

# Programmation en C

## Tableaux, chaînes de caractères et pointeurs

Alain CROUZIL

`alain.crouzil@irit.fr`

Département d'Informatique (Ddl)

Institut de Recherche en Informatique de Toulouse (IRIT)

Équipe Computational Imaging and Vision (MINDS)

Faculté Sciences et Ingénierie (FSI)

Université Toulouse III – Paul Sabatier (UPS)

Licence Informatique – Licence MIASHS  
2019-2020

# Sommaire

- 1 Tableaux
- 2 Chaînes de caractères
- 3 Pointeurs

# Sommaire

## 1 Tableaux

- Introduction
- Tableaux à une dimension
- Tableaux à deux dimensions

## 2 Chaînes de caractères

## 3 Pointeurs

# Introduction

## Définition

- Un tableau est un ensemble d'éléments de même type désignés par un identificateur unique.

## Accès aux éléments

- L'accès à chaque élément d'un tableau se fait par l'intermédiaire d'un ou de plusieurs indices qui désignent sa position au sein de l'ensemble.
- Les indices sont des entiers.

# Tableaux à une dimension (1/3)

## Déclaration

```
type_element nom_tableau[nombre_d_elements];
```

- *Exemple :*


```
int tab[10];
```

déclare un tableau tab comportant 10 éléments de type **int**.

- *Remarque :* les éléments d'un tableau sont toujours rangés à la suite les uns des autres en mémoire.


## Accès à un élément

```
nom_tableau[indice]
```

-  Le premier élément est toujours celui d'indice 0.

- *Exemples :*

- tab[0] désigne le premier élément du tableau tab ;
- tab[9] désigne, dans l'exemple ci-dessus, le dernier élément du tableau tab.

-  L'indice doit être un entier compris entre 0 et  $n - 1$ , où  $n$  est le nombre d'éléments du tableau. C'est au programmeur de s'en assurer !

## Tableaux à une dimension (2/3)

### Initialisation


Il est possible d'initialiser un tableau au moment de sa déclaration.

```
type_element nom_tableau[nombre_d_elements]={el1,el2,...,eln};
```

- *Exemple :*

```
int t[3]={1,2,3};
```

déclare un tableau de trois **int** et place les valeurs 1, 2 et 3 dans les trois « cases » de t.

-  Ce genre d'affectation « globale » du contenu d'un tableau n'est possible qu'au moment de sa déclaration.

# Tableaux à une dimension (3/3)

## Parcours d'un tableau

Exemple :

```
1  int i , tab[5];  
2  
3  /* Mise à zéro de tous les éléments du tableau */  
4  for ( i=0; i<5; i++)  
5      tab[i]=0;  
6  
7  /* Affichage des éléments du tableau */  
8  for ( i=0; i<5; i++)  
9      printf("%d_", tab[i]);  
10     printf("\n");
```

Depuis la norme C99, il est possible de déclarer la variable qui sert d'indice dans la boucle `for` :

```
int tab[5];  
for (int i=0; i<5; i++)  
    tab[i]=0;
```

# À vos boîtiers !

5





# Tableaux à deux dimensions (1/3)

## Déclaration

`type_element nom_tableau [nombre_de_lignes][nombre_de_colonnes];`

- *Exemple* : `int t[2][3];`  
déclare un tableau t comportant six éléments de type `int`.

## Accès à un élément

`nom_tableau[indice1][indice2]`

- *Exemple* : `t[1][2]`  
désigne l'élément du tableau t qui se trouve sur la ligne 1 et la colonne 2 :

	0	1	2
0			
1			X

# Tableaux à deux dimensions (2/3)

## En mémoire

- Les éléments d'un tableau sont rangés en mémoire « ligne par ligne ».
- Exemple :*

t[0][0]
t[0][1]
t[0][2]
t[1][0]
t[1][1]
t[1][2]

## Initialisation

- Exemple :* `int t[2][3]={1,2,3},{4,5,6}};`

	0	1	2
0	1	2	3
1	4	5	6

# Tableaux à deux dimensions (3/3)

## Parcours d'un tableau à deux dimensions

*Exemple de parcours ligne par ligne :*

```
1  int tab[2][3];
2
3  /* Mise à zéro de tous les éléments du tableau */
4  for (int i=0;i<2;i++)
5      for (int j=0;j<3;j++)
6          tab[i][j]=0;
7
8  /* Affichage des éléments du tableau */
9  for (int i=0;i<2;i++)
10 {
11     for (int j=0;j<3;j++)
12         printf("%d_",tab[i][j]);
13     printf("\n");
14 }
```

# À vos boîtiers !

6



# Sommaire

1 Tableaux

2 Chaînes de caractères

3 Pointeurs

# Chaînes de caractères (1/5)

## Convention

Une chaîne de caractères est constituée en mémoire d'une suite de caractères (octets contenant les codes des caractères) terminée par le caractère de code nul ('`\0`').

