> 为了让读者更加熟悉链表的操作，这里讲一个算法题：**链表的逆置**

问题描述：

比方说，一个不带头节点的单链表原来从头到尾存储的是（1, 2, 3, 4, 5），逆置后链表从头到尾存储的是（5，4，3，2，1）

解决思路：

我暂时想到的有三种解法：

1）从头到尾依次访问旧的链表节点，每访问一个，就将这个节点的数据用头插的方法插入到新的链表中，这样当旧的链表访问完毕时，新的链表就构造成功了，然后释放掉原来的链表空间，将头指针指向新的链表头节点。

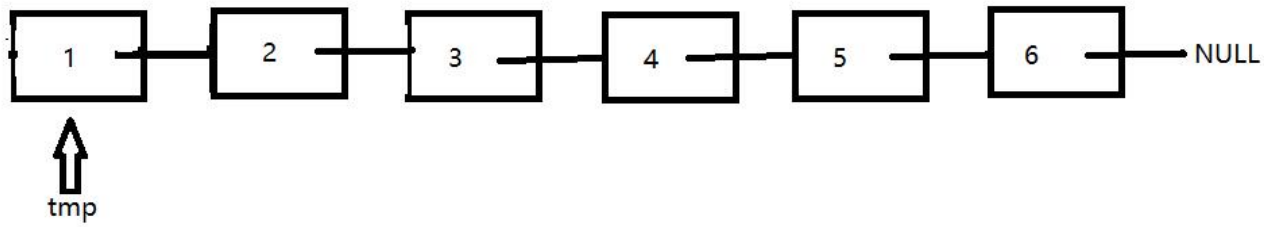
2）和第一个方法类似，也是从头到尾访问旧的链表节点，不同之处在于：每访问一个节点，就将这个节点插入到新的链表中，旧的链表访问完毕时，新的链表也构造成功了。这个因为没有申请新的节点空间，也没有释放旧的节点空间，所以 **效率上要比第一个方法好很多。**

