



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

# Systèmes Embarqués 1 & 2

## a.07 – C – Les pointeurs

Classes T-2/I-2 // 2018-2019

**Hes·SO**

Haute Ecole Spécialisée  
de Suisse occidentale

[Gac/a.07] T-2/I-2 // 09.2018

Daniel Gachet | HEIA-FR/TIC  
a.07 | 10.09.2018



## **Contenu**

---

- **Déclaration**
- **Affectations et déréférenciations**
- **Opérateurs et comparateurs**
- **Accès à des périphériques**
- **Objets dynamiques**
- **Conversion de types**



## Déclaration

Les pointeurs sont des éléments très importants d'un langage de programmation de haut niveau. Le pointeur est une variable qui contient l'adresse où est stockée une autre donnée.

Les pointeurs sont très souvent utilisés en C parce qu'ils permettent de:

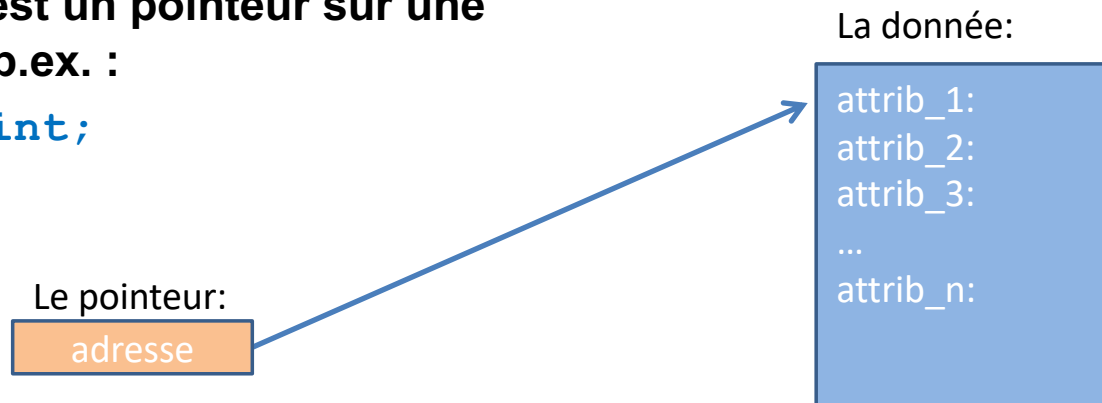
- ▶ Retourner des résultats lors d'un traitement d'information par une fonction
- ▶ Créer et traiter des objets dynamiquement
- ▶ Accéder à des registres de périphériques (hardware registers)

La déclaration d'un pointeur prend la forme suivante:

```
<type_name>* <variable_name>;
```

Le '**\***' indique que la variable est un pointeur sur une donnée de type **<type\_name>**, p.ex. :

