

## CS61B Lecture #4: Lists and Arrays

- Discussion section 113 (W4-5) is now in 85 Evans.
- *Always* turn homework in and do labs, even if you don't completely get it!
- **Readings for Friday:** *PIJ* sections 1.11-1.12.
- **Today:** Still working on *PIJ* 1.1-1.10 and 4.

Last modified: Wed Sep 4 02:38:36 2002

CS61B: Lecture #4 1

## Another Example: Non-destructive List Deletion

**Problem:** Remove all instances of X from the list L, leaving remaining items in original order.

```
/** The list resulting from removing all instances of X from L
 * non-destructively. */
static IntList removeAll (IntList L, int x) {
    // What goes here?
```

```
}
```

Last modified: Wed Sep 4 02:38:36 2002

CS61B: Lecture #4 2

## Non-Destructive List Deletion II

**What are the cases?**

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

Last modified: Wed Sep 4 02:38:36 2002

CS61B: Lecture #4 3

## Non-destructive List Deletion Solved

```
/** 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: Wed Sep 4 02:38:36 2002

CS61B: Lecture #4 4

## Iterative Non-destructive List Deletion

**Problem:** Same as before, but use iteration rather than recursion.

```

/** The list resulting from removing all instances of X from L
 * non-destructively. */
static IntList removeAll (IntList L, int x) {
    // What goes here?
}

```

## 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.

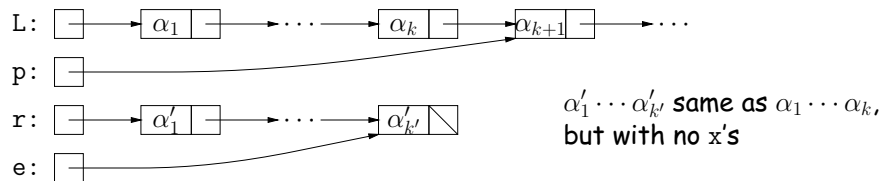
```

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!

## Possible Deletion Invariant

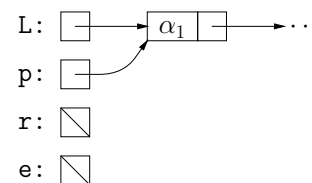
- Assume L points to the original list.
- Want r to point to result list.
- Let p point to the first item in L we haven't processed yet.
- Let e point to the last item of r (if any).



- If  $k'$  is 0, nothing (other than initial  $x$ 's) found,  $r$  and  $e$  null;
- If  $p$  is null, there's nothing more to process;
- If  $L$ ,  $p$ ,  $r$ , and  $e$  are all null,  $L$  is empty.

- **Question 1:** How do we get to this state in the first place?
- **Question 2:** How do we get back to this state (with  $k$  larger)?

## Question 1: Getting to the Invariant

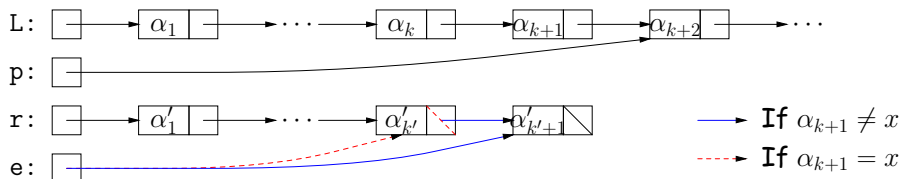


```

IntList p, r, e;
p = L;
r = e = null;

```

## Question 2: Getting Back to the Invariant

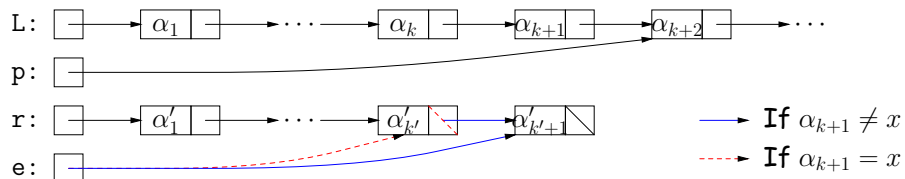


- How do we do that?
- (Caution: p and e can be null)

Last modified: Wed Sep 4 02:38:36 2002

CS61B: Lecture #4 9

## Getting Back to the Invariant: the Code



```

if (p.head != x) {
  if (r == null) // First non-deleted item
    r = e = new IntList (p.head, null);
  else { // Succeeding non-deleted items
    e.tail = new IntList (p.head, null);
    e = e.tail; // e = e.tail = new IntList(...); also OK
  }
}
p = p.tail;

```

Last modified: Wed Sep 4 02:38:36 2002

CS61B: Lecture #4 10

## Putting It All Together

- Stop when no more to do: p is null.

```

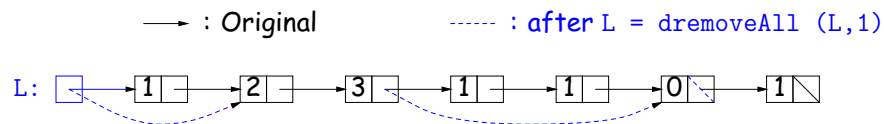
IntList p, r, e;
p = L;
r = e = null;
while (p != null) {
  if (p.head != x) {
    if (r == null)
      r = e = new IntList (p.head, null);
    else
      e = e.tail = new IntList (p.head, null);
  }
  p = p.tail;
}
return r;

```

Last modified: Wed Sep 4 02:38:36 2002

CS61B: Lecture #4 11

## Destructive Deletion



```

/** The list resulting from removing all instances of X from L.
 * The original list may be destroyed. */
static IntList dremoveAll (IntList L, int x) {
  // What goes here? (Recursively)

```

```

}

```

Last modified: Wed Sep 4 02:38:36 2002

CS61B: Lecture #4 12

## A Recursive, Destructive Delete

```
/** 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;
  if (L.head == x)
    return dremoveAll (L.tail, x);
  else {
    L.tail = dremoveAll (L.tail, x);
    return L;
  }
}
```

- Notice the similarity to the recursive `removeAll`.
- **Challenge for the reader:** How would you do this iteratively?

