

Inf 431 – Cours 17

# Calcul flottant Allocation mémoire

jeanjacqueslevy.net

secrétariat de l'enseignement:  
Catherine Bensoussan  
cb@lix.polytechnique.fr  
Aile 00, LIX,  
01 69 33 34 67

www.enseignement.polytechnique.fr/informatique/IF

# Flottants

## Plan

1. Représentation flottante
2. Erreurs dans les calculs flottants
3. Représentation IEEE 754
4. Allocation mémoire
5. Glâneur de cellules

### Bibliographie

David Goldberg, *What Every Computer Scientist Should Know About Floating-Point Arithmetic*, ACM Computing Surveys, 1991.

W. Kahan, *Why do we need a floating-point arithmetic standard ?*, UC Berkeley, Mars, 1981.

Vincent Lefèvre, Paul Zimmermann, *Arithmétique flottante*, INRIA Lorraine, 2004.

Jean-Michel Muller, *Le mauvais résultat tout de suite, ou le bon résultat trop tard*, ENS Lyon, 2004.

G. Dowek, J.J.L., *Allocation mémoire*, Langages de Programmation, Majeure 2 d'Informatique, Ecole polytechnique, 2004.

## Représentation des nombres (1/2)

- **entiers** : complément à 2 sur  $n$  bits ( $n = 8, 16, 32, 64$ )  
 $-2^{n-1} \leq x < 2^{n-1}$  (arithmétique modulo  $2^n$ )



$s = 0 \Rightarrow$  nombre positif  $v$

$s = 1 \Rightarrow$  nombre négatif  $v - 2^{n-1}$

$2^{31} = 2\,147\,483\,648$        $2^{63} = 9\,223\,372\,036\,854\,775\,808$

- **réels** : virgule fixe (par exemple  $p = |su| = 16, q = |v| = 16$ )



$x = su + v/2^q$  et réciproquement  $su \leftarrow \lfloor x \times 2^q \rfloor$

$-2^{p-1} \leq x \leq 2^{p-1} - 1/2^q$

$2^{15} = 32768$

Très utilisé en graphique interactif, car opérations entières.

- **réels** : flottant sur 32 ou 64 bits

mantisse  $m$  sur 24 ou 53 bits,

$$x = m 2^e$$

exposant  $e$  sur 8 ou 11 bits



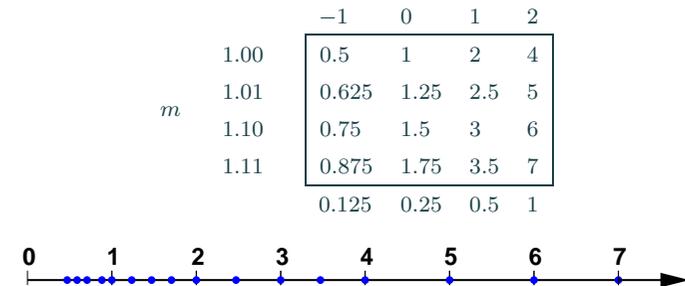
$2^{-127} \times 2^{-24} \leq |x| < 2^{128}$  pour le calcul numérique.

## Représentation des nombres (2/2)

- entiers : précision arbitraire  
représentation en tableaux, listes, structures dynamiques, ...  
beaucoup de bibliothèques de *bignums* (GMP, Maple, ocaml)  
les opérations dépendent du nombre de chiffres  
efficacité ?
- réels : exacts  
impossible, puisque l'égalité  $a = b$  est indécidable  
seules des approximations sont possibles
- réels : par intervalles

## Représentation flottante (2/2)

- la représentation flottante n'est pas unique  
 $0.000214 \times 10^2 = 0.214 \times 10^{-1}$
- la **représentation normalisée** est  $2.14 \times 10^{-2}$   
(premier chiffre  $d_0$  non nul)
- la représentation normalisée de 0 utilise des valeurs réservées de  $e$  :  
 $1.0 \times \beta^{e_{min}-1}$  (d'autres valeurs de  $m$  seront utilisées pour  $\pm\infty$ )
- $\beta = 2, p = 3, e_{min} = -1, e_{max} = 2 \Rightarrow 16$  représentations normalisées



