Représentation des nombres et récursivité

Algorithmique et structures de données, 2022-2023

P. Albuquerque (B410), P. Künzli et O. Malaspinas (A401), ISC, HEPIA

2022-11-02

En partie inspirés des supports de cours de P. Albuquerque

Types énumérés (1/2)

- Un **type énuméré**: ensemble de *variantes* (valeurs constantes).
- En C les variantes sont des entiers numérotés à partir de 0.

```
enum days {
    monday, tuesday, wednesday,
    thursday, friday, saturday, sunday
};
```

- On peut aussi donner des valeurs "custom"

```
enum days {
    monday = 2, tuesday = 8, wednesday = -2,
    thursday = 1, friday = 3, saturday = 12, sunday = 9
};
```

- S'utilise comme un type standard et un entier

```
enum days d = monday;
(d + 2) == monday + monday; // true
```

Types énumérés (2/2)

- Très utile dans les `switch ... case`

```
enum days d = monday;
switch (d) {
    case monday:
        // trucs
        break;
    case tuesday:
        printf("0 ou 1\n");
        break;
}
```

- Le compilateur vous prévient qu'il en manque!

Utilisation des types énumérés

Réusiner votre couverture de la reine avec des `enum`

A faire à la maison comme exercice!

Représentation des nombres (1/2)

- Le nombre 247.

Nombres décimaux: Les nombres en base 10

10^2	10^1	10^0
2	4	7

$$247 = 2 \cdot 10^2 + 4 \cdot 10^1 + 7 \cdot 10^0.$$

Représentation des nombres (2/2)

- Le nombre 11110111.

Nombres binaires: Les nombres en base 2

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	1	0	1	1	1

$$1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

Représentation des nombres (2/2)

- Le nombre 11110111.

Nombres binaires: Les nombres en base 2

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	1	0	1	1	1

$$1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

$$= 247.$$

Conversion de décimal à binaire (1/2)

Convertir 11 en binaire?

Conversion de décimal à binaire (1/2)

Convertir 11 en binaire?

- On décompose en puissances de 2 en partant de la plus grande possible

```
11 / 8 = 1, 11 % 8 = 3
3  / 4 = 0,  3 % 4 = 3
3  / 2 = 1,  3 % 2 = 1
1  / 1 = 1,  1 % 1 = 0
```

- On a donc

$$1011 \Rightarrow 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 11.$$

Conversion de décimal à binaire (2/2)

Convertir un nombre arbitraire en binaire: 247?

- Par groupe établir un algorithme.

Conversion de décimal à binaire (2/2)

Convertir un nombre arbitraire en binaire: 247?

- Par groupe établir un algorithme.

Algorithme

1. Initialisation

```
num = 247
tant que (2^N < num) {
    N += 1
}
```

Conversion de décimal à binaire (2/2)

Convertir un nombre arbitraire en binaire: 247?

- Par groupe établir un algorithme.

Algorithme

1. Initialisation

```
num = 247
tant que (2^N < num) {
    N += 1
}
```

2. Boucle

```
tant que (N >= 0) {
    bit = num / 2^N
    num = num % 2^N
    N -= 1
}
```

Les additions en binaire

Que donne l'addition `1101` avec `0110`?

- L'addition est la même que dans le système décimal

```
   1101          8+4+0+1 = 13
+  0110     +    0+4+2+0 =  6
-------     ----------------
  10011       16+0+0+2+1 = 19
```

- Les entiers sur un ordinateur ont une précision **fixée** (ici 4 bits).
- Que se passe-t-il donc ici?

Les additions en binaire

Que donne l'addition `1101` avec `0110`?

- L'addition est la même que dans le système décimal

```
   1101          8+4+0+1 = 13
+  0110     +    0+4+2+0 =  6
-------     ----------------
  10011       16+0+0+2+1 = 19
```

- Les entiers sur un ordinateur ont une précision **fixée** (ici 4 bits).
- Que se passe-t-il donc ici?

Dépassement de capacité: le nombre est "tronqué"

- `10011 (19) -> 0011 (3)`.
- On fait "le tour"."

Entier non-signés minimal/maximal

- Quel est l'entier non-signé maximal représentable avec 4 bit?

