

# Initiation à la programmation C

Rachida Chakir

[rachida.chakir@esiee.fr](mailto:rachida.chakir@esiee.fr)

3R-IN1A  
2024-2025

# Programme du cours

- ▶ Partie 1 : Les bases de la programmation en C
- ▶ Partie 2 : Pointeurs et fonctions
- ▶ **Partie 3 : Tableaux**
- ▶ Partie 4 : Allocation mémoire et Structures

# Tableaux

- ▶ Introduction
- ▶ Rappels sur les pointeurs
- ▶ Tableaux à taille fixe
  - ▶ Tableaux unidimensionnel
  - ▶ Tableaux bidimensionnel
- ▶ Chaînes de caractères

# Introduction

## Exercice 1- Moyenne générale d'un étudiant

Ecrire un programme permettant de saisir les notes des examens du 1er semestre d'un étudiant en Licence, puis les afficher avec sa moyenne générale.

## Solution

```
void main ()
{
    int i, nbMatiere = 8;
    float note = 0, somme = 0, moyenne = 0;
    for (i = 0; i < nbMatiere ; i++)
    {
        printf(" saisir la note numero %d : ", i +1 );
        scanf("%d", &note);
        somme += note;
    }
    moyenne += somme/nbMatiere;
    printf(" La moyenne est %d ", moyenne);
}
```

# Introduction

## Exercice 2 - Moyenne générale d'une promo

Ecrire un programme permettant de saisir les moyennes de tous les étudiants en Licence, puis calculer et afficher la moyenne générale de la promo.

## Solution

```
void main ()
{
    int i, nbEtudiant = 450;
    float M = 0, somme = 0, moyenne = 0;
    for (i = 0; i < nbEtudiant ; i++)
    {
        printf(" saisir la moyenne de l 'etudiant numero %d : ";
        scanf("%d", &M);
        somme += M;
    }
    moyenne += somme/nbEtudiant;
    printf(" La moyenne de la promo est %d ", moyenne);
}
```

# Introduction

## Exercice 2 - Etudiant au dessus de la moyenne générale de la promo

Ecrire un programme permettant de saisir et les moyennes de tous les étudiants en Licence, puis déterminer combien d'entre eux sont supérieures à la moyenne de la promo.

## Solution

```
void main ()  
{  
    int nbEtudiant = 450;  
    int NbMoyenSup = 0;  
    float M1, M2, M3, M4, ..., M448, M449, M450;  
    ...  
    moyenne = (M1 + M2 + ... + M449 + M450)/ nbEtudiant;  
    printf("La moyenne de la promo est %d ", moyenne);  
    if (M1 > moyenne) NbMoyenSup ++;  
    ...  
    if (M450 > moyenne) NbMoyenSup ++;  
}
```

Euhhhhhh !!!

Il doit y avoir un moyen plus simple et plus élégant pour écrire ça !!

# Introduction - Notion de tableaux

## Définition

- ▶ Un **tableau** est une variable composée de **données de même type, stockées de manière contiguë en mémoire** (les unes à la suite des autres).
- ▶ Le nombre qui, sert à repérer chaque donnée (ou **élément**) dans le tableau s'appelle **l'indice**

**Attention ! En C, un tableau commence à l'indice numéro 0**

Notre tableau de 9 notes a donc les indices 0, 1, 2,... et 8. Il n'y a pas d'indice 9 dans un tableau de 9 cases !



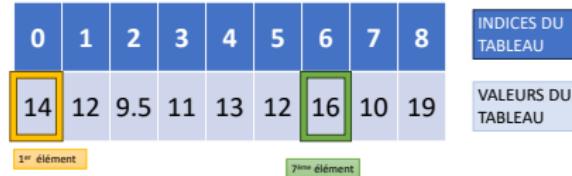
# Introduction - Notion de tableaux

## La définition d'un tableau nécessite trois informations

- ▶ Le type des éléments du tableau (un tableau est une suite de donnée de même type)
- ▶ Le nom du tableau (son identificateur)
- ▶ La taille ou longueur du tableau (c.a.d le nombre d'éléments dans le tableau).

type *identificateur* [taille]

# Introduction - Notion de tableaux



## Déclaration d'un tableau et accès à ses éléments

```
float notes[9]; /* declaration du tableau*/  
note[0] = 14;  
note[1] = 12;  
...  
note[7] = 10;  
note[8] = 19;
```

- ▶ Pour définir un tableau de 9 éléments de type *int* , Il suffit de rajouter le nombre de cases entre des crochets après le nom du tableau.
- ▶ Pour accéder à chaque élément du tableau, il faut écrire le nom du tableau suivi de l'indice de l'élément concerné entre crochets.

# Introduction - Notion de tableaux

- ▶ Le nom d'un tableau est un *pointeur*

## Example

```
1 float notes[9];
2 note[0] = 14;
3 note[1] = 12;
4 ...
5 note[7] = 10;
6 note[8] = 19;
7 printf("%p", notes);
9 printf("%f", notes[0]);
10 printf("%f", *notes);
```

- ligne 7. On affiche l'adresse d'un pointeur
- ligne 8. On affiche la valeur du 1er élément
- ligne 9. On affiche la valeur du 1er élément

La variable *notes* n'est rien d'autre qu'un pointeur sur *float*.

Elle est de type *float\**

# Rappel sur les pointeurs

Lors du travail avec des pointeurs, nous avons besoin de l'opérateur

- \* pour accéder au contenu d'une variable,
- \* pour accéder au contenu d'une adresse,
- & pour obtenir l'adresse d'une variable.

Exemple de déclaration d'un pointeur

```
int* p;
```

- ▶ \*p est de type *int*
- ▶ p est un pointeur sur *int*
- ▶ p peut contenir l'adresse d'une variable de type *int*

# Rappel sur les pointeurs

## □ Déclaration d'un point sur un *int*

```
int * p;
```

## □ Convention d'initialisation avec *NULL*

```
p = NULL;
```

## □ Affectation avec une adresse de variable

```
int a;  
p = & a;
```

## □ Déréférencement d'un pointeur

```
int b;  
b = *p;
```

# Rappel sur les pointeurs

## ❑ Exemple basique

```
int * p;  
int a = 4;
```



```
p = &a;  
int b;  
b = *p
```

*p* « pointe » sur *a* et *&a* est l'adresse de *a*

\**p* : contenu de la variable « pointée » par *p* ou « déréférencement » de *p*



# Rappel sur les pointeurs

```
float a = 0.5 ;  
int *p = &a;
```



Erreur de compilation: un pointeur est un pointeur des variables d'un type donné

```
float a = 0.5 ;  
float* p = &a ;
```

Une variable de type *truc\** pointe sur un *truc*

# Tableaux unidimensionnels

## Déclaration d'un tableau

```
int tab[5];
```

- ▶ `int` est le type des cases du tableau
- ▶ `tab` est le nom du tableau
- ▶ `5` est la taille du tableau
- ▶ le type de `tab` est `int*`

# Tableaux unidimensionnels

## Déclaration d'un tableau

```
#include <stdio.h>

int main () {
    int taille = 5;
    int tab[taille];
    ...
}
```

### Avertissement à la compilation

```
>> warning : ISO C90 forbids variable length array 'tab' [-Wvla]
```

# Tableaux unidimensionnels

## Déclaration d'un tableau

```
#include <stdio.h>
#define TAILLE 5

int main () {
    int tab[TAILLE];
    ...
}
```

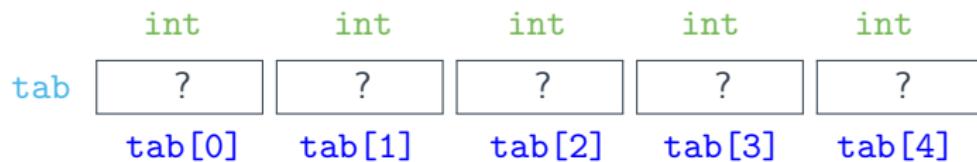
- ▶ **TAILLE** est la taille du tableau

# Tableaux unidimensionnels

## Déclaration d'un tableau et emplacement mémoire

```
int tab[5];
```

→ En C la numérotation des indices des tableaux commence à 0



→ **tab[i]** : case numéro *i* du tableau **tab**

Est ce que les cases du tableaux sont initialisés à la déclaration ?

