1. **Print Linked list**

**Program Code:**

class Solution {

public:

void printList(Node \*head) {

Node\* current = head;

while (current != nullptr) {

cout << current->data << " ";

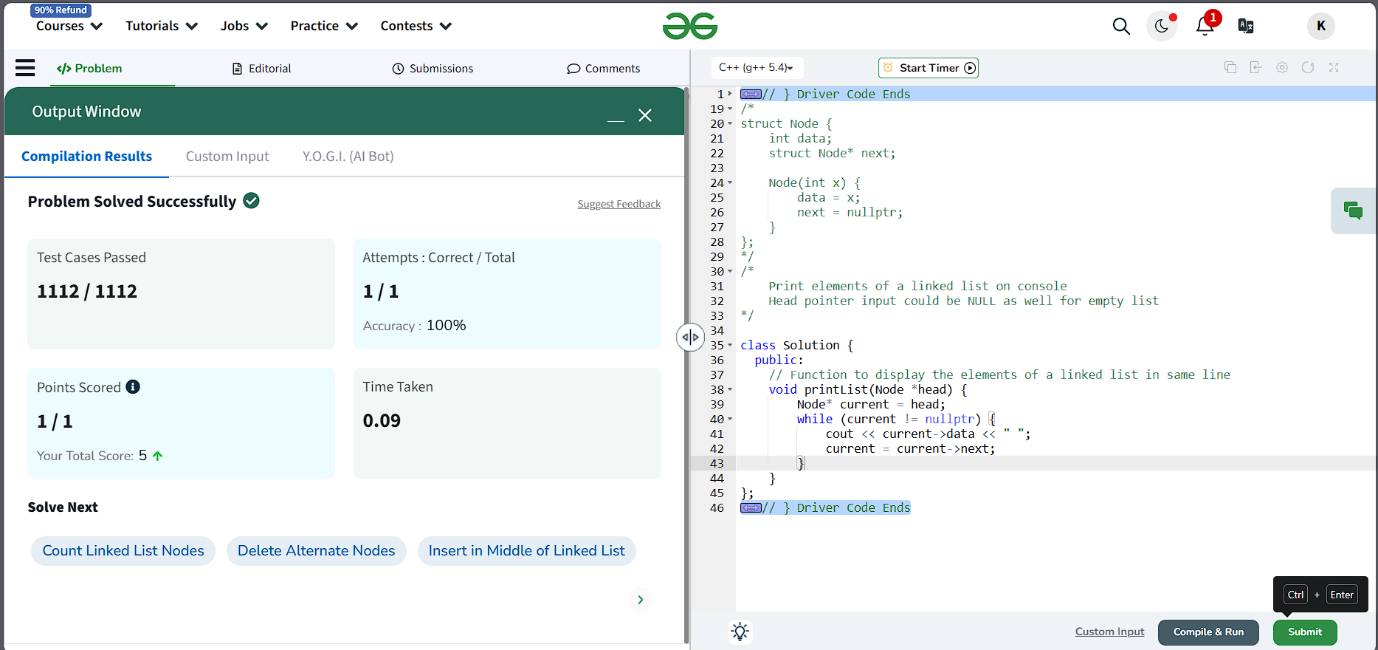
current = current->next;

}

}

};

**Output:**



1. **Remove duplicates from a Sorted List**

**Program Code:**

class Solution {

public:

    ListNode\* deleteDuplicates(ListNode\* head) {

        ListNode\* current = head;

        while(current && current->next)

        {

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

            current->next = current->next->next;

        } else {

            current = current->next;

