

# Bases du langage C



Julien Baste

IUT de Lille – Université de Lille

Séance 01

2022/2023

Le langage C est :

- développé par Dennis Ritchie et Kenneth Thompson dans les années 70
- utilisé pour écrire le noyaux UNIX
- norme ANSI (ou C89) définie par le livre de référence  
*The C Programming Language (2nd edition)* (Kernighan et Ritchie, 1988)
- plusieurs autres normes existent



- "Le langage C", B. Kernighan et D. Ritchie, ed. Dunod.
- "Méthodologie de la programmation en C", A. Braquelaire, ed. Dunod.
- "Programmer en langage C", C. Delannoy, ed. Eyrolles.

disponible dans la bibliothèque numérique de l'université :  
<http://univ.scholarvox.com.ressources-electroniques.univ-lille.fr/catalog/book/docid/88833881>

- Le **C** est l'ancêtre du **java** (et donc du **ijava**)  
vous retrouverez de nombreuses similitudes.
- La section moodle "**Exemples en C**" :
  - **Exemples commentés** des points techniques du C.
  - **Pensez à vous y référer régulièrement** en cas de doute.
- **Pour vous entraîner** aux bases du C, vous pouvez :
  - 1 regarder les fichiers de la section moodle "**Exemples en C**"
  - 2 les exécuter et tenter de comprendre comment ils fonctionnent
  - 3 modifier et jouer avec ces fichiers

- 1 On écrit notre programme dans un fichier source `Hello.java`
- 2 On compile avec la ligne de commande `javac Hello.java`
- 3 Cela crée le fichier exécutable `Hello.class`
- 4 On exécute le fichier hello : `java Hello`

## PROGRAMME SOURCE

```
#include <stdio.h>
int main(void) {
    printf("Hello word");
    return 0;
}
```

COMPILATION

gcc

## CODE MACHINE

```
00111011 ...
10001101 ...
00001101 ...
```

EXECUTION



- 1 On écrit notre programme dans un fichier source `main.c`
- 2 On compile avec la ligne de commande `gcc -Wall -o hello main.c`
- 3 Cela crée le fichier exécutable `hello` (grâce à l'option `-o`)
- 4 On exécute le fichier `hello` : `./hello` (si je suis dans le bon dossier)

Voir l'exemple "hello\_world"

Est-ce qu'en C je peux écrire <qqch> ?

- Le réponse sera très souvent : **OUI vous pouvez.**
- En pratique ça sera aussi : **Mais vous n'avez pas envie !**

**CONSEIL** : Utilisez un **code propre**, **lisible** et **que vous comprenez**.

**CONSEIL 2** : **Optimiser ne sert à rien** tant que l'on ne maîtrise pas totalement le langage et l'algorithmique.

**CONSEIL 3** : Pour le moment, préoccupez vous de faire du **code qui fonctionne**, et **que vous sachiez pourquoi**.

Pour **éviter de s'embrouiller** et de faire face aux **innombrables bugs** que peut générer le C, je ne **présenterais que certaines parties** du C et vous encouragerais à n'utiliser que celle-là.

Un programme C doit être constitué de la manière suivante :

- Les **appels aux bibliothèques standards**

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

- L'**appel à la fonction main** qui est la première fonction exécutée

```
int main (void) { }
```

Et **dans la fonction main** (comme dans toutes les fonctions), et **dans cet ordre** :

- **déclaration des variables** ex : `int i;`
- **initialisation des variables** ex : `i = 3;`
- le **corps de la fonction**
- retour de la **valeur de retour** ex : `return 0;`

Attention : Toutes les instructions dans une fonction doivent finir par un “;”

voir les exemples sur moodle



Chaque variable à un type. En R2.04, nous utiliserons uniquement les types `int`, `float` et `char`.

Toute **variable** utilisée dans un programme **doit être déclarée** en début de fonction.

Pour déclarer une variable, on écrit son type suivi de son nom.

`ex : int variable;` permet de déclarer la variable `variable` de type `int`.

Une affectation se fait grâce au signe “=” suivi de la valeur à affecter.

`ex : variable = 4;` affecte 4 à `variable`.

NOTE : déclarations et affectations sont des instructions, elles finissent donc par “;”

Notation usuelle : l’instruction `variable++;` remplace `variable = variable + 1;`

Voir l’exemple “variables”

Comme dans beaucoup de langage on retrouve les structures de contrôle suivantes :

- `if`
- `do...while`
- `while`
- `for`

Si l'on veut comparer l'entier  $a$  et l'entier  $b$  on peut écrire :

- $a == b$  :  $a$  égale  $b$
- $a != b$  :  $a$  différent de  $b$
- $a < b$  :  $a$  strictement plus petit que  $b$
- $a \leq b$  :  $a$  plus petit ou égal à  $b$
- $a > b$  :  $a$  strictement plus grand que  $b$
- $a \geq b$  :  $a$  plus grand ou égal à  $b$

La syntaxe est la même qu'en java, voir l'exemple "if\_et\_boucles"

À l'initialisation d'un tableau, il faut :

- déclarer le **type des éléments** du tableau
- déclarer la **taille du tableau** (cette taille **doit** être une **constante**).

ex : `int tableau[6]` crée un tableau d'entier de taille 6.

Dans le cas des tableaux, il est souvent préférable de **déclarer** le tableau **puis** de **l'initialiser**.  
Vous pouvez néanmoins l'initialiser lors de la déclaration :

```
int tableau[6] = {4, 2, 6, 8, 3, 1}.
```

Les cases d'un tableau de taille 6 sont numérotées de 0 à 5

On y accède par `tableau[0]`, `tableau[1]`, `tableau[2]`, `tableau[3]`, `tableau[4]`, et `tableau[5]`.

Voir l'exemple "tableaux"

Une chaîne de caractères est un tableau de char finissant par le caractère `\0`.

NOTE : `\0` n'a pas besoin d'être dans la dernière case du tableau.

```
char chaine[] = "j'aime les fleurs"
```

Vous pouvez modifier la chaîne en modifiant les caractères du tableau

```
chaine[4] = 'd';
```

Votre chaîne est désormais "j'aide les fleurs".

Voir exemple "chaine\_de\_caracteres"

Comme en ijava, vous pouvez partitionner votre code grace à des fonctions. Cette fonctionnalité est capitale pour la lisibilité de votre code.

En C, vous devez :

- Déclarer toutes vos fonctions au début de votre fichier (et après les “#include”) (Sauf la fonction `main` qui a un statut particulier).
- Écrire votre fonction

Dans une fonction il y a plusieurs points important :

- Vous devez annoncer le type des arguments.
- Vous devez annoncer le type de la valeur de retour.
- Vous devez retourner une valeur de ce type.

Voir l'exemple “fonctions”

```
#include <stdio.h>

int plus_un (int n);

int main (void)
{
    printf("plus_un(2) = %d\n", plus_un(2));
    return 0;
}

int plus_un (int n)
{
    return n+1;
}
```