## Représentation flottante (1/2)

- Réels en **notation scientifique**  
 $2.439 \times 10^5, -4.0019 \times 10^7, 2.23 \times 10^{-4},$   
 $\pm m \times 10^e$   
 $\pm m$  partie **significative** (mantisse)  $0 \leq m < 10$   
 $e$  **exposant**
- Avec une **base  $\beta$  arbitraire**  
 $\pm m \times \beta^e = \pm d_0.d_1d_2 \dots d_{p-1} \times \beta^e$   
vaut  
 $\pm(d_0 + d_1\beta^{-1}d_2\beta^{-2} \dots d_{p-1}\beta^{-(p-1)}) \times \beta^e \quad (0 \leq d_i < \beta, 0 \leq i < p)$
- $\beta = 10$  (exemples),  $\beta = 2$  (en machine)  
 $p$  nombre de chiffres significatifs  
 $e_{min} \leq e \leq e_{max}$
- le nombre de chiffres significatifs est **fixe**  
 $\Rightarrow$  **approximation** des nombres réels.
- 0.1 non représentable si  $\beta = 2$   
 $x$  tel que  $|x| < \beta^{e_{min}}$  ou  $|x| > \beta^{e_{max}}$  non représentable

## Erreur (1/4)

- erreur absolue** d'arrondi par rapport au nombre réel exact

$$\beta/2 \times \beta^{-p} \times \beta^e$$

- erreur relative** (*ulp* = *unit in the last place*)

$$\frac{1}{2}\beta^{-p} \leq \frac{1}{2}ulp \leq \frac{\beta}{2}\beta^{-p}$$

La **précision**  $\epsilon$  de la machine est définie par

$$\epsilon = \frac{\beta}{2}\beta^{-p}$$

- les opérations flottantes produisent des erreurs.
- l'opération la plus redoutable est la **soustraction**

## Erreur (2/4)

- $\beta = 10, p = 3$

réel	flottant
$x = 2.15 \times 10^{12}$	$2.15 \times 10^{12}$
$y = 0.000000000000000125 \times 10^{12}$	$2.15 \times 10^{12}$
$x - y = 2.149999999999999875 \times 10^{12}$	$0.00 \times 10^{12}$
$x' = 10.10$	$1.01 \times 10^1$
$y' = 9.93$	$0.99 \times 10^1$
$x' - y' = 0.17$	$0.02 \times 10^1$
$x'' = 1.000 \times 10^e$	$1.00 \times 10^e$
$y'' = 0.999 \times 10^e$	$0.99 \times 10^e$
$x'' - y'' = 0.001 \times 10^e$	$0.01 \times 10^e$

- l'erreur relative vaut  $(\beta^{-p+1} - \beta^{-p})/\beta^{-p} = \beta - 1$
- dans une soustraction, l'erreur relative est bornée par  $\beta - 1$   
 $\beta = 2 \Rightarrow$  **erreur absolue aussi importante que le résultat!**

## Erreur (4/4)

- $4ac \ll b^2 \Rightarrow$  on change de formules :

$$\frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad \frac{2c}{-b - \sqrt{b^2 - 4ac}}$$

$$\frac{-b - \sqrt{b^2 - 4ac}}{2a} \quad \frac{2c}{-b + \sqrt{b^2 - 4ac}}$$

- Règle : un test entre flottants n'a aucun sens si l'erreur de  $|a - b|$  n'est pas estimée :  

```
float a, b;
if (a > b) ...
else ...
```
- bugs célèbres dus au calcul flottant: missile Patriot (1991), plate-forme Sleipner A (1991), FDIV Intel (1994), Ariane 501 (1996), etc.

Exercice 3 L'addition flottante est-elle commutative ?

Exercice 4 L'addition flottante est-elle associative ?

