

Divers codages d'information en informatique.

Le bit, l'octet (le byte), le mot, l'hexadécimal.

Dans un ordinateur, l'information est stockée sous forme de tensions électriques. Soit la tension est inférieure à un seuil, soit elle est supérieure à un seuil. Dans une mémoire flash, elle est stockée sous la forme de présence ou d'absence d'électrons dans un condensateur. Sur un disque dur ou une bande magnétique, l'information est stockée dans des minuscules régions sous forme de magnétisation, soit dans un sens, soit dans l'autre sens. Autrefois, on stockait l'information sur des bandes perforées, soit il y avait un trou, soit il n'y en avait pas.

Dans tous les cas, l'information élémentaire n'a que deux possibilités, que l'on note généralement **0** et **1**. Cette information élémentaire est appelée un **bit**, abréviation de **BI**nary **uni**T.

On peut donc représenter l'information stockée dans un ordinateur, un disque dur etc. par une suite de bits, donc une suite de 0 et de 1.

Pour plus de commodité, on a regroupé ces bits en **octets** ou **bytes**. Un **octet** est une suite de 8 bits. En écrivant ces 8 bits sous forme de 8 chiffres valant soit 0 soit 1, cela correspond à écrire un nombre en notation binaire. On dit pour cela que toute l'information est stockée sous forme de nombres dans un ordinateur, ce qui n'est qu'une manière pratique de se représenter la mémoire d'un ordinateur.

Un **bit** n'a que deux valeurs possibles, qui sont zéro ou un.

Un **octet** a $2^8 = 256$ valeurs possibles, qui vont de 0000 0000 à 1111 1111 (0 à 255).

On regroupe souvent les octets par paires pour former des **mots de 16 bits**.

Un **mot de 16 bits** a $2^{16} = 65'536$ valeurs possibles.

On regroupe souvent les octets par quadruplets pour former des **mots de 32 bits**.

Un **mot de 32 bits** a $2^{32} = 4'294'967'296$ valeurs possibles, soit environ 4 Giga valeurs.

La limitation de mémoire de 4 Giga octets des systèmes d'exploitations de 32 bits vient de là.

On regroupe encore les octets par série de huit, pour former des **mots de 64 bits**.

Un **mot de 64 bits** a 2^{64} valeurs possibles, soit environ $16 \cdot 10^{18}$ valeurs.

Les systèmes d'exploitations 64 bits ne sont pas limités par la taille des mots de 64 bits.

Pour noter un octet ou un mot de 16 ou 32 bits, la notation binaire est désagréable. Pour cela on utilise la notation **hexadécimale**, qui regroupe des bits par série de 4, pour former des nombres allant de 0 à 15.

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

En notation hexadécimale, six "chiffres" sont ajoutés, que l'on note A, B, C, D, E et F.

Ainsi, un octet est noté à l'aide de deux nombres hexadécimaux.

0000 0000 = #00 Le # est une manière d'indiquer la numérotation hexadécimale.

1111 1111 = #FF

Exercice 1

Complétez les notations binaires et hexadécimales :

01011101 =

= 46

= CC

10000111 =

= 7A

= B0

Exercice 2

Complétez...

Un mot de 16 bits est représenté par ____ chiffres hexadécimaux.

Un mot de 32 bits est représenté par ____ chiffres hexadécimaux.

#FA18 représente un mot de ____ bits.

Le plus grand mot de 32 bits est noté en hexadécimal par _____

Codage des couleurs

Puisque l'œil humain est composé de 3 sortes de capteurs de couleurs, sensibles soit aux couleurs voisines du rouge, soit du vert, soit du bleu, on peut coder presque toutes les couleurs en indiquant la proportion de rouge, de vert et de bleue.

Une image sur un écran d'ordinateur ou de télévision est constitué de **pixels** (PIXture ELelements).

Chaque pixel a une couleur, déterminée par la quantité de rouge, de vert et de bleu. C'est le système **RVB** ou **RGB** (Red Green Blue en anglais).

Chacune des trois quantités est déterminée par un octet, donc une valeur comprise entre 0 et 255.

Donc un pixel est représenté par un nombre hexadécimal de 6 chiffres, les deux premiers indiquant la quantité de rouge, les deux suivants indiquent la quantité de vert, les deux derniers celle de bleue.

Exercice 3

Faites la correspondance entre le nombre hexadécimal et la couleur.

