Slide 1

Administrivia

- Try to at least look at Homework 7 for next time.

Slide 2

Multi-Dimensional Arrays in C — Recap/Review

- Multi-dimensional arrays can be quite useful, and many languages support them pretty well. C, alas — not so much.
- For small arrays, VLAs work well.
- For large arrays, other options are better. More later.

Multi-Dimensional Arrays, Example — ASCII Art

- We could write a simple "ASCII art" program that "draws" pictures using characters only, with:
  - a two-dimensional array of `char` as the "canvas", and
  - a simple text-menu-driven interface to print, set blocks, clear.

Slide 3

(Look at pre-written code.)

Just For Fun — Extreme ASCII Art

- Try `telnet towel.blinkenlights.nl`. Sometimes site is inaccessible, but when it works . . .

  Control-`]` then Enter, then `q`, to exit.
- (This has been around for a *long* time.)

Slide 4

Slide 5

Pointers Revisited

- Every time you call `scanf`, you pass it at least one parameter of the form `&x`. What does that mean? Also, when you look at `man` pages for some functions, they show function declarations with parameters of the form *type* `*`. What does that mean?
- To explain, we need one more kind of variable — *pointers*. A pointer, as its name suggests, points to something — namely, a location in memory. Typically a pointer "points to" a variable.

Slide 6

Pointers in C

- Many programming languages provide something like pointers. Unlike some more-recent languages, C allows you to have both pointer variables and non-pointer variables.
- To a first approximation, C pointers are just memory addresses — i.e., numbers — but they are declared to point to variables (or data) of a particular type. Examples:

```
int * pointer_to_int;
double * pointer_to_double;
```

- Can display value of pointer using `printf` with `%p`. Sometimes interesting in exploring how variables are laid out in memory (implementation-dependent).

Slide 7

Pointers in C — Operators

- `&` gets a pointer to something in memory. So for example you could write

```
int x;
int * x_ptr = &x;
```

- `*` "dereferences" a pointer. So for example you could change `x` above by writing

```
*x_ptr = 10;
```

- Special value `NULL` means the pointer "doesn't point to anything". Dereferencing a null pointer usually produces an error, as does deferencing an uninitialized pointer variable.

Slide 8

Pass By Reference, Sort Of — Review(?)

- Functions can only explicitly return a single value — a significant limitation. Pointers provide a way to get around that: By passing a pointer to something, rather than the thing itself, can in effect have a function return multiple things.
- To make this work, declare the function's parameters as pointers, and pass addresses of variables rather than variables. (This is how `scanf` does what it does, and why you need the `&`.)
- (The "sort of" in the slide title is because this is not true pass by reference as in, e.g., C++, but the effect is the same.)
- (We did an example of this a while back — the final version of the program to find roots of a quadratic equation.)

Slide 9

Minute Essay

- Questions? about arrays, pointers, . . . ? otherwise sign in.