Slide 1

Administrivia

- Homework 9 due date extended to next Monday. One more homework planned.
- Last quiz the Wednesday after break. Topic(s) TBA.
- Homework 6 sample solution posted.

Slide 2

Homework 9

- First problem should be doable once you understand what you have to do:

  Fill in body of `update_board()` so it assigns values to elements of `new_board` based on values of `old_board`, according to the rules of the game.

  Elements of the 2D array are `bools`. Functions `read_board()` and `print_board()` may be helpful in showing you how to work with a 2D array as defined in the starter code.
- Second problem is more difficult but should be doable. Sample program `echo-args.c` shows how to get numeric values from command-line arguments.

Slide 3

User-Defined Types — `struct`

- C's `structs` provide a way to define your own type: You define its fields and give it a name (its type), and then when you declare a variable with that type, you get a little box with places for all the fields.
- (Another example.)

Slide 4

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 5

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 6

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 7

Minute Essay

- Have you seen something like `struct`s somewhere else? possibly Matlab??
- Can you think of programs you might want to write in which they'd be useful?
- And best wishes for a good holiday!