# Chapitre 2

## Types et variables

# Plan

1. **Codage binaire**
2. Mémoire
3. Définition d'une variable
4. Nommer une variable
5. Typer une variable
6. Les types de base
7. Déclaration et affectation d'une variable
8. Codage des types élémentaires

# 2.1 Codage en binaire

## Manière de représenter les nombres en mémoire

Toute information est codée avec un système de numérotation binaire, donc en **base 2**
Chaque symbole correspond physiquement à une tension électrique

## Pour une base *B*, il y a *B* symboles différents

Décimal, **base 10** : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Binaire, **base 2** : 0, 1 un chiffre = un **bit** – ***bi**nary digi**t***

Octal, **base 8** : 0, 1, 2, 3, 4, 5, 6, 7

En C : `0, 01, 02, 03, 07, 010, 011, …` → précédé d'un « `0` »

Hexadécimal, **base 16** : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, **A, B, C, D, E, F**

En C : `0x0, 0x1, …, 0xF, 0x10, …` → précédé de « `0x` »

$A_{16}=10_{10}$, $B_{16}=11_{10}$, $C_{16}=12_{10}$, $D_{16}=13_{10}$, $E_{16}=14_{10}$, $F_{16}=15_{10}$

# 2.1 Nombre en base B vers nombre décimal

Valeur en *base 10* d'un nombre $(a_n\ a_{n-1}...a_1\ a_0)_B$ codé dans une base B :

$$nb_{10} = a_n \cdot B^n + a_{n-1} \cdot B^{n-1} + ... + a_1 \cdot B^1 + a_0 \cdot B^0$$

## Exemples

$(1234)_{10} = 1 \cdot \mathbf{10^3} + 2 \cdot \mathbf{10^2} + 3 \cdot \mathbf{10^1} + 4 \cdot \mathbf{10^0} = \mathbf{1234_{10}}$

$(11001)_2 = 1 \cdot \mathbf{2^4} + 1 \cdot \mathbf{2^3} + 0 \cdot \mathbf{2^2} + 0 \cdot \mathbf{2^1} + 1 \cdot \mathbf{2^0}$

$= 16 + 8 + 0 + 0 + 1 = \mathbf{25_{10}}$

$(1C04)_{16} = 1 \cdot \mathbf{16^3} + 12 \cdot \mathbf{16^2} + 0 \cdot \mathbf{16^1} + 4 \cdot \mathbf{16^0}$

$= 4096 + 12 \cdot 256 + 0 + 4 = 7172_{10}$

*avec A=10, B=11, C=12, D=13, E=14, F=15*

# Plan

1. Codage binaire
2. **Mémoire**
3. Définition d'une variable
4. Nommer une variable
5. Typer une variable
6. Les types de base
7. Déclaration et affectation d'une variable
8. Codage des types élémentaires

# 2.2 Qu'est-ce que la mémoire ?

La mémoire peut être vue comme un grand tableau dont les **cases** sont numérotées et dans lesquelles on peut mémoriser des informations

Le numéro des cases est `l'adresse "Adr"`

Les **valeurs** sont stockées dans les cases qui ont une taille de n bytes (1,2,4,8,…)

Certaines valeurs peuvent tenir sur :

- une case, comme **27** mémorisée ici à l'adresse `0x2`
- plusieurs cases, comme **"Hello"** aux adresses `0x4...0x9`

| Adr | Value | |
|---|---|---|
| 0x0 | | |
| 0x1 | | |
| 0x2 | 0x1B | |
| 0x3 | | |
| 0x4 | 0x48 | 'H' |
| 0x5 | 0x65 | 'e' |
| 0x6 | 0x6C | 'l' |
| 0x7 | 0x6C | 'l' |
| 0x8 | 0x6F | 'o' |
| 0x9 | 0x00 | '\0' |
| 0xA | | |
| 0xB | | |
| 0xC | 01001001 | |

=27

72 ?

# Plan

1. Codage binaire
2. Mémoire
3. **Définition d'une variable**
4. Nommer une variable
5. Typer une variable
6. Les types de base
7. Déclaration et affectation d'une variable
8. Codage des types élémentaires

# 2.3 Qu'est-ce qu'une variable ?

Une variable **nomme un endroit de la mémoire** où on mémorise une valeur d'un certain type

D'abord déclarer une variable en donnant un nom et un type, et optionnellement une valeur : initialisation