Entier non-signés minimal/maximal

- Quel est l'entier non-signé maximal représentable avec 4 bit?

$$(1111)_2 = 8 + 4 + 2 + 1 = 15$$

- Quel est l'entier non-signé minimal représentable avec 4 bit?

Entier non-signés minimal/maximal

- Quel est l'entier non-signé maximal représentable avec 4 bit?

$$(1111)_2 = 8 + 4 + 2 + 1 = 15$$

- Quel est l'entier non-signé minimal représentable avec 4 bit?

$$(0000)_2 = 0 + 0 + 0 + 0 = 0$$

- Quel est l'entier non-signé min/max représentable avec N bit?

Entier non-signés minimal/maximal

- Quel est l'entier non-signé maximal représentable avec 4 bit?

$$(1111)_2 = 8 + 4 + 2 + 1 = 15$$

- Quel est l'entier non-signé minimal représentable avec 4 bit?

$$(0000)_2 = 0 + 0 + 0 + 0 = 0$$

- Quel est l'entier non-signé min/max représentable avec N bit?

$$0 \text{ et } 2^N - 1.$$

- Donc `uint32_t`? maximal est?

Entier non-signés minimal/maximal

- Quel est l'entier non-signé maximal représentable avec 4 bit?

$$(1111)_2 = 8 + 4 + 2 + 1 = 15$$

- Quel est l'entier non-signé minimal représentable avec 4 bit?

$$(0000)_2 = 0 + 0 + 0 + 0 = 0$$

- Quel est l'entier non-signé min/max représentable avec N bit?

$$0 \text{ et } 2^N - 1.$$

- Donc `uint32_t`? maximal est?

$$2^{32} - 1 = 4'294'967'295$$

Les multiplications en binaire (1/2)

Que donne la multiplication de 1101 avec 0110?

- La mutliplication est la même que dans le système décimal

```
     1101                  13
*    0110     *             6
---------     --------------
     0000                  78
    11010
   110100
+ 0000000
---------     --------------
  1001110     64+0+0+8+4+2+0
```

Les multiplications en binaire (2/2)

Que fait la multiplication par 2?

Les multiplications en binaire (2/2)

Que fait la multiplication par 2?

- Décalage de un bit vers la gauche!

```
      0110
*     0010
---------
      0000
+    01100
---------
     01100
```

Les multiplications en binaire (2/2)

Que fait la multiplication par 2?

- Décalage de un bit vers la gauche!

```
      0110
*     0010
---------
      0000
+    01100
---------
     01100
```

Que fait la multiplication par 2^N?

Les multiplications en binaire (2/2)

Que fait la multiplication par 2?

- Décalage de un bit vers la gauche!

```
      0110
*     0010
---------
      0000
+    01100
---------
     01100
```

Que fait la multiplication par 2^N?

- Décalade de N bits vers la gauche!

Entiers signés (1/2)

Pas de nombres négatifs encore...

Comment faire?

Entiers signés (1/2)

Pas de nombres négatifs encore...

Comment faire?

Solution naïve:

- On ajoute un bit de signe (le bit de poids fort):

```
00000010: +2
10000010: -2
```

Problèmes?

Entiers signés (1/2)

Pas de nombres négatifs encore...

Comment faire?

Solution naïve:

- On ajoute un bit de signe (le bit de poids fort):

```
00000010: +2
10000010: -2
```

Problèmes?

- Il y a deux zéros (pas trop grave): 10000000 et 00000000
- Les additions différentes que pour les non-signés (très grave)

```
  00000010              2
+ 10000100          + -4
----------          ----
  10000110 = -6  !=    -2
```

Entiers signés (2/2)

Beaucoup mieux

- Complément à un:
  - on inverse tous les bits: `1001 => 0110`.

Encore un peu mieux

- Complément à deux:
  - on inverse tous les bits,
  - on ajoute 1 (on ignore les dépassements).

Entiers signés (2/2)

Beaucoup mieux

- Complément à un:
  - on inverse tous les bits: `1001 => 0110`.

