Slide 1

Administrivia

- Reminder: Homework 3 due today.

Slide 2

Minute Essay From Last Lecture

- (Review question, my answer.)
- Most people gave sensible answers. For the first question, a few mentioned that Minecraft has a “world seed” option(?). For the second question, several mentioned that it’s easier to reduce a large range to a small one than vice versa. Indeed!

Slide 3

Using Library Functions, Revisited

- (Prompted by a student question about what you need to do to use the library `sqrt` function in your code.)
- Recall from one of the early lectures that getting from source code to an executable file involves two steps, "compiling" and "linking".

  Compiling a call to a library function just requires that the compiler know about its parameters and return type. Get that via `#include`. Actual code (for library function) not needed until link step. For `gcc`, if not found in "standard" place(s), need additional flag(s) (e.g., `-lm`) to say where.

Slide 4

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).
- That is, in C, pointers are basically just memory addresses, though declared to point to variables (or data) of a particular type. Example:

```
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 we did in the sort program). 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 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` — strings are arrays of characters). 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 9

Pointers, Continued

- Calls to `scanf` should now make sense — the function is supposed to store values into variable(s), and with pass-by-value we can't do that unless we pass a pointer.
- (Simple example.)

Slide 10

Strings in C

- Many languages have nice ways of working with text (character strings). C does allow you to work with text, but what it provides is — no surprise — somewhat primitive.
- In C, strings are arrays of `char`s, with the convention that the actual text of interest is followed by a null character (8-bit zero, represented in code as `'\0'`.

Slide 11

Working with Strings in C

- You can operate on individual characters however you see fit (accessing them as elements of the array). Or you can access them using pointers to `char`. (Recall that arrays and pointers are interchangeable in most contexts.)
- There are some useful standard-library functions for working with characters; `man ctype.h` will list them.
- There are also standard library functions for some common operations (e.g., `strcmp` to compare two strings — returns -1/0/1 depending on which string is lexicographically first). Simplest way to find them may be `man -k string` and ignore everything but the last few screenfuls.
- `scanf` and `printf` use `%s` to read/write strings. (Use with caution — more later.)
- (Simple example.)

Slide 12

Minute Essay

- Anything comment-worthy about homework 3?