⇒ Une chaîne comportant  $n$  caractères occupera donc  $n + 1$  octets en mémoire.

## Représentation

Une chaîne de caractères est stockée dans un tableau de caractères dans lequel on met le caractère '`\0`' pour préciser la fin de la chaîne.

# Chaînes de caractères (2/5)

## Affectation

- Initialisation à la déclaration** : `char ch[4]="abc";`  
 Ici "abc" n'est pas une constante chaîne de caractères, mais une facilité d'écriture :  
`char ch[4]="abc";`  $\iff$  `char ch[4]={'a','b','c','\0'};`  
 On peut omettre la taille :  
`char ch[]="abc";` réserve 4 caractères.
- Affectation ou initialisation après la déclaration** :  
`char ch[4];`  
`ch="abc";` /\* ← instruction incorrecte ! \*/  
`ch[0]='a'; ch[1]='b'; ch[2]='c'; ch[3]='\0';` /\* ← instructions correctes \*/
- Utilisation des outils fournis par la bibliothèque standard de traitement des chaînes de caractères (`string.h`).

# Chaînes de caractères (3/5)

## Entrées-sorties de chaînes de caractères : affichage à l'écran

- Utilisation de printf avec le spécificateur de format %s :

```
char ch[]="bonjour";
```

```
printf("Voici_ma_chaine_:_%s!\n",ch);
```

produit :

```
Voici_ma_chaine_:_bonjour!↵
```

où ↵ représente le caractère « saut de ligne ».



# Démo



# Chaînes de caractères (4/5)

## Entrées-sorties de chaînes de caractères : lecture au clavier



Prévoir la place suffisante pour pouvoir stocker la chaîne !

- Utilisation de scanf avec le spécificateur %s :

```
char ch[10];
scanf("%s",ch);
```

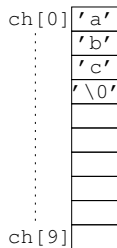


Pas de & ici !

Tapé sur le clavier :

abc\_def↵

- ' \0 ' est rajouté automatiquement.
- La lecture s'arrête au premier séparateur rencontré.



**Problème** si la chaîne tapée est trop longue (plus de 9 caractères ici).

# Chaînes de caractères (5/5)

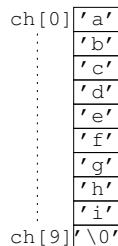
## Entrées-sorties de chaînes de caractères : lecture au clavier (suite)

- Utilisation de `scanf` avec le spécificateur `%s` et un gabarit :  
Permet de préciser le nombre maximum de caractères lus (on ne compte pas le `'\0'`) :

```
char ch[10];
scanf("%9s",ch);
```

Tapé sur le clavier :  
abcdefghijklmn↵

- Les caractères `ijklmn↵` restent dans le buffer d'entrée.



# Démo



# À vos boîtiers !

7



# Sommaire

1 Tableaux

2 Chaînes de caractères

3 Pointeurs

# Pointeurs (1/4)

## Définition

Un pointeur est une variable susceptible de contenir une adresse mémoire.

## Déclaration

```
type_pointé *identificateur;
```

## Opérateurs

- **Opérateur d'adresse :** *&expression*
  - *expression* doit désigner un objet de type quelconque qui possède une adresse mémoire.
  - L'adresse d'un objet est le numéro du premier octet de l'emplacement en mémoire que cet objet occupe.
  - L'expression *&expression* vaut l'adresse de cet objet.
- **Opérateur d'indirection :** *\*adresse*

L'opérateur d'indirection, ou de « déréréférenciation », ou de « déréréférencement », permet d'accéder à l'objet pointé :  
l'expression *\*adresse* désigne l'objet qui se trouve à l'adresse *adresse*.

## Adresses

140734669138656  
140734669138657  
140734669138658  
140734669138659

$$\begin{array}{r} 00000000 \\ - 00000000 \\ - 00000000 \\ - 00001100 \end{array}$$

x==12



# Pointeurs (2/4)

## Exemple

```
int x=12;
```

```
⋮
```

```
int *p;
```

### Adresses

140734669138640	????????
140734669138641	????????
140734669138642	????????
140734669138643	????????
140734669138644	????????
140734669138645	????????
140734669138646	????????
140734669138647	????????

p==?

