الكلية متعددة التخصصات - ورزازات

ⵜⴰⵖⵡⴰⵙⵜ ⵜⴰⵎⴰⵣⵉⵖⵜ – ⵡⵓⵔⵣⴰⵣⴰⵜ

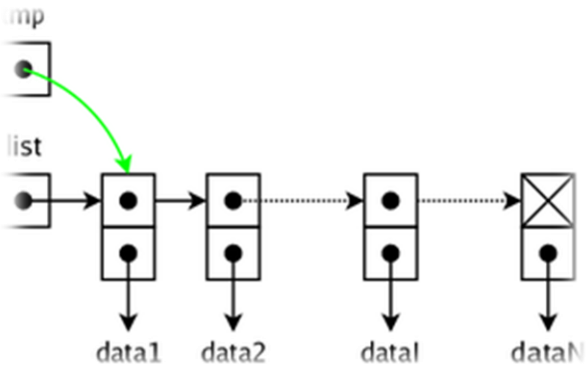
FACULTÉ POLYDISCIPLINAIRE DE OUARZAZATE

Sciences Mathématiques et informatique

Semestre 4

Support de cours

M22 : Structures de données en langage C



Professeur : Abdelhadi Bouain

Table de matière

1 Rappel

1.1 Un exemple de programme en langage C

Ce programme permet de lire deux nombres réels et de réaliser leur somme, soustraction ou multiplication selon le choix de l'utilisateur.

```c
#include <stdio.h>
main()
{
    /* déclaration des variables */
    float  x, y, resultat;
    char op;

    printf("Entrez le premier nombre : \n");
    scanf("%f", &x);           /* Entrer le premier nombre. */
    printf("Entrez le deuxième nombre : \n");
    scanf("%f", &y);           /* Entrer le deuxième nombre. */
    printf("Entrez l'opération + , - ou  *  : \n");
    op = getchar () ;          /* Entrer l'opération */

    if (op == '+')   /* vérifier si l'opération est une addition*/
                {
                            resultat = x+y;
                            printf ("la somme des deux nombres: %f ", resultat) ;
                }
         else if (op == '-') /* vérifier si l'opération est une soustraction*/
                {
                            resultat = x-y;
                            printf ("la soustraction des deux nombres: %f ", resultat) ;
                }
              else  /* Si c'est pas une addition ou soustraction alors réaliser la multiplication*/
                  {
                           resultat = x*y;
                            printf ("la multiplication des deux nombres: %f ", resultat) ;
                  }
}
```

1.2 Un deuxième exemple de programme en langage C

Ce programme permet de chercher la plus grande valeur (max) dans un tableau.

```c
#include <stdio.h>
main()
    {
      // Déclaration des variables
        int i, nbr, tab[100], max;

      /* Définition du nombre d'éléments à insérer dans le tableau*/
        printf("Entrez le nombre total d'éléments: ");
        scanf("%d", &nbr);

        // Stocker les nombres saisis par l'utilisateur
        for(i = 0; i < nbr; ++i)
        {
           printf("Entrer le nombre %d: ", i+1);
           scanf("%d", &tab[i]);
        }

        //considérer le premier élément comme max
        max = tab[0];

        // Boucle pour chercher le max dans le reste du tableau
        for(i = 1; i < nbr; i++)
        {
           if(max < tab[i])
                      {
                           max = tab[i];
                      }
        }

        printf("Le plus grand élément est %d", max);
     }
```

2 - Pointeurs et allocation dynamique de la mémoire.

2-1- Introduction.

Toute variable dans un programme occupe un espace dans la mémoire centrale (La RAM). La taille de l'espace mémoire qu'occupe une variable dépend de son type.

