

# COURS 1

PREMIERS  
PROGRAMMES  
EN C

# PARTIE (I)

COMMENT ÉCRIRE  
ET EXÉCUTER  
UN PROGRAMME

# QU'EST-CE QUE C ? POURQUOI C ENSEIGNÉ ?



un vieux langage de programmation (1972!)  
proche de la machine (on parle de langage **bas niveau**)




syntaxe old-school

= coder dans d'autres  
langages paraît ⊕ facile



langage très performant  
encore très utilisé pour les systèmes embarqués  
(par ex)

qui permet de comprendre les rouages  
de n'importe quel autre langage   
par ex, python est écrit en C

# PREMIER PROGRAMME EN C

En C, on **code** dans un fichier texte (d'extension .c)  
qu'on va **compiler** (= traduire le code en langage machine),  
ce qui fait un **exécutable** (qu'on pourra lancer à notre guise).

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

} le reste est du charabia obligatoire...

```
int main() {
```

```
// On "code" véritablement dans la zone qui commence ici...
```

```
printf("Allô le monde !\nEst-ce que tout va bien ?\nAllô le monde !\n");
```

```
// ... et qui va jusque là.
```

```
return EXIT_SUCCESS;
```

```
}
```

} ... qu'on comprendra plus tard.

instructions  
à écrire  
ici →

En C, chaque instruction se termine par ; (point virgule)

⚠ À part les espaces, chaque caractère compte! (minuscule ≠ majuscule)

# AFFICHER UN MESSAGE

Les instructions sont lues séquentiellement par la machine du haut vers le bas.

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

int main() {

    printf("Ce message apparaît en premier.\n");
    printf("Puis celui-là !\n");
    printf("Et encore après : celui-là ! (bon vous avez compris...) \n");

    return EXIT_SUCCESS;
}
```

Par des questions de lisibilité, on décale les instructions vers la droite: il s'agit d'une indentation

Première instruction `printf`

Syntaxe: `printf (" Votre message ");`

→ permet d'afficher `Votre message` sur l'écran

Note `\n` désigne un retour à la ligne.

# COMMENTAIRES

commentaires = annotations du code non lues par la machine, réservées pour le codeur (ou toute personne qui lira le code)

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

int main() {

    // Ce message ne sera pas lu par la machine.
    //

    printf("Ce message va être affiché.\n");

    /* Tout ce qui est entre les deux * ne sera pas lu :
    C'est un commentaire multi-lignes !

    printf("Par contre ce message NE va PAS être affiché.\n");
    */

    return EXIT_SUCCESS;
}
```

## 2 manières de commenter en C

- Commentaire sur une ligne  
// Commentaire ici

- Commentaire multi-lignes  
/\* Commentaire sur  
plusieurs  
lignes \*/  
à ne pas oublier

# COMMENT COMPILER ?

En TP, on utilisera **caseine**, une plateforme hébergée par l'Université de Grenoble qui permet de compiler et exécuter automatiquement vos programmes.

enregistrer      compiler + exécuter  
                    évaluer (caseine fait passer des tests à vos prog)

The screenshot shows the Caseine web IDE interface. The main editor displays a C program named 'premierprogramme.c'. The code includes `<stdio.h>` and `<stdlib.h>`, and contains a `main` function that prints 'Ceci est mon premier programme sur caseine.' and returns `EXIT_SUCCESS`. The interface has icons for saving, running, and evaluating. On the right, a sidebar shows the 'Note proposée : 5 / 5' and a 'Summary of tests' section indicating '4 tests run / 4 t'.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 /* Ecrivez votre programme ci-dessous.
6 Cliquez sur Save, puis testez en cliquant sur Run (la fusée).
7 Une fois que vous êtes satisfait, cliquez sur Evaluate (la case avec un Tick) */
8
9
10
11 int main(){
12
13     // Instruction(s) à écrire ci-dessous :
14     printf("Ceci est mon premier programme sur caseine.\n");
15
16     return EXIT_SUCCESS;
17
18 }
```

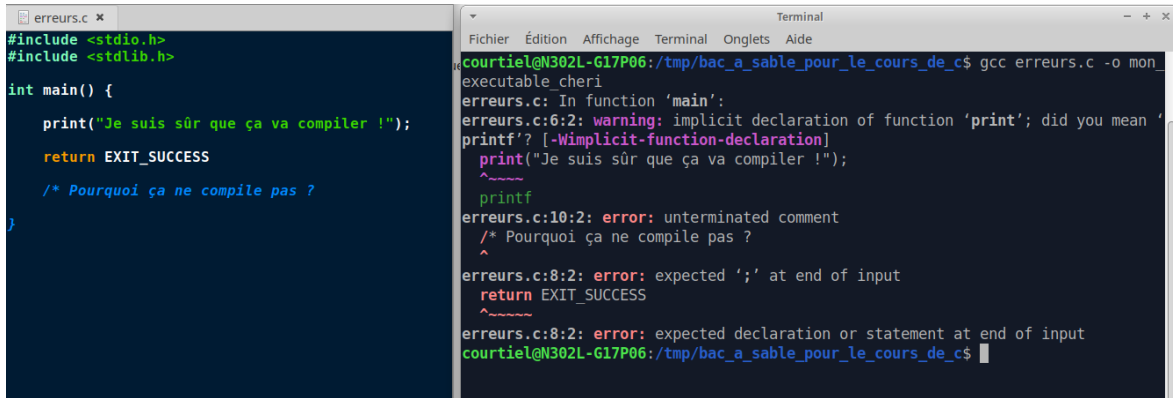
Sur linux

- Ouvrir un terminal dans le répertoire du prog
- Compiler : Écrire `gcc nom-du-prog.c -o nom_executable`  
Vous décidez du nom
- Exécuter : Écrire `./nom_executable`

Pour windows ou mac  
des compilateurs C  
existent...  
(minGW/gcc)

# ERREURS DE COMPILATION

Le programme doit être à 100% valide pour être compilé, sinon le compilateur renvoie des erreurs :



