NSI | 02

NOMBRES SIGNÉS, NOMBRES RÉELS

[Stéphane BEAUDET – Frédéric PEURIERE]

Savoir comment sont représentés les entiers relatifs en base 2 et être capable de les manipuler
Connaître la représentation approximative des nombres réels (à virgule flottante) et en mesurer les conséquences (capacité et arrondis)

Les entiers relatifs positifs sont donc codés en utilisant la décomposition binaire vue dans la partie précédente. Les entiers relatifs négatifs ou entiers signés seront codés grâce à la méthode du **complément à deux**

1. Représentation binaire d'un entier signé

1.1. Notion de bit de signe

Le bit de signe sera le premier bit de la représentation : 0 pour un positif et 1 pour un négatif

0								: entier relatif positif
1								: entier relatif négatif

Sur n bits, on code les entiers de -2^{n-1} à $2^{n-1}-1$: expliquer ….....

….....

En Python, c'est le type *integer* (entier en français) pour les positifs comme les négatifs

1.2 Méthode du complément à deux

Méthode	Exemple : écriture de -15
On écrit le nombre en valeur absolue en binaire (→ complément à 1)	$(15)_{10}=(00001111)_2$ sur 8 bits
On inverse les bits de cette écriture ($0 \leftrightarrow 1$)	$(11110000)_2=(240)_{10}$
On ajoute 1 donc $(00000001)_2$ sur 8 bits en tenant compte des éventuelles retenues (1+1=1 0)	$(11110001)_2=(241)_{10}$ et 241 – 256 = -15

2. Représentation approximative des nombres réels

En Python, c'est le type *float* (flottant en français).

2.1. On teste

Expliquer ce que fait la commande suivante en Python, tapée dans le shell si a et b sont deux nombres : `>>> a - b = = 0`

Taper puis noter ce que renvoie Python :

`>>> 4 - 4 = = 0` `>>> 4 - 4.0 = = 0` `>>> 0.2 + 0.8 -1 = = 0` `>>> 0.2 + 0.6 – 0.8 = = 0`

`>>> 0.1 + 0.5 – 0.6 = = 0` `>>> 0.1 + 0.2 - 0.3 = = 0` `>>> 0.7 + 0.2 - 0.9 = = 0`

Êtes vous surpris ? Pourquoi ?

Cherchez d'autres cas surprenants et écrivez en quelques uns ici :

Il faut donc ÉVITER de TESTER l'égalité de deux flottants

Afin de remédier à cela, nous allons effectuer un test du type $|a-b|<\epsilon$ où ϵ est une valeur proche de zéro à définir :

```
>>>print (abs(0.1+ 0.2 - 0.3)<1E-5)        # test adapté aux flottants
```

Que renvoie le shell ? Quelle information cela donne t il ?

Écrire une fonction ***Egal*** qui prend en argument 3 flottants a et b ,dont on fera la somme, et y, comparé à a+b en demandant à quelle précision ***eps*** (10^{-5} dans l'exemple) on teste à « eps près » l'égalité des deux flottants (a+b et y).

Tester cette fonction avec x=0.1+0.2 et y=0.3 et différentes valeurs de eps....

Ces « erreurs » proviennent de la représentation !

2.2. Écriture à virgule flottante

Décomposer dans la base 10, le nombre 3,3125 :

Puissance de 10	10^0	10^{-1}	10^{-2}	10^{-3}	10^{-4}
Valeur décimale de puissance					
Nombre base 10					

Ainsi $3,3125 = ...\times 10^0 + ...\times 10^{-1} + ...\times 10^{-2} + ...\times 10^{-3} + ...\times 10^{-4}$

Par analogie avec les décimaux, on peut écrire un nombre à virgule en notation binaire en utilisant les puissances négatives de 2 :

Puissance de 2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
Valeur décimale de puissance						
Nombre base 2	1	1	0	1	0	1

$(11,0101)_2 = 1\times2^1+1\times2^0+0\times2^{-1}+1\times2^{-2}+0\times2^{-3}+1\times2^{-4}$

$=.....+.....+.....+.....+.....+.....=.....,.....$

Nous allons encoder le nombre $(10,59375)_{10}$ en base 2 (**REFAITES CE CALCUL**)

Pour la partie entière $(10)_{10}=(1010)_2$

Pour la partie décimale $(0,59375)_{10}=(,10011)_2$ car 0,59375 x 2 = 1,1875 = **1** + 0,1875

0,1875 x 2 = 0,375 = **0** + 0,375

0,375 x 2 = 0,75 = **0** + 0,75

0,75 x 2 = 1,5 = **1** + 0,5

0,5 x 2 = 1 = **1** + 0

la partie décimale est nulle : on s'arrête

On vérifie que $(0,59375)_{10}=2^{-1}+2^{-4}+2^{-5}$ Nous avons donc $(10,59375)_{10}=(1010,10011)_2$

Mais cette méthode devient très vite trop complexe (pour les très petits ou très grands nombres)

On utilise donc la NOTATION SCIENTIFIQUE d'un décimal :

$s\ m\ .\ 10^n$ où s est le signe (+ ou -) , m est la mantisse (un nombre décimal) et n l'exposant (un entier relatif)

Par exemple : 173,95 = + 1,7395 . 10^2 = + 17,395 . 10^1

La virgule « flotte » de gauche à droite dans la mantisse

2.3. Écriture à virgule flottante en base 2

Le principe est le suivant : sur n bits, il y aura 1+ p + e bits occupés

$$\underbrace{s}_{\text{signe}} \cdot \left(\underbrace{x_1,\ x_2\ \ldots\ x_{p-1}\ x_p}_{\text{mantisse}} \right) \cdot \underbrace{2^e}_{\text{exposant}}$$

La norme IEEE-754 définit donc

Encodage	Signe s	n	Mantisse m	Valeur
32 bits	1 bit	8 bits	23 bits	$(-1)^s.(1+m).2^{(n-127)}$
64 bits	1 bit	11 bits	52 bits	$(-1)^s.(1+m).2^{(n-1023)}$

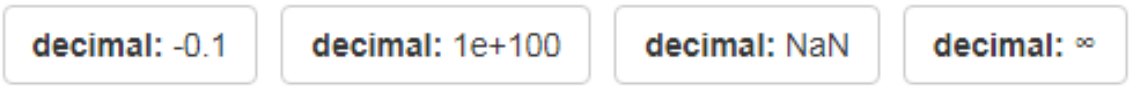
ATTENTION : le premier bit de la mantisse d'un nombre normalisé est **forcément 1**, il n'est donc pas représenté (mais **écrit** dans le tableau des valeurs)

Connectez vous alors à https://baseconvert.com/ieee-754-floating-point

Que pouvez vous dire de la représentation de 0 ?.....

Copier la représentation décimale de 0.1 en 32 bits (on peut faire de même en 64 bits) et coller la dans une calculatrice (par exemple https://web2.0calc.fr/)
Additionner de même la représentation de 0.2. Enfin comparez cette valeur avec celle de 0.3 (en faisant pas exemple une soustraction) Expliquez les messages précédents.

Testez aussi les exemples proposés en bas de page



Dépassement de capacité

Les nombres à virgule flottante sont définis sur un nombre fini de bits : il y a un nombre maximal représentable dans ce format.

Arrondis et conséquence

Rare sont les nombres décimaux représentables exactement en virgule flottante

Ainsi, ces arrondis font que l'égalité n'a pas de sens