



INFORMATIQUE

Sequence 1 : Premiers pas

Cours 1.1

v2025-09-05

IUT d'Annecy, 9 rue de l'Arc en Ciel, 74940 Annecy

HELLO WORLD

Sommaire

1	Introduction à l'informatique	2
1.1	Objectifs généraux	2
1.2	Systèmes de comptage	2
1.3	Représentation de l'information	3
1.4	Algorithmes	4
2	Hello, World : Premier programme	5
2.1	Explications	5
2.2	Compilation et exécution	5
3	Types et Variables	6
3.1	Les types de données en C	6
3.2	Déclaration et utilisation	6
3.3	Afficher une variable dans un texte	6
4	Quelques opérateurs mathématiques	8
4.1	Opérateurs	8
4.2	Fonctions usuelles de <code>math.h</code>	8
5	Les conditions en C	9
5.1	Structure générale : if / else	9
5.2	Opérateurs de comparaison	9
5.3	Opérateurs logiques	9
5.4	Un exemple simple	10

1 Introduction à l'informatique

1.1 Objectifs généraux

L'objectif de ce module est, avant tout, d'apporter des compétences de réflexion pour la résolution de problèmes. En effet, écrire un code informatique, c'est d'abord écrire un algorithme qui répond à une problématique.

Durant ce module, nous verrons le langage C car il est suffisamment bas niveau pour illustrer le fonctionnement des systèmes informatiques. C'est, par ailleurs, le langage utilisé pour développer les systèmes embarqués.

1.2 Systèmes de comptage

1.2.1 Le système unaire

Quand on compte sur nos doigts, on utilise un système **unaire** (base 1) :

- Chaque doigt levé représente une unité.
- Exemple : 3 doigts levés = le nombre 3.

1.2.2 Le système binaire

En binaire, chaque chiffre (appelé **bit**) peut prendre deux valeurs :

- 0 → inactif, éteint, faux
- 1 → actif, allumé, vrai



Exemple des ampoules

Si on a **3 ampoules** qui peuvent être allumées (1) ou éteintes (0) :

- 000 = 0
- 001 = 1
- 010 = 2
- ...
- 111 = 7

Avec 3 ampoules, on peut donc compter de 0 à 7.



A retenir : Le binaire en informatique

Tous les systèmes informatiques utilisent le binaire pour encoder toutes les informations. Les données sont stockées grâce à des **transistors** qui se comportent comme de minuscules interrupteurs.

1.2.3 L'octet

Historiquement, les données étaient envoyées par groupes de 8 bits sur les bus des premiers processeurs. Ces groupes ont été appelés des **octet** (**oct** pour "huit" et **et** pour "petit"). C'est depuis devenu une unité de mesure de la taille des données (octet, kilo-octet, Mega-octet, etc.).



A retenir : Un octet (byte)

Un octet est un groupe de huit bits. Il peut représenter 256 valeurs différentes (de 0 à 255).

- Exemple :
 - ◇ 00000101 = 5
 - ◇ 11111111 = 255

1.2.4 Conversion en décimal

Chaque bit a un **poids** qui correspond à une puissance de 2 :

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	32	64	16	8	4	2	1
0	0	0	0	0	0	0	0



Exemple

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	32	64	16	8	4	2	1
0	1	0	0	1	0	0	1

$$01001001 = 64 + 8 + 1 = \mathbf{73}$$

1.3 Représentation de l'information

Pour résumer : un système informatique n'a qu'une seule façon d'encoder une information : le binaire.

Pour chaque type de données, une convention est mise en place pour savoir comment un ordinateur comprendra cette information. Que ce soit pour manipuler un nombre entier, un nombre à virgule, du texte, une image, une vidéo, du sons, ou n'importe quel type de donnée, un système informatique utilisera des 0 et des 1 pour le représenter.

