Slide 1

Administrivia

- Reminder: Homework 1 due today.
- (Most?) sample programs from class will be on the Web, sometime after class.

Slide 2

Minute Essay From Last Lecture

- A few people had questions about `vi`. I'm hoping the homework has cleared those up(?).
- Several people wanted a bit of review of binary versus decimal. Coming up . . .

Slide 3

Binary Numbers (Review)

- 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.

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:
  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.)

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

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 doesn’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.

Slide 10

Machine Arithmetic — Integer Multiplication

- Multiplying binary numbers also works just like multiplying base-10 numbers — for each digit of the second operand, compute a partial result, and add them.
- (This can get slightly tricky, when adding more than two partial results involves carrying, but basic idea is straightforward extrapolation from how it works in base 10.)

Slide 11

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 12

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.

Slide 13

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 14

Scala and Representing Numbers — Review/Recap

- Computer hardware typically represents integers as a fixed number of binary digits, using "two's complement" idea to allow for representing negative numbers. Scala, 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 (e.g., `Int` versus `Long`).
- 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 Scala goes along with this and provides two different "sizes" (`Float` and `Double`).

Slide 15

Text Data

- Remember that computers represent everything using ones and zeros. How do we then get text? well, we have to come up with some way of "encoding" text characters as fixed-length sequences of ones and zeros — i.e., as small(ish) numbers.
- Several different encodings have been used over the years. One of the earliest schemes was ASCII, which uses 7 bits. That allows for 2^7 (128) characters, which is plenty for numbers, the Roman alphabet, and punctuation and other special characters. Great for English speakers, not so much for others. Unicode originated as a 16-bit encoding, which was thought to be plenty. That turned out not to be true, so Unicode is evolving. (Skim the Wikipedia article to get a sense of what issues are involved.)
- Programming languages make different choices about how to represent characters. Scala's `Char` type is 16-bit Unicode. (Some older languages use ASCII instead.) Single-character literals use single quotes.

Slide 16

Text Data, Continued

- Something else that's needed often enough to be discussed at this point — "strings" of characters.
- Again different programming languages make different choices, but most represent string literals using text contained in double quotes.

  Of course you might then ask how you put a double-quote character in a string! The answer — "escape characters". Described in more detail in textbook.
- Unlike the other types we've talked about (and booleans, which we haven't but which are described in the book), strings are *not* fixed in size. That sort of leads into the next topic . . .

Slide 17

Objects and Methods

- Text strings don't really correspond to anything the hardware can work with as directly as it works with integer and floating-point numbers. So how to represent them is left somewhat more to the discretion of the programming language. They're a simple example of a kind of thing we might want to be able to work with that's somewhat more complicated than what the hardware provides.
- To make working with things other than simple numbers easy, Scala, again like many (but not all!) programming languages has a notion of *objects* (i.e., it is an *object-oriented* (OO) language).
- Remember that we defined a type as a set of values together with some operations on them? In OO-speak, an *object* is something with a value of a particular type, and its *methods* are operations that the type says can be done on it (e.g., arithmetic operations on integers).

Slide 18

Objects and Methods in Scala

- In Scala (unlike some other popular programming languages), everything is an object. This makes some things very convenient (though it puts a certain distance between the language and the hardware, which *may* have negative effects on performance).
- Some operations on objects just do something, without any need for more information (e.g., `toInt` converts a `Double` to an `Int`). Others requre *parameters* (e.g., integer addition).
- Basic syntax for invoking an object's methods requires a period, the name of the method, parentheses, and any parameters. Scala allows many of these to be omitted if it can figure out what you mean. (Indeed, some methods that take no parameters must *not* be followed by parentheses.)

Slide 19

Objects and Methods in Scala

- Many useful "library" methods built into the language. The REPL provides some support in the form of tab completion. (Try some things with integers and strings!)
- Library methods include many for working with text strings, plus the `math` object. See book for details.

Slide 20

Variables

- We know enough — more than enough — at this point to use the Scala REPL as a calculator. But that's not really programming, since if we want to do the same calculation for different sets of values we'd have to retype everything.
- To do almost anything interesting, we need some way to save values and give them names, so we can reference them again. So Scala, like most programming languages, has a notion of *variables*, similar (but not identical!) to variables in math. (The biggest difference is that some Scala variables can take on different values as a calculation proceeds.)
- (To be continued.)

Slide 21

Minute Essay

- What is 110_2 in base 10?
- What's the (base 10) value of the largest number you can represent with 4 bits? (E.g., the largest number you can represent with 2 bits is 11_2, or 3_{10}.)

Slide 22

Minute Essay Answer

- 110_2 is 6_{10}.
- 15 (1111_2).