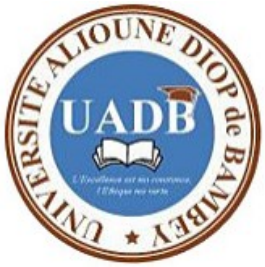
UNIVERSITÉ ALIOUNE DIOP DE BAMBEY
UFR SCIENCES APPLIQUÉES ET TECHNOLOGIE DE L'INFORMATION ET DE LA COMMUNICATION
DÉPARTEMENT
INFORMATIQUE



Chapitre 4 - TYPES DE DONNEES COMPOSÉS

Programmation en LANGAGE C

AMRT 2

2016-2017

A partir des types de base (caractère, entier, réel), on peut créer de nouveaux types, appelés ***types composés***, qui permettent de représenter des ensembles de données organisés.

I. Les énumérations

Une énumération est un type permettant de définir un ensemble de constantes, parmi lesquelles les variables de ce type prendront leur valeur.

Pour déclarer une variable de type énuméré, il faut d'abord définir le type.

I-1. Définition d'un type énuméré

En C, on définit un type énuméré en utilisant le mot-clé **enum** selon la syntaxe suivante :

enum nom_type { constante1, constante2, …, constanteN };

Exemples

enum couleur {bleu, blanc, rouge, vert, jaune, noir} ;

enum jour {Lundi, Mardi, Mercredi, Jeudi, Vendredi, Samedi, Dimanche} ;

Chaque constante est associée à un entier (son numéro d'ordre en partant de 0 pour le premier). Mais on peut aussi spécifier la valeur associée.

Exemple:

```
enum jour {Lundi=1, Mardi, Mercredi=5, Jeudi, Vendredi, Samedi,
        Dimanche};
```

Mardi prend la valeur 2, Jeudi la valeur 6,...

Les valeurs associées aux constantes ne sont pas modifiables dans le programme.

Après avoir défini un type énuméré, on peut l'utiliser pour déclarer une ou plusieurs variables de ce type.

Exemples:

```
enum jour j1;
enum jour j2 = Mardi;
```

Exemple

Voici un programme qui manipule un type énuméré pour les jours de la semaine :

```
#include <stdio.h>
#include <conio.h>
enum jour {Lundi=1,Mardi,Mercredi=5,Jeudi,Vendredi,Samedi,Dimanche};
main()
{  enum jour un_jour;
   printf("Donnez un numéro de jour entre 1 et 7") ;
   scanf("%d", &un_jour) ;
   switch(un_jour )
   {
     case 1 :
      printf("C'est Lundi"); break ;
      case 2 :
      printf("C'est Mardi"); break ;
      case 5 :
      printf("C'est Mercredi"); break ;
       default :
       printf("Ce n'est ni Lundi ni Mardi ni Mercredi");
    }
getch();
}
```

II. Les tableaux

Un tableau permet de rassembler sous un même nom un nombre fini d'éléments ayant **tous le même type**.

II-1. Tableaux à une dimension

Un tableau à une dimension peut être vu comme une liste d'éléments.

On le représente souvent comme une suite de **cases** contenant chacune une valeur.

Exemple: Tableau salaire

10000	8500	12300	13000	6500	9800
1ère case	2ème case	3ème case	4ème case	5ème case	6ème case

Un tableau possède un **nom** (ici salaire) et un nombre d'**éléments** (de cases) qui représente sa **taille** (ici 6).

La notion de « case contenant une valeur » doit faire penser à celle de variable. Et, en effet, les éléments d'un tableau correspondent à des emplacements contigüs en mémoire et sont, de ce fait, des **variables**.

Le tableau lui-même constitue aussi une variable. C'est une **variable composée** (par opposition aux variables simples) car il est constitué d'autres variables (les éléments du tableau) appelées variables indicées.

Un tableau à une dimension est parfois appelé vecteur.

II-1-1. Déclaration d'un tableau à 1 dimension

La syntaxe de la déclaration d'une variable tableau est la suivante:

type *nom_tableau* **[** *N* **];**

où
nom_tableau est l'identificateur du tableau
N est la taille du tableau et doit être une constante entière et positive.

