Types structurés

Présentation

- Les types structurés permettent de regrouper au sein d'une même entité des variables ayant entre elles des liens fonctionnels
  - Un type structuré est défini par la liste des variables qu'il contient, appelés « champs ».
- Ils améliorent la concision et la lisibilité des programmes
  - Simplifient la définition des variables
  - Simplifient le passage de paramètres

Structure (1)

- Définie par le mot clé struct, suivi du nom donné à la structure et de la liste de ses champs, entre accolades

```
struct Point_ {
   double x,y,z;            /* Coordonnées spatiales      */
};
struct Planete_ {
   inttype;                 /* Type de planétoïde         */
   double m ;               /* Masse du planétoïde        */
   struct Point_ pos;       /* Position du planétoïde     */
};
```


Structure (2)

- On définit les variables de types structurés comme les variables de types simples

```
struct Planete_ p1,p2;
```

- On peut aussi définir des variables lors de la définition de la structure

```
struct Planete_ {
   int type;                /* Type de planétoïde         */
   double m ;               /* M asse du planétoïde       */
   struct Point_ pos;       /* Position du planétoïde     */
}p1,p2;                     /* Voilà pourquoi le ";"...   */
```

Structure (3)

- On accède aux champs d'une structure au moyen de l'opérateur «.»

```
struct Planete_ plandat;
double masse;
plandat.m = 1e30;
plandat.pos.x = 12.34;
plandat.pos.y = 56.78;
plandat.pos.z = 90.12;
...
masse = plandat.m ;
```


Structure (4)

- On accède aux champs d'une référence de structure au moyen de l'opérateur « -> »

sptr -> champ ⇔ (*sptr).champ

```
struct Planete_ * planptr;
double masse;
planptr->m =1e30;          /*planptr est une référence, donc "->" */
planptr->pos.x = 12.34;    /* planptr->pos est un struct Point_ , */
planptr->pos.y = 56.78;    /* donc on utilise "."                 */
planptr->pos.z = 90.12;
...
masse = planptr->m ;
```

Structure (5)

- On peut utiliser les variables de type structure comme des variables de types simples

```
struct Planete_ p1,p2;
p1.pos.x = 12.34;
p2 = p1; /* Équivalent à une copie mémoire entre zones */
```

- On peut initialiser des structures en utilisant les accolades pour chaque structure et sous structures

```
struct Planete_ p1 = { 1,1e30,{ 12.34,56.78,90.12 } };
```


Structure (6)

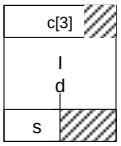
- Pour passer des types structurés en paramètres d'une fonction, il vaut mieux utiliser le passage par référence
  - Pas de recopie de l'intégralité des structures dans la pile lors de l'appel

```
void planeteA ffiche (struct Planete_ * planptr) {
   printf ("Type :% d\nM asse :% lf\nPosition :(% lf,% lf,% lf)\n",
           planptr->type,planptr->masse,
           planptr->pos.x,planptr->pos.y,planptr->pos.z);
}
...
planeteAffiche (&plandat);
```

Structure (7)

- Les données de la structure sont placées les unes après les autres en mémoire dans l'ordre dans lequel elles sont définies, aux alignements de types près
- La taille de la structure peut être supérieure à la somme des tailles de ses membres
- La taille d'une structure peut varier selon la machine

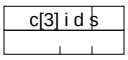
```
structBrol_ {
   char c[3];
   int i;
   double d;
   short s;
}
```



Union (1)

- Définie par le mot clé union, suivi du nom donné à la structure et de la liste de ses champs, entre accolades
- Analogue à la structure, sauf que les données partagent le même espace mémoire
  - Sert à stocker différents types de données dans le même objet sans consommer d'espace inutile

```
unionBrol_ {
   char c[3];
   int i;
   double d;
   short s;
}
```



