Slide 1

Administrivia

- How many more homeworks? Probably two, probably one problem each. First of them should be on the Web later today, to be due Tuesday after Thanksgiving. Watch your mail!

Slide 2

Multidimensional Arrays

- Arrays in most (? all that I can think of!) programming languages are similar to subscripted variables in math.
- In math one can have variables with more than one subscript, for example to represent elements of a two-dimensional matrix. The programming-language equivalent is a *multidimensional array*.

Multidimensional Arrays in C

Slide 3

- Declaring and using multidimensional arrays is similar to declaring and using one-dimensional arrays. Example:

```
int twoD[10][20];
twoD[5][6] = -1;
```

- Two-dimensional arrays can be visualized as consisting of rows and columns. The above example declares an array with 10 rows and 20 columns and then references an element in the row with index 5 and the column with index 6 (indices start at 0, remember). Analogous concepts apply to arrays with more dimensions, but they're harder to draw / visualize.

Multidimensional Arrays in C, Continued

Slide 4

- With the older standard for C (C89), passing multidimensional arrays to functions is ugly, unless the size of the array is known at compile time.
- The variable-length arrays of the newer standard seem to allow things to be done more nicely.
- (Example.)

Slide 5

Pointers

- 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. How this is done in C is lower-level than in some other languages — risky in some ways, but also exposes more about the underlying hardware.
- In C, pointers are just memory addresses — i.e., numbers — but they are declared to point to variables (or data) of a particular type. Example:

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

Slide 7

Pointers in C — Operators

- `&` gets the address of 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;
```

- (Example of using this to look at how things are laid out in memory.)

Slide 8

Pass By Reference

- A significant limitation we've had to deal with is that functions can only return a single value. Pointers provide a way to get around this restriction: By passing a pointer to something, rather than the thing itself, we can in effect have a function return multiple things.
- To make this work, typically you 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 `&`.)
- (Example.)

Slide 9

A Little About Strings in C

- (We'll talk about strings more shortly, but a preview now so we can use them — e.g., for file names!)
- Most programming languages provide a way to represent text (sequence of characters). C differs from some others you might use in providing only a very simple way that exposes details of the implementation.

Slide 10

Strings in C, Continued

- In C, a (character) string is an array of characters, with a *null character* (written `'\0'`) at the end. So to declare a variable to hold a string, you might write

```
char mystring[100];
```

- String literals (constants) are written with double quotes. Because of the way C treats pointers and arrays as almost equivalent, functions that take string parameters often declare them as having type `char *`. (Now the definitions of many functions, as given in their `man` pages, should make more sense!)

Slide 11

Strings in C, Continued

- To print a string, you can use `%s` with `printf`.
- To read a string from standard input or a file, you can use `scanf` or `fscanf` — but this is risky unless you limit how many characters are read! May be better to read a whole line with `fgets`.

Slide 12

Minute Essay

- None — sign in.