Pat O'Sullivan

Mh4718 Week 2

Week 2

0.0.0.1 Scientific Notation. Scientific notation, also known as standard form or as exponential notation, is a way of writing numbers that accommodates values too large or small to be conveniently written in standard decimal notation. Scientific notation has a number of useful properties and is often favored by scientists, mathematicians and engineers, who work with such numbers.
In scientific notation all numbers are written in the form:

$$a \times 10^b$$

where the *exponent* b is an integer, and the coefficient a is any real number (but see normalized notation below), called the *significand* or *mantissa* If the number is negative then a minus sign precedes a (as in ordinary decimal notation).

Any given number can be written in the form of $a10^b$ in many ways; for example 350 can be written as 3.5×10^2 or 35×10^1 or 350×10^0

In normalized scientific notation, the exponent b is chosen such that $1 \leq |a| < 10$. For example, 350 is written as 3.5×10^2. This form allows easy comparison of two numbers of the same sign in a, as the exponent b gives the number's order of magnitude.
Note that 0 itself cannot be written in normalised scientific notation since the mantissa would have to be zero and the exponent undefined.

Ordinary decimal notation	Scientific notation (normalized)
300	3×10^2
4,000	4×10^3
5,720,000,000	5.72×10^9
0.0000000061	6.1×10^9

Example 0.1

Base ten:

- $2318 = 2.318 \times 10^3$.
  2.318 is the *mantissa* and 3 is the *exponent*.
- $0.2318 = 2.318 \times 10^{-1}$
  2.318 is the *mantissa* and -1 is the *exponent*.
- $0.002318 = 2.318 \times 10^{-3}$
  2.318 is the *mantissa* and -3 is the *exponent*.

Base two:

- $111.01011 = 1.1101011 \times 10^{10}$.
  1.1101011 is the *mantissa* and 10 is the *exponent*.
  Don't forget that this is base two representation and so the base and exponent (denoted as 10) signifies two.
- $0.11101011 = 1.1101011 \times 10^{-1}$.
  1.1101011 is the *mantissa* and -1 is the *exponent*.
  Again this is base two representation and so 10 signifies two.
- $0.0011101011 = 1.1101011 \times 10^{-11}$.
  1.1101011 is the *mantissa* and -11 (that is -3 in base ten) is the *exponent*.

Note that no matter what base we are using in standardised scientific notaion the format is always: $mantissa \times 10^{exponent}$. The base always appears as 10. For this reason we frequently use a mixed notation when writing base two floating point numerals. This is a bit illogical but is actually clearer.

Example 0.2

- If 111.01011 is a base two numeral we sometimes write:

$111.01011 = 1.1101011 \times 2^2$ instead of 1.1101011×10^{10}.

- If 0.11101011 is a base two numeral we sometimes write:
  $0.11101011 = 1.1101011 \times 2^{-1}$ instead of $0.11101011 = 1.1101011 \times 10^{-1}$.

- If 0.0011101011 is a base two numeral we sometimes write:
  $0.0011101011 = 1.1101011 \times 2^{-3}$ instead of $0.0011101011 = 1.1101011 \times 10^{-11}$.

The easiest way to add two numbers in normalised scientific notation is to replace the smallest exponent (it the exponents are different) with the largest exponent and then add the mantissas.

Example 0.3

Base ten:

$$\begin{aligned} 8.237 \times 10^4 + 9.63 \times 10^2 &= 8.237 \times 10^4 + 0.0963 \times 10^4 \\ &= (8.237 + 0.0963) \times 10^4 = 8.3333 \times 10^4 \end{aligned}$$

Base two:

$$\begin{aligned} 1.1011 \times 10^{-3} + 1.1101 \times 10^2 &= 0.0000011011 \times 10^2 + 1.1101 \times 10^2 \\ &= (0.0000011011 + 1.1101) \times 10^2 = 1.1101011011 \times 10^2 \end{aligned}$$

0.0.1 Significant digits

The significant digits (also called significant figures and abbreviated sig figs, sign.figs, sig digs or s.f.) of a number are those digits that carry meaning contributing to its precision. This includes all digits except leading and trailing zeros where they serve merely as placeholders to indicate the scale of the number.

