Slide 1

Administrivia

- Homework 4 deadline extended to Friday.

Slide 2

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`. For example, I say there's something redundant in the following code:

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

  (Spot it?)

Slide 3

Recursive Functions — One More Example

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

```
int foobar(int a, 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 4

Recursive Functions — One More Example, 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 5

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 6

Sidebar: `gcc` Options

- I mentioned already the value of compiling with `-Wall`. There are many other options that I think are useful:
- `-pedantic` warn you about non-standard usage.
- `-std=c99` allows you to use features new with C99.
- `-O` optimizes, and also seems to find some problems not found without it.
- `-o` allows you to name the output file (default `a.out`).
- You aren't going to type all of those . . .

Slide 7

A Very Little About `make`

- `make` is a old-style UNIX tool for building programs. I'll talk more about it later, but for now it will be useful for always compiling with particular options:
- If you have, in the directory where you compile, a file `Makefile` similar to the one on the "sample programs" page, and you have a file `hello.c`, typing

  ```
  make hello
  ```

  will compile with the options I think are useful and put the result in `hello`. Execute with `./hello`.

Slide 8

Sidebar: "Undefined Behavior" in C

- You may have noticed that if you try to input a really large value with `scanf` you don't get either the right value or any kind of error.
- You might also notice that strange things happen when you try to compute a fairly large number using an `int`. (This is easy to do with our factorial program.)
- Both examples of what C calls "undefined behavior". Means that the language doesn't say what's supposed to happen. Might be different depending on compiler and options!
- A really careful programmer checks to make sure this can't happen. (Revise factorial program.)

Slide 9

Minute Essay

- None — quiz.