#FF0000 ↔ _____

#00FF00 ↔ _____

#0000FF ↔ _____

#888888 ↔ _____

_____ ↔ noir

_____ ↔ blanc

_____ ↔ gris foncé

_____ ↔ gris clair

_____ ↔ jaune

_____ ↔ magenta

_____ ↔ cyan

_____ ↔ brun

Codage de caractères, ASCII, iso-8859-1, iso-885-15, Windows-1252, utf-8

Chaque caractère est aussi codé par une séquence de bits. A l'origine, c'était facile, car les américains n'avaient besoin de coder que les 26 caractères alphabétiques minuscules, les 26 majuscules, les 10 chiffres et quelques caractères de ponctuation tels que : , . ; : ' ? ! + - = " () * % & / [] { } etc. 96 (=256 – 32) symboles suffisaient amplement.

Donc pour ces caractères, un octet est largement suffisant.

Les 32 premiers nombres, codent des caractères de contrôle, tel que le tabulateur, le retour à la ligne, le saut de ligne, le saut de page, la fin de fichier.

Les nombres allant de 32 à 127 codent les caractères cités précédemment.

Le 8^{ème} bit servait de bit de contrôle, pour tester si une erreur s'était introduite, par exemple lors d'une transmission. Ce bit de contrôle n'est plus utilisé de nos jours.

Ce codage sur 7 bits est le **code ASCII**. Il est **standard** et utilisé par tous les systèmes d'exploitation.

Avec l'apparition du DOS et l'utilisation d'ordinateurs en Europe, divers codages sont apparus pour coder les caractères accentués. Cela c'est fait de manière complètement anarchique, et de nombreux systèmes de codages ont étendus le code ASCII. Microsoft et Apple ont chacun développé leur système de codage.

En 1986, la norme **iso-8859-1** a défini le codage des caractères accentués, sur un octet.

Il définit le codage de 191 caractères.

Il manquait certains caractères, tel que le : œ.

La norme **iso-8859-15** a comblé en 1998 certains manques, en étendant le codage iso-8859-1.

Ce codage se nomme également **Windows-1252**.

L'avantage de ces systèmes de codage est qu'un caractère est codé sur un octet.

Pour les américains, du nord et du sud et les européens, cela suffit. Mais pas pour les arabes, les japonais, le chinois et une majorité du monde ! Pour cela d'autres systèmes de codages ont été défini. Au début des années 1990, le codage **UTF-8** est apparu. Il est utilisé actuellement dans plus de trois quarts des pages Web. Il a l'*avantage* de pouvoir coder plus d'un million de caractères, dont les caractères arabes, japonais, chinois et bien d'autres. Il a le *désavantage* d'utiliser de 1 à 4 octets pour coder un caractère. Il est *compatible avec le codage ASCII*. Donc les caractères de l'ASCII sont codés sur un octet en utf-8, de la même manière qu'en ASCII.

Par contre, les caractères accentués sont codés sur deux octets.

C'est ce système de codage qui est utilisé sur UNIX, Linux et MAC OSX.

D'autres systèmes de codage existent, tels que l'utf-16 et l'utf-32. Unicode est une définition encore plus générale.

Exercice 4

Complétez le tableau suivant : (# signifie codage en hexadécimal) c.f. <http://www.utf8-chartable.de/>

Caractère	code ASCII	code iso-8859-15	code utf-8
A			
	#61 = 97		
			#20 = 32
é			
ñ			
			#c3a6
			#e0a9a9

Codage des nombres entiers

Les nombres peuvent se classer dans trois catégories. Les nombres entiers positifs, les nombres entiers relatifs et les nombres à virgules. L'informatique étant limitée, aucune distinction n'est faite entre les nombres irrationnels et les nombres rationnels. Parmi tous les nombres, très peu peuvent être codés de manière exacte.

Exercice 5

Si on se limite aux **entiers positifs** :

un **octet** peut coder les nombres de 0 à 255.

un **mot de 16 bits** peut coder les nombres de 0 à _____

un **mot de 32 bits** peut coder les nombres de 0 à _____

un **mot de 64 bits** peut coder les nombres de 0 à _____

Le codage des **entiers relatifs**, suit une logique mathématique.

Remarquons que : $255 + 1 = 256 = 1\ 0000\ 0000$ en binaire.

Si on élimine le 9^{ème} bit, cela donne $255 + 1 = 0$, donc 255 représente le nombre entier négatif -1 .

Toujours en éliminant le 9^{ème} bit, $254 + 2 = 0$, donc 254 représente le nombre entier négatif -2 .

Sur un octet, on code les nombres entiers relatifs comme suit :

0 1 2 ... 127 sont les entiers positifs

255 254 ... 129 sont les entiers négatifs, de $-1, -2, \dots$ à -127 .