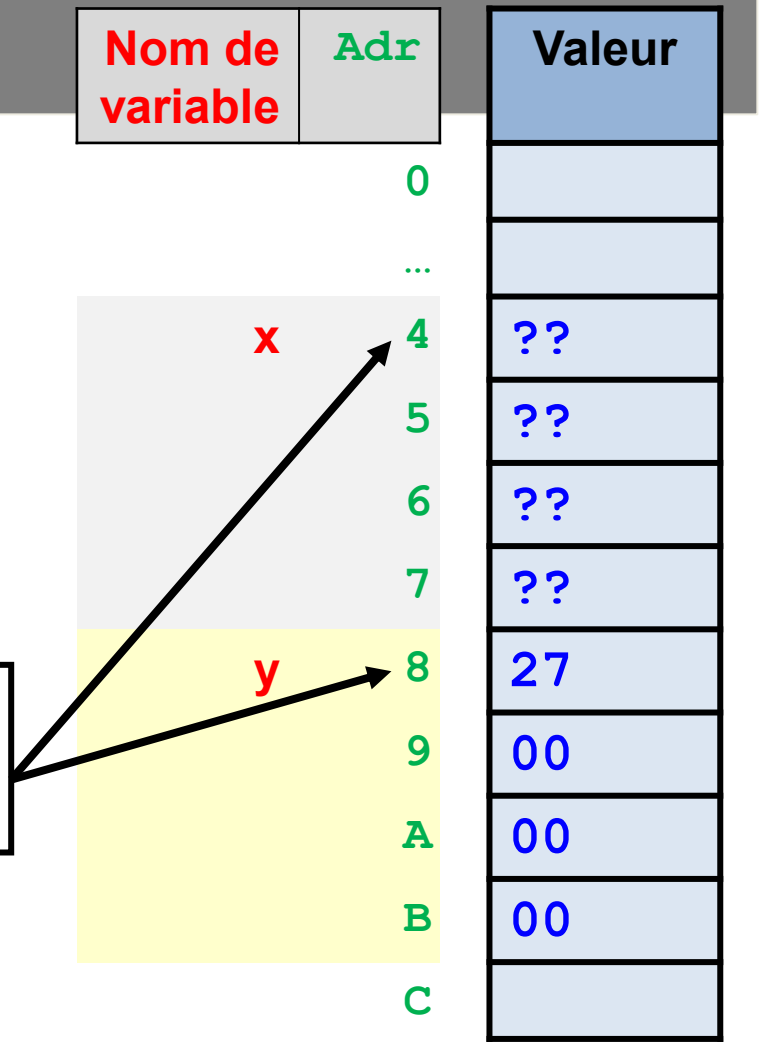
```
int x;
int y = 27;
```

Valeur initiale

Nom de la variable

Type pour un simple entier (`int`)

Le compilateur décide quelles places mémoire il utilise.