Le tableau suivant résume la taille de chaque type de variable en C :

Type de donnée	Signification	Taille (en octets)
Char	***Caractère***	***1***
unsigned char	Caractère non signé	1
short int	Entier court	2
unsigned short int	Entier court non signé	2
Int	***Entier***	*2 ou **4***
unsigned int	Entier non signé	2 ou 4
long int	*Entier long*	*4*
unsigned long int	Entier long non signé	4
Float	***Flottant (réel)***	***4***
Double	***Flottant double***	***8***
long double	Flottant double long	10

Remarque :

Pour identifier la taille d'une variable, vous pouvez utiliser l'opérateur **sizeof()** qui fournit la taille d'une variable en octets.

● **Exemple**:

```
main()
{
	int a;
	long int b;
	float c;
	long double d;

	printf("la taille d'un entier : %d  \n", sizeof(a));
	printf("la taille d'un long entier : %d  \n", sizeof(b));
	printf("la taille d'un réel : %d  \n", sizeof(c));
	printf("la taille d'un long réel : %d  \n", sizeof(d));

}
```

● **Résultat**

```
la taille d'un entier : 4
la taille d'un long entier : 4
la taille d'un reel : 4
la taille d'un long reel : 12
```

Vous pouvez aussi utiliser directement **sizeof** avec un type, par exemple :

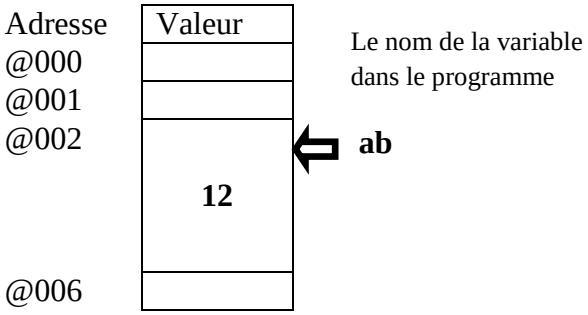
- **sizeof(int)** vaudra 4,
- **sizeof(double)** vaudra 8.

2-2 Adressage direct

Chaque variable a un ***type***, un ***identificateur (nom), une adresse*** et ***une valeur***.

● Exemple:

```
int ab ;
ab= 12;
```



La mémoire centrale

Ce programme permet de créer une variable **de type int** (lui réserver 4 octets dans la mémoire), cette variable est **nommée ab**. **La valeur** de ab est 12.

```
printf("% d", ab) ;
```

Permet d'afficher la valeur de la variable **ab**.

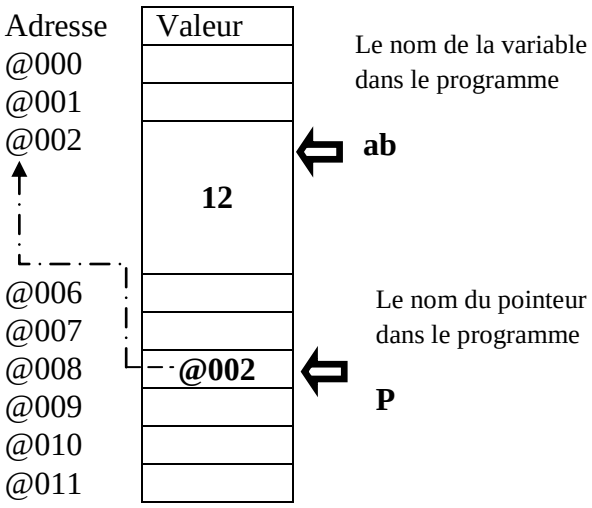
- Le principe de l'adressage direct est de manipuler une variable par son nom.

L'adressage direct *: L'*accès au contenu d'une variable par **le nom de la variable**.

2-3 Adressage indirect

Le principe de ***l'adressage indirect*** est de manipuler une variable **par un pointeur qui** contient l'adresse de cette variable.

●Exemple:



La mémoire centrale

Au lieu d'utiliser le nom de la variable ab en utilisera le pointeur P pour accéder à son contenu. La déclaration et l'utilisation des pointeurs sont traitées dans la section suivante.

2-4 Notion de pointeur.

Un pointeur est une variable qui peut contenir l'adresse mémoire d'une autre variable.

