Université Polytechnique HAUTS-DE-FRANCE

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES HAUTS-DE-FRANCE

Cours :

# Le Langage C

## Deuxième partie

v1.0

# SOMMAIRE

# I. LES TABLEAUX

## 1. Les tableaux à une dimension

### a. Exemples de déclarations

```
int tab1[5];                    // Déclaration d'un tableau de 5 entiers
flot tab2[3] = {1.1,2.2,3.3}; // Déclaration d'un tableau de 3 types char
char texte1[ ]="Bonjour";      // Déclaration d'un tableau de types char
                          // et initialisation de ce tableau avec les codes ASCII
                          // de la chaîne de caractères utilisée.
```

`int tab1[5];` Réserve l'emplacement pour 5 éléments de type int. Chaque élément est repéré par sa position dans le tableau nommé : indice.
En C, la première position porte le numéro 0. Donc ici les indices vont de 0 à 4 et le premier sera désigné par tab2[0] et le dernier tab2[4].

### b. Initialisation

```
float tab2[3] = {1.2,2.3,3.4}; // toutes les valeurs
float tab2[3] = {1.2,2.3}; // les premières
float tab2[] = {1.2,2.3,3.4}; // le compilateur va créer trois emplacements
```

### c. Remarque

- Aucun contrôle de débordement d'indice n'est mis en place ce qui peut, mais pas toujours, occasionner des erreurs.

### d. Exemple d'utilisation moyenne de 20 notes

Les tableaux utilisent souvent la boucle *for* :

```
#include <stdio.h>
int main()
{ int i, som, nbm ;
  float moy ;
  int t[20] ;

  for (i = 0 ; i < 20 ; i++)
  {
    printf ("donnez la note numéro %d : ", i + 1) ;
    scanf ("%d", &t[i]) ;
  }
  for (i = 0, som = 0 ; i < 20 ; i++) som += t[i] ;
  moy = som / 20 ;
  printf ("\n\n moyenne de la classe : %f\n", moy) ;

  for (i = 0, nbm = 0 ; i < 20 ; i++ )
    if (t[i] > moy) nbm++ ;

  printf ("%d élèves ont plus de cette moyenne", nbm) ;
  return 0;

}
```

## 2. Les tableaux à plusieurs dimensions

### a. Déclarations – exemples

En langage C, il est possible de définir des tableaux à plusieurs dimensions comme étant des tableaux de tableaux.
Exemples :

```
1. float x[10];// Une dimension
2. float point_2D[10][2];//10 points 2D avec ses coodonnées x,y
```

```
3. float point_2D[10][3];//10 points 3D avec ses coodonnées x,y,y
4. float notes[8][6];//6 notes pour 8 élèves
5.
6.  float notesElevesTrimestre [24][10][3];
7.  // les notes au maxi 10 pour les 24 élèves et pour les 3 trimestres
```

1. Déclaration d'un tableau à une dimension contenant, par exemple, les abscisses de 10 points-> **10 éléments**
2. Tableau de 10 éléments qui possède chacun 2 autres éléments. Dans l'exemple ce sont les enregistrements des coordonnées x et y de 10 points **->10*2=20 éléments**
3. Toujours un tableau à deux dimensions mais cette fois avec 3 éléments par point (x,y et z) **10*3=30 éléments**
4. Tableau de notes obtenues par des élèves à des devoirs (au maxi 6 notes et 8 élèves). **8*6=48 éléments**

6. Les notes des élèves de la classe par rapport aux devoirs pour les trois trimestres d'une année scolaire : **24*10*3=240*3=720 éléments**

### b. Accès

Par convention, les éléments déclarés dans un tableau *tab[x][y]* à 2 dimensions sont repérés par :

- *tab[lig][col]*
- où *lig* représente la ligne
- et *col* la colonne :

