Slide 1

Administrivia

- (None.)

Slide 2

Minute Essay From Last Lecture

- Most people had not tried writing programs on their own outside class. But a few had!
- One person asked about deciphering `gcc` error messages. They *can* be cryptic. As you get more experience with C and `gcc` some of them may make more sense, but if you can't make sense of the text of the message, it will usually have the line and column number where it found a problem, and often if you look carefully at that line or the preceding line you can spot the problem. (How to find a line number in `vim`? **:** followed by the line number. Also notice that the line number should show up in the bottom right corner of the window.)

Slide 3

Binary Numbers

- We humans usually use the decimal (base 10) number system, but other (positive integer) bases work too. (Well, maybe not base 1.) Binary (base 2) is more widely used in computers because it makes the hardware simpler.
- In base 10, there are ten possible digits, with values 0 through 9.

  In base 2, there are 2 possible digits (*bits*), with values 0 and 1.
- In base 10, 1010 means what? What about in base 2?

Slide 4

Converting Between Bases

- Converting from another base to base 10 is easy if tedious (just use definition).
- Converting from base 10 to another base? Let’s try to develop an “algorithm” (procedure) for that . . .

Slide 5

Decimal to Binary, Take 1

- One way is to first find the highest power of 2 smaller than or equal to the number, write that down, subtract it from the number, and continue:
  1. If $n = 0$, stop.
  2. Find largest p such that $2^p \leq n$.
  3. Write a 1 in the p-th output position.
  4. Subtract 2^p from n.
  5. Go back to first step.
- Is this okay? What's not quite right about it? (We don't say what to put in the positions that don't have ones in them.)
- (Example.)

Slide 6

Decimal to Binary, Take 2

- Another way produces the answer from right to left rather than left to right, repeatedly dividing by 2 (again n will be the number we want to convert):
  1. If $n = 0$, stop.
  2. Divide n by 2, giving quotient q and remainder r.
  3. Write down r.
  4. Set n equal to q.
  5. Go back to first step.
- Is this okay? What's not quite right about it? (We don't say to write down the remainders from right to left.)
- (Example.)

Recap

- Key ideas here — break problem down into a sequence of steps that we hope don't require much intelligence, just an ability to calculate, with some decision-making and repeating.
- Before moving back to programming and C, a little more about different number bases and how binary numbers are used to represent data . . .

Slide 7

Octal and Hexadecimal Numbers

- Binary numbers are convenient for computer hardware, but cumbersome for humans to write. Octal (base 8) and hexadecimal (base 16) are more compact, and conversions between these bases and binary are straightforward.
- To convert binary to octal, group bits in groups of three (right to left), and convert each group to one octal digit using the same rules as for converting to decimal (base 10).
- Converting binary to hexadecimal is similar, but with groups of four bits. What to do with values greater than 9? represent using letters A through F (upper or lower case).
- (Examples.)

Slide 8

Slide 9

Computer Representation of Integers

- Computers represent everything in terms of ones and zeros. For non-negative integers, you can probably guess how this works — number in binary. Fixed size (so we can only represent a limited range).
- How about negative numbers, though? No way to directly represent plus/minus. Various schemes are possible. The one most used now is “two’s complement”: Motivated by the idea that it would be nice if the way we add numbers didn’t depend on their sign. So first let’s talk about addition . . .

Slide 10

Machine Arithmetic — Integer Addition and Negative Numbers

- Adding binary numbers works just like adding base-10 numbers — work from right to left, carry as needed. (Example.)
- Two’s complement representation of negative numbers is chosen so that we easily get 0 when we add $-n$ and n.

  Computing $-n$ is easy with a simple trick: If m is the number of bits we’re using, addition is in effect modulo 2^m. So $-n$ is equivalent to $2^m - n$, which we can compute as $((2^m - 1) - n) + 1)$.
- So now we can easily (?) do subtraction too — to compute $a - b$, compute $-b$ and add.

Machine Arithmetic — Integer Multiplication and Division

Slide 11

- Hardware to multiply and divide basically follows the procedures humans can do on paper — multiply by computing and adding “partial sums”, divide via long division.
- (Details can get a little tricky, but basic idea is straightforward extrapolation from how it works in base 10.)

Binary Fractions

Slide 12

- We talked about integer binary numbers. How would we represent fractions?
- With base-10 numbers, the digits after the decimal point represent negative powers of 10. Same idea works in binary.

Slide 13

Computer Representation of Real Numbers

- How are non-integer numbers represented? usually as *floating point*. "IEEE 754 standard" spells out details; most current hardware implements it.
- Idea is similar to scientific notation — represent number as a binary fraction multiplied by a power of 2:

$$x = (-1)^{sign} \times (1 + frac) \times 2^{bias+exp}$$

  and then store $sign$ $frac$, and exp. Sign is one bit; number of bits for the other two fields varies — e.g., for usual single-precision, 8 bits for exponent and 23 for fraction. Bias is chosen to allow roughly equal numbers of positive and negative exponents.

Slide 14

Numbers in Math Versus Numbers in Programming

- The integers and real numbers of the idealized world of math have some properties not (completely) shared by their computer representations.
- Math integers can be any size; computer integers can't.
- Math real numbers can be any size and precision; floating-point numbers can't. Also, some quantities that can be represented easily in decimal can't be represented exactly in binary.
- Math operations on integers and reals have properties such as associativity that don't necessarily hold for the computer representations. (Yes, really!)

Slide 15

C and Representing Numbers — Integers

- Computer hardware typically represents integers as a fixed number of binary digits, using “two’s complement” idea to allow for representing negative numbers.
- C, like many (but not all!) programming languages bases its notion of integer data on this, but also has a notion of different types with different sizes. (A little more about this next time.)

Slide 16

C and Representing Numbers — Real Numbers

- Hardware also typically supports “floating-point” numbers, with a representation based on a base-2 version of scientific notation. This allows representing not only fractional quantities but also allows representing larger numbers than would be possible with fixed-length integers. Notice that only fractions that can be written with a denominator that’s a power of two can be represented exactly.
- Again C goes along with this and provides different “sizes” (`float` and `double`).

Slide 17

Minute Essay

- TBA