Slide 1

Administrivia

- Homework 1 grades mailed earlier today. If you (think you) turned something in and didn't get an e-mail from me with your grade (subject line will contain "csci 1120", check your spam folder and then ask.

  I will post a sample solution soon.
- If you haven't turned in Homework 2, please try to do so ASAP.
- Homework 3 was due today but a little more time may be needed. So extend due date to next week. More about the problem(s) today.

Slide 2

Homework 3 — Hints and Tips

- One of the objectives of this homework was for you to practice loops (both `for` and `while`) and working with arrays. In my thinking `for` loops are a good fit for the first problem, while a `while` loop works well for the second, and the first problem almost certainly needs at least one array.

  Both problems are probably slightly more challenging than the ones in the previous assignments. More about "random" numbers in the next few slides . . .
- 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

A Very Little About "Random" Numbers

- Homework 3 asks you to work with the library functions `srand()` and `rand()`. A few words about what they do . . .
- First, what we mean by "random" is (I think!) an interesting question with no obvious answer. What's often wanted is something that can't be predicted, and it's not clear we can get that with a system that's deterministic. Further, even if we could, we might not want that, since we often want to be able to repeat a test.
- So, often what we really want is a "pseudo-random number generator" — something that generates a sequence of numbers that looks random but are repeatable given some reproducible starting point.
- Early researchers apparently thought more-complex algorithms would give better results, but — not necessarily. Very simple algorithms can give quite good results.

Slide 4

A Very Little About "Random" Numbers, Continued

- Lots of uses for "random" sequences (e.g., so-called "Monte Carlo" methods for simulating things), so many libraries include function(s) to produce them.
- Typical library provides some way to set the starting point (the "seed") and then a function that when called repeatedly produces the sequence — `srand()` and `rand()` in standard C. Mostly these produce a large range of possible values. (Why is this good?)
- Some libraries also provide functions to map the full range to a smaller one (e.g., to simulate rolling a die). C doesn't, but there are some semi-obvious approaches. The problem on Homework 3 asks you to do a simple comparison of two of them.

Slide 5

Review — Loops and Arrays in C

- Two basic kinds of loops — `while` and `for` (three if you also count `do ... while`).
- Arrays conceptually similar to arrays in higher-level languages, but underlying implementation shows through more clearly, maybe, in the lack of safety checks and extra(?) features.

Slide 6

Arrays — Example(s)

- A familiar(?) thing to do with a collection of data — sort it.
- So let's sketch a program to sort an array. For now, have the program generate the data using `rand()`.

Slide 7

Pointers in C — Preview

- 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 8

Pointers in C — Preview Continued

- Pointers are declared as in these examples:

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

  (so the parameter types for some library functions may now make a little bit more sense).

- & gets a pointer to something in memory (so now maybe the way you call `scanf` makes a little bit more sense). example you could write
- To be continued . . .

Slide 9

Minute Essay

- With regard to `srand` and `rand`:
  - Why is it good to be able to generate the same sequence or different sequences?
  - Why is it good to have a large range of possible values?

Slide 10

Minute Essay Answer

- With regard to `srand` and `rand`:
  - It's useful to be able to generate the same sequence for testing and debugging. For actual use of an application, though, it's probably more useful to be able to generate different sequences.
  - Being able to generate a large range of possible values is probably good in its own right. But it's almost essential if the algorithm for generating the sequence is one that generates each element from the previous one, since otherwise once we repeat an output value the sequence just repeats.