

# A propos de types en langage C

L. Mounier

UGA - M2 CCI - PL1

29 septembre 2021

# Plan

- 1 Les types dans les langages de programmation
- 2 Les types de base du langage C
- 3 Définitions de type

# Notion de type

## Un type définit :

- un domaine de valeurs  
ex : les entiers sur  $[1..10]$
- un ensemble d'opérations possibles sur ce domaine  
ex : addition, multiplication, etc.

## → On associe un type :

- aux identificateurs d'un programme
  - variables :  
x : un entier
  - procédures (actions et fonctions) :  
ecrire : une action (la donnée x : un entier)  
f : une fonction (un entier)  $\rightarrow$  un entier
- aux valeurs des expressions :  $x + 3 * y$ ,  $f(x) - 5$ , etc.

# Objectifs dy typage

## ■ programmes corrects

```
x : un entier ;  
y : un booléen ;  
x := x + y ;
```

## ■ programmes lisibles (maintenables)

```
x : un euros ;  
y : un dollars ;  
x := TAUX_CHANGE * y ;
```

## ■ programmes efficaces

(représentation en machine → temps d'exécution, mémoire)

```
x : un entier sur 1 .. 10 ;  
y : un entier sur 0 .. 50000 ;
```

## Différentes formes de typage

- Des langages **non typés**  
→ un seul type “universel” pour toutes les expressions  
**ex** : Assembleur, langages de script, etc.
- Des langages à **typage faible**  
→ autorise des conversions de type implicites  
**ex** : langage C, C++, etc.
- Des langages à **typage fort**  
→ peu/pas de conversions de types  
**ex** : Java, CAML, etc.
- Typage **statique** : types calculés à la compilation
- Typage **dynamique** : types calculés à l'exécution
- *Vérification* de type vs *Inférence* de types

# Conversions de Types

→ Possibilité de modifier le type d'une valeur

## Implicite :

- effectué (silencieusement) par le compilateur
- autorisée par le langage

```
int x ;  
x = 4.2 ; // valeur 4.2 arrondie à 4
```

## Explicite :

- notifié par le programmeur
- vérifié par le compilateur

```
float x ;          // x de type reel  
x = (float) 10 ; // valeur 10 "transformée en reel"
```

# Plan

- 1 Les types dans les langages de programmation
- 2 Les types de base du langage C
- 3 Définitions de type

# Types “entier”

**Domaine de valeurs** , défini par deux paramètres :

- taille de la représentation mémoire : char, short, long, etc.
- entiers naturels vs relatifs : unsigned

**Exemples :**

- char  $\rightsquigarrow$   $[-128, 127]$ , unsigned char  $\rightsquigarrow$   $[0, 255]$
- int  $\rightsquigarrow$   $[-2^{31}, 2^{31} - 1]$ , unsigned int  $\rightsquigarrow$   $[0, 2^{32} - 1]$

⇒ Domaines de valeurs **finis** ...

**Opérations :**

- arithmétiques : +, -, \*, / (division entière), % (modulo)
- comparaison : ==, !=, <=, <, >=, >
- **logiques !** : ||, &&, !

⇒ Débordements arithmétiques possibles ! char x ; x = 255 + 1

;



# Type “caractère”

## Domaine de valeurs :

- l'ensemble des caractères du clavier ...
- ...codés par des **entiers** (code Ascii)  
→ en C le type “caractère” est un type “entier” ! (char)

## Opérations :

- arithmétiques ... : 'a' + 1, 'a' \* 2, ...
- comparaison ... : 'a' == 'b', 'a' < 'b', ...
- logiques ... : 'a' && 'b', ...

## Quelques constantes utiles :

- '\n' : fin de ligne, '\t' : tabulation
- '\0' : caractère “nul”

# Types “réels”

**Deux représentations** : float et double

**Opérations** :

- arithmétiques : +, -, \*, / (division réelle)
- comparaisons : ==, <=, !=, etc
- librairie `math.h` (trigo, racine carrée, puissance, etc.)

**Domaines de valeurs finis** :

- pb de précision
- pb de débordement

# Plan

- 1 Les types dans les langages de programmation
- 2 Les types de base du langage C
- 3 Définitions de type**

# Opérateur typedef

Définition de types “utilisateur”

T : le type xxx  $\rightsquigarrow$  `typedef xxx T ;`

Exemple d'utilisation :

- spécialiser un type existant
- définir un type énuméré
- définir un produit de types
- etc.

# Spécialiser un Type

→ renommer un type existant

## Limitations

- aucune vérification par le compilateur

```
typedef int dollars ;  
typedef int euros ;  
dollars x = 3 ;  
euros y = 2 ;  
x = x + y ; // autorise
```

- impossible de restreindre le domaine de valeurs  
**ex** :  $T$  : le type entier sur  $[2, 12]$

# Types énumérés

T : le type [A, B, ...Z]  $\rightsquigarrow$  `typedef enum {A, B, ... Z} T ;`

## Exemples :

```
typedef enum {lundi, mardi, mercredi, jeudi, vendredi} Jour ;
typedef enum {rouge, bleu, vert, jaune} Couleur ;
```

## Remarques :

- Valeurs des types énumérés sont représentées par des entiers :  
lundi  $\rightsquigarrow$  0, mardi  $\rightsquigarrow$  1, vert  $\rightsquigarrow$  2, etc.
- toutes les opérations sur les entiers sont valides ...

```
Jour j ; Couleur c ;
j = lundi + 2 ;
c = vert * 3 ; // c vaut 6, plus une couleur ...
if (c < vert) ...
if (c < mercredi) ... // pas de verification
```

# Produit de Types

T : le type  $\langle A : T1, B : T2 \rangle \rightsquigarrow$  `typedef struct {T1 A, T2 B;} T ;`

## Exemples :

```
typedef struct { int num ; int denum ; } Fraction ;  
Fraction f1, f2 ;
```

## Accès aux champs de la structure :

```
f1.num = f1.num + f2.num ; f1.denum = 3 ;
```

## Limitations :

- affectation : `f1 = f2 ; // OK`
- comparaison : `f1 == f2, f1 != f2 // NON !`
- opérations arithmétiques : `f1 = f1 + f2 // NON !`
- Entrées-Sorties : `printf("...", f1) ; // NON !`

## Retour sur les Entrées/Sorties

**Librairie** `stdio.h`

**Deux fonctions principales :**

- lecture : `scanf`
- écriture : `printf`

**Spécifier le type des valeurs lues/écrites :**

- entier : `%d` (décimal), `%u` (non signé), `%x` (hexadécimal)
- caractère : `%c`
- réel : `%f`

**Exemples :**

```
printf("%c", 'a') ; // affiche 'a'  
printf("%d", 'a') ; // affiche 97  
printf("%c", 98) ; // affiche 'b'
```