```
erreurs.c x
#include <stdio.h>
#include <stdlib.h>

int main() {
    print("Je suis sûr que ça va compiler !");
    return EXIT_SUCCESS
    /* Pourquoi ça ne compile pas ?
}

courtiel@N302L-G17P06:/tmp/bac_a_sable_pour_le_cours_de_cs$ gcc erreurs.c -o mon_
executable_cheri
erreurs.c: In function 'main':
erreurs.c:6:2: warning: implicit declaration of function 'print'; did you mean '
printf'? [-Wimplicit-function-declaration]
    print("Je suis sûr que ça va compiler !");
    ^~~~~
    printf
erreurs.c:10:2: error: unterminated comment
    /* Pourquoi ça ne compile pas ?
    ^
erreurs.c:8:2: error: expected ';' at end of input
    return EXIT_SUCCESS
    ^~~~~
erreurs.c:8:2: error: expected declaration or statement at end of input
courtiel@N302L-G17P06:/tmp/bac_a_sable_pour_le_cours_de_cs$
```

Il est important de lire les messages d'erreur !

Conseil : Commencez par corriger la 1<sup>ère</sup> erreur puis recompilez (jusqu'à que ça marche)



# EXPLICATION DE LA STRUCTURE CHARABIA D'UN FICHIER C

à lire à la maison pour celles et ceux qui se sentent à l'aise

inclusion de bibliothèques (= des collections de programmes de "base" qui sont déjà implantés sur l'ordinateur)

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

int main() {

    // On "code" véritablement dans la zone qui commence ici...

    printf("Allô le monde !\nEst-ce que tout va bien ?\nAllô le monde !\n");

    // ... et qui va jusque là.

    return EXIT_SUCCESS;
}
```

fonction principale

//  
fonction qui sera appelée en premier lors de l'exécution du programme.

- "int" signifie qu'elle renvoie un entier (le code de sortie du programme)

- "()" dit que la fonction n'a pas de paramètres  
- les accolades délimitent le champ de la fonction main

gère les entrées/sorties (obligatoire pour printf)

gère plein de fonctions basiques (ici EXIT\_SUCCESS)

un code de sortie :  
EXIT\_SUCCESS c'est pareil que 0 !  
0 indique que le programme s'est terminé sans erreur

# PARTIE III

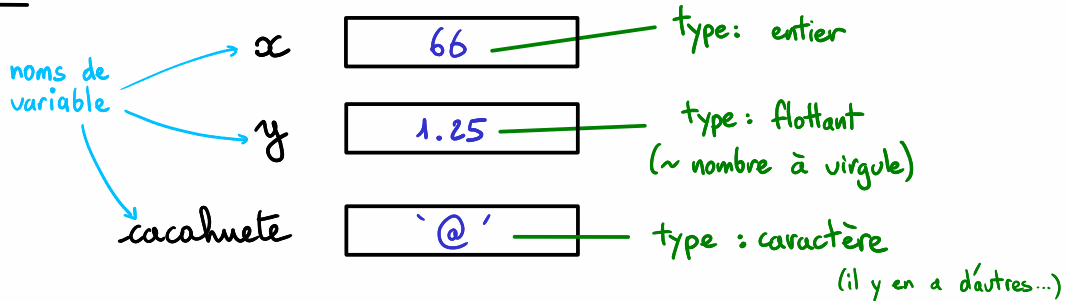
## LES VARIABLES

# VARIABLES

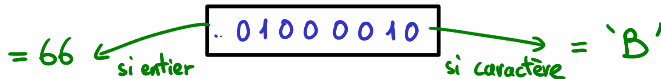
La programmation n'est que manipulation de variables

**variable** = case mémoire dans l'ordinateur contenant une valeur ayant un certain **type**

## Exemples



Dans les faits une case mémoire n'est constituée que de 0 et de 1



L'interprétation que l'ordinateur en fera dépendra du type.