- ▶ 0 dans la plupart des cas, mais non garanti !!

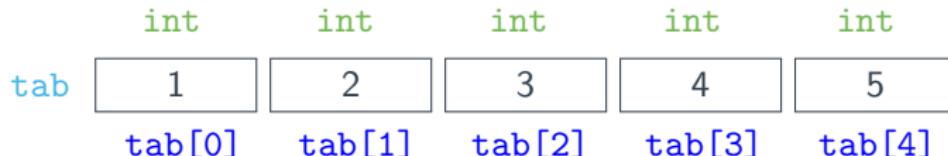
# Tableaux unidimensionnels

- ▶ Il existe une façon automatisée pour initialiser un tableau
- ▶ Elle consiste à placer les valeurs une à une entre accolades, séparés par des virgules {, , , }

## Déclaration et initialisation d'un tableau

```
#include <stdio.h>

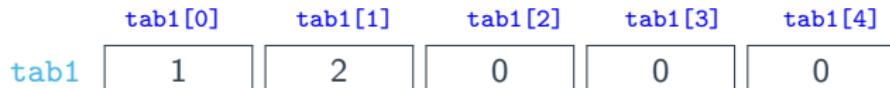
int main () {
    int tab[] = {1,2,3,4,5};
    ...
}
```



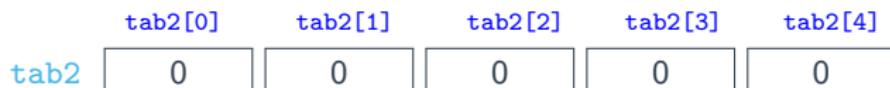
# Tableaux unidimensionnels

- On peut également définir seulement les valeurs des premières cases du tableau, les autres seront automatiquement mise à zero

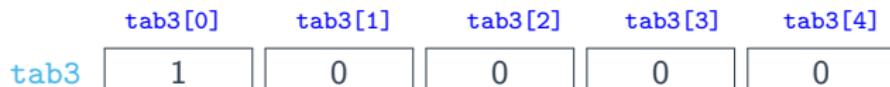
```
int tab1[5] = {1,2};
```



```
int tab2[5] = {0};
```



```
int tab3[5] = {1};
```

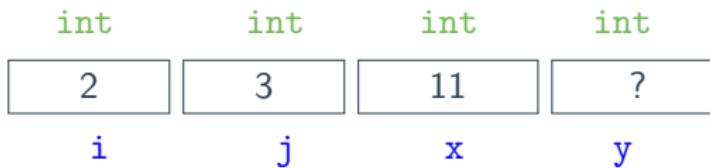
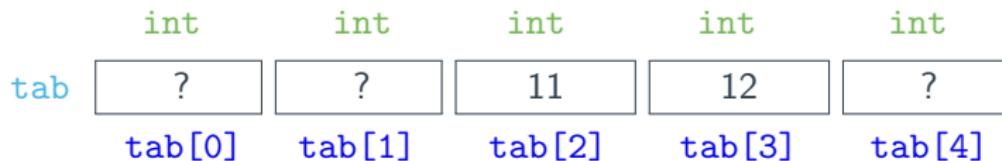


# Tableaux unidimensionnels

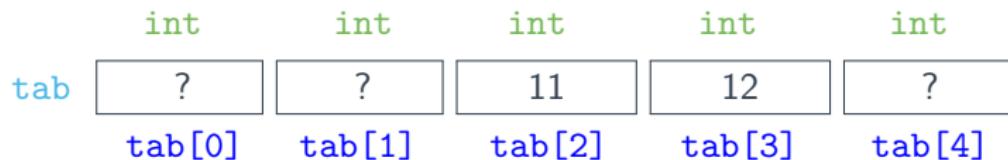
## Utilisation d'un tableau par valeur

### Ecriture

```
int tab[5];
int i = 2, j = 3, x = 11, y;
tab[i] = x, tab[j] = x+1;
```

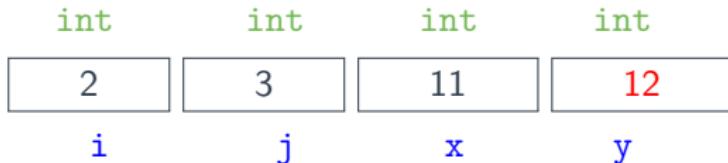


# Tableaux unidimensionnels

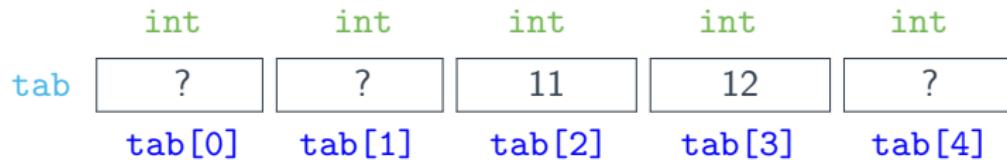


## Lecture

```
y = tab[j];
```

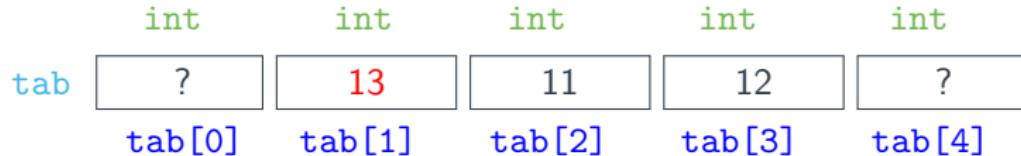


# Tableaux unidimensionnels

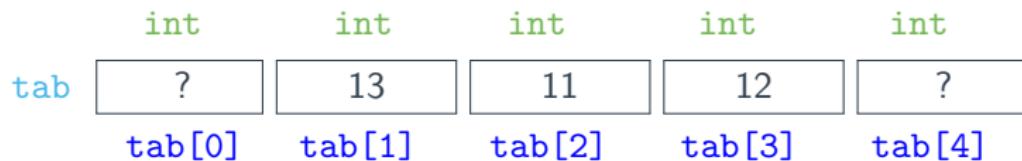


Ecriture : utilisation par index

```
tab[1] = 13;
```

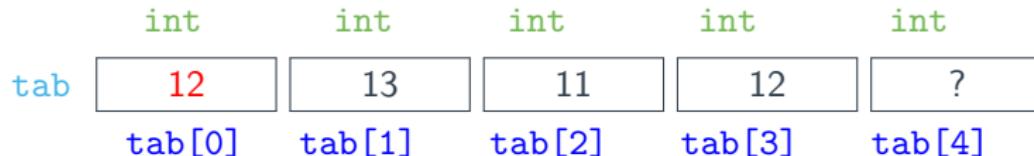


# Tableaux unidimensionnels

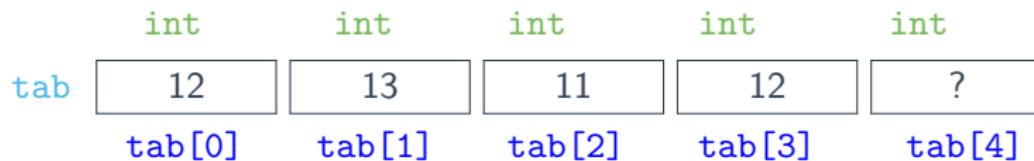


Ecriture : utilisation par index

```
tab[0] = tab[3];
```

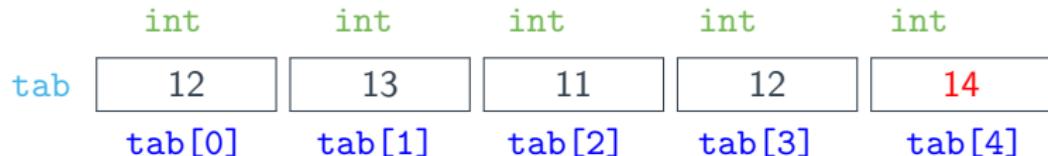


# Tableaux unidimensionnels



Ecriture : utilisation par index

```
tab[4] = 14;
```



# Tableaux unidimensionnels

## Parcourir un tableau

- ▶ Supposons qu'on veuille maintenant afficher les valeurs de chaque case du tableau. Le mieux est de se servir d'une boucle

```
int i;
float notes[9] = {14, 12, 9.5, 11, 13, 12, 16, 10, 19};
for (i = 0; \textcolor{red}{i} < 9; i++){
    printf(" %f \n", notes[i]);
}
```

