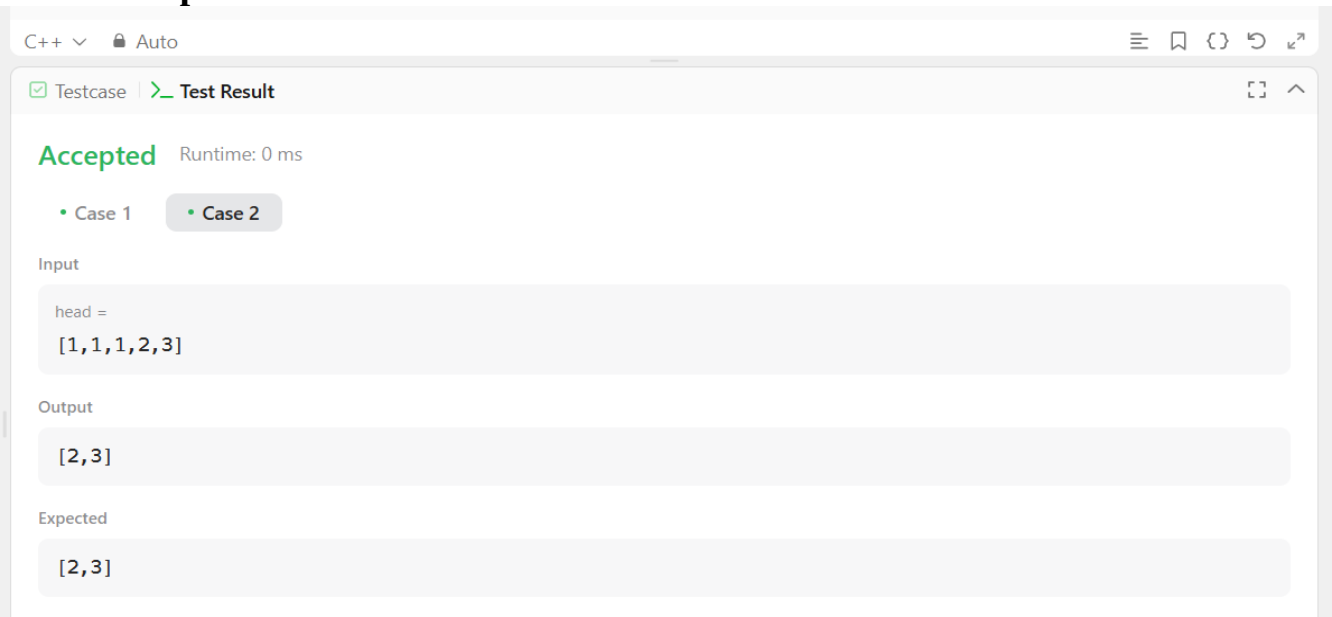


Delete middle node of a list

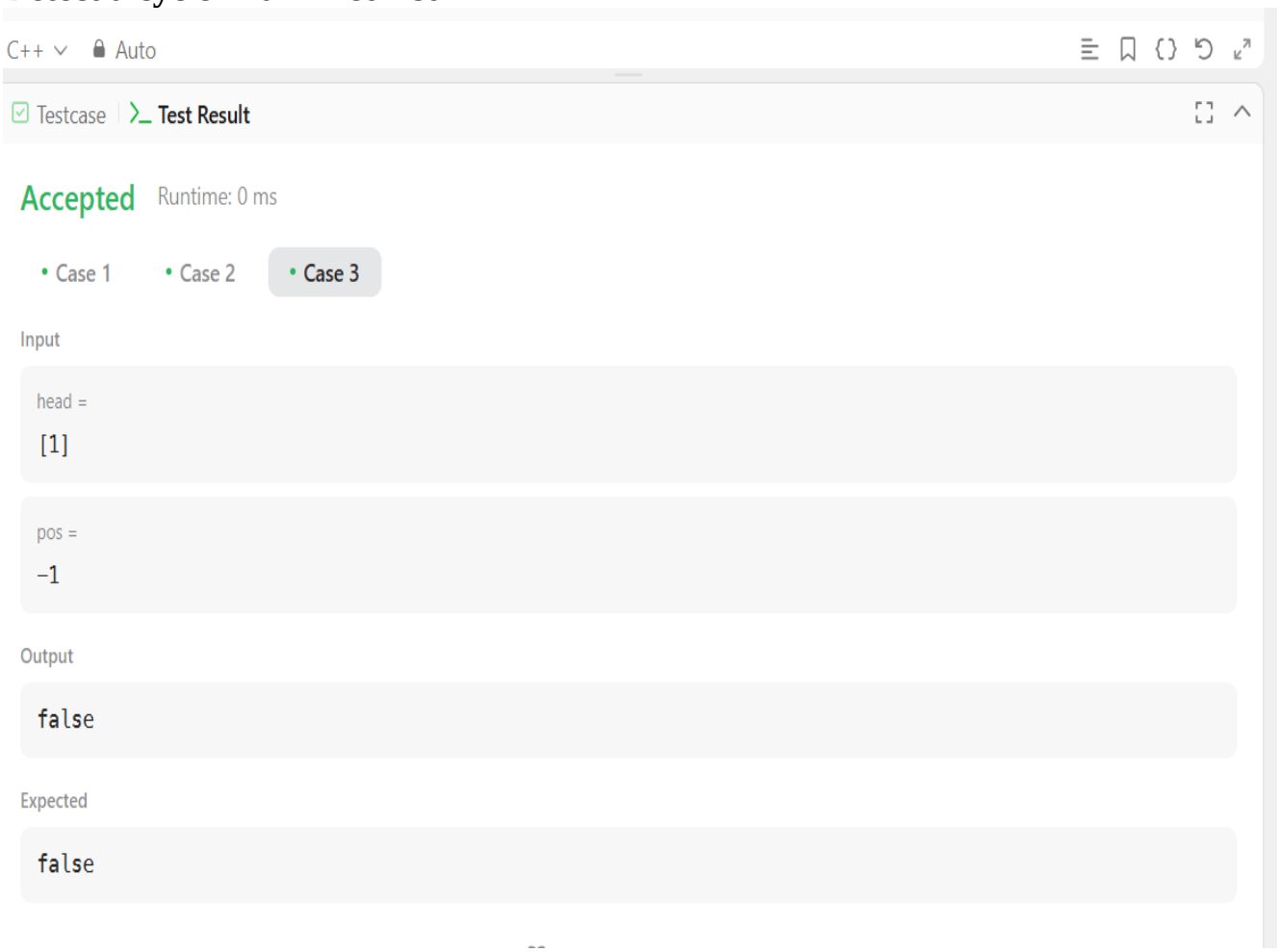
Merge two sorted linked lists



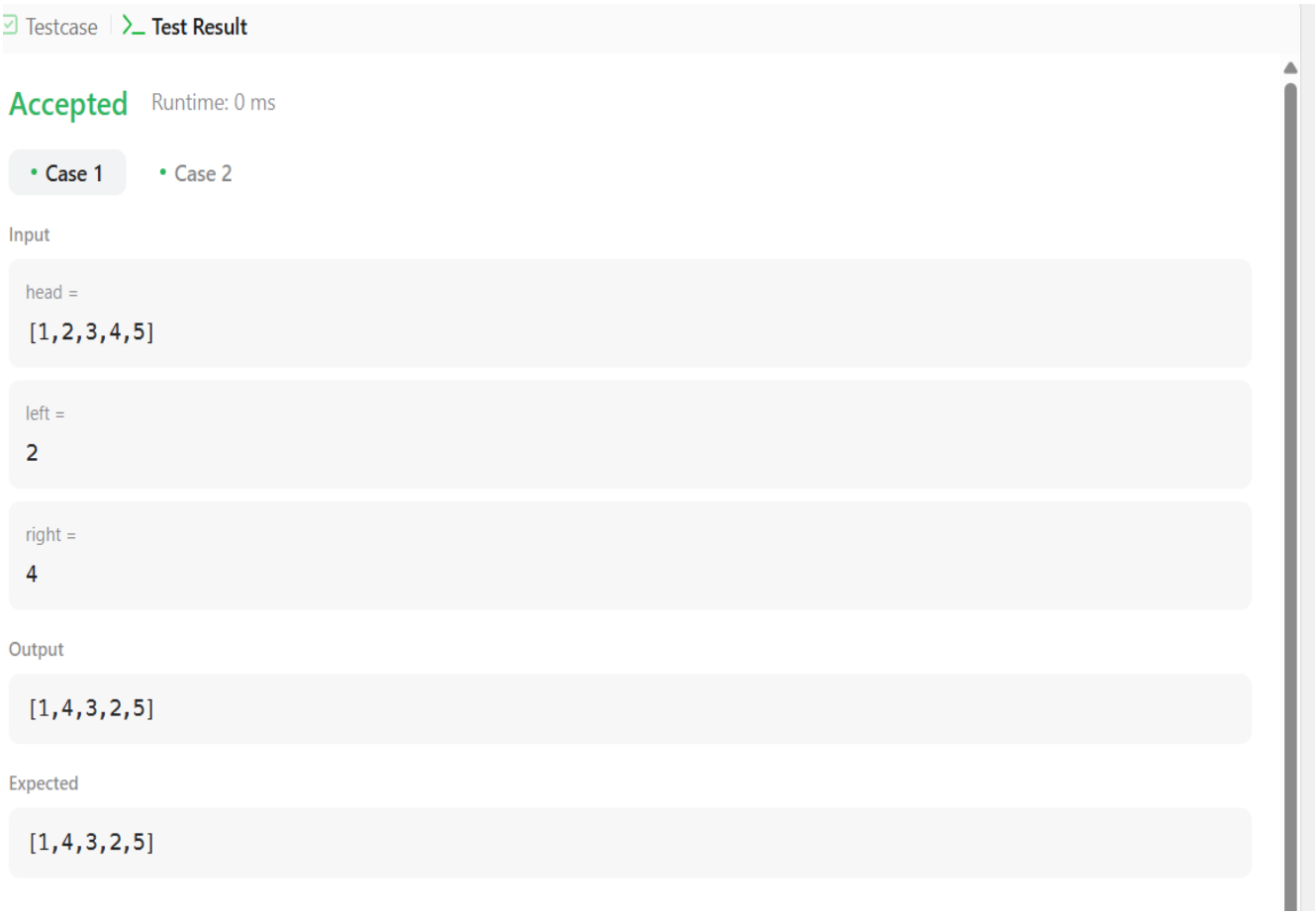
