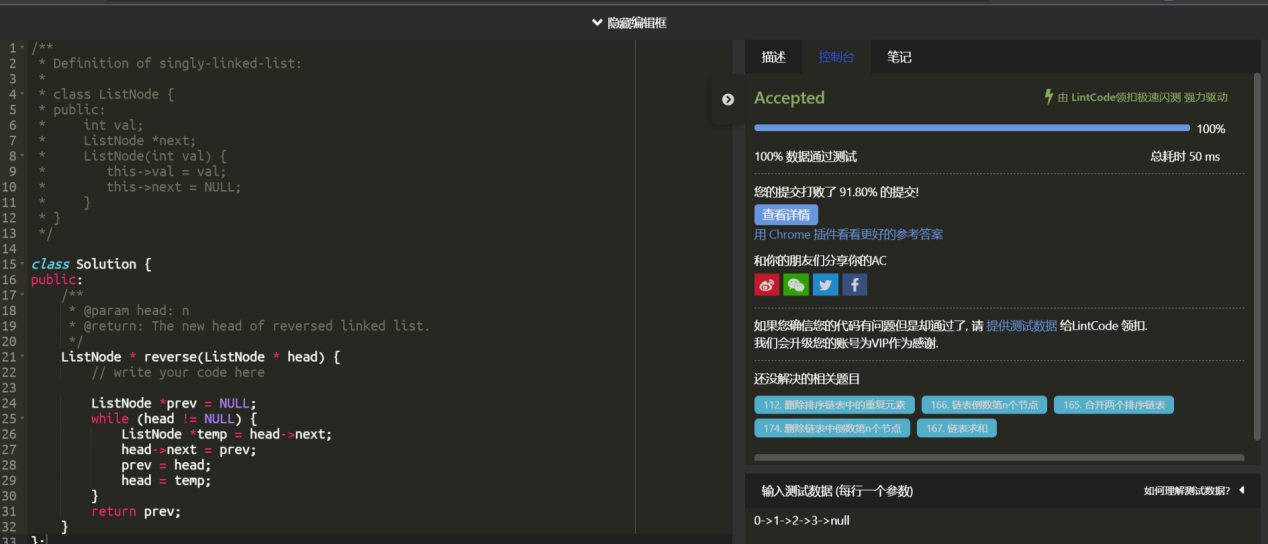
1.翻转链表



思路：

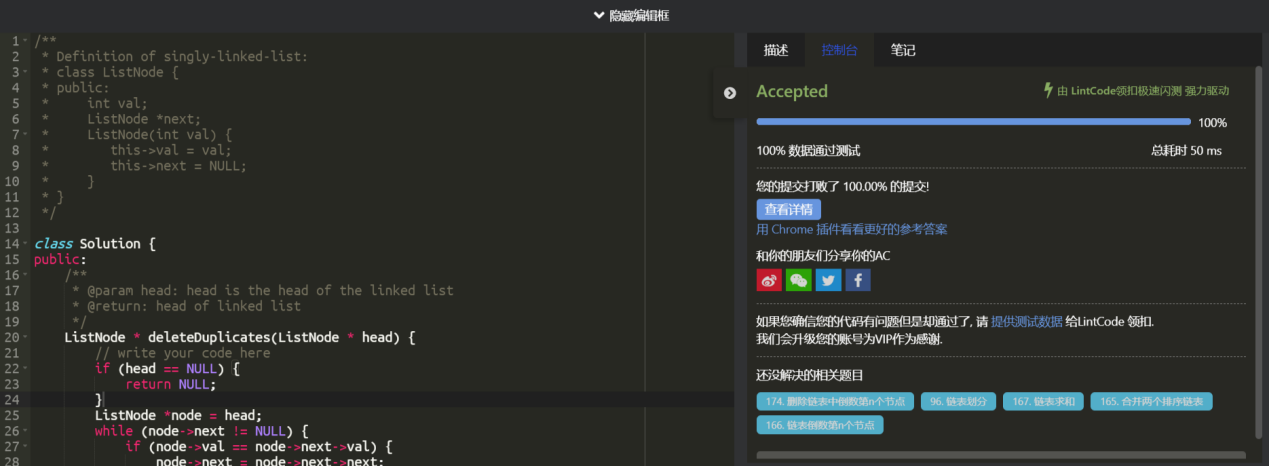
1.定义链表prev保存改变后的链表

2.设计循环，当当前结点为空时停止

3.定义中间变量temp存放当前节点所指向的地址；之后让当前结点指向prev的地址，第一次循环为NULL，再让prev等于当前结点的地址，最后将temp赋给head既实现当前结点后移，每一次循环就把当前的结点作为头，以实现反转

