Introduction à l'informatique

pour les mathématiques, la physique et les sciences computationnelles

Yann Thorimbert

Chapitre 2 - Partie 2
Codage des réels

Yann Thorimbert

UNIVERSITÉ DE GENÈVE
CENTRE UNIVERSITAIRE D'INFORMATIQUE

Chapitres du cours

1. Origines des ordinateurs et des réseaux informatiques
2. **Codage des nombres** ←
3. Codage des médias
4. Circuits logiques
5. Architecture des ordinateurs
6. Conception et exécution de programmes
7. Algorithmique, programmation et structures de données

Représentation des réels

- Les entiers naturels sont composés d'une information fondamentale :
  - Leur norme.
- Les entiers relatifs sont composés de deux informations fondamentales :
  - Leur signe.
  - Leur norme.
- Les nombres réels sont composés de trois informations fondamentales :
  - Leur signe.
  - Leur partie entière.
  - Leur partie fractionnaire.

Représentation des réels

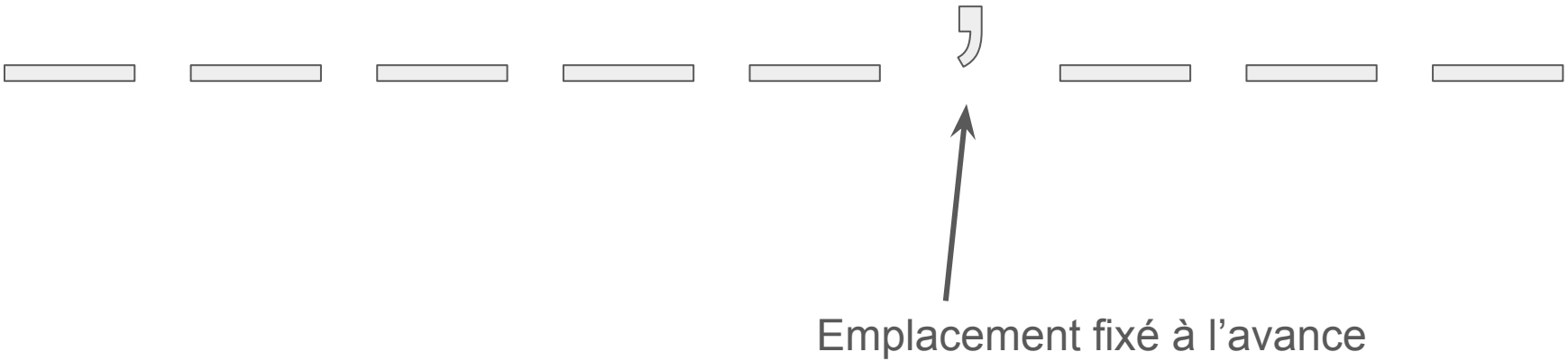
- Autrement dit : un réel inclut une information supplémentaire : la position de la virgule.

- Exemple pour le nombre -36.4, qui contient 5 informations pour 3 chiffres.

Signe	**a_2**	**a_1**	**a_0**	**Position virgule**
−1	3	6	4	2

Représentation de réels en virgule fixe

Exemple sur 8 bits, dont trois après la virgule :