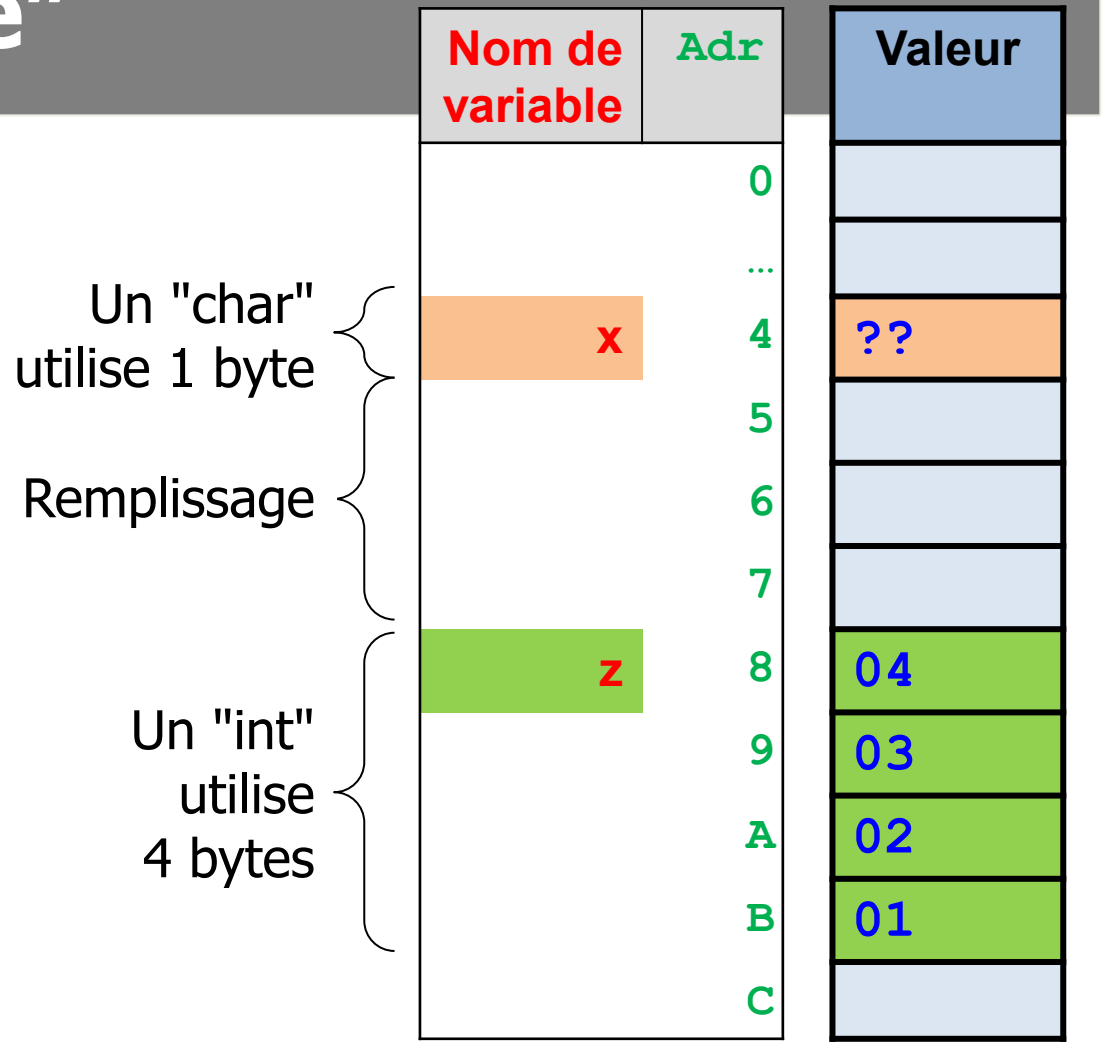


# 2.3 Variables "multi-byte"

Les différents types prennent plus ou moins de place mémoire. La plupart des ordinateurs mémorisent les valeurs sur des multiples de la taille d'un mot de base 8, 16, 32, 64 bits. Il peut y avoir du remplissage (*padding en anglais)*

```
char x;
int z = 0x01020304;
```

`0x` signifie que le nombre est en hexadécimal



# 2.3 Caractéristiques d'une variable

1. Son **identificateur** : le nom par lequel la donnée est désignée — fixe
2. Son **type** : la nature de la donnée associée à la variable et les opérations permises pour cette variable. Son type va également déterminer la façon dont la variable est stockée en mémoire et son domaine d'utilisation — fixe
3. Sa **valeur** à un instant donné. Elle peut varier au cours de l'exécution du programme — variable

# 2.3 Cycle de vie des variables

On distingue **4 opérations** sur les variables

La **déclaration** : déclarer un **nom** de variable en lui associant un **type**

L' **affectation** : **attribuer** une **valeur** à une **variable**

La **lecture** : **utiliser** la **valeur** associée à la variable

La **destruction** : **libérer** la **mémoire** utilisée

# Plan

1. Codage binaire
2. Mémoire
3. Définition d'une variable
4. **Nommer une variable**
5. Typer une variable
6. Les types de base
7. Déclaration et affectation d'une variable
8. Codage des types élémentaires

# 2.4 Noms de variables

Caractères autorisés par la norme

1. les **lettres** majuscules et minuscules (*case sensitive)*
2. les **chiffres** 0..9*
3. le caractère '_'

**⚠ Les noms ne peuvent pas commencer par un chiffre !**

Il est interdit d'utiliser un des **44 mots réservés** (***keywords***) du C (norme C11) pour les noms de variables.

# Mots réservés du langage C

Le langage, dans la norme C11, possède 44 mots-clés réservés **qui ne peuvent pas être utilisés comme identificateurs**

| | | | | |
|---|---|---|---|---|
| `_Alignas`(C11) | `_Alignof`(C11) | `_Atomic`(C11) | `_Bool`(C99) | `_Complex`(C99) |
| `_Generic`(C11) | `_Imaginary`(C99) | `_Noreturn`(C11) | `_Static_assert`(C11) | `_Thread_local`(C11) |
| `auto` | `break` | `case` | `char` | `const` |
| `continue` | `default` | `do` | `double` | `else` |
| `enum` | `extern` | `float` | `for` | `goto` |
| `if` | `inline`(C99) | `int` | `long` | `register` |
| `restrict`(C99) | `return` | `short` | `signed` | `sizeof` |
| `static` | `struct` | `switch` | `typedef` | `union` |
| `unsigned` | `void` | `volatile` | `while` | |

Référence: http://en.cppreference.com/w/c/keyword

# 2.4 Noms de variables

Les **noms** de variables doivent être **explicites**

```
quantity, unitPrice, xAxisSpeed,…
```

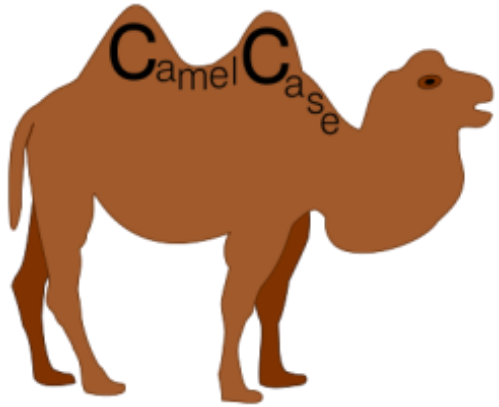
Il y a 2 manières usuelles d'écrire des noms de variables :

**camelCase** (lowerCamelCase)

Exemple : `exchangeRate`

**PascalCase** (UpperCamelCase)

Exemple : `ExchangeRate`




Il existe des conventions de codage (par exemple commentaires en anglais)

# Plan

1. Codage binaire
2. Mémoire
3. Définition d'une variable
4. Nommer une variable
5. **Typer une variable**
6. Les types de base
7. Déclaration et affectation d'une variable
8. Codage des types élémentaires

# 2.5 Typer une variable

Un programme doit gérer des informations stockées dans des variables.
En C, il faut "spécifier" la nature de chaque donnée en donnant son type.