        }

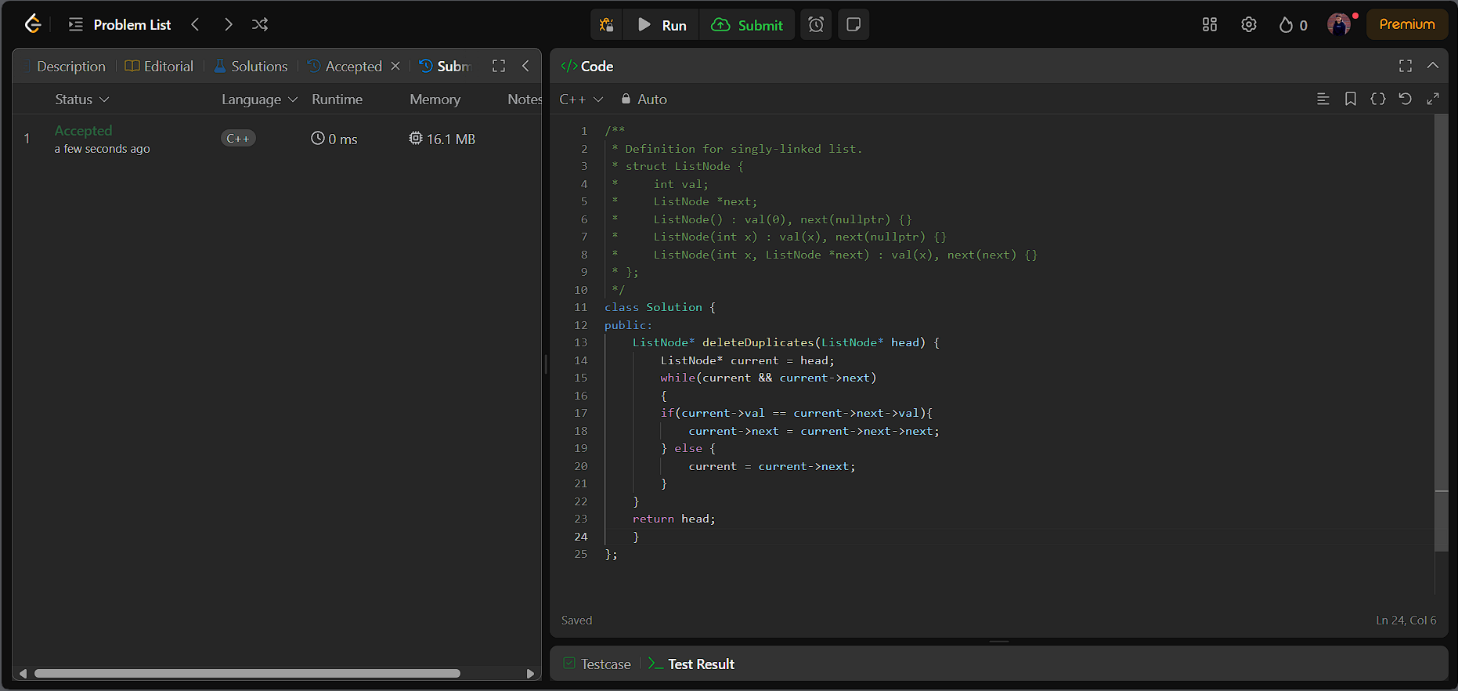
    }

    return head;

    }

};

**Output:**



1. **Reverse a Linked List**

**Program Code:**

class Solution {

public:

    ListNode\* reverseList(ListNode\* head) {

        ListNode\* current = head;

        ListNode\* prev = nullptr;

        ListNode\* nextNode;

        while(current != nullptr){

            nextNode = current->next;

            current->next = prev;

            prev = current;

            current = nextNode;

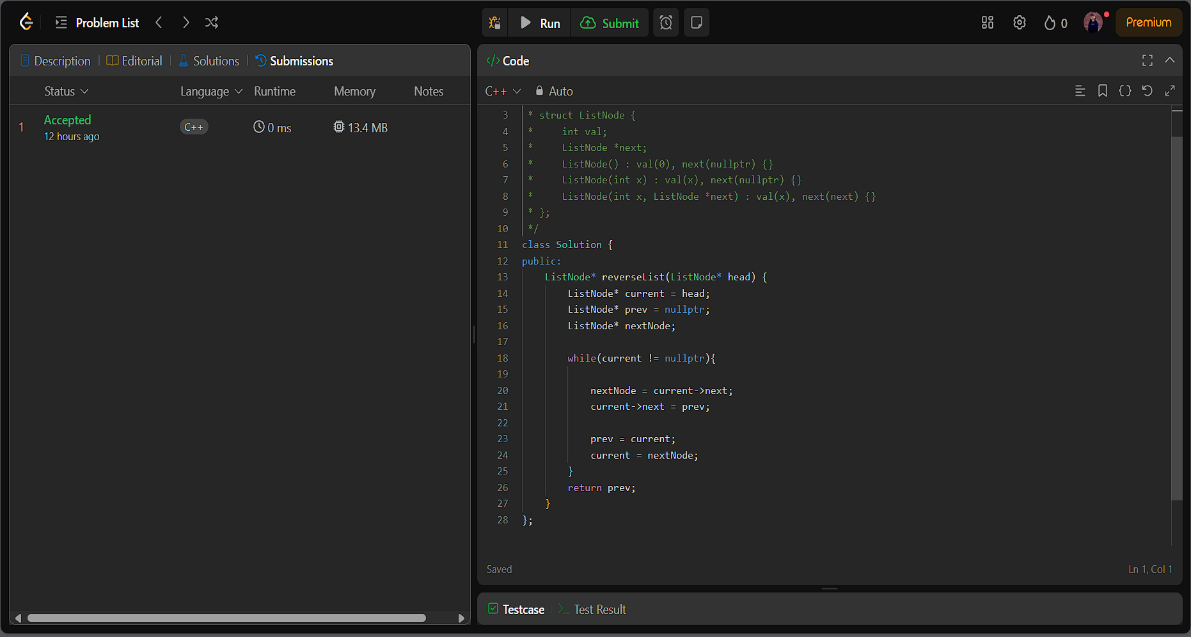
        }

        return prev;

