Préambule : Le Codage

« L'information n'est pas le savoir. Le savoir n'est pas la sagesse. La sagesse n'est pas la beauté. La beauté n'est pas l'amour. L'amour n'est pas la musique, et la musique, c'est ce qu'il y a de mieux. »
Frank Zappa

« Les ordinateurs sont comme les dieux de l'Ancien Testament : avec beaucoup de règles, et sans pitié. »
Joseph Campbell

« Compter en octal, c'est comme compter en décimal, si on n'utilise pas ses pouces »
Tom Lehrer

« Il y a 10 sortes de gens au monde : ceux qui connaissent le binaire et les autres »
Anonyme

C'est bien connu, les ordinateurs sont comme le gros rock qui tâche : ils sont **binaires**.

Mais ce qui est moins connu, c'est ce que ce qualificatif de « binaire » recouvre exactement, et ce qu'il implique. Aussi, avant de nous plonger dans les arcanes de l'algorithmique proprement dite, ferons-nous un détour par la notion de **codage binaire**. Contrairement aux apparences, nous ne sommes pas éloignés de notre sujet principal. Tout au contraire, ce que nous allons voir à présent constitue un ensemble de notions indispensables à l'écriture de programmes. Car pour parler à une machine, mieux vaut connaître son vocabulaire…

1. Pourquoi les ordinateurs sont-ils « binaires » ?

De nos jours, les ordinateurs sont ces machines merveilleuses capables de traiter du texte, d'afficher des tableaux de maître, de jouer de la musique ou de projeter des vidéos. On n'en est pas encore tout à fait à HAL, l'ordinateur de 2001 Odyssée de l'Espace, à « l'intelligence » si développée qu'il a peur de mourir… pardon, d'être débranché. Mais l'ordinateur paraît être une machine capable de tout faire.

Pourtant, les ordinateurs ont beau sembler repousser toujours plus loin les limites de leur champ d'action, il ne faut pas oublier qu'en réalité, ces fiers-à-bras ne sont toujours capables que d'une seule chose : faire des calculs, et uniquement cela. Eh oui, ces gros malins d'ordinateurs sont restés au fond ce qu'ils ont été depuis leur invention : de vulgaires calculatrices améliorées !

Lorsqu'un ordinateur traite du texte, du son, de l'image, de la vidéo, il traite en réalité des nombres. En fait, dire cela, c'est déjà lui faire trop d'honneur. Car même le simple nombre « 3 » reste hors de portée de l'intelligence d'un ordinateur, ce qui le situe largement en dessous de l'attachant chimpanzé Bonobo, qui sait, entre autres choses, faire des blagues à ses congénères et jouer au Pac-Man. Un ordinateur manipule exclusivement des **informations binaires**, dont on ne peut même pas dire sans être tendancieux qu'il s'agit de nombres.

Mais qu'est-ce qu'une information binaire ? C'est une information qui ne peut avoir que deux états : par exemple, ouvert - fermé, libre –occupé, militaire – civil, assis – couché, blanc – noir, vrai – faux, etc. Si l'on pense à des dispositifs physiques permettant de stocker ce genre d'information, on pourrait citer : chargé – non chargé, haut – bas, troué – non troué.

Je ne donne pas derniers exemples au hasard : ce sont précisément ceux dont se sert un ordinateur pour stocker l'ensemble des informations qu'il va devoir manipuler. En deux mots, la mémoire vive (la « RAM ») est formée de millions de composants électroniques qui peuvent retenir ou relâcher une charge électrique. La surface d'un disque dur, d'une bande ou d'une disquette est recouverte de particules métalliques qui peuvent, grâce à un aimant, être orientées dans un sens ou dans l'autre. Et sur un CD-ROM, on trouve un long sillon étroit irrégulièrement percé de trous.

Toutefois, la coutume veut qu'on symbolise une information binaire, quel que soit son support physique, sous la forme de 1 et de 0. Il faut bien comprendre que ce n'est là qu'une **représentation**, une image commode, que l'on utilise pour parler de toute information binaire. Dans la réalité physique, il n'y a pas plus de 1 et de 0 qui se promènent dans les ordinateurs qu'il n'y a écrit, en lettres géantes, « Océan Atlantique » sur la mer quelque part entre la Bretagne et les Antilles. Le 1 et le 0 dont parlent les informaticiens sont des signes, ni plus, ni moins, pour désigner une information, indépendamment de son support physique.