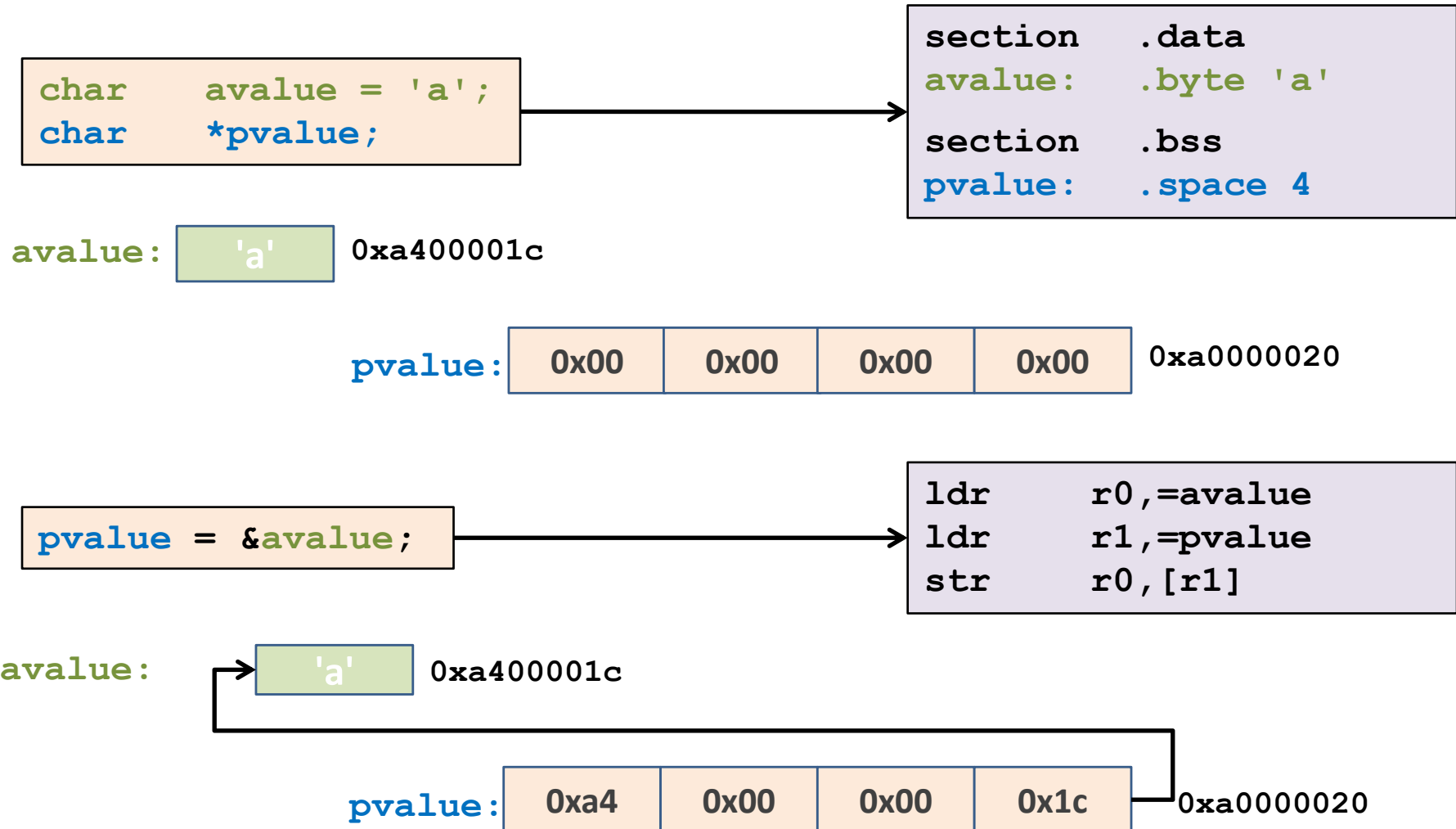
```
struct point* ppoint;
```





## Affectation

L'adresse d'un objet peut facilement être obtenue avec l'opérateur '&' et assignée à une variable de type pointeur, p. ex. :



