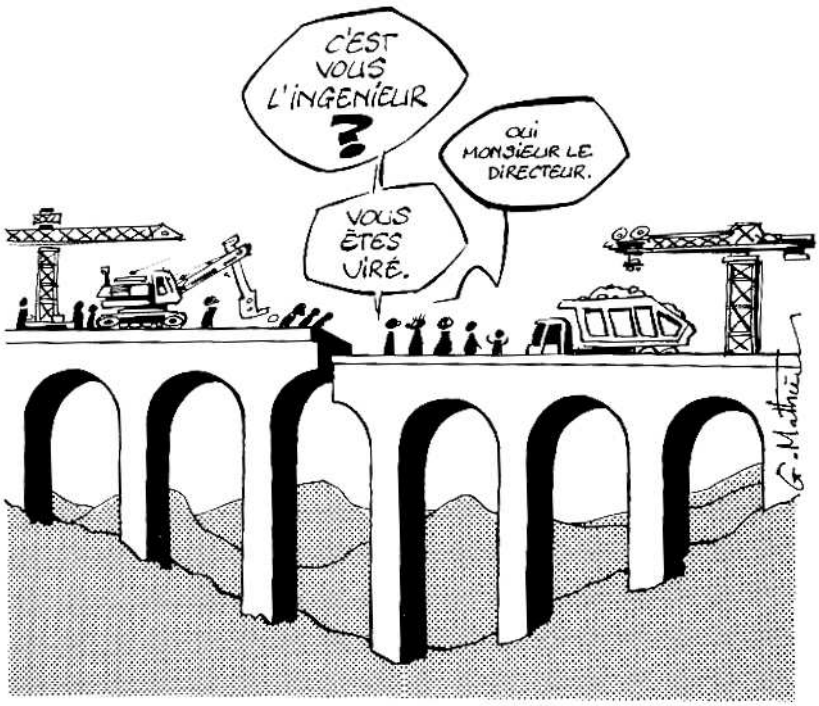
CHAPITRE

11 Codage des décimaux en machines

Table des matières



1 Introduction

Le langage C est un langage fortement typé : la déclaration des variables est beaucoup plus rigoureuse qu'en Python.

```
1 int jour = 5;
2 long valeur = 145265;
```

La déclaration d'une variable passe d'abord par la déclaration de son ***type***. Il permet d'indiquer au ***compilateur*** quel genre de données nous souhaitons stocker et réserver un ***espace mémoire*** conséquent.

Le tableau suivant donne quelques informations sur les types dans le langage C :

Type	***Signification***	***Taille*** en octets	***Plage des valeurs possibles***
`int`	Entier	2 ou 4 selon la taille du processeur	-32768 à 32767 ou -2 147 483 648 à 2 147 483 647
`unsigned int`	Entier non signé	2 ou 4 selon la taille du processeur	0 à 65535 ou 0 à 4 294 967 295
`long int`	Entier long	4	-2147483648 à 2147483647

Avec n bits, on peut :

- représenter les entiers positifs de 0 à $2^n - 1$.
- représenter les entiers positifs et négatifs de -2^{n-1} à $2^{n-1} - 1$.

Exercice n°1

On considère une architecture où les entiers longs signés sont codés sur 4 octets.

1. Donner la valeur décimale du plus grand entier qu'on peut coder. Donner ensuite sa valeur binaire.
2. Donner la valeur décimale du plus petit entier qu'on peut coder. Donner ensuite sa valeur binaire.
3. Quelle est la représentation binaire de 295 421 ?
4. Et celle de $-25\,814$?

Le nombre d'octet imposé par la norme permet de stocker des entiers même très grands.

Mais quel que soit l'architecture utilisée, nous serons toujours limité en capacité. L'ensemble $\mathbb{N}$ des entiers naturels, et pire, l'ensemble $\mathbb{Z}$ des entiers relatifs, sont des ensembles ***infinis***.

Dans un ordinateur, on ne peut stocker qu'un nombre ***fini*** d'entiers.

