CS 61B Discussion 2: Pointers Fall 2019

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;
IntList 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 Reverse

Implement the following method, which reverses an IntList non-destructively.

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

Extra: Implement the following method which destructively reverses an IntList L

```
/** Destructively reverses an IntList L. */
public static IntList reverseDestructive(IntList L) {
    if (L == null || L.tail == null) {
        return L;
    } else {
        IntList reversed = reverseDestructive(L.tail);
        L.tail.tail = L;
        L.tail = null;
        return reversed;
    }
}
```

3 Insertion

Implement the following method to insert an element into the given position of an IntList.

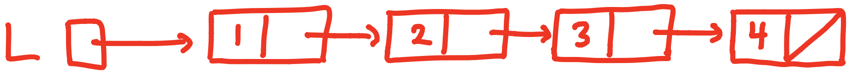
```
/** Insert a new item at the given position in L and return the resulting
  * IntList. If the position is past the end of the list, insert a new
  * node at the end of the list. For example if L is (1, 2, 4) then the
  * result of insert(L, 3, 2) would be (1, 2, 3, 4) */

/** Recursive solution */
public static IntList insert(IntList L, int item, int position) {
    if (L == null) {
        return new IntList(item, L);
    }
    if (position == 0) {
        L.tail = new IntList(L.head, L.tail);
        L.head = item;
    } else {
        L.tail = insert(L.tail, item, position - 1);
    }
    return L;
}

/** Iterative solution */
public static IntList insert(IntList L, int item, int position) {
    if (L == null) {
        return new IntList(item, L);
    }
    if (position == 0) {
        L.tail = new IntList(L.head, L.tail);
        L.head = item;
    } else {
        IntList current = L;
        while (position > 1 && current.tail != null) {
            current = current.tail;
            position -= 1;
        }
        IntList newNode = new IntList(item, current.tail);
        current.tail = newNode;
    }
    return L;
}
```

4 *Extra:* Shifting a Linked List

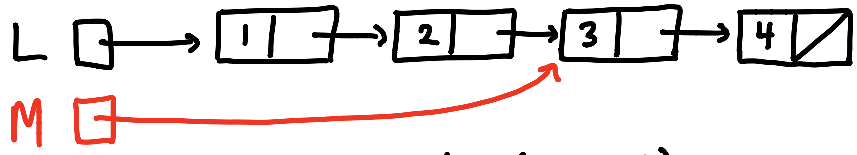
Implement the following methods to circularly shift an IntList to the left destructively and non-destructively.

```
/** Destructively shifts the elements of the given IntList L to
  * the left by one position (e.g. if the original list is
  * (5, 4, 9, 1, 2, 3) then this method should return the list
  * (4, 9, 1, 2, 3, 5)). Returns the first node in the shifted list.
  * Don't use 'new'; modify the original IntList. */
public static IntList shiftListDestructive(IntList L) {
        if (L == null) {
            return null;
        }
        IntList cur = L;
        while (cur.tail != null) {
            cur = cur.tail;
        }
        cur.tail = L;
        IntList ret = L.tail;
        L.tail = null;
        return ret;
}
```

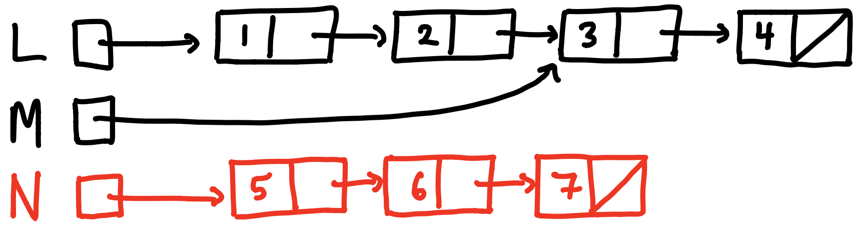
1. IntList L = IntList.list(1, 2,3, 4);



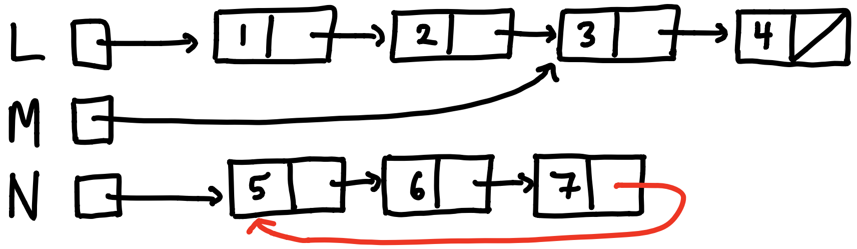
2. IntList M = L.tail.tail;



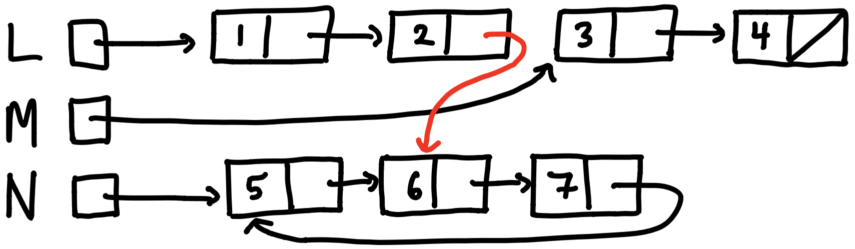
3. IntList N = IntList.list(5, 6,7);



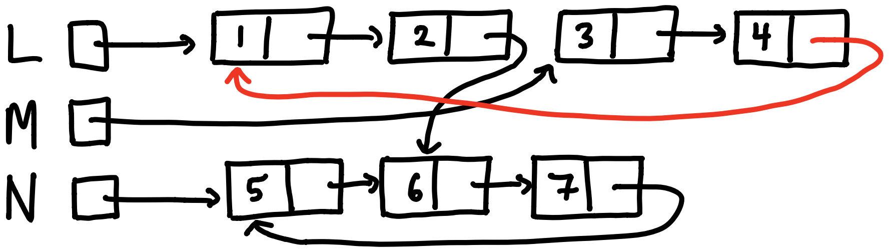
4. N.tail.tail.tail = N;



5. L.tail.tail = N.tail.tail.tail.tail;



6. M.tail.tail = L;


L
1
2
3
4
M
N
5
6
7