

Slide 1

### Administrivia

- Reminder: First quiz Monday. As noted previously, you will have access to the textbook, your notes (paper or electronic), and the course Web site, but the only allowed computer use is to access these (so no typing in code and trying it). Intended to take no more than 10 minutes.  
Likely questions include “what does this C program print to the screen?”, “write some C code to do the following”, and questions about the material on binary numbers.
- Homework 3 on the Web. Due next Friday. Two problems, both doable just with assignment, simple I/O, and conditional execution.

Slide 2

### Minute Essay From Last Lecture

- Several people mentioned that the seconds problem was similar to the counting-change example from class. That was on purpose!  
Several also mentioned that the examples in general were helpful. That also is what I aim for!
- A few also mentioned that actually writing programs helped clarify a lot of things. Indeed. “Not a spectator sport”? (Though I think there can be value in watching someone with more experience program — otherwise I wouldn’t do so much of it in class.)
- Several people mentioned being initially surprised by what you get in C when you write `5 / 9`. That’s a lot of the reason I have students do this program!

Slide 3

### 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 really can't, 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*.  
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).  
Using this feature to good effect is something of an art, but experience helps.

Slide 4

### 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.

### Defining and Using Functions

Slide 5

- 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 called `a` and `b`, which are basically variables local to the function. When we call `add` from `main`, 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`.

### Functions in C — Declaration Versus Definition

Slide 6

- Many languages let you put function definitions in any order you want, and even split them up among files.
- But some of 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 7

### 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 8

### “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!)

## C Library Functions

Slide 9

- 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.)

## Defining and Using Functions — Example

Slide 10

- As a somewhat contrived example, we could rearrange the “solve a quadratic equation” example from previous class.
- By putting the code to solve the equation and print results in a function, we can also easily have it print some examples/tests. Maybe do this before prompting for input?

## Minute Essay

- Any questions?

Slide 11