Floating point data

Floating point values can represent fractional entites (unlike *int)* and are thus quite useful in scientfic computations.

There are two types floating point variables in C.

float	32 bits
double	64 bits

Example

```
float a;
double c;
```

Floating point constants can be expressed in two ways

Decimal number	1024.123
Scientific notation	1.024123e+3

```
avogadro = 6.02214199e+23;
```

Format codes for floating point output:

%f	single precision floating point number in standard notation
%lf	double precsion floating point number in standard notation
%e	single precision in scientific notation
%le	double precision in scientific notation
%g	single precision with auto notation selection
%lg	double precision with auto notation selection

Length and precision modifiers:

`%9.3f` means a field width of 9 with 3 digits to the right of the decimal

Floating point input

```
fscanf(stdin, “%f”, &fpvalue);
```

works fine for standard and scientific notation with arbitrary number of digits to the right of the decimal.

Internal representation of floating point numbers

IEEE (Institute of Electrical and Electronics Engineers) has produced a standard for floating point arithmetic. This standard specifies how single precision (32 bit) and double precision (64 bit) floating point numbers are to be represented, as well as how arithmetic should be carried out on them.

Single Precision

The IEEE single precision floating point standard representation requires a 32 bit word, which may be represented as numbered from 0 to 31, left to right.

- The first bit is the sign bit, S,
- the next eight bits are the exponent bits, 'E',
- and the final 23 bits are the fraction 'F':

```
S EEEEEEEE FFFFFFFFFFFFFFFFFFFFFFF
0 1      8 9                    31
```

The value V represented by the word may be determined as follows:

- If E=255 and F is nonzero, then V=NaN ("Not a number")
- If E=255 and F is zero and S is 1, then V=-Infinity
- If E=255 and F is zero and S is 0, then V=Infinity
- If 0<E<255 then V=(-1)**S * 2 ** (E-127) * (1.F) where "1.F" is intended to represent the binary number created by prefixing F with an implicit leading 1 and a binary point.
- If E=0 and F is nonzero, then V=(-1)**S * 2 ** (-126) * (0.F) These are "unnormalized" values.
- If E=0 and F is zero and S is 1, then V=-0
- If E=0 and F is zero and S is 0, then V=0

Specific examples

```
  0 00000000 00000000000000000000000 = 0
  1 00000000 00000000000000000000000 = -0

  0 11111111 00000000000000000000000 = Infinity
  1 11111111 00000000000000000000000 = -Infinity

  0 11111111 00000100000000000000000 = NaN
  1 11111111 00100010001001010101010 = NaN

  0 10000000 00000000000000000000000 = +1 * 2**(128-127) * 1.0 = 2
  0 10000001 10100000000000000000000 = +1 * 2**(129-127) * 1.101 = 6.5
  1 10000001 10100000000000000000000 = -1 * 2**(129-127) * 1.101 = -6.5

  0 00000001 00000000000000000000000 = +1 * 2**(1-127) * 1.0 = 2**(-126)
  0 00000000 10000000000000000000000 = +1 * 2**(-126) * 0.1 = 2**(-127)
  0 00000000 00000000000000000000001 = +1 * 2**(-126) *
                                       0.00000000000000000000001 =
                                       2**(-149)  (Smallest positive value)
```

Floating point versus integer

- Integer values are equally spaced on the number line.
- The distance between any two adjacent ints is 1

This is not true of floating point

- One half of *all* the diffrent floating point numbers lie between 0 and 1!
- The largest positive int is 2^31 – 1.
- The largest postitive float is on the order of 2^127
- Floats even smaller than the largest positive int have greater than integral spacing!

```
#include <stdio.h>

int main()
{
   float x;
   float y;


   x = 123456789;
   y = 123456791;

   printf("%12.0f %12.0f \n", x, y);
}
 ==> ./a.out
   123456792    123456792
```

This problem occurs because while positive ints have 31 significant bits, *floats* have only 32-9 = 23.

Double precision

The problem can be rectified by the use of *double precision* variables.

They are declared as follows:

```
double x;
```

The are read or printed using the *%lf, %le, or %lg f*ormat codes.

It is *very important* the the %lf family of format codes be used with and *only with* doubles.