2-5 Déclaration d'un pointeur.

```
int * P ;
```

Déclarer une variable nommée P comme étant un pointeur sur des entiers.
Le pointeur P contiendra uniquement l'adresse d'un entier.

2-6 Affecter une adresse a un pointeur.

```
P = &ab ;
```

L'opérateur **&**ab représente l'adresse de ab. Après cette instruction P contiendra l'adresse de la variable ab, on dit que **P pointe sur ab.**

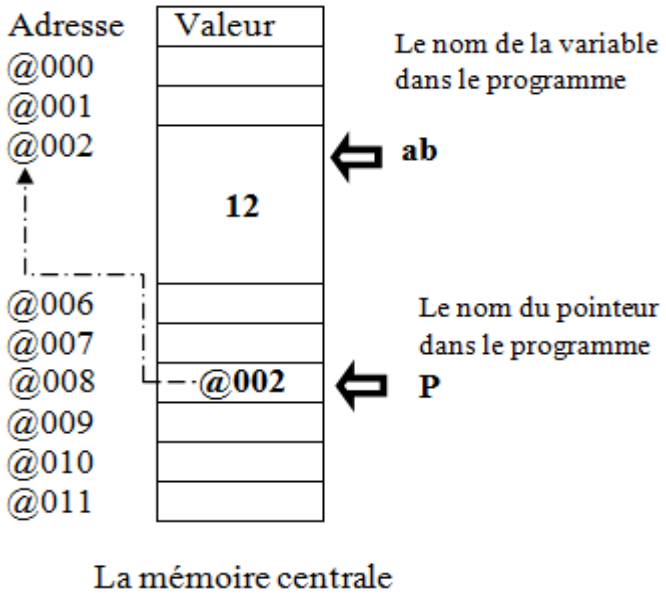
Un pointeur **doit toujours être initialisé** par une adresse ou bien une valeur NULL.

```
P = NULL ;
```

Par convention, un pointeur de valeur NULL ne pointe sur rien.

● **Exercice:** Créer un pointeur P sur une variable de type entier.

```
int ab ;
ab= 12 ;
int *P ;
P=&ab ;
```



2-7 Modifier la valeur d'une variable en utilisant un pointeur.

```
int ab ;
ab= 12 ;
int *P ;
P=&ab ;
*P = 86 ;
```

L'opérateur ***** permet d'accéder directement à la valeur de l'objet pointé. Ainsi, si P est un pointeur vers un entier ab, *P désigne la valeur de ab.

- **P** est équivalent à **&ab.**
- ***P** est équivalent à **ab.**

2-8 Quelques exemples.

❖ **Exemple 1** : Afficher l'adresse d'une variable.

```
int  a;
float b ;
printf("L'adresse de  a:   %x   \n",   &a  );
 printf("L'adresse de b:   %x  \n",    &b  );
```

❖ **Résultat**

```
L'adresse de  a:   28ff44
L'adresse de  b:   28ff40
```

❖ **Exemple 2** : Déclarer des pointeurs sur différents types.

```
int    *ip;            / * Pointeur sur un entier * /
double *dp;                    /* Pointeur sur un double */
float  *fp;                    /* Pointeur sur un réel*/
char   *ch                     /* Pointeur sur un caractère */
```

❖ **Exemple 3** : Déclarer un pointeur et afficher la valeur de la variable sur laquelle il pointe.

```
int a = 3;
int *p;
p = &a;
printf("Le contenu pointé par P est  = %d \n", *p);
```

Résultat :

```
Le contenu pointé par P est : 3
```

❖ **Exemple 4 :**

```
int *P, X ;
P = &X;
```

Les expressions suivantes sont équivalentes:

P	⇔	**&X**
***P**	⇔	**X**
Y = *P+1	⇔	**Y = X+1**
***P = *P+10**	⇔	**X = X+10**
***P += 2**	⇔	**X += 2**
++*P	⇔	**++X**
(*P)++	⇔	**X++**

Remarque :

- L'opérateur d'indirection * et d'adresse & ont une priorité **plus élevée** par rapport à l'addition, soustraction, multiplication et division.
- L'opérateur d'indirection * et d'adresse & ont une priorité **moins élevée** par rapport à la post-incrémentation ++ et la post-décrémentation –

❖ **Exemple 5 :** Soit les deux programmes suivants

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

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

Supposons qu'avant la dernière affectation de chacun de ces programmes, on est dans cette configuration :

objet	Adresse	Valeur
I	@1000	3
J	@1004	6
p1	@8000	@1000
p2	@9000	@1004

➢ Après l'exécution de la dernière instruction de chacun des deux programmes.

