Slide 1

Administrivia

- Reminder: Midterm Tuesday. Review sheet on the Web. Also quiz solutions and sample solutions to homeworks (Homework 3 solution soon).

Slide 2

About the Midterm

- Review class notes, homeworks, readings. If I didn't mention it in class, odds are I won't ask about it on the exam.
- Questions will be a mix of problems similar to those in quizzes, plus possibly some true/false, multiple choice, or short-answer.
- Open book, open notes. Okay to use a computer to review book, your notes and graded work, and the course Web site, but nothing else. No using the Scala interpreter/compiler to answer questions about programs, sorry.
- (Topic by topic through the review sheet.)

Slide 3

Recursion — Review/Recap

- A function (or definition) is recursive if it calls/uses itself. Obviously(?) there needs to be at least one base case too.
- Can be somewhat tricky to think about whether/how recursive functions work — it involves nested calls to the same function, one “inside” the other in some sense. May be helpful to take what I call a “static” perspective, focusing on the code and one call to the function rather than the whole bunch of nested calls.
- To do that, first be clear on what the function does — “computes n factorial”, or “computes the sum of array elements starting at this index”. Then ask . . .

Slide 4

Recursive Functions — “(How) Does it Work”?

- (How) does it work for the base case(s)?
- (How) does it work for the non-base cases, *assuming that the recursive calls work*, meaning that they do what the function is supposed to do, based on the definition you came up with.
- (How) does each recursive call get us closer to a base case?
- (In some ways this is a mirror image of induction, as in proofs by induction, where we start with small cases and construct more complex ones.)

Arrays and Lists — Review/Recap

Slide 5

- Scala provides two basic types of “sequences”, arrays and lists.
- Several ways to work with them. We start out by applying tools we already have (recursive functions), partly to get more practice with them. Also an opportunity to revisit “higher-order functions” (functions that use other functions as parameters) . . .

Higher-Order Functions — Review/Recap

Slide 6

- “Higher-order functions” (first discussed in chapter 5) are functions that use other functions as parameters (or as return values). Very useful concept, supported in fairly different ways in different languages.
- As an example of how this is useful — summing all elements of an array versus computing their product, versus finding the smallest or largest element, etc. Basic computation (a *reduction*) involves combining elements pairwise with a binary operator, and by using a higher-order functions we don’t have to repeat the parts that are the same.

Defining Higher-Order Functions in Scala

Slide 7

- Syntax illustrated by our example from class:

```
def arrayCombine(a : Array[Int], startIndex : Int,
  combine : (Int, Int) => Int, identity : Int) : Int = { /* .... */ }
```

  where `combine` is a parameter that is itself a function(!).

  (I could have put all of that on one line, but it would have been long.)

- Within the body of the function (`arrayCombine` in the example) we can call the parameter function (`combine`) as we usually do, e.g., `combine(1, 2)` to call the function with parameters 1 and 2.

Using Higher-Order Functions in Scala

Slide 8

- One option for function parameters is a named function:

```
def add(x : Int, y : Int) : Int = { x + y }
arrayCombine(a, 0, add, 0)
```

- Another option is a function literal:

```
arrayCombine(a, 0, (x, y) => ( x + y ), 0)
```

- Yet another option is a special form of a function literal:

```
arrayCombine(a, 0, _ + _, 0)
```

Example(s) Revisited

- We could now revise our array demo program to do still more things with the array — find minimum and maximum elements, for example. (Not done in class.)
- We could add similar functionality to our list demo program.

Minute Essay

- About how much time a week are you spending on this class outside of class? How much of it involves actually programming, or at least trying things in the REPL? (Keep in mind the minute-essay mantra — “no wrong answers”.)