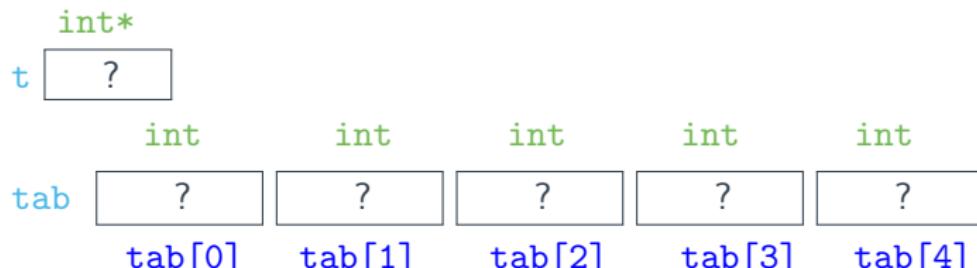
- ▶ La boucle parcourt le tableau à l'aide d'une variable appelée **i** ( c'est le nom le plus souvent utilisé pour parcourir un tableau !)
- ▶ Attention à ne pas tenter d'afficher la valeur de **notes [9]** ! Car on sera sorti du tableau, il y aura au mieux n'importe quoi, ou une erreur de segmentation car votre programme aura tenté d'accéder à une adresse qui ne lui appartient pas.

# Tableaux unidimensionnels

## Déclaration d'un tableau et utilisation par pointeurs

```
int* t;  
int tab[5];
```

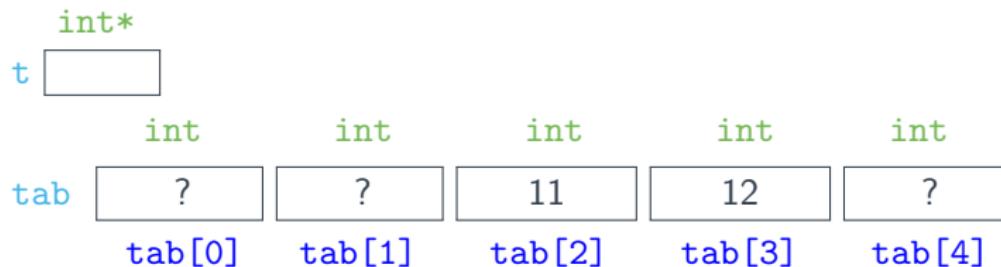
- ▶ le type de `tab` est `int*`
- ▶ `tab` est l'adresse de `tab[0]`
- ▶ `tab` est équivalent à `&tab[0]`



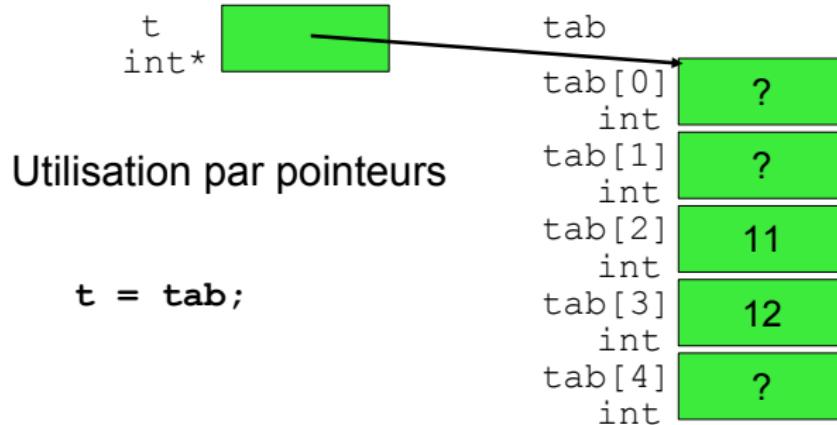
# Tableaux unidimensionnels

## Tableaux : utilisation par pointeurs

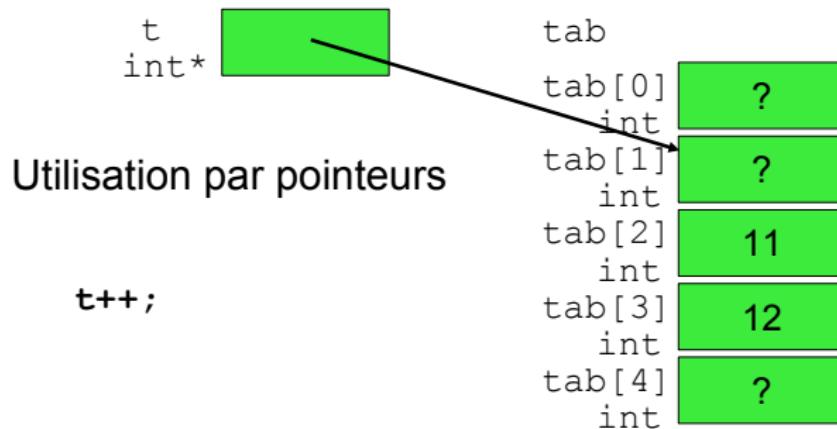
```
tab[2] = 11;  
tab[3] = 12;  
t = tab;
```