Virgule fixe | **Principe**

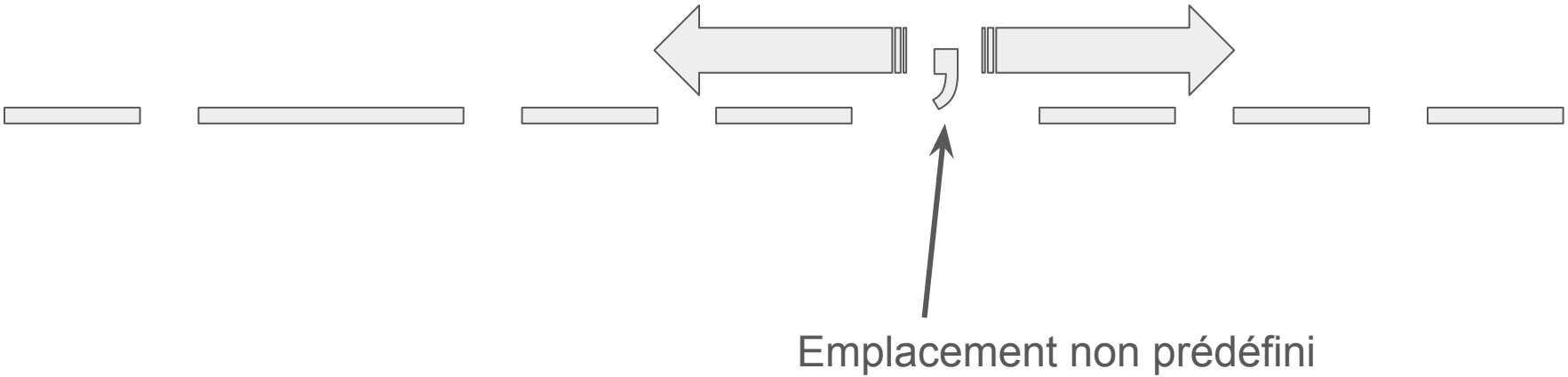
Virgule fixe | **Avantages et désavantages**

✔ Simplicité.

✔ Si l’on connaît la plage de valeurs à représenter, permet d’exploiter au mieux la mémoire disponible.

❌ Ne permet pas de représenter efficacement des valeurs avec des précisions ou des ordres de grandeurs très différents entre eux (comme par exemple des valeurs issues de deux appareils de mesure différents en physique).

Représentations de réels en virgule flottante



Virgule flottante | **Principe**

Virgule flottante | **Format de l'exposant**

- Finalement, il faut que *E* puisse être négatif.
- On pourrait utiliser la méthode du complément à 2 pour coder *E*…
- … mais pour des raisons pratiques, c'est la méthode du biais qui est utilisée.
- On dit que cod(*E*) est l'**exposant biaisé**.
  cod(*E*) = $\text{cod}_{\mathbb{Z}B+(k')}(E) = \text{cod}_{\mathbb{N}(k')}(E + 2^{k'-1} - 1)$, avec *k'* le nombre de bits de *E*.

Virgule flottante | **Norme IEEE 754**

- $n = s \cdot m \cdot 2^E$
- Formats courants :
  - Sur 32 bits : "**simple précision**". Souvent associé à l'alias **`float`**.
    1 bit pour *s*, 8 bits pour *E*, 23 bits pour *m* (dans cet ordre précis).
  - Sur 64 bits : "**double précision**". Souvent associé à l'alias **`double`**.
    1 bit pour *s*, 11 bits pour *E*, 52 bits pour *m* (dans cet ordre précis).

Virgule flottante | **Norme IEEE 754 - chiffres significatifs**

- Sur 32 bits : `float` ("**simple précision**"), 23 bits pour *m*.

  ⇒ La mantisse possède 23 chiffres significatifs en binaire. En base 10, elle prend au maximum la valeur 2^{23}- 1, ce qui s'écrit avec $\log_{10}(2^{23} - 1) + 1$ chiffres (c'est-à-dire environ 7 chiffres significatifs en décimal).

- Sur 64 bits : `double` ("**double précision**"), 52 bits pour *m*.

  ⇒ 16 chiffres significatifs en décimal.

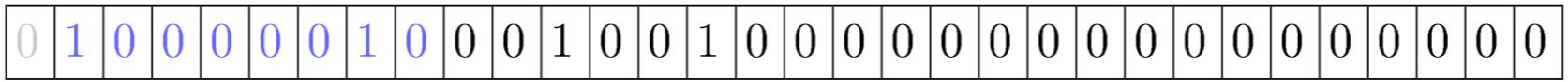
Virgule flottante | **Norme IEEE 754 - Exemples**

