Announcement

- Today: More pointer hacking.
- Handing in labs and homework: We'll be lenient about accepting late homework and labs for the first few. Just get it done: part of the point is getting to understand the tools involved. We will not accept submissions by email.
- For bugs, use bug-submit. There are instructions on the class homepage Announcements.

Destructive Incrementing

Destructive solutions may modify objects in the original list to save time or space:

```java
/** List of all items in P incremented by n. May destroy original. */
static IntList dincrList(IntList P, int n) {
    if (P == null)
        return null;
    else {
        P.head += n;
        P.tail = dincrList(P.tail, n);
        return P;
    }
}
```

```
X: [5, 45, 58]
Q: [5, 45, 58]
L: [5, 45, 58]
P: [5, 45, 58]
```

Another Example: Non-destructive List Deletion

If L is the list [2, 1, 2, 9, 2], we want `removeAll(L, 2)` to be the new list [1, 9].

```java
/** The list resulting from removing all instances of X from L * non-destructively. */
static IntList removeAll(IntList L, int x) {
    if (L == null)
        return null;
    else if (L.head == x)
        return removeAll(L.tail, x);
    else
        return new IntList(L.head, removeAll(L.tail, x));
}
```

Aside: How to Write a Loop (in Theory)

- Try to give a description of how things look on any arbitrary iteration of the loop.
- This description is known as a loop invariant, because it is true from one iteration to the next.
- The loop body then must
  - Start from any situation consistent with the invariant;
  - Make progress in such a way as to make the invariant true again.

```java
while (condition) {
    // Invariant: true here
    // Loop body
    // Invariant: again true here
    // Invariant: true and condition false.
    // So if (Invariant and not condition) is enough to insure we've got the answer, we're done!
```

Last modified: Wed Sep 11 14:20:02 2013
Iterative Non-destructive List Deletion

Same as before, but use front-to-back iteration rather than recursion.

```java
/** The list resulting from removing all instances of X from L
 * non-destructively. */
static IntList removeAll(IntList L, int x) {
    IntList result, last;
    result = last = null;
    for (; L != null; L = L.tail) {
        if (x == L.head)
            continue;
        else if (last == null)
            result = last = new IntList(L.head, null);
        else
            last = last.tail = new IntList(L.head, null);
    }
    return result;
}
```

Here, \(I\) is the loop invariant:
Result is all elements of \(L_0\) not equal to \(x\) up to and not including \(L\), and last points to the last element of result, if any. We use \(L_0\) here to mean “the original sequence of int values in \(L\).”

Iterative Destructive Deletion

```java
/** The list resulting from removing all instances of X from L.
 * Original contents of L may be destroyed. */
static IntList dremoveAll(IntList L, int x) {
    IntList result, last;
    result = last = null;
    while (L != null) {
        IntList next = L.tail;
        if (x != L.head) {
            if (last == null)
                result = last = L;
            else
                last = last.tail = L;
            L.tail = null;
        }
        L = next;
    }
    return result;
}
```

Destructive Deletion

```java
/** The list resulting from removing all instances of X from L.
 * The original list may be destroyed. */
static IntList dremoveAll(IntList L, int x) {
    if (L == null)
        return null;
    else if (L.head == x)
        return dremoveAll(L.tail, x);
    else {
        L.tail = dremoveAll(L.tail, x);
        return L;
    }
}
```