Après programme 1

objet	adresse	valeur
I	@1000	6
J	@1004	6
p1	@8000	@1000
p2	@9000	@1004

Après programme 2

objet	adresse	valeur
i	@1000	3
j	@1004	6
p1	@8000	@1004
p2	@9000	@1004

❖ **Exemple 6 :** Quelle est la valeur de a, b et c après exécution du programme suivant ?

```
main(){
    int a=1,b=2,c=3;
    int *P1,*P2;
    P1=&a;
    P2=&c;
    *P1=(*P2)++;
    P1=P2;
    P2=&b;
    *P1 -= *P2;
    ++*P2;
    P1 = &b;
    *P1 *= *P2;
    printf("A : %d, B: %d, C: %d", a, b, c);
}
```

Exemple d'exécution :

	A	B	C	P1	P2
int a=1, b=2, c=3;	1	2	3		
P1=&a				@A	
P2=&c					@C
*P1=(*P2)++	3		4		
P1=P2				@C	
P2=&b					@B
*P1 -= *P2			2		
++*P2		3			
P1 = &b				@B	
*P1 *= *P2		9			
	3	9	2	@B	@B

2-9 Arithmétique des pointeurs

Il est possible d'effectuer des opérations arithmétiques sur les pointeurs.

Les seules opérations valides sont les opérations d'addition et soustraction des entiers et la soustraction de pointeurs.

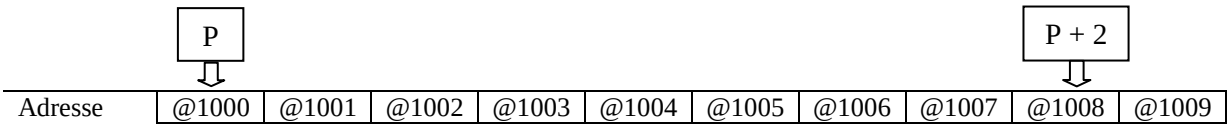
Elles sont définies comme suit :

> P + i = **adresse contenue dans** P + i * taille (élément pointé par p)

> P2 - P1 = (adresse contenue dans P2 - adresse contenue dans P1) / taille (éléments pointés par P1 et P2)

Exemple :

Si P est un pointeur sur les entiers et l'adresse contenue dans ce pointeur est 1000. Alors, P+2 est égal à l'adresse de P + 2 * taille d'un entier = 1000 + 2*4 = 10008.



2-10 Tableau simple et pointeurs.

En langage C, **l'identificateur (le nom) d'un tableau,** lorsqu'il est employé seul (sans indices à sa suite), est considéré comme **un pointeur (constant) sur le début du tableau.**

Dans la déclaration suivante :

```
int  t [10] ;
```

La notation **t** est alors totalement équivalente à **&t[0].**

Voici quelques exemples de notations équivalentes :

```
t+1                    &t [1]
t+i                    &t[i]
t[i]                   * (t+i)
```

- **Exemple 1 :** Pour remplir un tableau de 10 éléments par 1.

Méthode 1 : (sans utilisation de pointeur)

```
int i ;
for (i=0 ; i<10 ; i++)
t[i]  = 1 ;
```

Méthode 2 : (utilisation du nom du tableau comme pointeur)

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

Attention !!

Un nom de tableau est un pointeur constant. Autrement dit, dit la valeur de **t** ne doit pas changer. Une expression telle que **t= t+1** n'est pas valide.