L'attention du programmeur est alors portée sur la manipulation de ces valeurs dans ses programmes : que se passe t-il si des opérations mathématiques sur des entiers, donnent un résultat en dehors de la plage de représentation ?

L'***overflow*** ou l'***underflow*** sont les noms donnés au ***dépassement*** de mémoires.

Le comportement de votre processeur en cas de dépassement est en général anticipé même si il est difficile d'envisager tous les cas d'erreurs.

Exercice n°2
Supposons une architecture de 4 bits.Que fait alors la somme de `1011` avec `1100` ?

La version 3 de Python est capable de dépasser automatiquement le format 64 bits pour le stockage des entiers en passant sur un format `long`.

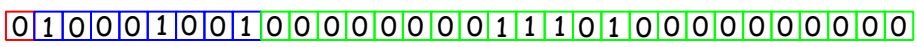
2 Codage des flottants en machine

On appelle ***flottant*** tout nombre ***décimal***, c'est-à-dire, tout nombre dont le développement décimal est ***fini***.

On s'intéresse ici à la façon dont les ***flottants*** sont stockés en mémoire sans théorie excessive...

Un flottant est stocké selon la norme IEEE754

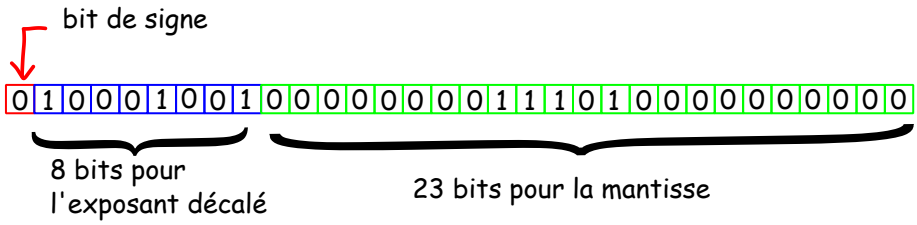
Voici un flottant !Il est beau , non ?



Vous remarquerez :

- qu'il y a ***32 bits*** ; il s'agit d'un flottant en ***simple précision*** !

- qu'il y a trois champs distincts : le premier pour le ***signe***, le deuxième pour l'***exposant*** et le troisième pour la ***mantisse***



Ainsi :

- Il s'agit d'un nombre ***positif*** car le bit de signe S est à 0 ;
- l'***exposant décalé*** vaut $E_d = 137$
- la ***mantisse*** vaut 00000000111010...

Ce codage représente alors le flottant :

$$x = (-1)^S \times 1,\ 00000000111010... \times 2^{E_d - 127}$$

c'est-à-dire :

$$x = 1,\ 00000000111010 \times 2^{10}$$

Mais comme la multiplication par des multiples de 10 fait décaler la virgule des nombres décimaux, la multiplication par des multiples de 2 fait décaler(il paraît qu'on dit « flotter... ») la virgule des nombres binaires. Ainsi :

$$x = 10000000011,1010$$

soit finalement :

$$x = 2^{10} + 2^1 + 2^0 + 2^{-1} + 2^{-3} = 1024 + 2 + 1 + 0.5 + 0.125 = 1027.625$$

? C'est quoi cette histoire d'exposant décalé ?

Les exposant décalés sont représentés par des ***entiers non signés*** sur 8 bits : leur valeur possible vont de 0 à 255. Donc l'exposant couvrira la plage d'exposant de -127 à 128.
La norme a choisi cette représentation pour la comparaison de deux nombres flottants de même signe.

À vous !

Exercice n°3
Quel est donc ce flottant codé par :

0	1000 0100	01100110000000000000000

i Le codage sur 32 bits des nombres flottants est appelée ***simple précision***. Il existe aussi la ***double précision*** qui permet de coder les flottants sur 64 bits(1 + 11 + 52).

Récapitulatifs :

	Signe	Exposant	Mantisse	Plus petit	Plus grand
Simple précision	1	8	23	$\pm 1.175 \times 10^{-38}$	$\pm 3.402 \times 10^{38}$
Double précision	1	11	52	$\pm 2.225 \times 10^{-308}$	$\pm 1.797 \times 10^{308}$

Le site `https://www.h-schmidt.net/FloatConverter/IEEE754.html` permet la conversion de décimaux selon la norme IEEE754.