    }

};

**Output:**



1. **Delete Middle Node of the List**

**Program Code:**

class Solution {

public:

    ListNode\* deleteMiddle(ListNode\* head) {

        if(head==NULL || head->next==NULL)

        return NULL;

        ListNode\* slow = head;

        ListNode\* fast= head;

        ListNode\* prev = head;

        while(fast != NULL && fast->next!=NULL){

            prev = slow;

            slow = slow->next;

            fast = fast->next->next; }

        prev->next = slow->next;

        delete slow;

        return head;

    }

};

**Output:** A screenshot of a computer

AI-generated content may be incorrect.

1. **Merge two Sorted Linked List**

**Program Code:**

class Solution {

public:

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

        if(list1 == NULL){

            return list2;

        }

        if(list2 == NULL){

            return list1;

        }

        if(list1->val <= list2->val){

            list1->next = mergeTwoLists(list1->next,list2);

            return list1;

        }else{

            list2->next = mergeTwoLists(list1,list2->next);

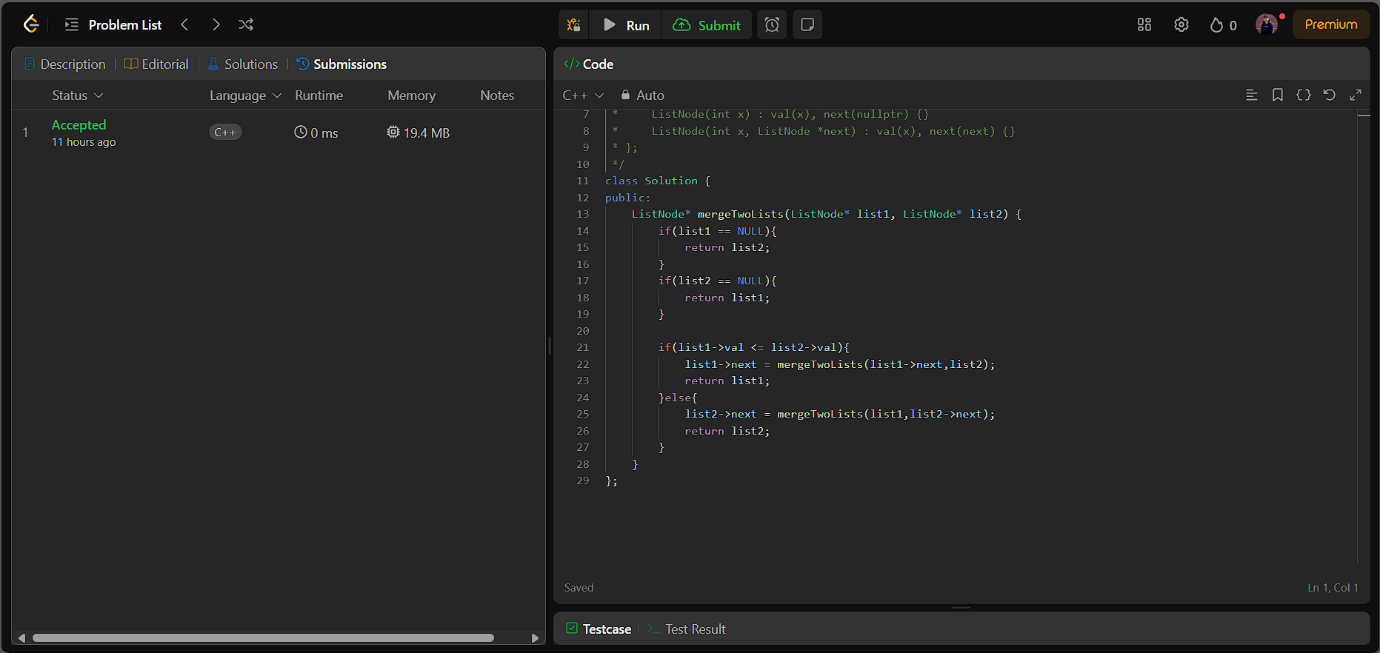
            return list2;

