

INF203 - Elements de programmation C

Victor Lambret

2017 - <https://github.com/VLambret/INF203>

Les bases en informatique

Les bases utiles

- ▶ La plupart des humains ont dix doigts et comptent donc en base dix
- ▶ Cette norme est tellement répandue que les humains n'ayant pas dix doigts comptent aussi en base 10
- ▶ Mais comme le disait une personne célèbre :

Il y a 10 catégories de personnes, les informaticiens et les autres

– Benjamin Franklin

Le binaire : la base 2

- ▶ Le matériel utilisé en informatique est en général limité à stocker l'information sous deux états.
- ▶ On combine ces états pour représenter l'information.
- ▶ La première base utile est donc la base 2

chiffres utilisés :

0 1

L'octal : la base 8

- ▶ Un chiffre octal correspond à 3 chiffres binaires, la conversion est donc facile
- ▶ Utilise les mêmes chiffres que la base 10 :
- ▶ On l'utilise pour les droits Unix (`chmod 764`)

chiffres utilisés :

0 1 2 3 4 5 6 7

L'hexadécimal : la base 16

- ▶ Un chiffre hexadécimal correspond à 4 chiffres binaires, la conversion est donc facile
- ▶ Permet une représentation compacte de données binaires :
11011110101011011011111011101111 se stocke mieux en DEADBEEF

chiffres utilisés :

0 1 2 3 4 5 6 7 8 9 A B C D E F

Exercice

Parmi les nombres suivants :

10	14	ADA	70	0
----	----	-----	----	---

- ▶ Combien sont valides en binaire ?
- ▶ Combien sont valides en octal ?
- ▶ Combien sont valides en décimal ?
- ▶ Combien sont valides en hexadecimal ?

Pour s'exercer

- ▶ Les conversions de base c'est pas super rigolo
- ▶ Pour s'entrainer ludiquement il existe l'application Android Flippy bit
- ▶ Il faut repousser une invasion d'aliens hexadécimaux en donnant leur valeur binaire

Notations en C

```
int n1 = 0b10;    // binaire
int n2 = 010;     // octal
int n3 = 10;      // decimal
int n4 = 0x10;    // hexadécimal
```

- ▶ Attention ! La notation C pour l'octal est très dangereuse, il est facile de la confondre avec du décimal
- ▶ Ces notations se retrouvent dans de nombreux autres langages

Types et représentations

Les entiers

- ▶ En C on peut utiliser plusieurs types de variables pour stocker un entier.
- ▶ Chaque type d'entier est associé à une taille en mémoire.
- ▶ En C la taille de ces types peut varier d'une architecture à l'autre.
- ▶ Il existe des types standard pour préciser la taille de la variable en octet

```
char      : 1 octets  
short     : 2 octets  
int       : 4 octets  
long      : 4 octets  
long long : 8 octets
```

Intervalles de valeurs

- ▶ On choisit le type en fonction de la taille de l'entier à représenter
- ▶ Un octet contient 8 bits, on peut donc stocker 2^8 , soit 256 valeurs différentes
- ▶ Sur 4 octets, soit 32 bits on peut stocker 2^{32} , soit 4 294 967 296 valeurs différentes
- ▶ Sur des entiers signés on dispose de la moitié de cette plage en positif, l'autre moitié en négatif

Exercice

- ▶ Combien de valeurs peut-on représenter avec les 5 doigts d'une main ?

Que choisir ?

- ▶ A moins de progrès fulgurants en médecine, l'âge d'un humain pourra être représenté par un char
- ▶ Par contre pour compter le nombre d'étudiants dans une université on peut prendre un int
- ▶ Pour compter le nombre d'être humains sur terre on devra utiliser un long long

Les caractères

La table ASCII

- ▶ Le standard ASCII associe à un caractère une valeur numérique
- ▶ 128 caractères sont définis de base, ce qui nécessite 7 bits de stockage
- ▶ On peut donc stocker cette valeur numérique sur un octet
- ▶ Le 8ème bit est utilisé dans des variantes pour représenter des caractères spécifiques à une langue

