Types Structures, Unions, Enumérés et Synonymes

1 Types structures (struct)

- Une structure est un nouveau *type de données composé de plusieurs* ***champs*** (ou ***membres***) qui sert à représenter un objet réel.
- Chaque champ est de type quelconque pouvant être, lui aussi, une structure.
- Le nom d'une structure n'est pas un nom de variable (***c'est un nom de type***).

Exemple :

Une date peut être représentée par les renseignements : jour, mois et année.

1.1 Définition d'une structure :

- ***En utilisant un nom pour la structure :***

```
struct <nom_structure>
{
 <type1> <nom_champ1> ;
 <type2> <nom_champ2> ;
 …
 <typeN> <nom_champN> ;
} ;
```

Par l'intermédiaire du nom de la structure, on peut déclarer plusieurs variables de ce type de structure n'importe où et chaque fois que c'est nécessaire.

Exemples :

1. ```
   struct date
   {
    int jour ;
    int mois ;
    int annee ;
   } ;
   ```
2. ```
   struct adresse
   {
    char nom[25], prenom[25] ;
    int n_rue ; /* numéro de rue */
    char rue[30] ;
    char ville[20] ;
   } ;
   ```
3. ```
   struct date date_de_naissance ;
   struct adresse adr1, adr2 ;
   struct complexe
   {
    double re ;
    double im ;
   } z1, z2 ; /* z1 et z2 deux variables de type complexe */
   ```
   ```

- ***Sans utilisation de nom (structure anonyme) :***

```
struct
{
 <type1> <nom_champ1> ;
 <type2> <nom_champ2> ;
 …
 <typeN> <nom_champN> ;
} <liste_de_variables> ;
```

Les variables de ce type structure doivent être déclarées *immédiatement*.

Exemple :

```
struct
{
 int heure ;
 int minute ;
 int seconde ;
} t1, t2 ;
```

1.2 Portée d'une structure :

Dépend de l'emplacement de sa déclaration :

- Si elle se situe au sein d'une fonction (*y compris la fonction* ***main***), alors elle n'est accessible que dans cette fonction.
- Si elle se situe en dehors d'une fonction, alors elle est accessible de toute la partie du fichier source qui suit l'emplacement de la déclaration.

1.3 Tableau de structures :

```
<type_structure> <NomTableau> [<dimension>] ;
```

Exemple :

```
struct client
{
 int compte ;
 char nom[20], prenom[20] ;
 float solde ;
} banque[1000] ; /* un tableau de 1000 clients au plus */
```

1.4 Imbrication de structures :

Exemple :

```
struct stage
{
 char nom[40] ;
 struct date debut, fin ;
} s, ts[10] ;
```

1.5 Pointeur sur une structure :

Exemple :

```
struct date *pd ;
```

1.6 Utilisation de structures :

Les structures peuvent être manipulées ***champ par champ*** ou ***dans leur ensemble***.

Opérations sur les champs :

- ***Accès à un champ d'une structure :***

```
<variable_structure>.<champ_structure>
```

Exemple :

```
struct date d ;
d.jour = 2 ;  /* accès au champ jour de la date d */
scanf("%d", &d.jour) ;
printf("%d", d.jour) ;
```

- ***Accès à un champ d'un pointeur de structure :***

```
<pointeur_structure>-><champ_structure>
```

Exemple :

```
struct date *pd, d ;
pd = &d ;
pd->jour = 5 ;   /* accès au champ jour */
```

Remarque :

Il y a équivalence entre `(*pd).jour` et `pd->jour`

Opérations sur les variables structures :

- ***Initialisation à la déclaration :***

Exemple :

```
struct date d = {4, 10, 1999} ;
```

- ***Affectation :***

  Les variables structures doivent être de même type (*à condition que des champs de la structure ne soient pas déclarés comme constantes*)

Exemple :

```
struct date d1, d2 = {4, 10, 1999} ;
d1 = d2 ;
```

- ***Opérateur d'adresse & :***

Exemple :

```
struct date d, *pd ;
pd = &d ;
```

- ***Opérateur sizeof :***

Exemple :

```
printf("taille structure date : %d\n", sizeof(struct date)) ;
```

1.7 Structure auto-référentielle (ou récursive):

- Un ou plusieurs champs de la structure est ***un pointeur sur elle-même***.
- Permet de représenter des suites (*finies*) de taille quelconque avec ajouts et suppressions efficaces d'éléments.
- Ces structures auto-référentielles requièrent généralement l'allocation dynamique pour allouer et libérer explicitement de la mémoire.

