

# 1- LANGUAGE C

---

VARIABLE, OPERATEUR, PRINTF - SCANF

Loïc Cuvillon

[l.cuvillon@unistra.fr](mailto:l.cuvillon@unistra.fr)

# Sommaire chapitre 1

- **Structure d'un programme C**
- Les variables et leur type
- Entrée/sortie : `printf` et `scanf`
- Opérateurs :
  - arithmétiques et conversion de type

# Structure d'un programme C

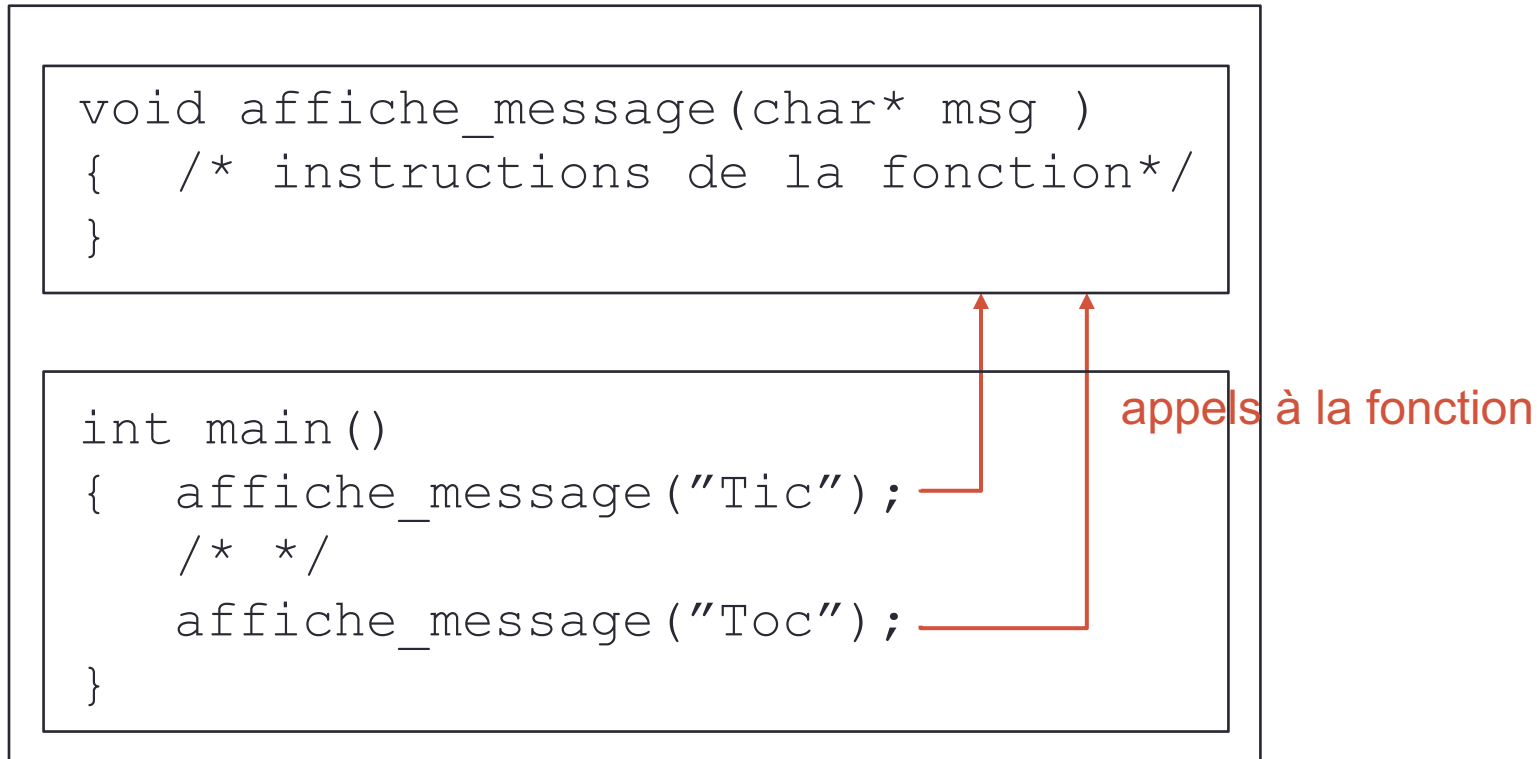
- Le C est une programmation impérative :  
une succession d'instruction à exécuter
- Les instructions sont regroupées dans des blocs d'instructions
  - un bloc est délimité par `{` et `}`

```
{    instruction 1;  
    instruction 2;  
}
```
  - un bloc peut contenir des sous-blocs

```
{    instruction 1;  
    {    instruction 2;  
    }  
}
```
  - une **fonction** : un bloc d'instructions identifié par un nom

