Slide 1

Administrivia

- Reminder: Homework 4 due next Monday.

Slide 2

I/O in C — Recap

- `getchar` and `putchar` read/write one character at a time.
- `scanf` and `printf` read/write other data types (converting from/to printable form).
- Some environments allow reading/writing files via "I/O redirection". But that's somewhat restrictive . . .

Slide 3

Streams

- C's notion of file I/O is based on the notion of a *stream* — a sequence of characters/bytes. Streams can be *text* (characters arranged into lines separated by something platform-dependent) or *binary* (any kind of bytes). UNIX/Linux doesn't make a distinction, but other operating systems do.
- An input stream is a sequence of characters/bytes coming into your program (think of characters being typed at the console).
- An output stream is a sequence of characters/bytes produced by your program (think of characters being printed to the screen, including special characters such as the one for going to the next line).

Slide 4

Streams in C

- In C, streams are represented by the type `FILE *` — i.e., a pointer to a `FILE`, which is something defined in `stdio.h`.
- A few streams are predefined — `stdin` for standard input, `stdout` for standard output, `stderr`) for standard error (also output, but distinct from `stdout` so you can separate normal output from error messages if you want to).
- To create other streams — next slide.

Slide 5

Creating Streams in C

- To create a stream connected with a file — `fopen`.
- Parameters, from its `man` page:
  - First parameter is the name of the file (for now, text in double quotes).
  - Second parameter is how we want to access the file – read or write, overwrite or append — plus a `b` for binary files.
  - Return value is a `FILE *` — a somewhat mysterious thing, but one we can pass to other functions. If `NULL`, the open did not succeed. (Can you think of reasons this might happen?)

Slide 6

Working With Streams in C

- To read from an input stream — `fscanf`, almost identical to `scanf`. To write to an output stream — `fprintf`, almost identical to `printf`. `fgetc` and `fputc` may also be useful.
- When done with a stream, `fclose` to tidy up. (Particularly important for output files, which otherwise may not be completely written out.)

Slide 7

Reading Text Strings

- Getting text-string input is surprisingly tricky. `scanf` (or `fscanf`) seems like an obvious choice, but:
  - it can't read a string that includes blanks, and
  - it has no nice way to limit the number of characters read to the size of the array being read into.

  .
- Getting a whole line is probably better. `gets()` is an obvious/simple choice for reading from standard input, but it also has no way to limit how much is read. `fgets()` is better. (Look at its `man` page.)

  (Also notice `puts()` — simple way to write out a text string.)

Slide 8

Minute Essay

- None — sign in.