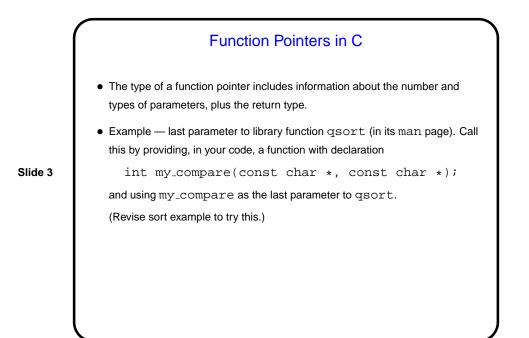
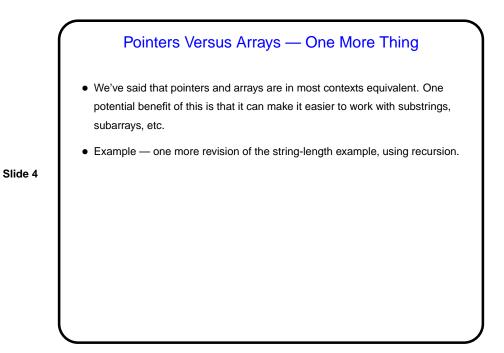


Slide 1

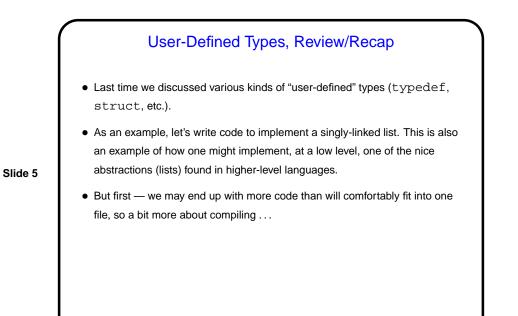
Function Pointers You know from more-abstract languages that there are situations in which it's useful to have method parameters that are essentially code. Some languages make that easy (functions are "first-class objects") and others don't, but almost all of them provide some way to do it, since it's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

Slide 2





Slide 6



A Little More About gcc • Many, many compiler options for gcc. One of the most useful is -Wall. • To automate using them every time, you can use the UNIX utility make ...



 Motivation: Most programming languages allow you to compile programs in pieces ("separate compilation"). This makes sense when working on a large program — when you change something, just recompile parts that are affected.

Slide 7

Slide 8

 Idea behind make — have computer figure out what needs to be recompiled and issue right commands to recompile it.

Makefiles First step in using make is to set up "makefile" describing how files that make up your program (source, object, executable, etc.) depend on each other and how to update the ones that are generated from others. Normally call this file Makefile or makefile. Simple example (assuming main.c #includes defs.h and foo.h): main: main.o foo.o gcc -o main main.o foo.o main.c: main.c defs.h foo.h gcc -c main.c foo.o: foo.c gcc -c foo.c When you type make, make figures out (based on files' timestamps) which files need to be recreated and how to recreate them.