3）采用就地逆置，即在旧的链表上直接逆置，操作完成后，旧的链表就被逆置成功了。**这种方法既容易理解，效率又比较理想**，我会重点讲这个方法。

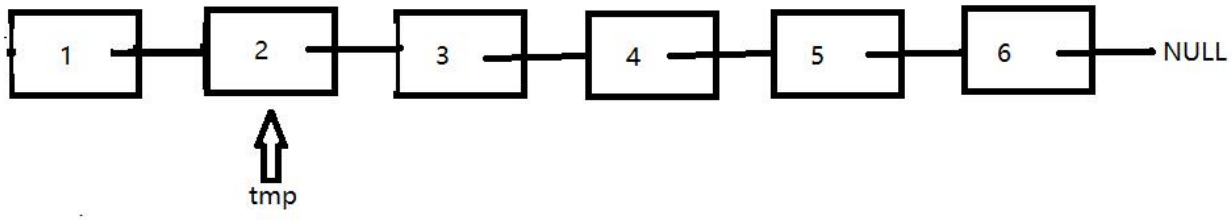
第一种方法

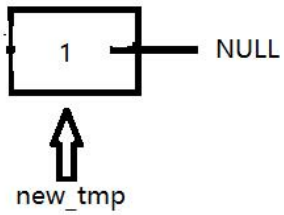
如图所示：

开始的时候定义两个指针，一个指向旧链表，一个指向新链表(开始的时候为空)。



然后申请一个节点并用旧链表的当前节点数值域填充新节点，将新节点采用**头插法**插入到新链表中，然后指向旧链表的指针向后走。

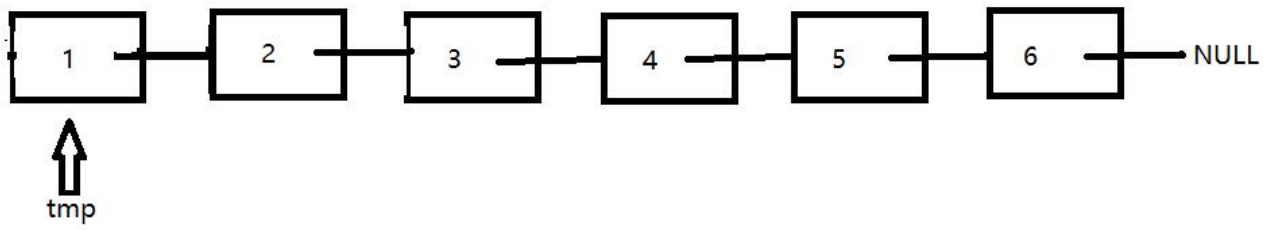




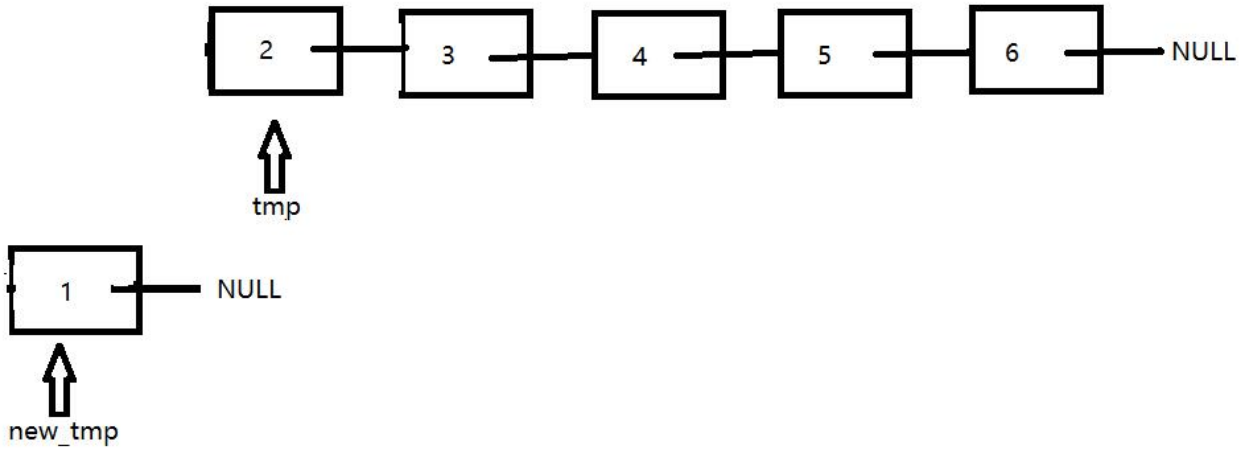
然后循环这个过程，直到遍历完旧链表。

第二种方法

开始的时候状态和第一种相同



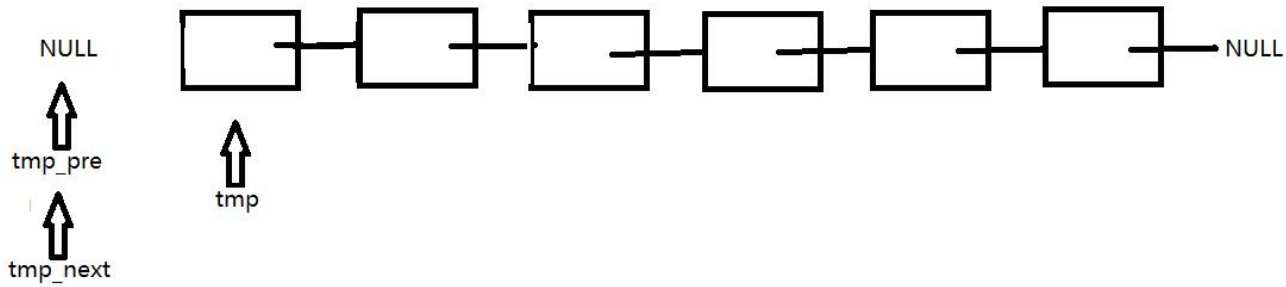
然后将旧链表的当前节点拆下来，**头插法**插入到新链表中。



重复这个过程，直到遍历完旧链表。

第三种方法

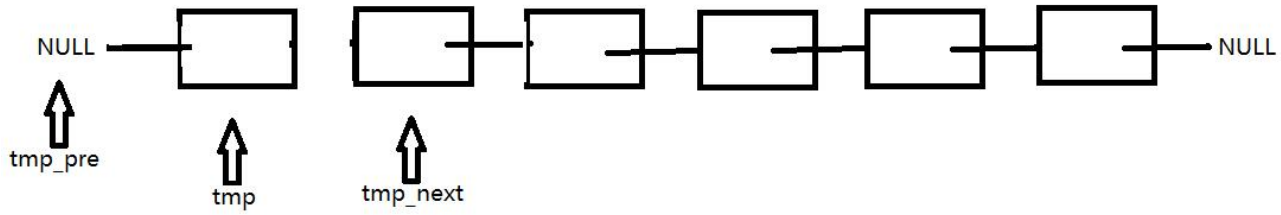
最开始定义三个指针 `tmp`， `tmp_pre`， `tmp_next`， 用 `tmp` 指向当前节点， `tmp_pre` 指向当前节点的上一个节点， `tmp_next` 指向当前节点的下一个节点。开始的时候 `tmp` 指向链表的第一个节点，其他两个指针指向 NULL。