Remove duplicates from sorted lists 2



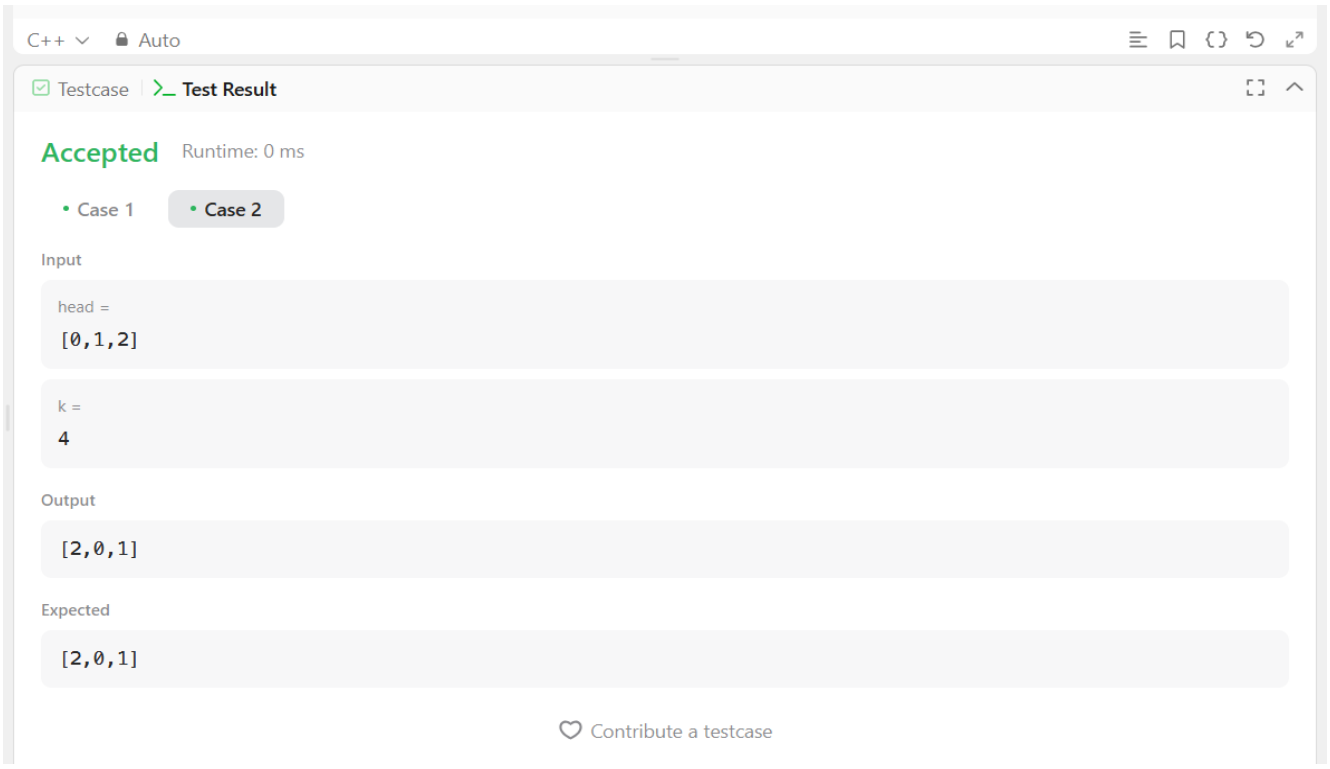
Detect a cycle in a linked list



Reverse linked list 2

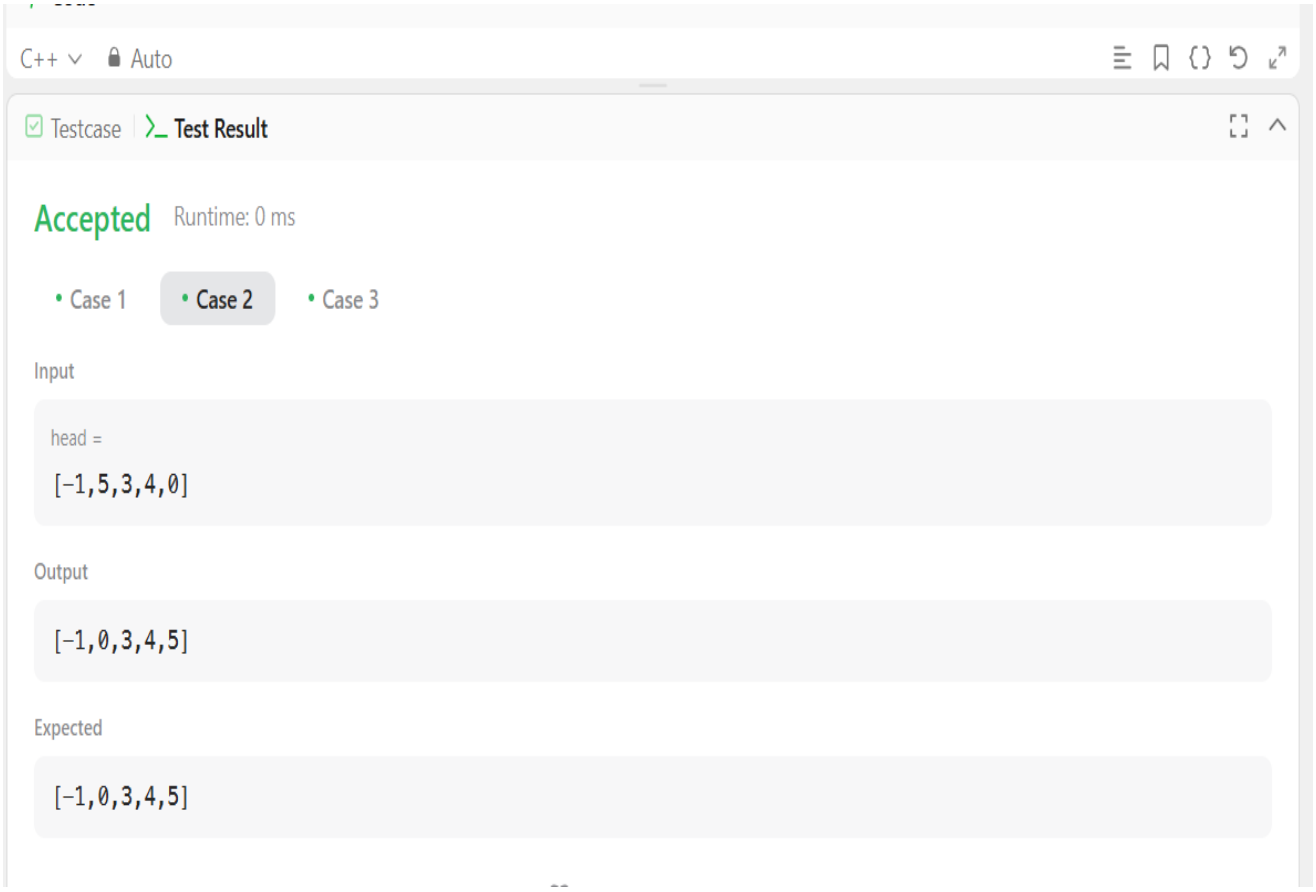


rotate a list



Merge k sorted lists

Sort List



5.Learning Outcome:

- Learn the concept of Link List and its operation.
- Learn the concept of implement cycle of link list.
- Understand the concepts with solving problems.
- Understand the concept of merge list and delete middle element of an list etc.
- Learn the concept of class and loops.