



C++ : Annexe 0 Combien avez-vous de doigts ?

Il est peu probable que vous utilisiez le langage C++ très longtemps sans être confronté à des nombres représentés autrement que sous leur forme habituelle. Même si les changements de représentation dont il va être question ici sont très loin d'être d'un usage quotidien, une certaine familiarité avec ces questions est certainement bénéfique (ne serait-ce que pour vous éviter d'avoir l'impression qu'on vous cache quelque chose !). Il faut même avouer qu'il n'est pas tout à fait exclu que vous soyez un jour confronté à une situation où une véritable maîtrise de ces techniques s'avère indispensable.

1 - Les êtres humains

Nous avons l'habitude d'utiliser un système numérique positionnel de base 10.

Ce système est dit "de base 10" parce qu'il utilise 10 chiffres : 0, 1, 2, 3, 4, 5, 6, 7, 8 et 9.

Il est dit positionnel parce que le "poids" d'un chiffre, c'est à dire la valeur qu'il exprime, dépend de sa position. Ainsi, dans 1999, les trois occurrences du chiffre 9 n'ont pas la même signification : il s'agit respectivement de neuf siècles, neuf décennies et neuf années.

Plus généralement, dans un système positionnel, un chiffre figurant dans un nombre a un poids égal à la base élevée à la puissance correspondant au nombre de chiffres figurant à sa droite. Illustrons ce principe en analysant un exemple :

écriture du nombre en base 10	1	9	9	9
-------------------------------	---	---	---	---

Pour chaque colonne, il est aisé de déterminer combien de colonnes figurent à sa droite :

nombre de chiffres à droite	3	2	1	0
-----------------------------	---	---	---	---

Avec ce nombre, nous pouvons calculer le poids de chaque colonne :

poids = base ^{nombre de chiffres à droite}	$10^3 = 1000$	$10^2 = 100$	$10^1 = 10$	$10^0 = 1$
---	---------------	--------------	-------------	------------

1999 veut donc, en fait, dire :

total	$(1 * 1000) + (9 * 100) + (9 * 10) + (9 * 1)$
-------	---

Il semble que l'adoption de la base 10 soit liée au fait que nous avons (normalement) cinq doigts à chaque main, soit un total de 10. Comme ces doigts peuvent facilement être utilisés pour compter (grâce à leur capacité à se plier et se déplier relativement indépendamment les uns des autres), le dénombrement jusqu'à 10 ne pose pas de gros problème. Au-delà de 10, il faut noter combien de fois on a épuisé son stock de doigts, d'où l'apparition de la notion de dizaine, puis de centaine (dizaine de dizaines), etc. Il n'en reste pas moins qu'on compte les secondes et les minutes par soixantaines, les heures par 24 et les huitres à la douzaine...

Pour ce qui est du principe positionnel, il apparaît pour la première fois à Babylone, 2000 ans avant notre ère, mais ne s'est généralisé que beaucoup plus récemment (le système romain, que nous utilisons encore dans certains cas, n'est pas positionnel). Une des difficultés soulevées par le principe de position est qu'il implique l'usage du zéro, concept qui n'est arrivé à maturité qu'en Inde, 500 ans **après** JC !

2 - Les ordinateurs

Si l'on conserve le principe positionnel tout en se limitant à l'utilisation de deux chiffres (0 et 1, par exemple), on obtient le système **binaire**. Comme dans le cas de la base 10, la représentation du premier-entier-plus-grand-que-le-plus-grand-des-chiffres nécessite le recours à une position supplémentaire. Le début de la suite des entiers est donc représenté ainsi : 0, 1, 10, 11, 100, 101...

La simple application du principe des poids permet de transformer très facilement l'écriture binaire d'un nombre en écriture en base 10. Prenons l'exemple de 1011 :

Écriture binaire du nombre	1	0	1	1
nombre de chiffres à droite	3	2	1	0
poids = base ^{nombre de chiffres à droite}	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
total (exprimé en base 10)	$(1 * 8) + (0 * 4) + (1 * 2) + (1 * 1) = 11$			

Ou, si vous préférez, une huitaine plus zéro "quatrième" plus une paire plus une unité font onze.

