Types structures, unions et synonymes

Structures

- Une **structure** est un nouveau type de données constitué par un ensemble de variables (champs) qui peuvent être hétérogènes et de types différents

- La différence avec le type tableau est que les champs d'un tableau sont tous homogènes et du même type

- Les structures permettent de représenter des objets réels caractérisées par plusieurs informations, par exemple :

  - Une personne caractérisée par son nom (chaîne), son âge (entier), sa taille (réel), …

  - Une voiture caractérisée par sa marque (chaîne), sa couleur (chaîne), son année modèle(entier), …

Déclaration d'une structure

La déclaration d'une structure s'effectue en précisant le nom de la structure, ainsi que le nom et le type de ses champs :

- Syntaxe en C :

```
struct  nom_structure
        { type 1 nom_champ1;
          type 2 nom_champ2;
          …
          type N nom_champN;
        };
```

- Exemple :

```
struct Personne
        { char Nom[20];
          int Age;
          float taille;
        };
```

Rq: Le nom d'une structure n'est pas un nom de variable, c'est le nom du type ou modèle de la structure

Déclaration d'une variable structure

- La déclaration d'une structure ne réserve pas d'espace mémoire

- La réservation se fait quand on définit des variables correspondant à ce modèle de structure. Ceci peut se faire soit :
  - après la déclaration de la structure, par exemple :

```
struct Personne p1, *p2, tab[10];
//p2 est une variable de type pointeur sur une structure Personne
// tab est une variable de type tableau de 10 éléments (de type Personne)
```

  - ou au moment de la déclaration de la structure

```
struct Personne
        { char Nom[20];
          int Age;
          float taille;
        } p1, *p2, tab[10];
```

Initialisation d'une structure

- Lors de la déclaration d'une variable structure, on peut initialiser ses champs avec une notation semblable à celle utilisée pour les tableaux en indiquant la liste des valeurs respectives entre accolades.

- Exemple :

```
struct date
        { unsigned short jour;
          char mois[10];
          unsigned short annee;
        } ;
```

struct date d1= {15,"Novembre", 2013};

Accès aux champs d'une structure

- L'accès à un champ d'une variable structure se fait par le nom de la variable suivi d'un point et du nom du champ

  nom_variable.nom_champ

  Exemple: p1.age représente le champ age de la variable structure p1

- Dans le cas d'une variable structure de type pointeur

  (ex : **struct Personne *p2**), on utilise en général l'opérateur **->** pour accéder aux champs

  nom_variable ->nom_champ (ex: p2->age)

- Remarque : Il est possible d'utiliser un point même dans le cas d'un pointeur, par exemple **(*p2).age** (les parenthèses sont ici indispensables)

Accès aux champs d'une structure : Exemple

Saisie et affichage d'une structure

- Soit la structure article définie par :

```
struct article
        { int numero; //un numéro qui identifie l'article
          char nom[20];
          int qte_stock; // quantité disponible en stock
          float prix;
        } ;
```

- Soit la déclaration de variables :

```
struct article art, *pt_art;
```

Saisissez les champs de la structure art, puis affichez son contenu

Exemple de Saisie et d'affichage d'une structure

```
/* Saisie des champs de la structure art */
printf ("Entrez respectivement les champs de l'article \n ");
scanf(" %d %s %d %f" , &art.numero, art.nom, &art.qte_stock,&art.prix);

/* Saisie des champs de la structure art */
printf (" Cet article a pour : \n ");
printf (" \t numéro : %d \n ", art.numero);
printf (" \t nom : %s \n ", art.nom);
printf (" \t quantité en stock : %d \n ", art.qte_stock);
printf (" \t prix : %f \n ", art.prix);
```

Refaites la saisie et l'affichage en utilisant le pointeur pt_art

Exemple de Saisie et d'affichage via un pointeur

Remarque : il faut d'abord initialiser le pointeur pt_art

```
/* Saisie des champs de la structure art */
printf ("Entrez respectivement les champs de l'article \n ");
scanf(" %d %s %d %f" , &pt_art->numero, pt_art->nom,
                        & pt_art->qte_stock, & pt_art->prix);

/*Affichage des champs de la structure art */
printf (" Cet article a pour : \n ");
printf (" \t numéro : %d \n ", pt_art->numero);
printf (" \t nom : %s \n ", pt_art->nom);
printf (" \t quantité en stock : %d \n ", pt_art.->qte_stock);
printf (" \t prix : %f \n ", pt_art->prix);
```

Composition des structures

- Les structures peuvent être composées de champs de n'importe quel type connu: types de base, pointeur, tableau ou structure.