2.删除排序链表中的重复元素

给定一个排序链表，删除所有重复的元素每个元素只留下一个。



思路：

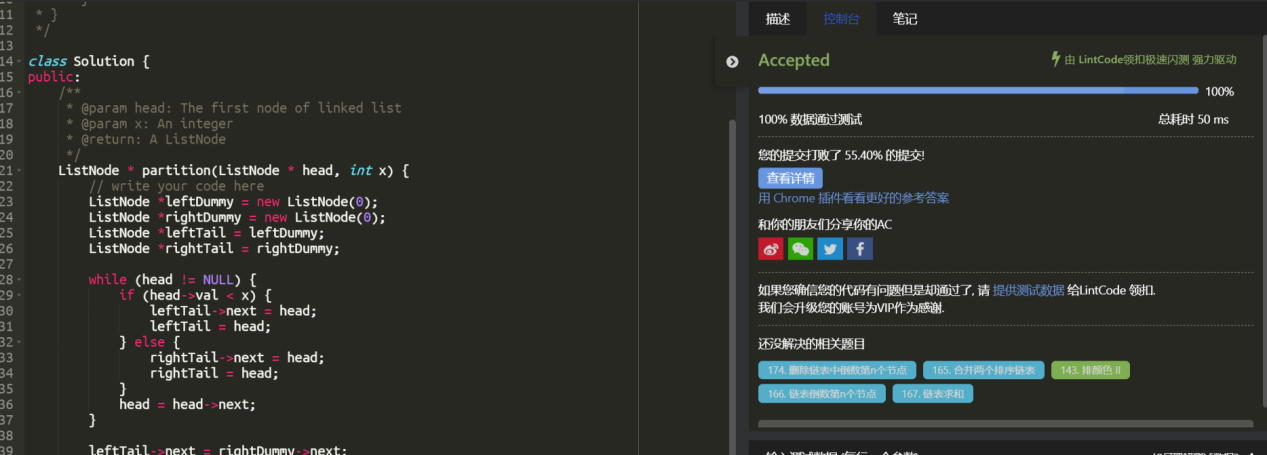
1.定义node定位第一个位置

2.循环判断数据域是否与后一个相同，如果相同则直接将指针指向后者的指针域所指地址，否则正常后移一个

3.链表划分

给定一个单链表和数值x，划分链表使得所有小于x的节点排在大于等于x的节点之前。

你应该保留两部分内链表节点原有的相对顺序。



思路：

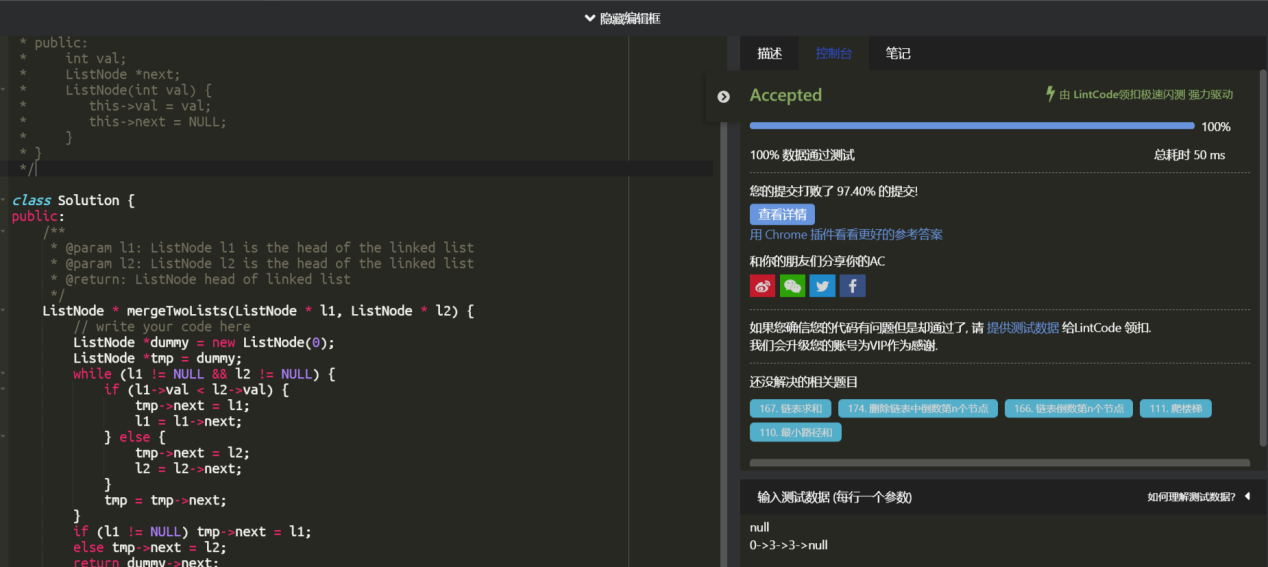
1.考虑用两个容器分别放小于x和大于x的数据，先定义两个新链表

