Slide 1

Administrivia

- Homework 6 on the Web. Due a week from Monday.

Slide 2

Files in C — Recap/Review

- Files in C based on notion of “streams” (input and output). Sequence of bytes coming in or going out.
- Predefined streams `stdin`, `stdout`, `stderr`. Together with I/O redirection (in shell) these give you a crude way to work with files.
- Or you can create your own streams, connected to files, with `fopen`.
- Read and write with `fgetc`, `fputc`, `fscanf`, `fprintf`. All very much like functions you already know, but with one more parameter (stream).

Slide 3

Files, Continued

- (Review examples from last time.)
- How to get filenames? could prompt user and read in text, but in my opinion there's no really good way to do that in C, so what I prefer instead . . .

Slide 4

Another Way to Get Input — Command-Line Arguments

- (We can't completely discuss this until a bit later, but it's so useful for working with files that we'll do just a bit now.)
- You may have observed that most of the commands you use don't prompt you for input, but instead decide what to do based on what you type on the command line after the command name? so the program must be getting that information somehow, but — how? "command-line arguments" (e.g., for the command `gcc -Wall hello.c` there are two command-line arguments).

  (And those commands? Many of them are C programs!)
- Most programming languages provide a way to access this text, often via some sort of argument to the main function/method.

Command-Line Arguments in C

- In C, command-line arguments are passed to `main` as an array of text strings (we'll talk later about arrays and text strings). So if you define `main` as

  ```
  int main(int argc, char * argv[]) { ....  }
  ```

  `argc` is the number of arguments, plus one, and `argv` is an array of strings containing the arguments.

  ("Plus one"? yes, one of the arguments is something system-dependent, often the path for the program's executable.)
- Reference individual arguments via `argv[0]`, `argv[1]`, `argv[2]`, etc.
- This turns out to be (I think!) a good way to pass text such as filenames to your program.

Files — Examples

- (Example of character-oriented I/O with files.)
- (Example of formatted I/O with files.)

Slide 7

Sidebar: `gnuplot`

- A tool I like for both quick interactive plots and nice-looking ones to use in papers is `gnuplot`. Available on most UNIX-like systems and (I think!) optionally for other operating systems. Home page at `gnuplot.sourceforce.net`. Can do 2D and 3D plots, the former with Cartesian or polar coordinates.
- To start it, `gnuplot`. Brings up a command-line interface. Online help available with `help`.

Slide 8

`gnuplot`, Continued

- Useful commands include `plot` to plot function(s) or data from file(s), `set` to set various things (e.g., x and y ranges).
- Default output to terminal, but with `set terminal` and `set output` you can instead store to a file in various formats.
- Can also put commands (`plot` etc.) in a file and execute batch-style, or with `load`. Useful if you want to regerate plots when data changes.
- (Examples.)

Minute Essay

- Can you think of problems you might want to solve for which a program using files would be a help?
- What tool(s) do you usually use to make plots (or do you)?