## Erreur (3/4)

- si on ajoute un chiffre de précision pendant la soustraction (une garde), l'erreur relative baisse à  $2\epsilon$
- pour calculer la précision  $\epsilon$  Exécution

```
static void calculerPrecision {
    double a = 1.0, b = 1.0;
    int i = 0;
    while (((a + 1.0) - a) - 1.0 == 0.0) {
        a = 2*a; ++i;
    }
    System.out.println(i);
    while (((a + b) - a) - b != 0.0) ++b;
    System.out.println(b);
}
```

- l'ordre dans lequel on fait les opérations est important  
 $\Rightarrow$  recettes numériques

Exercice 1 Calculer la précision en float.

Exercice 2 Montrer que la série harmonique converge en flottant.

## Représentation IEEE 754

- |   |   |   |
|---|---|---|
| s | k | f |
|---|---|---|

  - s signe (1 bit)
  - e exposant (8 bits) ( $-126 \leq e \leq 127$ )
  - f partie fractionnaire (23 bits) ( $0 \leq m < 2^{23}$ )

- valeurs de  $1.1754943508222875 \times 10^{-38}$  à  $3.402823466385288600000 \times 10^{-38}$

- conventions

$e = k - 127$	$f$	$m$
$-126 \leq k \leq 127$	$0 \leq f \leq 2^{23}$	$\pm 1.f \times 2^e$
128	0	$\pm \infty$
128	$\neq 0$	NaN
-127	0	$\pm 0$
-127	$\neq 0$	$\pm 0.f \times 2^{-126}$ (dénormalisé)

## Représentation IEEE 754

- |   |   |   |
|---|---|---|
| s | k | f |
|---|---|---|

s signe (1 bit)  
e exposant (11 bits) ( $-1022 \leq e \leq 1023$ )  
f partie fractionnaire (52 bits) ( $0 \leq m < 2^{52}$ )
- valeurs de  $2.22507385855072014 \times 10^{-308}$  à  $1.79766931348623157 \times 10^{308}$
- conventions

$e = k - 1023$	$f$	$m$
$-1022 \leq k \leq 1023$	$0 \leq f \leq 2^{52}$	$\pm 1.f \times 2^e$
1024	0	$\pm \infty$
1024	$\neq 0$	<i>NaN</i>
-1023	0	$\pm 0$
-1023	$\neq 0$	$\pm 0.f \times 2^{-1022}$ (dénormalisé)

## Allocation mémoire (1/4)

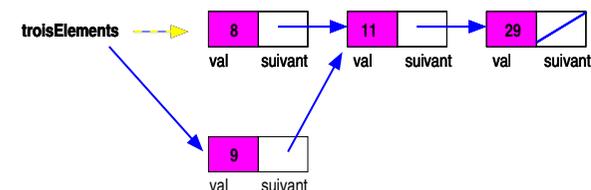
- les variables (statiques) globales et les variables locales ont une durée de vie n'excédant pas celle du bloc de leur déclaration.
- les fonctions appellent d'autres fonctions; de nouvelles variables locales sont créées, puis disparaissent.
- à chaque appel de fonction, un bloc d'activation contenant ses variables locales est créé (*stack frame*); il est supprimé au retour de la fonction.
- ces blocs ont une structure de pile.
- les variables (statiques) globales et les variables locales sont encore appelées **variables de la pile**.

## Allocation mémoire

## Allocation mémoire (2/4)

- d'autres données sont créés par `new`.
- leur durée de vie peut être aussi longue que le programme.
- ce sont les **données du tas**
- pourtant elles peuvent être inutiles à partir d'un certain moment

```
Liste troisElements =
    new Liste(8, new Liste(2, new Liste(29, null)));
troisElements.suivant.val = 11;
troisElements = new Liste(9, troisElements.suivant);
```

