Slide 1

Administrivia

- Homework 4 to be on the Web tomorrow (I hope!). I will send mail. To be due a week after being assigned.

Slide 2

One More Tip

- Remember to compile with `-Wall` and pay attention to any warning messages. (The first thing I do when students tell me their code doesn't work is . . . It doesn't always help but sometimes it does!)

Slide 3

Pointers in C — Overview

- C, in contrast to Scala and Java (and Python), makes an explicit distinction between things and pointers-to-things.
- In Python and Scala variables are pointers/references to objects, and you deal with them fairly abstractly. In Java, variables are either references to objects, or primitives, but one or the other.
- In C, you can have variables that are “things” (integers, floating-point numbers, etc.) and variables that are “pointers to things” (in some ways more like variables in Python and Scala, but very low-level and with fewer safety checks).

Slide 4

Pointers in C — Overview Continued

- That is, in C, pointers can be thought of as memory addresses (indices into large one-dimensional memory space — not always strictly true but a good first approximation), though declared to point to variables (or data) of a particular type.
- Example types:

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

Slide 5

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;
```

- You can also perform arithmetic on pointers (e.g., `++x_ptr`) — something not allowed in languages more concerned with safety.

Slide 6

Parameter Passing in C — Review

- In C, all function parameters are passed "by value" — which means that the value provided by the caller is copied to a local storage area in the called function. The called function can change its copy, but changes aren't passed back to the caller.
- An apparent exception is arrays — no copying is done, and if you pass an array to a function the function can change its contents (as you would want to do in, say, a sort function). Why "apparent exception"? because really what's being passed to the function is not the array but a pointer! so the copying produces a second pointer to the same actual data.
- This is at least simple and consistent, but has annoying limitations . . .

Slide 7

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.)

Slide 8

Pointers Versus Arrays

- In almost all contexts arrays and pointers are interchangeable.
- In particular, if you declare the type of a function parameter to be a pointer, you can pass it an array, and vice versa.

Slide 9

Strings in C — Overview

- C has a data type `char`, used for much the same purposes as characters in other language, *but* with a smaller minimum range (enough to represent 7-bit ASCII but not Unicode).
- C "strings" are null-terminated arrays of characters and can be worked with as arrays or using pointers. There are standard library functions for doing (some) things with characters and strings.
- (Examples as time permits.)

Slide 10

Minute Essay

- What about today's lecture was unclear? (Maybe the answer will be "nothing" but probably not.)