# Structure d'un programme C

- un code C = des fonctions (blocs) qui s'appellent l'une l'autre



- La première fonction à exécuter porte un nom spécifique : **main()**
  - fonction **obligatoire**, libre placement dans le fichier

# Helloworld.c : un code simple

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

nom de la  
fonction :  
main

```
int main()
```

```
{
```

```
    printf("Hello World!\n");
```

```
    return 0;
```

```
}
```

2 instructions

–un bloc  
d'instructions  
avec un nom  
= une fonction

# Helloworld.c : une fonction

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

nom de la  
fonction :  
main

```
int main()
```

```
{
```

```
    printf("Hello World!\n");
```

```
    return 0;
```

```
}
```

printf :  
fonction d'affichage fournie  
par la bibliothèque stdio

# HelloWorld.c : analyse

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

```
int main()  
{  
    printf("Hello World!\n");  
    return 0;  
}
```

`#include <stdio.h>` ← inclusion de la bibliothèque standard entrée/sortie  
= déclaration de `printf()`

```
int main()  
{  
    printf("Hello World!\n");  
    return 0;  
}
```



`#include <stdio.h>` ← inclusion de la bibliothèque standard entrée/sortie  
= déclaration de `printf()`

fonction retourne un entier (int)



```
int main()  
{  
    printf("Hello World!\n");  
    return 0;  
}
```

`#include <stdio.h>` ← inclusion de la bibliothèque standard entrée/sortie  
= déclaration de `printf()`

fonction retourne un entier (int)



`int main()` ← la fonction n'a pas de paramètre en entrée

```
{  
    printf("Hello World!\n");  
    return 0;  
}
```

`#include <stdio.h>` ← inclusion de la bibliothèque standard entrée/sortie  
= déclaration de `printf()`

fonction retourne un entier (int)



`int main()` ← la fonction n'a pas de paramètre en entrée

{

`printf("Hello World!\n") ;` ← “ ; ” à la fin de chaque instruction

`return 0 ;`

}

`#include <stdio.h>` ← inclusion de la bibliothèque standard entrée/sortie  
= déclaration de `printf()`

fonction retourne un entier (int)



`int main()` ← la fonction n'a pas de paramètre en entrée

{

`printf("Hello World!\n") ;` ← “;” à la fin de chaque instruction

`return 0 ;`

}

← valeur retournée par la fonction  
(0 pour `main()` par convention)

`#include <stdio.h>` ← inclusion de la bibliothèque standard entrée/sortie  
= déclaration de `printf()`

fonction retourne un entier (int)



`int main()` ← la fonction n'a pas de paramètre en entrée

{

`printf("Hello World!\n");` ← “;” à la fin de chaque instruction

`return 0;`

} ©

← valeur retournée par la fonction  
(0 pour `main()` par convention)




“ { } ” délimite un bloc d'instruction  
ici, délimite le corps de la fonction


# Mise en forme du code

```
#include <stdio.h>

int main()
{
    int a=67;

    /*a integer(entier)*/
    if (a==0)
    {
        printf("%i",a);
    }
    return 0;
}
```

 1 tabulation

 1 tabulation (indentation)