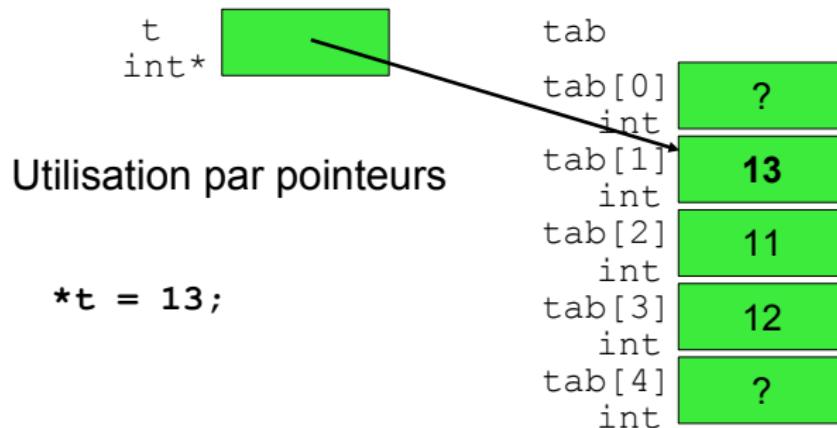
# Tableaux unidimensionnels



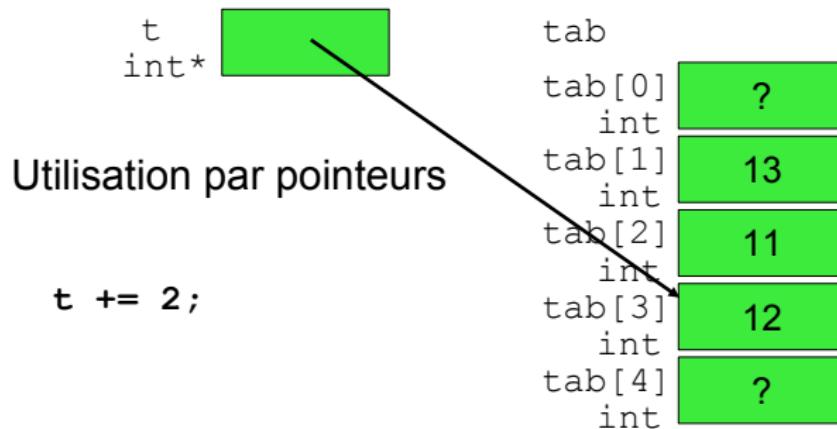
# Tableaux unidimensionnels



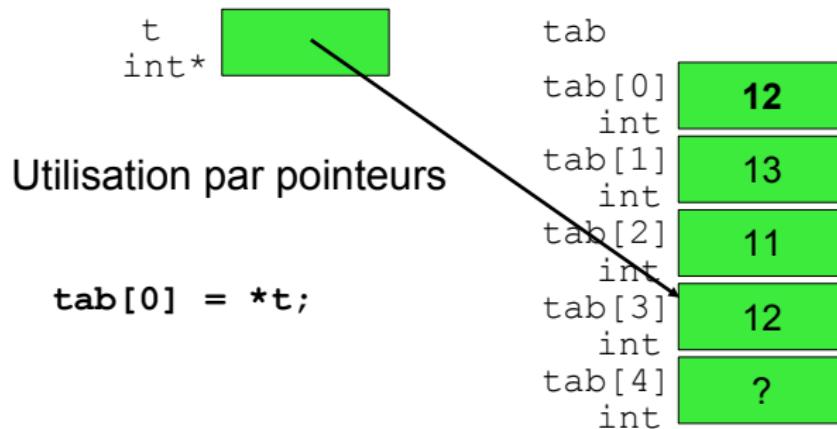
# Tableaux unidimensionnels



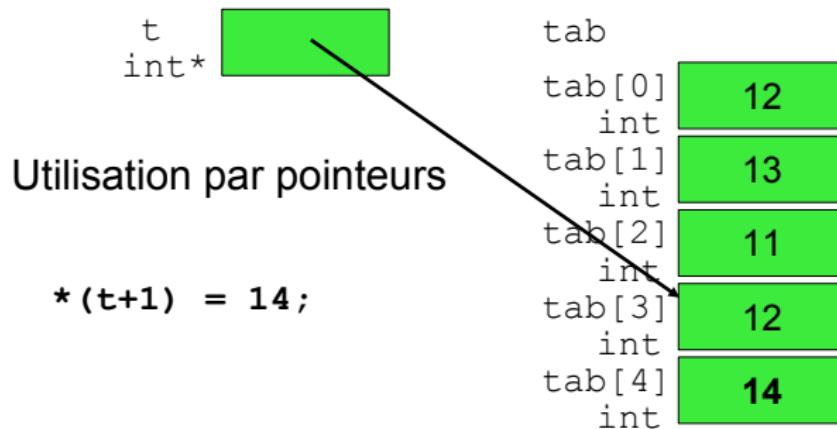
# Tableaux unidimensionnels



# Tableaux unidimensionnels



# Tableaux unidimensionnels



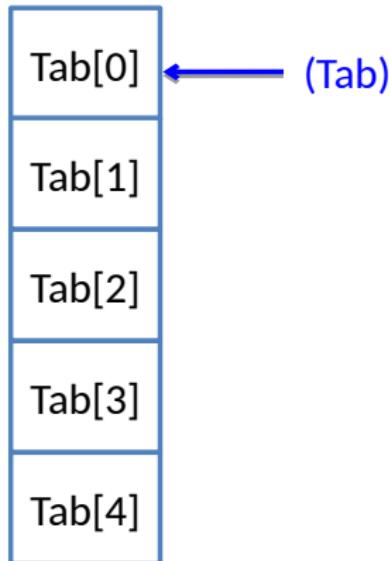
# Tableaux unidimensionnels

## Utilisation par pointeur

- ▶  $t + i$  : adresse de la  $i$ ème case suivant la case pointé par  $t$
- ▶  $t + i$  équivaut à  $\&t[i]$
- ▶  $*(t + i)$  : contenu de la  $i$ ème case suivant la case pointé
- ▶  $*(t + i)$  équivaut à  $t[i]$

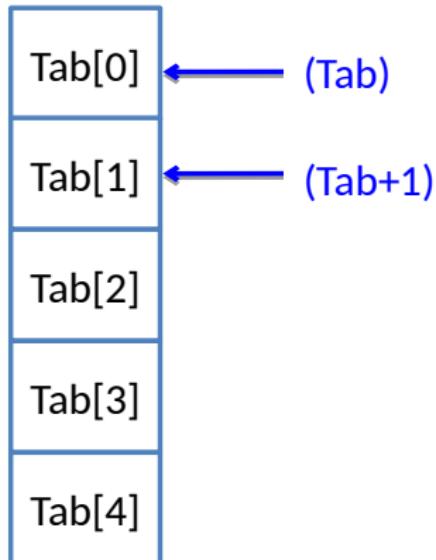
# Tableaux unidimensionnels

## Utilisation par pointeur



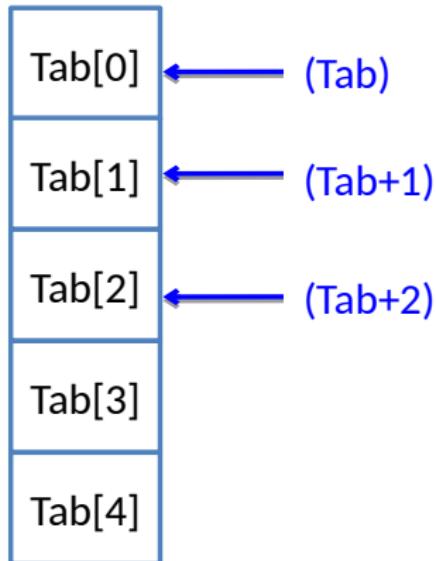
# Tableaux unidimensionnels

## Utilisation par pointeur



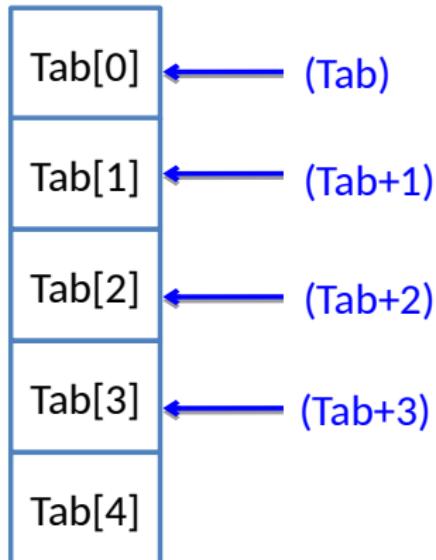
# Tableaux unidimensionnels

## Utilisation par pointeur



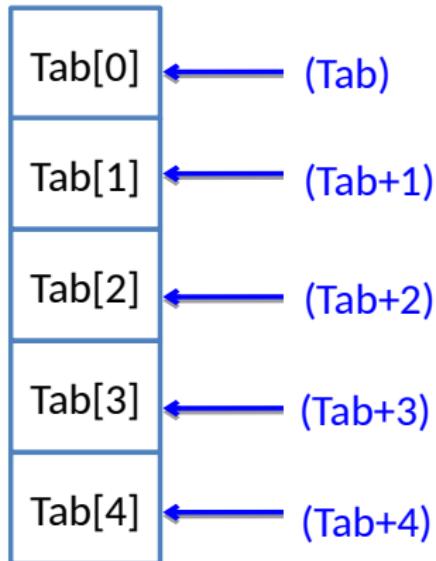
# Tableaux unidimensionnels

## Utilisation par pointeur



# Tableaux unidimensionnels

## Utilisation par pointeur



# Tableaux unidimensionnels

Comment manipuler un tableau dans une fonction ?

```
#include <stdio.h>

void remplirTab(int Table []);

int main ()
{
    int Tab[5];
    remplirTab(Tab);
}
```

