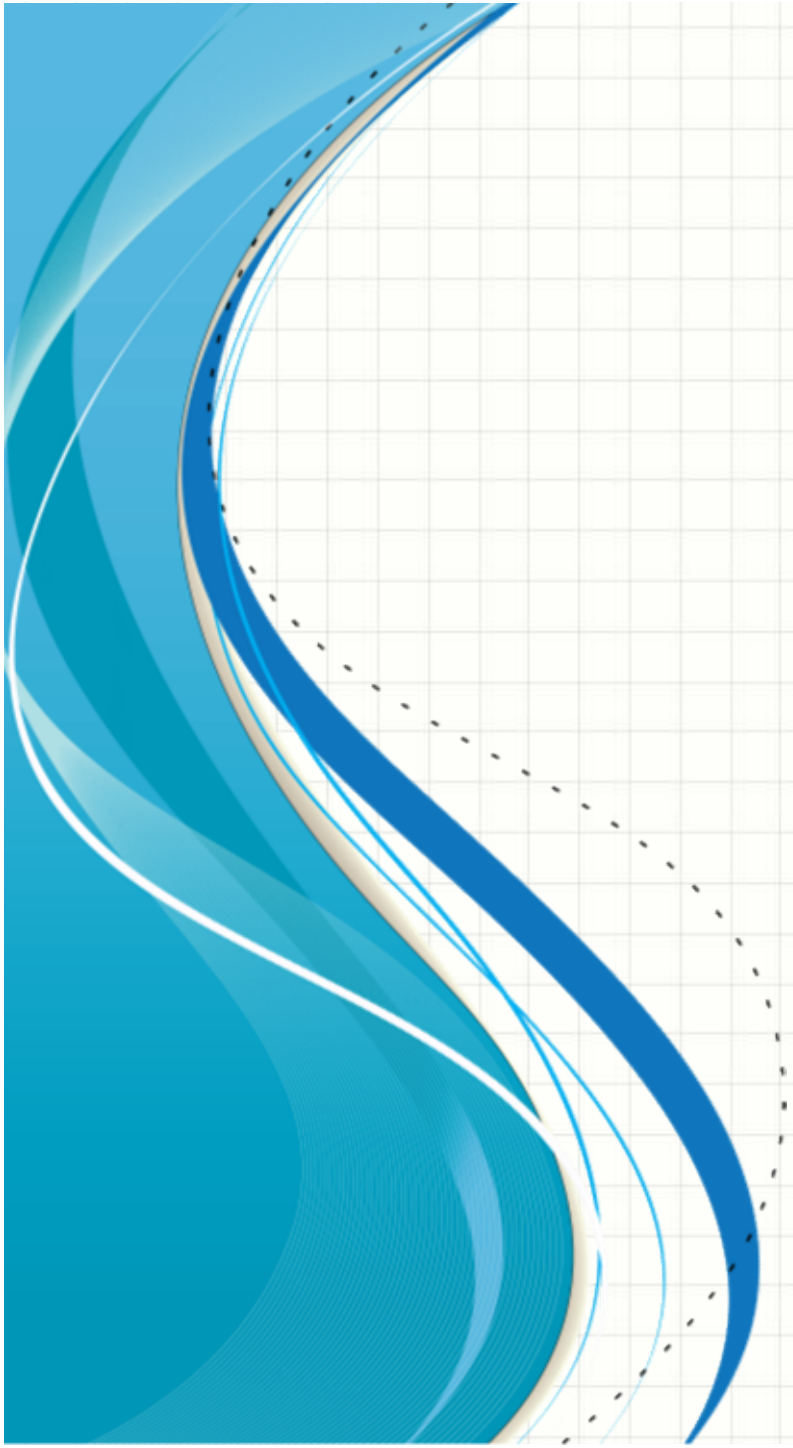


ALGORITHMIQUE AVANCÉE ET PROGRAMMATION

CHAP 3: LES POINTEURS

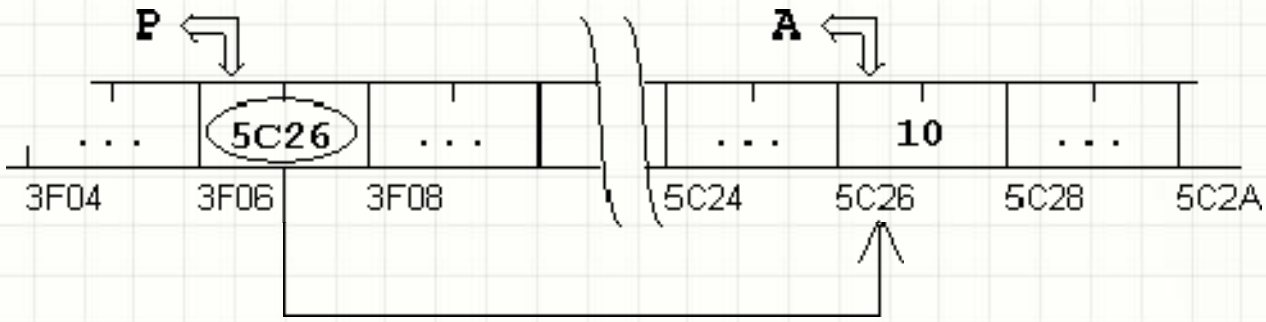
Dr. Ikbal Chammakhi Msadaa

Introduction

- Les **pointeurs** représentent un mécanisme permettant de manipuler les adresses.
- Un pointeur a pour valeur l'adresse d'un objet C d'un type donné (un pointeur est typé).
- Ainsi, un pointeur contenant l'adresse d'un entier sera de type *pointeur vers entier*.

Introduction

- Exemple
  - Soit A une variable contenant la valeur 10 et P un pointeur qui contient l'adresse de A. En mémoire, A et P peuvent se présenter comme suit:



Déclaration

- La manipulation des pointeurs se fait à travers:
  - L'opérateur **'adresse de': &** pour obtenir l'adresse d'une variable.
  - L'opérateur **'contenu de': *** pour accéder au contenu d'une adresse.
