In this article, we will talk more about details of our algorithm to reverse the linked list.

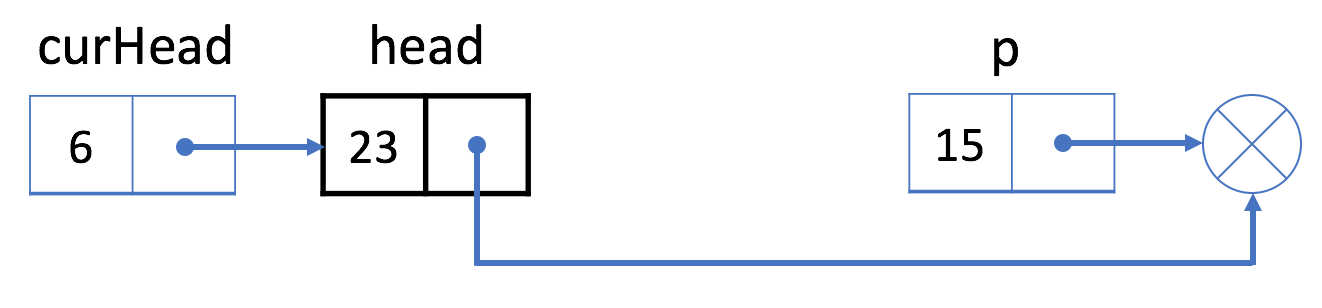
In the solution we mentioned previously, there are two nodes which we should keep track of: the original head node and the new head node.

Therefore, we need to use two pointers in one linked list at the same time. One pointer head always points at our original head node while another pointer curHead always points at our newest head node.

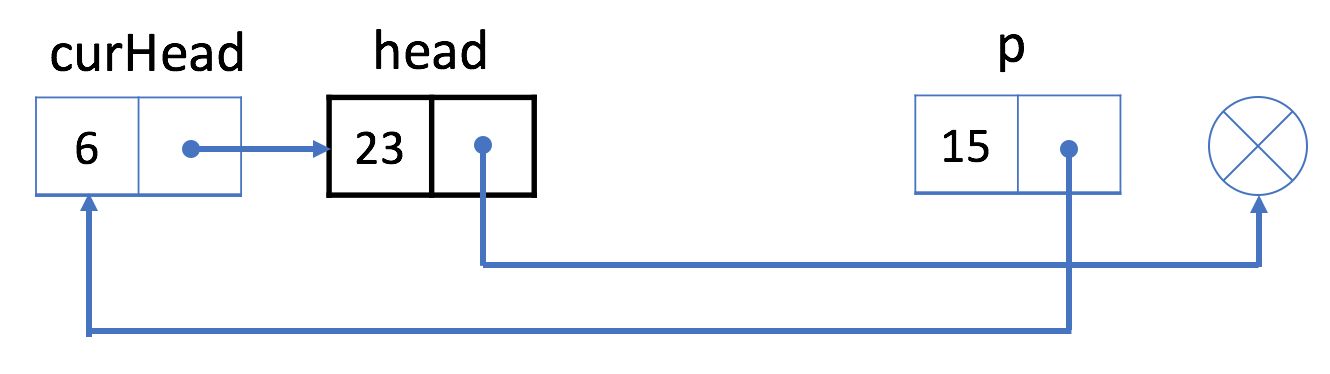
Let's focus on a single step (the 2nd step in the [previous article](https://leetcode.com/explore/learn/card/linked-list/219/linked-list-classic-problem/1204/)). Our goal is to move the next node of head, which is 15, to the head of the list.



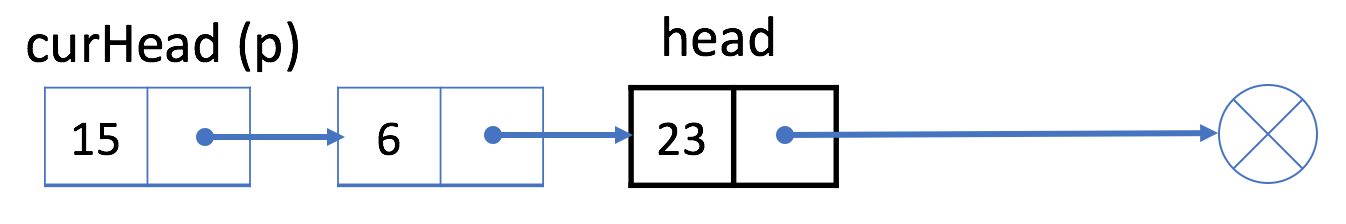
1. First, we use a temporary pointer p to indicate the next node of the head node. And link the "next" field of head to the "next" field of p.



2. Then, we link the "next" field of p to the curHead.



3. Now our linked list actually looks like the picture below. And we set curHead to be p.



By this way, we successfully move node 15 to the head of the list. And we can repeat this process until the next node of head is null.

*Reference Code*

Here we provide code in different languages for your reference:

|  |
| --- |
| /\*\*  \* Definition for singly-linked list.  \* public class ListNode {  \* int val;  \* ListNode next;  \* ListNode(int x) { val = x; }  \* }  \*/  class Solution {  public ListNode reverseList(ListNode head) {  if (head == null) {  return head;  }  ListNode currentHead = head;  while (head.next != null) {  ListNode p = head.next;  head.next = p.next;  p.next = currentHead;  currentHead = p;  }  return currentHead;  }  } |