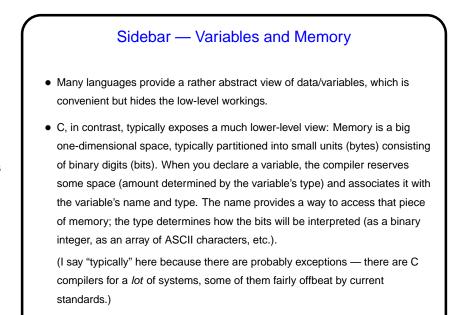


Slide 2



Slide 3

Pointers in C — Review/Recap

• C, in contrast to Python and Scala, makes an explicit distinction between things and pointers-to-things. Pointers-to-things are essentially memory addresses, though usually declared to point to variables (or data) of a particular type. (Exception: a "void" pointer can point to data of any type. See man page for memcpy for example.)

Slide 4

- Two useful operators are & ("address of") and * (dereference).
- In C, pointers and arrays are in some sense(s) equivalent not identical, but in many contexts interchangeable. This is reflected in the man pages for many functions (e.g., printf).

Parameter Passing in C

 In C, all function parameters are passed "by value" — which means that the value provided by the caller is copied to a local storage area in the called function. The called function can change its copy, but changes aren't passed back to the caller.

Slide 5

- An apparent exception is arrays no copying is done, and if you pass an array to a function the function can change its contents (as we did in the sort program). Why "apparent exception"? because really what's being passed to the function is not the array but a pointer! so the copying produces a second pointer to the same actual data.
- This is at least simple and consistent, but has annoying limitations ...

Pass By Reference (Sort Of)

 A significant potential limitation on functions is that a function can only return a single value. Pointers provide a way to get around this restriction: By passing a pointer to something, rather than the thing itself, we can in effect have a function return multiple things.

Slide 6

- To make this work, typically you declare the function's parameters as pointers, and pass addresses of variables rather than variables.
- The "sort of" of the title means that this isn't true pass by reference, as it exists in some other languages such as C++, but it can be used to more or less get the same effect.

(Example.)