Les informaticiens seraient-ils des gens tordus, possédant un goût immodéré pour l'abstraction, ou pour les jeux intellectuels alambiqués ? Non, pas davantage en tout cas que le reste de leurs contemporains non-informaticiens. En fait, chacun d'entre nous pratique ce genre d'abstraction tous les jours, sans pour autant trouver cela bizarre ou difficile. Simplement, nous le faisons dans la vie quotidienne sans y penser. Et à force de ne pas y penser, nous ne remarquons même plus quel mécanisme subtil d'abstraction est nécessaire pour pratiquer ce sport.

Lorsque nous disons que 4+3=7 (ce qui reste, normalement, dans le domaine de compétence mathématique de tous ceux qui lisent ce cours !), nous manions de pures abstractions, représentées par de non moins purs symboles ! Un être humain d'il y a quelques millénaires se serait demandé longtemps qu'est-ce que c'est que « quatre » ou « trois », sans savoir quatre ou trois « quoi ? ». Mine de rien, le fait même de concevoir des nombres, c'est-à-dire de pouvoir considérer, dans un ensemble, la quantité indépendamment de tout le reste, c'est déjà une abstraction très hardie, qui a mis très longtemps avant de s'imposer à tous comme une évidence. Et le fait de faire des additions sans devoir préciser des additions « de quoi ? », est un pas supplémentaire qui a été encore plus difficile à franchir.

Le concept de nombre, de quantité pure, a donc constitué un immense progrès (que les ordinateurs n'ont quant à eux, je le répète, toujours pas accompli). Mais si concevoir les nombres, c'est bien, posséder un système de notation performant de ces nombres, c'est encore mieux. Et là aussi, l'humanité a mis un certain temps (et essayé un certain nombre de pistes qui se sont révélées être des impasses) avant de parvenir au système actuel, le plus rationnel. Ceux qui ne sont pas convaincus des progrès réalisés en ce domaine peuvent toujours essayer de résoudre une multiplication comme 587 x 644 en chiffres romains, on leur souhaite bon courage !

2. La numérotation de position en base décimale

L'humanité actuelle, pour représenter n'importe quel un nombre, utilise un système de **numérotation de position**, à **base décimale**. Qu'est-ce qui se cache derrière cet obscur jargon ?

Commençons par la **numérotation de position**. Pour représenter un nombre, aussi grand soit-il, nous disposons d'un **alphabet** spécialisé : une série de 10 signes qui s'appellent les chiffres. Et lorsque nous écrivons un nombre en mettant certains de ces chiffres les uns derrière les autres, l'ordre dans lequel nous mettons les chiffres est capital. Ainsi, par exemple, 2 569 n'est pas du tout le même nombre que 9 562. Et pourquoi ? Quel opération, quel décodage mental effectuons-nous lorsque nous lisons une suite de chiffres représentant un nombre ? Le problème, c'est que nous sommes tellement habitués à faire ce décodage de façon instinctive que généralement nous n'en connaissons plus les règles. Mais ce n'est pas très compliqué de les reconstituer… Et c'est là que nous mettons le doigt en plein dans la deuxième caractéristique de notre système de notation numérique : son caractère **décimal**.

Lorsque j'écris 9562, de quel nombre est-ce que je parle ? Décomposons la lecture chiffre par chiffre, de gauche à droite :

9562, c'est 9000 + 500 + 60 + 2.

Allons plus loin, même si cela paraît un peu bébête :

9000
C'est 9 x 1000
Parce que le 9 est le quatrième chiffre en partant de la droite
500
C'est 5 x 100
Parce que le 5 est le troisième chiffre en partant de la droite
60
C'est 6 x 10
Parce que le 6 est le deuxième chiffre en partant de la droite
2
C'est 2 x 1
Parce que le 5 est le premier chiffre en partant de la droite

On peut encore écrire ce même nombre d'une manière légèrement différente. Au lieu de :

9 562 = 9 x 1 000 + 5 x 100 + 6 x 10 + 2,

On écrit que :

9 562 = (9 x 10 x 10 x 10) + (5 x 10 x 10) + (6 x 10) + (2)

Arrivés à ce stade de la compétition, je prie les allergiques de m'excuser, mais il nous faut employer un petit peu de jargon mathématique. Ce n'est pas grand-chose, et on touche au but. Alors, courage ! En fait, ce jargon se résume au fait que les matheux notent la ligne ci-dessus à l'aide du symbole de « puissance ». Cela donne :

$9\,562 = 9 \times 10^3 + 5 \times 10^2 + 6 \times 10^1 + 2 \times 10^0$

Et voilà, nous y sommes. Nous avons dégagé le mécanisme général de la représentation par numérotation de position en base décimale.

