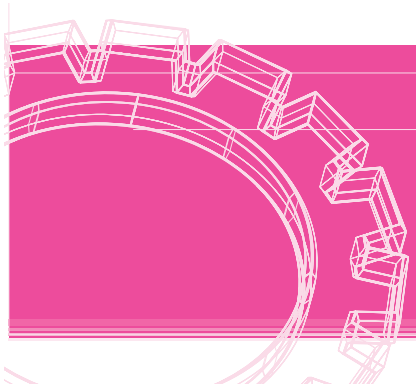
Chapitre 5

Les structures



Rappels

Déclaration d'un type structure et des variables de ce type

C++ permet de déclarer un type structure, de cette manière :

```
struct enreg {  int numero ;
                int qte ;
                float prix ;
             } ;
```

Cette déclaration définit un **type (modèle) de structure** mais ne réserve pas de variables correspondant à cette structure. Ce type s'appelle ici `enreg` et il précise le nom et le type de chacun des champs constituant la structure (`numero`, `qte` et `prix`).

Une fois un tel type de structure défini, nous pouvons déclarer des variables du type correspondant. Par exemple, l'instruction :

```
enreg art1, art2 ;
```

réserve deux emplacements nommés `art1` et `art2`, de type `enreg`, destinés à contenir chacun deux entiers et un flottant.

Les champs d'une structure peuvent être de n'importe quel type de base, d'un type tableau ou structure. De même, les éléments d'un tableau peuvent être d'un type structure.

Utilisation d'une (variable de type) structure

Un champ d'une structure peut être manipulé comme n'importe quelle variable du type correspondant. On désigne un champ donné en faisant suivre le nom de la variable structure de l'opérateur « point » (.), suivi du nom de champ, comme dans ces exemples utilisant les déclarations précédentes :

```
art1.numero    // champ numero de la structure art1
art2.prix      // champ prix de la structure art2
```

Contrairement au tableau, une variable de type structure peut être affectée à une variable de même type (même nom de modèle).

Initialisation des structures

Les structures de classe automatique ne sont pas initialisées par défaut. Celles de classe statique voient leurs champs initialisés à « zéro » (entier nul, flottant nul, caractère de code nul, pointeur nul). En toute rigueur, cette règle s'applique aux champs qui sont des scalaires ou des tableaux de scalaires. Si certains champs sont eux-mêmes des structures, la règle s'appliquera à chacun de leurs champs, et ainsi de suite.

À l'instar d'un tableau, une structure peut être initialisée lors de sa déclaration, comme dans cette instruction qui utilise le type `enreg` défini précédemment :

```
enreg art1 = { 100, 285, 200 } ;
```

On peut omettre certaines valeurs.

Exercice 47

Énoncé

Soit le modèle (type) de structure suivant :

```
struct point
{  char c ;
   int x, y ;
} ;
```

Écrire une fonction qui reçoit en argument une structure de type `point` et qui en affiche le contenu sous la forme :

```
point B de coordonnées 10 12
```

a. en transmettant en argument la **valeur** de la structure concernée,

b. en transmettant en argument l'**adresse** de la structure concernée,

c. en transmettant la structure concernée par **référence**.

Dans les trois cas, on écrira un petit programme d'essai de la fonction ainsi réalisée.

Solution

a. Voici la fonction demandée :

```
void affiche (point p)
{   cout << "point " << p.c << " de coordonnées "
         << p.x << " " <<p.y << "\n" ;
}
```

Notez que sa compilation nécessite obligatoirement la déclaration du type `point`.

Voici un petit programme complet qui affecte les valeurs `'A'`, `10` et `12` aux différents champs d'une structure nommée `s`, avant d'en afficher les valeurs à l'aide de la fonction précédente :

```
#include <iostream>
using namespace std ;

struct point
{  char c ;
   int x, y ;
} ;
void affiche (point p)
{   cout << "point " << p.c << " de coordonnées "
         << p.x << " " <<p.y << "\n" ;
}
main()
{   void affiche (point) ;   // déclaration  de affiche
    point s ;
    s.c = 'A' ;
```

```
    s.x = 10 ;
    s.y = 12 ;
    affiche (s) ;
}
```

b. Voici la deuxième fonction demandée, accompagnée de son programme de test :

```
#include <iostream>
using namespace std ;
struct point
{  char c ;
   int x, y ;
} ;
void affiche (point * adp)
{   cout << "point " << adp->c << " de coordonnées "
         << adp->x << " " << adp->y ;
}
main()
{   void affiche (point *) ;
    point s ;
    s.c = 'A' ;
    s.x = 10 ;
    s.y = 12 ;
    affiche (&s) ;
}
```

