Les types composés

Les types composés

A partir des types prédéfinis du C (caractères, entiers, flottants), on peut créer de nouveaux types, appelés *types composés*, qui permettent de représenter des ensembles de données organisées:

- Les structures
- Les unions
- Les énumérations

Les types composés

Les structures

Les structures(1)

- C'est un mécanisme permettant de grouper un certain nombre de variables de types différents au sein d'une même entité en utilisant le concept d'enregistrement.

- Un enregistrement est un ensemble d'éléments de types différents repérés par un nom. Les éléments d'un enregistrement sont appelés des *champs*. Le langage C possède le concept d'enregistrement appelé structure.

- **Déclaration de structure :**

- **<u>Méthode 1</u> : déclaration en précisant un nom pour la structure**

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

- On peut ensuite utiliser ce type structure pour déclarer des variables, de la manière suivante :

struct personne p1,p2; /* qui déclare deux variables de type
                                struct personne de noms p1 et p2 */

Les structures(2)

- **<u>Méthode 2</u> : déclaration en précisant un nom pour la structure et en déclarant des variables** struct personne

```
{
    char nom[20];
    char prenom[20];
    int  n_cin;
} p1,p2;
```

Déclare les deux variables **p1** et **p2** et donne le nom **personne** à la structure. Là aussi, on pourra utiliser ultérieurement le nom **struct personne** pour déclarer d'autres variables.

- **<u>Méthode 3</u> : déclaration en utilisant typedef**

```
typedef struct  personne        /* le nom du type peut être omis ici si on ne va l'utiliser à
{                                                    l'intérieur de la même structure */
    char nom[10];
    char prenom[10];
    int n_cin;
}personne;   /*il faut mettre le nom du nouveau type ici, si non à la déclaration de variable, on doit
                mettre struct personne p;…*/
```

- **<u>Utilisation</u> : On déclare des variables par personne et non par struct personne :**

personne x,y;

Les structures(3)

- **Accès aux membres d'une structure :**
  Pour désigner un membre d'une structure, il faut utiliser l'opérateur de sélection de membre '**.**' (Point).
- **Exemple:**

```
struct personne
{
  char nom[20];
  char prenom[20];
  int    age;
};
struct personne p1,p2;

p1.age=15 ;                 /* accès au troisième champs + affectation */
scanf ("%s",p1.nom) ;       /* lecture du premier champs */
printf("%d",p1.age) ;       /* affichage du troisième champs */
p2.nom[0] = 'X';
```

Les structures(4)

- **Initialisation d'une structure :**

Exemple1:

struct personne pr1 = {"Sami", "Ben sassi", 55};

Exemple2:

```
struct personne pr2;        /* déclaration de pr2 de type struct personne */
strcpy ( pr2.nom, "ahmed" ) ;   /* affectation de  "ahmed" au deuxième
      champs */
strcpy ( pr2.prenom, "sousou" ) ;
pr2.age = 164 ;
```

Remarque:

On ne peut pas faire l'initialisation d'une structure lors de sa déclaration.

Les structures(5)

- **Affectation de structures:**

  On peut affecter une structure à une variable structure de même type.

  ```
  struct personne pr1,pr2;
  pr1 = pr2 ;
  ```

- **Comparaison de structures:**

  Aucune comparaison n'est possible sur les structures, même pas les opérateurs **==** et **!=**.

Tableau de structures

Une déclaration de tableau de structures se fait selon le même modèle que la déclaration d'un tableau dont les éléments sont de type simple. Supposons que l'on ait déjà déclaré la **struct personne.**

```
struct personne t[100];     /* dec d'un tableau de 100
                               structures de type struct personne */
```

Pour référencer le nom de la personne qui a l'index **i** dans **t** on écrira : **t[i].nom**.

Composition de structures

- **Composition de structures:**
  Une structure permet de définir un type. Ce type peut être utilisé dans la déclaration d'une autre structure comme type d'un de ses champs.
- **Exemple :**

```
struct date
  {
      unsigned int jour;
      unsigned int mois;
      unsigned int annee ;
  };
struct personne
  {
        char    nom[20];
        char    prenom[20];
        struct date  d_naissance;
  };
```

Exercice

Créer une structure **point{int num;float x;float y;}**
Saisir 4 points, les ranger dans un tableau puis les afficher.

```
#include <stdio.h>
#include <conio.h>
typedef struct {int num;float x;float y;}
      point;
void main()
{
point p[4]; /* tableau de points */
int i;
float xx,yy;
/* saisie */
printf("SAISIE DES POINTS\n\n");
for(i=0;i<4;i++)
      {
      printf("\nRELEVE N¯%1d\n",i);
      p[i].num = i;
      printf("X= ");scanf("%f",&xx);
      printf("Y= ");scanf("%f",&yy);
      p[i].x = xx;p[i].y = yy;
      }
/* affichage*/
printf("\n\nAffichage\n\n");
for(i=0;i<4;i++)
      {
      printf("\nRELEVE N¯%1d",p[i].num);
      printf("\nX= %f",p[i].x);
      printf("\nY= %f\n",p[i].y);
      }
printf("\n\nPOUR SORTIR FRAPPER UNE
      TOUCHE ");
getch();
}
```

Les énumérations

- Les énumérations permettent de définir un type par la liste des valeurs qu'il peut prendre. Un objet de type énumération est défini par le mot clef **enum** et un identificateur de modèle suivis de la liste des valeurs que peut prendre cet objet.
- **Exemple1** :

```
enum jour {LUNDI, MARDI, MERCREDI, JEUDI, VENDREDI,
           SAMEDI, DIMANCHE};
enum jour j1, j2;
j1 = LUNDI;
j2 = MARDI;
enum jour {LUNDI, MARDI, MERCREDI, JEUDI, VENDREDI,
   SAMEDI, DIMANCHE} d1, d2;
enum {FAUX, VRAI} b1,b2;  /* mauvaise programmation */
```

Les unions(1)

- Une **union** désigne un ensemble de variables de types différents susceptibles d'occuper alternativement une **même zone mémoire**. Une union permet donc de définir un objet comme pouvant être d'un type au choix parmi un ensemble fini de types. Si les membres d'une union sont de longueurs différentes la place réservée en mémoire pour la représenter correspond à la taille du membre le plus grand.

```
union jour
 {
   int     numero;
   char  lettre;
} ;
union jour hier, demain ;   /* déclaration de deux variables de type union jour */
hier.numero = 5 ;
hier.lettre='D' ;                /* écrase la valeur précédente */
```

- **Remarque :**

  La différence sémantique entre les struct et les unions est la suivante : alors que pour une variable de type structure tous les membres peuvent avoir en même temps une valeur, une variable de type union ne peut avoir à un instant donné qu'un seul membre ayant une valeur.

Les unions(2)

- **Utilisation des unions**

```
enum type {ENTIER, FLOTTANT};
struct arith
{
  enum type typ_val;      /*   indique ce qui est dans u   */
  union
    {
        int i;
        float f;
    } u;
};
```

La **struct arith** a deux membres typ_val de type enum, et u de type union d'int et de float. On déclare des variables par :

```
struct arith a1,a2;
```

Les unions(3)

Puis on peut les utiliser de la manière suivante :

```
a1.typ_val = ENTIER;
a1.u.i = 10;
a2.typ_val = FLOTTANT;
a2.u.f = 3.14159;
```

- Si on passe en paramètre à une fonction un pointeur vers une struct arith, la fonction testera la valeur du membre typ_val pour savoir si l'union reçue possède un entier ou un flottant.