Slide 1

Administrivia

- (None?)

Slide 2

Minute Essay From Last Lecture

- Fewer people said they'd had trouble with the homeworks than in previous years, but judging by the number of people who came to office hours Friday many still didn't find it easy.
- One person commented on how there was no built-in way to fins max or min of two (or more?) numbers. True! C doesn't make it easy to define such functions only once for a variety of types (as some languages do). Many C programs use "macros" to get much the same effect, e.g.,

```
#define MAX(a, b) ((a) > (b) ? (a) : (b))
```

Slide 3

Programming Tip

- If you're testing multiple conditions, only one of which is meant to be true, probably best to do so with a chain of

  `if .... else if .... else if .... else`

  rather than a lot of `if`s.
- In such a chain, notice what you already know when you get to an `else`. Example — I say there's something redundant in the following code:

```
if (a < b) { .... }
else if (a >= b) { .... }
```

  (Spot it?)

Slide 4

Recursive Functions — Examples, Continued

- First an example I've used as a minute essay question in past years:
- Here is a C function that does — something.

```
unsigned int foobar(unsigned int a, unsigned int b) {
    if (b == 0) {
        return a;
    }
    else {
        return 1 + foobar(a, b-1);
    }
}
```

  What does `foobar(5, 2)` return?
- Can you say what this function seems designed to accomplish?

Slide 5

Recursive Functions — Examples, Continued

- `foobar(5, 2)` returns 7. Why . . .

```
foobar(5, 2) = 1 + foobar(5, 1)
             = 1 + 1 + foobar(5, 0)
             = 1 + 1 + 5
             = 7
```

- It's a roundabout way of doing addition!
- A cultural(?) note: The name of the mystery function (`foobar`) is one used often in CS when one needs a more or less meaningless name for something, along with variants `foo`, `bar`, and so forth. Apparently based on WWII-era acronym FUBAR.

Slide 6

Sidebar: Tracing Code

- A valuable skill to have is working through what the computer will do when it executes your program — "tracing code" (also known as "desk checking", from the days before desktop computers, or "playing computer").
- Idea is to write down names of variables, their values; when one changes, cross out old value and put in new one.
- (Short examples?)
- Also can be useful to enlist the computer's help with this, via "debug print" statements. Just remember to remove them (or at least comment them out) when you get things working!

Slide 7

Recursive Functions — Examples, Continued

- Last time we(?) mostly wrote a function to get an integer from stdin without using `scanf`, as a more complicated example of using functions and recursion.

  It's more difficult than I really expect you to do at this point, but I thought was also more interesting, and illustrates how functions can help you break a problem down into somewhat more manageable pieces.
- Really the function should also skip leading white space, if any.
- Also, having it just exit the program on error seems ugly, so let's do that another way — maybe do as `scanf` does and return something indicating success/failure and save the value through a pointer argument.
- (Revise the code . . .)

Slide 8

Minute Essay

- None — quiz.