Il existe d'autres normes qui peuvent aller jusqu'à 128 bits...

3 Comment coder un nombre décimal ?

Vous avez compris la norme qui permet le codage d'un nombre décimal, d'un flottant en machine. Mais concrètement comment trouver le codage d'un nombre décimal donné ?

Pour écrire un nombre flottant en respectant la norme IEEE754, il est nécessaire de commencer par écrire le nombre sous la forme $1,XXXX... \times 2^E$.

Prenons par exemple le nombre binaire $1010,11001$ dont l'écriture décimale est $10,78125$. Alors, on écrit :

$$1010,11001 = 1,0101101 \times 2^3$$

Donc :

- le bit de signe est 0 car le nombre est positif
- la mantisse est 01011010000000000000000 (23 bits pour elle !)
- et l'exposant décalé est 130 soit 1000 0010

D'où le codage selon la norme IEE754 :

0	1000 0010	01011010000000000000000

Exercice n°4

La représentation de ces 32 bits étant jugée... peu pratique, on donne sa version hexadécimale. Donnez la représentation hexadécimale de ce dernier nombre.

À vous !

Exercice n°5
Donner le codage selon cette norme du nombre binaire : $10001010,1001$

Mais peut-on coder n'importe quel décimal dans cette norme ?

La réponse est évidemment NON ! La norme ne permet de coder qu'un nombre ***fini*** de décimaux alors que l'ensemble des nombres décimaux en mathématiques $\mathbb{D}$ est ***infini*** , sans parler de l'ensemble des réels $\mathbb{R}$!

La difficulté réside dans le codage de la ***partie décimale***. Il existe un algorithme simple pour le réaliser :

Cette méthode consiste à multiplier la partie décimale par 2 jusqu'à obtenir 0 quand cela est possible ou, si on ne s'arrête pas, à obtenir assez de chiffres pour remplir la mantisse.

À chaque étape on garde le chiffre le plus à gauche du résultat (c'est à dire la partie entière) de la multiplication qui sera 1 ou 0, puis, on réitère la multiplication sur la partie décimale du résultat de la multiplication en supprimant le premier 1 s'il existe.

Un ***premier*** exemple !

Comment coder 0.4 ?

Le codage de 0.4 est alors :

$$0,0110011001100110011....$$

un développement périodique !

x	$2 \times x$	je garde...
0.4	0.8	0
0.8	1.6	1
0.6	1.2	1
0.2	0.4	0
0.4	0.8	0
0.8	1.6	1
...	...	...

Donc en respectant la norme IEEE754, le flottant 0.4 est codé par :

$$1,10011001100110011001100 \times 2^{-2} = 1,10011001100110011001100 \times 2^{125-127}$$

écriture qui fait donc apparaître les éléments dont nous avons besoin pour le codage dans la norme :

0	0111 11011	10011001100110011001100

En particulier, retenez que :

Le nombre 0.4 est représenté en mémoire par le nombre 0.40000000596046447753906250 !

Exercice n°6

En vous inspirant de la méthode précédente, coder les nombres décimaux 0.1, 2049.06 selon la norme IEEE754.

Le problème lié aux erreurs de précision implique que pour comparer deux valeurs en virgule flottante on ne peut pas utiliser l'opérateur d'égalité == comme on le ferait pour des entiers !

Il est nécessaire d'utiliser la valeur absolue de la différence des deux valeurs et de vérifier que cette différence est bien inférieure à un ϵ donné.

Historically, the Python prompt and built-in repr() function would choose the one with 17 significant digits, 0.10000000000000001. Starting with Python 3.1, Python (on most systems) is now able to choose the shortest of these and simply display 0.1.

Vous pouvez trouver les informations dont le système traite les flottants par l'instruction Python :

```
1 import sys
2 print(sys.float_info)
```