的一个节点可能代表很复杂的结构，节点值的复制会相当复杂，或者可能改变节点值这个操作都是被禁止的；再如，工程上的一个节点代表提供服务的一个服务器，外界对每个节点都有很多依赖，比如，示例中删除节点 2 时，其实影响了节点 3 对外提供的服务。

这种删除方式的具体过程请参看如下代码中的 removeNodeWired 方法。

```
public class Node {
        public int value;
        public Node next;

        public Node(int data) {
                this.value = data;
        }
}

public void removeNodeWired(Node node) {
        if (node == null) {
                return;
        }
        Node next = node.next;
        if (next == null) {
                throw new RuntimeException("can not remove last node.");
        }
        node.value = next.value;
        node.next = next.next;
}
```

向有序的环形单链表中插入新节点

【题目】

一个环形单链表从头节点 head 开始不降序，同时由最后的节点指回头节点。给定这样一个环形单链表的头节点 head 和一个整数 num，请生成节点值为 num 的新节点，并插入到这个环形链表中，保证调整后的链表依然有序。

【难度】

士 ★☆☆☆

【解答】

直接给出时间复杂度为 $O(N)$、额外空间复杂度为 $O(1)$的方法。具体过程如下：

1．生成节点值为 num 的新节点，记为 node。

2．如果链表为空，让 node 自己组成环形链表，然后直接返回 node。

3．如果链表不为空，令变量 pre=head，cur=head.next，然后令 pre 和 cur 同步移动下去，如果遇到 pre 的节点值小于或等于 num，并且 cur 的节点值大于或等于 num，说明 node 应该在 pre 节点和 cur 节点之间插入，插入 node，然后返回 head 即可。例如，链表 1->3->4->1->…，num=2。应该把节点值为 2 的节点插入到 1 和 3 之间，然后返回头节点。

4．如果 pre 和 cur 转了一圈，这期间都没有发现步骤 3 所说的情况，说明 node 应该插入到头节点的前面，这种情况之所以会发生，要么是因为 node 节点的值比链表中每个节点的值都大，要么是因为 node 的值比链表中每个节点的值都小。

分别举两个例子：示例 1，链表 1->3->4->1->…，num=5，应该把节点值为 5 的节点，插入到节点 1 的前面；示例 2，链表 1->3->4->1->…，num=0，也应该把节点值为 0 的节点，插入到节点 1 的前面。

5．如果 node 节点的值比链表中每个节点的值都大，返回原来的头节点即可；如果 node 节点的值比链表中每个节点的值都小，应该把 node 作为链表新的头节点返回。

具体过程请参看如下代码中的 insertNum 方法。

```
public class Node {
        public int value;
        public Node next;

        public Node(int data) {
                this.value = data;
        }
}

public Node insertNum(Node head, int num) {
        Node node = new Node(num);
        if (head == null) {
                node.next = node;
                return node;
        }
        Node pre = head;
        Node cur = head.next;
        while (cur != head) {
                if (pre.value <= num && cur.value >= num) {
                        break;
                }
                pre = cur;
                cur = cur.next;
        }
        pre.next = node;
        node.next = cur;
        return head.value < num ? head : node;
}
```