Slide 1

Administrivia

- Homework 1 and Homework 2 sample solutions on Web. (These are meant to be reasonable/good solutions, but not the only possible ones. As long as your code works you should get full credit.)
- Homework 3 will be on the Web tomorrow. (I will send mail.) Due a week from today.
- ACM is doing free tutoring in HAS 329, M/T/W/R 3:30pm–5pm.

Slide 2

Minute Essay From Last Lecture

- Most people found Homework 2 doable though maybe more time-consuming than they thought. (Previous experience with programming did seem to help — no surprise!)

  A comment I appreciated: “The scenarios were easy to understand until I tried to program them . . . ”
- One person commented on needing 5.0 rather than 5 in the temperature conversion program. (Why?)

Slide 3

Conditional Execution — A Bit More

- Notice that this

```
if (x < y) println("this")
if (x >= y) println("that")
```

is the same as

```
if (x < y) println("this")
else println("that")
```

(I.e., in the "else" part of an if/else you know the condition is false. Testing again makes the program longer and more error-prone.)

- (Finish/review bounding-box example from last time. Notice again the use of tuples and "pattern-matching".)

Slide 4

Functions — Review/Recap

- Functions are most useful for two things — decomposing problems into manageable pieces, and avoiding duplicating code.
- But they also provide one way to get something we don't have yet, namely repetition/iteration . . .

Slide 5

Repetition and Recursion

- Having if/else allows us to do a lot of things we couldn't do before, but there are still things we can't do easily, mostly involving some sort of repetition. Simple example — adding something to the grade program that would prompt for six quiz scores. Another example might be trying to use our bounding-box function to find a bounding box to enclose more than two rectangles, with the choice of how many up to the user.
- Scala provides many ways to do this. We will look at recursion first . . .

Slide 6

Recursion

- Basic idea of recursion is to solve a problem by defining
  - "base cases" we can easily, and
  - a way of reducing other cases to "smaller" instances of the problem
- Simple examples abound in math; a traditional first example is computing the factorial of an integer. We can define $n!$ as the product of the integers from 1 through n, or we can use a recursive definition:

$$n! = \begin{cases} n \cdot (n-1)! & \textit{if } n > 1 \\ 1 & \textit{otherwise} \end{cases}$$

  This is easy to convert into code in a language that supports recursion . . .

Slide 7

Recursion, Continued

- Key ideas in recursion:
  - One or more base cases that can be solved without recursion.
  - A way of splitting up other cases into one or more *smaller* recursive calls plus some other logic.
- Very important that recursive calls be somehow smaller, so that you eventually reach a base case!
- As one more example for now — function to “count down” (print numbers from starting point through 1).

Slide 8

Minute Essay

- None — quiz.