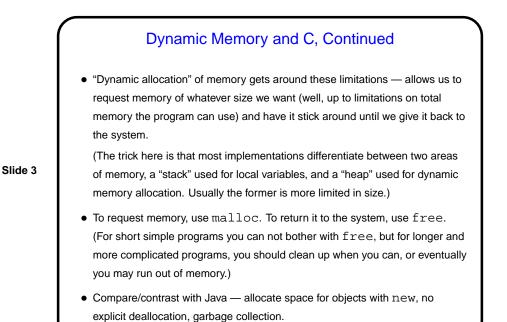
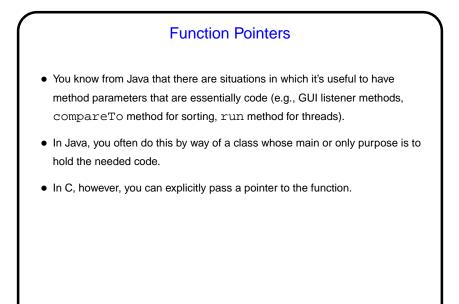


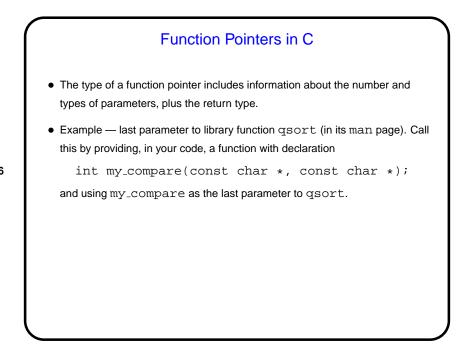
Dynamic Memory and C
With the C89 standard, you had to decide when you compiled the program how big to make things, particularly arrays — a significant limitation.
Variable-length arrays in C99 standard help with that, but don't solve all related problems:
In many implementations, space is obtained for them on "the stack", an area of memory that's limited in size.
You can return a pointer from a function, *but* not to one of the function's local variables (because these local variables cease to exist when you return from the function).

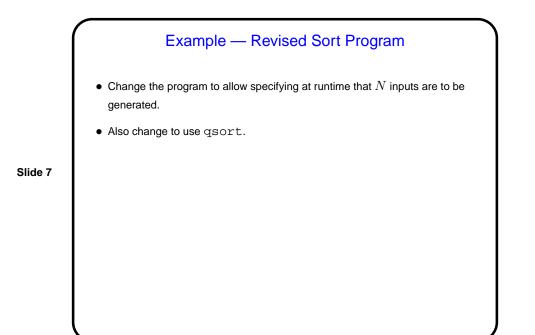


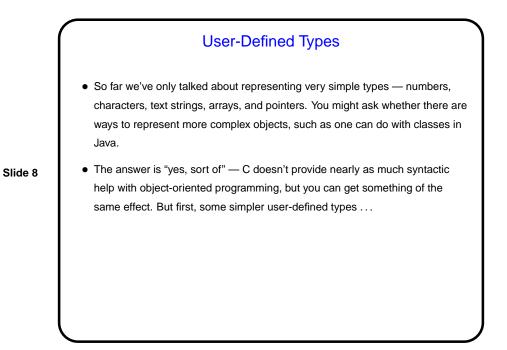
Dynamic Memory and C, Continued • Examples: int * nums = malloc(sizeof(int) * 100); char * some_text = malloc(sizeof(char) * 20); free(nums); • Some books/resources recommend "casting" value returned by malloc. Other references recommend the opposite! But you should check the value — if NULL, system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

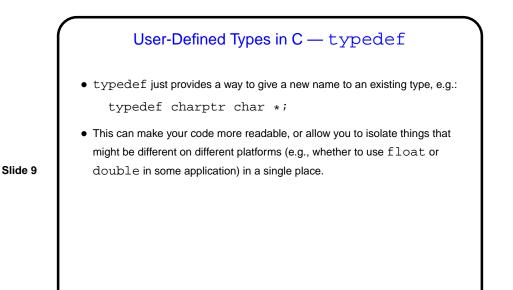


Slide 5









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;
This can make code more readable, and sometimes combines nicely with switch constructs.
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).

