



UNIVERSITÉ
CÔTE D'AZUR

Programmation impérative en C

Cours 3. Codage et mémoire

Olivier Baldellon

Courriel : `prénom.nom@univ-cotedazur.fr`

Page professionnelle : `https://upinfo.univ-cotedazur.fr/~obaldellon/`

LICENCE 2 — FACULTÉ DES SCIENCES ET INGÉNIERIE DE NICE — UNIVERSITÉ CÔTE D'AZUR

- 🍃 Partie I. Remarque préliminaire
- 🍃 Partie II. Codage des données
- 🍃 Partie III. Outils GNU pour coder
- 🍃 Partie IV. Table des matières

```
void f(void) {
    int tab[] = {0};
    int i = 0;;
    while (tab[i] != 777) {
        printf("t[%.2d] = %i\n",
              i, tab[i]);
        i++;
    }
    printf("t[%.2d] = %i\n",
          i, tab[i]);
    tab[i]=666;
}

void fonction(void) {
    f();
}

int main() {
    int variable = 777;
    fonction();
    printf("%d\n", variable);
    return 0;
}
```

*.c

- ▶ En C, chaque fonction peut accéder à-peu-près toute la mémoire allouée.

```
t[00] = 0
t[01] = 1
t[02] = 671837296
t[03] = 32764
t[04] = -1972874112
t[05] = 22038
t[06] = 671837328
t[07] = 32764
t[08] = -1972874089
t[09] = 22038
t[10] = 671837568
t[11] = 32764
t[12] = 0
t[13] = 777
666
```

stdout

- ▶ *Undefined Behavior*
- ▶ On peut même la modifier !
(dangereux!)

- ▶ On range les ordis et on sort une feuille **INTERRO SURPRISE!!!**
- ▶ Écrire une fonction qui affiche sur une ligne la représentation binaire d'un octet.
 - ▶ interdiction d'utiliser le modulo
 - ▶ ou la division entière par deux
 - ▶ Les bits doivent être affichés dans le bon ordre.
- ▶ Exemple :
 - ▶ pour 2 on affichera : 00000010
 - ▶ pour 131 on affichera : 10000011 (131 = 128 + 2 + 1)

- ▶ Pour nous aider à comprendre la mémoire, on va écrire une fonction affichant proprement 8 octets successifs en mémoire.
 - ▶ en décimal
 - ▶ en héra
 - ▶ en binaire

- ▶ Affichage voulu :

								*.c
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	
85	2	10	0	0	0	0	0	
55	02	0A	00	00	00	00	00	
01010101	00000010	00001010	00000000	00000000	00000000	00000000	00000000	
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	

- ▶ À faire chez vous en exercice

- 🍃 Partie I. Remarque préliminaire
- 🍃 Partie II. Codage des données
- 🍃 Partie III. Outils GNU pour coder
- 🍃 Partie IV. Table des matières

- ▶ À quoi sert un type ?
 - À clarifier son code (énumération plutôt qu'entier)
 - À structurer ses données (en les regroupant en structure)
 - À démontrer (en partie) la correction du programme.
- ▶ Mais ce n'est pas la raison historique.
- ▶ Le type permet d'indiquer comment interpréter les bits en mémoire
 - ▶ par exemple 'A' ou 65 ?
 - ▶ à chaque type correspond un encodage de donnée.

- ▶ Comment savoir ce qu'écrit C lorsqu'on stocke une donnée en mémoire ?

```
typedef struct { char r, g, b;} rgb;
typedef struct { char c;
                short s; } alignement1;
typedef struct { unsigned char petit1;
                unsigned short grand;
                unsigned char petit2; } alignement2;

union mon_type {
    long long_s; unsigned long long_u;
    int int_s; unsigned int int_u;
    float float_t;

    rgb rgb_t;
    alignement1 alignement1_t;
    alignement2 alignement2_t;
    unsigned char tableau[8];
};
```

*.c

- ▶ Méthode

- ▶ on stocke un type en mémoire : par exemple `x.long_s = 17`
- ▶ on affiche le tableau `x.tableau` pour voir ce qui est codé.