Pour **chacun des types** sont associées les notions :

**1. taille de l'occupation de la mémoire**
→ Définit un intervalle de valeurs possibles pour la variable

**2. ensemble d'opérations** réalisables

# 2.5 Typer une variable

**Le C est un langage à typage faible, explicite et statique**

## Typage faible

La force du typage indique que les types de données correspondent aux données manipulées. Un langage est fortement typé si :

1. Les conversions implicites de types sont formellement interdites
2. Les erreurs de types sont détectées

## Typage explicite

Le programmeur doit indiquer les types qu'il utilise

## Typage statique

La vérification de type se fait **lors de la compilation**, et non lors de l'exécution comme pour le typage dynamique

# Plan

1. Codage binaire
2. Mémoire
3. Définition d'une variable
4. Nommer une variable
5. Typer une variable
6. **Les types de base**
7. Déclaration et affectation d'une variable
8. Codage des types élémentaires

# 2.6 Types de base en C

Le C propose une vaste variété de types permettant de manipuler des valeurs :

**Types de base**

- Nombres entiers
- Nombres à virgule flottante
- Valeur booléenne

**Types dérivés**

- Tableaux
- Structures
- Pointeurs
- Fonctions

# 2.6 Types de base en C

| | (anonymes) | (nommés) |
|---|---|---|
| **Nombres entiers signés** | | |
| Petite taille | `signed char` | `enum` |
| Taille moyenne | `[signed] short` | |
| **Grande taille** | `[signed] int` | |
| Ultra grande taille | `[signed] long` | |
| **Nombres entiers non-signés** | (anonymes) | |
| Petite taille | `unsigned char` | |
| Taille moyenne | `unsigned short` | |
| Grande taille | `unsigned int` | |
| Ultra grande taille | `unsigned long` | |

# 2.6 Types de base en C

## Nombres à virgule flottante

Simple : `float`

**Grande précision : `double`**

Ultra grande précision : `long double`

## Valeur booléenne

Dépend de la révision, voir slide "Type logique"

# 2.6 Types des nombres entiers non signés

| Type | Taille [bits] | Domaine de valeurs |
|---|---|---|
| `unsigned char` | 8 | 0 à 255 |
| `unsigned short` | 16 | 0 à 65'535 |
| `unsigned int` | 32 | 0 à 4'294'967'295 |
| `unsigned long` | 32 | 0 à 4'294'967'295 |

**Remarque :** la taille des types `int` et `long` dépend de la plateforme

# 2.6 Types des nombres entiers signés

| Type | Taille [bits] | Domaine de valeurs |
|---|---|---|
| `char` | 8 | -128 à +127 |
| `short` | 16 | -32’768 à +32’767 |
| `int` | 32 | -2’147'483’648 à +2'147'483’647 |
| `long` | 32 | -2’147'483’648 à +2'147'483’647 |

# 2.6 Types des nombres à virgule flottante (réels)

| Type | Taille [bits] | Domaine de valeurs |
|---|---|---|
| `float` | 32 | $3.4*10^{-38}$ à $3.4*10^{38}$ |
| `double` | 64 | $1.7*10^{-308}$ à $1.7*10^{308}$ |

# 2.6 Type caractères

| Type | Taille [bits] | Domaine de valeurs |
|---|---|---|
| `char` | 8 | -128 à +127 |
| `unsigned char` | 8 | 0 à 255 |

| Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr |
|---|---|---|---|---|---|---|---|---|---|
| 64 | 40 | 100 | @ | @ | 96 | 60 | 140 | ` | ` |
| 65 | 41 | 101 | A | A | 97 | 61 | 141 | a | a |
| 66 | 42 | 102 | B | B | 98 | 62 | 142 | b | b |
| 67 | 43 | 103 | C | C | 99 | 63 | 143 | c | c |
| 68 | 44 | 104 | D | D | 100 | 64 | 144 | d | d |
| 69 | 45 | 105 | E | E | 101 | 65 | 145 | e | e |
| 70 | 46 | 106 | F | F | 102 | 66 | 146 | f | f |

# 2.6 Type logique

| Type | Taille [bits] | vrai | faux | version |
|---|---|---|---|---|
| `int` | 32 | ≠0 | =0 | < C99 |
| `_Bool` | 8 | ≠0 | =0 | C99 |
| `bool` | 8 | `true` | `false` | ≤ C23 * `#include <stdbool.h>` |

*** Depuis C99, `true` et `false` sont des macros définies dans le header `<stdbool.h>`.**

**Depuis C23, `true` et `false` sont des keywords à part entière. Plus besoin de `<stdbool.h>`.**

# Plan

1. Codage binaire
2. Mémoire
3. Définition d'une variable
4. Nommer une variable
5. Typer une variable
6. Les types de base
7. **Déclaration et affectation d'une variable**
8. Codage des types élémentaires

# 2.7 Déclarer une variable

Déclarer une variable avant de l'utiliser (typage explicite) :

```
type nomDeLaVariable;
type nomDeLaVariable1, nomDeLaVariable2;
```