- **Exemple 2 :**

```
int A[10];
int *P;
P=A ;
```

L'instruction:

P = A; est équivalente à **P = &A[0];**

Si P pointe sur une composante quelconque d'un tableau, alors P+1 pointe sur la composante suivante. Plus généralement,

*(P+1) désigne le contenu de A [1]

*(P+2) désigne le contenu de A [2]

❖ Le programme suivant permet de remplir un tableau de 10 éléments par 1, en utilisant un pointeur.

```
int i ;
int A[10];
int *P;
P=A ;
for ( i=0 ; i<10 ; i++)
* (P+i) = 1 ;
```

Ou bien

```
int i ;
int A[10];
int *P;
P=A ;
for ( i=0 ; i<10 ; i++ , P++)
* P = 1 ;
```

❖ **Exercice**

Soit P un pointeur qui 'pointe' sur un tableau A:

```
int A[] = {12, 23, 34, 45, 56, 67, 78, 89, 90};
int *P;
P = A;
```

Quelles valeurs ou adresses fournissent ces expressions:

***P+2**	14
***(P+2)**	34
&P+1	&P l'adresse du pointeur P dans la mémoire. &P + 1 l'adresse du pointeur suivant « à éviter - rarement utilisée »
&A[4]-3	&A[1]
A+3	&A[3]
&A[7]-P	7
P+(*P-10)	&A[2]
***(P+*(P+8)-A[7])**	23

2-11 Tableaux à deux dimensions et pointeurs.

- Déclaration d'un tableau à deux dimensions : (tableau de 3 lignes, 4 colonnes)

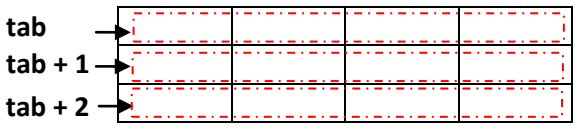
```
int tab[3] [4] ;
```

L'identificateur (le nom) d'un tableau, employé seul, représente l'adresse du début du tableau. Cependant, **il n'est plus du type int* (pointeur sur un entier) mais un pointeur sur un bloc de 4 entiers. Il s'agit donc en fait d'un pointeur vers un pointeur.**

Exemple :

```
int tab[3] [4] =  {     {1,2,3,4 },
                        {5,6,7,8 },
                        {9,10,11,12}
                  } ;
```

- Le nom du tableau « **tab** » représente **l'adresse du premier bloc** qui a la valeur {1, 2, 3,4}.
- Une expression telle que **tab +1** correspond à l'adresse de tab, **augmentée de 4 entiers** (et non plus d'un seul !), ainsi **tab + 1** représente l'**adresse du deuxième bloc qui a la valeur {5, 6, 7,8}.**



Voici quelques exemples d'adresses équivalentes :

Type : int **	Type : int*	
tab	tab[0]	&tab[0][0]
tab+1	tab[1]	&tab[1][0]
tab+i	tab[i]	&tab[i][0]

- Les notations **tab et &tab[0][0]** correspondent toujours à **la même adresse** , mais **l'incrémentation de 1 n'a pas la même signification pour les deux.**
- **tab est un pointeur**, qui pointe vers un objet lui-même de type pointeur d'entiers (tab [0]).
- **tab[0]** représente l'adresse (pointeur) de début du premier bloc.
- **tab[1]** représente l'adresse de début du deuxième bloc.

Comment utiliser un pointeur de type int * pour parcourir un tableau à deux dimensions (int **) ?

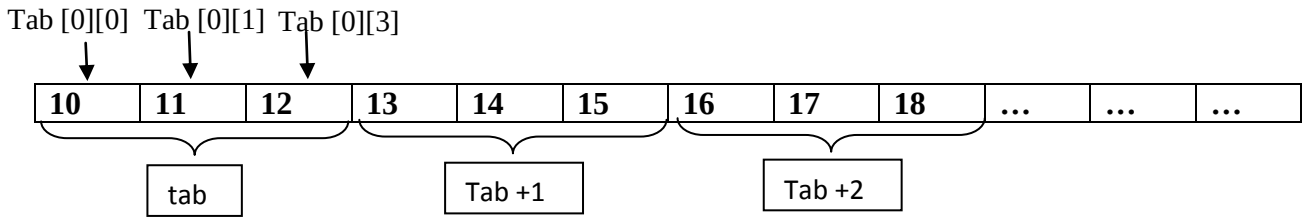
- Un **tableau 2 dimension en mémoire est un tableau unidimensionnel** dont chaque composante est un tableau unidimensionnel.