La transcription inverse n'est guère plus difficile : il suffit de partir de la gauche et, pour chaque colonne¹, de vérifier si le nombre restant à exprimer est ou non aussi grand que le poids. Lorsque c'est le cas, il faut mettre un 1 dans cette colonne et retrancher de la quantité restant à exprimer le poids de la colonne. Essayons d'écrire en binaire le nombre qui s'écrit 5 en base 10 :

nombre de chiffres à droite	3	2	1	0
poids = base ^{nombre de chiffres à droite}	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$

Comme 5 est inférieur à 8, la colonne de poids 8 contient 0 :

5 est inférieur à 8	0			
---------------------	---	--	--	--

Comme 5 est supérieur à 4, la colonne de poids 4 contient 1. Ce 1 représente en fait une "quatrième", ce qui veut dire qu'il nous reste 5 moins une "quatrième", c'est à dire 1.

5 est supérieur à 4 il reste à exprimer : $5 - 4 = 1$		1		
--	--	---	--	--

Le nombre qu'il nous faut maintenant exprimer (1) est inférieur à 2. La colonne des paires contient donc 0.

1 est inférieur à 2			0	
---------------------	--	--	---	--

La colonne des unités reçoit finalement 1 pour représenter la quantité qui nous reste.

1 est égal à 1				1
Écriture binaire du nombre	0	1	0	1

En d'autres termes, 5 vaut "zéro huitaine, une quatrième, zéro paire et une unité".

L'écriture binaire présente l'avantage de pouvoir être plaquée directement sur les systèmes de mémoire dont les composants ont deux états stables (cf. [Leçon 1](#)). Pour un usage humain, il faut bien reconnaître que les nombres écrits de cette façon sont peu lisibles, surtout à cause du fait qu'ils deviennent vite très longs : 1999, par exemple, s'écrit 0111 1100 1111 en binaire (il est d'usage, pour des raisons qui apparaîtront dans quelques instants, de regrouper les chiffres binaires par 4). Même en informatique, on n'utilisera donc ce type d'écriture que dans des cas très particuliers, par exemple lorsque les chiffres eux-mêmes, et non la valeur du nombre, revêtent une importance spéciale.

Le langage C++ n'autorise malheureusement pas l'usage de la représentation binaire des nombres dans les textes sources.

3 - Les programmeurs

S'il faut renoncer à la représentation binaire pour des raisons de confort, l'alternative a priori la plus séduisante est certainement l'écriture habituelle (en base 10). C'est effectivement le système qui est utilisé dans la plupart des cas. Il arrive toutefois que la représentation en base 10 s'avère peu pratique, parce qu'elle présente peu d'affinités avec la représentation binaire. Le système le plus généralement employé dans ces circonstances est la représentation **hexadécimale** (c'est à dire en base 16).

Pour pouvoir écrire les nombres en utilisant une base supérieure à 10, il faut disposer de plus de 10 symboles pour représenter les chiffres. Plutôt que d'inventer des symboles originaux, on choisit habituellement les lettres de l'alphabet, et, dans le cas de l'hexadécimal, il nous faut donc mobiliser les lettres de A à F². Un avantage évident de ce choix est que tout le monde connaît déjà l'ordre de ces symboles et est donc en mesure de reconstruire la table d'équivalences suivante :

¹ Il faut, bien entendu, utiliser un tableau ayant assez de colonnes ! (Le poids de la colonne la plus à gauche doit être au moins égal au nombre qu'il s'agit de représenter.)

² On utilise indifféremment les majuscules et les minuscules

chiffre hexadécimal	A	B	C	D	E	F
écriture en base 10	10	11	12	13	14	15

(de 0 à 9, le système hexadécimal utilise les chiffres habituels)

Le choix de lettres pour figurer les chiffres supérieurs à 9 présente aussi un aspect divertissant : certains nombres cessent de ressembler à des nombres. La représentation hexadécimale de celui que vous connaissez sous la forme 51 966 en est un exemple stimulant, et si vous rencontrez un jour 14 600 926, 16 435 934 ou 251 636 974 écrits en hexadécimal, vous aurez certainement un instant d'hésitation avant de vous convaincre qu'il s'agit bien de nombres.