Exemple de bout de code C déclarant un nombre en simple précision

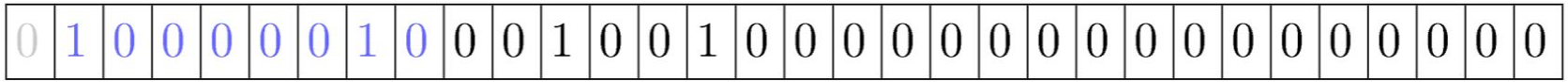
```
float monNombre = 9.125;
```

(En revanche, Python stocke aussi d'autres choses que le nombre lui-même. Par ailleurs, sur les architecture 64 bits, `float` est en général une double précision)

Séquence de bits stockée dans la mémoire de la machine :



Virgule flottante | **Norme IEEE 754 - Exemples**



Vérification ?

Rappel pour la simple précision :

$E = \mathrm{dec}(b_E) = \mathrm{dec}_{\mathbb{N}(k')}(b_E) - 2^{k'-1} + 1 = \mathrm{dec}_{\mathbb{N}(k')}(b_E) - 127$

Virgule flottante | **IEEE 754 - Représentation du zéro ?**

Essayons d’écrire le nombre le plus proche de zéro en précision simple :

Virgule flottante | **IEEE 754 - Nombres dénormalisés**

- Plage de valeurs qui rompt avec la convention $n = s \cdot m \cdot 2^E$.
- Concerne tous les nombres avec cod(E) = 0 et cod(m) ≠ 0.
- Passage à une technique similaire à la virgule fixe dans ce cas :
  - E = constante implicite (-126 pour simple précision).
  - m codée sans le bit implicite.
- Permet de représenter des quantités très petites.
- Pas utilisé dans ce cours.

Virgule flottante | **IEEE 754 - Nombres spéciaux**

Valeurs spécifiques de *E* et *m* réservées. Exemple en simple précision :

cod(*E*)	cod(*m*)	Signification
00000000	0 . . . 0	Zéro exact
00000000	≠ 0 . . . 0	Nombre dénormalisé
11111111	0 . . . 0	???
11111111	≠ 0 . . . 0	???

N.B : le bit de signe n'intervient pas ici.

Virgule flottante | **IEEE 754 - Nombres spéciaux**

Valeurs spécifiques de *E* et *m* réservées. Exemple en simple précision :

cod(*E*)	**cod(*m*)**	**Signification**
`00000000`	`0 . . . 0`	Zéro exact
`00000000`	`≠ 0 . . . 0`	Nombre dénormalisé
`11111111`	`0 . . . 0`	$s \cdot \infty$
`11111111`	`≠ 0 . . . 0`	NaN (**N**ot **a** **N**umber)

Valeur minimale normalisée : cod(*m*) = 0…0 et cod(*E*) = 00000001 donne environ $\pm 10^{-38}$ en simple précision.Valeur max donne environ $\pm 10^{38}$.

Virgule flottante | **IEEE 754 - Plage de valeurs**

Valeurs spécifiques de *E* et *m* réservées. Exemple en simple précision :

$-\infty$ à -10^{38}		-10^{-38} à 0	0	0 à 10^{-38}		10^{38} à ∞
overflow		underflow		underflow		overflow

Virgule flottante | **Précision finie et erreurs d'arrondi**

- Partie fractionnaire pas forcément représentable par une suite finie de bits.
- Partie fractionnaire pas forcément représentable par une suite finie de bits plus courte que **l'espace dont on dispose** !
- Deux problèmes de natures différentes :
  - Expression du nombre dans la base utilisée
  - Taille de la séquence de bits codant le nombre
- Par ailleurs : non seulement on représente un sous-ensemble de $\mathbb{R}$, mais encore au sein de ce sous-ensemble on fait des "sauts".

Rappel | **Développement fractionnaire**

- Un nombre peut avoir un développement fractionnaire fini dans une base mais infini dans une autre.
- Développement fini si $n = x / b^y$, où x et y sont des entiers positifs.
- Exemples :

  $0.5_{10} = 5 / 10^1 = 1 / 2^1$ a un développement fini en bases 10 et 2.

  $0.1_{10} = 1 / 10^1$ n'a pas de développement fini en base 2.