- Exemple de structure comportant un tableau et une structure :

```
struct Etudiant
        { int code;
          char Nom[20];
          struct date date_naissance;
          float notes[8]; // notes de l'étudiant dans 8 modules
        } E1,E2;
```

- on peut écrire E1.date naissance.annee pour accéder au champ annee de E1.date naissance qui est de type date

- E2.notes[3] représente la note du module 4 de l'étudiant E2

Manipulation des structures

- Les structures sont manipulées en général champ par champ. Toutefois, la norme ANSI permet d'affecter une structure à une autre structure de même type, par exemple : struct Etudiant E1,E2;

  E2=E1; est une instruction valide qui recopie tous les champs de E1 dans les champs de E2

- Rq: Il n'est pas possible de comparer deux structures. Les instructions if(E1==E2) ou if(E1!=E2) ne sont pas permises

- L'opérateur & permet de récupérer l'adresse d'une variable structure (ex : struct Personne p1, *p2; **p2=&p1**)

- L'opérateur sizeof permet de récupérer la taille d'un type structure ou d'une variable structure (ex : sizeof (struct Personne) ou sizeof (p1))

Structures et fonctions

Une structure peut être utilisée comme argument d'une fonction et transmise par valeur (ou par adresse via un pointeur)

Exemple de transmission par valeur

```
struct couple { int a;
                float b;};
void zero (struct couple s)
   { s.a=0; s.b=0;
     printf(" %d %f \n ", s.a, s.b);
   }
main()
  { struct couple x;
    x.a=1;x.b=2.3;
    printf("avant: %d %f \n ", x.a, x.b);
    zero(x);
    printf("après: %d %f \n ", x.a, x.b);
  }
```

Exemple de transmission par adresse

```
struct couple { int a;
                float b;};
void zero (struct couple *s)
   { s->a=0; s->b=0;
     printf(" %d %f \n ", s->a, s->b);
   }
main()
  { struct couple x;
    x.a=1;x.b=2.3;
    printf("avant: %d %f \n ", x.a, x.b);
    zero(&x);
    printf("après: %d %f \n ", x.a, x.b);
  }
```

Structures et fonctions (2)

Une structure peut être retournée par l'instruction return d'une fonction

Exemple

```
struct couple { int a;
                float b;};
struct couple oppose(struct couple s)
   {struct  couple z;
     z.a=-s.a; z.b=-s.b;
    return z;
   }
```

```
main()
 { struct couple x,y;
   x.a=1;x.b=2.3;
   printf("x: %d %f \n ",x.a,x.b);
   y=oppose(x);
   printf("y: %d %f \n ",y.a,y.b);
 }
```

Structures et fonctions : exercice

Pour la structure article définie auparavant :

- Ecrivez une fonction, nommée SaisieArticle, qui saisit les champs d'une variable article passé en paramètre
- Ecrivez une fonction, nommée AfficheArticle, qui affiche le contenu des champs d'une variable article passé en paramètre
- Ecrivez une fonction nommée SaisieTabArticle, qui remplit un tableau T de n articles. T et n sont des paramètres de la fonction.
- Ecrivez une fonction AfficheTabStock, qui affiche les articles de T ayant une quantité en stock >= à une valeur q. T, n et q sont des paramètres de la fonction
- Ecrivez un programme qui fait appel aux fonctions SaisieTabArticle et AfficheTabStock