Last modified: Wed Sep 4 02:38:36 2002

CS61B: Lecture #4 13

## New Topic: Arrays

- An array is structured container whose components are
  - **length**, a fixed integer.
  - a sequence of **length** simple containers of the same type, numbered from 0.
  - (.length field usually implicit in diagrams.)
- Arrays anonymous, like other structured containers.
- Always referred to with pointers.
- For array pointed to by A,
  - Length is `A.length`
  - Numbered component *i* is `A[i]` (*i* is the *index*)
  - Important feature: index can be *any integer expression*.

Last modified: Wed Sep 4 02:38:36 2002

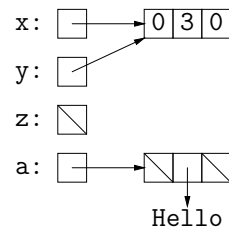
CS61B: Lecture #4 14

## A Few Samples

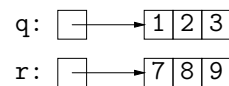
### Java

```
int[] x, y, z;
String[] a;
x = new int[3];
y = x;
a = new String[3];
x[1] = 2;
y[1] = 3;
a[1] = "Hello";
```

### Results



```
int[] q;  
q = new int[] { 1, 2, 3 };  
int[] r =  
  { 7, 8, 9 };
```



Last modified: Wed Sep 4 02:38:36 2002

CS61B: Lecture #4 15

## Example: Accumulate Values

**Problem:** Sum up the elements of array A.

```
static int sum (int[] A) {  
  int N;  
  N = 0;  
  for (int i = 0; i < A.length; i += 1)  
    N += A[i];  
  return N;  
}
```

*// For the hard-core: could have written*

```
int N, i;  
for (i=0, N=0; i<A.length; N += A[i], i += 1)  
  { } // or just ;
```

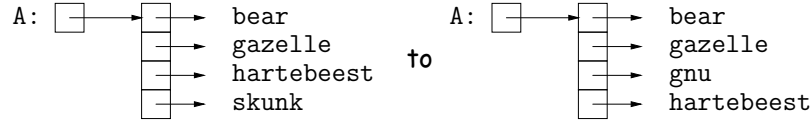
*// But don't: it's obscure.*

Last modified: Wed Sep 4 02:38:36 2002

CS61B: Lecture #4 16

## Example: Insert into an Array

**Problem:** Want a call like `insert (A, 2, "gnu")` to convert (destructively)



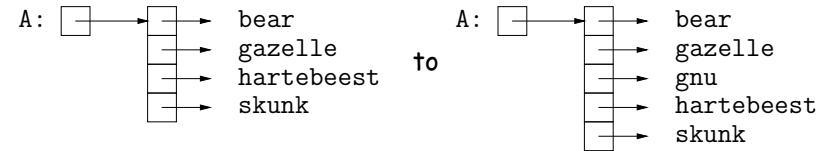
```
/** Insert X at location K in ARR, moving items
 * K, K+1, ... to locations K+1, K+2, ...
 * The last item in ARR is lost. */
static void insert (String[] arr, int k, String x) {
    for (int i = arr.length-1; i > k; i -= 1) // Why backwards?
        arr[i] = arr[i-1];
    // Alternative:
    // System.arraycopy (arr, k, arr, k+1, arr.length-k-1);
    //                    from      to      # to copy
    arr[k] = x;
}
```

Last modified: Wed Sep 4 02:38:36 2002

CS61B: Lecture #4 17

## Growing an Array

**Problem:** Suppose that we want to change the description above, so that `A = insert2 (A, 2, "gnu")` does *not* shove "skunk" off the end, but instead "grows" the array.



```
/** Return array containing same items as ARR, but with X
 * inserted at location K, and moving items K, K+1, ...
 * moved to locations K+1, K+2, ... 0<=K<=ARR.length */
static String[] insert2 (String[] arr, int k, String x) {
}
}
```

- Why do we need a different return type from `insert`??

Last modified: Wed Sep 4 02:38:36 2002

CS61B: Lecture #4 18

## Growing an Array: The Program

```
/** Return array containing same items as A, but with X
 * inserted at location K, and moving items K, K+1, ...
 * moved to locations K+1, K+2, .... */
static String[] insert2 (String[] arr, int k, String x) {
{
    String[] result = new String[arr.length + 1];
    System.arraycopy (arr, 0, result, 0, k);
    System.arraycopy (arr, k, result, k+1, arr.length-k);
    result[k] = x;
    return result;
}
```

Last modified: Wed Sep 4 02:38:36 2002

CS61B: Lecture #4 19