Slide 1

Administrivia

- Reminder: Homework 5 due Monday.
- Homework 6 on the Web. Due 10/26. This one is not trivial but I think gives you useful practice working with pointers.

Slide 2

Minute Essay From Last Lecture

- Responses (to question about readings) were quite varied! a few people had bought (or rented) a copy of the textbook and liked it, a few people were using the online tutorial and liked that, and a few people weren't really using either one and found that to be okay.

Slide 3

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, added 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).

Slide 4

Dynamic Memory and C, Continued

- "Dynamic allocation" of memory gets around these limitations — allows us to request memory of whatever size we want (well, up to limitations on total memory the program can use) and have it stick around until we give it back to the system.

  (The trick here is that most implementations differentiate between two areas of memory, a "stack" used for local variables, and a "heap" used for dynamic memory allocation. Usually the former is more limited in size.)
- To request memory, use `malloc`. To return it to the system, use `free`. (For short simple programs you can skip this, but not good practice, since in "real" programs you may eventually run out of memory.)
- Python and Scala hide most of this from you — allocating space for objects is automatic/hidden, and space is reclaimed by automatic garbage collection.

Slide 5

Dynamic Memory and C, Continued

- Simple examples:

```
int * nums = malloc(sizeof(int) * 100);
char * some_text = malloc(sizeof(char) *
20);
free(nums);
```

  though it's better style/practice to write

```
int * nums = malloc(sizeof(*nums) * 100);
char * some_text = malloc(sizeof(*some_text)
* 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 6

- (Example — "improved" sort program.)

Slide 7

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 8

Function Pointers in C

- The type of a function pointer includes information about the number and types of parameters, plus the return type.
- Example — last parameter to library function `qsort` (in its `man` page). Call this by providing, in your code, a function with declaration

```
int my_compare(const void *, const void *);
```

  and using `my_compare` as the last parameter to `qsort`.
- (Example — "improved" sort program.)

Slide 9

C's Variable Types, Revisited

- I've said in class that the C standard isn't specific about some things (e.g., exact range of `int` data type). Sample program `sizes.c` illustrates that. Almost all of our public machines are 64-bit, but we have one 32-bit machine left.
- Sample program using `sizeof` operator (yes it is an operator) gives different results . . .

Slide 10

This and That

- This: Return value from `main` is an `int`; convention is that 0 means success and anything else means failure. To help you remember, and also to help return something appropriate on failure, `stdlib.h` defines `EXIT_SUCCESS` and `EXIT_FAILURE`. Good to use them.
- That: You (probably? maybe?) know about `diff` to compare contents of two files. What you might not know about is `vimdiff`, which shows files side by side (or one above the other with `-o`) using colors to highlight differences. (The default color scheme isn't the best for this. Change with `:colorscheme`. Type that and a space and press "tab" repeatedly to cycle through options.)

Slide 11

Minute Essay

- Many current high-level languages manage memory for you, including garbage collection. What advantages do you think this has? What disadvantages? (Both as compared to doing it yourself, as you do in C.)

Slide 12

Minute Essay Answer

- Advantages: easier, less error-prone.
- Disadvantages: less control, possibly unpredictable performance (which in some contexts matters).