Méthodologie de conception et de programmation

Cours 1

N. de Rugy-Altherre - Vincent Demange

Présentation de C

Le C a été conçu en 1972 par Dennis Ritchie et Ken Thompson aux Bell Labs pour développer un système d'exploitation UNIX. Il est une évolution du langage B, en ajoutant notamment les types.

Entre 1983 et 1989 l'ANSI (American National Standards Institute) normalise le langage.

Les différentes versions de C sont notées par leur année : C99 pour la version de 1999, C18 pour la dernière (2018)

Les particularités de C

Bas niveau

- C est un langage bas niveau : gestion des ressources de l'ordinateur plus précise.
  - *Avantage* : optimisation ;
  - *Inconvénient* : programmation plus complexe.
- C est (assez) prévisible en terme d'occupation mémoire et de charge de calcul.

C est privilégié lorsque l'on cherche à :

- maitriser les ressources ;
- manipuler les systèmes d'exploitation ;
- prendre en considération l'architecture des ordinateurs.

Différences entre C et Python

	Python	**C**
Espaces et tab	essentiels	utiles
Types	aux valeurs	aux variables
Tableaux	taille variable	taille fixe
Boucles pour	itérateurs	répétitions quelconques
Pointeurs	implicites	explicites
Éxecution	interprété	compilé

C dans le vrai monde

C est très utilisé dans les :

- systèmes embarqués (avion, machine à laver, IRM, ...) ;
- systèmes d'exploitation (Linux, Windows) ;
- jeux vidéo (RPG par exemple) ;
- applications temps réel.

Mais les 5 langages de programmation les plus demandés par les employeurs en France en 2018 sont :

- Java (mentionné dans 42% des offres étudiées) ;
- SQL (35%) ;
- Javascript (27%) ;
- PHP (22%) ;
- .NET (17%).

Extensions de C

- C++ : extension orientée objets de C ;
- C# : dérivé de C++, proche de Java, développé par Microsoft ;
- Objective-C : autre extension orientée objets de C.

Syntaxe de base

- Chaque ligne se fini par un ;
- Les blocs d'instructions sont délimités par des accolades : { }

Exemple

```
int main(int argc, char *argv[]) {
    // generation de clefs
    key_t clef = ftok(FICHIER,NB);
    if(clef == -1) {
        perror("Erreur de creation de clef\n");
        exit(1);
    }

    // Lecture du message
    typedef struct pMsg {
        long mtype;
        int a;
    } pMsg;

    pMsg mes;
    printf("En attente d'un message sur le canal %i\n",fileId);
    if(msgrcv(fileId, &mes, sizeof(pMsg)-sizeof(long),1,0) == -1) {
        perror("Erreur dans la reception du message\n");
        exit(2);
    }
    return EXIT_SUCCESS;
}
```

Code de base

Code de base

Voici le code minimal pour écrire un programme en C.

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

int main(int argc, char * argv[]) {

    // code

    return EXIT_SUCCESS;
}
```

La signification des `include` et arguments sera expliquée plus tard.

Les commentaires

Utilité

Sert à rendre le code compréhensible pour :

- le concepteur ;
- le professeur ;
- les collègues.

Deux types de commentaires :

- `//` : commentaire d'une ligne ;
- `/* ... */` : commentaire sur plusieurs lignes.

Lisibilité du code

Attention

Ce cours est un cours de méthodologie de la programmation, pas un cours de C.

Nous serons particulièrement attentifs à la lisibilité du code : commentaires, présentation, découpe en fonctions...

Les commentaires

Obligatoire

- documentation des fonctions ;
- éléments de compréhension du programme ;
- éclaircissement de détails techniques.

Contrairement à Python, les noms et types des variables ne sont pas des commentaires en C.

Documentation

Dans ce cours nous verrons une façon de générer automatiquement une documentation Html ou pdf des programmes à partir de leur code.

Rappels : représentation des nombres

Binaire

Un entier peut se représenter uniquement avec des 0 et des 1 :

$$x = x_0 2^0 + x_1 2^1 + \ldots + x_n 2^n$$

Par exemple $(13)_{10} = (1101)_2$

Hexadécimal

Un entier peut aussi s'écrire en base 16 :

$$x = x_0 16^0 + x_1 16^1 + \ldots + x_n 16^n$$

Par exemple $(18)_{10} = (12)_{16}$

Notations des nombres

En pratique

	nombre	binaire	hexadécimal
	$(21)_{10}$	$(10101)_2$	$(15)_{16}$
en C	`21`	`0b10101`	`0x15`
	$(123)_{10}$	$(1111011)_2$	$(7B)_{16}$
en C	`123`	`0b1111011`	`0x7b`

En héxadécimal, 10 se note A, 11 se note B etc.

Les variables

Une variable est une case mémoire allouée par le système d'exploitation pour y conserver une valeur.

Variable

Une variable est définie par :

- son type ;
- son nom ;
- une valeur.

En C comme en Python = est le symbole d'affectation.

Attention

- Le symbole = n'est pas symétrique.
- Une variable non initialisée possède une valeur !

Les variables

Exemple