140734669138656	00000000
140734669138657	00000000
140734669138658	00000000
140734669138659	00001100

x==12

## Pointeurs (2/4)

## Exemple

```

int x=12;
:
:
int *p;
p=&x;

```

Adresses		
140734669138640	00000000	p==140734669138656
140734669138641	00000000	
140734669138642	01111111	
140734669138643	11111111	
140734669138644	01010111	
140734669138645	11110110	
140734669138646	00101010	
140734669138647	11100000	
140734669138656	00000000	x==12
140734669138657	00000000	
140734669138658	00000000	
140734669138659	00001100	

# Pointeurs (2/4)

## Exemple

```
int x=12;
```

```
:
```

```
int *p;
```

```
p=&x;
```

```
*p=7;
```

### Adresses

140734669138640

00000000

140734669138641

00000000

140734669138642

01111111

140734669138643

11111111

140734669138644

01010111

140734669138645

11110110

140734669138646

00101010

140734669138647

11100000

p==140734669138656

140734669138656

00000000

140734669138657

00000000

140734669138658

00000000

140734669138659

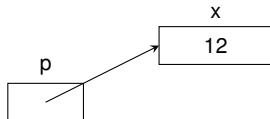
00000111

x==7

## Pointeurs (3/4)

### Représentation simplifiée

```
1 int x=12,*p;  
2 p=&x;
```



À partir de la ligne 2, on dit que :

- « p pointe sur x »  $\iff$  « p contient l'adresse de x »  
 $\iff$  « la valeur de la variable p est l'adresse de x »

### Utilisation des pointeurs

Les pointeurs sont indispensables pour :

- le passage des paramètres des fonctions
- la gestion dynamique de la mémoire

# À vos boîtiers !

8



# Pointeurs (4/4)

## Arithmétique des pointeurs

- $\text{pointeur} + \text{entier} = \text{valeur du pointeur} + (\text{entier} \times \overset{\text{octets}}{\text{taille de l'objet pointé}})$
- $\text{pointeur} - \text{entier} = \text{valeur du pointeur} - (\text{entier} \times \text{taille de l'objet pointé})$
- $\text{pointeur}_1 - \text{pointeur}_2 = (\text{valeur du pointeur}_1 - \text{valeur du pointeur}_2) / \text{taille de l'objet pointé}$

## Comparaisons de pointeurs

Pour comparer deux pointeurs, on peut utiliser les opérateurs : `==` `!=` `<` `<=` `>` `>=`

## Pointeur « nul »

La constante symbolique `NULL` vaut 0 et :

- désigne une adresse mémoire qui n'existe pas
- est définie dans `stdio.h`, `stdlib.h` et `stddef.h`

Après l'instruction `int *p=NULL;` on dit que `p` pointe sur « rien ».

# À vos boîtiers !

9



# Pointeurs et tableaux (1/3)

## Nom d'un tableau

Le nom d'un tableau est considéré comme une constante adresse qui vaut l'adresse du premier élément du tableau.

Après la déclaration `int tab[10];`

on a : `tab == &tab[0]`

## Conséquence

Accès aux éléments d'un tableau : `tab[i] == *(tab+i)`

## Exemple : parcours d'un tableau

```
int tab[10], *p, *fin;  
fin=tab+10;  
for (p=tab; p<fin; p++)  
    *p=0;
```





# Pointeurs et tableaux (2/3)



## Pointeurs et tableaux à deux indices

La déclaration :

```
int tab[3][2];
```

est équivalente à :

```
int (tab[3])[2];
```

Donc `tab[3]` est un tableau de 2 `int` et `tab` est un tableau de 3 tableaux de 2 `int`.

tab	tab[0]	tab[0][0]	
		tab[0][1]	---
tab[1]	tab[1]	tab[1][0]	
		tab[1][1]	---
tab[2]	tab[2]	tab[2][0]	
		tab[2][1]	---

On peut écrire :

```
tab[i][j] == (tab[i])[j]
           == *((tab[i])+j)
           == *(* (tab+i)+j)
```

# Pointeurs et tableaux (3/3)



## Propriétés

Si `nbc` contient le nombre de colonnes du tableau `tab` (2 dans l'exemple précédent), on peut écrire :

```
tab[i][j] == (*(tab+i)+j)
           == &tab[0][0]+i*nbc+j)
           == *(tab[0]+i*nbc+j)
```

```
tab[i] == *(tab+i)
        == &tab[i][0]
```

`tab[i]` est l'adresse du premier élément de la  $i^{\text{ème}}$  ligne.