Quelques exemples sont donnés dans cette section

1.3.1 ASCII : Représenter du texte

- Les lettres peuvent être codées en binaire via la norme **ASCII**, qui associe des nombres aux caractères. Par exemple, A correspond à 65 : 01000001.
- Exemple : un message texte pourrait avoir pour valeurs binaires 72, 73 et 33, correspondant aux caractères H, I et !.
- Une table ASCII est fournie avec la feuille *Aide-mémoire*

1.3.2 Unicode : Représenter plus de caractères

- Le binaire traditionnel ne permettait pas de représenter tous les caractères humains. **Unicode** a donc étendu ce système (plus de bits), intégrant notamment les lettre avec accents, les symboles de nombreuses langues ainsi que des emoji.
- Chaque plate-forme peut afficher les emoji différemment.

1.3.3 RGB : Représenter les couleurs

- Les images numériques se construisent à partir de trois composantes : Rouge, Vert, Bleu (**RGB**).
- Si on prend les valeurs 72, 73 et 33 (les mêmes que pour "HI!"), on obtient une nuance de jaune clair.
- Une image est simplement une collection de ces valeurs RGB. Une vidéo = une séquence d'images. La musique peut être codée de manière similaire.

1.4 Algorithmes

Un **algorithme** est une suite d'instructions logiques, organisées et ordonnées, permettant la résolution d'un problème.

Ce problème peut être très simple (ex : *Compter le nombre d'étudiant*), ou très complexe (ex : *Générer une image à partir d'un prompt de l'utilisateur*)



Propriétés d'un algorithme

- **Lisible** : Il doit être compréhensible par tout le monde.
- **Exécutable** : Chaque étape doit correspondre à une opération logique ou mécanique que le système peut réaliser
- **Fini** : L'algorithme doit pouvoir aboutir à une solution après un nombre fini d'étapes.
- **Déterministe** : A chaque étape, pour des entrées données, un seul chemin doit être possible
- **Détermination** : Avec les même conditions, un algorithme doit toujours produire les même résultats.
(Cette propriété ne s'applique pas aux algorithmes probabilistes, type IA)

1.4.1 Exemples d'algorithmes

- **Dans la vie quotidienne** :
 - ◊ Une recette de cuisine (étapes précises à suivre).
 - ◊ Changer une roue (suite d'actions logiques).
- **En informatique** :
 - ◊ Trier des cartes par ordre croissant.
 - ◊ Chercher un nom dans un annuaire.



Exemple de l'annuaire : Recherche dichotomique

1. Prendre l'annuaire
2. Ouvrir l'annuaire au milieu
3. Parcourir la page
4. *Si* la personne se trouve dans la page
 - (a) Appeler la personne
5. *Sinon, si* la personne se trouve plus tôt dans l'annuaire (ordre alphabétique)
 - (a) Ouvrir le livre au milieu de la partie droite
 - (b) Retourner à l'étape 3
6. *Sinon, si* la personne se trouve plus tard dans l'annuaire (ordre alphabétique)
 - (a) Ouvrir le livre au milieu de la partie gauche
 - (b) Retourner à l'étape 3
7. *Sinon*
 - (a) Quitter - La personne ne se trouve pas dans l'annuaire

C'est un **algorithme efficace** : au lieu de parcourir chaque nom un par un, on élimine la moitié des candidats à chaque étape.

2 Hello, World : Premier programme

```
#include <stdio.h>

int main(void)
{
    printf("hello, world\n");
}
```

2.1 Explications

Le programme précédent peut se décomposer de la façon suivante :

- Import de la bibliothèque pour afficher du texte.

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

- Fonction principale :

```
int main(void)
```

◊ Le contenu de cette fonction est compris entre les accolades : { ... }

- Appel de la fonction `printf`

```
printf("...");
```

- `\n` demande un retour à la ligne.
- En C, chaque instruction se termine par un ;

2.2 Compilation et exécution

Encore une histoire de 1 et de 0 ! Pour le moment, notre programme est écrit sous la forme de texte. Il est compréhensible par les humains (développeurs). Il faut, à présent, le convertir en un langage que le processeur comprend : des instructions binaires.

