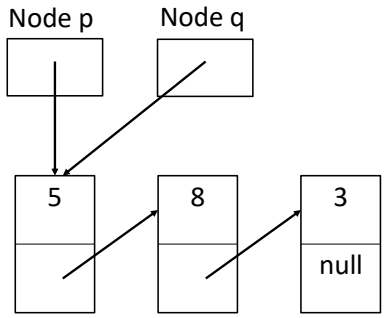
Module IN2002—Data Structures and Algorithms
Answers to Exercise Sheet 4

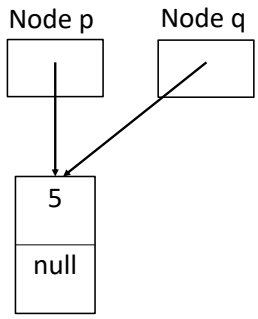
1. Suppose we have `Node p` and `Node q` referring to the same list of nodes containing `[5, 8, 3]`. Draw the list, and the results of the following statements, applied in order:

```
p.next = null;
p = null;
```

- The initial list is:

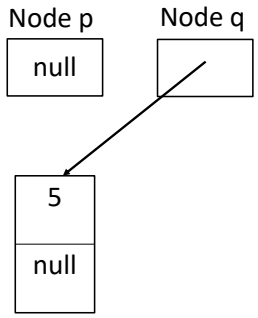



```
p.next = null;
```



- We would have both p and q referring to the same list with a single element (with info 5).

```
p = null;
```

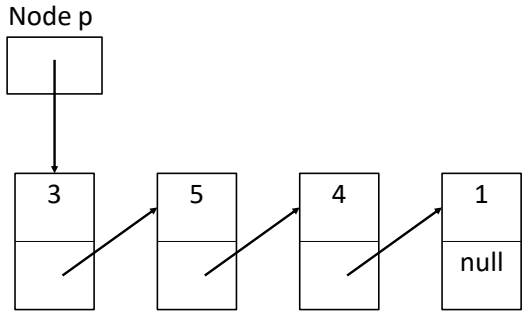


- This sets the value of p to null (no longer refers to a list), whereas q still refers to the same list as before.

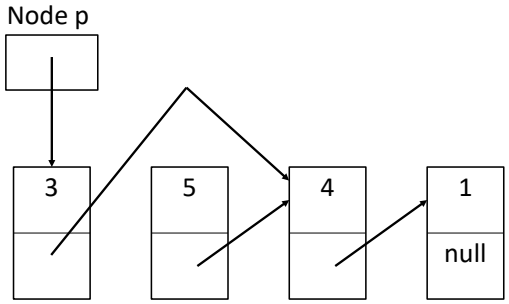
2. Suppose we have `Node p` referring to a list of nodes containing `[3, 5, 4, 1]`. Draw the list, and the results of the following statements, applied in order:

```
p.next = p.next.next;
p.next = new Node(7, p.next);
p.next.next = new Node(8);
p.next.next = p;
```

- The initial list is:

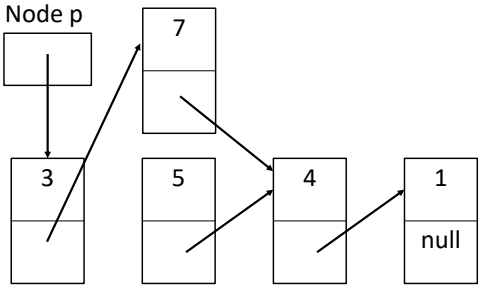



```
p.next = p.next.next;
```

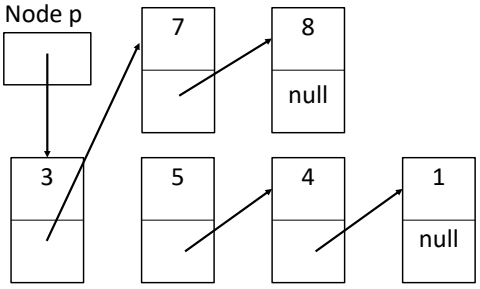


- We could have omitted the node with value 5 above, because nothing is pointing to it, so it is not really in the list any longer.

```
p.next = new Node(7, p.next);
```

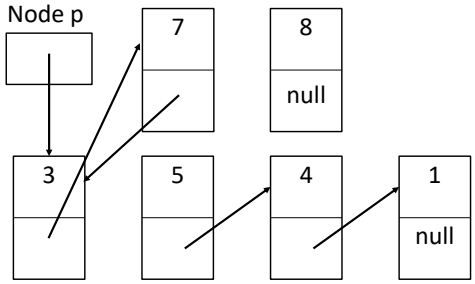



```
p.next.next = new Node(8);
```

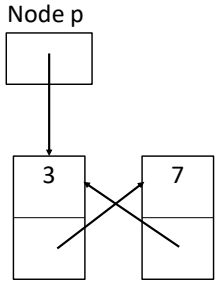


- We could have omitted the nodes with values 5, 4 and 1 above, because nothing is pointing to them, so they are not really in the list any longer.

```
p.next.next = p;
```



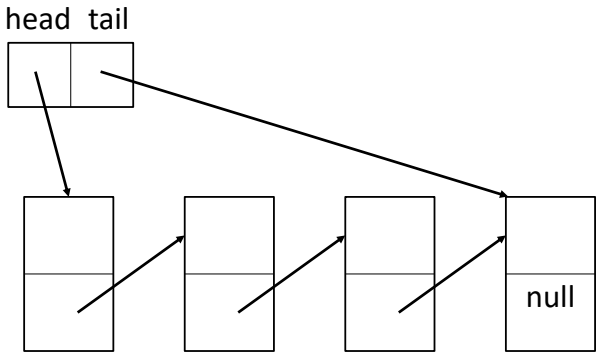
- OR, ignoring all the "dead" elements...