        }

    }

};

**Output:**



1. **Remove duplicated from sorted list 2**

**Program Code:**

class Solution {

public:

    ListNode\* deleteDuplicates(ListNode\* head) {

        if (head == nullptr || head->next == nullptr) {

            return head;

        }

        ListNode\* dummy = new ListNode(0, head);

        ListNode\* prev = dummy;

        ListNode\* curr = head;

        ListNode\* temp = head->next;

        bool flag = false;

        while (temp != nullptr) {

            if (curr->val != temp->val) {

                if (flag) {

                    prev->next = temp;

                    flag = false;

                } else {

                    prev = prev->next;

                }

            } else {

                flag = true;

            }

            temp = temp->next;

            curr = curr->next;

        }

        if (flag) {

            prev->next = temp;

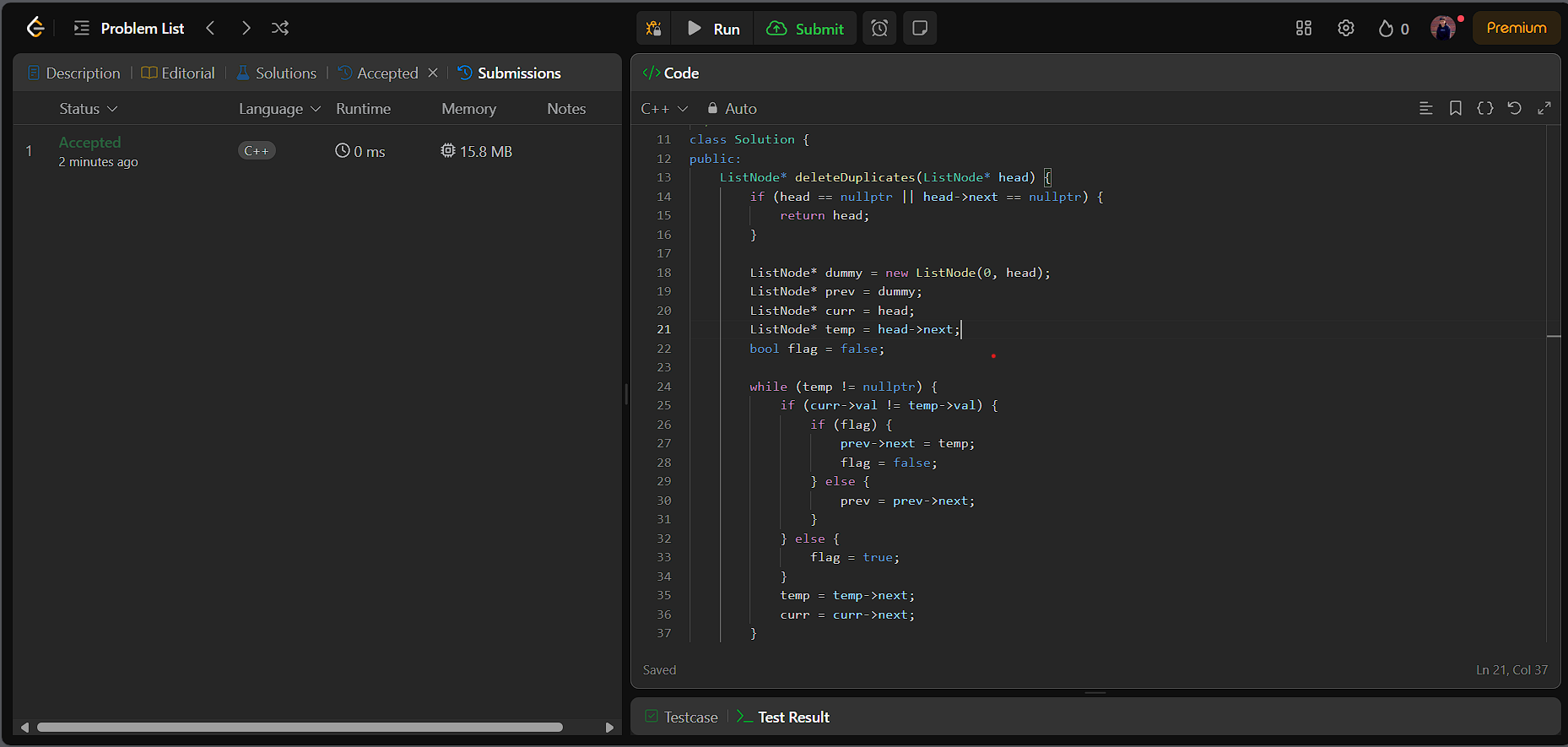
        }

        return dummy->next;