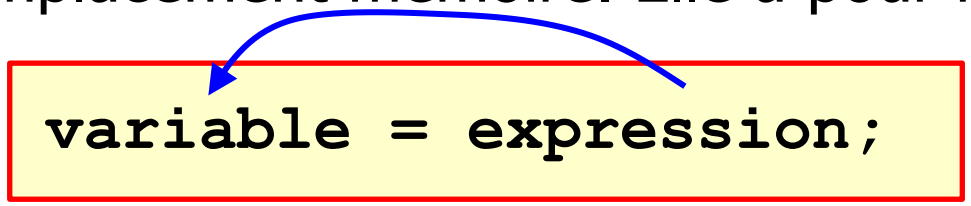
où `type` ∈ {`char, short, int, long, float, double,…, struct xx,…`}

**Exemples**

```
char   choice, tmp;
double fuelPrice, total;
long   barCode;
unsigned int test, i ,j, k;
```

# 2.7 Rôle et mécanisme de l'affectation

L'affectation est le mécanisme qui permet de placer une valeur dans un emplacement mémoire. Elle a pour forme :




**L'expression est soit**

une valeur littérale, par exemple `10` ou `'a'`

le résultat d'une instruction, par exemple `cos(5)` ou `a+10/c`

**Le symbole = signifie que la variable de gauche prend la valeur de l'expression de droite. Le sens est "←"**

# 2.7 Exemples d'affectation

Exemples pour des variables `n` et `p` déclarées de type entier :

```
n = 4; // n prend la valeur 4

p = 5 * n + 1; /* évalue l'expression mathématique et
                  mémorise le résultat (21) dans p */
```

**Attention à la signification** du symbole `=`

| | |
|---|---|
| `a = a + 1;` | a du **sens en informatique,** pas en mathématiques |
| ~~`a + 5 = 3;`~~ | a du **sens en mathématiques,** pas en informatique |
| `a = b ;` | **en informatique** est différent de `b=a;` |

# 2.7 Initialisation

Toute variable d'un programme **doit être déclarée**

```
type nomDeVariable;
```

et peut recevoir une valeur par **affectation**

```
nomDeVariable = valeur;
```

**Bonne habitude** : initialiser les variables à leur déclaration

**Exemple**

```
float  fuelPrice =1.63f,total=0.0f;
double x = 1.63;
long   barCode = 123456789 ;
int    test, i=0 ,j=0, k=0 ;
char   choice='Y', tmp='\0';
```

# 2.7 Tableaux

## Déclaration

```
float tab[3];
int numbers[100];
```

## Initialisation

```
float tab[3] = {1.2f, 3.4f, 5.6f};
int numbers[] = {5,6,7};
```

## Affectation

```
tab[0] = 3.5f;
tab[5] = 6.2f;
```

# 2.7 Portée des variables

```
#include <stdio.h>
#include <stdlib.h>

int a, b, z=0;                  // variables globales

int main(void)
{
    double x, y, z=1;           // variables locales
    {
        int u, v, z=2;          // variables locales
    }
    return 0;
}
```

Notions de **visibilité** (scope), **durée de vie,** masquage

# 2.7 Échanger 2 variables

Soit **a** et **b** deux variables de même type

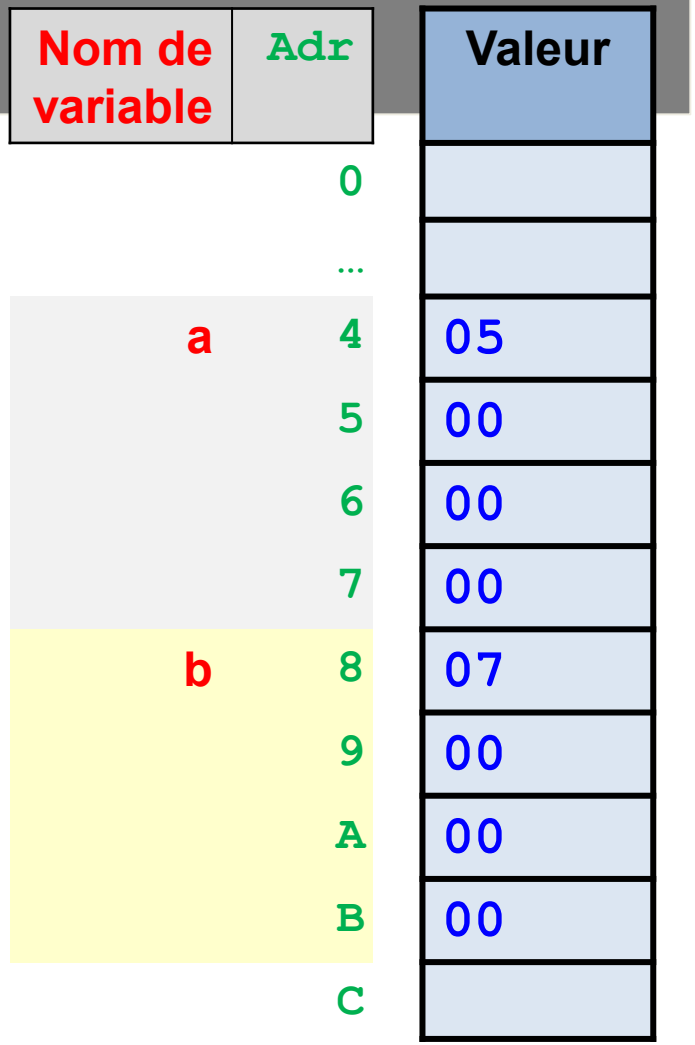
On souhaite échanger les valeurs de **a** et **b**

```
int a = 5;
int b = 7;

a = b;
b = a;
```

Que se passe-t-il ?