**FAUX** car la taille du tableau « Tab » ne sera pas connue dans la fonction « remplirTab »

# Tableaux unidimensionnels

Comment manipuler un tableau dans une fonction ?

```
#include <stdio.h>

void remplirTab(int Table [],int longueur);

int main ()
{
    int Tab[5];
    remplirTab(Tab,5);
}
```

**CORRECT** : la taille du tableau « Tab » est passé en argument à l'appel de la fonction « remplirTab »

# Passage de tableaux à une fonction

```
#include <stdio.h>

void saisir (int tab [], int tailleTab) {
    int i;
    for (i = 0; i < tailleTab ; i++){
        scanf("%d", &tab[i]);
    }
}

void afficheTab (int tab [], int tailleTab) {
    int i;
    for (i = 0; i < tailleTab ; i++){
        printf("tab[%d] = %d \n", i,tab[i]);
    }
}

int main (){
    int tab[4] = {0};
    saisir (tab, 4);
    afficheTab (tab, 4);
    return 0;
}
```

# Passage de tableaux à une fonction avec un pointeur (option 1)

```
#include <stdio.h>

void saisir (int* tab , int tailleTab) {
    int i;
    for (i = 0; i < tailleTab ; i++){
        scanf("%d", &tab[i]);
    }
}

void afficheTab (int* tab , int tailleTab) {
    int i;
    for (i = 0; i < tailleTab ; i++){
        printf("tab[%d] = %d \n", i,tab[i]);
    }
}

int main (){
    int tab[4] = {0};
    saisir (tab , 4);
    afficheTab (tab , 4);
    return 0;
}
```

# Passage de tableaux à une fonction avec un pointeur (option 2)

```
#include <stdio.h>

void saisir (int* tab , int tailleTab) {
    int i;
    for (i = 0; i < tailleTab ; i++){
        scanf("%d", tab + i );
    }
}

void afficheTab (int* tab , int tailleTab) {
    int i;
    for (i = 0; i < tailleTab ; i++){
        printf("tab[%d] = %d \n", i,* (tab+i));
    }
}

int main (){
    int tab[4] = {0};
    saisir (tab , 4);
    afficheTab (tab , 4);
    return 0;
}
```

# Tableaux bi-dimensionnels

- ▶ Un tableau à deux dimension est à interpréter comme un tableau à dimension N dont chaque élément est un tableau de dimension M

	Module n°1	Module n°2	Module n°3	...	Module n°M
Etudiant n°1	14.25	16.	18.	...	17
Etudiant n°2	3.5	5.	9.	...	4
⋮	⋮	⋮	⋮	⋮	⋮
Etudiant n°N	11	13.5	12	...	15

- ▶ On appelle N le nombre de lignes et M le nombre de colonnes du tableau, N et M sont alors les deux dimensions du tableau
- ▶ Un tableau à deux dimension contient donc  $N \times M$  éléments
- ▶ Comme pour les tableaux à une dimension, le nom d'un tableau est le représentant de l'adresse du premier élément du tableau (c.-à-d. l'adresse de la première ligne du tableau).
- ▶ Les éléments d'un tableau à deux dimensions sont stockées ligne par ligne dans la mémoire.

# Tableaux bi-dimensionnels

## Les tableaux bi-dimensionnel en C

```
int Tab[2][3];
```

- *int* : type des cases du tableau
- *Tab* : nom du tableau
- La taille du tableau est  $2 * 3 = 6$

Tab[0][0]
Tab[0][1]
Tab[0][2]
Tab[1][0]
Tab[1][1]
Tab[1][2]

- ▶ Les éléments d'un tableau à deux dimensions sont stockées ligne par ligne dans la mémoire ([ligne] [colonne]).

# Tableaux bi-dimensionnels

## Les tableaux bi-dimensionnel en C

```
int Tab[2][3];
```

- *int* : type des cases du tableau
- *Tab* : nom du tableau
- La taille du tableau est  $2 * 3 = 6$



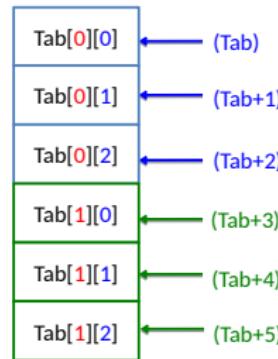
- ▶ Les éléments d'un tableau à deux dimensions sont stockées ligne par ligne dans la mémoire

# Tableaux bi-dimensionnels

## Les tableaux bi-dimensionnel en C

```
int Tab[2][3];
```

- *int* : type des cases du tableau
- *Tab* : nom du tableau
- La taille du tableau est  $2 * 3 = 6$



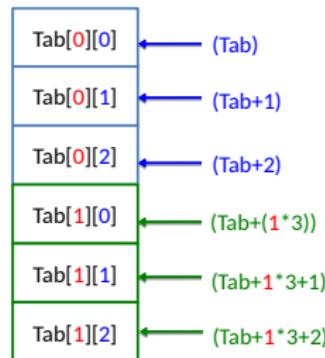
- ▶ Les éléments d'un tableau à deux dimensions sont stockées ligne par ligne dans la mémoire

# Tableaux bi-dimensionnels

## Les tableaux bi-dimensionnel en C

```
int Tab[2][3];
```

- *int* : type des cases du tableau
- *Tab* : nom du tableau
- La taille du tableau est  $2 * 3 = 6$



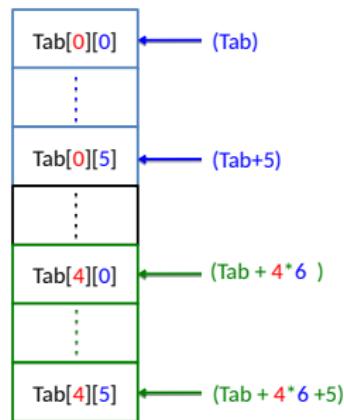
- ▶ Les éléments d'un tableau à deux dimensions sont stockées ligne par ligne dans la mémoire

# Tableaux bi-dimensionnels

## Les tableaux bi-dimensionnel en C

```
int Tab[5][6];
```

- *int* : type des cases du tableau
- *Tab* : nom du tableau
- La taille du tableau est  $5 * 6 = 30$



- ▶ Les éléments d'un tableau à deux dimensions sont stockées ligne par ligne dans la mémoire

# Tableaux bi-dimensionnels

## Initialisation des tableaux à deux dimensions

- ▶ Lors de la déclaration d'un tableau, on peut initialiser les éléments du tableau, en indiquant la liste des valeurs respectives entre accolades.
- ▶ À l'intérieur de la liste, les éléments de chaque ligne du tableau sont encore une fois comprises entre accolades.
- ▶ Pour améliorer la lisibilité des programmes, on peut indiquer les composantes dans plusieurs lignes.
- ▶ Lors de l'initialisation, les valeurs sont affectées ligne par ligne en passant de gauche à droite.
- ▶ Nous ne devons pas nécessairement indiquer toutes les valeurs : Les valeurs manquantes seront initialisées par zéro.

```
int A [3][10] = { { 0,10, 20, 30, 40, 50, 60, 70, 80, 90},  
                  {10, 11, 12, 13, 14, 15, 16, 17, 18, 19},  
                  { 1, 12, 23, 34, 45, 56, 67, 78, 89, 90} };  
  
float B [3][2] = { {1.05, 1.10},  
                   {0.086, 87},  
                   {12.5, 12.3} };
```

