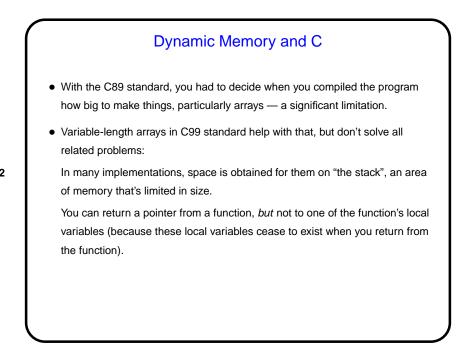
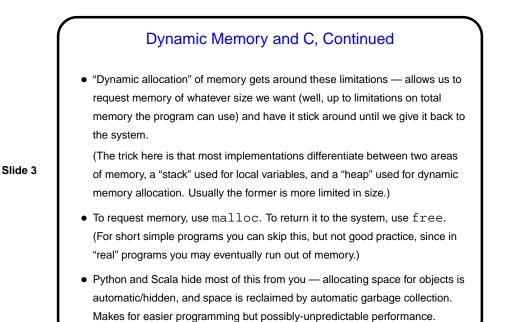


• Homework 5 on the Web. Nominally due next Monday (but deadline may be extended).

Slide 1





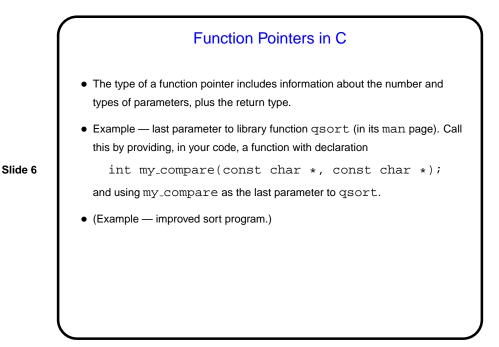
## Dynamic Memory and C, Continued • Examples: int \* nums = malloc(sizeof(int) \* 100); char \* some\_text = malloc(sizeof(char) \* 20); free(nums); • Some books/resources recommend "casting" value returned by malloc. Other references recommend the opposite! But you should check the value — if NULL, system was not able to get that much memory. • (Example — improved sort program.)

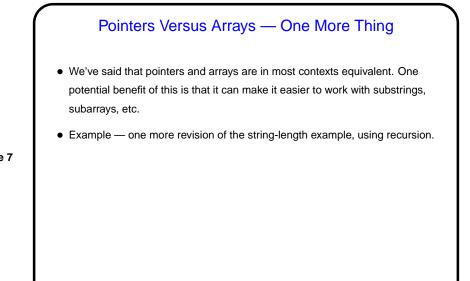
## **Function Pointers**

• You know from more-abstract languages that there are situations in which it's useful to have method parameters that are essentially code. Some languages make that easy (functions are "first-class objects") and others don't, but almost all of them provide some way to do it, since it's so useful — e.g., providing a "less-than" function for a generic sort.

Slide 5

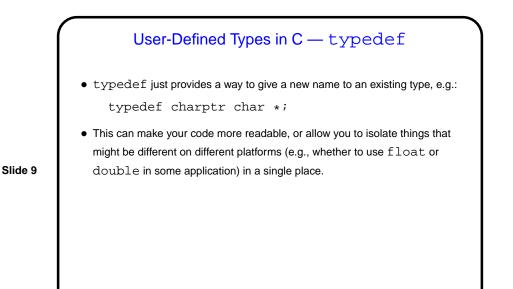
• In C, you do this by explicitly passing a pointer to the function.



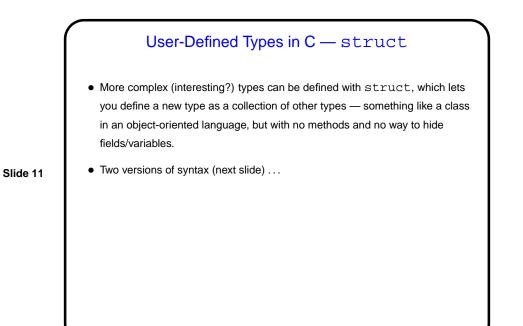


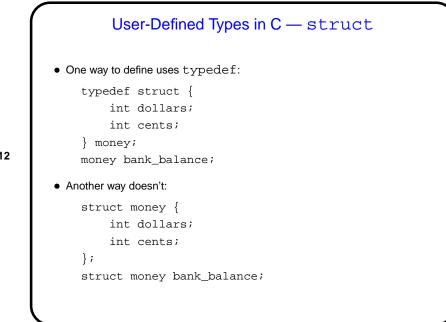
Slide 8

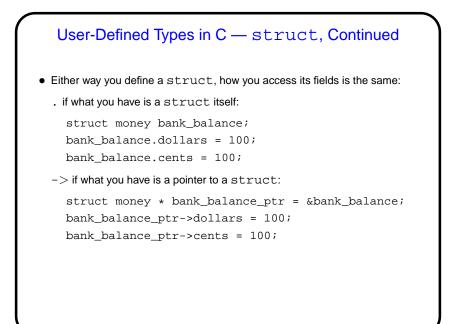
## User-Defined Types So far we've only talked about representing very simple types — numbers, characters, text strings, arrays, and pointers. You might ask whether there are ways to represent more complex objects, such as one can do with classes in object-oriented languages. The answer is "yes, sort of" — C doesn't provide nearly as much syntactic help with object-oriented programming, but you can get something of the same effect. But first, some simpler user-defined types ...

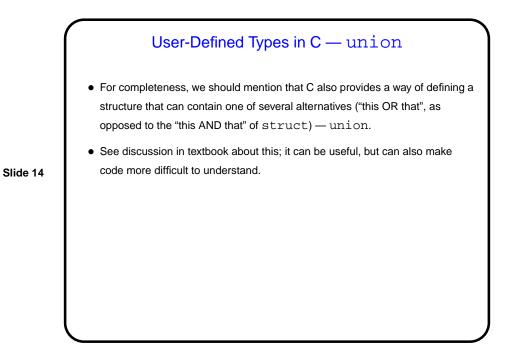


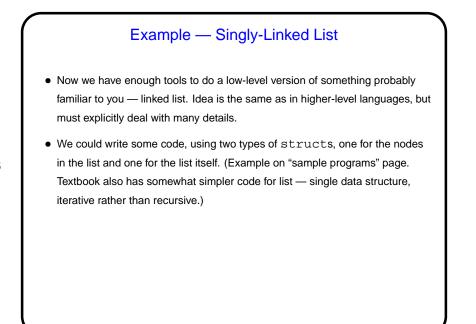
User-Defined Types in C — enum
In C (and in some other programming languages) an *enumeration* or an *enumerated type* is just a way of specifying a small range of values, e.g. enum basic\_color { red, green, blue, yellow }; enum basic\_color color = red;
Slide 10
Slide 10
Under the hood, C enumerated types are really just integers, though, and they can be ugly to work with in some ways (e.g., no nice way to do I/O with them).











## Separate Compilation and Makefiles - Review

 C (like many languages) lets you split large programs into multiple source-code files. Typical to put function and other declarations in files ending . h, function definition in files ending . c. Compilation process can be separated into "compile" (convert source to object code) and "link" (combine object and library code to make executable) steps.

Slide 16

• UNIX utility make can help manage compilation process. Can also be useful as a convenient way to always compile with preferred options.