  - une syntaxe de déclaration pour pouvoir déclarer un pointeur. **type *nom-du-pointeur;**

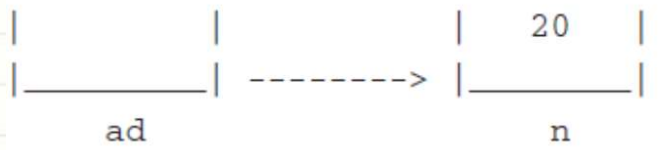
Déclaration

- Considérons les instructions suivantes:

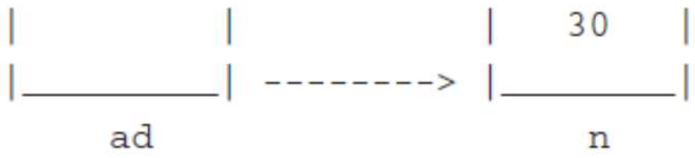
```
int * ad ;
int n ;
n = 20 ;
ad = &n ;
*ad = 30 ;
```

ad est déclaré comme un pointeur sur des entiers

L'adresse de l'entier n est placée dans la variable ad



La valeur 30 est affectée à l'entier (*ad) ayant pour adresse ad. Nous aurions eu le même résultat avec: n = 30 ;



Quelques exemples

```
int * ad1, * ad2, * ad ;
int n = 10, p = 20 ;
```

- ad, ad1 et ad2 sont des pointeurs sur des entiers.

```
int * ad1, ad2, ad ;
```

- Ici seul ad1 est un pointeur. ad et ad2 sont 2 entiers.

```
ad1 = &n ;
ad2 = &p ;
* ad1 = * ad2 + 2 ;
```

- Ad1 et ad2 contiennent les adresses de n et p.

```
* ad2 + 2
```
⇔
```
n = p + 2 ;
```