Exemple :

```
int tab[10] ;   //déclare une variable tableau de 10 entiers appelée tab.
```

Cette déclaration réserve un emplacement mémoire permettant de stocker 10 entiers .
tab[0] désigne le premier élément du tableau tab
tab[9] désigne le dernier élément du tableau tab

On peut initialiser un tableau lors de sa déclaration :

Exemple :

```
int T[] = {4,5,8,12,-3};          /* déclaration et initialisation du tableau T de
taille fixée à 5. On peut mettre le nombre d'éléments à l'intérieur des crochets, mais dans ce cas, ce n'est pas obligatoire.*/
```

Un élément particulier du tableau est désigné en précisant son **indice** (son numéro).

Le premier élément du tableau correspond à l'indice 0 et est désigné par *nom_tableau*[0].

Le deuxième élément du tableau correspond à l'indice 1 et est désigné par *nom_tableau*[1].

...

Le dernier élément du tableau correspond à l'indice N-1 et est désigné *nom_tableau*[N-1].

Exemple: Tableau salaire

10000	8500	12300	13000	6500	9800
0	1	2	3	4	5

Pour désigner un élément, l'indice peut être écrit sous forme de :

- une valeur, exemple: salaire[4]
- une variable, exemple: salaire [k], avec k de type entier
- une expression entière, exemple: salaire [k+1], avec k de type entier

Quelque soit sa forme, la valeur de l'indice doit être entière et comprise entre les valeurs minimale et maximale déterminées à la déclaration du tableau.

Par exemple, avec le tableau salaire, il n’est pas correct d'écrire salaire[10] ou salaire[15]. Ces expressions font référence à des éléments qui n'existent pas.

II-1-2. Manipulation d'un tableau à 1 dimension

Les éléments d'un tableau sont des variables appelées variables indicées qui s'utilisent exactement comme n'importe quelle autre variable classique. Autrement dit, elles peuvent faire l'objet d'une affectation, elles peuvent figurer dans une expression arithmétique, dans une comparaison, elles peuvent être affichées et saisies…

On ne peut pas manipuler un tableau de manière globale(affectation à un autre tableau, affichage, saisie…). Il faut le manipuler élément par élément (en le parcourant avec une boucle **Pour** par exemple).

II-2. Tableaux à plusieurs dimensions

On peut aussi avoir un tableau à plusieurs dimensions.

Dans ce cas, il faut préciser plusieurs indices pour désigner un élément.

Un tableau à plusieurs dimensions se déclare en ajoutant une paire de crochets et une taille pour chaque dimension.

Par exemple, la syntaxe de la déclaration d'une variable tableau à deux dimensions est la suivante:

type *nom_tableau*[N1][N2];

Exemple

```
double m[10][20];        /* m est une matrice de réels */
```

m[0][0] désigne l'*élément à la ligne 0, colonne 0*

m[9][19]; désigne l'*élément à la ligne 9, colonne 19*

Un tableau à deux dimensions est aussi appelé matrice.

Exemple

```
float tab[3][2] ;   /*matrice de 3 lignes et deux
                      colonnes  */
```

L’accès à un élément se fait en précisant un indice entre crochets pour chaque dimension.

Le premier indice de chaque dimension est 0.

	0	1
0	2	1
1	- 4.5	12
2	3	9

tab[0][0] désigne l'élément du tableau tab situé à la ligne 0 et à la colonne 0

tab[2][1] désigne l'élément du tableau tab situé à la ligne 2 et à la colonne 1

Remarque

Pour **parcourir tous les éléments d’un tableau à deux dimensions,** on peut utiliser **deux boucles Pour imbriquées :** la première boucle pour une dimension et la deuxième pour l’autre dimension.

II-3. Les chaînes de caractères

Une chaîne de caractère est gérée en langage C comme un tableau contenant des caractères mais avec la particularité que la dernière case du tableau utilisée pour la chaîne contient le caractère spécial \0 appelé caractère nul. Ce caractère représente la fin de la chaîne.

II-3-1. Déclaration d'une chaîne

Une chaîne de caractère peut être déclarée comme un tableau de caractères par l'instruction suivante :

char nom_chaîne[N] ;(Elle est dans ce cas limitée à N-1 caractères (+ '\0')).

Exemple

Pour déclarer une chaîne de 5 caractères par exemple, il faut utiliser un tableau de taille 6 pour avoir de la place pour le caractère nul en plus des 5 caractères.

```
char ch[6] ; //la chaîne ch est dans ce cas limitée à 5 caractères (+ \0).
```

Comme pour les tableaux numériques, on peut initialiser un tableau de caractères (ou chaîne de caractères) lors de sa déclaration :

Exemple

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

'b'	'o'	'n'	'j'	'o'	'u'	'r'	\0

le compilateur réserve un tableau de 8 octets (7 octets pour bonjour + 1 pour le caractère de fin de chaîne \0).