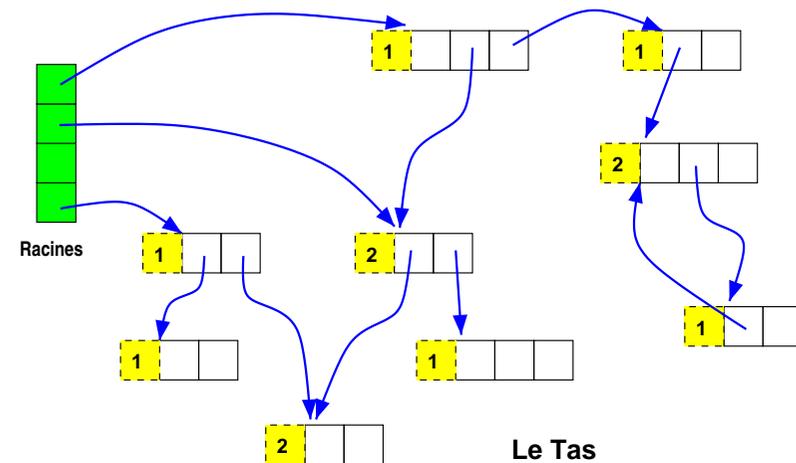


## Allocation mémoire (3/4)

- le GC, glâneur de cellules (*garbage collector*) récupère l'espace mémoire des données inutiles dans le tas.
- le GC est **intégré** à l'interpréteur de la JVM.
- il existe de **multiples** manières de faire le GC.  
Le GC de Java doit être cohérent avec l'existence de **plusieurs processus** (*multi-thread*).
- en général, le GC est appelé quand une allocation de mémoire (*new*) détecte qu'il **n'y a plus de place** dans le tas.
  - le GC récupère alors toute la place inutile ;
  - l'allocation dans le tas devient **possible**.
  - Si toujours pas de place, on appelle le **système d'exploitation** (Linux, Windows) pour augmenter la taille de la JVM.
  - Si impossible, on s'arrête pitoyablement.
- le GC provoque un **temps d'arrêt** dans l'exécution des programmes (sauf si le GC est incrémental — plus compliqué —)

## Glâneur de cellules (1/2)

Compteurs de références : chaque donnée du tas a un compteur de références.



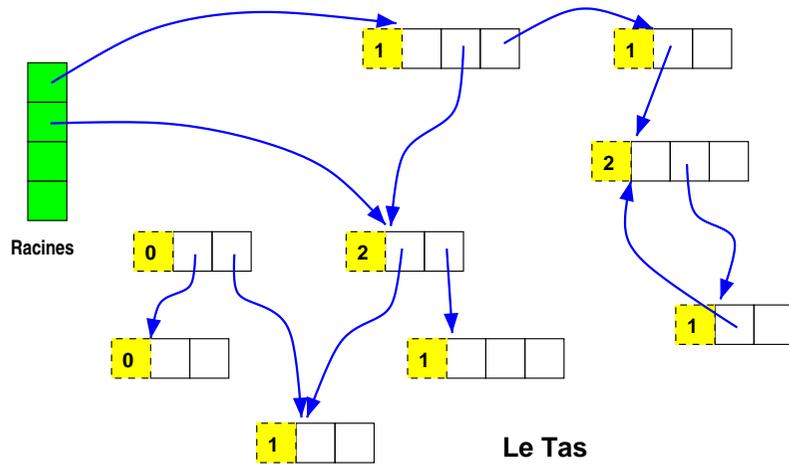
## Allocation mémoire (4/4)

- pas de GC dans les vieux langages de programmation (Algol, Pascal, C, C++)  
⇒ déallocation manuelle
- les langages de programmation modernes ont un GC (Scheme, ML, Java, Modula-3)
- GC ⇒ déallocation automatique de la mémoire du tas  
GC ⇒ programmation grandement **simplifiée**
- on peut contrôler l'allocation dans le tas, puisqu'elle est toujours explicite (*new*).

