Slide 1

Administrivia

- Homework 1 and 2 grades sent by e-mail. If you didn't get one — are you sure you turned something in?
- Homework 3 on the Web. Due next Monday. Should be doable with material up through last week.
- Reminder: Quiz 1 Wednesday. Topics will come from what we covered up through last week. Rules as described in "administrivia" last time.

Slide 2

Minute Essay From Last Lecture

- A few people said things that support that line I quote so often about programming not being a spectator sport.
- A few people commented on the (5/9) being evaluated using integer division. (That was part of the point of this problem.)
- Others commented on "int" having a limited range. Hm, interesting what examples people tried?

Slide 3

Quotes of the Day/Week/?

- From a key figure in the early days of computing:

  "As soon as we started programming, we found to our surprise that it wasn't as easy to get programs right as we had thought. Debugging had to be discovered. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent finding mistakes in my own programs." (Maurice Wilkes: 1948)

- From someone in a discussion group for the Java programming language:

  "Compilers aren't friendly to anybody. They are heartless nitpickers that enjoy telling you about all your mistakes. The best one can do is to satisfy their pedantry to keep them quiet :)"

Slide 4

Functions and Problem Decomposition

- So far all our programs have been one big chunk of code. This is okay for simple programs, but quickly becomes difficult to understand as problems get bigger.
- Further, some things we don't want to, or can't, really write ourselves, such as the code for input/output.
- So C, like many/most other programming languages, gives you a way of decomposing problems into subproblems. C calls them *functions*. Using this feature to good effect is something of an art, but may teach you something about problem decomposition in general, which is a useful skill.

Slide 5

Functions in C

- C functions are similar to functions in math, except that they can have side effects (similar to how evaluation of expressions can have side effects).
- We will talk a little now, and more next time, about how to define our own functions. Notice for now that every program you / we have written so far defines a function called `main`, and most of them use system library functions `scanf` and `printf`.

Slide 6

Functions in C, Continued

- Every function has
  - A name (where rules for names are the same as those for variables).
  - Zero or more inputs (called *parameters*).
  - A return type (`void` to indicate that the function doesn't return anything).
  - Some code to be executed when the function is called.
- When you call (use) a function, you
  - Supply values for inputs (pass in values for parameters).
  - Optionally, use the value returned by the function. The function call is an expression, as discussed previously, and its value is the value returned by the function.

Slide 7

Defining and Using Functions

- Simple example of defining and using a function to add two integers:

```
int add(int a, int b) {
    return a + b;
}
int main(void) {
    int result = add(1, 2);
    printf("%d\n", result);
    return 0;
}
```

- `add` has two parameters (a type of variable) called `a` and `b`. When we call `add` from `main`, the values 1 and 2 are copied into these variables. The code in `add` executes until it reaches a `return`. At that point, we go back to the calling function, and the value of the function call is whatever is after the keyword `return`.

Slide 8

Functions in C — Declaration Versus Definition

- Some languages let you put function definitions in any order you want, and even split them up among files.
- But this requires the compiler to be somewhat smarter than C compilers are required to be. In C, functions must either be defined or *declared* before being used.
- Function declarations give function name, number and types of parameters, and return type. Syntax is just like that for function definitions, except no parameter names needed, and body is replaced with a semicolon.
- For your own functions, you can either define them before using them, or define them in whatever order you like and put declarations at the top.
- For library functions? declarations are part of what's supplied by `#include` directives.

Slide 9

The `main` Function, Revisited

- As noted, every C program you / we have written so far includes a definition of a function called `main`. All complete C programs must have such a function.
- `main` is defined in your code:
  - It has no parameters. (Actually, it can — there's an alternative definition that allows it to accept command-line arguments, similar to the ones that follow commands such as `gcc`, `ls`, etc. Later!)
  - It returns an integer value.
- `main` is called by some type of environment (the command shell for us, when you type `a.out` after compiling). It gives your code the optional parameters (more about this later) and receives the value you return. Return value can be used to indicate success/failure (useful for shells that themselves support conditional execution).

Slide 10

"Hello World" Program, One More Time

- Historical/cultural aside: Among computer programmers, it's considered traditional that the first program one writes in a new language just prints "hello world" to the screen — maybe not the simplest possible program, but close. Particularly apt for C, because the tradition was begun by an early and still authoritative work on C (*The C Programming Language*, Kernighan and Ritchie).
- Almost all of this program, and other examples, should now more or less make sense! (Exceptions are representation of character strings, & syntax for parameters. Soon!

Slide 11

C Library Functions

- Standard C comes with a number of *library functions* to do things many programs want to do.
- Examples we've seen so far — `scanf`, `printf`.
- UNIX/Linux systems normally have `man` pages for these functions, describing parameters and return values in full detail (hence, not always easy reading).

  (Tip: `man printf` gives the `man` page for a command rather than the C function. Use `man 3 printf` to get what we want.)

  (Tip: When reading a `man` page, `h` will bring up a summary of what keys do what — page up/down, quit, etc.)

Slide 12

Defining and Using Functions — Example

- As a somewhat contrived example, we could rearrange the "solve a quadratic equation" example from last week:
- By putting the code to solve the equation and print results in a function, we can have it first do some examples/tests before prompting the user for inputs . . .

Slide 13

Minute Essay

- Any questions? otherwise just “sign in”.