Le numéro de la ligne, *lig,* est compris entre 0 et x-1
Le numéro de la colonne, *col,* est compris entre 0 et y-1

| | Colonne 0 | Colonne1 | Colonne2 |
|---|---|---|---|
| Ligne 0 | **x[0][0]** | **x[0][1]** | **x[0][2]** |
| Ligne 1 | **x[1][0]** | **x[1][1]** | **x[1][2]** |
| Ligne 2 | **x[2][0]** | **x[2][1]** | **x[2][2]** |

### c. Représentation graphique

Deux dimensions
Soit : `float notes[24][10];//10 notes pour 24 élèves`

devoirs

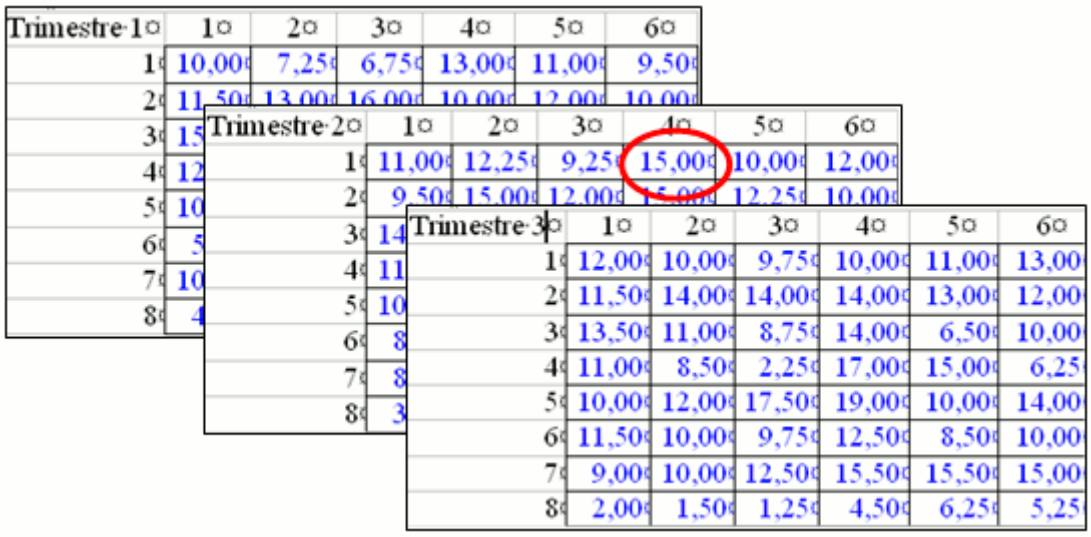
élèves

| | | | | | |
|---|---|---|---|---|---|
| 10,00 | 7,25 | 6,75 | 13,00 | 11,00 | 9,50 |
| 11,50 | 13,00 | 16,00 | 10,00 | 12,00 | 10,00 |
| 15,00 | 9,00 | 8,75 | 14,00 | 6,50 | 10,00 |
| 12,50 | 9,50 | 7,25 | 15,00 | 13,00 | 7,25 |
| 10,00 | 11,00 | 15,50 | 18,00 | 14,00 | 10,00 |
| 5,50 | 6,25 | 9,75 | 7,50 | 6,50 | 9,50 |
| 10,00 | 11,00 | 18,75 | 17,50 | 12,50 | 11,00 |
| 4,50 | 6,50 | 3,25 | 6,50 | 9,25 | 8,75 |

L'accès à la cinquième note du second élève est `notes[1][4]` donne la valeur 12.

Trois dimensions

`float notesElevesTrimestre [24][10][3];` représente les notes des élèves de la classe par rapport aux devoirs pour les trois trimestres d'une année scolaire.