Comment faire ?



# 2.7 Déclaration de constantes symboliques

1) Le **mot réservé** `const` interdit toute modification de la valeur d'une variable après sa déclaration

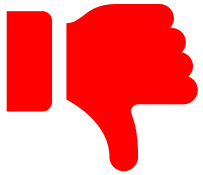
**Exemple**

```
const float PI = 3.1416;
PI = 3.14;              // erreur de compilation
```

→ toute constante doit être initialisée lors de sa déclaration

2) Par la **directive de précompilation** `#define`

```
#define PI 3.1416
```



**Attention :** éviter les constantes littérales, appelés "*magic numbers*"

# Plan

1. Codage binaire
2. Mémoire
3. Définition d'une variable
4. Nommer une variable
5. Typer une variable
6. Les types de base
7. Déclaration et affectation d'une variable
8. **Codage des types élémentaires**

# 2.8 Codage des types élémentaires

**Caractères** `char`

`'A', 'n'`

**Entiers** `int -> short, int, long`

`10, -100`

**Réels** `double -> float, double, long double`

`-1.0007, 887.82E-003`

# 2.8 Code ASCII pour les caractères

| Dec | Hx | Oct | Char | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 000 | NUL (null) | 32 | 20 | 040 |   | Space | 64 | 40 | 100 | @ | @ | 96 | 60 | 140 | ` | ` |
| 1 | 1 | 001 | SOH (start of heading) | 33 | 21 | 041 | ! | ! | 65 | 41 | 101 | A | A | 97 | 61 | 141 | a | a |
| 2 | 2 | 002 | STX (start of text) | 34 | 22 | 042 | " | " | 66 | 42 | 102 | B | B | 98 | 62 | 142 | b | b |
| 3 | 3 | 003 | ETX (end of text) | 35 | 23 | 043 | # | # | 67 | 43 | 103 | C | C | 99 | 63 | 143 | c | c |
| 4 | 4 | 004 | EOT (end of transmission) | 36 | 24 | 044 | $ | $ | 68 | 44 | 104 | D | D | 100 | 64 | 144 | d | d |
| 5 | 5 | 005 | ENQ (enquiry) | 37 | 25 | 045 | % | % | 69 | 45 | 105 | E | E | 101 | 65 | 145 | e | e |
| 6 | 6 | 006 | ACK (acknowledge) | 38 | 26 | 046 | & | & | 70 | 46 | 106 | F | F | 102 | 66 | 146 | f | f |
| 7 | 7 | 007 | BEL (bell) | 39 | 27 | 047 | ' | ' | 71 | 47 | 107 | G | G | 103 | 67 | 147 | g | g |
| 8 | 8 | 010 | BS (backspace) | 40 | 28 | 050 | ( | ( | 72 | 48 | 110 | H | H | 104 | 68 | 150 | h | h |
| 9 | 9 | 011 | TAB (horizontal tab) | 41 | 29 | 051 | ) | ) | 73 | 49 | 111 | I | I | 105 | 69 | 151 | i | i |
| 10 | A | 012 | LF (NL line feed, new line) | 42 | 2A | 052 | * | * | 74 | 4A | 112 | J | J | 106 | 6A | 152 | j | j |
| 11 | B | 013 | VT (vertical tab) | 43 | 2B | 053 | + | + | 75 | 4B | 113 | K | K | 107 | 6B | 153 | k | k |
| 12 | C | 014 | FF (NP form feed, new page) | 44 | 2C | 054 | , | , | 76 | 4C | 114 | L | L | 108 | 6C | 154 | l | l |
| 13 | D | 015 | CR (carriage return) | 45 | 2D | 055 | - | - | 77 | 4D | 115 | M | M | 109 | 6D | 155 | m | m |
| 14 | E | 016 | SO (shift out) | 46 | 2E | 056 | . | . | 78 | 4E | 116 | N | N | 110 | 6E | 156 | n | n |
| 15 | F | 017 | SI (shift in) | 47 | 2F | 057 | / | / | 79 | 4F | 117 | O | O | 111 | 6F | 157 | o | o |
| 16 | 10 | 020 | DLE (data link escape) | 48 | 30 | 060 | 0 | 0 | 80 | 50 | 120 | P | P | 112 | 70 | 160 | p | p |
| 17 | 11 | 021 | DC1 (device control 1) | 49 | 31 | 061 | 1 | 1 | 81 | 51 | 121 | Q | Q | 113 | 71 | 161 | q | q |
| 18 | 12 | 022 | DC2 (device control 2) | 50 | 32 | 062 | 2 | 2 | 82 | 52 | 122 | R | R | 114 | 72 | 162 | r | r |
| 19 | 13 | 023 | DC3 (device control 3) | 51 | 33 | 063 | 3 | 3 | 83 | 53 | 123 | S | S | 115 | 73 | 163 | s | s |
| 20 | 14 | 024 | DC4 (device control 4) | 52 | 34 | 064 | 4 | 4 | 84 | 54 | 124 | T | T | 116 | 74 | 164 | t | t |
| 21 | 15 | 025 | NAK (negative acknowledge) | 53 | 35 | 065 | 5 | 5 | 85 | 55 | 125 | U | U | 117 | 75 | 165 | u | u |
| 22 | 16 | 026 | SYN (synchronous idle) | 54 | 36 | 066 | 6 | 6 | 86 | 56 | 126 | V | V | 118 | 76 | 166 | v | v |
| 23 | 17 | 027 | ETB (end of trans. block) | 55 | 37 | 067 | 7 | 7 | 87 | 57 | 127 | W | W | 119 | 77 | 167 | w | w |
| 24 | 18 | 030 | CAN (cancel) | 56 | 38 | 070 | 8 | 8 | 88 | 58 | 130 | X | X | 120 | 78 | 170 | x | x |
| 25 | 19 | 031 | EM (end of medium) | 57 | 39 | 071 | 9 | 9 | 89 | 59 | 131 | Y | Y | 121 | 79 | 171 | y | y |
| 26 | 1A | 032 | SUB (substitute) | 58 | 3A | 072 | : | : | 90 | 5A | 132 | Z | Z | 122 | 7A | 172 | z | z |
| 27 | 1B | 033 | ESC (escape) | 59 | 3B | 073 | ; | ; | 91 | 5B | 133 | [ | [ | 123 | 7B | 173 | { | { |
| 28 | 1C | 034 | FS (file separator) | 60 | 3C | 074 | < | < | 92 | 5C | 134 | \ | \ | 124 | 7C | 174 | | | \| |
| 29 | 1D | 035 | GS (group separator) | 61 | 3D | 075 | = | = | 93 | 5D | 135 | ] | ] | 125 | 7D | 175 | } | } |
| 30 | 1E | 036 | RS (record separator) | 62 | 3E | 076 | > | > | 94 | 5E | 136 | ^ | ^ | 126 | 7E | 176 | ~ | ~ |
| 31 | 1F | 037 | US (unit separator) | 63 | 3F | 077 | ? | ? | 95 | 5F | 137 | _ | _ | 127 | 7F | 177 | &#127; | DEL |

