Slide 1

Administrivia

- Homework 7 on the Web. Due next Friday. (I hope at least one of the problems will seem interesting to you.)
- Scores on Quiz 4 were mostly disappointing (to me too).

Slide 2

Character Strings in C — Preview

- We'll talk more about text data soon, but for now a summary version:
- Text strings are represented as arrays of characters. Can vary in length; end of string indicated by a special character.
- Text in double quotes (e.g., in a call to `printf`) defines a string constant — so somewhere in memory there is an array of those characters.
- As we'll also discuss more soon, arrays and pointers in C are almost interchangeable, so a function that takes an array as a parameter can specify it as either an array or a pointer. (Look again now at the definition of `printf` and the type of its first parameter.)

Slide 3

Command-Line Arguments — Review

- 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.

Slide 4

Command-Line Arguments in C

- In C, command-line arguments are passed to `main` as an array of 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, the argument indexed 0 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.

Slide 5

Files and C — Review

- 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 to get input from more than one source, or save output in more than one place.
- So C (like many other programming languages) provides ways to work more generally with files. File I/O is based on the idea of "streams" (of characters or bytes).

Slide 6

Streams in C

- In C, streams are represented by the type `FILE *`. `FILE` is something defined in `stdio.h`. (As usual, the `*` means pointer — discussed a bit already, more 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 7

Creating Streams in C

- To create a stream connected with a file — `fopen`.
- Parameters specify filename and whether to open for reading or writing.

Slide 8

Working With Streams in C

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

Slide 9

Files — Review Examples

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

Slide 10

`gnuplot` — Review

- A tool I like for both quick interactive plots and nice-looking ones to use in papers is `gnuplot`.
- To start it, `gnuplot`. Brings up a command-line interface. Online help available with `help`.

Slide 11

`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.)

Slide 12

Minute Essay

- What are some kinds of plots you can imagine yourself wanting to make?
- If you had trouble with the first problem on the quiz, how did you approach it? did you try "tracing through" the code, writing down values of variables and updating them?