链表

14 链表中倒数第k个节点【1】【遍历结束条件尽量为最后一个空指针】

- **题目**
  输入一个链表，输出该链表中倒数第k个结点
- **思路**
  挺易错的，对于遍历链表的指针p，通过判断p是否为NULL来判断结束。

1. 首先指针p指向下个节点循环k次，若p不能运行k次，则返回空指针（链表长度小于k）。
2. 此时，p1指向第1个节点，p指向第k+1，while判断p是否为空，不为空则p1和p都指向下个节点；为空则返回p1指针

- **代码实现**

```
class Solution {
public:
    ListNode* FindKthToTail(ListNode* pListHead, unsigned int k) {
        ListNode *pre = pListHead, *cur_p = pListHead;
        while(k--){ //相当于pre在位置1，cur_p在k+1
            if(cur_p == NULL)
                return NULL;
            cur_p = cur_p->next;
        }
        while(cur_p){ //为空节点时结束
            cur_p = cur_p->next;
```

```
            pre = pre->next;
        }
        return pre;
    }
};
```

15 反转链表【1】【基础题】

- **题目**
  输入一个链表，反转链表后，输出新链表的表头。
- **思路**
  采用头插法不断插到反转链表。
- 代码实现

```
class Solution {
public:
    ListNode* ReverseList(ListNode* pHead) {
        ListNode * new_head = NULL; //新的头指针初始为NULL
        while(pHead){
            ListNode * temp = pHead; //temp保存要头插的节点
            pHead = pHead->next; //pHead切换到下一个节点
            temp->next = new_head; //把节点头插到第一个节点
            new_head = temp; //新的头指针重新指向头结点
        }
        return new_head;
    }
};
```

16 合并两个排序的链表【1】【创建头结点】

- **题目**
  输入两个单调递增的链表，输出两个链表合成后的链表，当然我们需要合成后的链表满足单调不减规则。
- **思路**

1. 创建一个头结点，不断的把两个排序链表添加进来。
   2.以其中一个链表为新链表（找出一个头结点），两个整合成一个。

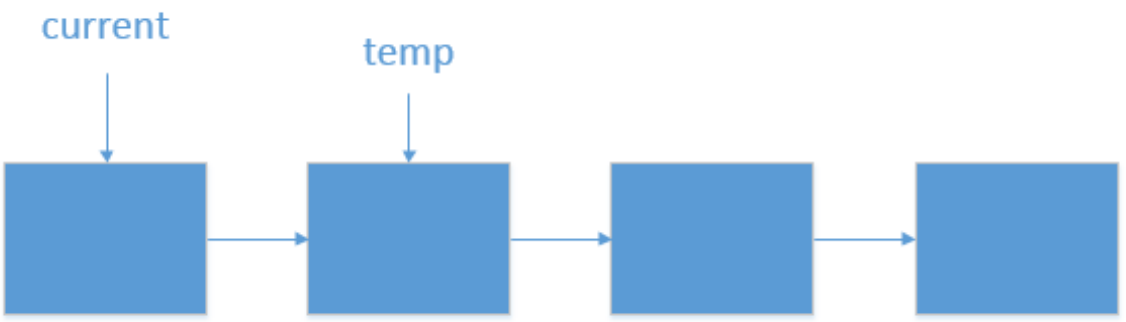
- 代码实现

```
class Solution {
public:
    ListNode* Merge(ListNode* pHead1, ListNode* pHead2){
        ListNode * new_head = new ListNode(-1);
        ListNode * cur_p = new_head;
        while(pHead1 && pHead2){
            ListNode * temp; //保存值比较小的节点
            if(pHead1->val < pHead2->val){
                temp = pHead1;
                pHead1 = pHead1->next;
            }
            else{
                temp = pHead2;
                pHead2 = pHead2->next;
            }
            cur_p->next = temp; //插到尾巴上
            cur_p = temp;
        }
        cur_p->next = pHead1 ? pHead1:pHead2;
        cur_p = new_head->next; //得到真实头指针
        delete new_head;
        return cur_p;
    }
};
```

25 复杂链表的复制【1】【链表遍历细节】

- **题目**
  输入一个复杂链表（每个节点中有节点值，以及两个指针，一个指向下一个节点，另一个特殊指针指向任意一个节点），返回结果为复制后复杂链表的head。（注意，输出结果中请不要返回参数中的节点引用，否则判题程序会直接返回空）
- **思路**
  该题实现复制一个链表，难点在于每个节点还有一个随机指针也需要复制。（额外需要注意这个随机指针，它可能指向节点，也可能指向空的）
  开始判断头指针是否为空，为空直接返回NULL。

由于有特殊随机指针的存在，先复制每个节点在原有节点的后面，再复制原有节点特殊指针的值，最后拆分两个链表。
拆分链表：




```
while(temp->next){
current->next = temp->next;
current = temp->next;
temp->next = current->next;
temp = current->next;
}
current->next = NULL;
```

- 代码实现

```
/*
struct RandomListNode {
    int label;
    struct RandomListNode *next, *random;
    RandomListNode(int x) :
            label(x), next(NULL), random(NULL) {
    }
};
*/
class Solution {
public:
    RandomListNode* Clone(RandomListNode* pHead)
    {
        if(!pHead)
            return NULL;
        RandomListNode * cur_p = pHead;
```

```
        while(cur_p){  //复制节点
            RandomListNode * node = new RandomListNode(cur_p->label);
            node->next = cur_p->next;
            cur_p->next = node;
            cur_p = node->next;
        }
        cur_p = pHead;
        while(cur_p){ //复制随机指针
            if(cur_p->random)
                cur_p->next->random = cur_p->random->next;
            cur_p = cur_p->next->next;
        }
        RandomListNode * copy_head = pHead->next, *  temp =
copy_head;
        cur_p = pHead;
        while(temp->next){ //拆分链表
            cur_p->next = temp->next;
            cur_p = temp->next;
            temp->next = cur_p->next;
            temp = cur_p->next;
        }
        cur_p->next = NULL;
        return copy_head;
    }
};
```