2.循环比较数据域，如果小于x则放入lefttail后，大于时考虑拼接时无法直接用变化后的指针，增加一个righdummy方便使用大于x部分的地址

3.拼接链表将左边部分的next直接用rightdummy的next赋值

4.返回结果时考虑无法直接用链表头的地址，在最开始添加leftdummy方便使用表头的地址

4.合并两个排序链表



思路：

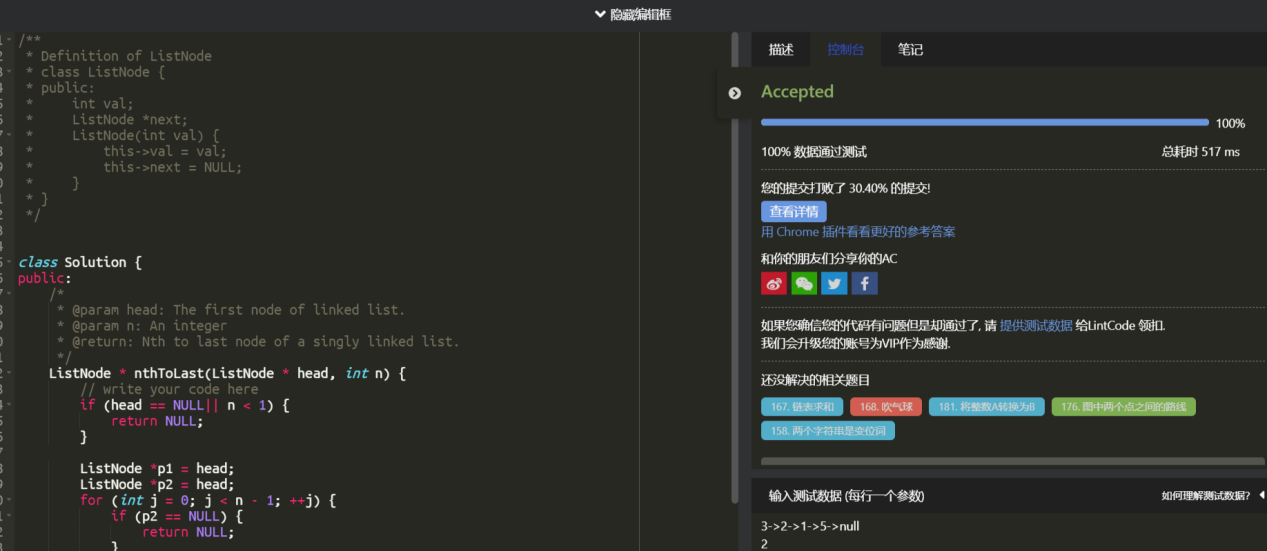
1.定义新链表表头dummy，考虑操作不可逆再定义一个tmp链表

2.循环，同时对排序的两个链表每个元素进行比较，永远只记录下较小的一个

3.注意两个链表有空链表的特殊情况

5.链表倒数第n个节点

