Slide 1

Administrivia

- Reminder: Homework 9 due Friday.
- Last quiz the Monday after break. Topics TBA.

Slide 2

User-Defined Types in C — `enum`

- In C (and in some other programming languages) an *enumeration* or an *enumerated type* is just a way of specifying a small range of values, e.g.

```
enum basic_color { red, green, blue, yellow };
enum basic_color color = red;
```

  This can make code more readable, and sometimes combines nicely with `switch` constructs.

- Under the hood, C enumerated types are really just integers, though, and they can be ugly to work with in some ways (e.g., no nice way to do I/O with them). Worth(?) noting that other languages (Scala for example) provide nicer ways to do this.

Slide 3

User-Defined Types in C — `union`

- For completeness we should mention that C also provides a way of defining a structure that can contain one of several alternatives ("this OR that", as opposed to the "this AND that" of `struct`) — `union`.
- See discussion in textbook about this; it can be useful, but can also make code more difficult to understand.

Slide 4

User-Defined Types and Library Code

- Library code often makes use of "opaque" types (e.g., `FILE`).
- Implementing this often involves separating functionality into interface (`.h` file containing type definitions, function declarations) and implementation (`.c` file containing function definitions. (Example later.)

Slide 5

Functions as Parameters to Other Functions

- There are situations in which it's useful to allow functions to use other functions as parameters.
- One example is sorting — the same algorithm applies to sorting anything for which there's a well-defined "less than" operator, so sorting `int`s is much like sorting `double`s, which in turn is much like sorting strings (except that things get a little tricky there because they have varying lengths). So — maybe a general sort function, with one parameter that represents the "less than" operation to use?
- Another example is our numerical-integration program — we could use much the same code to perform numerical integration on different functions if we could somehow make the function to be integrated a parameter.

Slide 6

Functions as Parameters to Other Functions, Continued

- Some languages provide nice built-in support for this idea ("functions are first-class objects"). Examples go back to early "functional languages" and include several more-recent languages such as Scala, Python, and Java (though it's a little clumsy in Java).
- C provides a way to get this effect, via "function pointers".

Slide 7

Function Pointers in C

- The type of a function pointer includes information about the number and types of parameters, plus the return type.
- Example — last parameter to library function `qsort` (in its `man` page). Call this by providing, in your code, a function with declaration

```
int my_compare(const void *, const void *);
```

and using `my_compare` as the last parameter to `qsort`.

(Revised sample program.)

Slide 8

Function Pointers

- Another good use would to something generalizes the code we wrote to approximate π with numerical integration:
- Pretty much the same calculations could be used to approximate any definite integral; just provide start and end points and function.
- (Look at code.)

Slide 9

Minute Essay

- Anything noteworthy about Homework 8?
- If you lost points on the second problem of Quiz 5 (which almost everyone did), can you say what you think went wrong?
- Anyone not planning to be here Monday?