128 est une autre manière de coder 0, il correspond à " -0 ".

Sur un **mot de 16 bits**, rappelons que $2^{16} = 65'536$.

$65'535 + 1 = 0$, si on élimine le 17^{ème} bit. Donc $65'535 = -1$.

$32'769 + 32'767 = 0$, donc $32'769 = -32'767$.

Sur un **mot de 32 bits**, rappelons que $2^{32} = 4'294'967'296$. La moitié vaut : $2'147'483'648$.

Le codage des nombres négatifs dépend du nombre d'octets choisi pour coder les nombres entiers.

Exercice 6

Sur un *mot de 32 bits*, à quel entier positif correspond le nombre -1 ?

En se limitant aux *octets*, que donne l'opération suivante, en nombre positif et en nombre négatif.

En nombre négatif, on parlera d'**overflow**, qui signifie que l'on dépasse les capacités de ce codage.

$69 + 61$

Sur un *mot de 32 bits*, dans le codage des entiers relatifs, quel est le plus grand nombre positif ?

Suggérez la taille d'un mot à considérer pour coder des entiers relatifs plus grand que mille milliards.

Dans ce système, quel sera le plus grand entier ? À quel nombre entier correspondra -1 ?

Dans un système de *mots de 16 bits*, donnez un exemple d'addition provoquant un overflow.

Dans un système de *mots de 8 bits*, donnez un sens à l'égalité suivante.

$2 + 250 = -4$

Codage des nombres à virgule

Le système de codage des nombres à virgules dans un ordinateur est assez complexe. On ne considérera que deux type de codage, en simple précision, sur 32 bits et en double précision, sur 64 bits. Ce dernier type de codage est le plus couramment utilisé de nos jours. Ces codages imitent l'écriture scientifique des nombres.

Codage simple précision = à virgule flottante en 32 bites : nombre = $s \cdot 2^{e-127} \cdot 1, \text{mantisse}$
 $s = 1$ ou -1 suivant la valeur du premier bit.
 $e =$ le nombre entier positif représenté par les 8 bits allant du 2^{ème} au 9^{ème}.
 La mantisse donne les 23 chiffres binaires après la virgule.

Codage double précision = à virgule flottante en 64 bites : nombre = $s \cdot 2^{e-1023} \cdot 1, \text{mantisse}$
 $s = 1$ ou -1 suivant la valeur du premier bit.
 $e =$ le nombre entier positif représenté par les 11 bits allant du 2^{ème} au 12^{ème}.
 La mantisse donne les 52 chiffres binaires après la virgule.

Il y a une exception si $e = 0$, alors le nombre = $s \cdot 2^{-127} \cdot 0, \text{mantisse}$ en simple précision.
 Il y a aussi une exception si $e = \text{maximum}$, le nombre représente $+$ ou $-$ l'infini.

Exemples en simple précision :

1 = 0 01111111 000000000000000000000000
 -2 = 1 10000000 000000000000000000000000
 13 = 0 10000010 101000000000000000000000

Le signe est positif.

le facteur multiplicateur vaut $2^{130-127} = 2^3 = 8$
 1,mantisse vaut $1 + 1 \cdot 1/2 + 0 \cdot 1/4 + 1 \cdot 1/8 + 0 = 13/8$
 Donc le nombre vaut bien $8 \cdot 13/8 = 13$.

0,625 = 0 01111110 010000000000000000000000
 le facteur multiplicateur vaut $2^{126-127} = 2^{-1} = 1/2 = 0,5$
 1,mantisse vaut $1 + 0 \cdot 1/2 + 1 \cdot 1/4 + 0 \cdot 1/8 + 0 = 1 + 1/4 = 1,25$
 Donc le nombre vaut bien $0,5 \cdot 1,25 = 0,625$

0,1 \approx 0 01111011 10011001100110011001100
 le facteur multiplicateur vaut $2^{123-127} = 2^{-4} = 1/16 = 0,0625$
 1,mantisse vaut $1 + 2^{-1} + 2^{-4} + 2^{-5} + 2^{-8} + 2^{-9} + 2^{-12} + 2^{-13} + 2^{-16} + 2^{-20} + 2^{-21}$
 Le produit des deux nombres ci-dessus vaut presque 0,1.
 Il n'est pas possible d'écrire 0,1 de manière exacte dans ce système de codage des nombres !