C'est le rôle du **compilateur** : Créer un fichier exécutable à partir d'un code `.c`. Dans ce module, cela se fera avec la commande **make**. Ensuite, on peut exécuter le programme.

```
make hello
./hello
```



Étapes

1. `make hello` → compile.
2. `./hello` → exécute.

Résultat :

```
hello, world
```



Erreurs fréquentes

- Oublier le ; → erreur de compilation.
- Oublier `#include <stdio.h>` → `printf` inconnu.
- Écrire le texte à afficher sans guillemets → interprété comme variable.

3 Types et Variables



Encore une histoire de binaire

On l'a vu, toutes les données sont représentées sous forme binaire (des 1 et des 0). C'est aussi le cas pour les variables que l'on utilisera. Or, l'ordinateur doit pouvoir déterminer s'il faut interpréter le mot binaire de cette variable comme un nombre, une lettre, une couleur ou autre. Il donc faut préciser à quel type est associé chacune des variables que l'on déclare.

3.1 Les types de données en C

Chaque variable a un type précis :

Type	Description	Exemple
<code>int</code>	nombres entiers	10, -5
<code>char</code>	un seul caractère	'a', 'Z'
<code>float</code>	nombres décimaux approximatifs	3.14
<code>double</code>	nombres décimaux avec plus de précision	3.141592
<code>long</code>	entiers plus grands que <code>int</code>	123456789
<code>string</code>	texte (via la bibliothèque CS50)	"Hello"
<code>bool</code>	vrai/faux (via la bibliothèque CS50)	true

3.2 Déclaration et utilisation



Déclaration de variable

Syntaxe : `<type> <nom>;` Exemple :

```
| int nombre;
```

On peut affecter une valeur en même temps :

```
| int nombre = 42;
```

3.3 Afficher une variable dans un texte

Pour afficher une variable dans une phrase, on utilise `printf` avec un **placeholder**.

```
#include <stdio.h>

int main(void)
{
    int age = 20;
    printf("J'ai %i ans.\n", age);
}
```

3.3.1 Placeholders courants

Spécificateur	Description	Exemple d'utilisation	Sortie
%i	entier	<code>printf("Valeur : %i\n", 42);</code>	Valeur : 42
%f	flottant (nombre à virgule)	<code>printf("Pi = %f\n", 3.141593);</code>	Pi = 3.141593
%.2f	flottant arrondi à 2 décimales	<code>printf("Pi = %.2f\n", 3.141593);</code>	Pi = 3.14
%c	caractère	<code>printf("Lettre : %c\n", 'A');</code>	Lettre : A
%s	texte (string)	<code>printf("Bonjour %s\n", "Alice");</code>	Bonjour Alice



Exemple

```
float pi = 3.14159;  
printf("Pi = %.2f\n", pi);
```

Affiche : Pi = 3.14

3.3.2 Imprécision des flottants

Puisqu'ils sont codés sur un nombre de bits finis, les nombres à virgule ont une précision limitée :

```
float x = 1.0 / 10.0;  
printf("%.20f\n", x);
```

Affiche par exemple :

```
| 0.10000000149011611938
```



Conseil

Toujours limiter l'affichage des décimales avec `%.2f`, `%.3f`, etc.

4 Quelques opérateurs mathématiques

En C, on dispose d'opérateurs de base pour manipuler des nombres. Pour des calculs plus avancés (racines, puissances, arrondis...), il existe la bibliothèque standard `math.h`. Elle fournit de nombreuses fonctions que l'on peut appeler dans nos programmes.