Alors, nous en savons assez pour conclure sur les conséquences du choix de la base décimale. Il y en a deux, qui n'en forment en fin de compte qu'une seule :

- Parce que nous sommes en base décimale, nous utilisons un alphabet numérique de dix symboles. Nous nous servons de dix chiffres, pas un de plus, pas un de moins.

- Toujours parce nous sommes en base décimale, la position d'un de ces dix chiffres dans un nombre désigne la puissance de dix par laquelle ce chiffre doit être multiplié pour reconstituer le nombre. Si je trouve un 7 en cinquième position à partir de la droite, ce 7 ne représente pas 7 mais 7 fois 10^4, soit 70 000.

Un dernier mot concernant le choix de la base dix. Pourquoi celle-là et pas une autre ? Après tout, la base dix n'était pas le seul choix possible. Les babyloniens, qui furent de brillants mathématiciens, avaient en leur temps adopté la base 60 (dite **sexagésimale**). Cette base 60 impliquait certes d'utiliser un assez lourd alphabet numérique de 60 chiffres. Mais c'était somme toute un inconvénient mineur, et en retour, elle possédait certains avantages non négligeables. 60 étant un nombre divisible par beaucoup d'autres (c'est pour cette raison qu'il avait été choisi), on pouvait, rien qu'en regardant le dernier chiffre, savoir si un nombre était divisible par 2, 3, 4, 5, 6, 10, 12, 15, 20 et 30. Alors qu'en base 10, nous ne pouvons immédiatement répondre à la même question que pour les diviseurs 2 et 5. La base sexagésimale a certes disparu en tant que système de notation des nombres. Mais Babylone nous a laissé en héritage sa base sexagésimale dans la division du cercle en soixante parties (pour compter le temps en minutes et secondes), et celle en 6 x 60 parties (pour les degrés de la géométrie et de l'astronomie).

Alors, pourquoi avons-nous adopté la base décimale, moins pratique à bien des égards ? Nul doute que cela tienne au dispositif matériel grâce auquel tout être humain normalement constitué stocke spontanément une information numérique : ses doigts !

3. La numérotation de position en base binaire

Les ordinateurs, eux, comme on l'a vu, on un dispositif physique fait pour stocker (de multiples façons) des informations binaires. Alors, lorsqu'on représente une information stockée par un ordinateur, le plus simple est d'utiliser un système de représentation à deux chiffres : les fameux 0 et 1. Mais une fois de plus, je me permets d'insister, le choix du 0 et du 1 est une pure convention, et on aurait pu choisir n'importe quelle autre paire de symboles à leur place.

Dans un ordinateur, le dispositif qui permet de stocker de l'information est donc rudimentaire, bien plus rudimentaire que les mains humaines. Avec des mains humaines, on peut coder dix choses différentes (en fait bien plus, si l'on fait des acrobaties avec ses doigts, mais écartons ce cas). Avec un emplacement d'information d'ordinateur, on est limité à deux choses différentes seulement. Avec une telle information binaire, on ne va pas loin. Voilà pourquoi, dès leur invention, les ordinateurs ont été conçus pour manier ces informations par paquets de 0 et de 1. Et la taille de ces paquets a été fixée à 8 informations binaires.

Une information binaire (symbolisée couramment par 0 ou 1) s'appelle un bit. Un groupe de huit bits s'appelle un octet (en anglais, byte)

Combien d'états différents un octet possède-t-il ? Le calcul est assez facile (mais il faut néanmoins savoir le refaire). Chaque bit de l'octet peut occuper deux états. Il y a donc dans un octet :

$2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 = 2^8 = 256$ possibilités

Cela signifie qu'un octet peut servir à coder 256 nombres différents : ce peut être la série des nombres entiers de 1 à 256, ou de 0 à 255, ou de –127 à +128. C'est une pure affaire de convention, de choix de codage. Mais ce qui n'est pas affaire de choix, c'est le nombre de possibilités : elles sont 256, pas une de plus, pas une de moins, à cause de ce qu'est, par définition, un octet.