Exemples en double précision :

1 = 0 011111111111 000
 -13 = 1 10000000010 101000
 -0,1 \approx 1 01111111011 10011001100110011001100110011001100110011001100110011001
 0,75 = 0 01111111110 1000

le facteur multiplicateur vaut $2^{126-127} = 2^{-1} = 1/2 = 0,5$
 1,mantisse vaut $1 + 1/2 = 1,5$
 Donc le nombre vaut bien $0,5 \cdot 1,5 = 0,75$

Pour qu'un nombre puisse s'écrire de manière exacte, il faut qu'il puisse s'écrire sous la forme d'un nombre entier divisé par 2 à la puissance un nombre entier.
 Donc la plupart des nombres à virgules sont approximés en informatique.

Utilisez des feuilles supplémentaires pour résoudre les exercices qui suivent.Exercice 7 :

En simple précision, quels sont les nombres représentés par :

A = 0 10000011 000100000000000000000000
 B = 0 01111101 010000000000000000000000
 C = 1 10000011 000001000000000000000000
 D = 0 01111111 10011001100110011001100
 E = 1 01111101 00110011001100110011001
 F = 0 10000000 10010010000111111011010
 G = 0 10000000 10010001111010111000010
 H = 0 10000000 01011011111100001010100
 I = 0 01111111 01101010000010011110011
 J = 0 01111111 10011110001101110111100

Exercice 8 :

En double précision, quels sont les nombres représentés par :

A = 0 100000000000 1001001000011111101101010100010001000010110100011000
 B = 0 100000000000 0101101111110000101010001011000101000101011101101001
 C = 0 011111111111 0110101000001001111001100110011111110011101111001101
 D = 0 011111111111 1001111000110111011110011011100101111111010010101000

Exercice 9 :

En vous basant sur l'exercice 7, pouvez-vous écrire la représentation binaire en double précision, du nombre 1,6 ? Et du nombre -0,3 ?

Exercice 10 :

Représentez en binaire, simple précision, les nombres suivants :

(Si vous êtes particulièrement curieux, vous pouvez les représenter aussi en double précision)

55
 -28
 Votre âge
 0,6
 -pi/4
 0,7
 Le plus grand nombre possible
 Le plus petit nombre positif

Exercice 11 :

Écrivez un programme en javascript, ou dans un autre langage, qui fait toutes les conversions de nombres...

C'est un projet en soi !

Initiation au C++

Ce qui suit est une initiation au C++, sous le système d'exploitation GNU/Linux, XUbuntu 14.4, donc avec l'environnement Xfce.

Nous utiliserons l'IDE (Integrated Development Environment) Code::Blocks.

Nous nous baserons sur le cours de l'excellent site Web "<http://fr.openclassrooms.com/>"

Sous : Cours > Informatique C++ on arrive à <http://fr.openclassrooms.com/informatique/c-1/cours>

Prenez la recommandation de l'équipe, "Programmez avec le langage C++"

<http://fr.openclassrooms.com/informatique/cours/programmez-avec-le-langage-c>

Je ne résumerai qu'une petite partie de ce cours, que je vous conseil de suivre.

Sous "[2. Les logiciels nécessaires pour programmer](#)"

Vérifiez si le logiciel suivant existe sous "Menu démarrer" > "Développement" > "**Code::Blocks IDE**"

Si ce n'est pas le cas, vous devez l'installer, ainsi qu'un compilateur, comme suit :

1) ouvrez un Terminal, soit dans "Accessoires" > "Émulateur de Terminal"

soit en tapant "super T" où la touche "super" est celle de "Windows", entre "Ctrl" et "Alt".

2) dans ce Terminal, tapez : `sudo apt-get install build-essential codeblocks`

Le mot de passe standard vous est demandé, puis l'installation se fait.

"build-essential" est le compilateur C++, codeblocks est l'IDE.

Sous "[3. Votre premier programme](#)"

Lancez "Code::Blocks IDE" sous "Développement"

File > New > Project...

Choisir : "Console application" Next> "C++" Next>

Écrivez un titre, je suggère : "ex001_hello_world" tout en minuscule

"Folder to create project in:" je suggère : "/home/bg/cpp" (bg étant votre identifiant)

Next>

Cochez la case "Create Release configuration", l'autre case, je la décoche.

Finish.

Cliquez sur la flèche, à gauche de "Sources".

Double cliquez sur main.cpp

Vous voyez votre premier programme ! Pour le compiler et l'exécuter, il y a 3 possibilités :

1) "Build" > "Build and run"

2) Pressez sur l'icône suit la flèche verte, celle qui a une flèche verte et un roue dentée jaune.

3) Pressez sur la touche F9.

Un fenêtre s'ouvre avec le texte : "Hello world!"