# Tableaux bi-dimensionnels

Considérons un tableau A de dimension  $N \times M$

## Accès aux éléments d'un tableau à deux dimensions

- ▶ Les indices du tableau varient de 0 à N-1 respectivement de 0 à M - 1
- ▶ L'élément de la  $i^{\text{ième}}$  ligne et de la  $j^{\text{ième}}$  colonne est noté :  $A[i-1][j-1]$

```
int A[ ][10] = { { 0, 10, 20, 30, 40, 50, 60, 70, 80, 90 },
                  { 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, },
                  { 1, 12 } } ;
```

- ▶ Le tableau A a 3 lignes et 10 colonnes

$A[0][0] \rightarrow 0$ ,  $A[0][1] \rightarrow 10$ ,  $A[0][2] \rightarrow 20$ ,

$A[1][0] \rightarrow 10$ ,  $A[1][1] \rightarrow 11$ ,  $A[1][2] \rightarrow 12$ ,

$A[2][0] \rightarrow 1$ ,  $A[2][1] \rightarrow 12$ ,  $A[2][2] \rightarrow 0$

# Tableaux bi-dimensionnels

## Parcours d'un tableaux à deux dimensions

```
# include <stdio.h>
# include <stdlib.h>
# define NBCOL 2

void afficherTableau (int tableau [ ][NBCOL], int nbLigne) ;

int main ( )
{
    int tableau [2][NBCOL] = {{10, 20},
                             {15, 35} };
    afficherTableau (tableau ,2);
    return 0;
}

void afficherTableau (int tableau [ ][NBCOL], int nbLigne )
{
    int i = 0;
    int j = 0;
    for (i = 0; i < nbLigne ; i++)
    {
        for (j = 0; j < NBCOL ; j++)
        {
            printf (" Tableau [%d][%d] = %d \n", i , j , tableau [ i ][ j ] );
        }
    }
}
```

# Tableaux bi-dimensionnels

## Les tableaux bi-dimensionnel en C

Comment manipuler un tableau à deux dimensions ?

```
#include <stdio.h>

void remplirTab(int Table [],int nbCol,int nbLin);

int main ()
{
    int Tab[5][6];
    remplirTab(Tab,5,6);
}
```

On ne peut pas mettre **int Table [][]**

Il faut à minima donner la dernière dimension

# Tableaux bi-dimensionnels

## Les tableaux bi-dimensionnel en C

Comment manipuler un tableau à deux dimensions ?

```
#include <stdio.h>

void remplirTab(int Table [][6],int nbCol,int nbLin);

int main ()
{
    int Tab[5][6];
    remplirTab(Tab,5,6);
}
```

# Chaines de caractères - rappel sur le type char

## Le type *char*

- ❑ Le type *char* permet de stocker des nombres compris entre 0 et 256. Mais il faut savoir qu'en C on l'utilise rarement pour ça.
  - Il est en fait prévu pour stocker un caractère
- ❑ En effet, la plupart des caractères « de base » sont codés entre les nombres 0 et 127. Une table fait la conversion entre les nombres et les lettres : la table ASCII

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[\NUL]	32	20	[\SPACE]	64	40	@	60	60	
1	1	[\START OF HEADING]	33	21	!	65	41	A	61	61	a
2	2	[\START OF TEXT]	34	22	"	66	42	B	62	62	b
3	3	[\END OF TEXT]	35	23	#	67	43	C	63	63	c
4	4	[\GROUP SEPARATOR]	36	24	%	68	44	D	64	64	d
5	5	[\ENQURY]	37	25	&	69	45	E	65	65	e
6	6	[\ACKNOWLEDGE]	38	26	\$	70	46	F	66	66	f
7	7	[\BEL]	39	27	_	71	47	G	67	67	g
8	8	[\HORIZONTAL SPACE]	40	28	{	72	48	H	68	68	h
9	9	[\NON-BREAKING SPACE]	41	29	}	73	49	I	69	69	i
10	A	[\LINE FEED]	42	2A	*	74	4A	J	6A	6A	j
11	B	[\VERTICAL TAB]	43	2B	+	75	4B	K	6B	6B	k
12	C	[\FORM FEED]	44	2C	-	76	4C	L	6C	6C	l
13	D	[\CARRIAGE RETURN]	45	2D	.	77	4D	M	6D	6D	m
14	E	[\SHIFT OUT]	46	2E	,	78	4E	N	6E	6E	n
15	F	[\SHIFT IN]	47	2F	/	79	4F	O	6F	6F	o
16	10	[\NULL]	48	30	0	80	50	P	70	70	p
17	11	[\DEVICE CONTROL 1]	49	31	1	81	51	Q	71	71	q
18	12	[\DEVICE CONTROL 2]	50	32	2	82	52	R	72	72	r
19	13	[\DEVICE CONTROL 3]	51	33	3	83	53	S	73	73	s
20	14	[\DEVICE CONTROL 4]	52	34	4	84	54	T	74	74	t
21	15	[\NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	75	75	u
22	16	[\SYNCHRONOUS CLEJ]	54	36	6	86	56	V	76	76	v
23	17	[\END OF TRANS. BLOCK]	55	37	7	87	57	W	77	77	w
24	18	[\START OF MEDIUM]	56	38	8	88	58	X	78	78	x
25	19	[\END OF MEDIUM]	57	39	9	89	59	Y	79	79	y
26	1A	[\SUBSTITUTE]	58	3A	:	90	5A	Z	7A	7A	z
27	1B	[\ESCAPE]	59	3B	_	91	5B	{	7B	7B	{
28	1C	[\FILE SEPARATOR]	60	3C	^	92	5C	}`	7C	7C	`
29	1D	[\GROUP SEPARATOR]	61	3D	~	93	5D	]	7D	7D	]
30	1E	[\RECORD SEPARATOR]	62	3E	>	94	5E	^	7E	7E	-
31	1F	[\UNIT SEPARATOR]	63	3F	?	95	5F	_	7F	7F	[DEL]

- Le langage C permet de faire très facilement la traduction lettre => nombre correspondant.

# Chaines de caractères - rappel sur le type char

## Le type *char*

- Le langage C permet de faire très facilement la traduction lettre <=> nombre correspondant.

Testons le programme suivant

```
int main() {
{
    char lettreMaj= 'A';
    char lettreMin= 'a';
    printf( " A => %d et a => %d\n" , lettreMaj,lettreMin);
    return 0;
}
```

A l'exécution il renvoie

```
A => 65 et a => 97
```

- En effet, la lettre 'a' n'est pas identique à la lettre 'A', l'ordinateur faisant la différence entre les majuscules et les minuscules (on dit qu'il « respecte la casse »).

# Chaines de caractères - rappel sur le type char

## Le type *char* (entrée et sortie)

### ❑ Entrée d'un caractère unique

- En ligne de commandes, un programme demande souvent un caractère à l'utilisateur.

Testons le programme suivant

```
int main( ) {  
    char m;  
    printf( " Entrer un caractere ? " );  
    scanf( "%c" , &m);  
    printf("caractere = %c \n", m);  
    printf("(code ascii=%d)\n", m);  
    return 0;  
}
```

- **%c** : format d'entrée-sortie pour les *char*
- **%d** : représentation *int* du *char*
- code ASCII d'un *char* : nombre entier de 0 à 255

A l'exécution il renvoie

```
> ./Prog.exe  
Entrer un caractere ? b ←  
caractere = b  
(code ascii=98)  
>
```

L'utilisateur a tapé sur la touche **b**  
puis la touche Entrée.

# Chaines de caractères - rappel sur le type char

## Le type *char* (entrée et sortie)

### ❑ Entrée d'un caractère unique

- En ligne de commandes, un programme demande souvent un caractère à l'utilisateur.

Testons le programme suivant

```
int main( ) {  
    char m;  
    printf( " Entrer un caractere ? " );  
    scanf( "%c" , &m);  
    printf("caractere = %c \n", m);  
    printf("(code ascii=%d)\n", m);  
    return 0;  
}
```

- **%c** : format d'entrée-sortie pour les *char*
- **%d** : représentation *int* du *char*
- code ASCII d'un *char* : nombre entier de 0 à 255

A l'exécution il renvoie

```
> ./Prog.exe  
Entrer un caractere ? b ←  
caractere = b  
(code ascii=98)  
>
```

L'utilisateur a tapé sur la touche **b**  
puis la touche Entrée.