- Par convention et lisibilité :
  - **une ligne = 1 instruction ;**
  - **indentation des instructions dans chaque (sous-) bloc**

Mais format libre :

- grâce au ';' indiquant fin d'1 instruction
- exple: code valide sur une seul ligne

```
#include <stdio.h>
int main(){int a=67;if (a==0){ printf ...
```

# Les commentaires

- Ignorés par le compilateur et délimités par `/* */`

```
/* ceci est un commentaire*/  
int main()  
{  
    /* suppression de l'affichage  
    printf("Hello World!\n"); */  
    return 0;  
}
```

- Tips: Utile pour désactiver l'exécution de lignes de codes

# Exercice 1

- Traiter l'exercice 1 en simultané avec l'enseignant



# Sommaire chapitre 1

- Structure d'un programme C
- **Les variables et leur type**
- Entrée/sortie : `printf` et `scanf`
- Opérateurs :
  - arithmétiques et conversion de type
  - &

# Une variable

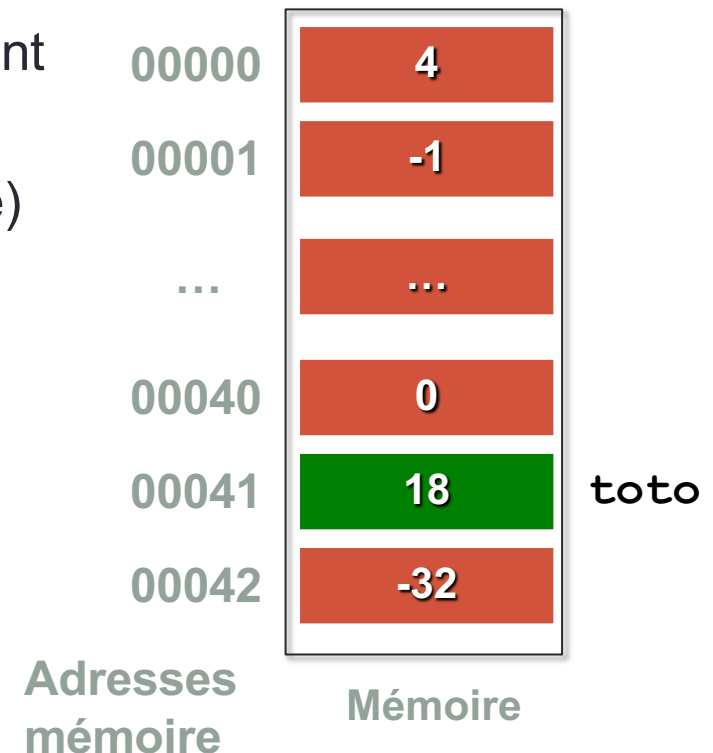
- Une variable :

un espace dans la mémoire pour stocker une valeur

- un nom/identifiant : pour la manipuler facilement
- une valeur
- (une adresse en mémoire : le lieu de stockage)

- Exemple :

la variable de nom `toto` vaut 18  
(mémorisée à l'adresse mémoire 41)



# Nom de variable

- Nom de variables (ou fonction) :
  - commence par une lettre ou \_
  - composé de lettres (sans accents), chiffres et \_
  - pas d'espace
  - sensible à la case : x et X sont 2 variables différentes
  - n'est pas un mot réservé de la syntaxe C (return, if, int, ...)
- Quiz : quel est le nom de variable correct ?

