## Administrivia

- Reminder: Homework 4 due next week.

- Homework 3 grades mailed. Most people did well. Several people lost points because they didn't check for input errors. Remember that the course Web site has some "sample programs" meant to be useful in doing homeworks. Feel free to copy code from them.

- Sample solution for Homework 3 posted (bottom of "lecture topics and assignments"). (So if you weren't happy with your answer — "couldn't this be simpler?" — compare with mine?) Also solutions to earlier assignments.

**Slide 1**

## Minute Essay From Last Lecture

- No clear consensus on how to get seconds per hours, etc. — some people calculated, some used Web search, and a few . . .

- A few did the calculation in the program. To me this is better! more readable, less chance of getting it wrong.

**Slide 2**

## Homework 3 Essays

- Nothing really stood out.

- A few people mentioned that the problems helped them learn topics covered in class. That's my goal!

**Slide 3**

- A few people mentioned looking up Euclid's algorithm to understand it. Understanding is good, though for this problem you *could* just turn the math definition into code without understanding it.

## A Very Little About "Random" Numbers

- Homework 4 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.

**Slide 4**

- 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.

- (Canonical reference — discussion in volume 2 of Knuth's *The Art of Computer Programming*. Very mathematical. Other references may be easier.)

- (Aside: In the process of getting TAOCP published, Knuth got curious about typesetting via computer and started a side project that eventually produced TEX. Classic example of a side project that turned into much more!)

## A Very Little About "Random" Numbers, Continued

- So, often what we really want is a "pseudo-random number generator" — something that generates a sequence of numbers that looks random but is 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 5**

## 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 4 asks you to do a simple comparison of two of them.

**Slide 6**

## Arrays — Review/Recap

- As in other languages, arrays give you a way to create the an indexed collection with all elements of the same type.

- Unlike most modern(?) languages, arrays in C are a thin veneer over the implementation and lack safety checks and object-oriented features such as built-in length.

**Slide 7**

## 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.

**Slide 8**

- In C (and 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).

## 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.

**Slide 9**

- Example types:

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

## 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

**Slide 10**

```
*x_ptr = 10;
```

(What do you think happens if `x_ptr` hasn't been initialized?)

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

## 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.

**Slide 11**

- 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 . . .

## 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.

**Slide 12**

- 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.)

- (Example?)

## 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 13**

## Minute Essay

- Any questions — about pointers, anything else?

**Slide 14**