# Chaines de caractères - rappel sur le type char

## Le type *char* (entrée et sortie)

### □ Entrée de 2 caractères

Testons le programme suivant

```
int main() {
    char m, n;
    printf("Entrer un caractere ? ");
    scanf("%c", &m);
    printf("caractere 1 = %c (code ascii=%d)\n", m,m);
    printf("Entrer un deuxième caractere ? ");
    scanf("%c", &n);
    printf("caractere 2 = %c (code ascii=%d)\n", n,n);
    return 0;
}
```

A l'exécution il renvoie

```
> ./Prog.exe
Entrer un caractere ? a
caractere 1 = a (code ascii=97)
Entrer un deuxième caractere ? caractere 2 =
                           (code ascii=10)
>
```



Ce programme ne marche pas.

Mais pourquoi ?

# Chaines de caractères - rappel sur le type char

## Le type *char* (entrée et sortie)

### ❑ Entrée de 2 caractères

Testons le programme suivant

```
int main() {
    char m, n;
    printf("Entrer un caractere ? ");
    scanf("%c", &m);
    printf("caractere 1 = %c (code ascii=%d)\n", m,m);
    printf("Entrer un deuxième caractere ? ");
    scanf("%c", &n);
    printf("caractere 2 = %c (code ascii=%d)\n", n,n);
    return 0;
}
```

A l'exécution il renvoie

```
> ./Prog.exe
Entrer un caractere ? a
caractere 1 = a (code ascii=97)
Entrer un deuxième caractere ? caractere 2 =
                           (code ascii=10)
>
```



- Le scanf avec %c lit tous les caractères tapés au clavier.

- Ici l'utilisateur a tapé sur la touche a puis sur la touche Entrée.

La touche Entrée correspond au caractère « saut de ligne » '\n' de code ascii = 10

# Chaines de caractères

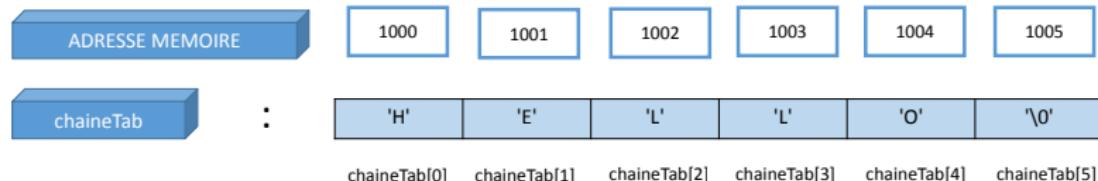
## Les chaines de caractères

- Les chaînes de caractères ne sont rien d'autre que des tableaux de *char*

Toutefois, une chaîne de caractères ne contient pas que des lettres ! Une chaîne de caractère doit impérativement contenir un caractère spécial à la fin de la chaîne, appelé « caractère de fin de chaîne ». Ce caractère s'écrit **"\0"**.

```
char chaineTab[ ] = "HELLO";
```

En tapant entre guillemets " " la chaîne que vous voulez mettre dans votre tableau, le compilateur C calcule automatiquement la taille nécessaire. C'est-à-dire qu'il compte les lettres et ajoute 1 pour placer le caractère "**"\0"** .



# Chaines de caractères

## Le symbole de fin de chaîne de caractères

- ▶ Le caractère '\0' permet tout simplement d'indiquer la fin de la chaîne
- ▶ Par conséquent, pour stocker le mot "hello" (qui comprend 6 lettres) en mémoire, il ne faut pas un tableau de 6 char mais de 7 !
- ▶ Chaque fois que vous créez une chaîne de caractères, vous allez donc devoir penser à prévoir de la place pour le caractère de fin de chaîne. Il faut toujours ajouter un bloc de plus dans le tableau pour stocker ce caractère '\0', c'est impératif !
- ▶ Oublier le caractère de '\0' est une source d'erreurs impitoyable du langage C.

# Chaines de caractères

## Le symbole de fin de chaîne de caractères

Grâce au caractère '\0'

- ▶ Vous n'aurez pas à retenir la taille de votre tableau car il indique que le tableau s'arrête à cet endroit
- ▶ Vous pourrez passer votre tableau de *char* à une fonction sans avoir à ajouter à coté une variable indiquant la taille du tableau
- ▶ Cela n'est valable que pour les chaînes de caractères (c'est à dire le type *char\**, qu'on peut aussi écrire *char[ ]*).
- ▶ ATTENTION : Pour les autres types de tableaux, vous êtes toujours obligés de retenir la taille du tableau quelques part.

# Chaines de caractères

## Création et initialisation de la chaîne

```
int main(){
    char chaine [] = "Hello";
    // La taille du tableau chaine est automatiquement calculee
    printf("%s", chaine);
    return 0;
}
```

- ▶ En tapant entre guillemets la chaîne que vous voulez mettre dans votre tableau, le compilateur C calcule automatiquement la taille nécessaire. C'est à dire qu'il compte les lettres et ajoute 1 pour placer le caractère '\0'.
- ▶ Il y a toutefois un défaut : ça ne marche que pour l'initialisation ! Vous ne pouvez pas écrire plus loin dans le code  
chaine = "Hello";

# Chaines de caractères

## Lire une chaîne de caractères

- ▶ On peut enregistrer une chaîne entrée par l'utilisateur via un *scanf*, en utilisant là encore le symbole *%s*
- ▶ Seul problème : on ne sait pas combien de caractères l'utilisateur va entrer.
- ▶ Il va falloir créer un tableau de *char* très grand, suffisamment grand !

```
int main(){
    char prenom [100];
    printf(" Comment t'appeles-tu ?" );
    scanf("%s", prenom);
    printf(" Salut %s, je suis heureux de te rencontrer !", prenom);
    return 0;
}
```

# Chaines de caractères

## Les chaines de caractères (entrée et sortie)

Testons le programme suivant

```
int main() {
    char motTabSortie[] = "Entrer un mot ?";
    char motTabEntree[100];
    printf(" %s ", motTabSortie);
    scanf("%s", motTabEntree);
    printf("Chaine = %s", motTabEntree);
    return 0;
}
```

A l'exécution il renvoie

```
> Prog.exe
Entrer un mot ? toto
Chaine = toto
```

- **%s** : format d'entrée-sortie pour les chaînes de caractères (un tableau `char []`)

# Chaines de caractères

## Les chaines de caractères

### Tableaux de char

```
char chaineTab1[] = "HELLO";
char chaineTab2[] = "GOODBYE";
char chaineTab3[50];

chaineTab1 = ChaineTab2; /*IMPOSSIBLE => Erreur */
chaineTab3 = ChaineTab1, /*IMPOSSIBLE => Erreur */
```



- ❑ Dans cet exemple, nous essayons de copier l'adresse de chaineTab2 dans chaineTab1, respectivement l'adresse de la chaîne constante dans chaineTab3. Ces opérations sont impossibles et illégales parce que l'adresse représentée par le nom d'un tableau reste toujours constante.
- ❑ 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 <stdio.h> ou <string.h>.

# Chaines de caractères

## Les chaines de caractères

- Mais on peut aussi attribuer l'adresse d'une chaîne de caractères constante à un pointeur sur `char`

```
int main () {
{
    char chaineTab[] = "HELLO";
    char * chainePointeur = "SALUT";
    printf("chaine de caractere n°= %s \n ", chainePointeur);
    return 0;
}
```

- chainePointeur** 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. La chaîne constante peut être lue, copiée ou affichée, mais pas modifiée.



