Slide 1

Administrivia

- Homework 4 to be on Web soon; due next week.

Slide 2

Pointers in C — Review/Continued

- Pointers in C are similar to, but not identical to, references in Java — with the key differences having to do with safety features and level of abstraction. (No surprise!)
- In C, pointers are just memory addresses — but they are declared to point to variables (or data) of a particular type. Example:

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

Slide 3

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;
```

- You can also perform arithmetic on pointers (e.g., `++x_ptr`) — something not allowed in Java, and another example of the languages' different design goals.

Slide 4

Pass By Reference (Sort Of)

- A significant potential limitation on functions is that a function 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.
- The "sort of" of the title means that this isn't true pass by reference, as it exists in some other languages such as C++, but it can be used to more or less get the same effect. Notice also that Java can't do this, though again there are mechanisms that can more or less get the same effect. (What?)

Slide 5

Pointers Versus Arrays

- In C, pointers and arrays are in some sense(s) equivalent — not identical, but in many contexts interchangeable.
- This is reflected in the `man` pages for many functions (e.g., `printf`). It also means that when you pass an array to a function, what you're actually passing is a pointer — so the array is not copied.

Slide 6

I/O in C — Some Very Basic Functions

- `getchar` gets one character and returns it as an `int`. The special value `EOF` indicates end of input. ("End of input"? control-D from terminal, more in next sidebar.)
- `putchar` writes out one character.
- Use this to write a very simple program that simply copies its input to its output . . .

Slide 7

I/O in C, Continued

- You already know about a function to write output to "standard output", `printf`. Many options, allowing a lot of control over what's printed.
- How about input? Counterpart of `printf` is `scanf` (skim man page). Simple to use, though error detection is somewhat crude, and reading text strings can be hazardous.