`notesElevesTrimestre[0][3][1]` représente la 4ème note du premier élève au trimestre 2 = **15,00**

## d. Initialisation

### a) *Deux dimensions*

La déclaration et l'initialisation peuvent se faire par :

```
int x[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11}
```

Mais il est préférable de la faire ainsi :

```
int x[3][4] = {{0,1,2,3}, {4,5,6,7}, {8,9,10,11}};
```

Ou pour encore plus de visibilité :

```
int x[3][4] = {
    {0,1,2,3},
    {4,5,6,7},
    {8,9,10,11}
    };
```

### b) *Trois dimensions*

De même, il est possible d'écrire :

```
int x[2][3][4] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
                  11, 12, 13, 14, 15, 16, 17, 18, 19,
                  20, 21, 22, 23
                 };
```

Mais il est préférable de la faire ainsi :

```
int x[2][3][4] =
    {
        { {0,1,2,3}, {4,5,6,7}, {8,9,10,11} },
        { {12,13,14,15}, {16,17,18,19}, {20,21,22,23} }
    };
```

## a. Exemples

```
int x[2][3][2] =
{
    { {0,1}, {2,3}, {4,5} },
    { {6,7}, {8,9}, {10,11} }
};

// affiche les valeurs de tous les éléments
for (int i = 0; i <= 1; ++i)
{
    for (int j = 0; j <=2; ++j)
    {
        for (int k = 0; k <=1; ++k)
        {
```

```
L'element a x[0][0][0] = 0
L'element a x[0][0][1] = 1
L'element a x[0][1][0] = 2
L'element a x[0][1][1] = 3
L'element a x[0][2][0] = 4
L'element a x[0][2][1] = 5
L'element a x[1][0][0] = 6
L'element a x[1][0][1] = 7
L'element a x[1][1][0] = 8
L'element a x[1][1][1] = 9
L'element a x[1][2][0] = 10
L'element a x[1][2][1] = 11
```

```
                printf("L'element a x[%d][%d][%d] = %d\n",i,j,k,x[i][j][k]);
            }
        }
    }
```

# II. LES CHAINES DE CARACTÈRES

## 1. Déclaration

Une chaîne de caractères (string en anglais) est une suite de caractères stockée dans un tableau de char et terminée par le caractère '\0'.
Le '\0' marque la fin de la chaîne quelle que soit la taille du tableau mais le nombre total de caractères dans la chaîne ne peut pas dépasser la taille du tableau.
Par exemple :

```
1.    char Str1[15];
2.    char Str2[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o'};
3.    char Str3[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o', '\0'};
4.    char Str4[ ] = "arduino";
5.    char Str5[8] = "arduino";
6.    char Str6[15] = "arduino";
```

1. Str1 : déclare un tableau de char sans initialisation.
2. Str2 : déclare un tableau de char (avec un élément supplémentaire) et le compilateur va rajouter le caractère null obligatoire
3. Str3 : ajout explicite du caractère null
4. Str4 : initialise avec une chaîne de caractère entre guillemets (double), le compilateur va dimensionner le tableau pour l'ajuster à la taille de la chaîne et va ajouter le caractère null
5. Str5 : initialise le tableau avec une taille explicite et une chaîne de caractères.
6. Str6 : initialise le tableau et laisse un espace supplémentaire pour une chaîne de caractères plus importante.

## 2. Ajout du caractère nul

```
char texte1[ ]="Bonjour";     // Déclaration d'un tableau de types char
                             // et initialisation de ce tableau avec les codes
```

- L'initialisation du tableau texte1 remplit la mémoire de la manière suivante :

| 'B' | 'o' | 'n' | 'j' | 'o' | 'u' | 'r' | '\0' |
|---|---|---|---|---|---|---|---|

'B', 'o', 'n', 'j', 'o', 'u', 'r' représentant les codes ASCII respectifs des caractères du tableau. Et '\0' va être ajouté automatiquement par le compilateur.
Le caractère '\0' étant le caractère de code ASCII 0, indiquant la fin de la chaîne de caractère.
Ce tableau comprend donc non pas 7 caractères, mais 8 caractères.