On peut aussi écrire:

```
char ch[] = {'b','o','n','j','o','u','r'};
```

II-3-2. Fonctions de manipulation de chaînes

La bibliothèque standard **string.h** fournit des fonctions de manipulations des chaînes de caractères.

- Après la déclaration, l'affectation d'une chaîne se fera par la fonction **strcpy**.

Exemple

```
strcpy(ch, ''hello'') ;
```

- La longueur (le nombre de caractères) d'une chaîne est donnée par la fonction **strlen**.

Cette fonction ne prend pas en compte le caractère \0.

Exemple

```
strlen(''bonjour'')        retourne 7.
```

– La concaténation de deux chaînes se fait par la fonction **strcat**.

Exemple

strcat(ch, ''world'') ; la chaîne contient alors ''hello world''

- La comparaison de deux chaînes se fait par la fonction **strcmp**.

Exemple :

n = **strcmp**(ch1, ch2) ;

n vaudra :

- un nombre négatif si ch1 < ch2 au sens syntaxique
- 0 si ch1 et ch2 sont identiques
- un nombre positif si ch1 > ch2

III. Les structures

Une structure permet de désigner sous un seul nom un ensemble d'éléments pouvant être de types différents. Chaque élément de la structure, appelé champ ou membre, est désigné par un identificateur.

Les variables de type structure sont aussi appelées structures.

III-1. Déclaration d'un type structure

En C, on définit un type structure en utilisant le mot-clé **struct** selon la syntaxe suivante :

```
struct nom_type
    { Type1 nom_champ1;
      ...
     TypeN nom_champN;  };
```

Exemple

```
struct complexe
        {
        float réelle;
        float imaginaire;
        };
```

permet de définir le type structure appelé struct complexe.

Après avoir défini un type structure, on peut l'utiliser pour déclarer des variables de ce type.

Exemple

```
struct complexe c;
```

On peut accéder aux champs d'une variable structure en utilisant l'opérateur . (point)

On écrit le nom de la variable suivi de **.** et du nom du champ.

Exemple

```
c.réelle = 0;
c.imaginaire = 1;
```

Remarques

- On peut déclarer une variable structure sans avoir défini au préalable le type structure. Pour cela, on utilise la syntaxe suivante:

```
struct nomStructure
{
    Type1 champ1;
    Type2 champ2;
     ...
} nomVariable;
```

- Les règles d'initialisation d'une structure lors de sa déclaration sont les mêmes que pour les tableaux. On écrit par exemple :

```
struct complexe z = {2. , 2.};
```

Exemple

```
struct complexe z1 = {2. , 2.};
struct complexe z2;
z2 = z1;
```

III-2. Les opérations sur les structures

- On accède aux différents champs d'une structure grâce à l'opérateur *point*, noté **"."**.

Par exemple, le champ appelé champ1 d'une variable structure x est désigné par l'expression *x.champ1*

- On peut effectuer sur le champ d'une structure toutes les opérations valides sur des variables de même type que ce champ.

- On peut appliquer l'opérateur d'affectation à une structure **(à la différence d'un tableau)**. Cela permet de copier tous les champs de la structure.

IV. Compléments sur les types en langage C

IV-1. Utilisant du mot clé typedef

Pour alléger l'écriture des programmes, on peut donner un nouvel identificateur à un type existant à l'aide du mot clé **typedef**.

- Pour les types de base ainsi que les énumérations et les structures, la syntaxe est la suivante :

typedef type synonyme;

Exemple 1

```
enum couleur {bleu, blanc, rouge, vert, jaune, noir} ;

typedef  enum couleur  couleur ;/*couleur devient le synomyme de  enum
          couleur */
main()
{ couleur c ;                    // c est une variable de type couleur
...
}
```

Exemple 2

```
struct complexe
   {
      double reelle;
      double imaginaire;
   };

typedef struct complexe complexe;   /*complexe est synonyme de

                                                    struct complexe
                    */
 main()
{
  complexe z;
   ...
}
```

- Dans le cas des tableaux, on peut définir un type tableau de la même manière qu'on déclare une variable tableau mais en écrivant d'abord le mot typedef.

Exemple

```
typedef int tab[10]; /* tab est un "type" tableau
                        de  10 entiers  */

tab T;  // T est une variable tableau de 10 entiers
```