    }

};

**Output:**

****

1. **Detect a Cycle in Linked List**

**Program Code:**

class Solution {

public:

    bool hasCycle(ListNode \*head) {

        ListNode\* fast = head;

        ListNode\* slow = head;

        while(fast != NULL && fast->next != NULL){

            fast = fast->next->next;

            slow = slow->next;

            if(fast == slow){

                return true;

            }

        }

        return false;

    }

};

**Output:**

**A screenshot of a computer program

AI-generated content may be incorrect.**

1. **Reverse Linked List 2**

**Program Code:**

class Solution {

public:

    ListNode\* reverseBetween(ListNode\* head, int m, int n) {

         if(!head || m == n){

            return head;

         }

         ListNode dummy(0, head);

         ListNode\* prev = &dummy;

         for (int i = 0; i < m - 1; ++i)

             prev = prev->next;

            ListNode\* tail = prev->next;

         for (int i = 0; i < n - m; ++i) {

            ListNode\* cache = tail->next;

            tail->next = cache->next;

            cache->next = prev->next;

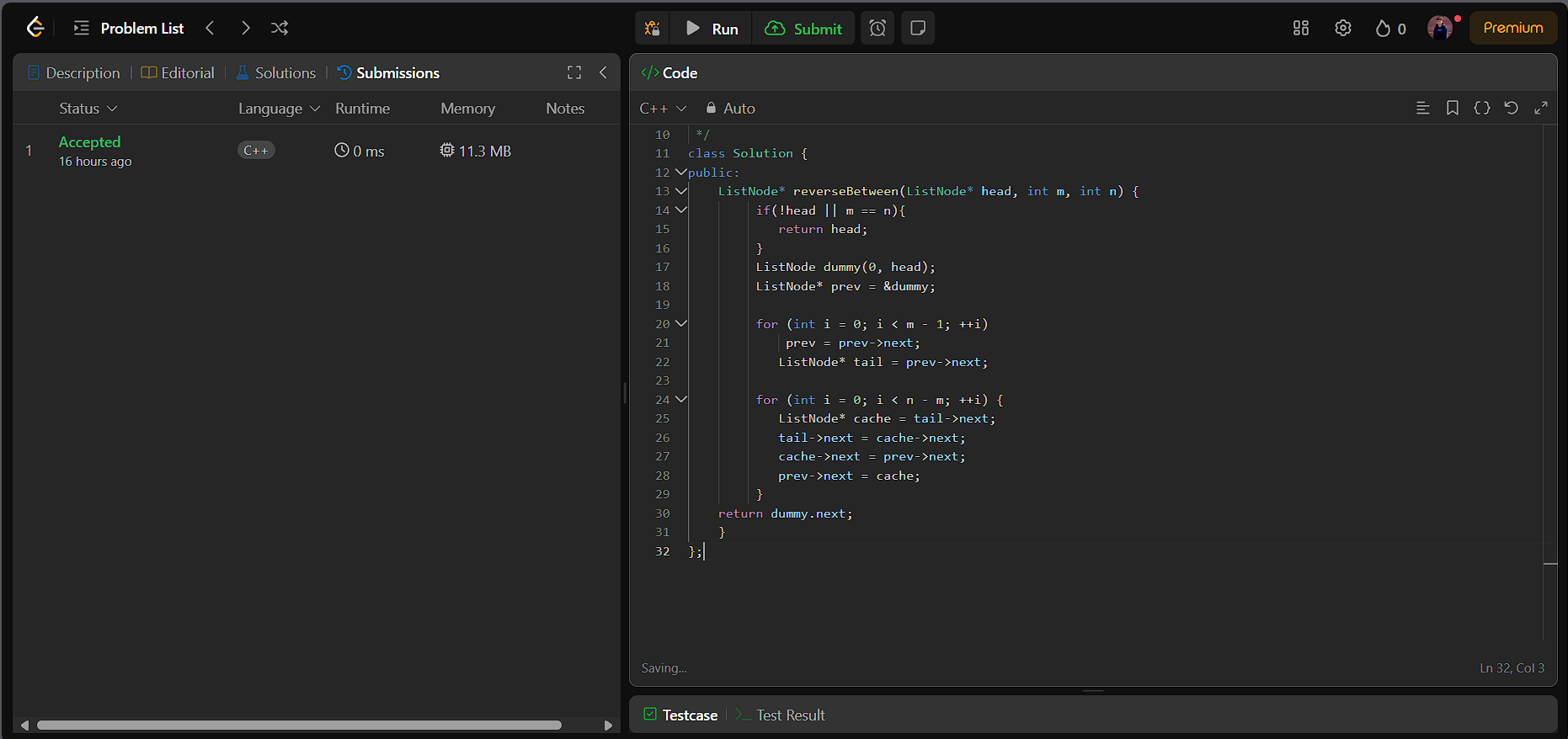
            prev->next = cache;