```
int i = 9;  // declaration et initialisation
i = 4;      // modification

int j;      // declaration
j = 0;      // initialisation
```

Les types

Les types pré-définis qu'on utilisera sont :

- **`char`** : un caractère ;
- **`int`** : un entier

Attention

En C il n'y a pas de booléens (remplacés par 0,1) ni de chaîne de caractères comme **types de base**.

Ils sont définis dans `stdbool.h` à inclure si nécessaire :

```
#include <stdbool.h>
```

Les variables et la mémoire

Utilité des types

Les types servent à déterminer le nombre d'octets alloués par l'ordinateur lors de la déclaration d'une variable.

- **`char`** : 1 octet ;
- **`int`** : 4 octets ;
- **`float`** : 4 octets.

Ainsi un **`int`** ou quatre **`char`** occupent autant de mémoire.

Les variables et la mémoire

ArtEoz

ArtEoz est un outil pédagogique permettant de visualiser l'état de la mémoire de votre programme.

Il accepte les langages :

- java
- python
- C

Cet outils, quoique très puissant, n'est pas complet (il n'accepte que les entiers en C, pour le moment).

Les types prédéfinis

Définition

Un **char** contient le point de code d'un caractère dont l'encodage est, en fonction de la machine :

- ASCII (sur 7 bits) ;
- un caractère LATIN-1 (sur 8 bits).

Le point de code est la position du caractère dans la table ASCII (65 pour `A`, 97 pour a).

En mémoire

Dans la case mémoire correspondant à la variable **char** est stockée un entier entre $[0, 2^8 - 1]$: le point de code.

Les types prédéfinis

Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
0	0	000	NULL	32	20	040	&#032;	Space	64	40	100	&#064;	@	96	60	140	&#096;	`
1	1	001	Start of Header	33	21	041	&#033;	!	65	41	101	&#065;	A	97	61	141	&#097;	a
2	2	002	Start of Text	34	22	042	&#034;	"	66	42	102	&#066;	B	98	62	142	&#098;	b
3	3	003	End of Text	35	23	043	&#035;	#	67	43	103	&#067;	C	99	63	143	&#099;	c
4	4	004	End of Transmission	36	24	044	&#036;	$	68	44	104	&#068;	D	100	64	144	&#100;	d
5	5	005	Enquiry	37	25	045	&#037;	%	69	45	105	&#069;	E	101	65	145	&#101;	e
6	6	006	Acknowledgment	38	26	046	&#038;	&	70	46	106	&#070;	F	102	66	146	&#102;	f
7	7	007	Bell	39	27	047	&#039;	'	71	47	107	&#071;	G	103	67	147	&#103;	g
8	8	010	Backspace	40	28	050	&#040;	(	72	48	110	&#072;	H	104	68	150	&#104;	h
9	9	011	Horizontal Tab	41	29	051	&#041;	)	73	49	111	&#073;	I	105	69	151	&#105;	i
10	A	012	Line feed	42	2A	052	&#042;	*	74	4A	112	&#074;	J	106	6A	152	&#106;	j
11	B	013	Vertical Tab	43	2B	053	&#043;	+	75	4B	113	&#075;	K	107	6B	153	&#107;	k
12	C	014	Form feed	44	2C	054	&#044;	,	76	4C	114	&#076;	L	108	6C	154	&#108;	l