## 3. Plusieurs dimensions = liste de chaînes

Il est possible de représenter une liste de chaînes de caractères à l'aide d'un tableau à deux dimensions, par exemple comme ceci :

```
char noms[][16]={"Alice", "Bob", "Samia", "Zheng-You"};
printf(noms[1]);// Affiche Bob
```

## 4. Fonctions de manipulations

Des fonctions, se trouvant dans <string.h> permettent de manipuler ces chaînes de caractères :

| Nom de la fonction | Utilité | Exemple |
|---|---|---|
| **strlen** | Renvoie la taille | `Printf("La taille est %d\n",strlen(nom))` |
| **strcpy** | Copie | `strcpy(nom2,nom);// Destination en premier, strncpy copie n caractères` |
| **strncpy** | Copie les n premiers caractères | `strncpy(nom2,nom,5);// Destination en premier, strncpy copie n caractères` |
| **strcat** | Ajoute | `strcat(nom,nom2); // Ajoute nom2 à la fin de nom` |

| | | |
|---|---|---|
| **strncat** | Ajoute n caractères | `strncpy(nom,nom2,5); // Ajoute les 5 premiers caractères de nom2 à la fin de nom` |
| **strcmp** | Compare deux chaines | `if (strcmp(nom,"Nestor")==0) // strcmp = 0-> égaux` |
| **strncmp** | Compare les n premiers caractère de deux chaines | `if (strncmp(nom,"Nestor",2)==0) // strcmp = 0-> égaux, strncmp compare les deux premiers caractères` |
| **atoi** | Converti une chaîne en int (entier) | `int entier= atoi("12");// Entier vaut 12` |
| **sprintf** | Stocke la chaîne formatée dans une autre chaîne | `int entier=12;`<br>`sprintf(formatage,"un entier = %d",entier);`<br>`// formatage[]="un entier = 12"` |

## 5. Exemple d'utilisation : une chaîne=un tableau

```c
#include <stdio.h>
#include <string.h>

int main()
{
    char texte[10];
    int indice,longueur;

    printf("texte ? ");
    scanf("%s", texte);
    longueur = strlen(texte); // longueur de la chaîne sans le '\0'
    printf("La taille du texte est de %d\n",longueur);
    for (indice = 0; indice <= longueur - 1; indice++)
        printf("indice=%d caractere=%c\n", indice,texte[indice]);

}
```

```
texte ? azer
La taille du texte est de 4
indice=0 caractere=a
indice=1 caractere=z
indice=2 caractere=e
indice=3 caractere=r
```

Dans l'exemple, *texte* est composé de cinq caractères : 'a', 'z', 'e', 'r' et '\0'. Mais *strlen(texte)* renvoie 4 car il y a 4 caractères non nuls.
L'indice commençant à 0 : texte[0]='a' jusqu'à texte[3]='r' -> l'indice va donc de 0 à strlen -1

## 6. Utilisation de tableau de char à la place des int

Pour rappel :

| | | | |
|---|---|---|---|
| **char** | 1 | -128 ($2^7$)<br>à<br>127 ($2^7$-1) | Représente un seul caractère (code ASCII) et peut être utilisé pour un « petit » entier compris entre -128 et 127. |
| **unsigned char** | 1 | 0<br>à<br>255 ($2^8$-1) | Représente un seul caractère (code ASCII) ou un entier compris entre 0 et 255. |
| **int** | 2 (µp 16 bits)<br>4 (32 bits) | -32768 (-$2^{15}$)<br>à<br>32767 ($2^{15}$-1) | Entier positif ou négatif. |
| **unsigned int** | idem | 0<br>à<br>65535 ($2^{16}$-1) | Uniquement les entiers positifs appelés « non signés » |

Utilisation de char pour économiser de la mémoire :
Au lieu d'utiliser des entiers qui vont prendre chacun 4 octets il est possible d'utiliser un char à la place qui ne va prendre qu'un seul octet (à condition que le nombre ne soit pas supérieur à 127/255)

Si la variable est comprise entre -128 et + 127, pour économiser de la mémoire on va préférer utiliser un type char :

```c
int tab2[3] = {-1,2,3}; // Déclaration d'un tableau de 3 entiers = 12 octets
char tab2[3] = {-1,2,3};      // Déclaration d'un tableau de 3 types char = 3 octets
```

Si la variable n'est uniquement positif :

```c
unsigned char tab2[3] = {1,2,3};
```