## Glâneur de cellules (2/2)

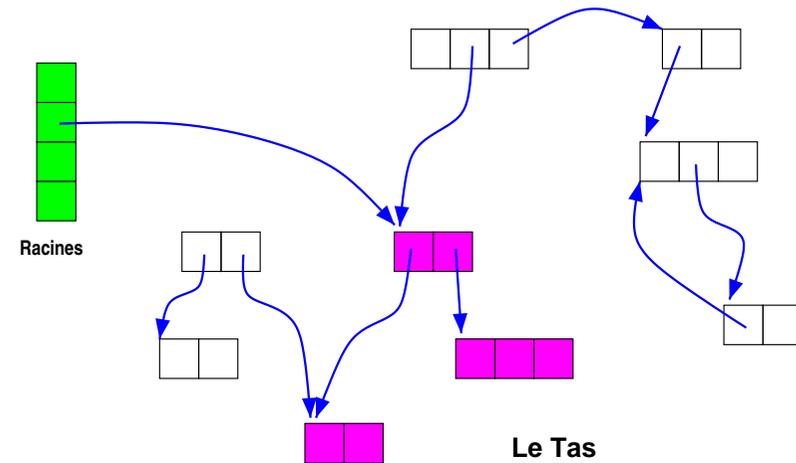
- le compteur de référence est mis à jour à **chaque affectation**.
- Si le compteur d'une cellule devient **zéro**. On libère la cellule, et on met à jour les compteurs des cellules pointées par la cellule libérée.
- méthode incrémentale. Pas de blocage. Bien pour le temps-réel.
- mais ne récupère pas les cycles.
- coût induit sur toutes les opérations de manipulation des références.

## Compteurs de références (1/2)

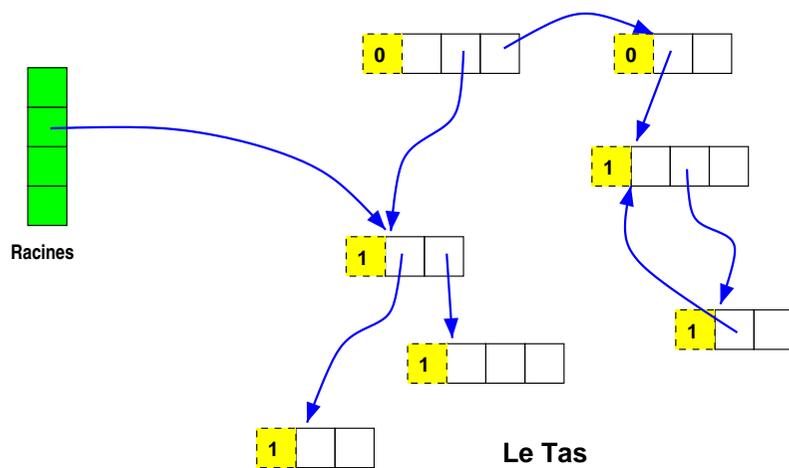


## Méthodes par traçage (1/2)

On marque les cellules atteignables depuis les racines. Et on libère les cellules non marquées.



## Compteurs de références (2/2)



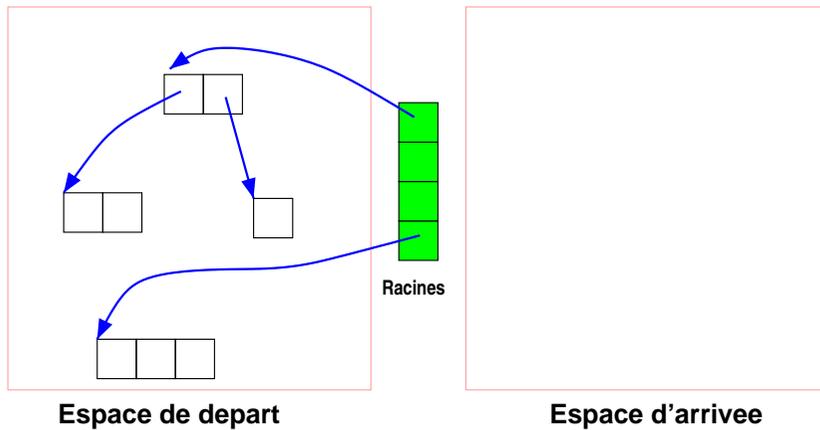
## Méthodes par traçage (2/2)

- La phase de marquage se fait par une simple **exploration en profondeur** du graphe d'accès à partir des racines. Son résultat se fait en général dans un **tableau de bits**, 1 bit par mot-mémoire.

