CS61B Lecture #4: Simple Pointer Manipulation

Announcements:
• Discussion 114 (3-4 Thurs.) is now in 289 Cory (used to be 3111 Etch.)
• Next week and (maybe) from then on, discussion 114 will be in 3102 Etcheverry.

Public Service Announcement:
• Residential Computing, which provides tech support in residence halls, is currently hiring in programming, marketing, system administration, and more. Flexible hours and work study, $12.74 to $19.32 an hour. Applications due Wednesday, February 1st, 2006; see the notice on www.rescomp.berkeley.edu.

Today: More pointer hacking.

Destructive Incrementing

Destructive solutions may modify 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;
    }
}
```

```java
/** List L destructively incremented * by n. */
static IntList dincrList (IntList L, int n) {
    // 'for' can do more than count!
    for (IntList p = L; p != null; p = p.tail)
        p.head += n;
    return L;
}
```

X = IntList.list (3, 43, 56);
/* IntList.list from HW #1 */
Q = dincrList (X, 2);

Another Way to View Pointers

• Some folks find the idea of "copying an arrow" somewhat odd.
• Alternative view: think of a pointer as a label, like a street address.
• Each object has a permanent label on it, like the address plaque on a house.
• Then a variable containing a pointer is like a scrap of paper with a street address written on it.
• One view:

```
last: 
result: 5 45
```

• Alternative view:

```
last: #3
result: 5 #3 45
```

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));
}
```

CS61B: Lecture #4
**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 value of \( L \).”

**Iterative 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) {
    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;
}
```

**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!

**Destructive Deletion**

\[ \text{--- : Original} \quad \text{----- : after } Q = \text{dremoveAll}(Q,1) \]

```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;
    }
}
```

**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;
}
```