CS 61B Discussion 2: Pointers Fall 2017

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 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. */
/** Iterative solution */
public static IntList insert(IntList L, int item, int position) {
    if (L == null || position == 0) {
        return new IntList(item, L);
    }
    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;
}

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

3 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 L2 = null;
    while (L != null) {
        L2 = new IntList(L.head, L2);
        L = L.tail;
    }
    return L2;


}
```

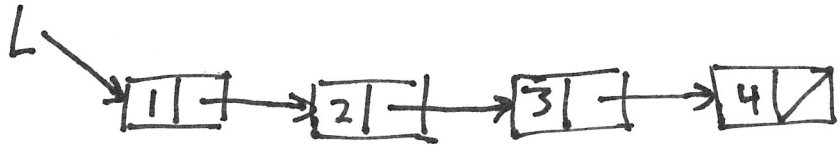
4 Extra for Experts: Shifting a Linked List

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

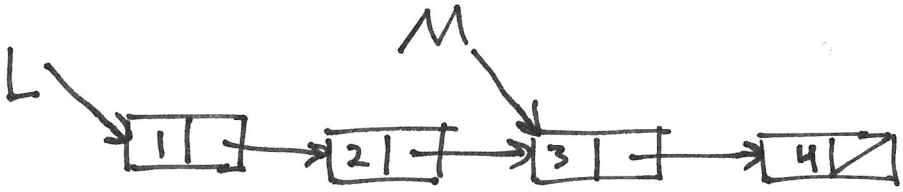
```
/** Destructively shifts the elements of the given IntList L to
  * the left by one position (e.g. if the original list is
  * (5, 4, 9, 1, 2, 3) then this method should return the list
  * (4, 9, 1, 2, 3, 5)). Returns the first node in the shifted list.
  * Don't use 'new'; modify the original IntList.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

/** Non-destructively shifts the elements of the given IntList L
  * to the left by one position. Returns the first node in the shifted list.
  * Don't modify the original IntList. */
public static IntList shiftListNondestructive(IntList L) {
        if (L == null) {
            return null;
        }
        if (L.tail == null) {
            return new IntList(L.head, null);
        }
        IntList cur = L.tail;
        IntList ret = new IntList(cur.head, null);
        IntList L2 = ret;
        while (cur.tail != null) {
            cur = cur.tail;
            L2.tail = new IntList(cur.head, null);
            L2 = L2.tail;
        }
        L2.tail = new IntList(L.head, null);
        return ret;
}
```

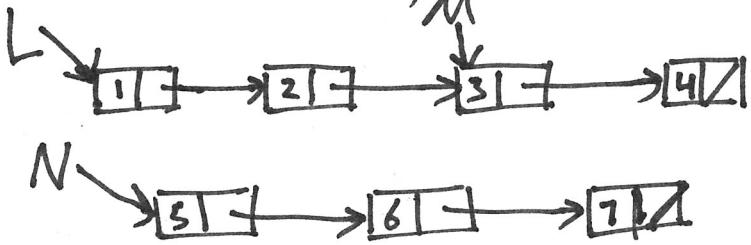
```
IntList L = IntList.list(1, 2, 3, 4);
```



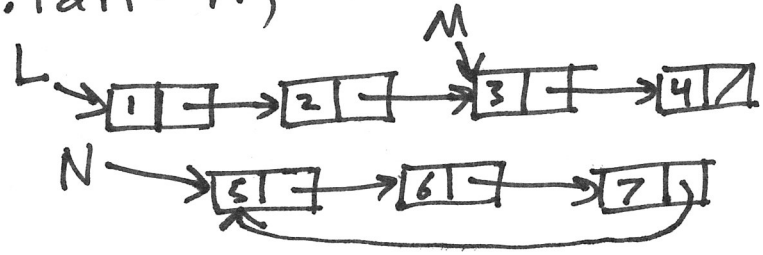

```
IntList M = L.tail.tail;
```



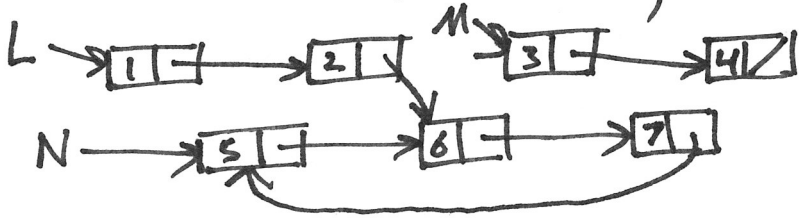

```
N = IntList.list(5, 6, 7);
```




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




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




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