5 Binary Data

5.1 Numeric Limit

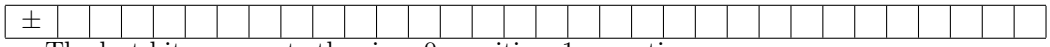
- Various number types have different memory requirements
  - More precision requires more bytes of memory

5.1.1 sizeof

"**sizeof()**" returns the size of data type in byte.

```
std::cout << "sizeof(char)=" << sizeof(char) << std::endl;
std::cout << "sizeof(short)=" << sizeof(short) << std::endl;
std::cout << "sizeof(int)=" << sizeof(int) << std::endl;
std::cout << "sizeof(long)=" << sizeof(long) << std::endl;
std::cout << "sizeof(float)=" << sizeof(float) << std::endl;
std::cout << "sizeof(double)=" << sizeof(double) << std::endl;
std::cout << "sizeof(long double)=" << sizeof(long double) << std::endl;
```

5.1.2 Integer (int) 4 bytes = 8×4=32bits



The last bit represents the sign; 0: positive, 1: negative.
$2^{31} = 2147483648$, therefore, **int** can represent -2147483648 to 2147483647.
$(0)_{10} = 00000000000000000000000000000000$
$(1)_{10} = 00000000000000000000000000000001$
$(-1)_{10} = 10000000000000000000000000000001$
$(2147483647)_{10} = 01111111111111111111111111111111$
$(-2147483648)_{10} = 10000000000000000000000000000000$

5.1.3 IEEE754 (Standard for Floating Point Arithmetic)

Floating Point Representation:

- Scientific Notation -1.23456×10^7 : [sign][significand]x[base=10]^[exponent]

Binary Floating Point Representation

- -1.001010×2^3 : [sign][significand]x[base=2]^[exponent]
- $\pm b_0.b_{-1}b_{-2}b_{-3}b_{-4}b_{-5}b_{-6}\cdots b_{-(p-1)} \times 2^E$

5.1.4 IEEE754 single precision (float)

4 bytes = 32 bits

1bit	8bits	23bits
s	e	f
sign	exponent	fraction

The fraction is digits after decimal point, f = $b_{-1}b_{-2}b_{-3}b_{-4}b_{-5}\cdots b_{-23}$.
$b_0 = 1$ for all numbers except **e=0**. b_0 is the hidden bit, called **normalized**, and not stored.
Base 10 number : $(-1)^s \times \left(1 + b_{-1} \times 2^{-1} + b_{-2} \times 2^{-2} + b_{-3} \times 2^{-3} + b_{-4} \times 2^{-4} + b_{-5} \times 2^{-5} + \cdots + b_{-23} \times 2^{-23}\right) \times 2^{e-127}$
Since the fraction is represented by 23bits, the precision is approximately 7 decimal digits ($\log_{10}\left(2^{23}\right) = 6.92\cdots$)

5.1.5 IEEE754 double precision (double)

8 bytes = 64 bits

1bit	11bits	52bits
s	e	f
sign	exponent	fraction

The fraction is digits after decimal point, $\mathrm{f} = b_{-1}b_{-2}b_{-3}b_{-4}b_{-5}\cdots b_{-52}$.

$b_0 = 1$ for all numbers except **e=0**. b_0 is the hidden bit, called **normalized**, and not stored.

Base 10 number : $(-1)^s \times \left(1 + b_{-1} \times 2^{-1} + b_{-2} \times 2^{-2} + b_{-3} \times 2^{-3} + b_{-4} \times 2^{-4} + b_{-5} \times 2^{-5} + \cdots + b_{-52} \times 2^{-52}\right) \times 2^{e-1023}$

Since the fraction is represented by 52bits, the precision is approximately 15 decimal digits ($\log_{10}\left(2^{52}\right) = 15.65\cdots$)

5.1.6 Special Numbers

- zero = | ± | 00000000 | 00000000000000000000000 | s=0or1, e=0000, f=0000 (all bits are 0 (off))
- $\pm\infty$ infinity = | ± | 11111111 | 00000000000000000000000 | s=0or1, e=11111 (all bits are 1 (on)), f=000000 (all 0)
- NaN, not a number = | ± | 11111111 | non zero | s=0or1, e=11111 (all bits are 1). f=non-zero

