

**Lab Exercises.** (Due: Tues., 18 September 2001 in lab) Copy the directory `$master/hw/lab2` to a directory of your own, using commands such as

```
mkdir lab2
cp $master/hw/lab2/* lab2
```

or simply

```
cp -r $master/hw/lab2 .
```

(Sometimes we are tardy getting these directories set up. If you are working on this before lab and get messages such as “Permission denied” or “cannot access,” it means we’re not quite ready yet, or you misspelled something.)

In this directory, you should find a file called `README`, with directions about what to do during the scheduled lab. We strongly suggest that you read over this file *before* going to your lab.

You are intended to finish the lab exercises *in lab* and have your TA check them off. You can have any TA in any lab section check off your lab. As you can see, you have 11 days to complete the lab.

**Homework Exercises.** (Due: Fri., 14 September 2001 at midnight) Create a directory to hold your answers to this homework set. Copy the files from `$master/hw/hw2` into this directory. Use the command `submit hw2` to submit your solutions to the problems below.

1. Complete the following Java functions so that they perform as indicated in their comments. The files `IntList.java` and `IntList2.java` in `~cs61b/hw/hw2` contain the declarations of the classes `IntList` and `IntList2`. Put your answers in a file called `Lists.java`, using the template in `~cs61b/hw/hw2`. You may find the functions in the file `~cs61b/hw/hw2/ListUtil.java` to be useful for testing.

```
/** The list of lists formed by breaking up L into "natural runs":
 * that is, maximal ascending sublists, in the same order as
 * the original. For example, if L is (1, 3, 7, 5, 4, 6, 9, 10),
 * then result is the three-item list ((1, 3, 7), (5), (4, 6, 9, 10)).
 * Destructive: creates no new IntList items, and may modify the
 * original list pointed to by L. */
static IntList2 naturalRuns (IntList L) {
    /* *Fill in here* */
}
```

2. Complete the following Java functions so that they perform as indicated in their comments. Remember that some arrays can have zero elements. Put your answers to this problem (all parts) in a file named `Arrays.java` (copy the template in `~cs61b/hw/hw2/Arrays.java`). You may find the contents of the file `~cs61b/hw/hw2/ArrayUtil.java` useful in testing your answers.

a. `/** A new array consisting of the elements of A followed by the  
* the elements of B. */  
static int[] catenate(int[] A, int[] B) {  
 /* *Fill in here* */  
}`

b. `/** The array formed by removing LEN items from A,  
* beginning with item #START (counts from 0). */  
static int[] remove (int[] A, int start, int len) {  
 /* *Fill in here* */  
}`

c. `/** The array of arrays formed by breaking up A into  
* maximal ascending lists, without reordering.  
* For example, if A is {1, 3, 7, 5, 4, 6, 9, 10}, then  
* returns the three-element array  
* {{1, 3, 7}, {5}, {4, 6, 9, 10}}. */  
static int[][] naturalRuns (int[] A) {  
 /* *Fill in here* */  
}`

3. Define a class that allows the following operations:

```
// Q is an empty collection of Strings.
OrderedStrings Q = new OrderedStrings();
Q.add ("zebra");
Q.add ("dog");
Q.add ("aardvark");
Q.add ("gnu");
// Print Q in order.
while (! Q.isEmpty ())
    System.out.println (Q.removeFirst ());
```

That is, an `OrderedStrings` is a collection of `Strings` to which one can add new strings (with `add`), remove and return the smallest (first) remaining string with `removeFirst`, and test for emptiness with `isEmpty`. Aside from `String`, don't use any existing library classes in the implementation, except for testing purposes. You can define additional classes if you find it useful. Put your solution in file `OrderedStrings.java`.