Slide 1

Administrivia

- Reminder: Homework 4 due today.
- Homework 5 on Web, due next Thursday.

Slide 2

UNIX/Linux Tip

- I strongly encourage using `gcc`'s optional flags `-Wall` and `-pedantic`, or at least `-Wall`. But that's a lot to type every time. So:
- Remember that the up arrow cycles through previous commands.
- Or copy the `Makefile` from the "Sample programs" page here. into the directory with your programs, and type `make hello` to compile `hello.c`. Note that the result will be called `hello` rather than `a.out` (so to run it you type `hello` rather than `a.out`).

Slide 3

Functions — Review

- Functions are somewhat like math functions — zero or more inputs, one output (*return value*) or none.
- *Defining* a function – specify
  - Its name (same rules as for variables — letters, numbers, underscores).
  - What parameters it needs (types, “local names” — `void` if none.
  - What type of thing it returns (`void` if nothing).
  - Some code. Can include *local variables*. If function returns something other than `void`, must include at least one `return` followed by the value to return.
- *Declaring* a function — just give name, parameters, return type. Definition can be somewhere else in the program.

Slide 4

Functions — Review, Continued

- *Calling* a function — give its name, values for parameters. This is an expression (in the same sense as, say, `x+1`) and — unless the function returns `void` — has a value, which can be assigned to a variable, used as part of a boolean expression for conditional execution, etc.
- Since a function call is an expression — when we come to one, we evaluate it:

  Pause what’s currently happening. Copy values of input variables to function’s parameters. Execute code in function until we get to a `return`, or the ending curly brace. Whatever expression follows `return` is the function’s (return) value. Continue execution in “caller” using return value.

  Notice that executing code in the function may produce “side effects” (e.g., printing something).

Slide 5

Repetition — Review/Recap

- Several ways to repeat something — recursion. loop constructs discussed last time.
- Which to use? in general, the one that makes the programs easiest for humans to understand — worry about efficiency only when it matters.
- Key ideas to think about in designing loops:
  - What is it you want to repeat? and what's different about each repetition? This should tell you what the loop body is, and what variable(s) in it will change.
  - For the variables that will change, what should their initial value be? How do they change from one iteration to the next?
  - When do you stop repeating?

Slide 6

Examples of Using Loops

- Look again at the "convert English to metric" example program. We could make some improvements (or changes anyway):
  - First change it so it lets you do multiple conversions without running the program again.
  - Now change the function that gets a number so if you type in something other than a number, it asks again.
- Suppose we want to read in a bunch of numbers and print their sum. How to do that? (Next time.)

Minute Essay

Slide 7

- None — quiz.