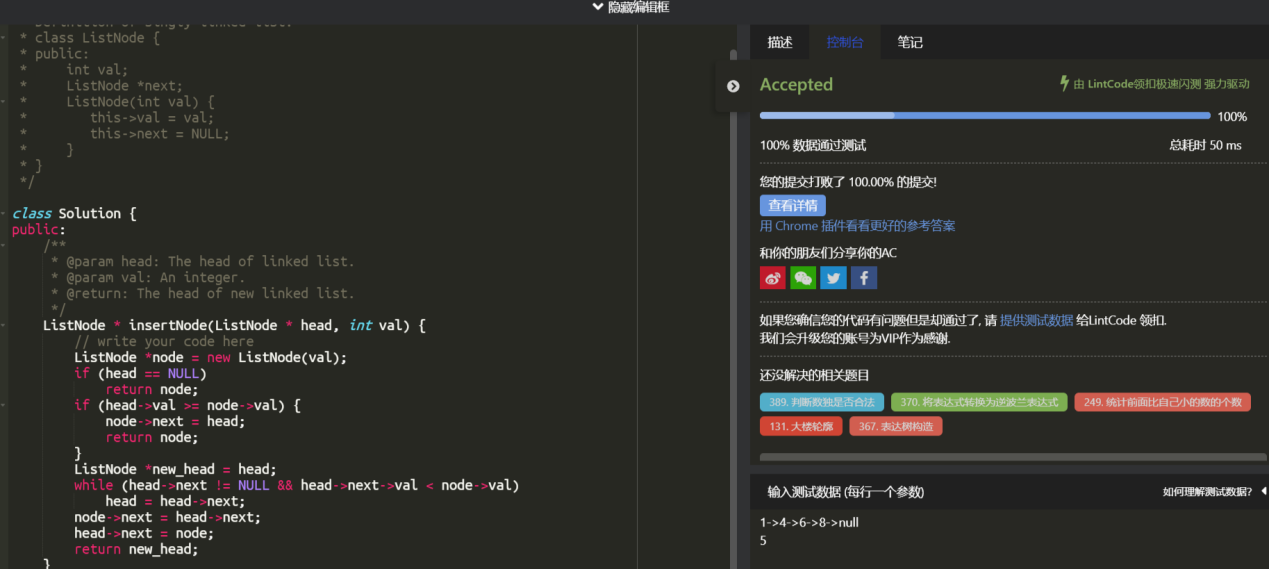
找到单链表倒数第n个节点，保证链表中节点的最少数量为n。



思路：

1.定义两个指针，前一个先走到第n个，然后两个指针同时开始走，当快的那个指针走到链表末尾时，慢的指针即为倒数第n个

6.在排序链表中插入一个节点



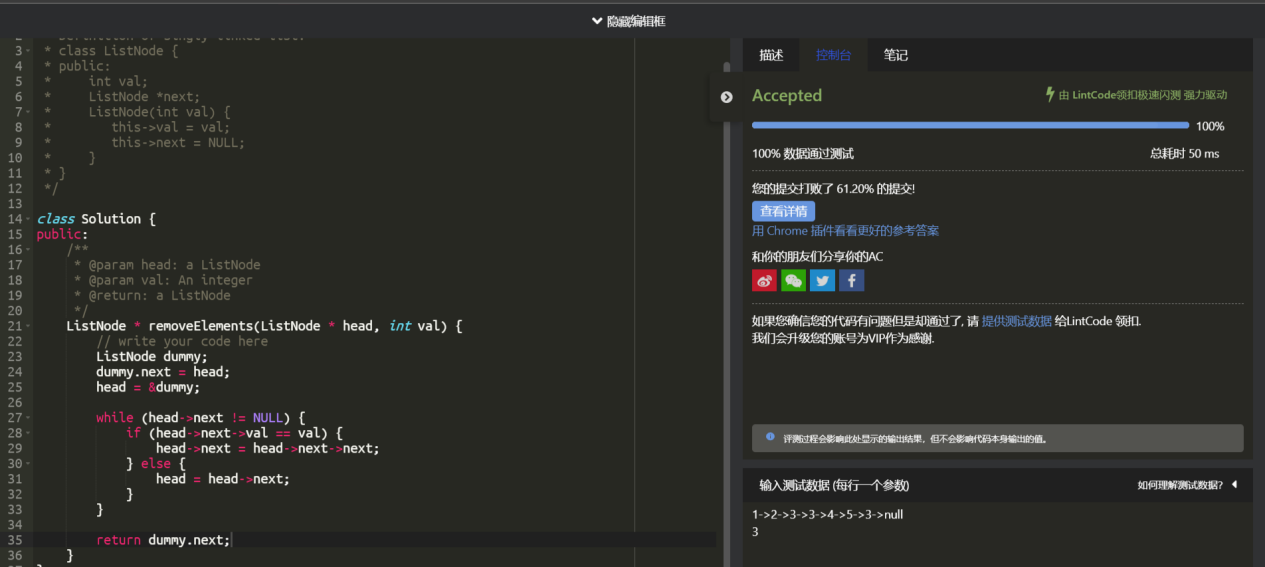
思路：

1.遍历比较链表和所需插入的节点的数据域

2.考虑三种情况：当原链表为空时直接返回结点；当原链表的第一个数据都大于结点时，直接将结点的指针域改为原链表；一般情况，比较当原链表数据小于结点时跳过，否则将前一个的指针替换为结点，结点的指针替换为链表的后一个