3. Work out (using pictures) what the following procedure does:

```
public void modify(SLList list) {
     if (list.head != null && list.head.next != null) {
          Node tmp = list.head.next;
          list.head.next = tmp.next;
          tmp.next = list.head;
          list.head = tmp;
     }
}
```

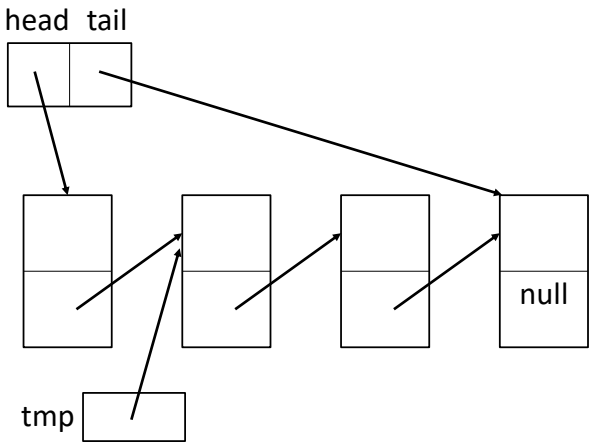
- Let's try the code on this list:



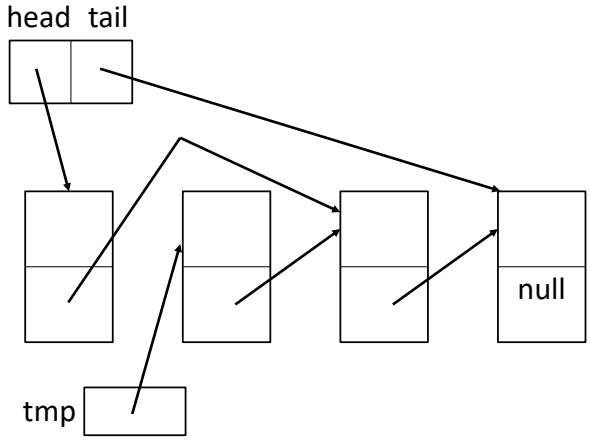

```
     if (list.head != null && list.head.next != null) {
```

- Checks that the list is not empty and has at least two elements.

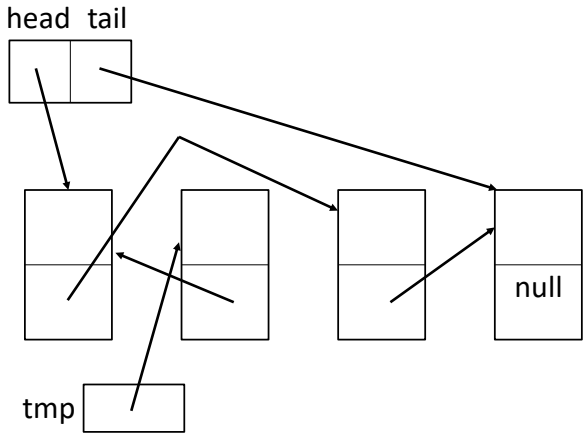
```
Node tmp = list.head.next;
```



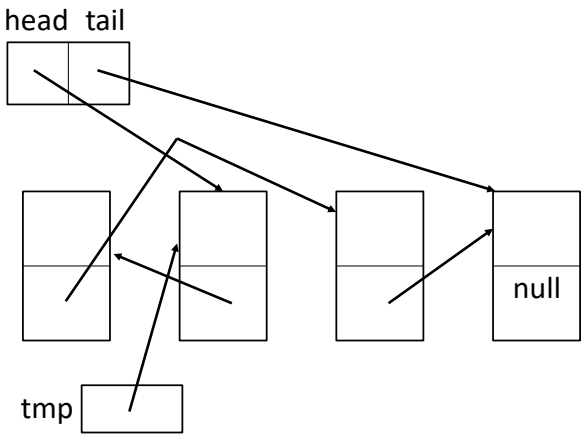

```
list.head.next = tmp.next;
```




```
tmp.next = list.head;
```




```
list.head = tmp;
```



- So what the algorithm does is to switch the two first elements in the list.

4.

a) Write a method that checks whether two singly linked lists have the same elements (in any order). You should not modify either list. What time (in big-O notation) does this method take?

- Assuming that there are no duplicates and that the lists have the same length:

```
boolean comparelists(SLList list1, SLList list2) {
     Node p1 = list1.head ;
     while (p1 != null) {
          Node p2 = list2.head;
          while (p2.info != p1.info) {
               if (p2 == null)
                    return false;
               p2 = p2.next;
          }
          p1 = p1.next;
     }
     return true;
}
```

- This function is $O(n^2)$ time. The outside loop traverses list p1, requiring O(n) iterations. The internal loop traverses p2, requiring O(n) each time. Since the loops are nested, the overall time is $O(n)xO(n) = O(n^2)$

- Note that comparing lengths requires O(n) time, so it would not change the time complexity found in this case.

b) What would be the time complexity if you were checking whether the lists have the same elements in the same order?

- O(n), as for this, both lists would be traversed in parallel, at the same time. In other words, the nodes of list1 are visited one at the time (O(n)), and only one node of list2 is visited in each iteration (O(1)). $O(n)xO(1) = O(n)$.