CHAPITRE 1 : LES STRUCTURES

I. Introduction :

Jusqu'à présent, nos programmes utilisaient des données simples : caractères, entiers, flottants, et tableaux des précédents. Ce n'était donc que dans l'esprit du programmeur qu'un lien fonctionnel pouvait éventuellement s'établir entre des données de natures différentes. Par exemple, dans le cas des matrices, nous avions été amenés à utiliser deux données :

- une variable qui contient la taille de la matrice ;
- une variable de type tableau à deux dimensions qui contient les valeurs de la matrice.

Lorsque le nombre de données augmente, il devient très problématique de les manipuler sous cette forme « éclatée » : chaque fois que l'on veut manipuler une nouvelle entité, il faut déclarer un nombre de variables important. De plus, si l'on souhaite utiliser une procédure sur l'une de ces entités, il faut lui passer en argument toutes les variables utiles pour le traitement : ceci devient rapidement très lourd, source d'erreurs (si on intervertit deux arguments, par exemple), et potentiellement source de dégradation de performance.

C'est pour résoudre ce type de problème que les langages de haut niveau proposent un mécanisme d'agrégation, permettant de regrouper des données. On désigne en général par *structure de données* un tel objet agrégé. On trouve également la terminologie *d'enregistrement* ou de *record*.

Le langage C permet au programmeur de construire ses propres types de données agrégées.

II. Définition :

Une structure est un type d'objet composé qui permet de désigner sous un même nom un ensemble d'objets de types différents, appelés champs de cette structure.

Définir une variable de type structure suppose avoir d'abord défini le modèle de cette structure en décrivant l'ensemble des champs contenus et en lui associant un nom. Un modèle de structure constitue en fait un nouveau type de variable défini par l'utilisateur

III. Création d'un type structure :

Syntaxe :

```
struct NomdelaStructure {
type champ1;
type champ2;
type champn;
...
      }; /* ne pas oublier le ; */
```

Cette instruction permet de définir un nouveau modèle (type) de structure : chaque champ est constitué par un type et un nom de champ.

Exemple :

```
struct  personne {
          char nom[32] ;
          char prenom[32] ;
          int age ;
          } ;

/* Cette structure contient trois champs de différents types */
```

IV. Création des variables de type "structure" :

- Lorsqu'un modèle de structure est défini, il devient possible de définir des variables ayant ce type :

```
struct  nom_de_la_structure      var1;
```

Cette instruction permet de définir une nouvelle variable var1 du type "structure nom_de_la_structure". Une telle variable est elle-même souvent appelée "structure" (par simplification).

- Ou bien dans la même instruction :

```
struct NomdelaStructure {
type champ1;
type champ2;
type champn;
...
} var1;
```

Exemple 1:

```
struct personne p1 ;       /*  la variable p1 de type "struct personne" */
struct personne fils1,  fils2 ;       /* plusieurs variables*/
```

Exemple 2:

```
struct  personne {
          char nom[32] ;
          char prenom[32] ;
          int age ;
          } p1, fils1;  /* deux variables de type struct personne*/
```

V. Accès aux champs d'une structure :

En C, il est possible d'utiliser une structure de deux manières :

- en travaillant **individuellement** sur chacun de ses champs,
- en travaillant de manière "**globale**" sur l'ensemble de la structure.

V-1. Utilisation des champs d'une structure :

Chaque champ d'une structure peut être manipulé comme n'importe quelle variable du type correspondant. La désignation d'un champ se note en faisant suivre le nom de la variable structure de l'opérateur "point" **(.)** suivi du nom de champ tel qu'il a été défini dans le modèle.

Exemple :

```
struct personne p1 ;
p1.age= 20 ;
printf ("%d", p1.age) ;
scanf ("%s",p1.nom) ;
puts(p1.nom) ;
```

V-2. Utilisation globale d'une structure (Affectation de structures) :

Le langage C permet d'effectuer des affectations de structures, en utilisant l'opérateur d'affectation **=**, comme pour n'importe quel autre type de données. Notez que celui-ci affecte l'ensemble des champs de la structure ; si vous ne souhaitez copier que quelques uns de ses champs, il vous faudra les copier un par un. De plus, il n'est possible d'affecter que des structures de même type !

Exemple :

```
fils1 = fils2 ;     /* affectation du contenu de fils2 dans fils1 */
```

Une telle affectation globale remplace :

```
strcpy(fils1.nom,fils2.nom);
strcpy(fils1.prenom,fils2.prenom);
fils1.age= fils2.age;
```

VI. Initialisation d'une structure :

Comme les tableaux, Il est possible en langage C d'initialiser une structure entière en une seule affectation, de la manière qui suit :

```
struct personne p1={"Mrabit", "Hicham", 30};
```

VII. Utilisation de typedef :

Le mot-clé **typedef** permet d'associer un nom à un type donné. On l'utilise suivi de la déclaration d'un type (en général une structure) puis du nom qui remplacera ce type. Ceci permet, par exemple, de s'affranchir de l'emploi de **struct** à chaque utilisation d'une variable de type personne. Il n'est pas alors nécessaire de donner un nom à la structure. L'exemple précédent peut donc se réécrire de la manière suivante :

```
typedef struct  {
        char nom[32] ;
        char prenom[32] ;
        int age ;
        } personne ;

personne p1 ;
p1.age= 20 ;
printf ("%d", p1.age) ;
scanf ("%s",p1.nom) ;
puts(p1.nom) ;
```

VIII. IMBRICATION DE STRUCTURES :

Dans nos exemples d'introduction des structures, nous nous sommes limités à une structure simple ne comportant que trois champs d'un type de base. Mais chacun des champs d'une structure peut être d'un type absolument quelconque : pointeur, tableau, structure,... De même, un tableau peut être constitué d'éléments qui sont eux-mêmes des structures. Voyons ici quelques situations classiques.

