Slide 1

Administrivia

- Reminder: Homework 8 due Monday. How many more . . . One more comparable to recent ones, or two short ones.
- Quiz 6 (last one!) scheduled for Wednesday after holiday.

Slide 2

Minute Essay From Last Lecture

- Most people's responses — I think for many, `struct`s are new enough that what you might use them for isn't clear yet, or not clear enough to express easily!
- (I would say that mostly what they give you is a way to express some things in a way that's easier to understand, though "opaque types" do provide something I'm not sure how you could easily get otherwise . . .)

Slide 3

User-Defined Types and Library Code, Revisited

- Library code often makes use of “opaque” types (e.g., `FILE`).
- One useful thing about this — libraries can be written in terms of these types and implemented differently on different systems, with application programmers not needing to know how implemented. (E.g., a `FILE` could be a `struct` containing who knows what, or an index into an O/S-built table, or . . .)

Slide 4

Bitwise Operators

- In what we’ve done so far, we’ve dealt with most data without needing to know exactly how it’s represented in terms of 0s and 1s (though knowing a little about that helps you understand limitations and pitfalls).
- However, for various reasons it can be useful or even necessary to work with individual bits — e.g., working with image data (where a “pixel” is represented by some collection of n-bit fields), or working at a very low level with I/O devices. Some system-specific functions callable from C also take as parameters integers that are the result of combining bits.
- So C, like many programming languages, provides operators to allow that . . .

Slide 5

Bitwise Operators and C

- Bitwise “and” (both bits): `&`
- Bitwise “or” (either bit): `|`
- Bitwise “exclusive or” (either bit, not both): `^`
- Bitwise negation/complement (unary operator, flips bits): `~`
- Left and right shifts (specify how many bits): <<, >>).
- All work on integer types.

Slide 6

Bitwise Operators and C, Continued

- In many programming languages, sizes of integer types are fixed, which can make it easier to do this kind of thing.
- In C, however . . . (so you need to be a little careful).

Bitwise Operations

- As an example of using some of these operators and also of using a `union`, write a program to show the bits in a `long`, two ways.
- Trying it, on a 64-bit system and also on a rather old 32-bit system, some results are surprising.

Slide 7

Bit Manipulation in C

- A typical "use case" for these operators is in working with an integer that's not really an integer so much as a collection of bits, each with a meaning (flags used to communicate with an I/O device, e.g.).
- Typically define "masks" for individual bits or collections of bits, giving them names (via `#define`) and then use bitwise operators to set, clear, test.
- As noted, lack of standardized size for most integer types can be a problem, but C99 introduced some fixed-size integer types (`<stdint.h>`).
- (Example.)

Slide 8

Minute Essay

- What is the result of applying bitwise operators as follows:

  1110_2 `&` 1001_2 (bitwise and)

  1110_2 `|` 1001_2 (bitwise or)

  1110_2 `^` 1001_2 (bitwise exclusive or)

  `~`1110_2 (bitwise negation)

Minute Essay Answer

- What is the result of applying bitwise operators as follows:

  1110_2 `&` 1001_2 is 1000_2

  1110_2 `|` 1001_2 is 1111_2

  1110_2 `^` 1001_2 is 0111_2

  `~`1110_2 is 0001_2