Bond\_007  
\_007

007\_Bond  
abc=bca

\$toto  
tété

return  
day\*month

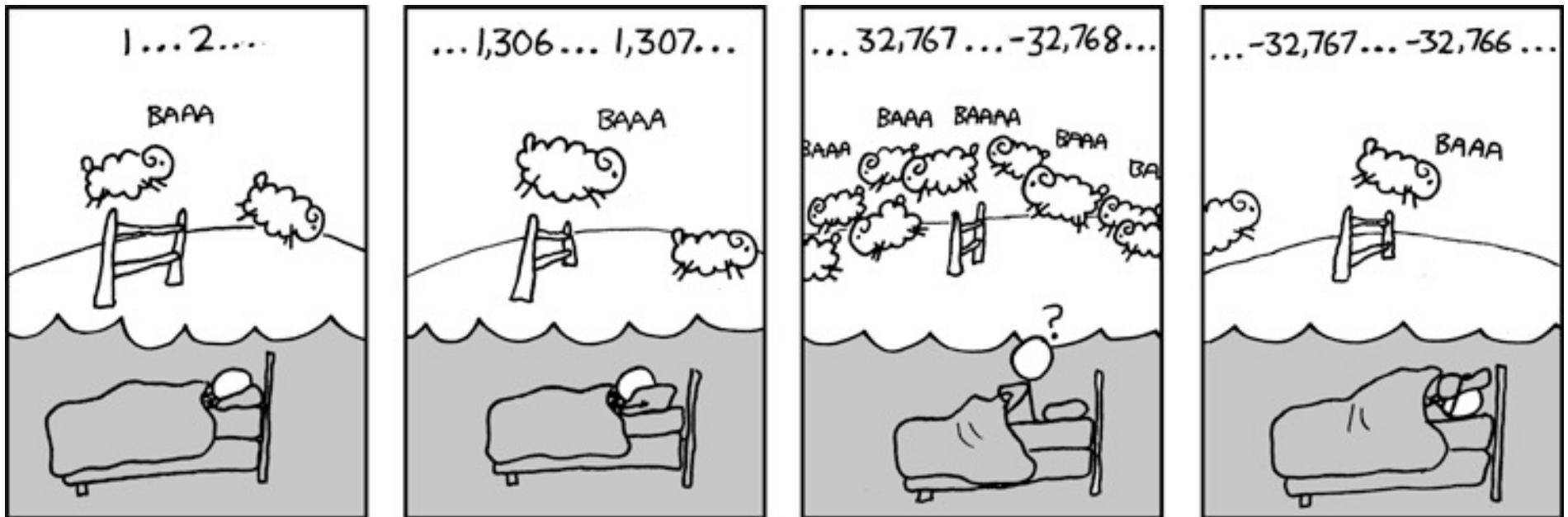
# Type pour les variables

- Le type d'une variable définit l'espace mémoire qui doit lui être réservé et donc les valeurs qu'elle peut prendre

Type	Octets réservés en mémoire (unix)	Valeurs stockables	
char	1	-128 à 127	Nombres entiers
short	2	-32 768 à 32 767	
int	4	-2 147 483 648 à 2 147 483 647	
long	4	-2 147 483 648 à 2 147 483 647	
float	4	$-3.4 \cdot 10^{38}$ à $3.4 \cdot 10^{38}$	Nombres décimaux (-1.69,...)
double	8	$-1.7 \cdot 10^{308}$ à $1.7 \cdot 10^{308}$	

Note : short  $\equiv$  short int      et      long  $\equiv$  long int

# Type pour les variables



Attention aux overflow/underflow (dépassement de capacité)

# Représentation des caractères

- le type `char` utilisé pour mémoriser le code d'un caractère
- un tableau ASCII de conversion entre: valeur numérique  $\leftrightarrow$  caractère

- Code ASCII : American Std Code for Information Interchange

Décimal	Caractère	Décimal	Caractère
00	NUL	65	A
10	Line Feed	66	B
32	Espace	...	...
46	. (point)	90	Z
48	0 (zéro)	97	a
49	1	98	b
...	...	...	...
57	9	122	z
61	=		

- Le caractère d'imprimerie 'A' est codé par l'entier 65 en mémoire
- Le caractère d'imprimerie '0' est codé par l'entier 48 en mémoire 😊

# Types

- Le C est un langage explicitement et statiquement typé (type définitif de la variable lors de la création)
- Ces types de base suffisent :
  - `int` pour les nombres entiers
  - `double` pour un nombre décimal, à virgule
  - `char` pour un caractère, ou un entier de 256 valeurs

# Les constantes

Type	exemples	signification
entier	11	entier 11 (base 10)
	0xB	(base hexadécimale)
décimal	0.01	double
	1e-2 ou 1.0e-1	double
caractère	'A'	le caractère A
chaîne de caractères (en argument de printf)	"hello world" "hello" "world"	chaîne littérale (idem.)
	"A"	la chaîne composée d'un caractère : A !