Encore un peu mieux

- Complément à deux:
  - on inverse tous les bits,
  - on ajoute 1 (on ignore les dépassements).
- Comment écrit-on -4 en 8 bits?

Entiers signés (2/2)

Beaucoup mieux

- Complément à un:
  - on inverse tous les bits: `1001 => 0110`.

Encore un peu mieux

- Complément à deux:
  - on inverse tous les bits,
  - on ajoute 1 (on ignore les dépassements).
- Comment écrit-on `-4` en 8 bits?

```
 4 =  00000100

        --------
-4 =>   00000100


       11111011
     + 00000001
     ----------
       11111100
```

Le complément à 2 (1/2)

Questions:

- Comment on écrit `+0` et `-0`?
- Comment calcule-t-on `2 + (-4)`?
- Quel est le complément à 2 de `1000 0000`?

Le complément à 2 (1/2)

Questions:

- Comment on écrit +0 et -0?
- Comment calcule-t-on `2 + (-4)`?
- Quel est le complément à 2 de `1000 0000`?

Réponses

- Comment on écrit +0 et -0?

```
+0 = 00000000
-0 = 11111111 + 00000001 = 100000000 => 00000000
```

- Comment calcule-t-on `2 + (-4)`?

```
  00000010          2
+ 11111100        +  -4
----------        -----
  11111110           -2
```

- En effet

```
11111110 => 00000001 + 00000001 = 00000010 = 2.
```

Le complément à 2 (2/2)

Quels sont les entiers représentables en 8 bits?

Le complément à 2 (2/2)

Quels sont les entiers représentables en 8 bits?

```
01111111 =>  127
10000000 => -128 // par définition
```

Quels sont les entiers représentables sur N bits?

Le complément à 2 (2/2)

Quels sont les entiers représentables en 8 bits?

```
01111111 =>  127
10000000 => -128 // par définition
```

Quels sont les entiers représentables sur N bits?

$$-2^{N-1} ... 2^{N-1} - 1.$$

Remarque: dépassement de capacité en `C`

- Comportement indéfini!

Nombres à virgule (1/3)

Comment manipuler des nombres à virgule?

$$0.1 + 0.2 = 0.3.$$

Facile non?

Nombres à virgule (1/3)

Comment manipuler des nombres à virgule?

$$0.1 + 0.2 = 0.3.$$

Facile non?

Et ça?

```c
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
    float a = atof(argv[1]);
    float b = atof(argv[2]);
    printf("%.10f\n", (double)(a + b));
}
```

Nombres à virgule (1/3)

Comment manipuler des nombres à virgule?

$$0.1 + 0.2 = 0.3.$$

Facile non?

Et ça?

```c
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) {
    float a = atof(argv[1]);
    float b = atof(argv[2]);
    printf("%.10f\n", (double)(a + b));
}
```

Que se passe-t-il donc?

Nombres à virgule (2/3)

Nombres à virgule fixe

2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}	2^{-4}
1	0	1	0	.	0	1	0	1

Qu'est-ce ça donne en décimal?

Nombres à virgule (2/3)

Nombres à virgule fixe

2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}	2^{-4}
1	0	1	0	.	0	1	0	1

Qu'est-ce ça donne en décimal?

$$2^3 + 2^1 + \frac{1}{2^2} + \frac{1}{2^4} = 8 + 2 + 0.5 + 0.0625 = 10.5625.$$

Limites de cette représentation?

Nombres à virgule (2/3)

Nombres à virgule fixe

2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}	2^{-4}
1	0	1	0	.	0	1	0	1

Qu'est-ce ça donne en décimal?

$$2^3 + 2^1 + \frac{1}{2^2} + \frac{1}{2^4} = 8 + 2 + 0.5 + 0.0625 = 10.5625.$$

Limites de cette représentation?

- Tous les nombres > 16.
- Tous les nombres < 0.0625.
- Tous les nombres dont la décimale est pas un multiple de 0.0625.

Nombres à virgule (3/3)

Nombres à virgule fixe

- Nombres de $0 = 0000.0000$ à $15.9375 = 1111.1111$.
- Beaucoup de “trous” (au moins 0.0625) entre deux nombres.

Solution partielle?