         }

    return dummy.next;

    }

**Output:**

****

1. **Rotate a List**

**Program Code:**

class Solution {

public:

    ListNode\* rotateRight(ListNode\* head, int k) {

        if (!head || !head->next || k == 0)

        return head;

        ListNode\* tail;

         int length = 1;

        for (tail = head; tail->next; tail = tail->next){

            ++length;

        }

        tail->next = head;

        const int t = length - k % length;

         for (int i = 0; i < t; ++i){

            tail = tail->next;

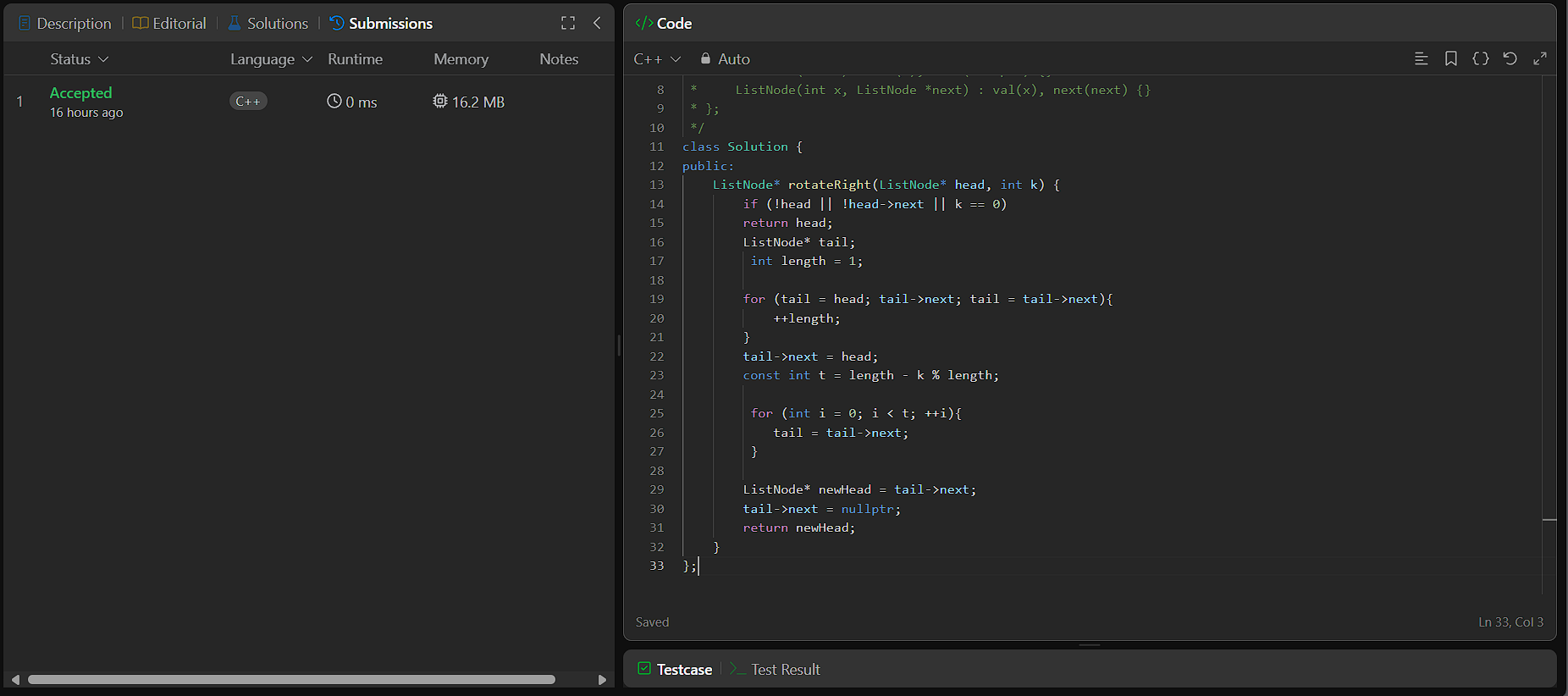
         }

        ListNode\* newHead = tail->next;

        tail->next = nullptr;

        return newHead;

**Output:**

****

1. **Sort List**

**Program Code:**

class Solution {

public:

    ListNode\* sortList(ListNode\* head) {

        if(head == NULL || head ->next == NULL)

            return head;