Union (2)

```
enum Type_ {
  TYPE_ENTIER,
  TYPE_FLOTTANT
};
struct Valeur_ {
  enum Type_ type;
  union {
      int i;
     double d;
  } val;
};
...
switch(valptr->type) {/*valptr est de type struct Valeur_ **/
  caseTYPE_ENTIER :
     printf ("% d",valptr->val.i);
     break;
  caseTYPE_FLOTTANT :
     printf ("% lf",valptr->val.d);
     break;
  }
```


Types énumérés (1)

- Une énumération est un sous-ensemble du type int auquel est associé un nombre fini de valeurs symboliques

```
enum Feu_ {              /* N om de l'énumération */
            FEU _VERT,          /* Constantes symboliques
*/
     FEU _O RA N GE, /* En majuscules de préférence */
     FEU _RO U GE } ; /* Ici le ";" est obligatoire */
}
```

Types énumérés (2)

- On définit des variables de types énumérés comme des variables de types simples
  enum Feu_ f1,f2;
- On peut aussi définir des variables lors de la définition de la structure

```
enum Feu_ {
    FEU _VERT,
    FEU _ORANGE,
    FEU _ROUGE
}f1,f2;          /* Voilà pourquoi le ";"... */
```


Types énumérés (3)

- On utilise les valeurs symboliques comme des constantes numériques

```
switch (f1) {
  case FEU_VERT :
          je_passe ();
          break;
  case FEU_ORANGE :
          if (trop_pres ()) {
             je_passe ();
             break;
          }
  default :                    /* FEU_ROUGE */
          je_stoppe ();
  }
if (f2 == FEU_ORANGE)
```

Types énumérés (4)

- Par défaut, les constantes sont numérotées consécutivement à partir de 0
- On peut spécifier explicitement leur valeur au moyen de l'opérateur « = »

```
enum Premiers10_ {
        UN = 1,         /* Cette constante vaut 1 et pas 0          */
        DEUX,           /*On numérote consécutivement, donc 2*/
        TROIS,          /*Pareil:  celle-là vaut 3                 */
        CINQ = 5,       /* Celle-ci vaut 5                         */
        SEPT= 7         /*Et celle-là vaut7                        */
};
```


Définition de types (1)

- • n définit un nouveau type au m oyen du m ot clé typedef typedef ancien_type nouveau_type • Perm et de dissocier la définition des variables de leur im plém entation – Typage fonctionnel et non plus m atériel – M asquage de l'im plém entation typedef float A ge; /* Définition :un A ge est un float */ m ain (){ A ge a; /* n déclare a comme un A ge */ a = 18.5; /* n utilise a comme un float */ }

Définition de types (2)

- typedef perm et de cacher l'im plém entation du nouveau type ainsidéfini (type simple,structure,union ou énum ération)

- struct Point_ { /* Définition et om m age de la structure */ double x; double y;
  - }; typedef struct Point_ Point; /* Définition du type */ m ain (){
  - Point p; /* Définition d'un point; ne sait pas ce que c'est */
  - p.x = 12.34;
- p.y = 45.67; }

Définition de types (3)

- n peut effectuer la définition de la structure et du type enm êm e tem ps
- typedef struct Point_ { /* Définition de la structure */ double x; double y;
- } Point; /* N om m age du type */
- typedefenum Booleen_ { FAU X, VRAI
- } Booleen;
- typedef struct Cellule_ { structCellule_ *precptr;/* "struct" cartypepasdéfini*/ struct Cellule_ * suivptr; CelluleDonnées celldat;

- } Cellule; /* À partir d'icile type est défini*/

Exercices

- • U n étudiant est définipar un num éro,un nom et un prénom . En plus,un étudiant a une m oyenne. U ne classe est un ensem ble d'au plus 30 étudiants,proposer des structures de données pour représenter cette classe. écrire les fonctions perm ettant de calculer la m oyenne de la classe,la m eilleur note,la plus m auvaise note,…