Exemple : *Liste chaînée de réels*

```
struct cellule
{
 double elt ;
 struct cellule *suiv ;
} ;
```

Chaque cellule a deux champs, elt et suiv. elt est un réel, alors que suiv est un pointeur sur une structure cellule. La valeur de suiv est soit l'adresse en mémoire d'une cellule soit le pointeur **NULL**.

```
struct cellule *liste = NULL ; /* initialement, liste vide */
  /* Ajout du réel 2.5 à la liste */
liste = (struct cellule *) malloc (sizeof(struct cellule)) ;
if (liste != NULL)
  {
   liste->elt = 2.5 ;
   liste->suiv = NULL ;
  }
```

Fonctions et structures :

- ***Retour d'une variable structure par une fonction :***

Exemple :

```
struct date newdate()
{
 struct date d ;
 printf("Jour (1, 2, …, 31) : ") ; scanf("%d", &d.jour) ;
 printf("Mois (1, 2, …, 12) : ") ; scanf("%d", &d.mois) ;
 printf("Année (1900, …, 1999) : ") ; scanf("%d", &d.annee) ;
 return d ;
}
```

- ***Passage par valeur en argument d'une variable structure à une fonction :***

Exemple :

```
int chekdate(struct date) ;
```

- ***Passage par adresse en argument d'une variable structure à une fonction :***

Exemple :

```
void lire_date(struct date *pd)
{
 printf("Jour (1, 2, …, 31) : ") ; scanf("%d", &(*pd).jour) ;
 printf("Mois (1, 2, …, 12) : ") ; scanf("%d", &(*pd).mois) ;
 printf("Année (1900, …, 1999) : ") ; scanf("%d", &pd->annee) ;
}
```

1.8 Champs de bits :

Un mot machine (*par exemple, mot de 16 bits*) peut être utilisé pour *stocker plusieurs données*, chaque donnée occupe un certain nombre de bits.

Syntaxe de définition :

```
struct <nom_structure>
{
 unsigned int <nom_champ1> : <nombre_de_bits> ;
 unsigned int <nom_champ2> : <nombre_de_bits> ;
 …
 unsigned int <nom_champN> : <nombre_de_bits> ;
} ;
```

Exemple :

```
struct langue
 {
  unsigned int anglais : 1 ;
  unsigned int allemand : 1 ;
  unsigned int espagnol : 1 ;
  unsigned int japonais : 1 ;
  unsigned int russe : 1 ;
 } ;
 struct employe1 ;
 {
  …
  int anglais ;
  int allemand ;
  int espagnol ;
  int japonais ;
  int russe ;
  …
 } ListeEmpl1[1000] ;
 struct employe2 ;
 {
  …
  struct langue L ;
  …
 } ListeEmpl2[1000] ;
```

Soit x le nombre d'octets occupés par les autres champs (désignés par …) de la structure employe1 ou de la structure employe2. Calculer l'espace mémoire occupé par ListeEmpl1 et ListeEmpl2. Que remarquez-vous ?

2 Types unions (union)

- Les unions permettent l'utilisation d'***un même espace mémoire*** par des données de types différents à des moments différents :
  - Une union ne contient qu'une donnée à la fois.
  - Le système alloue un emplacement mémoire tel qu'il pourra contenir le champ de plus grande taille appartenant à l'union.
- ***Syntaxe de définition :***

```
union <nom_union>
{
 <type1> <nom_champ1> ;
 <type2> <nom_champ2> ;
 …
 <typeN> <nom_champN> ;
} ;
```

Exemple :

```
union zone
{
 int entier ;
 long entlong ;
 float flottant ;
 double flotlong ;
} z1, z2;
```

3 Types énumérés (enum)

Permettent d'exprimer ***des valeurs constantes*** de type entier en associant ces valeurs à des noms.

3.1 Syntaxe de définition :

```
enum <nom_énumération>
{
 <identificateur1> ;
 <identificateur2> ;
 …
 <identificateurN> ;
} ;
```

- Les identificateurs sont considérés comme des constantes entières.
- Le compilateur associe au 1er identificateur la constante 0, au 2ème la constante 1, … et au N^{ème} la constante N+1.

Exemples :

1. ```
   enum couleurs
   {rouge, vert, bleu} rvb ;
   ```
2. ```
   enum jour
   {Lundi, Mardi, Mercredi, Jeudi, Vendredi, Samedi, Dimanche};
   enum jour j1, j2, j ;
   ```

3.2 Opérations sur les variables de type énuméré :

- ***Affectation :***

Exemple :

```
j1 = Lundi ;
j2 = j1 ;
```

- ***Comparaison :***

Exemple :

```
if (j == Lundi) printf("Le jour est un Lundi\n") ;
```

- ***Incrémentation, décrémentation :***

Exemple :

```
j2 = Dimanche ; j2-- ;
for (j = Lundi ; j<Samedi ; j++)
   …
```

4 Types synonymes (typedef)

- **typedef** permet de définir *des types nouveaux synonymes* à des types existants.
- **typedef** ne réserve pas d'espace mémoire. Le nom est un type ; il est donc inaccessible comme une variable.
- ***Syntaxe de définition :***

```
typedef <type> <nom_de_remplacement1>,
               <nom_de_remplacement2>,
               …
               <nom_de_remplacementN> ;
```

Exemples :

1. ***Type synonyme d'un type simple*** *:*

```
typedef int entier, boolean ;
typedef float reel ;
entier e1 = 23, te[50] = {1, 2, 3, 4, 5, 6, 7} ;
int i ;
i = e1 + te[20] ;
te[20] = i - 60 ;
```

2. ***Type synonyme d'un type tableau*** *:*

```
typedef int tab[10] ;
tab tt; /* tt est un tableau de 10 entiers */
typedef float matrice[10][20] ;
matrice a ;
```

3. ***Type synonyme à une structure*** *:*

```
typedef struct
              {
               int jour ;
               int mois ;
               int annee date ;
              } date ;
date d, *ptd ;
```

4. ***Type synonyme d'un pointeur*** *:*

```
typedef char *chaine ;
chaine ch ;
```