# 2.8 Table ASCII pour les caractères

| DECIMAL VALUE | ➡ | 0 | 16 | 32 | 48 | 64 | 80 | 96 | 112 | 128 | 144 | 160 | 176 | 192 | 208 | 224 | 240 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ⬇ | HEXA DECIMAL VALUE | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 0 | 0 | BLANK (NULL) | ► | SP | 0 | @ | P | ‘ | p | Ç | É | á | ▒ | | | ∝ | ≡ |
| 1 | 1 | ☺ | ◄ | ! | 1 | A | Q | a | q | ü | æ | í | ▒ | | | β | ± |
| 2 | 2 | ☻ | ↕ | " | 2 | B | R | b | r | é | Æ | ó | ▓ | | | Γ | ≥ |
| 3 | 3 | ♥ | ‼ | # | 3 | C | S | c | s | â | ô | ú | | | | π | ≤ |
| 4 | 4 | ♦ | ¶ | $ | 4 | D | T | d | t | ä | ö | ñ | ┤ | | | Σ | ⌠ |
| 5 | 5 | ♣ | § | % | 5 | E | U | e | u | à | ò | Ñ | ╡ | | | σ | ⌡ |
| 6 | 6 | ♠ | ▬ | & | 6 | F | V | f | v | å | û | ª | ╢ | | | μ | ÷ |
| 7 | 7 | BEL | ↨ | ' | 7 | G | W | g | w | ç | ù | º | ╖ | | | τ | ≈ |
| 8 | 8 | BS | ↑ | ( | 8 | H | X | h | x | ê | ÿ | ¿ | ╕ | | | Φ | ° |
| 9 | 9 | HT | ↓ | ) | 9 | I | Y | i | y | ë | Ö | ⌐ | ╣ | | | Θ | • |
| 10 | A | LF | → | * | : | J | Z | j | z | è | Ü | ¬ | ║ | | | Ω | · |
| 11 | B | VT | ← | + | ; | K | [ | k | { | ï | ¢ | ½ | ╗ | | | δ | √ |
| 12 | C | FF | FS | , | < | L | \ | l | ¦ | î | £ | ¼ | ╝ | | | ∞ | n |
| 13 | D | CR | GS | – | = | M | ] | m | } | ì | ¥ | ¡ | ╜ | | | φ | ² |
| 14 | E | ♫ | RS | . | > | N | ^ | n | ~ | Ä | ₧ | « | ╛ | | | ∈ | ■ |
| 15 | F | ☼ | US | / | ? | O | _ | o | △ | Å | ƒ | » | ┐ | | | ∩ | BLANK |