Structures et fonctions : exercice (2)

```
struct article
 {int numero;
  char nom[20];
  int qte_stock;
  float prix;
} ;

void SaisieArticle (struct article * art)
{printf (" numero ? \n");
scanf (" %d",&art->numero);
printf (" nom ? \n ");
scanf("%s",art->nom);
printf (" quantité en stock ? \n ") ;
scanf (" %d" , &art->qte_stock);
printf (" prix ?\n ");
scanf (" %f" ,&art->prix);
}
```

```
void AfficheArticle (struct article  art)
{printf (" Cet article a pour : \n ");
printf (" \t numéro : %d \n ", art.numero);
printf (" \t nom : %s \n ", art.nom);
printf (" \t stock : %d \n ", art. qte_stock);
printf (" \t prix : %f \n ", art.prix);
}

void SaisieTabArticle(struct articleT[ ], int n)
 {int i;
  for(i=0;i<n;i++)
   {printf (" saisie de l'article %d \n " ,i+1);
    SaisieArticle (&T[i]);
   }
 }
```

Structures et fonctions : exercice (3)

```
void AfficheTabStock(struct article *T, int n, int q)
{ int i;
   printf (" Les articles ayant un stock   >=%d sont : \n " ,q);

  for(i=0;i<n;i++)
      if(T[i]. qte_stock>=q)
            AfficheArticle(T[i]);
}

main( )
 { struct articleT[10];
   SaisieTabArticle(T, 10);
   AfficheTabStock(T, 10, 2);
 }
```

Définition de types synonymes: typedef

- En C, on peut définir des types nouveaux synonymes de types existants (simples, pointeur, tableau, structure,…) en utilisant le mot clé **typedef**. Ces nouveaux types peuvent être utilisées ensuite comme les types prédéfinis

- **Exemple d'un type synonyme d'un type simple :**

  typedef int entier; définit un nouveau type appelé **entier** synonyme du type prédéfini **int**

  entier i=4,T[10] ; le type entier est utilisé ici pour déclarer des variables

  Remarque : l'intérêt de typedef pour les types de base est limité puisqu'il remplace simplement un nom par un autre

- **Exemple d'un type synonyme d'un type pointeur :**

```
typedef  int * ptr  entier;
ptr_entier p1,p2;      p1 et p2 sont des pointeurs sur des int

typedef  char* chaine;
chaine ch;
```

Exemples de typedef

- **Type synonyme d'un type tableau :**

```
typedef  float  tableau[10];
tableau T;
```
T est un tableau de 10 réels

```
typedef  int  matrice[4][5];
matrice A;
```
A est une matrice d'entiers de 4 lignes et 5 colonnes

- **Type synonyme d'un type structure :**

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

- date est le nom du nouveau type et non d'une variable
- On peut utiliser directement ce type, par exemple: **date d, *p;**
- Ceci permet simplement d'éviter l'emploi du mot clé struct dans les déclarations de variables

Structures récursives

- Une structure est dite récursive si elle contient des pointeurs vers elle même, par exemple :

```
struct Individu
{ char *Nom;
  int Age;
 struct Individu *Pere, *Mere;
};
```

```
struct noeud
{ int val;
 struct noeud *suivant;
};
```

- Ce genre de structures est fondamental en programmation car il permet d'implémenter la plupart des structures de données employées en informatique (seront étudiées en cours de Structures de données)

Unions

- Une **union** est un objet défini par un ensemble de données (comme une structure) mais qui ne peut prendre qu'une seule donnée à la fois à un instant donné

- Une union se déclare comme une structure :
  - **union nom_union**
    { type 1 nom_champ1;
    type 2 nom_champ2;
    …
    type N nom_champN;
    };

  Exemple :

  ```
  union IntFloat
  {  int n;
     float x;
  } A;
  ```

  - A un instant donné, la variable A contiendra un entier ou un réel, mais pas les deux !
  - C'est au programmeur de connaître à tout instant le type de l'information contenue dans A

Unions (2)

- Les unions permettent d'utiliser un même espace mémoire pour des données de types différents à des moments différents
- Lorsque l'on définit une variable correspondant à un type union, le compilateur réserve l'espace mémoire nécessaire pour stocker le champ de plus grande taille appartenant à l'union
- La syntaxe d'accès aux champs d'une union est identique à celle pour accéder aux champs d'une structure (dans l'exemple, Le programmeur utilisera A.n s'il désire manipuler l'information de type int, sinon il utiliseraA.x)
- De façon générale, une union se manipule comme une structure