Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Systèmes Embarqués 1 & 2

a.09 – C – Les données

Classes T-2/I-2 // 2017-2018

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz

Daniel Gachet | HEIA-FR/TIC
a.09 | 23.10.2017

Contenu

- Déclaration de variables
- Nombres entiers
- Nombres booléens
- Nombres réels
- Chaînes de caractères
- Constantes
- Classes de mémorisation

En C, la déclaration de variables prend la forme suivante:

```
TYPE var_name [= init_value] {, var_name [= init_value]};
```

Par exemple:

```
int   var1, var2;          → identique à  int var1; int var2;
float var4;
int   var5, var6 = 10;     → identique à  int var5; int var6 = 10;
float var7 = 1.0;
char *var8 = "hello world!";
int  *var9, var10;         → identique à  int *var9; int var10;
```

Remarque:

- Il est préférable de ne déclarer qu'une variable par ligne.

Attention:

- La déclaration de variables doit impérativement être faite au début d'un block, avant la première instruction/opération (restriction éliminée depuis **C99**)

Nombres entiers

C est capable de traiter aussi bien des nombres signés que non-signés et de longueurs différentes.

Limites des nombres signés: -2^{n-1} .. $2^{n-1} - 1$ (voir le fichier <limit.h>)

Limites des nombres non-signés: 0 .. $2^{n} - 1$

Types signés:	non-signés:	Bits:	stdint.h:
`signed char`	`unsigned char`	`8`	`[u]int8_t`
`signed short int`	`unsigned short int`	`souvent 16`	`[u]int16_t`
`signed int`	`unsigned int`	`16 or 32`	
`signed long int`	`unsigned long int`	`32`	`[u]int32_t`
`signed long long int`	`unsigned long long int`	`64(GNU)`	`[u]int64_t`

Remarque:

Pour les types **`short`**, **`long`** et **`long long`**, le préfix **`int`** peut être omis.
Pour les données de types signés, le qualificatif **`signed`** peut également être omis.

Attention:

Le langage ne spécifie pas si les données de type **`char`** sont de type signé ou non.

Opérateurs arithmétiques & comparateurs

Opérateurs:

+	addition
-	soustraction
*	multiplication
/	division
%	reste/modulo

Comparateurs:

==	égal
!=	différent
<	plus petit que
>	plus grand que
<=	plus petit ou égal
>=	plus grand ou égal

Attention:

- C ne performe aucun test de dépassement de capacité.
- Une division par zéro (0) produit un crash de l'application.

Opérateurs bit à bit

Opérateurs:

¦	OU bit à bit
&	ET bit à bit
^	OU exclusif (bit à bit)
~	complément à 1
<<	décalage sur la gauche
>>	décalage sur la droite

Attention:

Le comportement de l'opérateur bit à bit « **>>** » sur les nombres signés n'est pas défini par le langage et dépend du compilateur.

Nombres booléens et les opérateurs

Le langage C ne définit aucun type spécifique pour les données de type logique. Par contre, il propose 3 opérateurs permettant d'exécuter des opérations logiques sur des nombres entiers.

Opérateurs:

`&&`	ET logique
`||`	OU logique
`!`	NON logique

Le type `bool` a été introduit en C99. Le type est défini dans le fichier `<stdbool.h>`, lequel fournit également les constantes `true` (1) et `false` (0).

Remarque:

Toutes les valeurs différentes de 0 (!= 0) représentent un vrai « true », tandis que
la valeur 0 (== 0) représente un faux « false ».

Nombres réels

C est également capable de traiter des nombres réels (nombres à virgule flottante).

`float`	**simple précision**
`double`	**double précision**
`long double`	**précision étendue**

Opérateurs & Comparateurs:

A l'exception du modulo (`%`), tous les opérateurs et comparateurs des nombres entiers sont également supportés sur les nombres réels.

Remarque:

Sur la plupart des systèmes embarqués, le traitement des nombres réels est effectué par des bibliothèques d'émulation de coprocesseur mathématique. Ceci rend le temps de calcul souvent très long.

Attention:

Le langage ne spécifie pas la précision de ces types. Celle-ci est dépendantes des microprocesseurs utilisés et des compilateurs. La plage de valeur est défini dans le fichier <float.h>.

Caractères et chaîne de caractères (strings)

C supporte, comme tous les autres langages de programmation, un type de données «caractère» et les chaînes de caractères (strings) afin de permettre aux humains d'interagir avec les machines.

