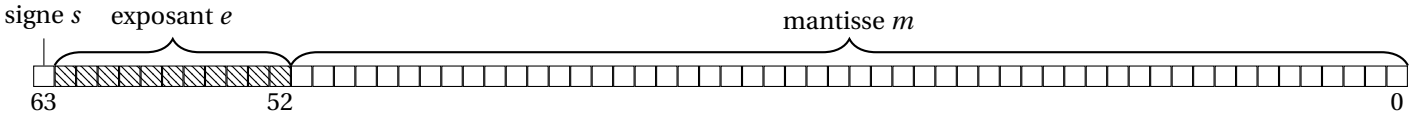
TD2 : Représentation des flottants en machine, norme IEEE 754

Le type `double` du `C` est défini dans la norme IEEE 754 [1] comme un *nombre à virgule flottante en double précision*. Il est codé sur 8 octets.
Cette norme repose sur un double constat :

- on manipule rarement de très grands nombres,
- plus les nombres manipulés sont grands, moins on se soucie de la précision.

Un nombre flottant est composé de trois éléments : son signe, son exposant et sa partie significative (liée à la mantisse), comme montré dans la figure ci-dessous.



Les valeurs de s, e et m sont lues en base 2. On appelle *biais* la valeur

$$b = 2^{10} - 1 = 1023\,,$$

où l'exposant 10 dans le calcul est obtenu par $11-1$, où 11 est le nombre de bits qui permettent de coder e. Ce biais sert à représenter des exposants négatifs sans passer par le complément à 2 (qui rendrait les comparaisons compliquées).
Le réel représenté par un nombre à virgule flottante en double précision est :

valeur de e	valeur de m	valeur du réel représenté	
0	0	± 0	zéros
0	$\neq 0$	$(-1)^s \times m * 2^{-52} \times 2^{-(b-1)}$	flottant *dénormalisé*
$2^{11}-1$	0	$\pm\infty$	
$2^{11}-1$	$\neq 0$	`NaN` (Not a Number)	
autre	quelconque	$(-1)^s \times \underbrace{(1 + m * 2^{-52})}_{\text{partie significative}} \times 2^{e-b}$	flottant *normalisé*

Le nombre $1 + m * 2^{-52}$ doit être vu comme 1 suivi d'une virgule puis des chiffres de m (en base 2), tandis que le nombre $m * 2^{-52}$ doit être vu comme 0 suivi d'une virgule puis des chiffres de m (en base 2).

Exercice 1 Quels réels (en base 10) sont représentés de la façon suivante en nombres à virgule flottante double précision par la norme IEEE 754 ?

1	1	0	0	0	0	0	0	0	0	0	0	1	1	0⋯0

0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	1	0⋯0

Exercice 2 Donner la représentation en nombre à virgule flottante en double précision (en suivant la norme IEEE 754) des nombres 11.28125 et 1035.28125.

Correction :
Pour rappel, il faut déterminer 3 valeurs :

- le signe (1 bit)
- l'exposant biaisé (11 bits)
- la mantisse (52 bits)

Codage de 11.28125
On note $\overline{n}^2$ une représentation en base 2 et $\overline{n}^2$ une représentation en base 10 (si rien n'est précisé, c'est de la base 10).
Le nombre est positif, donc le bit de signe est à 0.

1. IEEE : Institute of Electrical and Electronics Engineers

Notons $k = 11.28125$. On commence par écrire ce nombre en base 2, c'est-à-dire par l'exprimer comme une somme de puissances de 2 :

$$11.28125 = 2^3 + 2^1 + 2^0 + 2^{-2} + 2^{-5}.$$

En base 2, k s'écrit donc $\overline{1011.01001}^2$:

2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}
1	0	1	1	•	0	1	0	0	1

Comme ce nombre n'est pas très petit, on va pouvoir le code comme un flottant normalisé, sous la forme $1.\cdots \times 2^{\cdots}$:

$$\overline{1.01101001}^2 \times (\overline{2}^{10})^3.$$

Rappel : multiplier par 2^3 revient à décaler la virgule de 3 vers la droite en base 2.
La partie derrière la virgule dans cette écriture normalisée est la mantisse (complétée à droite par des 0).
L'exposant biaisé est 3 (puisqu'on multiplie par 2^3) donc $e - b = 3$ où $b = 2^{10} - 1$. Donc $e = 2^{10} + 2$, soit en base 2 :

$$e = \overline{10000000010}^2,$$

à compléter éventuellement par des 0 avant pour arriver à 11 bits (sur cet exemple c'est inutile).
Au final la représentation de k en virgule flottante est :

