ALGORITHMIQUE AVANCÉE ET PROGRAMMATION

CHAP 6: LES STRUCTURES

Dr. Ikbal Chammakhi Msadaa

Classes: 1 TA

Introduction

- Nous avons déjà vu comment le tableau permettait de désigner sous un seul nom un ensemble de valeurs de même type, chacune d'entre elles étant repérée par un indice.
- La structure, quant à elle, va nous permettre de désigner sous un seul nom **un ensemble de valeurs pouvant être de types différents**. L'accès à chaque élément de la structure (nommé *champ*) se fera, cette fois, non plus par une indication de position, mais par son nom au sein de la structure.

Les types synonymes

- Il est possible grâce au mot-clé **typedef** de définir un synonyme pour un type déjà existant. Ainsi la définition suivante :

typedef int entier ;

- définit un nouveau type appelé entier ayant les mêmes caractéristiques que le type prédéfini int. Une fois cette définition réalisée, nous pouvons utiliser ce nouveau type pour définir des variables et nous pouvons mélanger les variables de ce type avec des variables entières dans des expressions.

```
typedef int entier;
entier e1=23, e2=5, te[7]={1,2,3,4,5,6,7};
int i;
i = e1 + e2;
te[3] = i - 60;
```


Déclaration de structures

- Une structure est un objet composé de plusieurs champs qui sert à représenter un objet réel ou un concept. Par exemple une voiture peut être représentée par les renseignements suivants : la marque, la couleur, l'année, etc.
- Nous pouvons définir une structure ainsi :
- **Solution 1 :**

```
struct nom_de_la_structure {
/* Définition de la structure */
} ;
```

- Ceci fait, le nouveau type de données sera: **struct nom_de_la_structure** et nous pourrons déclarer une variable ainsi :

```
struct nom_de_la_structure nom_variable;
```

- Cependant, la répétition du mot-clé **struct** est peu appréciée. Nous préférerons donc souvent la syntaxe suivante.
- **Solution 2 :**

```
typedef struct {
/* Définition de la structure */
} nom_du_type;
```

- Cette fois-ci, le nouveau type de données s'appelle **nom_du_type** (nous avons créé la structure et en même temps nous avons défini un synonyme avec **typedef**).
- Nous déclarerons une variable ainsi :

```
nom_du_type nom_variable;
```

Utilisation des champs de structures

Exemple:

```
#define LONGUEUR 40
struct personne{
   char  nom [LONGUEUR];
   char  prenom [LONGUEUR];
   int   age;
};
struct personne p;
```

```
#define LONGUEUR 40
typedef struct {
   char  nom [LONGUEUR];
   char  prenom [LONGUEUR];
   int   age;
} personne;
personne p;
```

- La seconde solution est plus simple et plus élégante à l'usage.
- L'accès aux éléments d'une structure, que nous appelons aussi champ, se fait selon la syntaxe :

nom_de_variable.nom_du_champ

Utilisation des champs de structures

Exemple:

```
#include <stdio.h>
typedef struct {
   char  nom [40];
   char  prenom [20];
   int   age;
} personne;
int main () {
   personne p;
   printf("Veuillez entrer le nom de la personne:");
   scanf("%s",p.nom);
   printf("Veuillez entrer le prénom de la personne:");
   scanf("%s",p.prenom);
   printf("Veuillez entrer l'âge de la personne:");
   scanf("%d",&p.age);   /* ne pas oublier le & !!! */
   printf("Voici les caractéristiques de cette personne:\n");
   printf("nom=%s\n",p.nom);
   printf("prenom=%s\n",p.prenom);
   printf("age=%d\n",p.age);
   return 0;
}
```

Utilisation globale de structures

- Exemple:

```
typedef struct {
        int numero ;
        int qte ;
        float prix ;
} article ;
article  art1,  art2 ;
```

- Il est possible d'affecter à une structure le contenu d'une structure définie à partir du même modèle.
- Par exemple, si les structures art1 et art2 ont été déclarées suivant le modèle article défini précédemment, nous pourrons écrire :

art1 = art2 ;

Initialisation de structures

- En l'absence d'initialisation explicite, les structures de classe statique sont, par défaut, initialisées à zéro ; celles possédant la classe automatique ne sont pas initialisées par défaut (elles contiendront donc des valeurs aléatoires).
- Voici un exemple d'initialisation de notre structure art1, au moment de sa déclaration :

article art1 = { 100, 285, 2000 } ;

Structure comportant des tableaux

```
typedef struct {
        char nom[30] ;
        char prenom [20] ;
        float heures [31] ;
} employe;
employe emp;
```

- On peut, par exemple, imaginer que ces structures permettent de conserver pour un employé d'une entreprise les informations suivantes:
  - nom ;
  - prénom ;
  - nombre d'heures de travail effectuées pendant chacun des jours du mois courant.

- Que représentent emp.heures[4], emp.nom[0], &emp.heures[4] ?
-
-
-
-

employe emp = {"Makni", "Ridha", { 8, 7, 8, 6, 8, 0, 0, 8}};

Structures comportant d'autres structures

- Il est possible qu'ne structure comporte d'autres structures.
- Par exemple, on introduit deux attributs de type **struct date** à la structure **employe.**

```
struct date
{
    int jour;
    int mois;
    int annee;
};

struct employe
{
    char nom[30];
    char prenom[20];
    float heures[31];
    struct date date_embauche;
    struct date date_poste;
};
```

- date_embauche désigne la date d'embauche de l'employé.
- date_poste désigne la date de nomination au dernier poste occupé.
- Si on déclare:

struct employe emp;

Alors:
emp.date_embauche.annee désigne l'année d'embauche de emp.

Il est possible de faire des affectations globales du genre:

emp.date_poste = emp.date_embauche;

Transmission de la valeur d'une structure

```c
#include <stdio.h>

struct point
{
    int x;
    int y;
};

void fct(struct point);

int main()
{
    struct point p;
    p.x = 5;
    p.y = 7;

    printf("les positions avant l'appel de fct sont x= %d et y=%d\n", p.x, p.y);
    fct(p);
    printf("les positions apres l'appel de fct sont x= %d et y=%d\n", p.x, p.y);
    return 0;
}

void fct(struct point pt)
{
    pt.x = 0;
    pt.y = 0;
    printf("les positions dans fct sont x= %d et y=%d\n", pt.x, pt.y);
    return ;
}
```


Transmission de l'adresse d'une structure : l'opérateur ->

```c
#include <stdio.h>

struct point
{
    int x;
    int y;
};

void fct(struct point*);

int main()
{
    struct point p;
    p.x = 5;
    p.y = 7;

    printf("les positions avant l'appel de fct sont x= %d et y=%d\n", p.x, p.y);
    fct(&p);
    printf("les positions apres l'appel de fct sont x= %d et y=%d\n", p.x, p.y);
    return 0;
}

void fct(struct point *pt)
{
    pt->x = 0;
    pt->y = 0;
    printf("les positions dans fct sont x= %d et y=%d\n", pt->x, pt->y);
    return ;
}
```