1.两数相加

Input: (7 -> 2 -> 4 -> 3) + (5 -> 6 -> 4)

Output: 7 -> 8 -> 0 -> 7

- 链表是最高位到最低位

```
class Solution
{
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
        stack<int>st1;
        stack<int>st2;
            while(l1!=nullptr) { //将链表遍历入栈
                st1.push(l1->val);
                l1=l1->next;
            }
            while(l2!=nullptr){ //将链表遍历入栈
                st2.push(l2->val);
                l2=l2->next;
            }
        int c=0; //c是进位
        ListNode* New=nullptr;
        while(!st1.empty()|||!st2.empty()||c!=0){ //循环条件：只要有一个栈不为空或者进位不为空
            int a=st1.empty()?0:st1.top(); //a就是链表1中的值
            int b=st2.empty()?0:st2.top(); //b就是链表2中的值
            if(!st1.empty()) st1.pop(); //栈不为空时 将a出栈
            if(!st2.empty()) st2.pop(); //栈不为空时 b出栈
            int cc=a+b+c; //cc是两结点的值之和
            c=cc/10; //进位如果超过10 ,/10得1 => 进一位 c=1
            cc%=10; //对10取余即取个位
            ListNode* ccnode=new ListNode(cc); //用cc值作为新结点
            ccnode->next=New; //头插法 New一开始指向空
            New=ccnode; //再更新New指向刚刚插在自己上的结点 这样就会实现头插法
    }
        return Next;
    }
};
```

2.回文链表

快慢指针

慢指针边走边掉头，当快指针走完了，前半个链表已经逆转结束了，然后将两个长度相等的链表挨个对比判断是否相等。

```
class Solution {
public:
    bool isPalindrome(ListNode* head) {
        ListNode*fast=head,*slow=head,*cur=head,*pre=nullptr;
        if(head==nullptr||head->next==nullptr){
            return true;
        }
        while(fast&&fast->next){//当快指针走完了，前半个链表已经逆转结束了。
            cur=slow; //cur记录当前位置
            slow=slow->next;
            fast=fast->next->next;
            //头插法逆转前半部分
            cur->next=pre;
            pre=cur;
        }
        if(fast) slow=slow->next;//如果fast为null  说明链表长度为单数 slow已经在中间的结点。
                                 //如果fast还没到null说明长度为偶数 再让slow作为后半部分的第一个
                                 //slow用作遍历后半链表

       //这时候开始遍历两个子链表 判断是否相等 如果相等则为回文链表
        while(pre&&slow){
            if(pre->val==slow->val){
                pre=pre->next;
                slow=slow->next;
            }
            else return false;
        }
        return true;
    }
};
```

3.合并有序链表

4.相交链表

5.分隔链表

6.删除倒数第n个结点

7.奇偶链表