- ▶ Comment est stocké un entier ?

x.long_s = 655957								*.C												
+-----+-----+-----+-----+-----+-----+-----+-----+																				
	85		2		10		0		0		0		0		0		0		0	
	55		02		0A		00		00		00		00		00		00		00	
	01010101			00000010			00001010			00000000			00000000			00000000			00000000	
+-----+-----+-----+-----+-----+-----+-----+-----+																				

- ▶ Pas facile de comprendre ce qu'il se passe...
- ▶ Pour comprendre, choisissons un entier en héra.

x.long_s = 0xCAFE0230								*.C												
+-----+-----+-----+-----+-----+-----+-----+-----+																				
	48		2		254		202		0		0		0		0		0		0	
	30		02		FE		CA		00		00		00		00		00		00	
	00110000			00000010			11111110			11001010			00000000			00000000			00000000	
+-----+-----+-----+-----+-----+-----+-----+-----+																				

- ▶ Comment écrire 0x1234CAFE (en hexa) dans la mémoire ?
 - ▶ Gros-Boutisme : dans le bon ordre (naturel pour nous autres humains)
 - ▶ Gros-Boutisme : on commence par les bits de poids fort
 - ▶ Petit-Boutisme : à priori ce que votre machine utilise (ARM, x86, x86-64)
 - ▶ Petit-Boutisme : on commence par les bits de poids faibles

Le Gros-Boutisme *big-endian*

12	34	CA	FE
----	----	----	----

Le Petit-Boutisme *little-endian*

FE	CA	34	12
----	----	----	----

- ▶ Intérêt de petit boutisme ?
 - ▶ L'entier 17 est codé pareil quelque soit son type : `int`, `short`, etc.
 - ▶ Plus rapide pour les calculs (qui commencent souvent avec les bits de poids faibles)
- ▶ Ne pas tenter de lire un `int` dans la mémoire octet par octet
 - ▶ sauf pour la curiosité : **risqué et non portable**
 - ▶ les opérations bits à bits marchent indépendamment du boutisme.
 - ▶ utilisez-les si besoin de décomposer un nombre (tp2).

*.C

```
x.int_s = 1
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
|  1  |  0  |  0  |  0  |  0  |  0  |  0  |  0  |
| 01  | 00  | 00  | 00  | 00  | 00  | 00  | 00  |
|00000001|00000000|00000000|00000000|00000000|00000000|00000000|00000000|
+-----+-----+-----+-----+-----+-----+-----+-----+
```

```
x.long_s = 1
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
|  1  |  0  |  0  |  0  |  0  |  0  |  0  |  0  |
| 01  | 00  | 00  | 00  | 00  | 00  | 00  | 00  |
|00000001|00000000|00000000|00000000|00000000|00000000|00000000|00000000|
+-----+-----+-----+-----+-----+-----+-----+-----+
```

```
x.int_s = -1 ou x.int_u = 4294967295
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
| 255 | 255 | 255 | 255 |  0  |  0  |  0  |  0  |
| FF  | FF  | FF  | FF  | 00  | 00  | 00  | 00  |
|11111111|11111111|11111111|11111111|00000000|00000000|00000000|00000000|
+-----+-----+-----+-----+-----+-----+-----+-----+
```

```
x.long_u = -1
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
| 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 |
| FF  | FF  | FF  | FF  | FF  | FF  | FF  | FF  |
|11111111|11111111|11111111|11111111|11111111|11111111|11111111|11111111|
+-----+-----+-----+-----+-----+-----+-----+-----+
```

- ▶ On définit $-n$ comme l'unique nombre vérifiant $n + (-n) = 0$

- ▶ Supposons que l'on travaille avec un octet (8 bits).
 - $255 = \overline{1111\ 1111}_2$ et donc $255 + 1 = \overline{1\ 0000\ 0000}_2$
 - Mais la retenue finale disparaît (on travaille sur 8 bits).
 - On a donc $255 + 1 = 0$ et par suite $255 = -1$

- ▶ En pratique pour tout nombre n on a : $n + \sim n = \overline{11\dots11}_2$
 - et en particulier $n + \sim n + 1 = \overline{00\dots00}_2$
 - Finalement, on code $-n$ par le nombre $\sim n + 1$

- ▶ Comment sont codé les flottants ? Prenons l'exemple du nombre 5.

x.float_t = 5.0;								*.C
0	0	160	64	0	0	0	0	
00	00	A0	40	00	00	00	00	
00000000	00000000	10100000	01000000	00000000	00000000	00000000	00000000	

- ▶ Quel est le rapport avec le nombre 5 ? (car oui, il y en a un !)
- Conservons seulement 32 bits et remettons les dans l'ordre.
- **01000000 10100000 00000000 00000000**
 - ▶ 1 bit pour le **signe** : (0 pour +1 et 1 pour -1)
 - ▶ 8 bits pour l'**exposant biaisé** : ici **1000000 1** donne 129.
 - ▶ 24 bits pour la **mantisse** : ici **0100000 00000000 00000000** ce qui donne 2097152.
- ▶ Bref, rien de bien compliqué...
 - ▶ des questions ?

