Administrivia

• Reminder: Homework 3 due today.

If you can't finish completely by the due date/time, but you have something that represents at least a good start, *send me what you have* and submit a revised/improved version as soon as you can. You lose fewer points that way, I think you learn more, and I'd rather grade code that works than code that doesn't!

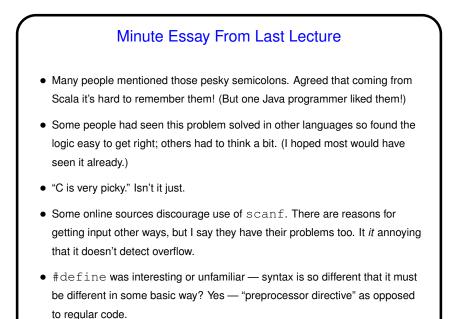
- Sample solution to Homework 2 posted (also Homework 1, though it's kind of trivial), linked from bottom of "lecture topics and assignments" page.
- Homework 4 on the Web; due in two weeks. Both problems should be doable with material covered through today, but I plan to talk more about library function rand (), used in the second problem, next time.

More Administrivia

• If you wonder about the fact that the e-mail address TMail shows for me isn't the same as the one I give in my course materials — it's a long story, but you can use either one.

Also, no need to put your name in subject lines when turning in homework or minute essays — enough to say which course and which assignment (e.g, "hw1" or "minute essay").

• For what it's worth: Some of you give your programs names that include your name. If it helps you, go ahead, but I grade each person's work in a separate directory, so there should be no risk of confusion. (Over the years I've worked up a semi-automated system for grading programming assignments. More about it on request?)



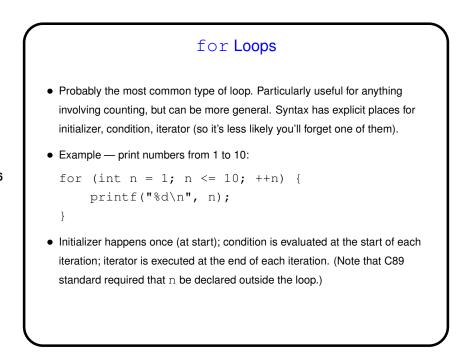
Repetition — Loops
C, like most/many procedural languages, offers several syntaxes for repetition. Recursion (discussed already) is one, but often not the most straightforward.
All have some way of expressing common elements (explicitly, rather than the "do for all" syntax allowed by some languages): *Initializer* (as its name suggests). *Condition* (determines whether repetition continues). *Body* (code to repeat). *Iterator* (something that moves on to next iteration).

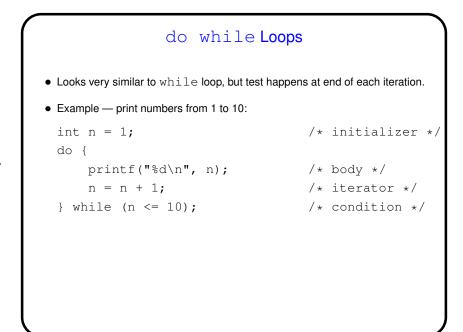
Worth noting that C, being fairly minimalist, doesn't offer some of the nice features for repetition Scala does.

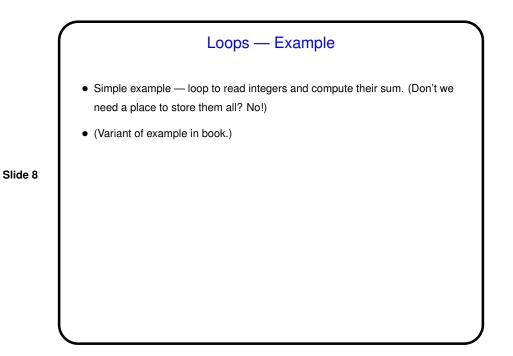
Slide 3

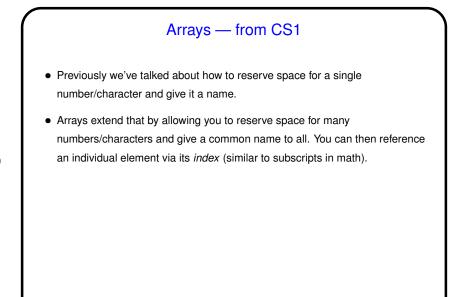
while Loops • Probably the simplest kind of loop. You decide where to put initializer and iterator. Test happens at start of each iteration. • Example — print numbers from 1 to 10: int n = 1;/* initializer */ while (n <= 10) { /* condition */ printf("%d\n", n); /* body */ /* iterator */ n = n + 1; } • Various short ways to write n = n + 1: n += 1; n++; ++n; What do you think happens if we leave out this line?

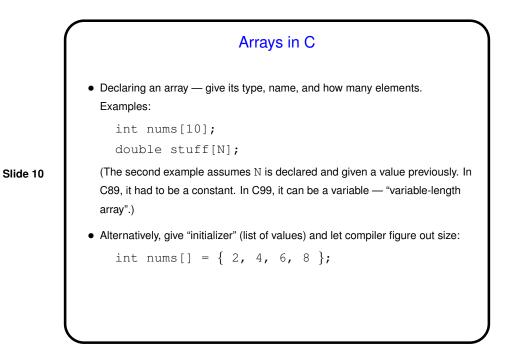
Slide 5

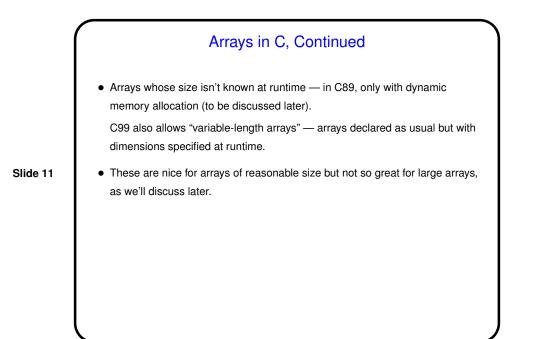


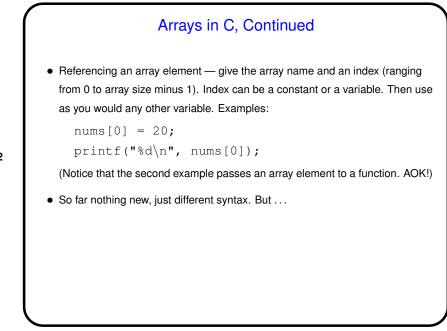


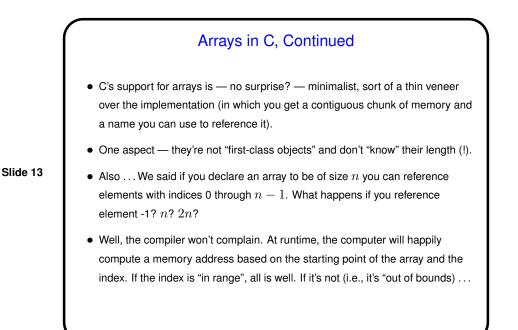












Arrays in C, Continued (What happens if you try to access an array with an index that's out of bounds?) "Results are unpredictable" ("undefined behavior" in C-speak). Maybe it's outside the memory your program can access, in which case you may get the infamous "Segmentation fault" error message (or with newer compilers you may get a screenful of equally cryptic messages). Almost worse is if it's not — then what's at the computed memory address might be some other variable in your program, which will then be accessed/changed. This is the essence of the *buffer overflows* you hear mentioned in connection with security problems. Why this behavior? Well, C was designed to compile to efficient code, and checking indices "costs". If you want it, put it in! (And very often you should.)