La retranscription en base 10 d'un nombre écrit en hexadécimal est tout aussi facile que celle d'un nombre écrit en binaire. La seule difficulté concerne, au moment du total, la conversion des chiffres supérieurs à 9 à l'aide du tableau précédent :

Écriture hexadécimale du nombre	1	A	8	C
nombre de chiffres à droite	3	2	1	0
poids = base nombre de chiffres à droite	$16^3 = 4096$	$16^2 = 256$	$16^1 = 16$	$16^0 = 1$
total	$(1 * 4096) + (A * 256) + (8 * 16) + (C * 1)$ $= (1 * 4096) + (10 * 256) + (8 * 16) + (12 * 1)$ $= 6796$			

Une "quatre-mille-quatre-vingt-seizaine" plus dix "deux-cent-cinquante-sixaines" plus huit "seizaines" plus douze unités font six mille sept cent quatre vingt seize ?

De même, la méthode employée pour écrire un nombre en hexadécimal ne diffère de celle utilisée pour le binaire qu'en un point : lorsque la quantité restant à exprimer dépasse le poids de la colonne, ce n'est pas 1 qu'il faut placer dans cette colonne, mais le résultat de la division entière de la quantité par le poids (si ce résultat est supérieur à 9, il faudra utiliser le chiffre hexadécimal correspondant). Exprimons en hexadécimal l'année de la prise de la Bastille :

nombre de chiffres à droite	3	2	1	0
poids = base nombre de chiffres à droite	$16^3 = 4096$	$16^2 = 256$	$16^1 = 16$	$16^0 = 1$
<i>1789 est inférieur à 4096</i>	0			
$1789 / 256 = 6$		6		
<i>il reste à exprimer : $1789 - (256 * 6) = 253$</i>				
$253 / 16 = 15$			15	
<i>il reste à exprimer : $253 - (16 * 15) = 13$</i>				
$13 / 1 = 13$				13
Écriture hexadécimale du nombre	0	6	F	D

Dans un contexte informatique, la base 16 présente un intérêt parce qu'un seul chiffre hexadécimal contient exactement la même quantité d'information qu'un groupe de quatre chiffres binaires. En d'autres termes, un nombre hexadécimal à deux chiffres correspond exactement à un octet. Cette propriété³ se traduit concrètement de deux façons : le passage d'une représentation binaire à une représentation hexadécimale (ou d'une représentation hexadécimale à une représentation binaire) est très facile et, mieux encore, ce type de transcription est souvent inutile !

L'équivalence chiffre hexadécimal/nombre à quatre chiffres binaires permet de décomposer les transcriptions : pour passer d'une représentation hexadécimale à une représentation binaire, il suffit de remplacer chaque chiffre hexadécimal par les quatre chiffres binaires correspondant, et les chiffres hexadécimaux peuvent être traités indépendamment les uns des autres. Inversement, pour retranscrire un nombre exprimé en binaire en notation hexadécimale, il suffit de regrouper les chiffres binaires par paquets de 4 (en commençant par la droite, et en complétant à gauche avec des 0, si c'est nécessaire) et de remplacer chaque paquet par le chiffre hexadécimal correspondant. Les programmeurs qui font un usage intensif de ces opérations connaissent les équivalences par cœur, et les autres utilisent une table qu'il est facile de reconstituer en cas de besoin :

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Bin	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

³ Il s'agit d'une conséquence assez directe du fait que 16 est la quatrième puissance de 2.

Voici, par exemple, comment on peut transcrire en binaire le nombre qui s'écrit F3 58 A9 02 en hexadécimal :

F	3	5	8	A	9	0	2
1111	0011	0101	1000	1010	1001	0000	0010

Vous pouvez vous convaincre définitivement de la simplicité des transcriptions entre hexadécimal et binaire en essayant de retrouver l'écriture en base 10 du nombre utilisé dans l'exemple précédent. Vous pouvez partir soit de sa forme binaire (1111 0011 0101 1000 1010 1001 0000 0010), soit de sa forme hexadécimale (F3 58 A9 02), mais, si vous n'utilisez pas de calculatrice, l'effort et le temps nécessaires seront démesurés par rapport à ceux exigés par le passage de l'hexadécimal au binaire (ou inversement).

