

CS61B Lecture #5: Simple Pointer Manipulation

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 home-page Announcements.

Last modified: Fri Aug 31 13:32:54 2012

CS61B: Lecture #5 1

Destructive Incrementing

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

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

Diagram illustrating the destructive incrementing process. The original list X (3, 43, 56) is modified to Q (5, 45, 58). The diagram shows pointers X, Q, L, and P. X and Q point to the first node (5, 45). L points to the second node (45, 58). P points to the third node (58, null).

```
/** 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;
}
```

Last modified: Fri Aug 31 13:32:54 2012

CS61B: Lecture #5 2

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].

```
/** 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));
}
```

Last modified: Fri Aug 31 13:32:54 2012

CS61B: Lecture #5 3

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.
- ```
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: Fri Aug 31 13:32:54 2012

CS61B: Lecture #5 4

## Iterative Non-destructive List Deletion

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

/\*\* 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) {

/\* L != null and  $\mathcal{I}$  is true. \*/

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);

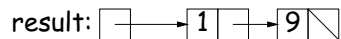
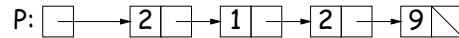
}

return result;

}

Here,  $\mathcal{I}$  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."



last: removeAll (P, 2)  
P does not change!

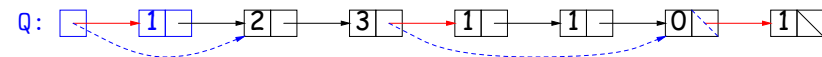
Last modified: Fri Aug 31 13:32:54 2012

CS61B: Lecture #5 5

## Destructive Deletion

→ : Original

----- : after Q = dremoveAll (Q,1)



/\*\* 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;

}

}

Last modified: Fri Aug 31 13:32:54 2012

CS61B: Lecture #5 6

## Iterative Destructive Deletion

/\*\* 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;

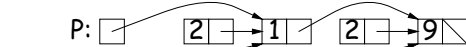
}

L = next;

}

return result;

}



next: P = dremoveAll (P, 2)

Last modified: Fri Aug 31 13:32:54 2012

CS61B: Lecture #5 7