The concept of significant digits is often used in connection with rounding. Rounding to n significant digits is a more general-purpose technique than rounding to n decimal places, since it handles numbers of different scales in a uniform way. For example, the population of a city might only be known to

the nearest thousand and be stated as 52,000, while the population of a country might only be known to the nearest million and be stated as 52,000,000. The former might be in error by hundreds, and the latter might be in error by hundreds of thousands, but both have two significant digits (5 and 2). This reflects the fact that the significance of the error (its likely size relative to the size of the quantity being measured) is the same in both cases.
The rules for identifying significant digits when writing or interpreting numbers are as follows:

- All non-zero digits are considered significant. For example, 91 has two significant digits (9 and 1), while 123.45 has five significant digits (1, 2, 3, 4 and 5).
- Zeros appearing anywhere between two non-zero digits are significant. Example: 101.12 has five significant digits: 1, 0, 1, 1 and 2.
- Leading zeros are not significant. For example, 0.00052 has two significant digits: 5 and 2.
- Trailing zeros in a number containing a decimal point are significant. For example, 12.2300 has six significant digits: 1, 2, 2, 3, 0 and 0. The number 0.000122300 still has only six significant digits (the zeros before the 1 are not significant). In addition, 120.00 has five significant digits. This convention clarifies the precision of such numbers; for example, if a result accurate to four decimal places is given as 12.23 then it might be understood that only two decimal places of accuracy are available. Stating the result as 12.2300 makes clear that it is accurate to four decimal places.
- The significance of trailing zeros in a number not containing a decimal point can be ambiguous. For example, it may not always be clear if a number like 1300 is accurate to the nearest unit (and just happens coincidentally to be an exact multiple of a hundred) or if it is only shown to the nearest hundred due to rounding or uncertainty. Various conventions exist to address this issue:
  - A bar may be placed over the last significant digit; any trailing zeros following this are insignificant. For example, $13\bar{0}0$ has three significant digits (and hence indicates that the number is accurate to the nearest ten).
  - The last significant digit of a number may be underlined; for example, $2\underline{0}000$ has two significant digits.
  - A decimal point may be placed after the number; for example 100. indicates specifically that three significant digits are meant.

How many significant figures do the following numbers have?

1) 1234 1) 4

2) 0.023 2) 2

3) 890 3) 2

4) 91010 4) 4

5) 9010.0 5) 5

6) 1090.0010 6) 8

7) 0.00120 7) 3

8) 3.4 x 104 8) 2

9) 9.0 x 10-3 9) 2

10) 9.010 x 10-2 10) 4

11) 0.00030 11) 2

12) 1020010 12) 6

13) 780. 13) 3

14) 1000 14) 1

15) 918.010 15) 6

16) 0.0001 16) 1

17) 0.00390 17) 3

18) 8120 18) 3

19) 7.991 x 10-10 19) 4

20) 72 20) 2

MSExcel stores 15 significant digits.

Example 0.4

In MSExcel format a cell as a number and enter `2^50`.
The value displayed is 1125899906842620.00 and it is clear that this is not correct since 2^{50} is not divisible by 5.
In fact we can determine that the last digit of 2^{50} is 4.
However, *the value displayed is not necessarily the value stored* and we can show this hear by subtracting 1125899906842620.00 from the value in the cell which holds 2^{50}.

We can determine the number of digits in an integer using the $\log_{10}$ function. (Which we shall denote simply as log from now on.)

Recalling that $\log(10^n) = n$ and that log is an increasing function we see that, if x is a number with n digits then

$$10^{n-1} \leq x < 10^n \Rightarrow n - 1 \leq \log(x) < n$$

therefore the number of digits in x is given by $[\log(x)]+1$ where $[\log(x)]$ denotes the integer part of x.

Applying this to 2^{50} we see that

$$\log(2^{50}) = 50\log(2) \approx 15.0515$$

which tells us that 2^{50} has 16 digits which is one more than Excel can display.

0.0.2 Bits and Bytes

The reason computers use the base-2 system is because it makes it a lot easier to implement them with current electronic technology. You could wire up and build computers that operate in base-10, but they would be much more right now. On the other hand, base-2 computers are relatively cheap.