- ▶ Comment obtenir 5 à partir de ces trois valeurs.

$$\text{signe} \times \left(1 + \frac{\text{mantisse}}{2^{23}} \right) \times 2^{\text{exposant}-127}$$

- ▶ De même que :

- ▶ en décimal $524 = 5,24 \times 10^2$
- ▶ en binaire cinq vaut $\overline{101} = \overline{1,01} \times 2^2$

$$5 = +1 \times \overline{1,01} \times 2^{129-127}$$

- ▶ Pourquoi $1 + \text{mantisse} \times 2^{-23}$?

- ▶ En binaire le premier chiffre vaut toujours 1 : inutile de le stocker.
- ▶ On conserve les 23 premiers bits après la virgule : c'est la mantisse.
- ▶ $1 + \text{mantisse} \times 2^{-23} = 1.\text{0100000 0000000 0000000}$

- ▶ C'est la norme IEEE 754 utilisée dans presque tous les langages.

```
typedef struct{ char r, g, b;} rgb;

typedef struct {
    char c;
    short s;
} alignement1;
```

*.C

- ▶ Les structures sont codées comme des tableaux

```
x.rgb_t : r=11 g=12 et b=13
```

*.C

11	12	13	0	0	0	0	0
0B	0C	0D	00	00	00	00	00
00001011	00001100	00001101	00000000	00000000	00000000	00000000	00000000

- ▶ Avec des cases inutilisées parfois ! Pourquoi?

```
x.alignement1_t c=5 et s=0x0706
```

*.C

5	0	6	7	0	0	0	0
05	00	06	07	00	00	00	00
00000101	00000000	00000110	00000111	00000000	00000000	00000000	00000000

```
typedef struct { unsigned char petit1;
                unsigned short grand;
                unsigned char petit2; } alignement2;
```

*.C

▶ Un champs de longueur n doit être sur une adresse multiple de n

▶ grand est un **short** sur deux octets : l'adresse doit être paire.

▶

0	1	2	3	4	5
---	---	---	---	---	---

 Adresses 2 et 4 valides

0	1	2	3	4	5
---	---	---	---	---	---

▶

0	1	2	3	4	5
---	---	---	---	---	---

 Adresses 1 et 3 invalides

0	1	2	3	4	5
---	---	---	---	---	---

▶ La taille de la structure doit être un multiple de la taille de ces champs.

▶ La structure ne peut pas être de taille 5

▶ Le compilateur rajoute une case après (et y met n'importe quoi : 77)

```
x.alignement2_t petit1=0xFF grand=0xCAFE petit2=0xAA
```

*.C

```
+-----+-----+-----+-----+-----+-----+-----+-----+
| 255  |   0  | 254  | 202  | 170  |  77  |   0  |   0  |
|  FF  |  00  |  FE  |  CA  |  AA  |  4D  |  00  |  00  |
|11111111|00000000|11111110|11001010|10101010|01001101|00000000|00000000|
+-----+-----+-----+-----+-----+-----+-----+-----+
```

▶ Le compilateur fait ce travail pour vous, mais comment connaître la taille ?





- ▶ Pour connaître la taille des données en mémoire on utilise `sizeof`
 - ▶ l'unité est le *byte*, donc concrètement l'octet.
- ▶ On peut l'utiliser avec un type : `sizeof(int)`
- ▶ On peut l'utiliser avec une variable : `sizeof(variable)`

```
typedef struct{
    unsigned char petit1;
    unsigned short grand;
    unsigned char petit2;
} alignement2;

int main() {
    ma_struct s;
    /* Affiche 1 2 1 : logique */
    printf("%ld %ld %ld\n", sizeof(s.petit1)
           , sizeof(s.grand)
           , sizeof(s.petit2));

    /* Affiche 6 : Pourquoi ? */
    printf("%ld\n", sizeof(s));
    printf("%ld\n", sizeof(alignement2));
    return 0;
}
```

*.C

-  Partie I. Remarque préliminaire
-  Partie II. Codage des données
-  Partie III. Outils GNU pour coder
-  Partie IV. Table des matières

- ▶ 1976 : Stallman crée un jeu de macro pour l'éditeur TECO
 - ▶ **Editing Macros** : c'est la première version d'Emacs
- ▶ 1984 : Création du projet GNU (logiciels libres)
 - ▶ Pour que les utilisateurs garde contrôle de leur machine
 - ▶ Pour perpétuer la tradition hackers (collaboration libre et communautaire)
 - ▶ GNU = **G**NU is **N**ot **U**nix : un clone libre d'Unix
- ▶ Les Outils GNU
 - ▶ GNU/Emacs : une des nombreuses versions d'Emacs (depuis 1984)
 - ▶ La GPL (licence libre)
 - ▶ GCC : le compilateur
 - ▶ GDB : le débogueur
 - ▶ La glibc
 - ▶ Bison et Flex (version libre de Yacc et Lex) pour les compilateurs
 - ▶ Bash et les utilitaires Coreutils (`ls`, `mv`, `grep`, etc.)
 - ▶ (Gnome, Gtk, The Gimp étaient à l'époque dans le projet GNU)
 - ▶ Hurd (remplacé par Linux : d'où l'expression GNU/Linux)