# DÉCLARER UNE VARIABLE

Pour pouvoir utiliser une variable, il faut au préalable la **déclarer**, autrement dit lui donner un nom, un type.  
Une case mémoire lui sera donc allouée.

↑  
C'EST OBLIGATOIRE!

Description	Syntaxe	Effet en mémoire
Déclarer une variable entière de nom <u>"mon_entier"</u> <i>vous décidez</i>	<code>int mon_entier;</code>	<i>Cet espace est maintenant réservé pour cette variable</i> <code>mon_entier</code> <span style="border: 1px solid black; padding: 2px 10px;">???</span>
Déclarer une variable flottante	<code>float toto;</code>	<code>toto</code> <span style="border: 1px solid black; padding: 2px 10px;">???</span>
Déclarer une variable de type caractère	<code>char c;</code>	<code>c</code> <span style="border: 1px solid black; padding: 2px 10px;">???</span>

 **ATTENTION** On ne peut pas prédire la valeur d'une variable qui n'a été que déclarée. C'est souvent 0, mais pas toujours! Beaucoup de bugs viennent de là.

# MODIFIER LES VARIABLES

On peut donner une valeur / changer la valeur d'une variable grâce au signe d'affectation "="

## AFFECTATION

Syntaxe:

ma-variable = [expression] ;

Effets:

- 1 - Calcule la valeur de l'expression de droite
- 2 - Met cette valeur dans la variable "ma-variable"  
L'ancienne valeur de la variable est effacée.

Ex de programme:

```
1. int x;
2. x = 12;
3. x = 10 - 2 * 7;
4. x = x + 3;
```

peut aussi s'écrire "int x = 12;"

EN  
M  
E  
M  
O  
I  
R  
E

x

<del>12</del>	<del>-4</del>	-1
---------------	---------------	----

La variable x contient successivement 12 puis -4 et à la fin -1.



**ACHTUNG**

En C, le simple égal n'a rien à voir avec l'égalité. Par ex,  $x = 3;$  ne signifie pas "x est égal à 3" mais "la variable x reçoit 3".  
Conseil: Remplacez dans votre tête "=" par " $\leftarrow$ ". Ex:  $x \leftarrow 3;$

# INTERACTIONS AVEC L'UTILISATEUR

**utilisateur** = toute personne qui exécute votre programme

## Afficher le contenu d'une variable

On peut injecter la valeur d'une variable **entière** à l'intérieur d'un message qu'on veut afficher en écrivant par exemple

```
printf("La valeur de ma variable est %d.\n", mon-entier);
```

Si on veut afficher un **flottant**, on remplace `%d` par `%f`

Si on veut afficher un **caractère**, on remplace `%d` par `%c`

Syntaxe plus générale: `printf(" .. %u .. %u ... %u .... ", var1, var2, var3...);`  
dépend du type de `var1`, `var2`, `var3`...

## Récupérer une valeur saisie par l'utilisateur

Pour demander à l'utilisateur de rentrer un entier et de le stocker dans une variable `mon-entier`, on écrira

```
scanf("%d", &mon-entier);
```

⚠ Ne pas oublier `&`!

Pour un flottant, on utilisera `%f` et pour un caractère `%c`

# UN EXEMPLE RÉCAPITULATIF

Programme qui demande une lettre (= un caractère) puis le répète :

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

int main() {

    char lettre;

    printf("Dites-moi une lettre et je vais vous répéter cette lettre.\n");
    printf("Saisissez une lettre : ");

    scanf("%c",&lettre);

    printf("Vous avez rentré la lettre '%c', comme la première lettre dans le mot '%coute'.\n",lettre,lettre);

