



Université Internationale  
de Casablanca

UNIVERSITÉ RECONNUE PAR L'ÉTAT

# Programmation Structurée 2 (Suite)

---

Pr. EL OUKKAL Sanae



# NOTION DE POINTEUR

# Pointeurs

- La notion de pointeur est spécifique aux langages C et C++.
- Une adresse est un emplacement donné en mémoire.
- Un pointeur est une variable qui contient l'adresse d'une autre variable de n'importe quel type.

# Pointeurs

- Un pointeur est une variable qui permet de stocker une adresse, il est donc nécessaire de comprendre ce qu'est une adresse.
- Lorsque l'on exécute un programme, celui-ci est stocké en mémoire, cela signifie que chaque variable que l'on a défini a une zone mémoire qui lui est réservée, et la taille de cette zone correspond au type de variable que l'on a déclaré.

# Pointeurs

- On peut donc accéder à une variable de 2 façons :
  - Grâce à son nom,
  - Grâce à l'adresse du premier bloc alloué à la variable,

# Pointeurs: Exemple

- `int *ptr ;`
  - Le pointeur « ptr » contient l'adresse mémoire d'une zone mémoire capable de contenir une variable de type « int »,
  - La définition d'un pointeur n'alloue en mémoire que la place mémoire nécessaire pour stocker ce pointeur et non pas ce qu'il pointe,
  - Il est préférable d'initialiser le pointeur avant de l'utiliser.

## La syntaxe (déclaration ou définition)

Type \*identificateur ;

Ou bien

Type\* identificateur ;

- Le caractère ' \* ' avant l'identificateur  $p$  indique que  $p$  est un pointeur.
- $int$  spécifie le type du pointeur qui indique le type de l'objet pointé par  $p$ ;

# Pointeurs:

## Les Opérateurs Unaires & et \*

- &:
  - Pour l'adresse d'une variable,
  
- \*:
  - Pour le contenu de l'adresse pointée, on parle alors de la valeur *déréférencée* d'un pointeur,



# Pointeurs:

## Les Opérateurs Unaires & et \*

- Exemple :

```
int a, * p; /* on définit une variable entière a et un pointeur p
           sur un entier */
```

```
p = &a; /* initialisation du pointeur :p contient alors l'adresse
        de a, il pointe sur a*/
```

```
*p = 4; /* on donne la valeur 4 à la valeur déréférencée du
        pointeur p; la variable a vaut donc 4 */
```

# Pointeurs

- Ainsi, pour un pointeur  $p$ , il ne faut pas confondre les 3 notions essentielles :
  - $\&p$ :
    - Adresse du pointeur  $p$ ,
  - $p$ :
    - Valeur du pointeur, soit l'adresse d'une autre variable,
  - $*p$ :
    - Contenu de l'adresse pointée par  $p$ ; valeur déréférencée du pointeur  $p$ .

# Pointeurs

- \* et & sont deux opérateurs unaires de même priorité avec liaison de l'opérande de la droite vers la gauche.
- Si un pointeur p pointe sur une variable a alors \*p peut être utilisé partout où a pourrait l'être.

# Pointeurs

- Exemple : Voici un bloc d'instructions :

{

```
int a, b, * p;
```

```
a = 10; /* initialisation de a */
```

```
p = &a; /* initialisation de p */
```

```
b = *p + 2; /* soit b = 10 + 2 */
```

```
*p = *p + 4; /* soit a = a + 4 */
```

```
*p += 11; /* soit a += 11 */
```

```
*p = 0; /* soit a = 0 */
```

}

# Incrémentation des Valeurs Déréférencées

- Les équivalences suivantes ont lieu :
  - $++*p \leftrightarrow ++(*p)$ :
    - pré\_incrémentation de la valeur déréférencée.
  - $*++p \leftrightarrow *(++p) \leftrightarrow ++p$ :
    - pré\_incrémentation du pointeur (adresse "suivante")
  - $*p++ \leftrightarrow *(p++) \leftrightarrow p++$ :
    - post\_incrémentation du pointeur (adresse "suivante").

# Incrémentation des Valeurs Déréférencées: Exemple

{

```
int x, y, * p;
```

```
x = 2;
```

```
p = &x;
```

```
++*p; /* soit ++x donc x vaut 3 */
```

```
(*p)++; /* soit x++ donc x vaudra 4 à la  
prochaine utilisation */
```

```
y = ++*p; /* y vaut 5 */
```

```
y = (*p) ++; /* y vaut 5 et x vaut 6 */
```

```
y = *p; /* y vaut 6 */ }
```

# Décrémentation des Valeurs Déréférencées

- Les règles et problèmes rencontrés sont les mêmes que ceux liés à l'incrémentation.

# Pointeurs & Affectation

- Les pointeurs étant des variables, l'affectation entre deux pointeurs est possible :  $p1 = p2$ ;
- Cette instruction copie la valeur de  $p2$  dans  $p1$ .
  - Autrement dit  $p1$  pointe vers le même objet que  $p2$ .
- Attention :
  - Il faut que les pointeurs soient de même type. Sinon un warning est indiqué à la compilation quant à un problème potentiel d'alignement en mémoire.
  - Il convient alors de recourir à des conversions explicites de pointeurs.



# Pointeurs et Argument de Fonctions

- Une utilisation fréquente des pointeurs est de les passer en arguments des appels de fonction.
- En C, le passage des paramètres se fait par valeur.
- Ainsi si une fonction modifie les arguments, cette modification n'est valable que dans le corps de la fonction. Au retour de l'appel, les arguments ont conservé leurs valeurs. Les valeurs des arguments sont copiées dans les paramètres formels sur la pile.