Quelques exemples

```
main()
{
  int i = 3, j = 6;
  int *p1, *p2;
  p1 = &i;
  p2 = &j;
  *p1 = *p2;
}
```

objet	adresse	valeur
i	4831836000	3
j	4831836004	6
p1	4831835984	4831836000
p2	4831835992	4831836004

objet	adresse	valeur
i	4831836000	6
j	4831836004	6
p1	4831835984	4831836000
p2	4831835992	4831836004

```
main()
{
  int i = 3, j = 6;
  int *p1, *p2;
  p1 = &i;
  p2 = &j;
  p1 = p2;
}
```

objet	adresse	valeur
i	4831836000	3
j	4831836004	6
p1	4831835984	4831836004
p2	4831835992	4831836004

Arithmétique des pointeurs

- La valeur d'un pointeur étant un entier, on peut lui appliquer un certain nombre d'opérateurs arithmétiques classiques. Les seules opérations arithmétiques valides sur les pointeurs sont :
  - **l'addition d'un entier à un pointeur**. Le résultat est un pointeur de même type que le pointeur de départ ;
  - **la soustraction d'un entier à un pointeur**. Le résultat est un pointeur de même type que le pointeur de départ ;
  - **la différence de deux pointeurs pointant tous deux vers des objets de même type**. Le résultat est un entier.
    - Exemple: Si p et q sont deux pointeurs sur des objets de type *type*, l'expression p - q désigne un entier dont la valeur est égale à :

      (valeur de p – valeur de q)/sizeof(*type*)
- Notons que **la somme de deux pointeurs n'est pas autorisée**.

Arithmétique des pointeurs

- Si i est un entier et p est un pointeur sur un objet de type type,
  - l'expression p + i désigne un pointeur sur un objet de type type dont la valeur est égale à la valeur de p incrémentée de:

    i * sizeof(type).
  - Il en va de même pour la soustraction d'un entier à un pointeur,
  - et pour les opérateurs d'incrémentation et de décrémentation ++ et --

```
main()
{
  int i = 3;
  int *p1, *p2;
  p1 = &i;
  p2 = p1 + 1;
  printf("p1 = %ld \t p2 = %ld\n",p1,p2);
}
```

p1 = 4831835**984** p2 = 4831835**988**

```
main()
{
  double i = 3;
  double *p1, *p2;
  p1 = &i;
  p2 = p1 + 1;
  printf("p1 = %ld \t p2 = %ld\n",p1,p2);
}
```

p1 = 4831835**984** p2 = 4831835**992**

Arithmétique des pointeurs

- **Les opérateurs de comparaison** sont également applicables aux pointeurs, à condition de comparer des pointeurs qui pointent vers **des objets de même type**.
- L'utilisation des opérations arithmétiques sur les pointeurs est particulièrement utile pour parcourir des tableaux.

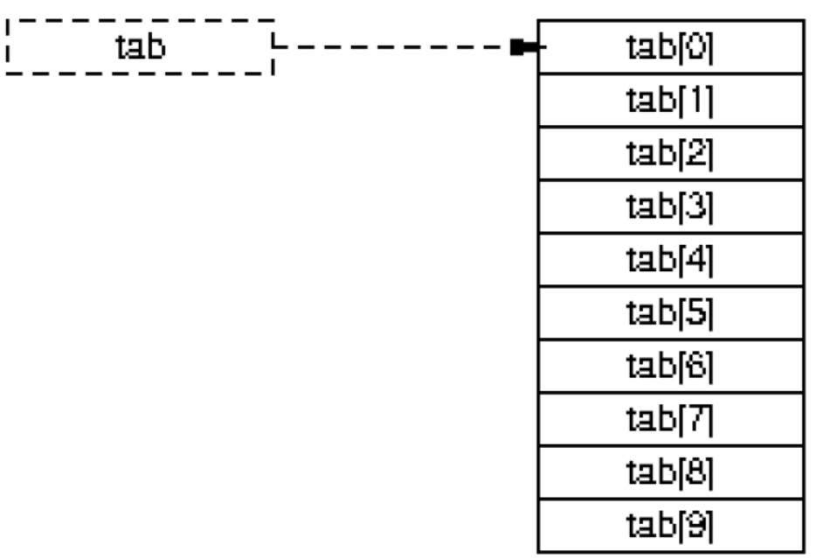
```
#define N 5
int tab[5] = {1, 2, 6, 0, 7};
main()
{
  int *p;
  printf("\n ordre croissant:\n");
  for (p = &tab[0]; p <= &tab[N-1]; p++)
    printf(" %d \n",*p);
  printf("\n ordre decroissant:\n");
  for (p = &tab[N-1]; p >= &tab[0]; p--)
    printf(" %d \n",*p);
}
```