Notez que l'on doit, cette fois, faire appel à l'opérateur `->`, à la place de l'opérateur « point » (`.`), puisque l'on travaille avec un pointeur sur une structure, et non plus avec la valeur de la structure elle-même. Toutefois l'usage de `->` n'est pas totalement indispensable, dans la mesure où, par exemple, `adp->x` est équivalent à `(*adp).x`.

c. Voici la troisième fonction demandée, accompagnée de son programme de test :

```
#include <iostream>
using namespace std ;

struct point
{  char c ;
   int x, y ;
} ;
void affiche (point & p)
{   cout << "point " << p.c << " de coordonnées "
         << p.x << " " << p.y ;
}
main()
{   void affiche (point &) ;
    point s ;
    s.c = 'A' ;
    s.x = 10 ;
    s.y = 12 ;
    affiche (s) ;
}
```

Remarque

Au lieu d'affecter des valeurs aux champs `c`, `x` et `y` de notre structure `s` (dans les trois programmes d'essai), nous pourrions (ici) utiliser les possibilités d'initialisation offertes par le langage C, en écrivant :

```
point s = {'A', 10, 12} ;
```

Exercice 48

Énoncé

Soit le type structure `enreg` défini ainsi :

```
const int NMOIS = 12 ;
struct enreg
{  int stock ;
   float prix ;
   int ventes [NMOIS]
}
```

Écrire une fonction nommée `raz` qui « met à zéro » les champs `stock` et `ventes` d'une structure de ce type, transmise en argument. La fonction ne comportera pas de valeur de retour.

Écrire un petit programme d'essai qui affecte tout d'abord des valeurs aux différents champs d'une telle structure, avant de leur appliquer la fonction `raz`. On affichera les valeurs de la structure, avant et après appel (on pourra s'aider d'une fonction d'affichage).

Solution

Ici, pour que la fonction puisse modifier la valeur d'une structure reçue en argument, il est nécessaire qu'elle en reçoive, soit la référence, soit l'adresse. Voici un exemple complet utilisant la première possibilité :

```
#include <iostream>
using namespace std ;
const int NMOIS = 12 ;
struct enreg
{  int stock ;
   float prix ;
   int ventes [NMOIS] ;
} ;
void raz (enreg & s)
{ s.stock = 0 ;
  for (int i=0 ; i<NMOIS ; i++)
    s.ventes[i] = 0 ;
  return ;
}
```

```
void affiche (enreg  s)   // transmission par valeur ici
{ cout << "stock : " << s.stock << "\n" ;
  cout << "prix :  " << s.prix << "\n" ;
  cout << "ventes : " ;
  for (int i = 0 ; i<NMOIS ; i++)  cout << s.ventes[i] << " " ;
  cout << "\n" ;
}

main()
{ void raz (enreg &) ;
  enreg e = {12, 5.25, {12, 23, 4, 8, 4, 9, 5, 2, 7, 2, 8, 7} } ;
  cout << "contenu avant raz :\n" ;
  affiche (e) ;
  raz (e) ;
  cout << "contenu apres raz :\n" ;
  affiche (e) ;
}
```

Voici un exemple d'exécution de ce programme :

```
contenu avant raz :
stock : 12
prix :  5.25
ventes : 12 23 4 8 4 9 5 2 7 2 8 7
contenu apres raz :
stock : 0
prix :  5.25
ventes : 0 0 0 0 0 0 0 0 0 0 0 0
```

À titre indicatif, voici ce que serait notre fonction `raz`, avec une transmission par pointeur :

```
void raz (enreg * ads)
{ ads->stock = 0 ;
  for (int i=0 ; i<NMOIS ; i++)
    ads->ventes[i] = 0 ;
  return ;
}
```

Dans la fonction `main`, sa déclaration et son appel deviendraient :

```
void raz (enreg *) ;
raz (&e)
```

Exercice 49

Énoncé

Soit le type structure suivant, représentant un point d'un plan :

```
struct point { char c ;     // nom attribué au point
               int x, y ;   // ses coordonnées
             }
```

Écrire une fonction qui reçoit en argument l'adresse d'une structure du type `point` et qui renvoie en résultat une structure de même type correspondant à un point de même nom et de coordonnées opposées.

