CS 61B Discussion 2: Pointers Fall 2014

1 Boxes and Pointers

Draw a box and pointer diagram to represent the IntLists after each statement.

```
IntList L = IntList.list(1, 2, 3, 4);
IntList M = L.tail.tail;
N = IntList.list(5, 6, 7);
N.tail.tail.tail = N;
L.tail.tail = N.tail.tail.tail.tail;
M.tail.tail = L;
```

See last page for solution

2 Squaring a Linked List

Implement the following methods to square an IntList destructively and non-destructively.

```
/** Destructively squares each element of the given IntList L.
  * Don't use 'new'; modify the original IntList. */
public static void squareListDestructive(IntList L) {
    while (L != null) {
        L.head *= L.head;
        L = L.tail;
    }
}

/** Non-destructively squares each element of the given IntList L.
  * Don't modify the original IntList */
public static IntList squareListNondestructive(IntList L) {
    if (L == null) {
        return null;
    }
    IntList L2 = new IntList(L.head * L.head, null);
    IntList L2tracker = L2;
    L = L.tail;
    while (L != null) {
        IntList addition = new IntList(L.head * L.head, null);
        L2tracker.tail = addition;
        L2tracker = L2tracker.tail;
        L = L.tail;
    }
    return L2;
}
```

3 Palindrome

Implement the following two methods which determine whether an IntList is a palindrome.

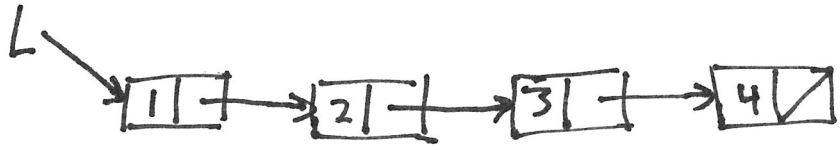
```
/** Non-destructively reverses an IntList L.
  * Do not modify the original IntList. */
public static IntList reverseNondestructive(IntList L) {
    IntList L2 = null;
    while (L != null) {
        L2 = new IntList(L.head, L2);
        L = L.tail;
    }
    return L2;
}

/** Returns whether the IntList L is a palindrome or not,
  * or if it reads the same backwards as forwards. Hint: you may
  * want to use reverseNondestructive. */
public static boolean isPalindrome(IntList L) {
    if (L == null) {
        return false;
    }
    IntList reversed = reverseNondestructive(L);
    while (L != null) {
        if (L.head != reversed.head) {
            return false;
        }
        L = L.tail;
        reversed = reversed.tail;
    }
    return true;
}
```

4 Cycles

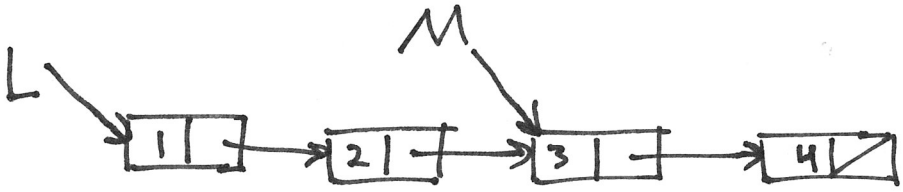
Implement the following method that determines whether an IntList contains a loop.

```
/** Finds whether there is a cycle in the given IntList L */
public static boolean findCycle(IntList L) {
    if (L == null) {
        return false;
    }
    IntList hare = L;
    IntList tortoise = L;
    while (hare != null && hare.tail != null) {
        tortoise = tortoise.tail;
        hare = hare.tail.tail;
        if (hare == tortoise) {
            return true;
        }
    }
    return false;
}
```

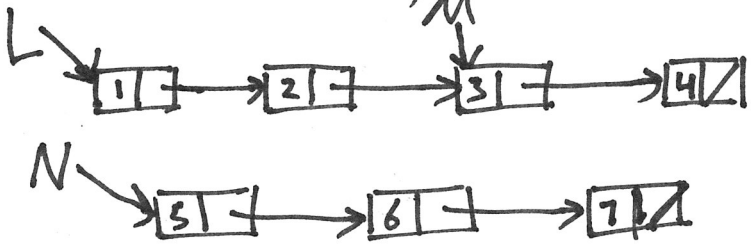
```
IntList L = IntList.list(1, 2, 3, 4);
```



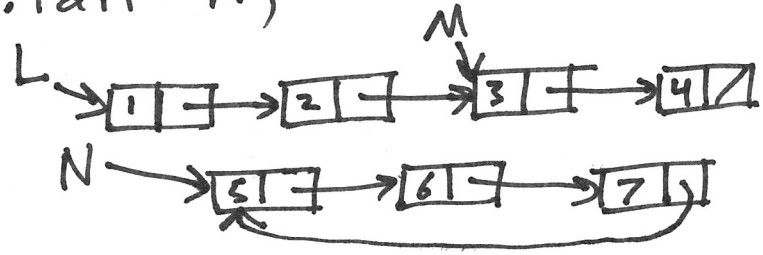

```
IntList M = L.tail.tail;
```



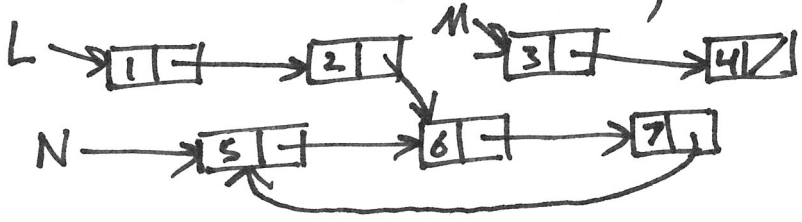

```
N = IntList.list(5, 6, 7);
```




```
N.tail.tail.tail = N;
```




```
L.tail.tail = N.tail.tail.tail.tail;
```




```
M.tail.tail = L;
```