# Déclaration de variables

- Une variable doit être déclarée avant utilisation

- **Type nom;**

```
int monAge;
```

```
double sommedArgent;
```

```
unsigned short nombre_de_trains;
```

- ANSI C: Les déclaration de variables ne peuvent se faire qu'au début d'un bloc d'instructions ( délimité par { et })
  - déclaration à faire au tout début de chaque fonction !

# Déclaration de variable

- Initialiser la valeur d'une variable lors de sa déclaration avec une constante (**fortement conseillé**):

```
int main()  
{  
    int monAge = 0 ;  
    char uneLettre = 'z' ;  
    return 0 ;  
}
```

- Que vaut une variable non initialisée ?

```
int monAge;
```

- aléatoire : le contenu de la mémoire quand elle a été réquisitionnée pour la variable !

# Affectation

- Affecter, donner une (nouvelle) valeur à une variable:

```
monAge=31;  
sommedArgent = 1432.63;
```

- Déclaration multiple de variables de même type :

```
int monAge=31, maDate, monMois;  
char lettre1, lettre2;
```

- **le symbole = est l'opérateur d'affectation**

# Entrée/Sortie : printf

```
int nombreDeVies =0;  
  
printf("Il vous reste %d vies\n", nombreDeVies);
```

- ❑ afficher le contenu de **la chaîne de caractères** en remplaçant les
  - `%d`, `%e`, `%f`, `%c` : par la valeur des variables en paramètres supplémentaires avec le format précisé
  - `\n` : par un retour à la ligne (newline)



# Exercice 2

- Traiter l'exercice 2

# Entrée/Sortie : scanf

- Entrée: lire un nombre ou caractère(s) saisi au clavier

```
int age = 0;  
scanf("%d", &age);
```

- `scanf` - met en pause le programme
  - lit la saisie au clavier après un appui sur la touche [Entrée]
  - si le format correspond, met à jour la valeur de la variable en paramètre
- même format que `printf`, sauf pour la lecture d'un double : `%lf`  
Ne pas utiliser `\n` avec `scanf` pour éviter les problèmes !
- **nom de variable précédé de l'opérateur adresse &**  
(en paramètre : l'adresse mémoire de la variable, i.e. où écrire en mémoire la valeur)

## Exemple : somme de 2 entiers saisies au clavier

```
int main()
{
    int nbr1=0, nbr2=0;
    double resultat;

    printf("Entrer le nombre 1: ");
    scanf("%d", &nbr1);
    printf("Entrer le nombre 2: ");
    scanf("%d", &nbr2);

    resultat= nbr1 + nbr2;
    printf(" %d + %d = %f\n",nbr1, nbr2, resultat);

    return 0;
}
```



```
int main()  
{  
    int nbr1=0, nbr2=0;  
    double resultat;
```

-variables déclarées au début du bloc !  
-initialisées de préférence avec une valeur

```
    printf("Entrer le nombre 1: ");
```

```
    scanf("%d", &nbr1);
```

```
    printf("Entrer le
```

```
    scanf("%d", &nbr2);
```

-scanf() lit le format %d (un entier<sub>10</sub>) au clavier  
- valeur stockée -à l'adresse mémoire (&) de nbr1  
-dans nbr1

```
    resultat= nbr1 + nbr2;
```

```
    printf(" %d + %d = %f\n", nbr1, nbr2, resultat);
```

```
    return 0;
```

```
}
```

printf() affiche "la chaine de caractères" en remplaçant  
- %d par l'entier en paramètre dans l'ordre d'apparition , %f...  
- \n pour un retour à la ligne

# Exercice 3

- Traiter l'exercice 3