Slide 1

Administrivia

- Reminder: Homework 5 due today. (Sketch out a solution in class for second problem.)

Slide 2

Input/Output Redirection in UNIX/Linux

- We talk about `scanf` reading input “from the user” or “from the keyboard”, and `printf` printing “to the screen”.
- But that’s not quite right — really, `scanf` reads from *standard input*, and `printf` writes to *standard output*.
- What’s the difference? can *redirect* standard input/output to use (text) files instead. Example:

  ```
  myprogram < test1-in > test1-out
  ```

  to have `myprogram` get its input from `test1-in` rather than the keyboard, and put its output in `test1-out` rather than showing it on the screen. (Overwrites `test1-out`. To append instead, use `>> test1-out`.)

Slide 3

Files and C

- Why files? You probably already know: Things stored in memory vanish when you turn the computer off; to preserve them, usually save them as *files*.
- We know one way for a C program to get its input from a file, or write its output to a file — I/O (input/output) redirection. But this makes it difficult or impossible to also get input from the keyboard, write output to the screen.
- So C (like many other programming languages) provides ways to work more generally with files.

Slide 4

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 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 5

Streams in C

- In C, streams are represented by the type `FILE *`. `FILE` is something defined in `stdio.h`. The `*` means pointer (which we'll talk about later).
- 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 6

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 7

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.)
- Examples next time.

Slide 8

Minute Essay

- None — quiz.