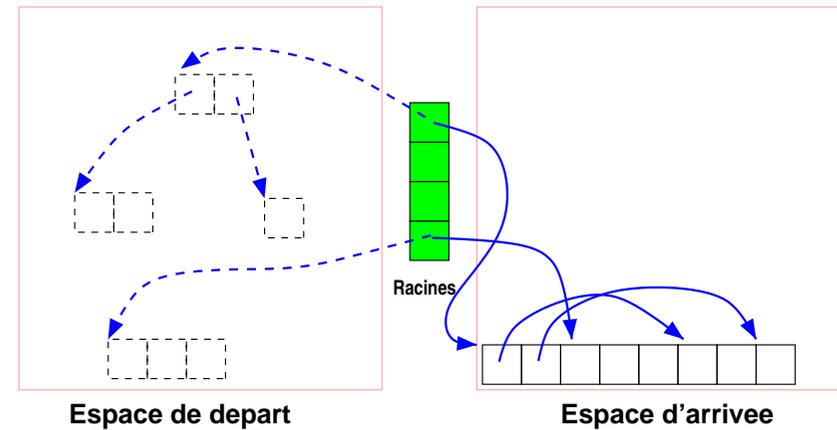
La deuxième phase consiste à **balayer séquentiellement** le tas en récupérant tous les mots-mémoire non marqués.

- tourne en peu de mémoire
- autorise les GC conservatifs (pour C ou C++).
- problème  $\equiv$  **fragmentation**
- $\frac{c_1 N + c_2 T}{T - N}$  (où  $T$  taille du tas,  $N$  mots occupés)

## Collecteurs par recopie (1/5)



## Collecteurs par recopie (3/5)



## Collecteurs par recopie (2/5)

- tas divisé en 2 zones égales :
  - départ (*from-space*)
  - arrivée (*to-space*).

On alloue des cellules dans la zone de départ à partir d'un index `disponible` indiquant l'emplacement suivant le dernier mot alloué.

Quand `disponible` est au bout de la zone, on **recopie** les cellules accessibles depuis les racines de la zone départ vers la zone d'arrivée.

- Les cellules recopiées sont **compactées**. On permute les rôles des deux zones, l'index `disponible` se retrouvant derrière le dernier mot recopié.  
⇒ pas de fragmentation
- il faut disposer de la **moitié de la mémoire** (en fait on alloue de la mémoire virtuelle, mais cela peut entraîner des défauts de page au moment du GC).

## Collecteurs par recopie (4/5)

- il faut mettre à jour les contenus des cellules recopiées pour que les références pointent dans la zone d'arrivée.
- Dans l'algorithme de [Cheney, 70], la recopie se fait par un parcours en largeur du graphe des cellules accessibles à partir des racines.
- Quand on recopie une cellule de la zone de départ vers la zone d'arrivée, on laisse un renvoi vers sa nouvelle adresse dans le premier mot de cette cellule dans la zone de départ.
- Pour le parcours, on utilise une file dont les index de début `debut` et de fin `disponible` (en fait le premier mot derrière la file) sont initialement positionnés sur le début de la zone d'arrivée.
- On recopie d'abord les mots directement pointés par les racines, que l'on range dans la file.
- Tant que la file n'est pas vide, on recopie les mots pointés par le premier élément de la file.

## Collecteurs par recopie (5/5)

- Pour copier une cellule, on regarde d'abord si son premier mot pointe vers la zone d'arrivée. Si c'est le cas, rien à faire.
- Sinon, on copie la cellule au bout de la file dans la zone d'arrivée. Et on met un pointeur de renvoi dans le 1er mot de son ancienne adresse.

$\frac{cN}{T/2-N}$  (où  $T$  taille du tas,  $N$  mots occupés)

# A l'année prochaine !

## Langages de programmation

- GC par générations (2 en Ocaml)
  - GC incrémentaux
  - GC concurrents
  - GC distribués
  - polymorphisme
  - modules paramétrés
  - certification du code généré
  - analyse de flot et interférences, etc.
- ⇒ cf. en Majeure 1 (cours Langages de programmation)

L'informatique est **diverse** et **intéressante** ;  
c'est aussi devenu la **première** industrie mondiale ;  
allez en apprendre **plus** en **Majeures d'informatique**.