0	1	0	0	0	0	0	0	0	0	1	0	0	1	1	0	1	0	0	1	0⋯0

Codage de 1035.28125
Le nombre est positif, donc le bit de signe est à 0.
Notons $\ell = 11.28125$. On commence par écrire ce nombre en base 2, c'est-à-dire par l'exprimer comme une somme de puissances de 2 (voir l'explication à la fin de l'énoncé pour avoir un algorithme pour cette étape-là) :

$$1035.28125 = 2^{10} + 2^3 + 2^1 + 2^0 + 2^{-2} + 2^{-5}.$$

En base 2, ℓ s'écrit donc $\overline{10000001011.01001}^2$:

2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}
1	0	0	0	0	0	0	1	0	1	1	•	0	1	0	0	1

Comme ce nombre n'est pas très petit, on va pouvoir le code comme un flottant normalisé, sous la forme $1.\cdots \times 2^{\cdots}$:

$$\overline{1.00000010110100 1}^2 \times (\overline{2}^{10})^{10}.$$

La partie derrière la virgule dans cette écriture normalisée est la mantisse (complétée à droite par des 0).
L'exposant biaisé est ici 10 (puisqu'on multiplie par 2^{10}) donc $e - b = 10$ où $b = 2^{10} - 1$. Donc $e = 2^{10} + 8 + 1$, soit en base 2 :

$$e = \overline{10000001001}^2,$$

à compléter éventuellement par des 0 avant pour arriver à 11 bits (sur cet exemple c'est inutile).
Au final la représentation de k en virgule flottante est :

0	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	1	1	0	1	0	0	1	0⋯0

Exercice 3 Expliquer pourquoi le test

```
0.3 - 0.2 == 0.1
```

s'évalue à faux.

Correction :
Il ne suffit pas ici d'évoquer le fait qu'il y a des erreurs d'arrondi parce que certains nombres s'écrivent avec une infinité de chiffres après la virgule en base 2. Celà explique certes qu'il puisse y avoir des expressions qui devraient être égales et ne le sont pas, mais ne dit rien sur cette expression en particulier.
Il ne suffit même pas de donner les développements en base 2 des trois nombres qui interviennent dans l'opération pour montrer que ces trois nombres ont des développements infinis en base 2 : ça ne prouve toujours pas qu'il y a une erreur d'arrondi sur l'opération.
Une seule solution pour démontrer le résultat : donner les représentations des trois nombres en `double`, et

faire le calcul pour exhiber l'erreur d'arrondi. En fait il suffit de pointer un chiffre qui ne sera pas le même, et clairement l'erreur se produit le plus loin possible de la virgule. On va donc essayer de comprendre la fin de la représentation en `double` de ces trois nombres.
Les représentations en base 2 de 0.3, 0.2 et 0.1 valent (voir à la fin pour comprendre comment on les obtient) :

$$\overline{0.1}^{10} = \overline{0.0\underbrace{0011}_{\text{répété indéfiniment}}}^{2}$$
$$\overline{0.2}^{10} = \overline{0.\underbrace{0011}_{\text{répété indéfiniment}}}^{2}$$
$$\overline{0.3}^{10} = \overline{0.0\underbrace{1001}_{\text{répété indéfiniment}}}^{2}$$

On doit maintenant trouver la représentation en `double` de ces trois nombres, qu'on déduit aisément des écritures précédentes, en décalant la virgule jusqu'à obtenir un 1 juste à gauche de la virgule, puis on a 52 chiffres significatifs.
Donc en `double`, les trois nombres sont en fait représentés par

$$\overline{0.1}^{10} : \overline{0.0001\underbrace{1001}_{\text{13 fois}}}^{2}$$
$$\overline{0.2}^{10} : \overline{0.001\underbrace{1001}_{\text{13 fois}}}^{2}$$
$$\overline{0.3}^{10} : \overline{0.01\underbrace{0011}_{\text{13 fois}}}^{2}$$

On voit immédiatement que le 56ème chiffre après la virgule de la représentation de 0.1 est 1, alors que le 56ème chiffre après la virgule de la différence entre les représentation de 0.3 et 0.2 est 0. Ce qui explique que le test `0.3 - 0.2 == 0.1` s'évalue à faux car il n'y a pas de chiffres significatifs au-delà du 56ème.