Nombres à virgule (3/3)

Nombres à virgule fixe

- Nombres de $0 = 0000.0000$ à $15.9375 = 1111.1111$.
- Beaucoup de “trous” (au moins 0.0625) entre deux nombres.

Solution partielle?

- Rajouter des bits.
- Bouger la virgule.

Nombres à virgule flottante (1/2)

Notation scientifique

- Les nombres sont représentés en terme:
  - Une mantisse
  - Une base
  - Un exposant

$$\underbrace{22.1214}_{\text{nombre}} = \underbrace{221214}_{\text{mantisse}} \cdot \underbrace{10}_{\text{base}}^{\overbrace{-4}^{\text{exp.}}},$$

Nombres à virgule flottante (1/2)

Notation scientifique

- Les nombres sont représentés en terme:
  - Une mantisse
  - Une base
  - Un exposant

$$\underbrace{22.1214}_{\text{nombre}} = \underbrace{221214}_{\text{mantisse}} \cdot \underbrace{10}_{\text{base}} \overbrace{{}^{-4}}^{\text{exp.}},$$

On peut donc séparer la représentation en 2:

- La mantisse
- L'exposant

Nombres à virgule flottante (2/2)

Quel est l'avantage?

Nombres à virgule flottante (2/2)

Quel est l'avantage?

On peut représenter des nombres sur énormément d'ordres de grandeur avec un nombre de bits fixés.

Différence fondamentale avec la virgule fixe?

Nombres à virgule flottante (2/2)

Quel est l'avantage?

On peut représenter des nombres sur énormément d'ordres de grandeur avec un nombre de bits fixés.

Différence fondamentale avec la virgule fixe?

La précision des nombres est **variable**:

- On a uniquement un nombre de chiffres **significatifs**.

$$123456 \cdot 10^{23} + 123456 \cdot 10^{-23}.$$

Quel inconvénient y a-t-il?

Nombres à virgule flottante (2/2)

Quel est l'avantage?

On peut représenter des nombres sur énormément d'ordres de grandeur avec un nombre de bits fixés.

Différence fondamentale avec la virgule fixe?

La précision des nombres est **variable**:

- On a uniquement un nombre de chiffres **significatifs**.

$$123456 \cdot 10^{23} + 123456 \cdot 10^{-23}.$$

Quel inconvénient y a-t-il?

Ce mélange d'échelles entraîne un **perte de précision**.

Nombres à virgule flottante simple précision (1/4)

Aussi appelés *IEEE 754 single-precision binary floating point*.

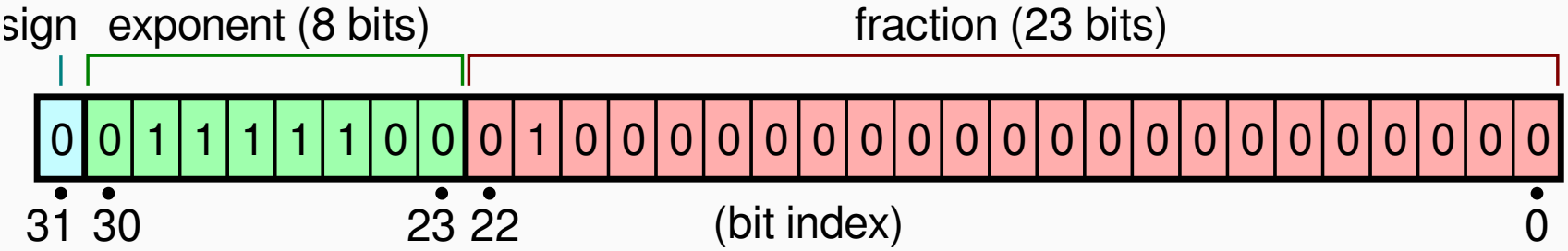

Figure 1: Nombres à virgule flottante 32 bits. Source: Wikipedia

Spécification

- 1 bit de signe,
- 8 bits d'exposant,
- 23 bits de mantisse.

$$(-1)^{b_{31}} \cdot 2^{(b_{30}b_{29}\ldots b_{23})_2 - 127} \cdot (1.b_{22}b_{21}\ldots b_0)_2,$$