Si l'on veut coder des nombres plus grands que 256, ou des nombres négatifs, ou des nombres décimaux, on va donc être contraint de mobiliser plus d'un octet. Ce n'est pas un problème, et c'est très souvent que les ordinateurs procèdent ainsi.

En effet, avec deux octets, on a 256 x 256 = 65 536 possibilités.

En utilisant trois octets, on passe à 256 x 256 x 256 = 16 777 216 possibilités.

Et ainsi de suite, je ne m'attarderai pas davantage sur les différentes manières de coder les nombres avec des octets. On abordera de nouveau brièvement le sujet un peu plus loin.

Cela implique également qu'un octet peut servir à coder autre chose qu'un nombre : l'octet est très souvent employé pour coder du texte. Il y a 26 lettres dans l'alphabet. Même en comptant différemment les minuscules et les majuscules, et même en y ajoutant les chiffres et les signes de ponctuation, on arrive à un total inférieur à 256. Cela veut dire que pour coder convenablement un texte, le choix d'un caractère par octet est un choix pertinent.

Se pose alors le problème de savoir quel caractère doit être représenté par quel état de l'octet. Si ce choix était librement laissé à chaque informaticien, ou à chaque fabricant d'ordinateur, la communication entre deux ordinateurs serait un véritable casse-tête. L'octet 10001001 serait par exemple traduit par une machine comme un T majuscule, et par une autre comme une parenthèse fermante ! Aussi, il existe un standard international de codage des caractères et des signes de ponctuation. Ce standard stipule quel état de l'octet correspond à quel signe du clavier. Il s'appelle l'ASCII (pour *American Standard Code for Information Interchange*). Et fort heureusement, l'ASCII est un standard universellement reconnu et appliqué par les fabricants d'ordinateurs et de logiciels. Bien sûr, se pose le problème des signes propres à telle ou telle langue (comme les lettres accentuées en français, par exemple). L'ASCII a paré le problème en réservant certains codes d'octets pour ces caractères spéciaux à chaque langue. En ce qui concerne les langues utilisant un alphabet non latin, un standard particulier de codage a été mis au point. Quant aux langues non alphabétiques (comme le chinois), elles payent un lourd tribut à l'informatique pour n'avoir pas su évoluer vers le système alphabétique…

Revenons-en au codage des nombres sur un octet. Nous avons vu qu'un octet pouvait coder 256 nombres différents, par exemple (c'est le choix le plus spontané) la série des entiers de 0 à 255. Comment faire pour, à partir d'un octet, reconstituer le nombre dans la base décimale qui nous est plus familière ? Ce n'est pas sorcier ; il suffit d'appliquer, si on les a bien compris, les principes de la numérotation de position, en gardant à l'esprit que là, la base n'est pas décimale, mais binaire. Prenons un octet au hasard :

1 1 0 1 0 0 1 1

D'après les principes vus plus haut, ce nombre représente en base dix, en partant de la gauche :

$1 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 =$
$1 \times 128 + 1 \times 64 + 1 \times 16 + 1 \times 2 + 1 \times 1 =$
$128 + 64 + 16 + 2 + 1 =$
211

Et voilà ! Ce n'est pas plus compliqué que cela !

Inversement, comment traduire un nombre décimal en codage binaire ? Il suffit de rechercher dans notre nombre les puissances successives de deux. Prenons, par exemple, 186.

Dans 186, on trouve 1 x 128, soit 1 x 2^7. Je retranche 128 de 186 et j'obtiens 58.
Dans 58, on trouve 0 x 64, soit 0 x 2^6. Je ne retranche donc rien.
Dans 58, on trouve 1 x 32, soit 1 x 2^5. Je retranche 32 de 58 et j'obtiens 26.

Dans 26, on trouve 1 x 16, soit 1 x 2^4. Je retranche 16 de 26 et j'obtiens 10.
Dans 10, on trouve 1 x 8, soit 1 x 2^3. Je retranche 8 de 10 et j'obtiens 2.
Dans 2, on trouve 0 x 4, soit 0 x 2^2. Je ne retranche donc rien.
Dans 2, on trouve 1 x 2, soit 1 x 2^1. Je retranche 2 de 2 et j'obtiens 0.
Dans 0, on trouve 0 x 1, soit 0 x 2^0. Je ne retranche donc rien.