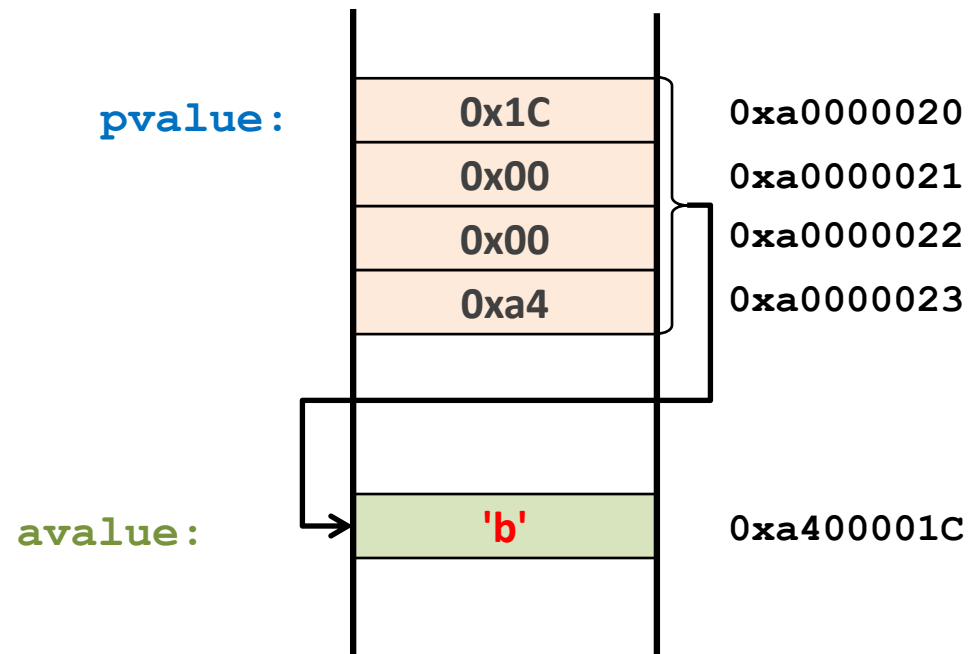


## Déréférenciation: types de base

L'opérateur '\*' permet d'accéder le contenu référencé par le pointeur (accès indirect – déréférenciation), p. ex.:

`*pvalue = 'b';`

```
ldr    r1,=pvalue
ldr    r5,[r1]
mov    r6,#'b'
strb   r6,[r5]
```





```
struct point apoint;  
struct point* ppoint = &apoint
```

**équivalent**

ppoint:

&apoint

10

30

La forme `int x = *ppoint.x` n'est pas valide. En effet, le compilateur interprète l'affectation comme `int x = *(ppoint.x)`, en déréférençant le membre `x` et non pas `ppoint`.



## Déréférenciation: tableaux et pointeurs

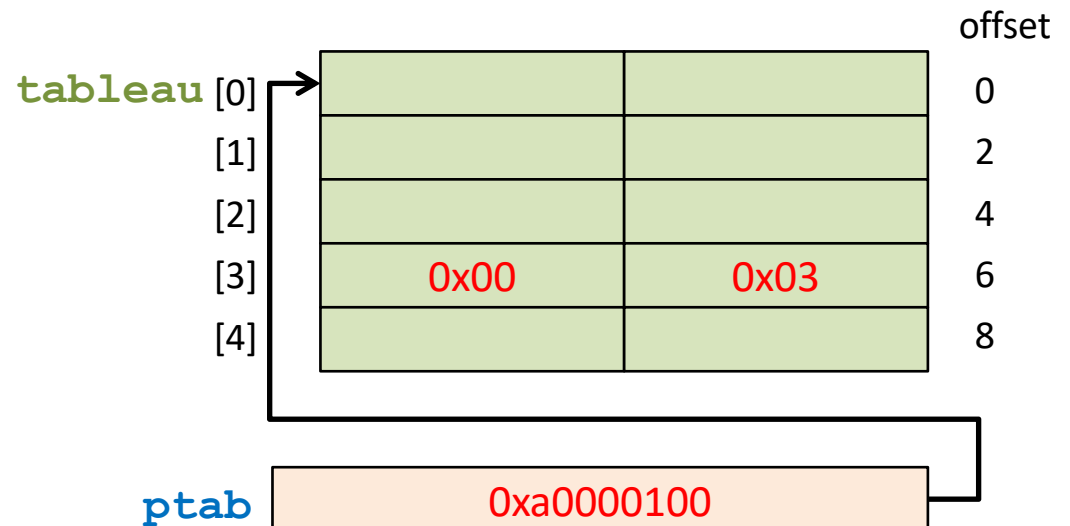
C ne fait que peu de différences entre les pointeurs et les tableaux. Il permet ainsi d'accéder les éléments d'un tableau en déréférençant un pointeur et vice-versa, p. ex.:

```
short  tableau[5];  
short *ptab;
```

```
section .bss  
tableau: .space 5*2  
ptab:    .space 4
```

```
ptab = tableau;  
ptab[3] = 3;
```