Calculer la valeur décimale du nombre ci-dessus

Nombres à virgule flottante simple précision (2/4)

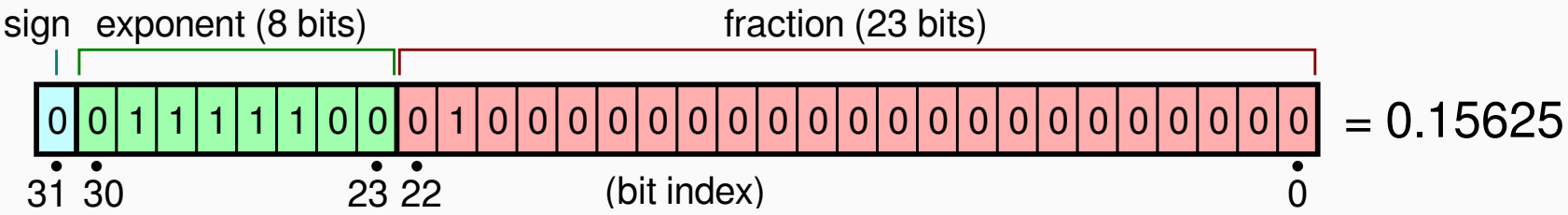

Figure 2: Un exercice de nombres à virgule flottante 32 bits. Source: Wikipedia

Nombres à virgule flottante simple précision (2/4)

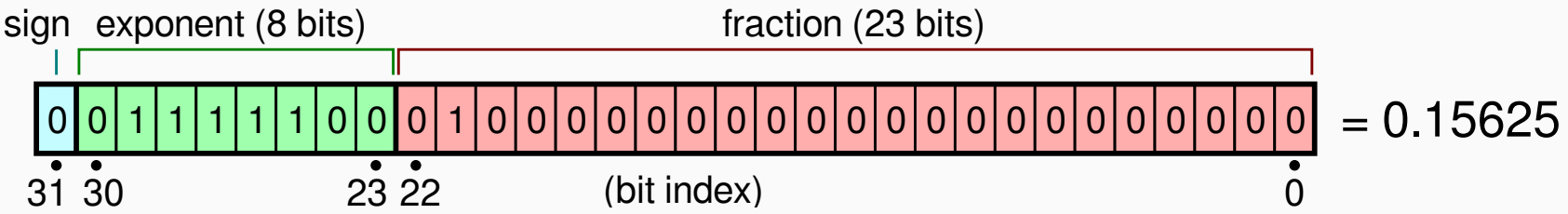

Figure 2: Un exercice de nombres à virgule flottante 32 bits. Source: Wikipedia

$$\text{exposant} = \sum_{i=0}^{7} b_{23+i} 2^i = 2^2 + 2^3 + 2^4 + 2^5 + 2^6 = 124 - 127, \quad (1)$$

$$\text{mantisse} = 1 + \sum_{i=1}^{23} b_{23-i} 2^{-i} = 1 + 2^{-2} = 1.25, \quad (2)$$

$$\Rightarrow (-1)^0 \cdot 2^{-3} \cdot 1.25 = 0.15625 \quad (3)$$

Nombres à virgule flottante simple précision (3/4)

Quel nombre ne peux pas être vraiment représenté?

Nombres à virgule flottante simple précision (3/4)

Quel nombre ne peux pas être vraiment représenté?

Zéro: exception pour l'exposant

- Si l'exposant est 00000000 (zéro)

$$(-1)^{\text{sign}} \cdot 2^{-126} \cdot 0.\text{mantisse},$$

- Sinon si l'exposant est 00000001 à 11111110

valeur normale,

- Sinon 11111111 donne NaN.

Nombres à virgule flottante simple précision (4/4)

Quels sont les plus petits/grands nombres positifs représentables?

Nombres à virgule flottante simple précision (4/4)

Quels sont les plus petits/grands nombres positifs représentables?

$$0\ 0\ldots0\ 0\ldots01 = 2^{-126}\cdot 2^{-23} = 1.4\ldots\cdot 10^{-45}, \tag{4}$$

$$0\ 1\ldots10\ 1\ldots1 = 2^{127}\cdot(2-2^{-23}) = 3.4\ldots\cdot 10^{38}. \tag{5}$$

Combien de chiffres significatifs en décimal?