❖ **Exemple :**

- **Soit le tableau suivant « tab ».**

10	**11**	**12**
13	**14**	**15**
16	**17**	**18**

- **En mémoire le tableau est représenté comme suite :**



- **Pour utiliser un pointeur simple afin de parcourir le tableau à 2 dimensions :**

1. Convertir tab qui est de type int ** à int *.

Solution :

```
int tab[3] [3] ;
int  *P ;
P = (int *) tab ;
```

2. Trouver l'adresse de chaque élément en utilisant la formule suivante :

`&tab[i][j] = P+ i*TC +j`

i : indice de la ligne.

J : indice de la colonne.

TC : **T**aille réservée pour les **c**olonnes (nombre de colonnes déclaré)

P : Pointeur sur le début du tableau P= (int *) tab.

Exemple :

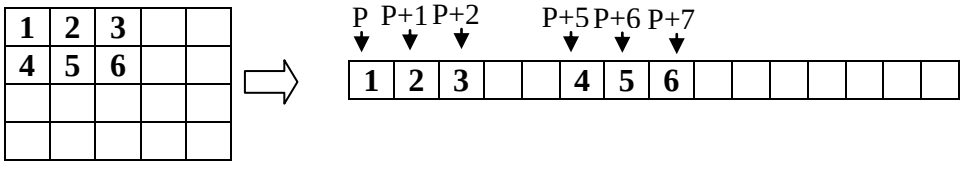
Si on déclare un tableau M de 4 lignes 5 colonnes.

```
int M [4][5] ;
```

TC = 5 ;

Attention : Si dans le programme nous avons utilisé uniquement 2 lignes, 3 colonnes, **TC** est = 5 (**réservé au début du programme).**

Exemple :



&M[i][j] = P+i*5+j

&M[**1**][**2**] = P+**1***5+**2** = P+ 7

&M[0][1] = P+0*5+1 = P+1

❖ **Exercice**

Soit le programme suivant :

```
int tab[3] [4] =  {      {10,20,30,40 },
                         {50,60, 70,80},
                         {90, 100, 110,120}
                  } ;
int * P ;
P=  (int*) tab ;
```

Quelles valeurs fournissent ces expressions ?

***P**	10
***(P+10)**	110
***P+5**	15
***tab[1]**	50
***(tab[2]+2)**	110

❖ **Exercice**

- Écrire un programme qui affiche le contenu d'un tableau 2D en utilisant les pointeurs.

Solution :

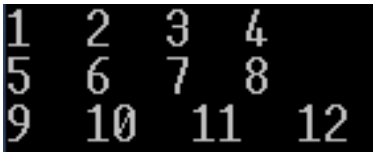
```
#include <stdio.h>
main()
{
int tab[3] [4] =  {      {1,2,3,4 },
                         {5, 6,7,8 },
                         {9,10,11,12}
                   };

int* P,i,j ;

P= (int*)tab ;

for(i=0; i<3 ; i++)
  {
    for(j=0; j<4; j++)
      {
        printf(" %d ", *(P+i*4+j));
      }
      printf("\n");
  }
}
```

- **Résultat :**



D'autres méthodes peuvent être utilisées pour parcourir un tableau en utilisant les pointeurs. Un exemple est présenté ci-dessous.