4.1 Opérateurs

Symbole	Opération
+	addition
-	soustraction
*	multiplication
/	division
%	modulo (reste de la division entière)

```
int a = 7;
int b = 2;
printf("a / b = %i\n", a / b); // !! Affiche 3 !!
printf("a %% b = %i\n", a % b);
```



Attention

La division entre entiers **supprime la partie décimale**.

4.2 Fonctions usuelles de `math.h`

Vous pourrez trouver plus de fonctions usuelles (mathématiques ou non) sur le site manual.cs50.io. Pour utiliser ces fonctions, il faut inclure la bibliothèque `math.h`.

```
#include <math.h>
```

Fonction	Description	Exemple	Résultat
<code>sqrt(x)</code>	Racine carrée de <code>x</code>	<code>sqrt(9)</code>	3.0
<code>pow(x,y)</code>	Puissance : <code>x</code> élevé à <code>y</code>	<code>pow(2, 3)</code>	8.0
<code>fabs(x)</code>	Valeur absolue (flottant)	<code>fabs(-5.5)</code>	5.5
<code>floor(x)</code>	Arrondi par défaut (partie entière inférieure)	<code>floor(3.7)</code>	3.0
<code>ceil(x)</code>	Arrondi supérieur	<code>ceil(3.2)</code>	4.0
<code>round(x)</code>	Arrondi à l'entier le plus proche	<code>round(3.6)</code>	4.0

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

int main(void) {
5   printf("sqrt(9) = %.1f\n", sqrt(9));
   printf("pow(2, 3) = %.1f\n", pow(2, 3));
   printf("fabs(-5.5) = %.1f\n", fabs(-5.5));
   printf("floor(3.7) = %.1f\n", floor(3.7));
   printf("ceil(3.2) = %.1f\n", ceil(3.2));
10  printf("round(3.6) = %.1f\n", round(3.6));
}
```


5 Les conditions en C

En C, les conditions permettent d'exécuter certaines parties du programme uniquement si des critères sont vérifiés.

5.1 Structure générale : if / else

```
if (<condition1>) // Si la condition 1 est vraie
{
    // Partie exécutée si la <condition1> est vérifiée
}
5 else if (<condition2>) // Sinon, si la condition 2 est vraie
{
    // Partie exécutée si la <condition2> est vérifiée
}
10 else // Sinon, dans tous les autres cas
{
    // Partie exécutée par défaut
}
```

La structure if peut prendre plusieurs formes :

- if seul
- if puis else
- if puis else if puis else

On peut placer autant de else if que l'on veut à la suite.

5.2 Opérateurs de comparaison

Les comparaisons sont faites à l'aide des opérateurs suivants :

Symbole	Signification	Exemple (x=5, y=10)	Résultat
==	égal	x == 5	vrai
!=	différent	x != y	vrai
<	inférieur	x < y	vrai
>	supérieur	x > y	faux
<=	inférieur ou égal	x <= 5	vrai
>=	supérieur ou égal	y >= 11	faux



Piège ! Ne pas confondre affectation et comparaison

L'opérateur = sert à affecter une variable et l'opérateur == sert à comparer deux variables ! Si vous les confondez, rien ne fonctionnera...

5.3 Opérateurs logiques

On peut combiner plusieurs conditions avec les opérateurs logiques :

Symbole	Signification	Exemple (x=5, y=10)	Résultat attendu
&&	ET logique	(x > 0 && y < 20)	vrai
	OU logique	(x < 0 y > 100)	vrai

```
if ((x > 0) && (x < 10))
{
    printf("x est compris entre 0 et 10\n");
}
5
if ((x < 0) || (x > 100))
{
    printf("x est en dehors de l'intervalle [0,100]\n");
}
```

5.4 Un exemple simple

Si x est une variable entière, le code suivant vérifie si x est positif, négatif ou nul.

```
if (x > 0)
{
    printf("x est positif\n");
}
5
else if (x < 0)
{
    printf("x est négatif\n");
}
else
10
{
    printf("x vaut zéro\n");
}
```