        ListNode \*temp = NULL;

        ListNode \*slow = head;

        ListNode \*fast = head;

        while(fast !=  NULL && fast -> next != NULL)

        {

            temp = slow;

            slow = slow->next;

            fast = fast ->next ->next;

        }

        temp -> next = NULL;

        ListNode\* l1 = sortList(head);

        ListNode\* l2 = sortList(slow);

        return mergelist(l1, l2);

    }

    ListNode\* mergelist(ListNode \*l1, ListNode \*l2)

    {

        ListNode \*ptr = new ListNode(0);

        ListNode \*curr = ptr;

        while(l1 != NULL && l2 != NULL)

        {

            if(l1->val <= l2->val)

            {

                curr -> next = l1;

                l1 = l1 -> next;

            } else {

                curr -> next = l2;

                l2 = l2 -> next;

            }

        curr = curr ->next;

        }

        if(l1 != NULL)

        {

            curr -> next = l1;

            l1 = l1->next;

        }

        if(l2 != NULL)

        {

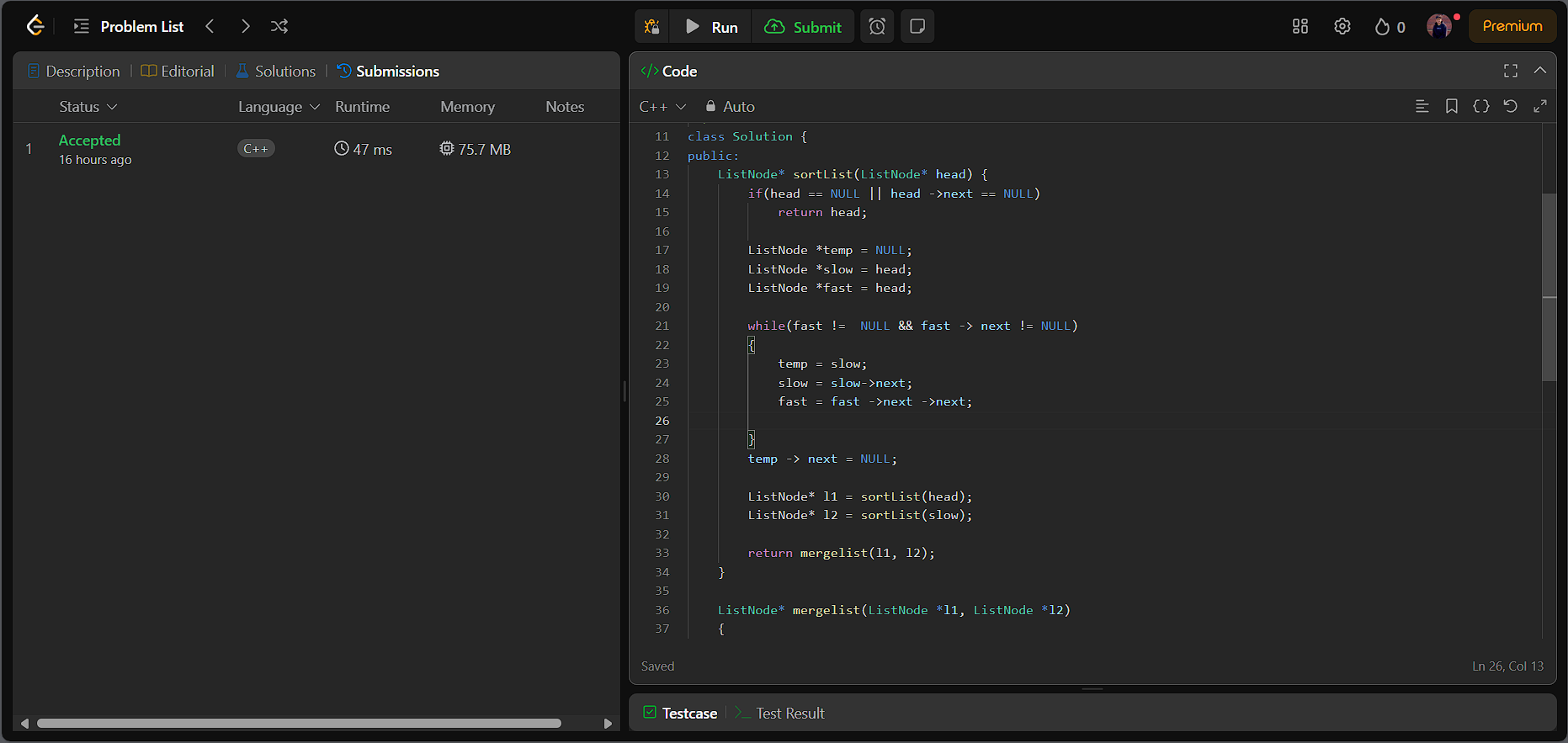
            curr -> next = l2;

            l2 = l2 ->next;