Nombres à virgule flottante simple précision (4/4)

Quels sont les plus petits/grands nombres positifs représentables?

$$0\ 0\ldots0\ 0\ldots01 = 2^{-126}\cdot 2^{-23} = 1.4\ldots\cdot 10^{-45}, \tag{4}$$

$$0\ 1\ldots10\ 1\ldots1 = 2^{127}\cdot(2-2^{-23}) = 3.4\ldots\cdot 10^{38}. \tag{5}$$

Combien de chiffres significatifs en décimal?

- 24 bits $(23+1)$ sont utiles pour la mantisse, soit $2^{24}-1$:
  - La mantisse fait $\sim 2^{24}\sim 10^{7}$,
  - Ou encore $\sim \log_{10}(2^{24})\sim 7$.
- Environ **sept** chiffres significatifs.

Nombres à virgule flottante double précision (64bits)

Spécification

- 1 bit de signe,
- 11 bits d'exposant,
- 52 bits de mantisse.

Nombres à virgule flottante double précision (64bits)

Spécification

- 1 bit de signe,
- 11 bits d'exposant,
- 52 bits de mantisse.

Combien de chiffres significatifs?

- La mantisse fait $\sim 2^{53} \sim 10^{16}$,
- Ou encore $\sim \log_{10}(2^{53}) \sim 16$,
- Environ **seize** chiffres significatifs.

Plus petit/plus grand nombre représentable?

Nombres à virgule flottante double précision (64bits)

Spécification

- 1 bit de signe,
- 11 bits d'exposant,
- 52 bits de mantisse.

Combien de chiffres significatifs?

- La mantisse fait $\sim 2^{53} \sim 10^{16}$,
- Ou encore $\sim \log_{10}(2^{53}) \sim 16$,
- Environ **seize** chiffres significatifs.

Plus petit/plus grand nombre représentable?

- Plus petite mantisse et exposant: $\sim 2^{-52} \cdot 2^{-1022} \sim 4 \cdot 10^{-324}$,
- Plus grande mantisse et exposant: $\sim 2 \cdot 2^{1023} \sim \cdot 1.8 \cdot 10^{308}$.

Précision finie (1/3)

Erreur de représentation

- Les nombres réels ont potentiellement un **nombre infini** de décimales
  - $1/3 = 0.\overline{3}$,
  - $\pi = 3.1415926535....$
- Les nombres à virgule flottante peuvent en représenter qu'un **nombre fini**.
  - $1/3 \cong 0.33333$, erreur $0.00000\overline{3}$.
  - $\pi \cong 3.14159$, erreur $0.0000026535....$

On rencontre donc des **erreurs de représentation** ou **erreurs d'arrondi**.

Précision finie (1/3)

Erreur de représentation

- Les nombres réels ont potentiellement un **nombre infini** de décimales
  - $1/3 = 0.\overline{3}$,
  - $\pi = 3.1415926535....$
- Les nombres à virgule flottante peuvent en représenter qu'un **nombre fini**.
  - $1/3 \cong 0.33333$, erreur $0.00000\overline{3}$.
  - $\pi \cong 3.14159$, erreur $0.0000026535....$

On rencontre donc des **erreurs de représentation** ou **erreurs d'arrondi**.

Et quand on calcule?

- Avec deux chiffres significatifs

$$8.9 + (0.02 + 0.04) = 8.96 = 9.0, \tag{6}$$

$$(8.9 + 0.02) + 0.04 = 8.9 + 0.04 = 8.9. \tag{7}$$

Précision finie (1/3)

Erreur de représentation

- Les nombres réels ont potentiellement un **nombre infini** de décimales
  - $1/3 = 0.\overline{3}$,
  - $\pi = 3.1415926535....$
- Les nombres à virgule flottante peuvent en représenter qu'un **nombre fini**.
  - $1/3 \cong 0.33333$, erreur $0.00000\overline{3}$.
  - $\pi \cong 3.14159$, erreur $0.0000026535....$

On rencontre donc des **erreurs de représentation** ou **erreurs d'arrondi**.