Une représentation affichable

	30	40	50	60	70	80	90	100	110	120

0:	(2	<	F	P	Z	d	n	x	
1:)	3	=	G	Q	[e	o	y	
2:	*	4	>	H	R	\	f	p	z	
3:	!	+	?	I	S]	g	q	{	
4:	"	,	@	J	T	^	h	r		
5:	#	-	7	A	K	U	_	i	s	}
6:	\$.	8	B	L	V	`	j	t	~
7:	%	/	9	C	M	W	a	k	u	SUP
8:	&	0	:	D	N	X	b	l	v	
9:	^	1	;	E	O	Y	c	m	w	

Retour sur le type char

- ▶ Un caractère ascii est stocké comme une valeur numérique sur 1 octet
- ▶ Le type char fait un octet
- ▶ Le nom du type char laisse entendre qu'il est fait pour stocker un caractère
- ▶ C'est vrai ! Mais c'est avant tout une valeur numérique sur un octet

Entier ou caractère

- ▶ le f de printf signifie “format”
- ▶ %c est un format caractère
- ▶ %d est un format entier

```
char    c = 65;  
int     i = 65;  
  
printf("c1=%c, c2=%d\n", c, c);  
printf("i1=%c, i2=%d\n", i, i);
```

Qu'affiche ce code ?

Reponse

c1=A, c2=65

i1=A, i2=65

- ▶ La même chose !
- ▶ Autrement dit : l'entier stocke aussi bien la valeur numérique d'un caractère qu'un char, on perd juste de la place inutile.

Exercice

- ▶ Ecrire le corps de la fonction C suivante :

```
char capitalize(char c);
```

- ▶ Si c est une lettre minuscule la fonction renvoie la lettre majuscule associée
- ▶ Si le paramètre n'est pas une lettre majuscule, renvoie c

Les tableaux

Déclaration

- ▶ En C un tableau est un ensemble d'éléments consécutifs de même type
- ▶ On précise le nombre d'éléments entre crochets

```
int tableau[10];
```

On accède à un élément en donnant son indice dans le tableau

```
tableau[3] = 42;
```

Indices

Attention, le code suivant marche !

```
int tableau[10];  
tableau[99] = 42;
```

- ▶ Le C ne vérifie pas que l'indice donné est dans le tableau
- ▶ La commande au dessus marche, il va écrire dans la mémoire correspondant à la 99ème case du tableau si elle existait
- ▶ Cette case mémoire contient vraisemblablement autre chose, son contenu sera écrasé
- ▶ Pour éviter ce bug il faut vérifier qu'on accède aux cases de 0 à taille - 1

Les chaines de caractères

Déclaration

```
char *name = "Bonjour !\n";
```

- Une chaîne de caractères est comme un tableau de char, on peut accéder à un caractère particulier ainsi :

```
putchar(name[0]);
```

Préciser la taille

Comme les tableaux on peut aussi préciser la taille du buffer à la création de la variable :

```
char name[20] = "Bonjour !\n";
```

Si la taille du buffer est insuffisante pour la chaîne stockée on obtient un warning :

```
capitalize.c:12:17: warning: initializer-string for array  
of chars is too long
```

```
    char name[5] = "Bonjour !\n";
```

Débordement

- ▶ Comme les tableaux on peut accéder à une case en dehors de la chaîne.
- ▶ Comme pour les tableaux c'est un bug et ça arrive souvent !
- ▶ C'est une faille classique exploitée dans les programmes (on cracke la Wii grâce à un overflow sur le nom du cheval de Link dans Zelda)

```
putchar(name[999]);
```

Connaître de la chaîne stockée

- ▶ Comme on le voit ici la mémoire réservée peut être bien plus grande que la taille de la chaîne stockée
- ▶ Comment connaître la taille de la chaîne ?

```
char name[20] = "Toto";
```

- ▶ En fait dans le buffer la chaîne est stockée sous la forme de 5 caractères :

'T' 'o' 't' 'o' '\0'

- ▶ Le caractère '\0' termine la chaîne.
- ▶ Les fonctions C manipulant des chaînes (comme puts) dépendent de la présence de ce '\0' de fin pour bien fonctionner

Exercice

- ▶ Ecrire la fonction C suivante qui affiche en majuscule le mot passé en paramètre
- ▶ On réutilise ici la fonction capitalize précédente

```
void capitalizeWord(char * word);
```

struct

Utilisation

- ▶ En C il est possible de créer un nouveau type en regroupant d'autres types
- ▶ Cela permet de définir des objets plus complexes à partir d'objets plus basiques
- ▶ Ex : Une position avec 2 entiers, un instant avec 3 entiers

```
struct instant {  
    int heure;  
    int minute;  
    int seconde;  
};
```


Déclaration d'une variable struct

On crée une variable de type structure ainsi :

```
struct instant maintenant;
```

Accès aux champs

- Pour accéder aux champs d'une structure on donne le nom de la variable puis .nomduchamp

Exemple :

```
maintenant.heure = 12;  
maintenant.minute = 0;  
maintenant.seconde = 0;
```

Les pointeurs

Echange de deux entiers

On veut créer une fonction échangeant deux entiers :

```
void echange (int a, int b) {  
    int c ;  
    c = a ;  
    a = b  
    b = c ;  
}
```

Est-ce que cette fonction marche ?