Il ne me reste plus qu'à reporter ces différents résultats (dans l'ordre !) pour reconstituer l'octet. J'écris alors qu'en binaire, 186 est représenté par :

1 0 1 1 1 0 1 0

C'est bon ? Alors on passe à la suite.

4. Le codage hexadécimal

Pour en finir avec ce préambule (sinon, cela deviendrait de la gourmandise) , on va évoquer un dernier type de codage, qui constitue une alternative pratique au codage binaire. Il s'agit du **codage hexadécimal**, autrement dit **en base seize**.

Pourquoi ce choix bizarre ? Tout d'abord, parce que le codage binaire, ce n'est tout de même pas très économique, ni très lisible. Pas très économique : pour représenter un nombre entre 1 et 256, il faut utiliser systématiquement huit chiffres. Pas très lisible : parce que d'interminables suites de 1 et de 0, on a déjà vu plus folichon.

Alors, une alternative toute naturelle, c'était de représenter l'octet non comme huit bits (ce que nous avons fait jusque là), mais comme deux paquets de 4 bits (les quatre de gauche, et les quatre de droite). Voyons voir cela de plus près.

Avec 4 bits, nous pouvons coder 2 x 2 x 2 x 2 = 16 nombres différents. En base seize, 16 nombres différents se représentent avec un seul chiffre (de même qu'en base 10, dix nombres se représentent avec un seul chiffre).

Quels symboles choisir pour les chiffres ? Pour les dix premiers, on n'a pas été chercher bien loin : on a recyclé les dix chiffres de la base décimale. Les dix premiers nombres de la base seize s'écrivent donc tout bêtement 0, 1, 2, 3, 4, 5, 6, 7, 8, et 9. Là, il nous manque encore 6 chiffres, pour représenter les nombres que nous écrivons en décimal 10, 11, 12, 13, 14, 15 et 16. Plutôt qu'inventer de nouveaux symboles (ce qu'on aurait très bien pu faire), on a recyclé les premières lettres de l'alphabet. Ainsi, par convention, A vaut 10, B vaut 11, etc. jusqu'à F qui vaut 15.

Or, on s'aperçoit que cette base hexadécimale permet une représentation très simple des octets du binaire. Prenons un octet au hasard :

1 0 0 1 1 1 1 0

Pour convertir ce nombre en hexadécimal, il y a deux méthodes : l'une consiste à faire un grand détour, en repassant par la base décimale. C'est un peu plus long, mais on y arrive. L'autre méthode consiste à faire le voyage direct du binaire vers l'hexadécimal. Avec l'habitude, c'est nettement plus rapide !

<u>Première méthode :</u>

On retombe sur un raisonnement déjà abordé. Cet octet représente en base dix :

1 x 2^7 + 0 x 2^6 + 0 x 2^5 + 1 x 2^4 + 1 x 2^3 + 1 x 2^2 + 1 x 2^1 + 0 x 2^0 =
1 x 128 + 1 x 16 + 1 x 8 + 1 x 4 + 1 x 2 + 0 x 1 =
128 + 16 + 8 + 4 + 2 =
158

De là, il faut repartir vers la base hexadécimale.

Dans 158, on trouve 9 x 16, c'est-à-dire 9 x 16^1. Je retranche 144 de 158 et j'obtiens 14.
Dans 14, on trouve 14 x 1, c'est-à-dire 14 x 16^0. On y est.
Le nombre s'écrit donc en hexadécimal : 9 E

Deuxième méthode :

Divisons 1 0 0 1 1 1 1 0 en 1 0 0 1 (partie gauche) et 1 1 1 0 (partie droite).

1 0 0 1, c'est 8 + 1, donc 9
1 1 1 0, c'est 8 + 4 + 2 donc 14

Le nombre s'écrit donc en hexadécimal : 9 E. C'est la même conclusion qu'avec la première méthode. Encore heureux !

Le codage hexadécimal est très souvent utilisé quand on a besoin de représenter les octets individuellement, car dans ce codage, tout octet correspond à seulement deux signes.

Allez, assez bavardé, on passe aux choses sérieuses : les arcanes de l'algorithmique…