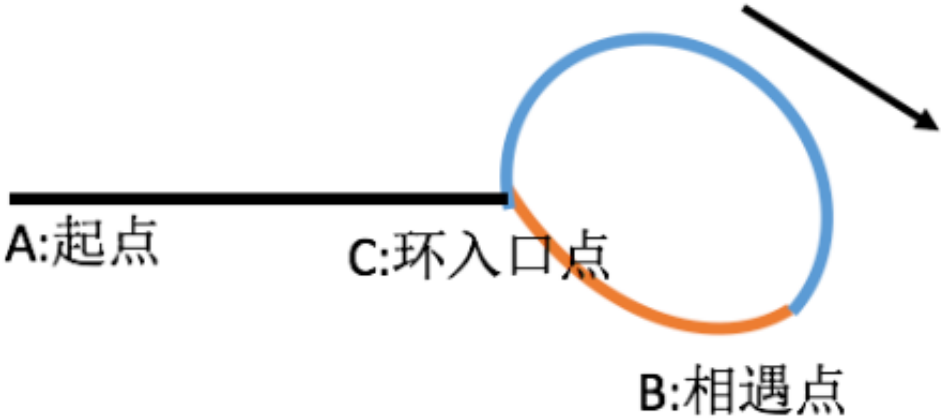
36 两个链表的第一个公共节点

- **题目**
  输入两个链表，找出它们的第一个公共结点
- **思路**
  一个想法真的很巧妙。
  A链表非公共长度x，公共长度z，总长度x+z
  B链表非公共长度y，公共长度z，总长度y+z
  通过A,B链表遍历x+y+z来相遇，若长度相同提前相遇。
  后来想了一下，如果A,B链表长度不同，且无公共节点时，此程序会进入死循环。
- 代码实现

```
class Solution {
public:
    ListNode* FindFirstCommonNode( ListNode* pHead1, ListNode*
pHead2) {
        ListNode * p1 = pHead1, *p2 = pHead2;
        while(p1 != p2){
            p1 = p1 ? p1->next:pHead2;
            p2 = p2 ? p2->next:pHead1;
        }
        return p1;
    }
};
```

55 链表中环的路口节点【1】

- **题目**
  给一个链表，若其中包含环，请找出该链表的环的入口结点，否则，输出NULL。
- **思路**
  1、首先确定头指针不为空且下个节点也不为空（一个节点也能形成环）
  2、接着定义两个指针p1，p2都指向头指针，p1走一步，p2走两步（保证p2的两步都不能指向空），总会在环中相遇，P1走的x步，p2走的2x步，且2x-x=kn，差值是环长度的k倍，假设相遇B点。
  3、再p2指向头指针，p1保持原来的指针，p1和p2各走一步，直到p1=p2，此时就为入口。
  为什么就是路口呢？
  设C到B顺时针长度为len，他俩走2x步肯定会在B点相遇，那么它俩同时倒退len步是不是就退到了C点，所以他们肯定在环路口点C相遇，走的步数最大是 2x-len（可能会更小，不确定）。



- 代码实现

```
class Solution {
public:
    ListNode* EntryNodeOfLoop(ListNode* pHead){
        if(!pHead || !pHead->next)
            return NULL;
        ListNode * p1 = pHead, p2 = pHead;
        while(p2 && p2->next){    //p2不为空
            p1 = p1->next;
            p2 = p2->next->next;  //每次比p1多走一步
            if(p1==p2){         //相遇的时候
                 p2 = pHead;
                while(p1!=p2){  找环的入口点
                    p1 = p1 -> next;
                    p2 = p2 -> next;
                }
                return p1;
            }
        }
        return NULL;
    }
};
```

56 删除链表中重复的节点【2】

- **题目**
  在一个排序的链表中，存在重复的结点，请删除该链表中重复的结点，重复的结点不保留，返回链表头指针。 例如，链表1->2->3->3->4->4->5 处理后为 1->2-

>5

- **思路**
  对于此类第一个节点不确定的情况，采用新建一个头结点。对于需要删除节点的，肯定是需要保存上一个节点的指针。
  首先cur指向第一个节点，
  主循环中，保证cur以及cur->next非空，
  如果当前节点和下个节点相等，则保存值equal_val，cur递归遍历直到指针为空或指向的val不等于equal_val时，把pre->next指向cur，这样就删去了全部重读的节点。
  如果当前节点和下个节点不相等，则pre = cur，cur = cur->next;
- 代码实现

```
class Solution {
public:
    ListNode* deleteDuplication(ListNode* pHead){
        if(!pHead)
            return NULL;
        ListNode * new_head = new ListNode(-1);
        new_head->next = pHead;
        ListNode * pre = new_head, *cur_p = pHead; //cur_p保存当前节点，pre 保存上个不重复节点
        while(cur_p && cur_p->next){
            if(cur_p->val == cur_p->next->val){
                int equal_val = cur_p->val;
                while(cur_p && cur_p->val == equal_val)
                    cur_p = cur_p->next;
                pre->next = cur_p;
            }
            else{
                pre = cur_p;
                cur_p = cur_p->next;
            }
        }
        return new_head->next;
    }
};
```