Execution

```
int x= 42 ;  
int y= 13 ;  
  
echange(x, y) ;  
printf("maintenant x vaut %d et y vaut %d \n", x, y) ;
```

donne à l'exécution :

maintenant x vaut 42 et y vaut 13

Les valeurs de x et y ne sont pas échangées

Passage de paramètre de fonction

- ▶ En C, les paramètres d'une fonction sont passés par copie
- ▶ C'est à dire que les fonctions travaillent sur une copie des variables passées en paramètre
- ▶ La fonction précédente a donc échangé les valeurs des copies mais pas des originaux !

Passage par référence

- ▶ En C, les variables sont stockées dans une case mémoire
- ▶ La taille de la case mémoire dépend du type
- ▶ Toutes les cases mémoire ont une adresse

Obtenir l'adresse d'une variable

En C on obtient l'adresse d'une variable à l'aide de l'opérateur &

```
int variable;  
&variable
```

&variable est ici une adresse

Les pointeurs

- ▶ On peut vouloir stocker une adresse dans une variable, ou la passer en paramètre d'une fonction
- ▶ On a donc besoin de manipuler des variables dont le type est une adresse
- ▶ Ces types sont nommés des pointeurs !

Déclaration d'un pointeur

```
int n;  
int *pnombre;  
pnombre = &n;
```

- ▶ Le type de pnombre est `int *`
- ▶ On précise dans le type d'un pointeur le type de contenu pointé

Remarque générale sur les pointeurs 1/3

L'erreur est humaine. . .

– *Benjamin Franklin*

Remarque générale sur les pointeurs 2/3

Mais pour les vraies bêtises il faut un ordinateur
– Benjamin Franklin

Remarque générale sur les pointeurs 2/3

Et les pointeurs pour ça c'est le top du top
– Benjamin Franklin

Passage de paramètre par référence

- ▶ Tout en restant prudent, on peut maintenant modifier la fonction `echange` pour utiliser des pointeurs :

```
void echange2 (int *pa, int *pb)
{
    int c;
    c = *pa ;
    *pa = *pb;
    *pb = c;
}
```

Résultat de l'exécution

```
int x= 42 ;  
int y= 13 ;  
  
echange2(&x, &y) ;  
printf("maintenant x vaut %d et y vaut %d \n", x, y) ;
```

donne à l'exécution :

maintenant x vaut 13 et y vaut 42

Pointeurs et tableaux

Similarités

- ▶ On peut utiliser un pointeur comme si on accédait à un tableau :

```
char tname[] = "Toto";  
char * pname = tname;  
  
putchar(tname[2]);  
putchar(pname[2]);
```

- ▶ ici les deux appels à putchar sont équivalents

Différences

- ▶ Un pointeur est une variable de type adresse, on peut changer sa valeur
- ▶ Cela signifie que l'objet pointé peut changer à l'exécution
- ▶ Un tableau pointe toujours sur le même ensemble d'objets

```
char tname1[] = "Sarah";  
char tname2[] = "Julien";  
char * pname = tname1;  
  
pname = tname2;  
tname1 = tname2;
```

donne l'erreur suivante :

```
echange.c:35:9: error: assignment to expression with array  
    tname1 = tname2;
```

Paramètres

- ▶ Utilisé en tant que paramètre on peut utiliser la notation pointeur ou la notation tableau sans différence.
- ▶ Les deux prototypes de fonctions sont ici parfaitement équivalents :
- ▶ L'usage courant en C est en général d'utiliser la notation pointeur dans ce genre de cas

```
int printString(char * string);  
int printString(char string[]);
```

Exercice

- Ecrire une fonction qui convertit une chaîne de caractères en entier :

```
int mon_atoi(char chaine[]);
```