# Pointeurs et Argument de Fonctions

- Pour conserver les modifications au retour de l'appel, les arguments doivent être des adresses. Les paramètres doivent être alors des variables pouvant contenir des adresses, autrement dit des pointeurs.
  - Exemple: `void echange (int * x, int * y);`
- Ce mode de passage des paramètres s'appelle passage par référence ou aussi passage par valeur ou par copie (car ce sont les valeurs des pointeurs qui sont copiées sur la pile).

# Pointeurs et Tableaux

- En C, une étroite relation existe entre les tableaux et les pointeurs.
- Toute opération mettant en œuvre un sous-tableau ou un élément d'un tableau peut être réalisée à l'aide de pointeurs.
- Le code généré à partir des pointeurs sera en général plus rapide.
  - L'opérateur primaire `[ ]` permet l'accès à un élément d'un tableau.
  - `a [n-1]` permet l'accès au nième élément d'un tableau `a`.
  - Les indices des tableaux commencent à 0.

# Pointeurs et Tableaux:

## Exemple

- Considérons les instructions suivantes :

```
int x;
```

```
int a [10];/* définition d'un tableau de 10 entiers */
```

```
int * pa;/* définition d'un pointeur d'entier */
```

```
a[0] = 4;/* affecte la valeur 4 au 1er élément */
```

```
a[3] = 2;/* affecte la valeur 2 au 4ème élément */
```

```
pa = &a[0];/* pa pointe sur le 1er élément du tableau. pa  
contient l'adresse du 1er élément */
```

```
x = *pa;/* copie la valeur du 1er élément du tableau dans  
la variable x. Alors x vaut 4. */
```

# Pointeurs et Tableaux:

## Exemple

- *N.B.*
- L'opérateur primaire [ ] à une priorité plus élevée que celle de l'opérateur unaire &.
- La liaison pour [ ] se fait de la gauche vers la droite.
- Ainsi :
  - $\&a[0] \leftrightarrow \&(a[0])$

# Pointeurs et Tableaux

- Si  $pa$  pointe sur le  $n$ ème élément d'un tableau, la notation  $pa + i$  permet de pointer sur le  $(n + i)$ ème élément.
- Exemples:
- Si  $pa = \&a[0]$  alors :
  - $pa + 1 \leftrightarrow \&a[0] + 1 \leftrightarrow \&a[0 + 1] \leftrightarrow \&a[1]$
- Si  $pa = \&a[3]$  alors :
  - $pa + 4 \leftrightarrow \&a[3] + 4 \leftrightarrow \&a[3 + 4] \leftrightarrow \&a[7]$

# Pointeurs et Tableaux

- Plus généralement :
- Si  $pa = \&a[n]$  alors :
  - $pa + i \leftrightarrow \&a[n] + i \leftrightarrow \&a[n + i]$ ,
- La notation  $*(pa + i)$  est la valeur déréférencée du pointeur  $pa + i$  .
- Si  $pa = \&a[n]$  alors:
  - $*(pa + i) \leftrightarrow a[n + i]$ ,

# Pointeurs et Tableaux

- Exemples :

- Si  $pa = \&a[0]$  alors :

$*(pa + 1) \leftrightarrow *(\&a[0] + 1) \leftrightarrow *(\&a[0 + 1]) \leftrightarrow *(\&a[1]) \leftrightarrow a[1]$

- Si  $pa = \&a[3]$  alors :

$*(pa + 4) \leftrightarrow *(\&a[3] + 4) \leftrightarrow *(\&a[3 + 4]) \leftrightarrow *(\&a[7]) \leftrightarrow a[7]$



# Pointeurs et Tableaux

- Plus généralement :
- Si  $pa = \&a[n]$  alors :  
$$*(pa + i) \leftrightarrow *(\&a[n] + i) \leftrightarrow *(\&a[n + i]) \leftrightarrow a[n + i]$$
- Les parenthèses dans l'expression  $*(pa + i)$  sont nécessaires pour l'accès au  $(n + i)^{\text{ème}}$  élément, si  $pa = \&a[n]$ .

# Pointeurs et Tableaux:

## Exemple

```
int x, y;
int a [10];/* définition d'un tableau de 10 entiers */
int * pa;/* définition d'un pointeur d'entier */
int * pb;/* définition d'un pointeur d'entier */
a[0] = 4;/* affecte la valeur 4 au 1er élément */
a[3] = 2;/* affecte la valeur 2 au 4ème élément */
a[7] = 11;/* affecte la valeur 11 au 8ème élément */
pa = &a[0];/* pa pointe sur le 1er élément du
tableau. pa contient l'adresse du 1er élément */
```

# Pointeurs et Tableau:

## Exemple (suite)

`x = *pa; /* copie la valeur du 1er élément du tableau dans la variable x. Alors x vaut 4. */`

`pb = pa + 3; /* pb = &a[3]*/`

`x = *pb; /* x vaut 2 */`

`y = *(pa + 3); /* y vaut 2 */`

`x = *(pb + 4); /* x vaut 11 */`

`x = *pb + 4; /* x vaut 2 + 4 */`

### ■ *N.B.*

- En C, le nom d'un tableau est synonyme de la location mémoire de son premier élément :

$\&a[0] \leftrightarrow a$  d'où :  $pa = \&a[0] \leftrightarrow pa = a$



# BONNE CHANCE