7.删除链表中的元素

删除链表中等于给定值 val 的所有节点



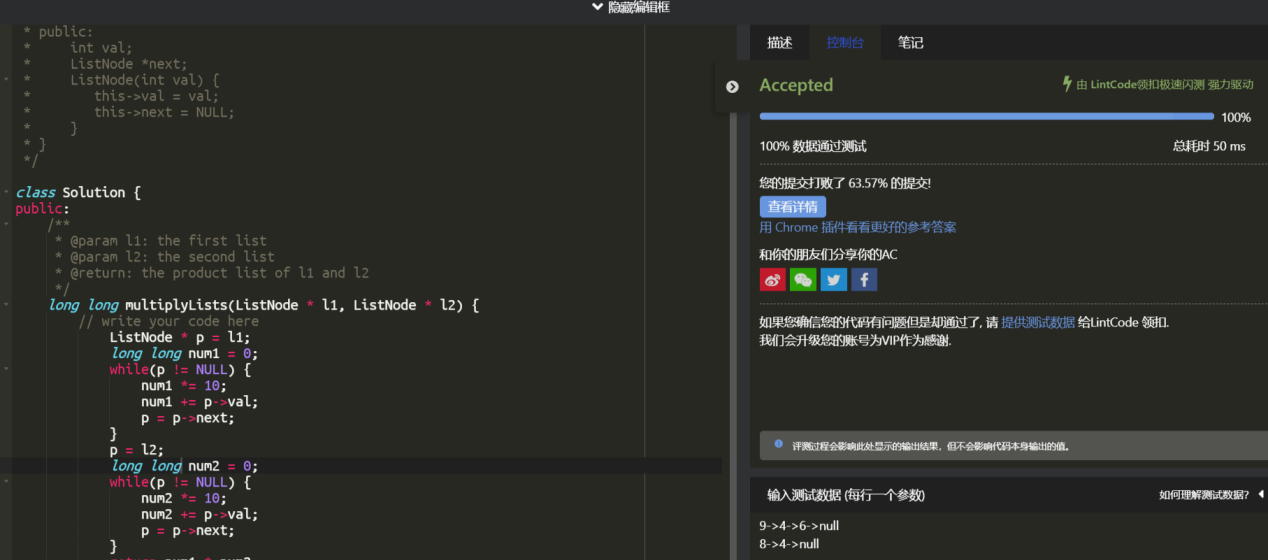
思路：

1.算法很简单，直接遍历比较值是否相等，相等则指针替换

2.要注意的是，当第一个数据即为删除数据时，如果不做处理无法找到，因此相当于在原链表表头前再加一个结点

8.两数相乘

给出两个链表形式表示的数字,写一个函数得到这两个链表相乘乘积。



思路：

1.将两个链表化为数据后直接输入相乘的结果

2.从高位开始每次将原来是数乘10再加上当前结点的数，可以得到链表中的数

9.相反的顺序存储

给出一个链表，并将链表的值以倒序存储到数组中。



思路：

1.用递归，这里需要让先读取的数据后输出。

2.定义一个新函数，当为空时直接返回，否则再运行本身，数据传入当前结点的下一个。这样在读取到null之前函数不会将数据放入容器，此时再运行上一层的push\_back，即可将后输入的数据先放入容器