然后更新 `tmp` （当前节点）的指针域，让其指针域指向上一个节点 `tmp_pre`，但是如果直接更新，那么当前节点的下一个节点就找不到了，所以在更新当前节点的指针域之前，先保存下一个节点的地址，也就是让 `tmp_next` 指向 `tmp` 的下一个节点。

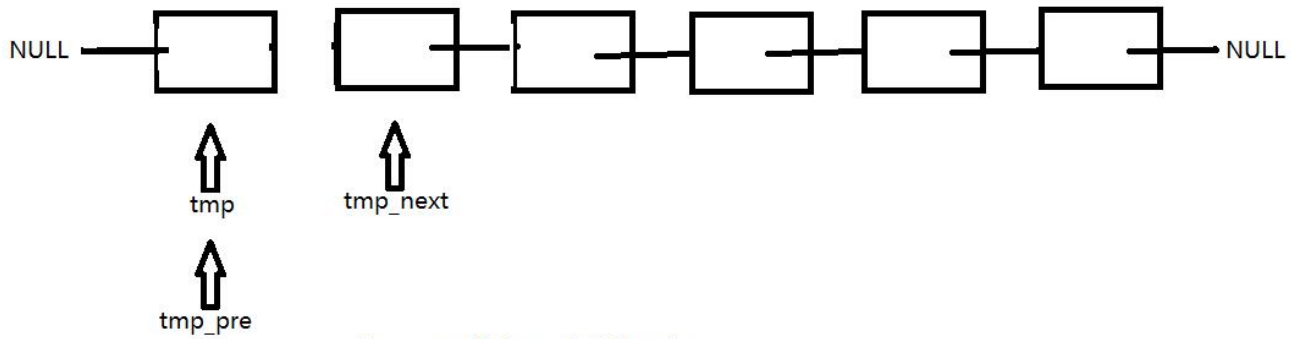
因为这个过程是循环进行的，所以要让指针沿着链表向后走。让 `tmp` 指向它的下一个节点，也就是刚才 `tmp_next` 指向的节点，因为 `tmp` 永远指向当前节点， `tmp_pre` 永远指向当前节点的上一个节点，所以在让 `tmp` 向后走之前，先让 `tmp_pre` 指向 `tmp` 指向的节点。我画图把这个过程走一遍，结合图看这段话就能更容易理解。

1）



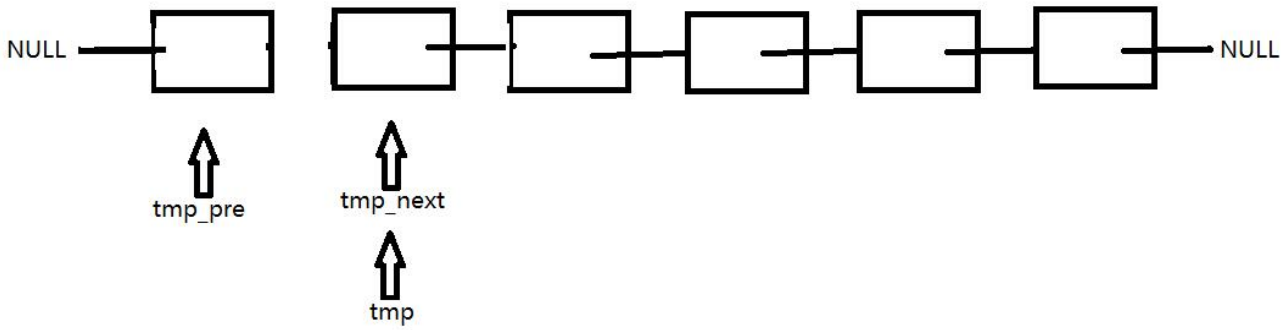
让tmp_next指向tmp的下一个节点，然后更新tmp的指针域

2)



让tmp_pre指向tmp指向的节点

3)



让tmp指向tmp_next指向的节点（也就是第一步保存的tmp的下一个节点）

然后重复上面那个过程，直到走到链表的结尾，逆置成功。

实现代码

> 第三种方法编码实现

```
void ListReverse(PNode* head){
    assert(head);
    PNode ptr_pre = NULL;
    PNode ptr = *head;
    PNode ptr_next = NULL;
    while (ptr){
        ptr_next = ptr->_link;
        ptr->_link = ptr_pre;
        ptr_pre = ptr;
        ptr = ptr_next;
    }
    *head = ptr_pre;
}
```