Floating Point Numbers

Floating point numbers model a (tiny finite) subset of reals;
• almost all real values don’t have exact representation (e.g. 1/3)
• numbers close to zero have higher precision (more accurate)

C has two floating point types
• float ... typically 32-bit quantity (lower precision, narrower range)
• double ... typically 64-bit quantity (higher precision, wider range)

Literal floating point values: 3.14159, 1.0/3, 1.0e-9

```c
printf("%10.4lf", (double)2.718281828459);
// displays 2.7183
printf("%20.20lf", (double)4.0/7);
// displays 0.57142857142857139685
```
Floating Point Numbers

IEEE 754 standard ...

- scientific notation with *fraction* $F$ and *exponent* $E$
- numbers have form $F \times 2^E$, where both $F$ and $E$ can be -ve
- $\text{INFINITY} =$ representation for $\infty$ and $-\infty$ (e.g. $1.0/0$)
- $\text{NAN} =$ representation for invalid value (e.g. $\sqrt{-1.0}$)

Fraction part is *normalised* (i.e. $1.2345 \times 10^2$ rather than $123.45$)

In binary, exponent is represented relative to a bias value $B$

- if the unsigned exponent value is $e$, the actual value is $e - B$
Example of normalising the fraction part in binary:

- 1010.1011 is normalized as \(1.0101011 \times 2^{011}\)
- 1010.1011 = 10 + 11/16 = 10.6875
- 1.0101011 \times 2^{011} = (1 + 43/128) \times 2^3 = 1.3359375 \times 8 = 10.6875

The normalised fraction part always has 1 before the decimal point.

Example of determining the exponent in binary:

- Assume an 8-bit exponent, then bias \(B = 2^{8-1} - 1 = 127\)
- Valid bit patterns for exponent 00000001 .. 11111110
- Exponent values -126 .. 127
Floating Point Numbers

Internal structure of floating point values

![Diagram of single precision floating point number representation]

![Diagram of double precision floating point number representation]

More complex representation than int because 1.\textit{dddddedd}
Floating Point Numbers

Example (single-precision):

\[
150.75 = 10010110.11
\]

// normalise fraction, compute exponent
= 1.001011011 * 2 ** 7

// determine sign bit,
// map fraction to 24 bits,
// map exponent relative to baseline
= 0 100000110 001011011 0000000000000000

where red is sign bit, green is exponent, blue is fraction

Note: \( B=127, \ e = 2^7 \), so exponent = = 134 = 10000110
Exercise: Floating point → Decimal

Convert the following floating point numbers to decimal. Assume that they are in IEEE 754 single-precision format.

0 10000000 11000000000000000000000
1 01111110 100000000000000000000000