Souvent, il n'est même pas nécessaire de procéder à la moindre transcription, car les caractéristiques intéressantes sont directement visibles dans le nombre hexadécimal.

Une de ces caractéristiques importantes est le nombre de cases mémoires nécessaires pour stocker un nombre. Imaginons par exemple que nous avons à stocker une quantité entière positive dont nous savons qu'elle n'excédera jamais 16 777 507. Combien d'octets de mémoire faut-il consacrer à ce stockage ? Le simple nombre de chiffres utilisés pour écrire un nombre en base 10 ne permet pas de savoir quelle sera la longueur de sa transcription binaire (qui détermine la place occupée en mémoire). La réponse à cette question serait en revanche immédiate si la limite était exprimée en hexadécimal : 1 00 01 23 ne tient pas sur trois octets, il en faut donc en prévoir quatre⁴.

L'expressivité de l'écriture hexadécimale devient encore plus importante lorsque l'information stockée en mémoire n'a aucune signification numérique. La transcription du binaire en base 10 lui fait alors perdre certaines particularités qui sont au moins en partie conservée en hexadécimal. Prenons comme exemple la séquence binaire 101010101010101. L'alternance parfaite des 1 et des 0 y saute aux yeux. Il peut s'agir d'un pur hasard, mais il est également possible que cette régularité ait une signification. Au cours d'une opération de débogage, de tels patterns "trop beaux pour être vrais" attirent souvent l'attention et permettent parfois de trouver l'origine du problème. Si la séquence en question est présentée sous la forme d'un nombre⁵ écrit en base 10, c'est 21 845, et cette séquence de chiffres ne présente absolument rien de remarquable. Sous forme hexadécimale, c'est 55 55, et le lecteur a de bonnes chances d'y détecter une régularité (avec un peu d'habitude, l'alternance des 1 et des 0 y est même presque **plus** visible que dans la forme binaire).

La notation hexadécimale étant, dans bien des cas, la meilleure façon de "montrer le binaire", le langage C++ offre la possibilité de l'utiliser directement dans les textes sources. Il suffit qu'une séquence de chiffres soit précédée par les deux caractères "0x" (le chiffre zéro et la lettre x minuscule) pour que le compilateur interprète la séquence comme représentant un nombre en hexadécimal.

```
const int seize = 0x10; //initialise correctement la constante seize
```

4 - Conclusion

Si, à la question posée par le titre, vous n'avez répondu ni "10" ni "16", mais "0x10", félicitations ! Vos seize doigts vous permettent une maîtrise quasi-naturelle de l'hexadécimal, ce qui vous ouvre un brillant avenir dans l'industrie du logiciel. Dans le cas contraire, inutile d'apprendre le présent document par cœur, mais essayez tout de même de vous souvenir d'où vous le rangez, on ne sait jamais...

⁴ On peut même aller plus loin : le quatrième octet n'aura jamais une valeur supérieure à 1. On n'utilise donc effectivement qu'un seul de ses huit bits. Le nombre 1 00 01 23 occupe donc exactement $1 + (3 * 8) = 25$ bits.

⁵ Si le contenu de la mémoire n'a "aucune signification numérique", pourquoi serait-il présenté sous la forme d'un nombre (quelle que soit la base choisie) ? C'est une bonne question, et la seule réponse honnête me semble être : "Normalement, il ne devrait pas être représenté ainsi. Mais si vous êtes en train de déboguer, c'est justement parce que rien ne se passe normalement..." Lorsqu'on ne sait plus comment l'interpréter, présenter l'information sous forme de nombres écrits en hexadécimal est le meilleur compromis qu'on ait trouvé pour obtenir la concision qui fait cruellement défaut à la représentation binaire, sans pour autant sacrifier tout rapport direct avec la réalité du codage, comme le fait la représentation en base 10.