VIII-1. Structure comportant des tableaux :

Soit la déclaration suivante :

```
struct  personne {
        char nom[32] ;
        char prenom[32] ;
        int age ;
        float heures_travail [31] ;
        } ;
personne p1 ;
```

- La notation : ***p1. heures_travail [4]*** désigne le cinquième élément du tableau ***heures_travail*** de la structure ***p1.***
- La notation **&** ***p1. heures_travail [4]*** représente l'adresse du cinquième élément tableau ***heures_travail*** de la structure ***p1.***

VIII-2. Tableau de structures :

Les N personnes d'une entreprise ont tous la même structure :

```
struct  personne {
          char nom[32] ;
          char prenom[32] ;
          int age ;
          float salaire ;
          } ;
```

Déclaration du tableau dont les éléments ont la structure *personne*:

```
struct personne tab [taille] ;
```

Exemple :

```
struct personne
{
          char nom[20];
          char prenom[20];
          int age;
};

main()
{         int i;
          struct personne p[5];
          for(i=0;i<5;i++)
                    { printf("structure N %d ",i+1);
                    scanf("%s",p[i].nom);
                    scanf("%s",p[i].prenom);
                    scanf("%d",&p[i].age);
                    }
}
```

IX. Pointeurs de structure :

Il est aussi possible d'accéder aux champs d'une structure en utilisant un pointeur sur cette structure :

- Si p est un pointeur sur une structure, on peut accéder à un membre de la structure pointé par l'expression: **(*p).membre**
- Cette notation peut être simplifiée grâce à l'opérateur *pointeur de membre de structure*, noté **->** (tiret et supérieur) : **p->membre**

Exemple :

```
struct personne *pf ;              /* Définition d'un pointeur sur une structure de type "struct
                                   personne" */
struct personne  fils1;
pf = &fils1 ;              /* l'adresse de fils1 est affectée au pointeur pf */
pf->age = 15 ;             /* on met 15 dans le champ age de la structure fils1 pointée par pf */
fils2.age = pf->age + 2 ;

strcpy( pf->nom, "Toto" ) ;

printf(" Donnez votre prenom : " ) ;
scanf("%s", pf->prenom ) ;
```

X. Structures et Fonctions :

X-1. Transmission de la valeur d'une structure :

On peut transmettre une structure par valeur (contrairement au tableau) et on travaille dans l'appelée sur une copie. :

Exemple :

```
struct personne
{
        char nom[20];
        char prenom[20];
        int age;
};
void Afficher_personne( struct personne p)
{
printf(" le nom est : %s\n" ,p.nom);
printf(" le prénom est : %s\n" ,p.prenom);
printf(" \'age est : %d\n" ,p.age);
}
main()
{
struct personne p1={"Mrabit", "Hicham", 30};
Afficher_personne(p1);
}
```

X-2. Transmission de l'adresse d'une structure : (l'opérateur ->)

Pour changer les valeurs de la structure, on envoie un pointeur :

Exemple :

```
struct personne
{           char nom[20];
           char prenom[20];
           int age;
};
void changer_personne( struct personne *p)
{
strcpy((*p).nom,"Alaoui");    /* ou strcpy(p->nom,"Alaoui"); */
strcpy((*p).prenom,"Hassan"); /* ou strcpy(p->prenom,"Hassan"); */
(*p).age=20; /* ou p->age=20; */
}
void Afficher_personne( struct personne p)
{
printf(" le nom est : %s\n" ,p.nom);
printf(" le prénom est : %s\n" ,p.prenom);
printf(" \'age est : %d\n" ,p.age);
}
main()
{struct personne p1={"Mrabit", "Hicham", 30};
Afficher_personne(p1);
changer_personne(&p1);
Afficher_personne(p1);
}
```

X-3. Structure en valeur de retour:

Le langage C autorise à réaliser des fonctions qui fournissent en retour la valeur d'une **structure. :**

Exemple :

```
struct personne
{       char nom[20];
        char prenom[20];
        int age;
};
struct personne changer_personne( struct personne p)
{
strcpy(p.nom,"Alaoui");
strcpy(p.prenom,"Hassan");
p.age=20;
return p;
}
void Afficher_personne( struct personne p)
{
printf(" le nom est : %s\n" ,p.nom);
printf(" le prénom est : %s\n" ,p.prenom);
printf(" \'age est : %d\n" ,p.age);
}
main()
{struct personne p1={"Mrabit", "Hicham", 30};
Afficher_personne(p1);
p1=changer_personne(p1);
Afficher_personne(p1);
}
```