CS61B Lecture 4

Wednesday, January 29, 2020

Announcements

- **Handing in labs and homework**: We'll be lenient about accepting late and homework and labs for lab1, lab2 and hw0. Just get it done. Do not submit via email.
- Course staff will interpret the absence of a central repository for you, or a lack of a lab1 submission, as your intention to drop the course.
- HW1 will be released tonight, and Project 1 will be released on Friday.

> **A Small Test of Understanding**
> In Java, the keyword `final` in a variable declaration means the variable's value may not be changed after the variable is initialized.
>
> Is the following class valid?
>
> ```
> public class Issue {
>     private final IntList aList = new IntList(0, null)
>
>     public void modify(int, k) {
>         this.aList.head = k;
>     }
> }
> ```
>
> Why or why not?
>
> **Answer:** Turns out this is valid. is valid. Although `modify` changes the `head` variable of the object pointed to by `aList`, it does not modify the contents of `aList` itself (which is a pointer).

Destructive Incrementing

In our previous version of `incrList`, the method we designed was non-destructive; that is to say the `IntList` we passed in to the function is not changed, and instead returns a new `IntList`.

Let's now build a destructive method called `dincrList`. **Destructive** solutions may modify objects in the original list to save time or space:

```
/** Destructively add N to L's items. Recursive solution. */
static IntList dincrList(IntList P, int n) {
    if (P == null) {
        return null;
    } else {
        P.head += n;
        P.tail = dincrList(P.tail, n);
        return P;
    }
}
```

```
/** Destructively add N to L's items. Iterative solution. */
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;
    }
}
```

List Deletion

Non-Destructive

Recursive

If L is the list `[2,1,2,9,2]`, we want `removeAll(L,2)` to be the new `[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;
    } else {
        return new IntList(L.head,, removeAll(L.tail, x));
    }
}
```

Iterative

Same as before, but using iteration (front-to-back) 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 = null;
last = result;
for ( ; L != null; L = L.tail) {
    if (x == L.head) {
        continue; // equivalent to pass in Python
    } else if (last == null) {
        result = new IntList(L.head, null);
        last = result;
    } else
        last.tail = new IntList(L.head, null);
        last = last.tail;
    }
```

```
return result;
}
```

> **Multiple Assignment in Java**
> Instead of writing:
>
> ```
> last.tail = ...;
> last = last.tail;
> ```
>
> You can also write:
>
> ```
> last = last.tail = ...;
> ```
>
> These two pieces of code are fundamentally identical: it tells Java to assign some value to the pointer at `last.tail`, then treats the assignment statement as an expression whose value is assigned the pointer at `last`.

Destructive

Same as before, but we will modify the original list that is passed in:

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

/** The list resulting from removing all Xs from L
 *  destructively. Iterative solution */
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;
}
```