Pressez la touche Enter pour fermer cette fenêtre.

Essayez d'écrire un autre texte à la place de "Hello world", puis de relancer.

Vous pouvez aussi dupliquer cette ligne et changer le texte. Amusez-vous un peu...

Même un texte avec des accents fonctionne, ce qui n'était pas évident autrefois.

La ligne : `#include <iostream>` inclue la bibliothèque servant au "Input" et au "Output".

"int main()" est la fonction principale, celle qui s'exécute au début du lancement de votre programme.

Elle retourne un "int", c'est-à-dire un nombre entier.

"cout << "Hello world!" << endl;" dirige vers la sortie standard le texte, avec un retour à la ligne.

"return 0;" est la valeur retournée par cette fonction. "0" indique qu'il n'y a pas eu d'erreurs.

Sauvez le projet, fermez-le : "File" > "Close project"
 Créez un nouveau projet : "File" > "New" > "Project" > "Console application"
 Appelez-le "ex002_sommes"

Il est aussi possible de prendre le premier projet, de l'enregistrer comme un "Template"
 "File" > "Save project as template..."
 Puis "File" > "New" > "Project" > "User templates" puis prendre celui qu'on vient de sauver en
 "Template"

Ainsi je vous conseil d'avoir plusieurs projets pour sauvegarder divers exemples.

Sous "[4. Utiliser la mémoire](#)"

Le C++ est un langage typé. Chaque variable a un type bien défini. Regardez les type défini dans cette section.

Lisez la suite si vous voulez, je la saute.

Sous "[5. Une vraie calculatrice](#)" regardez ce que vous voulez.

Pour la suite, je vous laisse voir.

Voici un petit programme pour apprendre, avec des commentaires.

C'est celui que je mets dans "ex002_sommes"

```
#include <iostream> // pour les entrées - sorties (In & Out)
#include <cmath> // pour les fonctions mathématiques, sqrt = racine carrée.
using namespace std;

int main()
{
  int nCount = 0; // indice qui va parcourir les entiers de 1 à nMax
  int nMax = 0; // valeur maximum de l'indice de la somme
  double vSum = 0; // résultat de la somme

  // Demande à l'utilisateur la valeur de nMax
  cout << "nombre de termes à sommer : ";
  cin >> nMax; // attend la saisie d'un nombre entier positif.

  // Boucle effectuant la somme
  for (nCount = 1; nCount <= nMax; nCount++) {
    // tenez compte de l'indentation !
    vSum = vSum + 1.0 / (1.0*nCount * nCount);
    // sans le "1.0*nCount", le programme est très limité, car
    // on dépasse rapidement la capacité de codage des nombres entiers.
  }

  // Ce premier résultat ne permet pas de deviner ce que cette somme vaut.
  cout << "La somme des inverses des carrés de 1 à " << nMax
    << "vaut : " << vSum << endl;

  // Autre affichage, qui donne plus d'espoir de deviner la valeur de la somme
  // lorsque nMax tend vers l'infini.
  cout << "Racine carré de 6 fois la somme ci-dessus : " << sqrt(6*vSum) << endl;

  return 0;
}
```

On peut ajouter : cout << setprecision(15); // pour définit une précision.

À ajouter dans l'en-tête : #include <iomanip> // pour le formattage, tel que setprecision, setw, setfill('-'), setbase

Sous "6. Les structures de contrôle"

Apprenez à utiliser les structures de contrôles :

4 structures de contrôle importantes :

```
if (...) { ... } else if (...) { ... } else { ... }
```

```
for ( i=1; i<10; i++) { ... }
```

```
while (...) { ... }
```

```
do { ... } while (...);
```

Exercice 1

Écrivez un programme, qui calcule la plus grande valeur de nMax, telle que la somme des carrés de nombres de 1 à nMax soit le plus grand nombre inférieur à 1'000'000 !

Exercice 2

Écrivez le programme suivant :

(Il tient sur 45 lignes, avec espaces et des commentaires !)

L'ordinateur tire un nombre au hasard entre 1 et 100.

Utilisez :

```
#include <time.h>
```

```
srand (time(NULL)); // pour initialiser le générateur de nombres aléatoires
```

```
nNbrCache = rand() % 100; // pour tirer un nombre entier au hasard entre 0 et 99.
```

Le joueur écrit un nombre.

S'il est correct, il a gagné. On indique le nombres d'essais.

S'il est trop petit, ou trop grand, l'ordinateur l'indique et le joueur essaye à nouveau.

Cela initie à la méthode *dichotomique*.