Et quand on calcule?

- Avec deux chiffres significatifs

$$8.9 + (0.02 + 0.04) = 8.96 = 9.0, \tag{6}$$

$$(8.9 + 0.02) + 0.04 = 8.9 + 0.04 = 8.9. \tag{7}$$

Même pas associatif!

Précision finie (2/3)

Erreur de représentation virgule flottante

$$(1.2)_{10} = 1.\overline{0011} \cdot 2^0 \Rightarrow 0\ 01111111\ 00110011001100110011010.$$

Erreur d'arrondi dans les deux derniers bits et tout ceux qui viennent ensuite

$$\varepsilon_2 = (00000000000000000000011)_2.$$

Ou en décimal

$$\varepsilon_{10} = 4.76837158203125 \cdot 10^{-8}.$$

Précision finie (3/3)

Comment définir l'égalité de 2 nombres à virgule flottante?

Précision finie (3/3)

Comment définir l'égalité de 2 nombres à virgule flottante?

Ou en d'autres termes, pour quel $\varepsilon > 0$ (appelé `epsilon-machine`) on a

$$1 + \varepsilon = 1,$$

pour un nombre à virgule flottante?

Précision finie (3/3)

Comment définir l'égalité de 2 nombres à virgule flottante?

Ou en d'autres termes, pour quel $\varepsilon > 0$ (appelé `epsilon-machine`) on a

$$1 + \varepsilon = 1,$$

pour un nombre à virgule flottante?

Pour un `float` (32 bits) la différence est à

$$2^{-23} = 1.19 \cdot 10^{-7},$$

Soit la précision de la mantisse.

Comment le mesurer (par groupe)?

Précision finie (3/3)

Comment définir l'égalité de 2 nombres à virgule flottante?

Ou en d'autres termes, pour quel $\varepsilon > 0$ (appelé `epsilon-machine`) on a

$$1 + \varepsilon = 1,$$

pour un nombre à virgule flottante?

Pour un `float` (32 bits) la différence est à

$$2^{-23} = 1.19 \cdot 10^{-7},$$

Soit la précision de la mantisse.

Comment le mesurer (par groupe)?

```
float eps = 1.0;
while ((float)1.0 + (float)0.5 * eps != (float)1.0) {
    eps = (float)0.5 * eps;
}
printf("eps = %g\n", eps);
```

Erreurs d'arrondi

Et jusqu'ici on a encore pas fait d'arithmétique!

Multiplication avec deux chiffres significatifs, décimal

$$(1.1)_{10} \cdot (1.1)_{10} = (1.21)_{10} = (1.2)_{10}.$$

En continuant ce petit jeu:

$$\underbrace{1.1 \cdot 1.1 \cdots 1.1}_{\text{10 fois}} = 2.0.$$

Alors qu'en réalité

$$1.1^{10} = 2.5937...$$

Soit une erreur de près de 1/5e!

Erreurs d'arrondi

Et jusqu'ici on a encore pas fait d'arithmétique!

Multiplication avec deux chiffres significatifs, décimal

$$(1.1)_{10} \cdot (1.1)_{10} = (1.21)_{10} = (1.2)_{10}.$$

En continuant ce petit jeu:

$$\underbrace{1.1 \cdot 1.1 \cdots 1.1}_{\text{10 fois}} = 2.0.$$

Alors qu'en réalité

$$1.1^{10} = 2.5937...$$

Soit une erreur de près de 1/5e!

Le même phénomène se produit (à plus petite échelle) avec les `float` ou `double`.

La récursivité

Exemple de récursivité (1/2)

La factorielle

```
int factorial(int n) {
    if (n > 1) {
        return n * factorial(n - 1);
    } else {
        return 1;
    }
}
```

Exemple de récursivité (1/2)

La factorielle

```
int factorial(int n) {
    if (n > 1) {
        return n * factorial(n - 1);
    } else {
        return 1;
    }
}
```

Que se passe-t-il quand on fait `factorial(4)`?

