Chapitre III

Les structures

I. Concepts

- Une structure est une collection de plusieurs variables (champs) groupées ensemble pour un traitement commode
- Les variables d'une structure sont appelées ***membres*** et peuvent être de n'importe quel type, par exemple des tableaux, des pointeurs ou d'autres structures

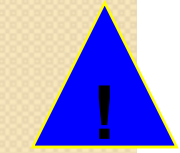
```
struct Membre
{
    char    nom[80];
    char    adresse[200];
    int     *numero;
    float   amende[10];
};
```

Les étapes sont:

- déclarer le type de la structure
- utiliser ce type pour créer autant d'instances que désirées
- Accéder aux membres des instances

II. Déclaration des structures

- Les structures sont définies en utilisant le mot-clé **struct**



```
struct Date
{
    int  jour;
    int  mois;
    int  an;
};
```

```
struct Membre
{
    char            nom[80];
    char            adresse[200];
    int             numero;
    float           amende[10];
    struct  Date    emprunt;
    struct  Date    creation;
};
```

```
struct Livre
{
    char  titre[80];
    char  auteur[80];
    float  prix;
};
```

```
struct Pret
{
    struct      Livre    b;
    struct      Date     due;
    struct      Membre  *who;
};
```

II.1 Déclaration des instances

- Une fois la structure définie, les instances peuvent être déclarées
- Par abus de langage, on appellera structure une instance de structure

```
struct Date
{
    int   jour;
    int   mois;
    int   an;
} hier, demain;

struct Date paques;
struct Date semaine[7];

struct Date nouvel_an = { 1, 1, 2001 };
```

Déclaration avant ';' .

Initialisation .

III. Accéder aux champs d'une structure

- Les champs sont accédés par le nom de l'instance, suivi de "." suivi du nom du champ

struct Membre m;

```
printf("nom = %s\n", m.nom);
printf("numéro de membre = %d\n", m.numero);

printf("amendes: ");
for(i = 0; (i < 10) && (m.amende[i] > 0.0); i++)
    printf("%.2f Euros", m.amende[i]);

printf("\nDate d'emprunt %d/%d/%d\n", m.emprunt.jour,
                    m.emprunt.mois, m.emprunt.an);
```

III. I Affectation des structures

- L'opération d'affectation = peut se faire avec des structures
- Tous les champs de la structure sont copiés (aussi les tableaux et les sous-structures)

```
struct Membre    m = {
      "Arthur Dupont",
      .....
};

struct Membre temp;

temp = m;
```

III.2 Quand la structure est un pointeur !

Utiliser p->name

- L'écriture p->name est synonyme de (*p)->name, où p est un pointeur vers une structure

```
void    affiche_membre (struct Membre *p)
{
    printf("nom = %s\n", p->nom);
    printf("adresse = %s\n", p->adresse);
    printf("numéro de membre = %d\n", p->numero);

    printf("\nDate d'affiliation %d/%d/%d\n",
        p->creation.jour, p->creation.mois, p->creation.an);
}
```

III. Déclaration de types synonymes

La déclaration d'une variable se fait selon le format suivant :

<type> <nom_de_variable>;

Supposons qu'un programme comporte un grand nombre de déclarations de variables utilisant des noms de types longs comme : struct etudiant etud1;

struct produit p1;

Ce type de déclaration peut être simplifié par la déclaration typedef qui permet de définir des types synonymes aux types utilisés.

typedef <type> <nom_synonyme_1> <nom_synonyme_2> ... ;

Exemple

```
struct produit
{
int numero;
int qte;
double prix;
};
struct produit art1, art2;
```

Ce type de structure peut être redéfini de 2 manières :

1ère manière :

```
struct produit
{
int numero;
int qte;
double prix;
};
typedef struct produit pro;
```

2ème manière :

```
typedef struct produit
{
int numero;
int qte;
double prix;
}pro;
```

Pro art1,art2;

Exercice:

Soit la définition de la structure suivante :

```
Struct produit
{
int numero;      //numéro d'un produit
char* nom;       //nom du produit
};
```

1. Ecrire la fonction SAISIE(...) permettant de saisir un produit.

2. Ecrire la fonction AFFICHE(...) qui reçoit en paramètre l'adresse d'une variable de type produit et en affiche les champs.

3. Ecrire la fonction INVERSER(...) qui reçoit en paramètre l'adresse d'une variable de type produit et qui renvoie en résultat une variable de même type correspondant à un produit de même numéro et avec un nom dont l'ordre des caractères est inversé. Pour inverser l'ordre des caractères du nom du produit, utiliser la fonction INV_CH (chaîne) qui retourne la chaîne donnée en paramètre avec ordre inversé des caractères.