# III. LES OPÉRATEURS

## 1. Définition

| Fonctions | O | Description | Exemples |
|---|---|---|---|
| Identificateurs | () | Appel de fonction | |
| | [] | Indice de tableau | tableau[3]=5; |
| opérateurs unaires | ! | Négation logique (NOT) | b=!a; (si a>0 => b=0, si a=0 =>b=1) |
| | ~ | Complément binaire bit à bit | b=~a |
| | - | Moins unaire | b=-a; |
| | + | Plus unaire | b=+a; |
| | ++ | Préincrément ou postincrément | b=a++; (b=a puis a=a+1) |
| | -- | Prédécrément ou postdécrément | b=a--; (b=a puis a=a-1) |
| | & | Adresse de | b=&a; (b égale l'adresse de a) |
| | * | Indirection (adressage indexé) | b=*a; ( b=contenu de l'adresse de a) |
| opérateurs binaires | * | Multiplication | c=a*b; |
| | / | Division | c=a/b; |
| | + | Plus binaire | c=a+b; |
| | - | Moins binaire | c=a-b; |
| | << | Décalage à gauche | c=a<<b; (a est décalé b fois à gauche) |
| | >> | Décalage à droite | c=a>>b; (a est décalé b fois à droite) |
| | & | ET entre bits | c= a & b; (ET logique bit à bit) |
| | ^ | OU exclusif entre bits | c= a ^b; |
| | \| | OU entre bits | c= a\|b; |
| Tests | < | Strictement inférieur | if a < b |
| | <= | Inférieur ou égal | if a >= b |
| | > | Strictement supérieur | if a > b |
| | >= | Supérieur ou égal | if a >= b |
| | == | Egal | if a ==b (si a est égale à b) |
| | != | Différent | if a != b |
| | && | ET logique | if ((a=5) && (b=2)) |
| | \|\| | OU logique | if ((a=5) \|\|(b=2)) |
| | ?: | Condition | z=(a>b)?a:b (Si a>b a z=a sinon z=b) |
| Affectation | = | Affectation simple | a=b; (a prend la valeur de b) |
| Auto-affectations | *= | Affectation produit | a*=2 (a=a*2) |
| | /= | Affectation quotient | a/=2 (a= a/2) |
| | %= | Affectation reste | a%=2 (a= le reste de a/2) |
| | += | Affectation somme | a+=2 (a=a+2) |
| | -= | Affectation différence | a-=2 (a=a-2) |
| | &= | Affectation ET entre bits | a&=5 (a=a&5) |
| | ^= | Affectation OU EX entre bits | a^=5 (a=a^5) |
| | \|= | Affectation OU entre bits | a\|==5 (a=a\|=5) |
| | <<= | Affectation décalage gauche | a<<=5 (a=a<<5) |
| | >>= | Affectation décalage droite | a>>=5 (a=a>>5) |