The only data that a computer can understand is on and off. But those two simple commands can be grouped into millions of combinations and it is the way they are grouped in series that creates complex data.

The basic unit is called a bit (binary digit). Each bit has an electronic switch, or gate. If the gate is open the bit is on and electricity can go through. The computer reads on or open switches as a number 1. If the gate is closed or off, the electricity is blocked and the computer reads off bits as 0. So computers use binary numbers, and therefore use binary digits in place of decimal digits.

Bits are rarely seen alone in computers. They are almost always bundled together into 8-bit collections, and these collections are called bytes. Why are there 8 bits in a byte? This is similar to the question, "Why are there 12 inches in a foot?" The 8-bit byte is something that people settled on through trial and error over the past 50 years. There are 256 possible combinations of 1/0 in a byte.

The abbreviation for bit is a lowercase "b"; the abbreviation for byte is an uppercase "B".
Bytes are then grouped together to former larger units of measurements. The larger units usually involve bundling in powers of two.

The prefixes for the multiples are based on the metric system. The nearest power of 2 to 1,000 is 2^{10} or 1,024; thus 1,024 bytes was named a *Kilobyte*. So, although a metric *kilo* equals 1,000 (e.g. one kilogram = 1,000 grams), a binary *Kilo* equals 1,024 (e.g. one Kilobyte = 1,024 bytes). Not surprisingly, this can lead to some confusion.

Binary Measurements of Capacity

Unit	**Abbreviation**	**Value**
bit	b	0 or 1
byte	B	8 bits
Kilobyte	KB	1024 bytes
Megabyte	MB	1024 Kilobytes
Gigabyte	GB	1024 Megabytes
Terabyte	TB	1024 Gigabytes
Petabyte	PB	1024 Terabytes
Exabyte	EB	1024 Petabytes

To make the situation even more confusing, transfer speeds over the Internet are usually measured in bits rather than bytes and unit groupings revert to decimal. Notice that lower case letters are used in these cases in order to differentiate from the binary byte groupings in the above table:

Measurements of transfer speeds

Unit	**Abbreviation**	**Value**
bit	b	0 or 1
byte	B	8 bits
kilobit	kb	1000 bits
Megabit	Mb	1000 kilobits
Gigabit	Gb	1000 Megabits
Terabit	Tb	1000 Gigabits
Petabit	Pb	1000 Terabits
Exabit	Eb	1000 Petabits

In December 1998, the International Electrotechnical Commission (IEC) approved a new IEC International Standard. Instead of using the metric prefixes for multiples in binary code, the new IEC standard invented specific prefixes for binary multiples made up of only the first two letters of the metric prefixes and adding the first two letters of the word “binary”. Thus, for instance, instead of Kilobyte (KB) or Gigabyte (GB), the new terms would be kibibyte (KiB) or gibibyte (GiB).

0.1 Variable Storage in C++

0.1.1 Storage of Integers

Variables of type **int** are stored using 4 bytes (i.e. 32 bits.) Positive integers are stored in straightforward base two format.

Example 0.5

If we have the line:

int n=1059;

in a C++ program then the compiler will store the value of n in 4 bytes and, since 1059 = $(10000100011)_2$, the 4 bytes will be filled as follows:

$$\overbrace{00000000}^{\text{byte 4}}\ \overbrace{00000000}^{\text{byte 3}}\ \overbrace{00000100}^{\text{byte 2}}\ \overbrace{00100011}^{\text{byte 1}}$$

0.1.1.1 The largest positive integer that can be stored.

The largest positive integer which can be stored occurs when all bits except the 32nd bit is 1. The 32nd bit must be 0 to indicate that we are storing a positive integer. Therefore the largest positive integer that can be stored is:

$$\overbrace{01111111}^{\text{byte 4}}\ \overbrace{11111111}^{\text{byte 3}}\ \overbrace{11111111}^{\text{byte 2}}\ \overbrace{11111111}^{\text{byte 1}} = 2^{30} + 2^{29} + \cdots + 2^{2} + 2^{1} + 2^{0}$$

Now since

$$\overbrace{111\ldots111}^{\text{31 ones}} = 1\overbrace{000\ldots000}^{\text{31 zeros}} - 1$$

we see that that the largest **int** value that can be stored is $2^{31} - 1$.