Correction :
Comment obtenir la représentation en base 2 d'un nombre ?
Première remarque : si on connaît la représentation en base 2 de deux nombres m et n (pas nécessairement entiers), notons-les bin(m) et bin(n), la représentation en base 2 de $m+n$ est la somme de bin(m) et bin(n). On peut se servir de ce point pour décomposer un nombre en somme de nombres qu'on sait représenter en base 2.
Deuxième remarque : si on connaît la représentation en base 2 d'un nombre n, notons-la bin(n), alors on connaît la représentation en base 2 de $2n$: on concatène 0 à droite de bin(n), et de $n/2$ (division euclidienne) : on supprime le dernier chiffre de bin(n).

Pour un entier n.
Un entier n est la somme de son chiffre des unités et de l'entier obtenu en remplaçant son chiffre des unités par 0, quelle que soit la base considérée, on en déduit que :

$$\text{bin}(n) = \underbrace{\text{bin}(n/2)\,0}_{\text{décalage de la virgule à droite}} + \underbrace{n\%2}_{\text{chiffre des unités}} .$$

Ce qui donne directement un algorithme pour trouver la décomposition en base 2 d'un entier :

Algorithme 1 Représentation en base 2 d'un entier

entrée : entier naturel n
sortie : représentation de n en base 2
1 : $a \leftarrow$ chaîne vide
2 : **si** $n > 1$ **alors**
3 : $\quad a \leftarrow$ Représentation en base 2 de l'entier $n/2$ (division euclidienne)
4 : **fin si**
5 : $b \leftarrow n\%2$ (reste de la division euclidienne)
6 : **renvoyer** concaténation de a et b

Pour un décimal x.
On commence par traiter la partir entière de x avec l'algorithme précédent. Il reste à traiter la partie décimale. On peut être tenté de multiplier par une puissance de 10 pour se ramener à un entier, mais c'est une très mauvaise idée, car bin$(10x)$ ne permet pas d'obtenir facilement bin(x). En revanche, on obtient bin(x) très facilement à partir de bin$(2x)$: il suffit de décaler la virgule d'un rang à gauche.
Donc si on multiplie k fois x par 2 pour un certain k et qu'on arrive à un entier, on est ramené au cas précédent : on trouve bin$(2^k x)$ et on décale la virgule de k rangs à gauche.
Évidemment, pour un décimal x, il n'existe pas nécessairement de k tel que $2^k x$ est un entier. Mais on peut en fait déduire les chiffres au fur et à mesure : au lieu de multiplier par 2 jusqu'à obtenir un entier, puis de récupérer tous les chiffres, on alterne le produit par 2 et la récupération d'un chiffre.

Algorithme 2 Représentation en base 2 d'un décimal strictement inférieur à 1

entrée : décimal positif $x < 1$, entier k
sortie : k premiers chiffres après la virgule de la représentation de x en base 2

```
1 : x ← 2x //remarque : x < 2
2 : a ← partie entière de x //remarque : a ∈ {0, 1}
3 : x ← x − a //remarque : x < 1
4 : b ← chaîne vide
5 : si k > 1 alors
6 :     b ← k − 1 premiers chiffres de la représentation en base 2 du décimal x
7 : fin si
8 : renvoyer concaténation de a et b
```

Faisons un exemple sur 0.3 :

$$0.3 \xrightarrow{\times 2} \underline{0}.6 \overset{-0}{\hookrightarrow} 0.6 \xrightarrow{\times 2} \underline{1}.2 \overset{-1}{\hookrightarrow} 0.2 \xrightarrow{\times 2} \underline{0}.4 \overset{-0}{\hookrightarrow} 0.4 \xrightarrow{\times 2} \underline{0}.8 \overset{-0}{\hookrightarrow} 0.8 \xrightarrow{\times 2} \underline{1}.6 \overset{-1}{\hookrightarrow} 0.6 \xrightarrow{\times 2} \underline{1}.2$$

Il est clair qu'à partir de là ça va être périodique, donc :

$$\text{bin}(0.3) = \overline{0.0\underbrace{1001}_{\text{répété indéfiniment}}}^{2}.$$