- ▶ Initialement un éditeur de texte mais aussi un « *shell* »
- ▶ « *shell* » : un environnement pour interagir avec une machine
 - ▶ Le shell Unix
 - ▶ Gnome Shell (l'interface graphique sous Linux)
 - ▶ un navigateur web (mail, vidéo, tchat)
- ▶ Caractéristique
 - ▶ On peut tout faire dans Emacs
 - ▶ C'est du mode texte
 - ▶ mais plus interactif que la ligne de commande
 - ▶ automatisable et scriptable (contrairement au interface graphique)
 - ▶ avec le contrôle (contrairement au web)

- ▶ Tout est du texte stocké dans des buffers
- ▶ les buffers sont affichés dans les fenêtres
- ▶ Les buffers contiennent des fichiers mais pas uniquement
- ▶ Application sous Emacs :
 - ▶ le mode dired
 - ▶ le mode compilation
 - ▶ le débogueur
 - ▶ les pages de manuel
 - ▶ Magit (un mode pour git excellent)
 - ▶ Org Mode (un mode pour prendre des notes excellent)
 - ▶ un gestionnaire d'emploi du temps
 - ▶ un navigateur web, un lecteur pdf, un psychanalyste, tetris

- ▶ Éditions
 - ▶ Je modifie
 - ▶ Je sauvegarde **C-x C-s**
 - ▶ GOTO compilation
- ▶ Compilation
 - ▶ Je compile **C-c C-c**
 - ▶ En cas d'erreur **M-s M-e** puis GOTO Éditions
 - ▶ GOTO exécution
- ▶ Exécution
 - ▶ **Alt-tab** pour aller sur le terminal (déjà ouvert)
 - ▶ **Flèche du haut** + **Entrée** pour récupérer la commande
 - ▶ **Alt-tab** pour revenir sous Emacs GOTO Éditions

- ▶ Très pratique pour comprendre les erreurs de segmentation
 - ▶ les fameuses *segmentation fault*
 - ▶ Il n'y a pas d'exceptions en C
 - ▶ le débogueur permet de retrouver l'endroit où le code a planté
 - ▶ ainsi que la pile d'appel.
- ▶ Il faut ajouter l'option `-g` à la compilation
 - ▶ `gcc -Wall -ansi -pedantic -g toto.c -o toto`
- ▶ Pour le lancer : `gdb ./toto`
 - ▶ puis on tape `r` ou `run`
 - ▶ puis `bt` ou `backtrace`
 - ▶ voir `bt full` pour le détail
- ▶ Plus intuitif sous Emacs

Merci pour votre attention

Questions



Hello world !



Cours 3 — Codage et mémoire

Partie I. Remarque préliminaire

La mémoire est en accès libre

Afficher en binaire

Afficher un tableau d'octet

Partie II. Codage des données

Concept de types

Union et inspection mémoire

Union : exemple

Gros-boutisme et petit-boutisme

Nombres négatifs

Les nombres négatifs : explications

Les flottants

Les flottants : explications

Structures

Structure et alignement

Taille d'un type

Partie III. Outils GNU pour coder

GNU et Emacs


Qu'est-ce qu'Emacs

Fonctionnement

Travailler efficacement sous Emacs

Le débogueur

Partie IV. Table des matières

- ▶ © 2026 — Olivier Baldellon
- ▶ Ce document est publié sous licence **CC-BY Attribution 4.0** 
 - Vous êtes autorisé à :
 - ▶ **Partager** — copier, distribuer et communiquer le matériel par tous moyens et sous tous formats pour toute utilisation, y compris commerciale.
 - ▶ **Adapter** — remixer, transformer et créer à partir du matériel pour toute utilisation, y compris commerciale.
 - Selon les conditions suivantes :
 - ▶ **Attribution** — Vous devez créditer l'œuvre, intégrer un lien vers la licence et indiquer si des modifications ont été effectuées à l'œuvre. Vous devez indiquer ces informations par tous les moyens raisonnables, sans toutefois suggérer que l'Offrant vous soutient ou soutient la façon dont vous avez utilisé son œuvre.
- ▶ <https://creativecommons.org/licenses/by/4.0/deed.fr>