13	D	015	Carriage return	45	2D	055	&#045;	-	77	4D	115	&#077;	M	109	6D	155	&#109;	m
14	E	016	Shift Out	46	2E	056	&#046;	.	78	4E	116	&#078;	N	110	6E	156	&#110;	n
15	F	017	Shift In	47	2F	057	&#047;	/	79	4F	117	&#079;	O	111	6F	157	&#111;	o
16	10	020	Data Link Escape	48	30	060	&#048;	0	80	50	120	&#080;	P	112	70	160	&#112;	p
17	11	021	Device Control 1	49	31	061	&#049;	1	81	51	121	&#081;	Q	113	71	161	&#113;	q
18	12	022	Device Control 2	50	32	062	&#050;	2	82	52	122	&#082;	R	114	72	162	&#114;	r
19	13	023	Device Control 3	51	33	063	&#051;	3	83	53	123	&#083;	S	115	73	163	&#115;	s
20	14	024	Device Control 4	52	34	064	&#052;	4	84	54	124	&#084;	T	116	74	164	&#116;	t
21	15	025	Negative Ack.	53	35	065	&#053;	5	85	55	125	&#085;	U	117	75	165	&#117;	u
22	16	026	Synchronous idle	54	36	066	&#054;	6	86	56	126	&#086;	V	118	76	166	&#118;	v
23	17	027	End of Trans. Block	55	37	067	&#055;	7	87	57	127	&#087;	W	119	77	167	&#119;	w
24	18	030	Cancel	56	38	070	&#056;	8	88	58	130	&#088;	X	120	78	170	&#120;	x
25	19	031	End of Medium	57	39	071	&#057;	9	89	59	131	&#089;	Y	121	79	171	&#121;	y
26	1A	032	Substitute	58	3A	072	&#058;	:	90	5A	132	&#090;	Z	122	7A	172	&#122;	z
27	1B	033	Escape	59	3B	073	&#059;	;	91	5B	133	&#091;	[	123	7B	173	&#123;	{
28	1C	034	File Separator	60	3C	074	&#060;	<	92	5C	134	&#092;	\	124	7C	174	&#124;	|
29	1D	035	Group Separator	61	3D	075	&#061;	=	93	5D	135	&#093;	]	125	7D	175	&#125;	}
30	1E	036	Record Separator	62	3E	076	&#062;	>	94	5E	136	&#094;	^	126	7E	176	&#126;	~
31	1F	037	Unit Separator	63	3F	077	&#063;	?	95	5F	137	&#095;	_	127	7F	177	&#127;	Del

asciicharstable.com

Les types

Un entier peut être signé ou non, **signed** ou **unsigned**, et précédé d'un attribut de précision **short** ou **long**.
Ainsi en fonction de la taille de l'entier à stocker il faudra choisir un type (ces paramètres peuvent changer en fonction de la machine) :

Types entiers

signed char	$[-2^7; 2^7[$	1 octet
unsigned char	$[0; 2^8[$	1 octet
short int	$[-2^{15}; 2^{15}[$	2 octets
unsigned short int	$[0; 2^{16}[$	2 octets
int	$[-2^{31}; 2^{31}[$	4 octets
unsigned int	$[0; 2^{32}[$	4 octets
unsigned long int	$[0; 2^{64}[$	8 octets

Les types

Les valeurs maximales et minimales des différents types entiers sont définies dans la bibliothèque standard `limits.h`. Le mot clef **`sizeof`** indique le nombre d'octets nécessaires pour stocker le type demandé :

```
taille = sizeof(unsigned short)
```

Conversions

Il est possible de convertir des éléments d'un type à un autre.
Il suffit d'indiquer le type voulu entre parenthèses :