⇒ Une **précision arbitraire** doit être choisie pour approximer un réel quelconque sur un nombre fini de bits.

Rappel | **Développement fractionnaire**

- On a vu qu'en base b, multiplier n par b^k revient à décaler k fois la virgule.
- Si $n = x / y$ possède un développement fractionnaire fini de longueur k, alors $b^k \cdot x / y$ n'est pas une fraction.
- Autrement dit, cela signifie que les facteurs premiers de y doivent se retrouver dans $b^k \cdot x$ si le développement fractionnaire est fini.
- Si $n = x / y$ est irréductible, alors par définition x et y n'ont pas de facteurs premiers communs. Par conséquent, tous les facteurs premiers de y doivent également faire partie de ceux de b^k.
- Exemple en base 10 : ¾ = 3 / (2·2) et comme 10 = 2·5, pour retrouver tous les facteurs premiers du dénominateur dans le facteur b^k, le développement fractionnaire possède k = 2 chiffres.

Virgule flottante | **Précision finie et erreurs d'arrondi**

Exemple en simple précision :

- $0.1 = (1.100110011001100\ldots)_2 \cdot 2^{-4}$

- Le codage sur un nombre fini de bits nécessite de **tronquer** la série.

- $\text{dec}_{\mathbb{R}32}(\text{cod}_{\mathbb{R}32}(0.1)) = 0.100000001490116119384765625$

Virgule flottante | **Précision finie (exemples de code)**

En C :

```c
#include <stdio.h>
int main()
{
    float a = 0.1f;
    printf("Valeur : %.27f\n", a);
    return 0;
}
```

En Python:

```python
a = 0.1
print(f"{a:.27f}") #on demande d'afficher 27 décimales
```

Virgule flottante | **Piège des comparaisons (C)**

```c
#include <stdio.h>
int main(){
    printf("Le programme commence.\n");
    float val = 1.0f;
    for (int i = 0; i < 10; i++)
    {
        val -= 0.2; //on soustrait 0.2 à val
        if(val == 0) //ceci devrait se produire après 5 étapes (ou pas?)
        {
            printf("La valeur est nulle\n");
        }
    }
    printf("Le programme est terminé.\n");
    return 0;
}
```

(Diapositive hors champ)

Virgule flottante | **Piège des comparaisons (Python)**

```
val = 1.0
while val != 0:
    val -= 0.2 # Essayez avec 0.25
print("Programme terminé.")
```

Virgule flottante | **Accumulation des erreurs (C)**

L'erreur d'arrondi peut se propager à travers des calculs répétés.

```
#include <stdio.h>
int main()
{
    const float delta = 0.2; // increment (essayer 0.25f aussi!)
    const long N = 10000000; // nb d'additions
    float a = 0.; // compteur
    const float theo_result = delta * N;
    printf("delta * N = %f * %ld = %f\n", delta, N, theo_result);
    for(long i=0; i<N; i++)
        a = a + delta;
    printf("delta + delta + ... (N fois) = %f\n", a);
}
```

(Diapositive hors champ)

Virgule flottante | **Accumulation des erreurs (Python)**

L’erreur d’arrondi peut se propager à travers des calculs répétés.

```
tot:float = 0.
n:int = int(1e7)
increment:float = 0.3
expected:float = n * increment
for i in range(n):
    tot += increment
print(f"Attendu: {expected}, tot: {tot}, rel: {(expected - tot)/expected}")
```

Virgule flottante | **Annulation catastrophique**

- Exemple frappant lié aux erreurs d'arrondi en général (pas forcément en programmation).
- On chronomètre deux coureurs : A = 9.44 s et B = 9.56 s.
  $\Rightarrow \Delta = 0.12$ s.
- Si on utilise un chronomètre précis au dixième, alors A' = 9.4 s et B' = 9.6 s.
  $\Rightarrow \Delta' = 0.2$ s.