5.1.7 Smallest Positive Number

- **Denormalized numbers**

  The number representations described above are called **normalized**, meaning that the implicit leading hidden binary digit is a 1. To reduce the loss of precision when an underflow occurs, IEEE 754 includes the ability to represent fractions smaller than are possible in the normalized representation, by making the hidden leading digit a 0. Such numbers are called **denormal**. A denormal number is represented with a exponent of all 0 bits, which represents an exponent of −126 in single precision (not −127), or −1022 in double precision (not −1023).

- Single Precision

  $N_{min} = 2^{-23} \times 2^{-126} = 2^{-149} \sim 1.4012985{\times}10^{-45}$

- Double Precision

  $N_{min} = 2^{-52} \times 2^{-1022} = 2^{-1074} \sim 4.940656458412465 \times 10^{-324}$

5.1.8 Largest Positive Number

- Single Precision

  $N_{max} = 1.1111...1 \times 2^{emax-127} = (2 - 2^{-23}) \times 2^{127} \sim 3.4 \times 10^{38}$

- Double Precision

  The range of positive number is from

  $N_{max} = 1.1111...1 \times 2^{emax-1023} = (2 - 2^{-52}) \times 2^{1023} \sim 1.8 \times 10^{308}$

5.2 Bits / Binary

The byte is a unit of digital information in computing and telecommunications that most commonly consists of 8 bits. Historically, the byte was the number of bits used to encode a single character of text in a computer and for this reason it is the smallest addressable unit of memory in many computer architectures. 8 bits binary digits can represent decimal values of 0 through 255.

A bit is the basic unit of information in computing and digital communications. A bit can have only one of two values, (yes/no, on/off, true/false). The most common representation of these values are 0 and 1. The term bit is a contraction of binary digit.

Computer use binary arithmetic two values "0" and "1". In the 8-bit ASCII code, a lowercase "a" is represented by the bit string "01100001". Since binary digits are too long, Hexadecimal (base 16) digits are usually used.

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Binary	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111	10000
Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10

5.2.1 Conversion from binary to decimal (base 10) number,

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

$= b_0 \times 2^0 + b_1 \times 2^1 + b_2 \times 2^2 + b_3 \times 2^3 + b_4 \times 2^4 + b_5 \times 2^5 + b_6 \times 2^6 + b_7 \times 2^7$

- For example, 255=11111111, which is FF in base 16.

5.2.2 Binary Values after Decimal Point

b_3	b_2	b_1	b_0	b_{-1}	b_{-2}	b_{-3}	b_{-4}

$= b_0 \times 2^0 + b_1 \times 2^1 + b_2 \times 2^2 + b_3 \times 2^3 + b_{-1} \times 2^{-1} + b_{-2} \times 2^{-2} + b_{-3} \times 2^{-3} + b_{-4} \times 2^{-4}$

- For example, 14.75=1110.11

Base 10	Base 2
0.5	0.1
0.25	0.01
0.125	0.001
0.0625	0.0001
0.03125	0.00001
0.015625	0.000001
0.0078125	0.0000001
0.00390625	0.00000001
0.001953125	0.000000001
0.0009765625	0.0000000001

It is impossible to represent base 10 numbers with base 2 exactly, for example, 0.1=0.0001100110..... irrational number!! (causing **rounding errors**)

5.2.3 K byte, M byte, T byte, P byte ...

- K byte(Kilo Byte) = 2^{10}= 1024 $\sim 10^3$bytes
- M byte(Mega Byte) = 2^{20}= 1048576 $\sim 10^6$bytes
- G byte(Giga Byte) = 2^{30}= 1073741824 $\sim 10^9$bytes
- T byte(Tera Byte) = 2^{40}= 1099511627776 $\sim 10^{12}$bytes
- P byte(Peta Byte) = 2^{50}= 1125899906842624 $\sim 10^{15}$bytes
- E byte(Exa Byte) = 2^{60}= 1.152921504606847e+18 $\sim 10^{18}$bytes