## 2. Exemple

```
#include <stdio.h>
char a, b, c;
int i1, i2, i3, i4; long l1, l2; float f1, f2, f3; double d1;

main()
```

```
{
i1 = i2 = i3 = 11;      /* voila des exemples d'affectations multiples */
a = b = c = 40;
f1 = f2 = f3 = 12.987;

/* Opérations arithmétiques de base */
i3 = -i2;   /* donne i3 = -11 */
i3 = i1 + i2;     /* donne i3 = 22 */
i3 = i1 - i2;     /* donne i3 = 0 */
i3 = i1 * i2;     /* donne i3 = 121 */
i3 = i1 / i2;     /* donne i3 = 1 */
i3 = 15+(i1 = 4*i2);          /* donne i1= 44 puis i3 = 59 */
i4 = i3 / i2;     /* donne i4 = 5 */
i4 = i3 % i2;     /* % = modulo donc i4 = 8 */
i4 = i3 >> 2;     /* i4 = i3 apr,s 2 décaléges … droite (*4) */
i4 = i3 << 3;     /* i4 = i3 apr,s 3 décaléges … gauche (/8) */

/* Opérations arithmétiques de niveau 2 : automodifications */
a = a + 1;  /* incrémentation de a */
a += 1;     /* idem */
a++;  /* idem APRES utilisation de a */
++a;  /* idem AVANT utilisation de a */
a = a - 1 ; /* décrémentation de a */
a -= 1;     /* idem */
a = 40;
b = a--;    /* b = 40 puis a = 39 */
b = --a;    /* a = 38 puis b = 38 */
a += 10;    /* a+10, a = 48 */
b -= 5;     /* b-5, b = 33 */
f1 *= 1.5;  /* f1 * 1.5, f1 = 8.658 */
f2 /= 2.2;  /* f2 / 2.2, f2 = 5.9032 */
a <<= 1;    /* a décalé 1 fois à gauche a = 24 */
b >>= 2;    /* b décalé 2 fois à droite b = 132 */

/* Opérateurs Logiques de niveau 1 */
b = 15; c = 12;
a = b & c;  /* a = b ET c donc a = 10 */
a = b | c;  /* a = b OU c donc a = 17 */
a = b ^ c;  /* a = b OU EXCLUSIF c donc a = 7 */
a = ~b;     /* a = NOT b donc a = $EF */

/* Opérations de niveau 3 : Affectation conditionnelle */
/* syntaxe générale : (Test) ? action_si_oui : action_si_non; */
i1 = (i2 > 3) ? 2 : 10;  /* Cette instruction doit se lire:
si i2 est supérieur à 3 alors i1 devient 1 sinon, i1 devient 10 */
c = (a > b) ? a : b;    /* c est le maxi d'entre a et b */
c = (a > b) ? b : a;    /* c est le mini d'entre a et b */

c = (a >= 0) ? a : -a;  /*  c est la valeur absolue de a */

/* Opérateurs divers */
a = sizeof (f1);  /* sizeof retourne la taille en octet de */
b = sizeof (l1);  /* l'expression paramètre */
f2 = int(15) / int(4);  /* conversion de type : CAST. En absence des */
f2 = int (15/4);  /* commandes de cast, f2 vaudrait 3.75, ici 3 */

/* Comparaisons logiques de base */
if (i1 == i2) i= -13;   /* puisque i1 = i2, i3 deviendra -13 */
if (i1 > i3) a = 'A';         /* donne 'A' (65) … la variable a */
if (!(i1 > i3)) a = 'B';      /* puisque i1 >i3, a ne change pas */
if (b <= c) f1 = 0.0;   /* puisque b = c alors f1 devient 0 */
if (f1 != f2) f3 = i3/2;      /* comme f1 <> f2, f3 devient 6.5 */

/* Comparaisons logiques de niveau 2 */
/* bas,es sur le fait qu'une valeur vraie est diff,rente de 0 et qu'une valeur fausse est
nulle */
if (i1 = (f1 != f2)) i3 = 111;      /* comme f1 <> f2, le résultat du test
```

```
est vrai, le résultat logique du test vrai est affecté à i1 et i3 change */
if (i1) i3 = 222;       /* i1 est vrai donc i3 change */
if (i1 != 0) i3 = 333;  /* i1 vrai donc non nul donc i3=333 */
if (i1 = i2) i3 = 444;  /* i2 recoit i1, cette valeur non nulle
donc vraie  i3 = 222 */

/* Comparaisons logiques niveau 3 : Tests multiples */
i1 = i2 = i3 = 77;
if ((i1 == i2) && (i1 == 77)) i3 = 33;    /* && = ET logique: i3 change */
if ((i1 > i2) || (i3 > 12)) i3 = 22;      /* || = OU logique: i3 change */
if (i1 && i2 && i3) i3 = 11;  /* tous non nuls donc i3 change */
if ((i1=1)&&(i2=2)&&(i3=3)) i3 = 44;      /* Attention, affectations des
variables donc i3 change */
if ((i1==2)&&(i2=3)) i3 = 55; /* i1 <> 2 donc i3 ne change pas */
}
```