

Cours 2 - C

Imad Kissami

Université Mohammed VI Polytechnique - Licence S3

28 Octobre 2020

Les énumérations

- Les énumérations sont des types définissant un ensemble de constantes qui portent un nom que l'on appelle énumérateur.

```
// la couleur peut etre rouge (valeur 0),  
jaune (valeur 1), verte (valeur 20), ...  
enum color {  
    rouge,  
    jaune,  
    verte = 20,  
    bleu  
};  
// La constante d est 0, e =1, f= 2, et g = 4  
enum { d, e, f, g =f+2 };
```

Les énumérations

Exemple -suite:

```
int main()
{
    enum color col;
    col = verte;
    printf("%d_%d_%d_%d_\n", col, verte, rouge, g);
}
```

Résultat:

20 20 0 4

Constructeurs homogènes

Des objets plus complexes peuvent être formés à l'aide des constructeurs homogènes :

- les constructeurs de pointeurs ;
- les constructeurs de vecteur ;
- les constructeurs de fonction.

Symbole	Objet construit
*	pointeur
[]	vecteur
()	fontion

- Exemple:

```
char lignes[100]; // vecteur de 100 caractères
int *p_entier ; // pointeur d'entier
double fonc(); // fonction retournant un réel double précision .
```

Constructeurs homogènes

Ces constructeurs peuvent se combiner entre eux, permettant ainsi de définir des objets encore plus complexes.

```
char *chaines[100];  
int mat[100][40];  
char ** argv ;
```

Le constructeur homogène "*" est moins prioritaire que les deux autres. De ce fait, les déclarations précédentes permettent de définir respectivement :

- un vecteur de 100 pointeurs de caractère ;
- un vecteur de 100 éléments, chaque élément étant un vecteur de 40 entiers ;
- un pointeur de pointeur de caractère.

Constructeurs homogènes : "struct"

Les constructeurs hétérogènes permettent de définir des objets renfermant des entités de nature différente. Ce sont : les structures ; Les structures permettent de regrouper des objets dont les types peuvent être différents. La syntaxe générale est la suivante :

```
struct [ nom ] {  
    <liste-de-déclarations>  
};
```

Les objets regroupés sont les membres ou composantes de la structure les contenant.

Remarques :

- les structures sont un exemple de définition de nouveaux types ;
- lors de la définition d'une structure des objets peuvent être déclarés et seront du type associé à celle-ci.

Constructeurs homogènes : "struct"

Exemple 1:

```
#include "stdio.h"
#include "stdlib.h"

struct cellule{
    char a;
    char b;
};

int main(){

    int a = 12;
    struct cellule cel;
    cel.a = 'b';

    printf("%d_%c\n", a, cel.a);

    return 0;
}
```

Résultat:

12 b

Constructeurs homogènes : "struct"

Exemple 2:

```
struct {  
    char c;  
    unsigned int i;  
    float tab [10];  
    char *p ;  
} a , b;  
struct cellule {  
    char **p ;  
    int *t[10];  
    int (*f)();  
};  
  
struct cellule cel1 , *cel2 ;  
  
struct cellule cel[15];  
  
struct boite {  
    struct cellule cel1 ;  
    struct cellule *cel2 ;  
    struct boite *boite_suivante ;  
    int ent ;  
} b1 , b2 , *b3 ;
```


Définition de types

Il existe plusieurs manières de se définir de nouveaux types :

- au moyen des constructeurs hétérogènes comme struct ;
- au moyen du constructeur typedef ;
- au moyen d'expressions de type.

A la différence des constructeurs hétérogènes qui créent de nouveaux types, le constructeur typedef permet seulement de donner un nouveau nom à un type déjà existant :

Exemple 2:

```
typedef long size_t ;
typedef unsigned long Cardinal ;
typedef char * va_list ;
typedef _Complex float Complexe ;
typedef int Matrice [10][20];

Complexe c1 , *c2 ;
Cardinal nombre ;
va_list arg ;
size_t dimension ;
Matrice tab , *ptr_mat ;

typedef struct cellule {
    Cardinal n;
    struct cellule *ptr_suivant ;
} Cellule ;

Cellule cel1 , * cel2 ;
```

Les tableaux

- **Définition :**

Comme tous les langages, C permet d'utiliser des tableaux. On nomme ainsi un ensemble d'éléments de même type (en nombre déterminé) désignés par un identificateur unique ; chaque élément est repéré par un indice précisant sa position au sein de l'ensemble.

- **Exemple :** `float tab[10]` : // déclaration d'un tableau de 10 éléments de type float.

Nous souhaitons déterminer, à partir de vingt notes d'élèves (fournies en données), combien d'entre elles sont supérieures à la moyenne de la classe.

Les tableaux

Exemple :

```
#include "stdio.h"

int main(){

    float moy =0;
    int som =0;
    int nbm =0;
    int a = 4;

    float t[] = {12, 15, 17, 11};

    for (int i=0; i<a ; i++) som += t[i] ;
    moy = som / a ;
    printf(" Moyenne_de_la_classe : %d\n" ,moy);

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

    printf(" %d_eleves_ont_plus_de_cette_moyenne"\n" ,nbm);
    return 0;
}
```

Les tableaux

Résultat :

```
Moyenne de la classe : 13
2 eleves ont plus de cette moyenne
```

Les tableaux à plusieurs indices

- **Définition** : Comme la plupart des langages, C autorise les tableaux à plusieurs indices (on dit aussi à plusieurs dimensions). La déclaration : `int tab [5][3];`
- **Arrangement en mémoire** : Les éléments d'un tableau sont rangés suivant l'ordre obtenu en faisant varier le dernier indice en premier.
`t[0][0] -> t[0][1] -> t[0][2] -> t[1][0] -> t[1][1] -> t[1][2]`

Initialisation des tableaux

- **Initialisation :**

- `int tab[5] = { 10, 20, 5, 0, 3 } ;`
- `int tab [3] [4] = { { 1, 2, 3, 4 } ,
 { 5, 6, 7, 8 } ,
 { 9,10,11,12 } }`
- `int tab [3] [4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 } ;`

Pointeurs :

Exemple :

```
int *ad1, *ad2, *ad ;  
int n = 10, p = 20 ;
```

```
ad1 = &n ;  
ad2 = &p ;
```

*ad1 = *ad2 + 2 ;	→	n = p + 2
*ad1 += 3	→	n = n + 3

Pointeurs :

Pointeur nullptr :

```
int main {  
f(0); // Entier  
f(NULL); // NULL est convertit en entier  
}
```

Résultat :

```
Avec un entier  
Avec un entier
```


Arithmétique des pointeurs

- Exemple:

```
float tab[4] = { 13.45 , 4.56, 1.2, 67.4 }; // tab est l'adresse du tableau
float *p1 = tab ; // p1 est un pointeur qui pointe sur le 1er élément du tableau
for (int i=0 ; i<4 ; i++) // Boucle d'affichage des flottants
{ printf("%f , ", p1); // Affichage du flottant pointé par p1
  p1 = p1 + 1; // p1 pointe sur le flottant suivant
}
```

