IUT ANNECY UNIVERSITÉ SAVOIE MONT BLANC

Informatique

Sequence 1 : Premiers pas

Cours 1.1

v2025-09-03

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 Les 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 `math.h` . . . 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 binare pour encoder toutes les informations Les données sont stoquée 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` $= 64 + 8 + 1 =$ **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 `!`.
- La Figure 1 représente cette table ASCII.

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 symbôles de nombreuse 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.

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

FIGURE 1 – Table ASCII

1.4 Les algorithmes

Un **algorithme** est une suite d'instructions permettant de résoudre un problème.

Propriétés d'un algorithme

- Il doit être **fini** (il s'arrête).
- Il doit être **précis** (chaque étape est claire).
- Il doit être **réalisable** par une machine.

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.

Algorithme de recherche dichotomique

1. Ouvrir l'annuaire au milieu.
2. Si le nom cherché est avant, prendre la moitié gauche.
3. Sinon, prendre la moitié droite.
4. Répéter jusqu'à trouver le nom.

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

```c
#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.

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

- Fonction principale :

```c
int main(void)
```

  - ◇ Le contenu de cette fonction est compris entre les accolades : `{ ... }`

- Appel de la fonction `printf`

```c
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
`int`	nombres entiers	`10`, `-5`
`char`	un seul caractère	`'a'`, `'Z'`
`float`	nombres décimaux approximatifs	`3.14`
`double`	nombres décimaux avec plus de précision	`3.141592`
`long`	entiers plus grands que `int`	`123456789`
`string`	texte (via la bibliothèque CS50)	`"Hello"`
`bool`	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	`printf("Valeur : %i\n", 42);`	`Valeur : 42`
`%f`	flottant (nombre à virgule)	`printf("Pi = %f\n", 3.141593);`	`Pi = 3.141593`
`%.2f`	flottant arrondi à 2 décimales	`printf("Pi = %.2f\n", 3.141593);`	`Pi = 3.14`
`%c`	caractère	`printf("Lettre : %c\n", 'A');`	`Lettre : A`
`%s`	texte (string)	`printf("Bonjour %s\n", "Alice");`	`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)

```c
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`

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

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

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

int main(void) {
    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));
    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

```c
if (<condition1>) // Si la condition 1 est vraie
{
    // Partie exécutée si la <condition1> est vérifiée
}
else if (<condition2>) // Sinon, si la condition 2 est vraie
{
    // Partie exécutée si la <condition2> est vérifiée
}
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

```c
if ((x > 0) && (x < 10))
{
    printf("x est compris entre 0 et 10\n");
}

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.

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