Slide 1

Administrivia

- Homework 7 on the Web. Official due date is next Wednesday, but accepted without penalty through the following Wednesday (holiday week).

  One problem has you working with strings and I think is very doable. The other is meant as an interesting example of a 2D array and I *think* is not too hard, but you might want to start early in case of questions.

Slide 2

Data Representation — "It's All Ones and Zeros"

- At the hardware level, all data is represented in binary form — ones and zeros. (Why? hardware for that is apparently simpler to build.)
- How then do we represent various kinds of data? First a short review of binary numbers . . .

Slide 3

Binary Numbers

- Humans usually use the decimal (base 10) number system, but other (positive integer) bases work too. (Well, maybe not base 1.)
- 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.
- Everything in base 2 works the same as base 10, if you think about how base 10 actually works, so to speak:

  In any base, digits represent increasing powers of the base (so, 1s, 10s, 100s, 1000s, etc., for base 10, and 1s, 2s, 4s, 8s, etc., for 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? Two algorithms 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.
- In somewhat sloppy pseudocode (letting n be the number we want to convert):

  while $(n > 0)$

  find largest p such that $2^p \leq n$

  write a 1 in the p-th output position

  subtract 2^p from n

  end while

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):

  while $(n > 0)$

  divide n by 2, giving quotient q and remainder r

  write down r

  set n equal to q

  end while

  (Again, this is a bit sloppy.)

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).

Slide 8

Computer Representation of Integers

- So now you can probably guess how non-negative integers can be represented using ones and zeros — 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 9

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. (This simplifies one part of processor design — more in Computer Design.)

Slide 10

Binary Fractions

- 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 11

Computer Representation of Real Numbers

- How are non-integer numbers represented? usually as *floating point*.
- 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.
- Current most common format — "IEEE 754". Read up on it sometime (Wikipedia article seems okay) — *lots* of "who knew?" details!

Slide 12

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 in binary.
- Math operations on integers and reals have properties such as associativity that don't necessarily hold for the computer representations. (Yes, really!)
- (Two "floating point is strange" examples.)

Slide 13

Multi-Dimensional Arrays in C, Revisited

- Multi-dimensional arrays in C are apt to be kind of messy:
- Fixed-size arrays work fine but are kind of inflexible.
- VLAs work well but have limitations: Size is often limited, and if you try to make one too big, the program is apt to fail without a clear warning of what's wrong. Also you can't (easily) return them from functions.
- Possibly best choice for large arrays is to represent, e.g., a 2D array as an array of pointers to the base type. Two ways to set that up . . .

Slide 14

Multi-Dimensional Arrays in C — Dynamically Allocated

- One way is to first make an array of pointers (by calling `malloc`), then fill it with pointers to rows, each obtained with a call to `malloc`. Freeing everything is a bit tedious but not hard.
- Another way is to make an array of pointers (with `malloc`), then allocate one big space for the data (as in Homework 6) and set elements of array of pointers to point into it. Freeing everything is then not too hard.
- (Example.)

Slide 15

Minute Essay

- Homework 7 asks you to complete a program to play Conway's Game of Life. Have you heard of this game?
- I don't have a definite plan for the next two classes (except to do evaluations the last class), but some things we could look at are multithreading (OpenMP and/or Pthreads) or text-mode full-screen processing with `ncurses`. Or — other requests?