- **Méthode 2 :**

```
#include <stdio.h>

main()
{
int tab[3] [4] =  {      {1,2,3,4 },
                         {5,6,7,8 },
                         {9,10,11,12}
                  };

int * P,i,j ;


for (i=0; i<3 ; i++)
  {
    P= tab[i] ;
    for (j=0; j<4 ; j++, P++)
              {
                printf("%d  ", *P);
              }
    printf("\n");
 }

 system("pause");
}
```

2-12 Tableaux et pointeurs - Résumé

Variable – valeur – Adresse	
int A	déclare une variable simple du type int.
A	désigne le contenu de A.
&A	désigne l'adresse de A.
Tableau simple et utilisation du nom du tableau comme pointeur	
int B[]	déclare un tableau d'éléments du type int.
B	désigne l'adresse de la première composante de B. ((cette adresse est toujours constante).
B[i]	désigne le contenu de la composante i du tableau.
***(B+i)**	désigne le contenu de la composante i du tableau

Utilisation de pointeur avec un tableau simple	
int *P	déclare un pointeur sur des éléments du type **int**.
P	désigne l'adresse contenue dans P
***P**	désigne le contenu de l'adresse dans P
P=B	P pointe sur le tableau B
P	désigne l'adresse de la première composante
P+i	désigne l'adresse de la i-ème composante derrière P
***(P+i)**	désigne le contenu de la i-ème composante derrière P
Utilisation de pointeur avec un tableau 2 Dimensions	
int A[3][4]	Déclare un tableau 2 dimensions (3 lignes, 4 colonnes)
int *P = (int*)A	Déclarer un pointeur P et lui affecter l'adresse du tableau 2D. « Convertir A qui est de type int ** à int *)
&A[I][J]	P+I***4**+J « Formule (P+I***TC**+J) »
A[I][J]	*(P+I***4**+J)

2-13 Chaînes de caractères pointeurs.

- Une chaine de caractères est une suite d'octets en mémoire, terminée par un caractère nul ('\0').

'h'	'e'	'l'	'l'	'o'	'\0'

- Pour désigner une chaine de caractères, on donne simplement l'adresse de son premier octet.

❖ **Déclaration d'une chaine de caractère comme un tableau :**

```
char chaine[] = "hello";
char chaine[] = { 'h','e','l','l','o', '\0' };
char B[45] = "Deuxième chaîne un peu plus longue";
 char C[30];
```

Dans cette déclaration la chaine de caractères est un tableau (son nom est un pointeur constant) alors les instructions suivantes sont impossible:

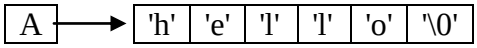
```
A = B;                    /* ERREUR !!! */
C = "Bonjour !";          /* ERREUR !!! */
Chaine = "Abdelhadi "     /* ERREUR !!! */
```

- Une instruction telle que chaine = "Salut"; ne marche que lors de la définition et initialisation

- Pour changer le contenu d'un tableau, nous devons changer les composantes du tableau l'une après l'autre (p.ex. dans une boucle) ou déléguer cette charge à une fonction de <string.h>.

❖ **Déclaration d'une chaine de caractère comme pointeur sur char:**

```
char *A = "Hello";
 char *B = "SMI S4";
char* salut ;
A = B;
salut = "coucou" ;
```



- A **est un pointeur** qui est initialisé de façon à ce qu'il pointe sur une chaîne de caractères constante stockée quelque part en mémoire.
- **Le pointeur peut être modifié** et pointer sur autre chose.

  A=B ;

  A= "Une autre chaine"
- **La chaîne constante** peut être lue, copiée ou affichée, **mais pas modifiée**.

- **Résumé :**

  Une chaîne déclarée comme tableau :

  Char A [20]="Bonjour" ;
  - La taille du tableau est égale au nombre de caractères de la chaine +1 (' \0') .
  - Le nom A doit toujours pointer sur la même adresse.
  - Le contenu de A peut être changé.

  Une chaîne déclarée comme pointeur :

  Char *B = "Bonjour" ;
  - B est un pointeur qui peut pointer sur n'importe quelle **chaîne de caractères constante** stockée en mémoire.
  - **La chaîne constante** peut être lue, copiée ou affichée, **mais pas modifiée**.