    return EXIT_SUCCESS;
}
```

La lettre entrée apparaît 2 fois dans le message

int x;

# OPÉRATIONS SUR LES ENTIERS

Les classiques - à ressortir en TP

## ADDITION

$$x = 11 + 2;$$

$$x \quad \boxed{13}$$

## SOUSTRACTION

$$x = 11 - 2;$$

$$x \quad \boxed{9}$$

## MULTIPLICATION

$$x = 11 * 2;$$

$$x \quad \boxed{22}$$

## DIVISION ENTIÈRE

$$x = 11 / 2;$$

$$x \quad \boxed{5}$$

## RESTE DIVISION EUCLIDIENNE

$$x = 11 \% 2;$$

$$x \quad \boxed{1}$$

c'est ce nombre

$$\begin{array}{r} 11 \quad 2 \\ -10 \\ \hline 1 \quad 5 \end{array}$$

⚠ Ca tronque à l'entier inférieur

Si on veut la division des flottants  
il faudra écrire  $11 / (\text{float}) 2$   
ou  $11 / 2.0$

## ASTUCES

→ Un nombre nb est pair si  $nb \% 2 == 0$   
→ Le chiffre des unités d'un nombre nb est  
donné par  $nb \% 10$

## INCRÉMENTATION

= augmentation d'une variable de 1

$$x++;$$

$$x \quad \boxed{11} \rightsquigarrow x \quad \boxed{12}$$

(pareil que  $x = x + 1;$ )

## DÉCRÉMENTATION

= réduction d'une variable de 1

$$x--;$$

$$x \quad \boxed{11} \rightsquigarrow x \quad \boxed{10}$$

(pareil que  $x = x - 1;$ )

(Il existe encore plein d'autres opérations : décalage de bits, ou bit à bit, ...  
mais leur usage est plus anecdotique et demande de maîtriser l'écriture binaire)



# PARTIE (III)

## STRUCTURES CONDITIONNELLES

# STRUCTURES CONDITIONNELLES

Comment exécuter des instructions différentes selon l'état d'une/plusieurs variables

Si ... alors

SYNTAXE

```
if ( condition ) {  
    Instructions qui seront  
    exécutées si la  
    condition est vraie  
}
```

EXEMPLE

```
int x;  
scanf("%d", &x);  
if ( x % 2 == 0 ) {  
    printf("Le nombre saisi est pair.\n");  
}
```

Exemples de conditions

- $a == b$  (a est égal à b)
- $a != b$  (a différent de b)
- $a > b$  (a strictement supérieur à b)
- $a >= b$  (a supérieur ou égal à b)

Si ... alors ... sinon

SYNTAXE

```
if ( condition ) {  
    Instructions qui seront  
    exécutées si la  
    condition est vraie  
}  
else {  
    Instructions qui seront  
    exécutées si la  
    condition est fausse  
}
```

EXEMPLE

```
float y;  
scanf("%f", &y);  
if ( y != 0 ) {  
    printf("L'inverse de %f est %f.", y, 1/y);  
}  
else {  
    printf("pas d'inverse pour 0!");  
}
```

# STRUCTURES CONDITIONNELLES

Si une "condition" n'est pas "binaire", on peut enchaîner les "else if"

Exemple : L'utilisateur saisit un flottant, le programme lui indique s'il est strictement positif, strictement négatif ou nul.

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

int main() {

    float z;

    printf("Saisissez un nombre à virgules : ");
    scanf("%f",&z);

    if ( z > 0 ) {
        printf("Ce nombre est strictement positif.\n");
    }
    else if ( z < 0 ) {
        printf("Ce nombre est strictement négatif.\n");
    }
    else {
        // Si les conditions "z>0" et "z<0" sont fausses,
        // alors z vaut forcément 0.
        printf("Ce nombre est 0.\n");
    }

    return EXIT_SUCCESS;
}
```

# CONNECTEURS LOGIQUES

Parfois on veut combiner plusieurs conditions ensemble.  
On utilise alors des **connecteurs logiques**.

NOM	SYNTAXE	SIGNIFICATION	EXEMPLE
ET	condition 1 && condition 2	Pour que ce soit vrai, il faut et suffit que <u>condition 1</u> <u>et</u> condition 2 soient vraies	if ( 10 <= x && x <= 20 ) { printf("x est entre 10 et 20 (inclus).\n"); }
OU	condition 1    condition 2	Pour que ce soit vrai, il faut et suffit que <u>condition 1</u> <u>ou</u> condition 2 soient vraies (ou les deux)	if ( n == 42    n == 69 ) { printf("Ton nombre est rigolo.\n"); }
NON	! condition	Pour que ce soit vrai, il faut et suffit que condition soit fausse.	if ( !( 10 <= x && x <= 20 ) ) { printf("x n'est pas entre 10 et 20.\n"); }
OU EXCLUSIF (XOR)	condition 1 ^ condition 2	Pour que ce soit vrai, il faut et suffit que <u>condition 1</u> <u>ou</u> condition 2 soient vraies, mais pas les deux.	if ( a < 0 ^ b < 0 ) { printf("Le produit de a par b est négatif.\n"); }