Précision des *float* et *double*

Code avec *numeric_limits*

```
cout << "digits10     "  << numeric_limits<float>::digits10;      // 6
cout << "max_digits10 "  << numeric_limits<float>::max_digits10;  // 9
```

=> Le compilateur retourne bien pour un environnement donné les valeurs 6 et 9 pour les floats

learncpp.com

https://www.learncpp.com/cpp-tutorial/floating-point-numbers

Assuming IEEE 754 representation:

Size	Range	Precision
4 bytes	$\pm 1.18 \times 10^{-38}$ to $\pm 3.4 \times 10^{38}$	6-9 significant digits, typically 7
8 bytes	$\pm 2.23 \times 10^{-308}$ to $\pm 1.80 \times 10^{308}$	15-18 significant digits, typically 16
80-bits (typically uses 12 or 16 bytes)	$\pm 3.36 \times 10^{-4932}$ to $\pm 1.18 \times 10^{4932}$	18-21 significant digits
16 bytes	$\pm 3.36 \times 10^{-4932}$ to $\pm 1.18 \times 10^{4932}$	33-36 significant digits

On y voit bien le « typically 7 » mais est-ce que « typiquement » suffit à dire que la valeur retenue est 7 ?

Plus loin dans ce même site au chapitre « Floating Point precision » on y lit qu'un *std::cout << 9.87654321f* va justement afficher sur 6 chiffres « 9.87654 » (1.5) précisément car le nombre de chiffres significatifs présumé et 6.

When outputting floating point numbers, std::cout has a default precision of 6 -- that is, it assumes all floating point variables are only significant to 6 digits (the minimum precision of a float), and hence it will truncate anything after that.

… et de continuer
Float values have between 6 and 9 digits of precision, with most float values having at least 7 significant digits.

cppreference.com

https://en.cppreference.com/w/cpp/types/numeric_limits/digits10

The value of std::numeric_limits<T>::digits10 is the number of base-10 digits that can be represented by the type `T` without change, that is, any number with this many significant decimal digits can be converted to a value of type `T` and back to decimal form, without change due to rounding or overflow

*The standard 32-bit IEEE 754 floating-point type has a 24 bit fractional part (23 bits written, one implied), which may suggest that it can represent 7 digit decimals (24 * std::log10(2) is 7.22), but relative rounding errors are non-uniform and some floating-point values with 7 decimal digits do not survive conversion to 32-bit float and back: the smallest positive example is 8.589973e9, which becomes 8.589974e9 after the roundtrip. These rounding errors cannot exceed one bit in the representation, and `digits10` is calculated as (24-1)*std::log10(2), which is 6.92. Rounding down results in the value 6.*

Stackoverflow

https://stackoverflow.com/questions/12815179/number-of-significant-digits-for-a-floating-point-type

According to the standard, not all decimal number can be stored exactly in memory. Depending on the size of the representation, the error can get to a certain maximum. For `float` this is `0.0001%` (6 significant digits = `10^-6` = `10^-4 %`)

6 significant digits means that the maximum error is approximately +/- 0.0001%. The single float value actually has about 7.2 digits of precision

GMB : On retrouve le 7.2 mentionné dans cppreference mais qui est passé à 6 à cause des arrondis possibles.