Discrete Structures

Discrete — defined for a finite or countable set of values

Structure — something made up of a number of parts that are held or put together in a particular way

So...

Discrete structure — something made of multiple parts, where each part comes from a finite set

Discrete Structure in Computers

— Computers manipulate bits – a finite set! – {0,1}.

— Anything defined as a sequence of bits is a discrete structure. (`long`, `int`, `char`, `byte`, `double`, etc.)

— Anything defined as a collection of these types is a discrete structure.

```
public class Point { int x, y; }
```

— Programs are discrete structures. Finte sets = language keywords, primitive types, etc.

Discrete Structure in Computers

— Computers are layers upon layers of discrete structures.

— Need more experience reasoning about things defined this way.

This class

- Practice thinking about & manipulating discrete structures.
  - Computer science topics: numeric representation, combinational circuits, finite automata.
  - Discrete mathematics topics: sets, logic and proof, functions and relations, sequences and recursion, counting techniques.
- Goal: develop language, notation, basic concepts.

Example: Interpreting binary numeric data

Most data processed by computers is numeric.

If we have 32 bits and want to interpret numerically.

$$b_1 b_2 b_3 \cdots b_{32}$$

the three most common ways to do so are
- `unsigned int` (C/C++): Only non-negative integers
- `int`: Allows negative integers
- `float`: Real numbers

Example: Unsigned integers

When you first learned about binary...

$$b_1 b_2 b_3 \cdots b_{32}$$

Each bit position has a power-of-two place value.

$$\text{value} = (b_1)(2^{31}) + (b_2)(2^{30}) + (b_3)(2^{29}) + \cdots + (b_{32})(2^0)$$

Example: Signed integers

To get negative values, make most significant bit place value *negative*.

$$b_1 b_2 b_3 \cdots b_{32}$$

$$\text{value} = -(b_1)(2^{31}) + (b_2)(2^{30}) + (b_3)(2^{29}) + \cdots + (b_{32})(2^0)$$

This representation is called *32-bit two's complement*.
(Must say number of bits to know which bit is negtive.)

Example: Two's complement

Java `byte` is 8-bit two's complement. Given `11001010` what's it's value interpreted as a `byte`?

Letting $b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8 = 11001010$, then the value

$$
\begin{aligned}
&= -(b_1)(2^7) + (b_2)(2^6) + (b_3)(2^5) + (b_4)(2^4) + (b_5)(2^3) + (b_6)(2^2) + (b_7)(2^1) + (b_8)(2^0) \\
&= -(1)(2^7) + (1)(2^6) + (0)(2^5) + (0)(2^4) + (1)(2^3) + (0)(2^2) + (1)(2^1) + (0)(2^0) \\
&= -2^7 + 2^6 + 2^3 + 2^1 \\
&= -128 + 64 + 8 + 2 = -54
\end{aligned}
$$

Range: `10000000` ... `01111111` = -128 ... 127.

Example: 32-bit IEEE-754 Floating Point

Like scientific notation: sign, exponent, significand.

$$b_1 \quad | \quad b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9 \quad | \quad b_{10} b_{11} \ldots b_{32}$$

Let
$s = b_1$ (sign bit)
$e = b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9$ interpreted excess-127 (exponent)
$f = b_{10} b_{11} \ldots b_{32}$ as a fixed point fraction (significand)

$$(-1)^s \times (1 + f) \times 2^e$$

Excess notation

A less common way to represent negative integers.

— Interpret the bits as an unsigned integer.
— Subtract the "excess".

Exmaple: 11001010 in excess-127.
11001010 is 202 as an unsigned integer.
Subtract 127 yields 75.

Fixed point fraction

— In decimal numbers, numbers right of decimal point have place values $\frac{1}{10}$, $\frac{1}{100}$, $\frac{1}{1000}$, etc.

— The same can be done in binary. Numbers right of decimal point have place values $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, etc.

— Bits for f should be interpreted like fractional part. 10011 ==> 1/2 + 1/16 + 1/32 ==> $f = 19/32$.