```
short int i = 234;
int j = (int) i;
```

Attention

Le résultat dépend de la valeur à convertir et des types des variables en jeu.

Les types

Flottant

Les flottants (nombres à virgule flottante) approchent les réels. La précision de l'approximation est variable mais limitée.

Deux types de flottant :

- **`float`** sur 32 bits ;
- **`double`** sur 64 bits.

Les tailles données servent à représenter : le signe, la mantisse et l'exposant (resp. les chiffres avant et après la virgule).

Les opérateurs

- **Arithmétiques** : `+`, `-`, `*`, `/`, `%`.
  - a%b : reste de la division euclidienne de *a* par *b* ;
  - a/b : du même type que *a* et *b* : entier si les opérandes sont des entiers, flottant si les opérandes sont des flottants ;
  - pas d'opérateur prédéfini pour la puissance.
- **Relationnels** : `==`, `!=`, `>`, `>=`, `<`, `<=`
  La valeur retournée est un **`int`** valent `0` si c'est faux, `1` sinon.
- **Booléens** : && (et), || (ou), ! (non).
- **Opérations au niveau des bits** (bit à bit) :
  & (et), | (ou), ^ (ou exclusif), ~ (complément à 1),
  << (décalage à gauche), >> (décalage à droite).

Les booléens

Les booléens

En C :

- `Vrai` est représenté par l'entier `1`
- `Faux` est représenté par l'entier `0`

Opérateurs

Des opérateurs composés existent pour simplifier les écritures

- **Composés** : `+=`, `-=`, `*=`, `/=`, `%=`
- **Incrémentation** : `++` et `--`

Exemples

`a += b;` équivalent à `a = a + b;`

`i++;` ou `++i;` équivalents à `i = i + 1;`

Les constantes

Définition

Une constante est une variable définie et initialisée au début du programme et ne pouvant pas être modifiée ensuite. Il y a trois types de constantes :

- constantes typées de C :
  ```
  const int var = 5;
  ```
- constantes du pré-processeur, définies en début de fichier :
  ```
  #define MAX 100
  ```
- Les énumérations :
  ```
  enum DIRECTION {NORD, SUD, EST, OUEST};
  ```

Une constante pré-processeur est remplacée avant la compilation. Elle n'occupe pas de mémoire.

Affichage

Printf

`printf(s,v1,v2,...,vk)` : affiche la chaîne de caractères `s` en remplaçant dans l'ordre les `%t` par les valeurs `vi`.

Les valeurs possibles de `t` dépendent du type de la variable `vi` :

- `d` ou `i` : entier décimal signé ;
- `u` : entier décimal non signé ;
- `f` : flottant décimal ;
- `c` : caractère ;
- `s` : chaîne de caractère.

D'autres existent : `x` pour affichage hexadécimal, `e` pour affichage d'un flottant sous forme mantisse/exposant, ...

Affichage

Printf exemples

```
int i = 0;
float f = 1.224;
char c = 64;
printf("i=%d, f=%f, c=%c\n", i, f, c);
```

Affiche : `i=0, f=1.224000, c=@`

\n correspond à un saut de ligne (non systématique).

On peut afficher un **char** comme un **int** ou inversement :

Printf

```
char c = 83;
printf("%d %c\n",c,c);
```

Affiche : `83 S`

```
int i = 83;
printf("%d %c\n",i,i);
```

Affiche : `83 S`

Conditionnelles **if**

```
if(condition1) {
    // instructions 1
} else if(condition2) {
    // Instructions 2
}
...
} else {
    // instructions 3
}
```

Branchement conditionnel standard :

- un seul **if** ;
- au plus un **else** ;
- autant de **else if** que l'on veut (éventuellement aucun).

Conditionnelles **switch**

```
switch(expression) {
    case(constante1):
        // instructions 1
        break;
    case(constante2):
        // instructions 2
        break;
    ...
    default:
        // instructions par defaut
        break;
}
```

Si la valeur `expression` est égale à l'une des $\texttt{constante}_i$, le bloc d'instructions correspondant est exécuté.
S'il n'y aucune correspondance c'est le bloc d'instructions par défaut. Le bloc **`default`** est facultatif.

Conditionnelles **switch**

Exemple

```
switch(age) {
    case 2:
        printf("Bonjour bebe");
        break;
    case 18:
        printf("Madame");
        break;
    default:
        printf("Vieux ?");
        break;
}
```

Boucles

Boucle **while**

```
int i = 0;
while(i < 100) {
    // instructions
    i++;
}
```

Boucles

Boucle **for**

La boucle **for** comporte trois parties :

- initialisation, p. ex. `i = 0;`
- condition de continuation, p. ex. `i < 100;`
- incrémentation, p. ex. `i++`.

Exemples

```
for(i = 0; i < 100; i++) {
    // instructions
}

for(i = 100; i >= 0; i-=2) {
    // instructions
}
```

Sources

- Programmation en langage C, *Anne Canteaut*, INRIA

`http://www-rocq.inria.fr/codes/Anne.Canteaut/COURS_C`

- `https://openclassrooms.com/fr/courses/19980-apprenez-a-programmer-en-c`