# 2.8 Nombres entiers non signés

Un byte (1 octet) = ensemble de 8 bits

Un byte peut prendre $2^8 = 256$ valeurs différentes,
entre 0 (00000000) et 255 (11111111)

Exemple de byte non-signé :
11000100 = 128 + 64 + 4 = 196

# 2.8 Nombres entiers signés

**Byte signé** de -128 jusqu'à 127
Valeurs négatives : `11111111` = -1 à `10000000` = -128
Valeurs positives : `00000000` = 0 à `01111111` = 127

Passer de valeurs positives à négatives et réciproquement :

- Inverser tous les bits
- Ajouter 1

} Complément à $2^n$

**Exemple** : passer de 65 à -65 et réciproquement

| | |
|---|---|
| a) 65 | `01000001` |
| Inverser tous les bits | `10111110` |
| Ajouter 1 | `10111111` = -65 |
| b) -65 | `10111111` |
| Inverser tous les bits | `01000000` |
| Ajouter 1 | `01000001` = 65 |

C:\Windows\System32\calc.exe

# 2.8 Nombres à virgule flottante

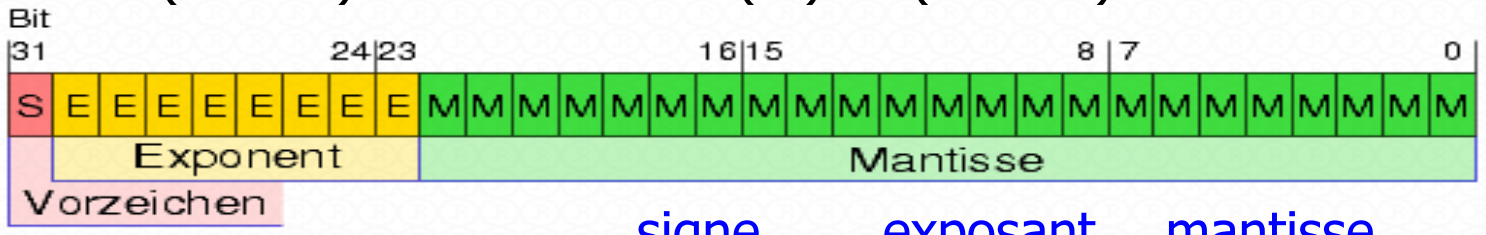
Norme: **norme informatique IEEE 754**

Les nombres à virgule flottante (`float`, `double`, `long double`)

- un **signe S**
- une **mantisse entière M**
- un **exposant E**

Simple précision (32 bits) : valeur = $(-1)^{S} * (1 + M) * 2^{(E-127)}$

Double précision (64 bits) : valeur = $(-1)^{S} * (1 + M) * 2^{(E-1023)}$




| | signe | exposant | mantisse | # chiffres significatifs |
|---|---|---|---|---|
| Simple précision (float, 32 bits) | 1 bit | 8 bits | 23 bits | ≈ 7 |
| Double précision (double, 64 bits) | 1 bit | 11 bits | 52 bits | ≈ 16 |

https://www.youtube.com/watch?v=ji3SfCIm8TU
https://www.h-schmidt.net/FloatConverter/IEEE754.html

# 2.8 Nombres à virgule flottante

Exemple : -3141.5 = -1.5339 x $2^{11}$

$-3141{,}5 \quad \text{alias} \quad ob\ -\ 1{,}100010001011 \times 2^{11}$

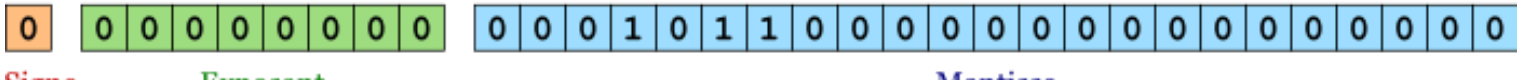
*Bit implicite* **1** *(nombre normalisé)*

$11 + 127 = 138 = ob10001010$


Exemple : 1.0101905 x $10^{-39}$ = 0.0859375 x $2^{-126}$

$\approx 1{,}0101905 \times 10^{-39} \quad \text{alias} \quad ob\ +\ 0{,}0001011 \times 2^{-126}$

*Bit implicite* **0** *(nombre dénormalisé)*

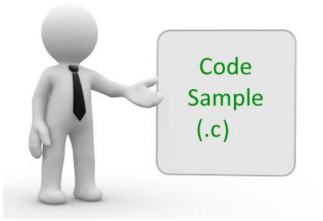


# Différents codages des valeurs 1 et -1

```
char   c=1; 00000001
int    i=1; 00000000 00000000 00000000 00000001
float  f=1.f; 00111111 10000000 00000000 00000000
double d=1.; 00111111 11110000 00000000 00000000 00000000 00000000 00000000

char   c=-1; 11111111
int    i=-1; 11111111 11111111 11111111 11111111
float  f=-1.f; 10111111 10000000 00000000 00000000
double d=-1.; 10111111 11110000 00000000 00000000 00000000 00000000 00000000

char c = '1'; 00110001
```



# Exercices

[Exercices du chapitre 02]