COURS

Représentation des données — Binaire et hexadécimal

1re

Programme de première générale, enseignement de spécialité — BO spécial n°1 du 22 janvier 2019

Additionnez 0,1 et 0,2 dans n'importe quel langage : le résultat n'est pas 0,3. Ce n'est ni un bug ni une imprécision de la machine, c'est une conséquence directe de la base 2. Comprendre pourquoi évite l'une des erreurs les plus tenaces de la programmation débutante.

1 Compter en base 2

DÉFINITION

Le **binaire** est la numération en **base 2** : chaque chiffre, appelé **bit**, vaut 0 ou 1, et chaque position vaut une **puissance de 2** croissante de droite à gauche. Un **octet** est un groupe de **8 bits**. Convertir du **binaire** vers le décimal consiste à additionner les puissances de 2 correspondant aux bits à 1.

$$1011_2 = 8 + 0 + 2 + 1 = 11$$

EXEMPLE

Combien de valeurs différentes un octet peut-il représenter ?

1. Chaque **bit** prend **2** valeurs, et les 8 bits sont indépendants.
2. Le nombre de combinaisons est donc le produit :
   $2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 = 256$
3. Soit **256** valeurs — de 0 à 255 pour des entiers non signés.

256 valeurs

DÉFINITION

L'**hexadécimal** est la **base 16**, qui emploie les chiffres 0 à 9 puis les lettres A à F pour 10 à 15. Son intérêt est pratique : **un** chiffre **hexadécimal** correspond exactement à **quatre bits**, et un **octet** s'écrit donc avec **deux** chiffres. C'est pourquoi on l'emploie pour les couleurs, les adresses mémoire et les octets bruts — il abrège le **binaire** sans en trahir la structure.

$$FF_{16} = 15 \times 16 + 15 = 255$$

2 Les entiers négatifs

RÈGLE

La représentation en **complément à 2** permet de coder les entiers **négatifs** sans signe séparé. Le bit de poids fort vaut alors une puissance **négative** : sur 8 bits, il vaut −128, et les sept autres gardent leurs valeurs positives. L'intérêt est décisif : l'addition fonctionne **sans cas particulier**, le même circuit additionne des nombres positifs et négatifs. Sur 8 bits, on représente ainsi les entiers de **−128 à 127**.

$$sur n bits, de -2^{n-1} à 2^{n-1} - 1$$

Cette plage explique les **débordements** : ajouter 1 à la plus grande valeur représentable ne donne pas la valeur suivante mais la plus **petite**, le compteur repartant à l'autre extrémité. Un programme qui ne vérifie pas ses bornes produit alors un résultat parfaitement déterministe et parfaitement faux — c'est l'origine de nombreux incidents célèbres.

3 Coder les caractères

PROPRIÉTÉ

La table **ASCII** associe un code à chaque caractère de l'alphabet latin non accentué, sur **7 bits**, soit 128 caractères. Elle ne suffit évidemment pas aux autres écritures. **UTF-8** résout le problème par un codage de **longueur variable** : les caractères **ASCII** y gardent leur code sur **un octet** — ce qui assure la compatibilité —, et les autres en occupent deux, trois ou quatre. Un texte accentué occupe donc **plus d'octets que de caractères**.

D'où une conséquence pratique constante : la **longueur en caractères** d'une chaîne et sa **taille en octets** ne coïncident pas. Compter l'un pour l'autre produit des chaînes tronquées au milieu d'un caractère, et c'est l'origine des caractères illisibles qu'on voit encore apparaître dans certains logiciels.

4 Les nombres à virgule

RÈGLE

Un **flottant** est codé selon la norme **IEEE 754** : un signe, un **exposant** et une **mantisse**, sur un nombre **fini** de bits. Or un nombre décimal aussi simple que **0,1** n'a **pas d'écriture finie en base 2**, exactement comme un tiers n'en a pas en base 10. Le **flottant** n'en stocke donc qu'une **approximation**, et les erreurs s'accumulent au fil des calculs.

LE PIÈGE QUI COÛTE LE PLUS DE POINTS

Tester l'égalité de deux nombres flottants avec un opérateur d'égalité stricte, et s'étonner que la somme de 0,1 et 0,2 ne vaille pas 0,3. — Aucun des trois nombres n'est stocké exactement : l'égalité porte donc sur des **approximations**, et rien ne garantit que deux chemins de calcul différents aboutissent au même arrondi. Le test échoue alors sans qu'aucune erreur de programmation n'ait été commise. La solution est de comparer à une **tolérance** près : vérifier que l'écart absolu est inférieur à un seuil choisi.

Le réflexe : Sur des **flottant**s, comparer un **écart** à un seuil, jamais deux valeurs.

CONVERTIR UN NOMBRE ENTRE BASES

1. Identifier la **base de départ** et la **base d'arrivée** avant toute chose.
2. Vers le décimal : additionner chaque chiffre multiplié par la **puissance** de la base correspondant à sa position.
   $8 + 2 + 1 = 11$
   1011 en base 2 vaut 11 en décimal.
3. Depuis le décimal : diviser successivement par la base et lire les **restes de bas en haut**.
4. Entre **binaire** et **hexadécimal** : regrouper les bits **par quatre**, aucune division n'est nécessaire.
5. Vérifier l'**ordre de grandeur** : un octet ne dépasse pas 255 en non signé.

À VÉRIFIER

- Ai-je vérifié la **base** de départ avant de convertir ?
- Mes puissances de 2 sont-elles comptées **de droite à gauche** à partir de 0 ?
- Pour passer du **binaire** à l'**hexadécimal**, ai-je regroupé **par quatre bits** ?
- En **complément à 2**, ai-je donné au bit de poids fort une valeur **négative** ?
- Ai-je distingué **longueur en caractères** et **taille en octets** en **UTF-8** ?
- Ai-je comparé mes **flottant**s à une **tolérance** près ?

À RETENIR

- **Binaire** : base 2, chaque position vaut une **puissance de 2**.
- Un **octet** = **8 bits** = **256** valeurs, de 0 à 255 en non signé.
- **Hexadécimal** : base 16 ; **un chiffre = quatre bits**, un octet = deux chiffres.
- Entre binaire et hexadécimal, **regrouper par quatre** — pas de division.
- **Complément à 2** : le bit de poids fort vaut -2^{n-1}, plage −128 à 127 sur 8 bits.
- Avantage : l'addition fonctionne **sans cas particulier**.
- Un **débordement** est déterministe : le défaut est dans le programme.
- **ASCII** : 7 bits, 128 caractères. **UTF-8** : longueur **variable**, compatible ASCII.
- **Longueur en caractères ≠ taille en octets**.
- **IEEE 754** : un **flottant** stocke une **approximation**.
- Comparer des flottants par un **écart** à un seuil, jamais par égalité stricte.