- chaineTab** est un tableau qui a exactement la grandeur pour contenir la chaîne de caractères et la terminaison '\0'. Les caractères de la chaîne peuvent être changés, mais le nom **chaineTab** pointera toujours sur la même adresse en mémoire



# Chaines de caractères

## Tableaux de chaînes de caractères

- ▶ Un tableau de chaînes de caractères correspond à un tableau à deux dimensions du type `char`, où chaque ligne contient une chaîne de caractères.
- ▶ Déclaration `char Jour[7][9]` réserve l'espace en mémoire pour 7 mots contenant 9 caractères (dont 8 caractères significatifs).

```
char JOUR[7][9] = {"lundi", "mardi", "mercredi",
                    "jeudi", "vendredi",
                    "samedi", "dimanche"};
int i = 2;
printf("Aujourd'hui, c'est %s ! \n", JOUR[i]);
```

- ▶ ATTENTION : Des expressions comme `JOUR[i]` représentent l'adresse du premier élément d'une chaîne de caractères. N'essayez donc pas de 'modifier' une telle adresse par une affectation directe !

# Chaines de caractères

## Les paramètres de la fonction *main*

```
int main ( int argc , char* argv [ ] ) {  
...  
return 0;  
}
```

### Les paramètres : *argc* et *argv*

- ❑ Le paramètre *argc* est le nombre de paramètres passé au programme.  
Ce nombre est toujours > 1 car le premier paramètre est toujours le nom de l'exécutable.
  
- ❑ Le paramètre *argv* est un tableau de chaînes de caractères (ici des *char\**) contenant les paramètres effectivement passés au programme.

# Chaines de caractères

## Les paramètres de la fonction *main*

Le programme simple suivant affiche les paramètres qui lui sont passés depuis le Terminal:

```
//prg.c
int main ( int argc , char * argv [ ] ) {
    int i;
    // affichage des arguments
    printf ("Nombre d'arguments passes au programme : %d\n", argc);
    for(i = 0 ; i < argc ; i++) {
        printf (" argv[%d] : '%s'\n" , i, argv[i]);
    }
    return 0 ;
}
```

Une fois ce programme compilé, si l'utilisateur le lance depuis le Terminal en tapant :

\$ ./prg.exe test.txt 2

alors le programme affichera dans le Terminal:

```
Nombre d'arguments passes au programme : 3
argv[0] : './prg.exe'
argv[1] : 'test.txt'
argv[2] : '2'
```

puis retournera 0 (tout s'est bien passé).

# Chaines de caractères

## Les paramètres de la fonction *main*

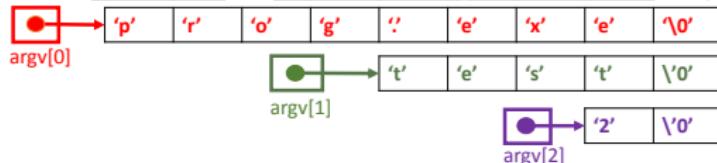
Le programme simple suivant affiche les paramètres qui lui sont passés depuis le Terminal:

```
//prog.c
int main ( int argc , char * argv [ ] ) {
    int i;
    // affichage des arguments
    printf ("Nombre d'arguments passes au programme : %d\n", argc);
    for(i = 0 ; i < argc ; i ++){
        printf (" argv[%d] : '%s'\n" , i, argv[i]);
    }
    return 0 ;
}
```

Une fois ce programme compilé, si l'utilisateur le lance depuis le Terminal en tapant :

\$ ./prog.exe test 2

Alors argc vaut 3 et argv est tableau de dimension 3 composé de pointeur sur char



# Chaines de caractères

## Erreurs à ne pas commettre

- ▶ Une variable de type caractère n'est pas une chaîne, aussi il est très dangereux de fournir un caractère en argument à une fonction qui attend une chaîne.
- ▶ Impossible de faire des comparaisons avec l'opérateur ==  
`if (chaine == "salut")` donne un résultat imprévisible, quelque soit le contenu de la variable *chaine*
- ▶ Impossible de faire des affectations en chaînes :

```
char chaine1 [] = "salut";
char chaine2 [6];
chaine2 = chaine 1
```

Message d'erreur du compilateur ;  
Error : ISO C++ forbids assignment  
of arrays

- ▶ Impossible de faire des concaténations avec des chaînes ou des caractères à l'aide de l'opérateur + (comme en python)

# Chaines de caractères

- ▶ Pour travailler avec les chaines (faire des comparaisons, des copies, des concaténations, etc, ....), il faut utiliser des fonctions spéciales
- ▶ Ces fonctions sont placées dans le fichier *string.h*
- ▶ Pensez donc à inclure `#include < string.h >` en haut des fichiers .c où vous en avez besoin.  
Si vous ne le faites pas, l'ordinateur ne connaîtra pas ces fonctions car il n'aura pas les prototypes, et la compilation plantera.
  - ▶ *strlen* : calculer la longueur d'une chaîne
  - ▶ *strcpy* : copier une chaîne dans une autre
  - ▶ *strcat* : concaténer 2 chaînes
  - ▶ *strcmp* : comparer 2 chaînes

# A retenir

- ▶ Un tableau est un ensemble fini d'éléments de même type, stockés en mémoire à des adresses contiguës.
- ▶ On accède à un élément du tableau en lui appliquant l'opérateur `[]`
- ▶ Les index des éléments d'un tableau vont de `0` à `nbr_element - 1`
- ▶ La taille d'un tableau DOIT être connue statiquement par le compilateur.

```
#define TAILLE 20
int Tab[TAILLE];
```

Impossible donc d'écrire `int Tab[n]` si `n` est une variable.

- ▶ Initialisation et réservation automatique

```
int nom_du_tableau [] = {constante1, ..., constanteN};
```

→ Réservation d'espace en mémoire pour `N` éléments de type `int`

# A retenir

## Relation entre Pointeurs et Tableaux

- ▶ Il n'y a pas à proprement parler de type tableau en C
- ▶ Un tableau correspond en fait à un pointeur CONSTANT (non modifiable) sur l'adresse du premier élément du tableau.

- $\text{tab} \Leftrightarrow \&(\text{tab}[0])$  (adresse du 1er élément)
- $*\text{tab} \Leftrightarrow \text{tab}[0]$  (valeur du 1er élément)
- $*(\text{tab} + k) \Leftrightarrow \text{tab}[k]$  (valeur du  $(k+1)$ -ième élément)
- $\text{tab} + k \Leftrightarrow \&(\text{tab}[k])$  (adresse du  $(k+1)$ -ième élément)

→ Pointeurs et tableaux se manipulent donc exactement de même manière.

- ▶ La manipulation de tableaux possède certains inconvénients par rapport à la manipulation des pointeurs !

# A retenir

- ▶ Aucune opération globale n'est autorisée sur un tableau.  
**On ne peut pas écrire tab1 = tab2;.**  
→ Il faut effectuer l'affectation pour chacun des éléments du tableau
- ▶ On ne peut pas créer de tableaux dont la taille est une variable du programme
- ▶ On ne peut pas créer de tableaux bidimensionnels dont les lignes n'ont pas toutes le même nombre d'éléments.  
→ Ces opérations deviennent possibles dès que l'on manipule des pointeurs alloués dynamiquement.

# A retenir

## Tableaux multi-dimensionnels

- ▶ En C, un tableau multidimensionnel est considéré comme un tableau dont les éléments sont eux-même des tableaux
- ▶ Un tableau à deux dimensions se déclare de la manière suivante :

```
int mat[10][20];
```

- ▶ Le compilateur allouera une zone mémoire permettant de stocker de manière contiguë 10 tableaux de 20 entiers.
- ▶ Si l'on souhaite utiliser la réservation automatique, il faudra quand même spécifier toutes les dimensions sauf la première (le nombre de lignes dans le cas d'un tableau bidimensionnel).

```
int mat[] [3] = {{1, 0, 1}, {0, 1, 0}};
```