Pointeurs et tableaux

- Tout tableau en C est en fait un pointeur constant. Dans la déclaration:

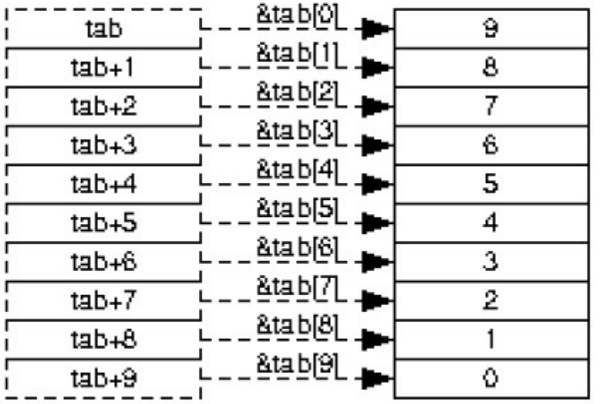
  int tab[10];

- tab est un pointeur constant (non modifiable) dont la valeur est l'adresse du premier élément du tableau.
- Autrement dit, **tab** a pour valeur **&tab[0]**. On peut donc utiliser un pointeur initialisé à tab pour parcourir les éléments du tableau.



Pointeurs et tableaux

- Quelques exemples de notations équivalentes:
  - tab+1 ⇔ &tab[1]
  - tab+i ⇔ &tab[i]
  - tab[i] ⇔ *(tab+i)



- Exemple: pour initialiser les valeurs d'un tableau de 10 entiers à 1:

```
int i ;
for (i=0; i<10; i++)
    *(tab+i) = 1;
```

ou

```
int *p;
for (p=tab, p<tab+10; p++)
        *p = 1;
```

Pointeurs et tableaux

- Un tableau à deux dimensions est, par définition, un tableau de tableaux. Il s'agit donc en fait d'un pointeur vers un pointeur.
- Considérons le tableau à deux dimensions défini par : int tab[M][N];
- tab est un pointeur, qui pointe vers un objet lui-même de type pointeur d'entier. tab a une valeur constante égale à l'adresse du premier élément du tableau, &tab[0][0].
- De même tab[i], pour i entre 0 et M-1, est un pointeur constant vers un objet de type entier, qui est le premier élément de la ligne d'indice i.
- tab[i] a donc une valeur constante qui est égale à &tab[i][0].
- On déclare un pointeur qui pointe sur un objet de type **type *** (deux dimensions) de la même manière qu'un pointeur, c'est-`a-dire

type **nom-du-pointeur;

Pointeurs et tableaux

Exemple 1:

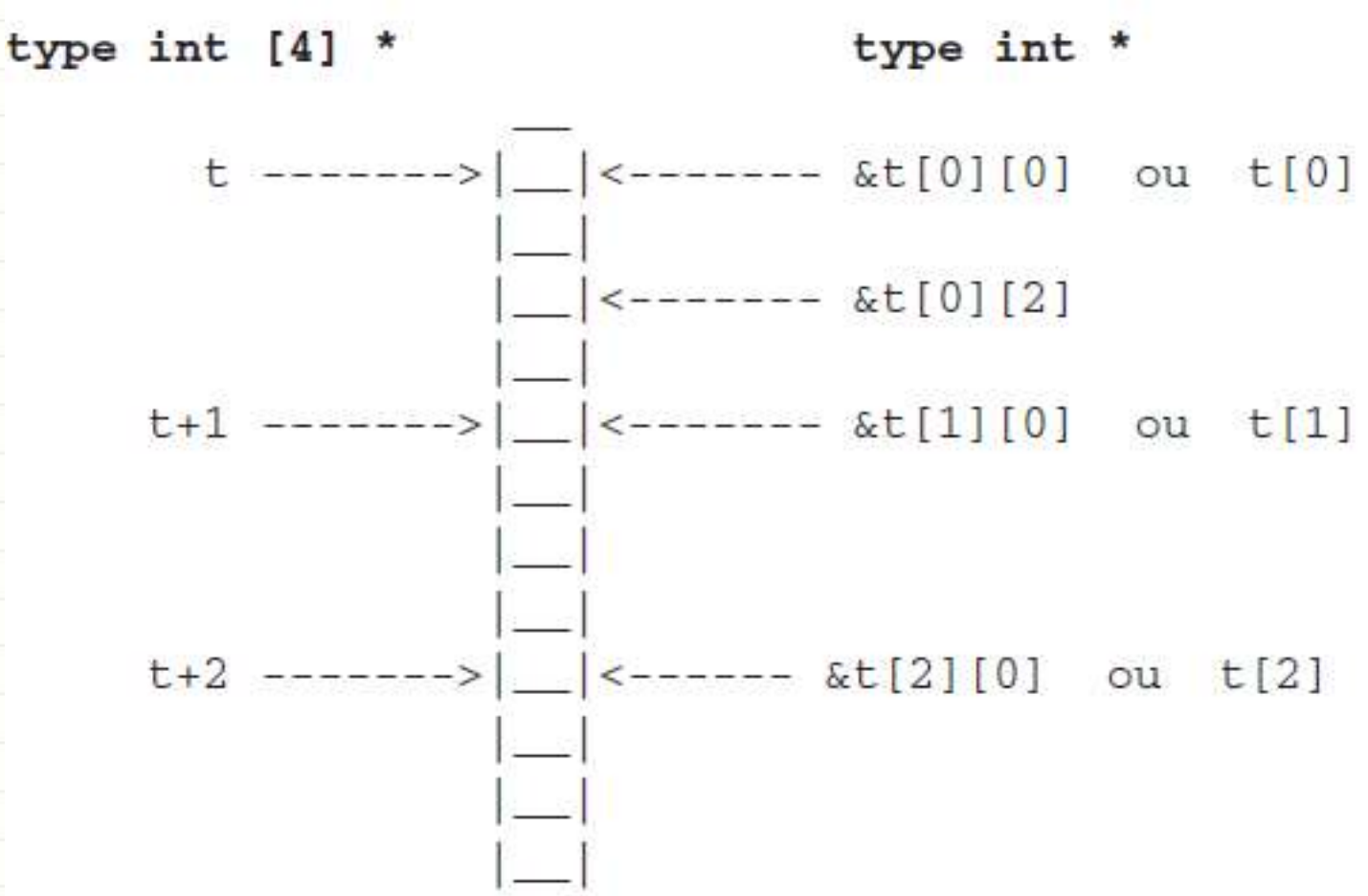
- int t[3][4];
- t est l'adresse de début d'un tableau t. t n'est plus de type int *. t est de type pointeur sur un bloc de 4 entiers:

  int [4] *

- Dans ces conditions, une expression telle que t+1 correspond à l'adresse de t, augmentée de 4 entiers (et non plus d'un seul !). Ainsi, les notations t et &t[0][0] correspondent toujours à la même adresse, mais l'incrémentation de 1 n'a pas la même signification pour les deux.
- t[0], t[1] ou t[i] ont un sens. t[0] représente l'adresse du 1er bloc de 4 entiers de t. t[1] celle du second bloc...
- Ces notations sont donc équivalentes:

```
t[0]    &t[0][0]
t[1]    &t[1][0]
```

Pointeurs et tableaux



Pointeurs et tableaux

- Déclaration:

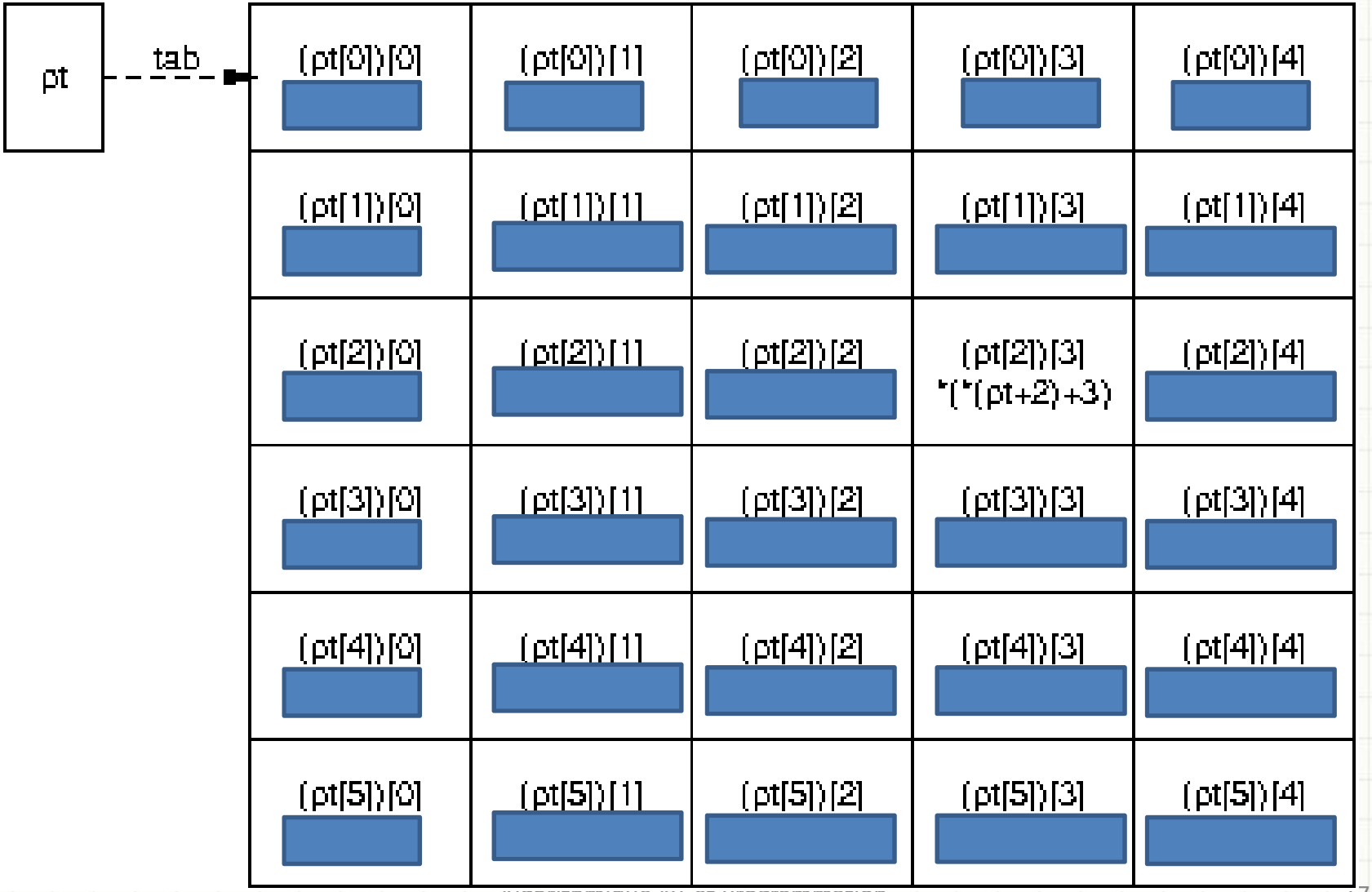
<Type> *<NomTableau>[<N>]

déclare un tableau <NomTableau> de <N> pointeurs sur des données du type <Type>

– Dans l'exemple précédent:

int t[3][4] ⇔ int *t[4];

Pointeurs et tableaux



Pointeurs et tableaux