        }

        return ptr->next;

**Output:**

****

1. **Detect a cycle in Linked List 2**

**Program Code:**

class Solution {

public:

    ListNode \*detectCycle(ListNode \*head) {

        ListNode \*slow = head, \*fast = head;

        while (fast && fast->next) {

            slow = slow->next;

            fast = fast->next->next;

            if (slow == fast) break;

        }

        if (!(fast && fast->next)) return NULL;

        while (head != slow) {

            head = head->next;

            slow = slow->next;

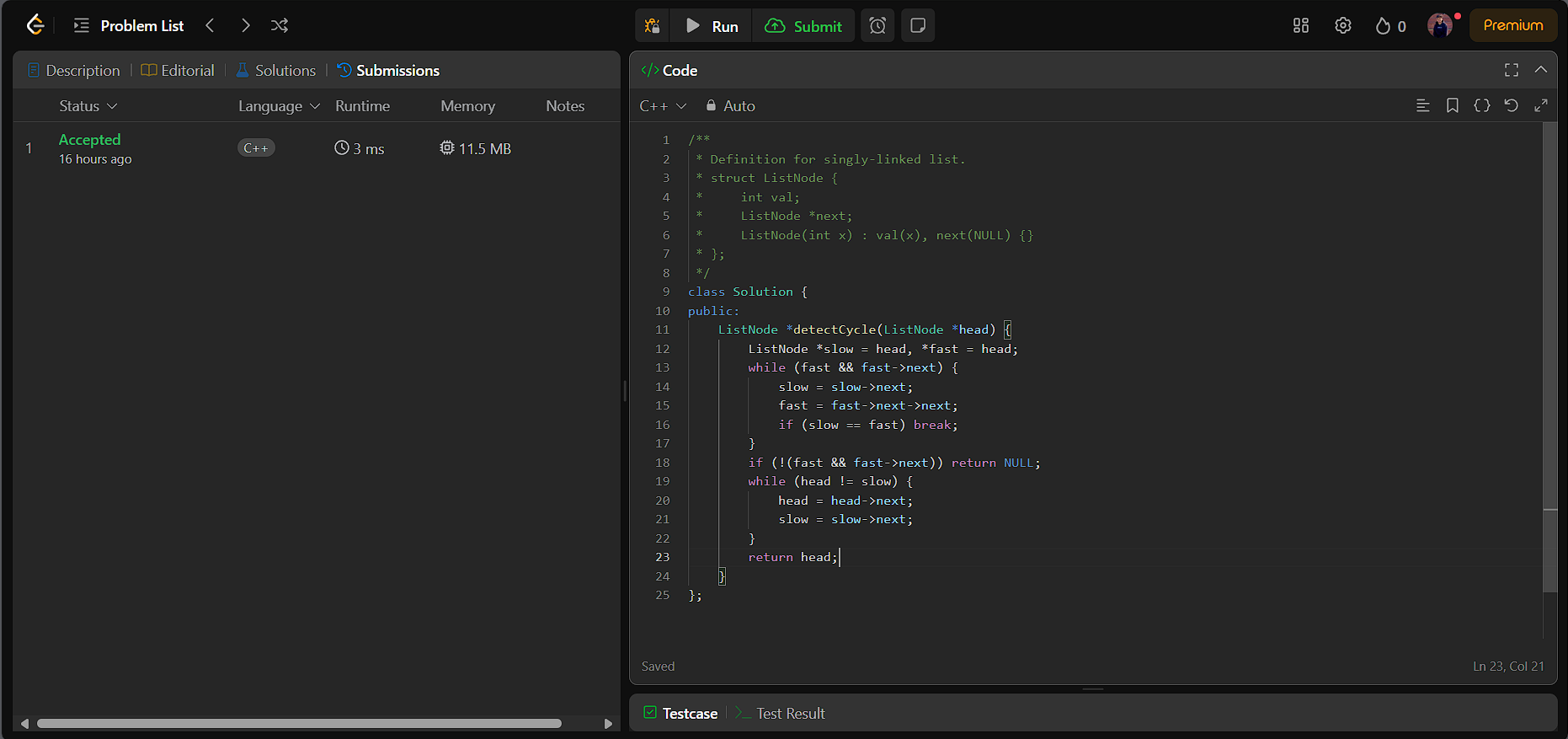
        }

        return head;

    }

};

**Output:**

****