11 Floating-point numbers

Fractional numbers in C are represented in *floating-point.* The IEEE standard provides for *single precision* or 32 bits, and *double precision* or 64 bits.

In C, `float` means single precision floating-point, and `double` (which is sufficiently accurate for most purposes) is 64 bits.

A single precision floating point number is stored in 32 bits as follows.

- The high-order (leftmost) bit defines the *sign*: 0 for positive and 1 for negative. This is *completely different* from 2s complement integers. Let the sign be s ($s = \pm 1$).
- The next 8 bits define the *exponent.* It can be negative, but is not in 2s-complement form: rather, it is *biased.* If the face value is x, then the true value is $x - 127$. Let the true exponent be e.
- The last 23 bits define the *mantissa* between 1.0 and 2 (assuming the number is nonzero: zero is represented by 32 zero-bits).

  For nonzero numbers, If $b_1 b_2 \ldots b_{23}$ are the mantissa bits, then the true mantiissa is

  $$1.b_1 b_2 \ldots b_{23}$$

  Call it m.
- On Intel processors (used in Maths), the four bytes are stored *little endian*.
- Finally: the number represented is

  $$s \times m \times 2^e.$$
- **Rounding.** In deriving the floating-point representation of a number x, the low-order bit of the mantissa should be *rounded*, not truncated.

Example. Calculate the floating binary representation of $80/9$. Divide by 2^3 to get

$$10/9$$

This is between 1 and 2, so the exponent is 3 and the mantissa is $10/9$.

Biased exponent: add 127.

```
0111 1111
       11
1000 0010
```

j	r_j	$2r_j$	b_{j+1}
0	$\frac{1}{9}$	$\frac{2}{9}$	0
1	$\frac{2}{9}$	$\frac{4}{9}$	0
2	$\frac{4}{9}$	$\frac{8}{9}$	0
3	$\frac{8}{9}$	$\frac{16}{9}$	1
4	$\frac{7}{9}$	$\frac{14}{9}$	1
5	$\frac{5}{9}$	$\frac{10}{9}$	1
6	$\frac{1}{9}$	$\frac{2}{9}$	0

This is a point of recurrence, and the pattern will repeat. Therefore

$$\frac{10}{9} = 1.000111000111000111\ldots$$

It can be checked by summing a geometric series

$$\begin{aligned}
1 + \frac{7}{64} + \frac{7}{64^2}\cdots \\
= 1 + \frac{7}{64}\left(\sum_{j\geq 0}\frac{1}{64^j}\right) \\
= 1 + \frac{7}{64}\left(\frac{1}{1-1/64}\right) = \\
1 + \frac{7}{64}\frac{64}{63} = \\
1 + 1/9 = 10/9.
\end{aligned}$$

Now for the rounding.

$$\frac{10}{9} = 1.0001\ 1100\ 0111\ 0001\ 1100\ 011\ |\ 1$$

round

$$1.0001\ 1100\ 0111\ 0001\ 1100\ 100$$

Combining all of this:

```
0  1000 0010    0001 1100 0111 0001 1100 100
0100 0001 0000 1110 0011 1000 1110 0100
4       1    0 e    3    8    e    4

e4 38 0e 41 .... little endian
```

11.1 Double precision

The double-precision layout is, briefly,

1+11+52, bias 1023

(and little endian).

Example. $-5/1152 = -5/(9 \times 128)$.

- Sign bit 1.
- Normalise: mantissa becomes $10/9$, same as before.
- Exponent:

$$-5/1152 = -\frac{10/9}{256}$$

and $256 = 2^8$, so the exponent is -8.

- It is easier to compute the biased exponent directly in binary

```
  0 1111111111
-        1000
  0 1111110111
```

- `000111 000111 000111 000111 000111 000111 000111 000111 0001 : 11 000`

  Round up

  ```
  000111 000111 000111 000111 000111 000111 000111 000111 0010
  ```

Putting together

```
1 01111110111 000111 000111 000111 000111 000111 000111 000111 000111 001
1011111101110001110001110001110001110001110001110001110001110010
1011 1111 0111 0001 1100 0111 0001 1100
   b    f    7    1    c    7    1    c
0111 0001 1100 0111 0001 1100 0111 0010
   7    1    c    7    1    c    7    2

Little endian 72 1c c7 71 1c c7 71 bf
```