10.删除链表中倒数第n个节点

给定一个链表，删除链表中倒数第n个节点，返回链表的头节点。



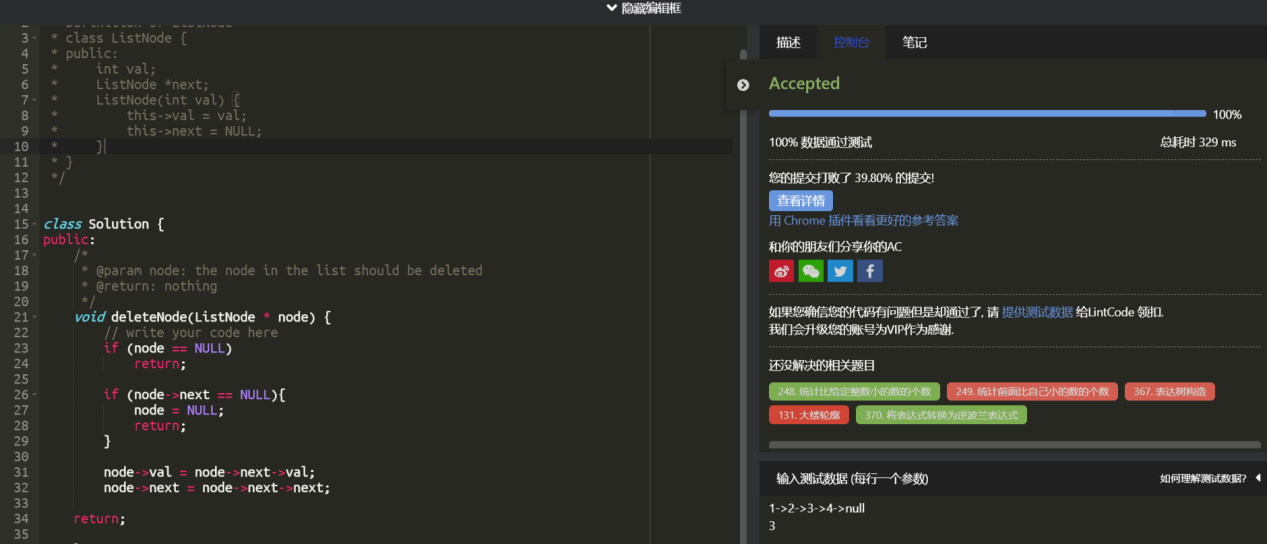
思路：

1.算法与第5题相同

2.当找到倒数第n个结点后，替换指针域为后第二个结点

11.在O(1)时间复杂度删除链表节点

给定一个单链表中的一个等待被删除的节点(非表头或表尾)。请在在 O(1) 时间复杂度删除该链表节点。



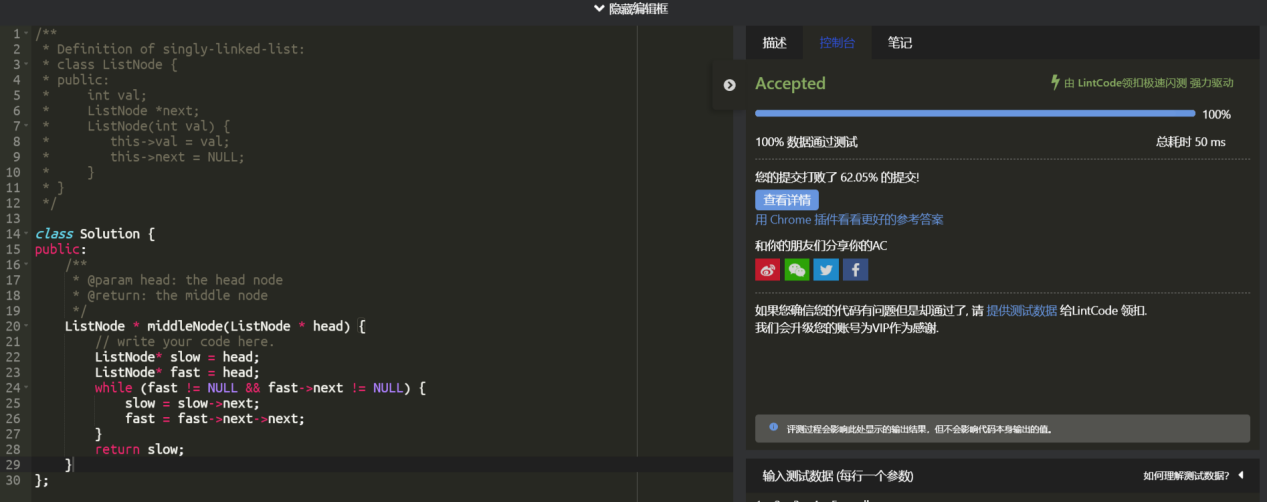
思路：

1.将结点的数据域替换为后一个结点数据域

2.指针域替换为后第二个结点地址

12.链表的中间结点

给定一个带有头结点 head 的非空单链表，返回链表的中间结点。如果有两个中间结点，则返回第二个中间结点。



思路：

1.定义两个指针，一个速度是另一个的2倍既慢指针每次只向后走一个结点，快指针每次走两个结点

2.当快指针走到链表结尾时，慢指针即为中间结点