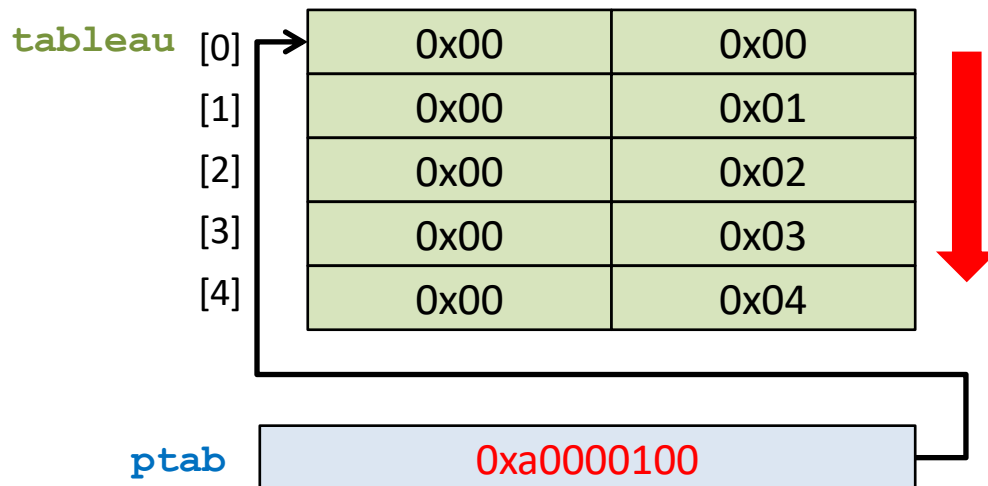
```
ldr    r0,=tableau  
ldr    r1,=ptab  
str    r0,[r1]  
  
ldr    r3,[r1]  
mov    r4,#3  
strh   r4,[r3,#6]
```





## Déréférenciation: tableaux et pointeurs (II)

```
ptab = tableau;  
for(int i=0; i<5; i++)  
{  
    *ptab++ = i;  
}
```



```
ldr    r0,=tableau  
ldr    r1,=ptab  
str     r0,[r1]  
  
mov     r2, #0  
ldr     r3,[r1]  
b       test  
  
loop:   strh    r2,[r3],#2  
        add     r2,#1  
  
test:   cmp     r2,#5  
        blo     loop
```





## Opérateurs arithmétiques & comparateurs

---

### Opérateurs:

+	addition
-	soustraction
++	incrémentation
--	décrémentation

### Comparateurs:

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

### Remarque:

L'arithmétique sur les pointeurs tient compte de la taille du type de donnée référencié pour effectuer l'opération, p. ex.

```
char *pc = (char*)100;  pc++;  // → pc == 101
long *pl = (long*)100;  pl++;  // → pl == 104
```



Lorsqu'un pointeur référence les registres d'un périphérique avec accès mémoire direct (**memory mapped device**), l'adresse mémoire où sont localisés les registres prend la forme suivante:

```
struct hw_regs {  
    uint16_t reg1;  
    uint16_t reg2;  
    uint16_t reg3;  
    uint16_t reg4;  
};  
  
volatile struct hw_regs * hwreg = (struct hw_regs *)0xd0000000;
```



Les pointeurs permettent la manipulation dynamique d'objets. Ceux-ci peuvent être créés et supprimés simplement avec l'aide des méthodes "`malloc`", "`calloc`" et "`free`" de la bibliothèque standard `<stdlib.h>`, p. ex.:

```
// declaration of the operators malloc/calloc & free
#include <stdlib.h>

// creation of dynamic objects
struct point* ppoint = malloc (sizeof(struct point));
// ou
struct point* ppoint2 = calloc (1, sizeof(*ppoint2));

// deletion of dynamic objects
free (ppoint);  ppoint  = 0;
free (ppoint2); ppoint2 = NULL;
```

La constante `NULL` ou la valeur `0` indique que le pointeur ne référence aucun objet (`NULL` est défini dans `<stddef.h>`).



## Conversion de types

---

C permet de convertir la valeur d'une donnée d'un type (source) dans un autre type (cible). On parle aussi de coercition ou en anglais de « type casting ».

C distingue deux types de conversion, l'implicite et la conversion explicite.

La conversion implicite est utilisée lors de l'évaluation d'expressions impliquant les types de base du langage, p.ex. les `char` et les `short` sont convertis en `int`.

La conversion explicite est couramment utilisée par le programmeur lors de manipulation de pointeurs et d'objets dynamiques. La forme la plus simple est l'utilisation de `union`, permettant de voir une source sous plusieurs types. La deuxième prend la forme suivante:

```
type_a value_a;  
type_b value_b = (type_b) value_a;
```