Slide 1

Administrivia

- Homework 8 still in work. By Monday I hope, and you'll have a week to work on it.

Slide 2

Minute Essay From Last Lecture

- Most people thought the workload was okay, though a couple mentioned that the homeworks were getting harder. Not a surprise, though I'll try to keep them reasonable.
- About Homework 7, nothing really stood out, except a few people mentioned researching memoization. I didn't intend that you'd need to do that!
- About the quiz (which by the way was Quiz 4 — yes, its heading was wrong), nothing really stood out. My only suggestion if you have trouble understanding what code is doing is to trace through it step by step, writing down what values are assigned to variables.

Slide 3

Arrays — Review/Recap

- VLAs nicer to work with but can be limited in size. (Bubble-sort example programs — notice that creating a VLA with a size that's 'too big" crashes the program, while creating an array of the same size with `malloc` is more likely to work, or at least to fail in a less inscrutable way.)
- Dynamic memory allocation is more flexible. Multi-dimensional arrays are a little tricky, but doable (review examples).

Slide 4

Text Data — Single Characters

- `char` is considered an integer type and can be worked with as such. Note that while these days ASCII is by far the most common encoding, standard doesn't require that, and there are other possibilities.
- Many library functions for working with single characters (e.g., `isalpha`).
- Character literals represented using single quotes.
- Can read in / print single characters with `scanf` or `printf` using `%c`. Or can use `getchar`, `putchar`. Note that `getchar` returns an `int`. Why? so it can return special value `EOF` when no more input.

Slide 5

Text Data — Strings

- Most more-recent languages have nice ways of working with "strings" of text data that hide details and provide nice functionality.
- C, in contrast, provides a bare-bones version, in which text strings are represented as arrays of `char`, with an end-of-string character (`'\0'`) that allows an array of fixed size to store strings of different sizes.

  Simple but subject to all the perils of arrays!
- String literals represented using double quotes. Can include "escape" characters (e.g., `'\n'`.)

Slide 6

Text Strings — Output

- Can use `printf` with `%s`.
- Can also use `puts` (which adds a newline).

Slide 7

Text Strings — Input

- Surprisingly (or not, given C's minimalist implementation of arrays), no nice way to do this!
- Can use `scanf`, but no nice/general way to be sure you don't overflow array, and getting something that includes whitespace may be tricky.
- Can get a whole line with `fgets`, but must provide a fixed-size array (so, what size?) and deal with newlines.
- `gets` looks useful but observe what its `man` page says(!).
- Consider processing data character by character, or using command-line arguments.

Slide 8

Working With Text Strings in C

- Once you have some "strings" in your program, what can you do with them?
- You can work on them as arrays of character (that's what they are) or using pointers (as in the example earlier with an array of `int`s). (Example.)

Slide 9

Minute Essay

- None really — sign in (unless questions?).