`char`	**un seul caractère**
`char*, char[]`	**une chaîne ouverte de caractères**
`char[n]`	**une chaîne avec un nombre fixe de caractères**

Opérateurs & Comparateurs:

Sur les données de type char, tous les opérateurs et comparateurs des nombres entiers peuvent être utilisés. Par contre, C n'offre aucun support pour le traitement des chaînes de caractères. Celui-ci est laissé au programmeur. Cependant, les librairies "string.h" et "ctype.h" mettent à disposition des fonctions facilitant grandement le travail.

Remarque:

Pour traiter les strings, la maîtrise des tableaux et pointeurs est indispensable.
Il est important de noter que les strings sont/doivent être terminés par un zéro ('\0').

Attention:

Le langage ne spécifie pas si les données de type `char` sont de type signé ou non.

Caractères et chaîne de caractères (exemples)

```
char  a_character   = 'a';
```
→ un caractère seul

```
char* a_1st_string     = "Hello Guys";
```
→ pointe sur une chaine «constante» de 10 caractères + '\0'

```
char  a_2nd_string[]   = "Bonjour tout le monde";
```
→ tableau de 21 caractères + '\0'

```
char  a_3rd_string[32] = "Good Morning";
```
→ 12 caractères + '\0' contenus dans un tableau de 32 éléments

C supporte le concept de constantes pour tous les types de base.

Nombres entiers (`int`):

- Décimal : `1234`
- Octal : `0ooo` `(017, 0377)` → $(15_{10}, 255_{10})$
- Hexadécimal : `0xhh` or `0XHH` `(0xf, 0XFF)` → $(15_{10}, 255_{10})$
- Suffixes:

`l(L)`	pour un `long int`	`(-771)`
`ll(LL)`	pour un `long long int`	`(771l)`
`u(U)`	pour un `unsinged int`	`(77u)`
`ul(UL)`	pour un `unsigned long int`	`(77ul)`
`ull(ULL)`	pour un `unsigned long long int`	`(77ull)`

Nombres réels (`double`):

- Notation décimale : `123.45`
- Notation scientifique : `1e-2`
- Notation combiné: `123.45e10`
- Suffixes:

`f(F)`	pour un `float`	`(123.45f)`
`l(L)`	pour un `long double`	`(198.45e-48l)`

Caractère (`char`):

- Un caractère : `'x'` → caractère x, `'0'` → caractère 0 (zéro), …
- Octal : `'\ooo'` `('\170', '\60')` → `('x', '0')`
- Hexadécimal : `'\xhh'` `('\x78', '\x30')` → `('x', '0')`
- Escape:

`\n`	newline	`\\`	backslash
`\'`	single quote	`\"`	double quote
`\b`	backspace	`\v`	vertical tab
`\f`	formfeed	`\t`	horizontal tab
`\a`	alert (bell)	`\?`	question mark
`\r`	carriage return		

Strings (`char*`):

- un string: `"I am a non empty string"`
- un string vide: `""`
- une concaténation de strings: `"A "  "concatenated "  "string"`

Il existe deux manières de déclarer des constantes en C:

Constantes symboliques :

```
#define  name   replacement_text
p.ex.  #define GRETTINGS   "Hello World!"
       #define LOWER       100
       #define UPPER       345
       #define ELEMENTS    (UPPER - LOWER + 1)
```

Variables constantes :

```
[static] const TYPE name = value;
p.ex.  const char   GRETTINGS[] = "Hello World!";
       const uint32 LOWER       = 100;
       const uint32 UPPER       = 345;

mais
       const uint32 ELEMENTS    = UPPER - LOWER + 1;
n'est pas supportée
```

Le langage C définit différentes classes de mémorisation des variables. Celles-ci prennent la forme suivante:

```
<classe> <type> declarator;
```

Variable	Classe	Portée	Durée de vie	Explication
locale	auto	bloc	bloc	Lassé au choix du compilateur (défaut)
	register			La variable est placée dans un registre
	volatile			La variable est placé sur la pile
	static		programme	La variable est placé avec les variables globales
globale	extern	fichier source avec les définitions et après chaque déclaration tous les modules du programme	programme	Référence sur une variable globale
	static	fichier source avec définition		La variables est placé avec les variables globales dans une zone mémoire globale au programme et dédiée à cet effet