Exemple de récursivité (1/2)

La factorielle

```
int factorial(int n) {
    if (n > 1) {
        return n * factorial(n - 1);
    } else {
        return 1;
    }
}
```

Que se passe-t-il quand on fait `factorial(4)`?

On empile les appels

			factorial(1)
		factorial(2)	factorial(2)
	factorial(3)	factorial(3)	factorial(3)
factorial(4)	factorial(4)	factorial(4)	factorial(4)

Exemple de récursivité (2/2)

La factorielle

```
int factorial(int n) {
    if (n > 1) {
        return n * factorial(n - 1);
    } else {
        return 1;
    }
}
```

Exemple de récursivité (2/2)

La factorielle

```
int factorial(int n) {
    if (n > 1) {
        return n * factorial(n - 1);
    } else {
        return 1;
    }
}
```

Que se passe-t-il quand on fait `factorial(4)`?

Exemple de récursivité (2/2)

La factorielle

```
int factorial(int n) {
    if (n > 1) {
        return n * factorial(n - 1);
    } else {
        return 1;
    }
}
```

Que se passe-t-il quand on fait `factorial(4)`?

On dépile les calculs

```
1
factorial(2)   2 * 1 = 2
factorial(3)   factorial(3)   3 * 2 = 6
factorial(4)   factorial(4)   factorial(4)   4 * 6 = 24
```

La récursivité (1/4)

Formellement

- Une condition de récursivité - qui *réduit* les cas successifs vers...
- Une condition d'arrêt - qui retourne un résultat

Pour la factorielle, qui est qui?

```
int factorial(int n) {
    if (n > 1) {
        return n * factorial(n - 1);
    } else {
        return 1;
    }
}
```

La récursivité (2/4)

Formellement

- Une condition de récursivité - qui *réduit* les cas successifs vers...
- Une condition d'arrêt - qui retourne un résultat

Pour la factorielle, qui est qui?

```
int factorial(int n) {
    if (n > 1) { // Condition de récursivité
        return n * factorial(n - 1);
    } else {     // Condition d'arrêt
        return 1;
    }
}
```

La récursivité (3/4)

Exercice: trouver l'ε-machine pour un `double`

La récursivité (3/4)

Exercice: trouver l'ε-machine pour un `double`

Rappelez-vous vous l'avez fait en style **impératif** plus tôt.

La récursivité (3/4)

Exercice: trouver l'ε-machine pour un `double`

Rappelez-vous vous l'avez fait en style **impératif** plus tôt.

```
double epsilon_machine(double eps) {
    if (1.0 + eps != 1.0) {
        return epsilon_machine(eps / 2.0);
    } else {
        return eps;
    }
}
```

La récursivité (4/4)

Exercice: que fait ce code récursif?

```
void recurse(int n) {
    printf("%d ", n % 2);
    if (n / 2 != 0) {
        recurse(n / 2);
    } else {
        printf("\n");
    }
}
recurse(13);
```

La récursivité (4/4)

Exercice: que fait ce code récursif?

```
void recurse(int n) {
    printf("%d ", n % 2);
    if (n / 2 != 0) {
        recurse(n / 2);
    } else {
        printf("\n");
    }
}
recurse(13);
```

```
recurse(13): n = 13, n % 2 = 1, n / 2 = 6,
    recurse(6): n = 6, n % 2 = 0, n / 2 = 3,
        recurse(3): n = 3, n % 2 = 1, n / 2 = 1,
            recurse(1): n = 1, n % 2 = 1, n / 2 = 0.

// affiche: 1 1 0 1
```

La récursivité (4/4)

Exercice: que fait ce code récursif?

```
void recurse(int n) {
    printf("%d ", n % 2);
    if (n / 2 != 0) {
        recurse(n / 2);
    } else {
        printf("\n");
    }
}
recurse(13);
```

```
recurse(13): n = 13, n % 2 = 1, n / 2 = 6,
    recurse(6): n = 6, n % 2 = 0, n / 2 = 3,
        recurse(3): n = 3, n % 2 = 1, n / 2 = 1,
            recurse(1): n = 1, n % 2 = 1, n / 2 = 0.

// affiche: 1 1 0 1
```

Affiche la représentation binaire d'un nombre!