Virgule flottante | **Annulation catastrophique**

- On chronomètre deux coureurs : A = 9.44 s et B = 9.56 s.
  $\Rightarrow \Delta = B - A = 0.12$ s.
- Si on utilise un chronomètre précis au dixième, alors $A_c = 9.4$ s et $B_c = 9.6$ s.
  $\Rightarrow \Delta' = 0.2$ s et alors $\Delta' / \Delta \approx 167$ % !
- Pourtant $A' / A \approx 99.6\%$ et $B' / B \approx 100.4\%$.
- Soustraire deux **bonnes** approximations de nombres proches entre eux peut mener à une **mauvaise** approximation du résultat.

Virgule flottante | **Annulation catastrophique**

- Deux nombres proches entre eux donnent un résultat proche de zéro lorsqu'on les soustrait.
- Par conséquent, les bits issus de l'arrondi endossent une importance relative plus grande que pour des nombres plus éloignés de zéro.
- Dans certains cas de programmation scientifique, il est important de réfléchir aux potentielles erreurs d'arrondi et, si besoin, trouver des solutions pour les contourner.
- Exemple : Beaucoup d'équations physiques ont la forme d'une relaxation vers un équilibre : $x_{t+1} \sim k(x_t - x_0)$.

(Diapositive hors champ)

Exemple de code avec annulation catastrophique (C)

```c
// NB : essayez la version double également !
#include <stdio.h>
int main() {
    float a = 1.000001;
    float b = 1.000000;
    float c = 0.000001;

    float result = (a - b) / c;
    printf("Ceci devrait valoir 1: %.12f\n", result);

    return 0;
}
```

(Diapositive hors champ)

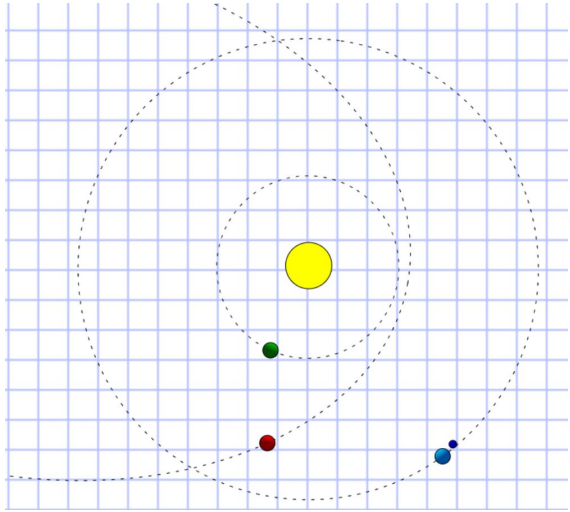
Exemple de code avec annulation catastrophique (Python)

```
a = 1.000001
b = 1.000000
c = 0.000001
result = (a - b) / c #devrait valoir exactement 1
print(f"Ceci devrait valoir 1: {result:.12f}")
```

Écart entre nombres

Comment coder la position d'un personnage dans un jeu en monde ouvert ?

Ou d'une planète dans une simulation physique ?



Écart entre nombres

- Plus grande valeur représentable avec un **entier non signé** sur 32 bits : ~ 2^{32}
- Plus grande valeur représentable en **simple précision** : ~ 10^{38}
- Est-ce étrange ?

- Le "prix à payer" de la représentation en virgule flottante est que le poids des bits dépend de la valeur de l'exposant ⇒ **l'écart entre nombres n'est pas constant, mais proportionnel à la valeur du nombre codé !**

Écart entre nombres | **Calcul de l'écart**

Écart entre nombres | **Exemple en Python**

```
dx = 1e-8

x = 1e8
print(f"{x + dx:.23f}") #100000000.00000001…

x = 1e9
print(f"{x + dx:.23f}") #1000000000.0000000…
```

Écart entre nombres | **Exemple en C**

```c
#include <stdio.h>
int main() {
    float x = 10000; // E=9, km=23
    float dx = 1e-2; // Essayez de varier ceci
    float newX = x + dx;
    printf("%.7f", newX);
    return 0;
}
```

Illustration pratique du problème de l'écart entre nombres