Écrire un petit programme d'essai.

Solution

Voici ce que pourrait être notre fonction (nous avons reproduit la déclaration de `point`, laquelle pourrait éventuellement figurer dans un fichier en-tête séparé qu'on incorporerait par une directive `#include`) :

```
#include <iostream>
using namespace std ;
struct point
   { char c ;
     int x, y ;
   } ;
point sym (point * adp)
{  point res ;
   res.c = adp->c ;
   res.x = - adp->x ;
   res.y = - adp->y ;
   return res ;
}
```

Notez la « dissymétrie » d'instructions telles que `res.c = adp->c` ; on y fait appel à l'opérateur « `.` » à gauche et à l'opérateur « `->` » à droite (on pourrait cependant écrire `res.c = (*adp).c`).

Voici un exemple d'essai de notre fonction (ici, nous avons utilisé les possibilités d'initialisation d'une structure pour donner des valeurs à `p1`) :

```
main()
{  point sym (point *) ;
   point p1 = {'P', 5, 8} ;
   point p2 ;
   p2 = sym (&p1) ;
   cout << p1.c << " " << p1.x << " " << p1.y << "\n" ;
   cout << p2.c << " " << p2.x << " " << p2.y << "\n" ;
}
```

Remarque Si l'énoncé ne l'avait pas imposé, nous aurions pu transmettre par référence l'argument de la fonction `sym`. Nous aurions pu également utiliser une transmission par valeur, ce qui n'aurait guère été pénalisant pour une information de si petite taille. En théorie, ce choix du mode de transmission (valeur, référence ou adresse) existe également pour la valeur de retour. Toutefois, cette dernière est généralement (comme c'est le cas ici) créée dans une variable locale à la fonction. Dans ces conditions, en transmettre l'adresse ou la référence reviendrait à renvoyer l'adresse de quelque chose destiné à disparaître. En toute rigueur, on pourrait renvoyer l'adresse d'un emplacement alloué dynamiquement, mais encore faudrait-il définir clairement à qui incomberait la responsabilité de sa suppression ultérieure.

Exercice 50

Énoncé

Soit la structure suivante, représentant un point d'un plan :

```
struct point
   { char c ;       // nom du point
     int x, y ;     // coordonnées
   } ;
```

1. Écrire la déclaration d'un tableau (nommé `courbe`) de `NP` points (`NP` supposé défini par une constante).

2. Écrire une fonction (nommée `affiche`) qui affiche les valeurs des différents « points » du tableau `courbe`, transmis en argument, sous la forme :

```
point D de coordonnées 10 2
```

3. Écrire un programme qui :

– lit en données des valeurs pour le tableau `courbe` ;
– fait appel à la fonction précédente pour les afficher.

Solution

1. Il suffit de déclarer un tableau de structures :

```
struct point courbe [NP] ;
```

2. Comme `courbe` est un tableau, on ne peut qu'en transmettre l'adresse en argument de `affiche`. Il est préférable de prévoir également en argument le nombre de points. Voici ce que pourrait être notre fonction :

```
void affiche (point courbe [], int np)
     /* courbe : adresse de la première structure du tableau */
     /*     (on pourrait écrire point * courbe)              */
     /* np : nombre de points de la courbe                   */
{  int i ;
   for (i=0 ; i<np ; i++)
      cout << "point " << courbe[i].c << " de coordonnées "
           << courbe[i].x << " " << courbe[i].x << "\n" ;
}
```

Comme pour n'importe quel tableau à une dimension transmis en argument, il est possible de ne pas en mentionner la dimension dans l'en-tête de la fonction. Bien entendu, comme, en fait, l'identificateur `courbe` n'est qu'un pointeur de type `point *` (pointeur sur la première structure du tableau), nous aurions pu également écrire `point * courbe`.

Notez que, comme à l'accoutumée, le « formalisme tableau » et le « formalisme pointeur » peuvent être indifféremment utilisés (voire combinés). Par exemple, notre fonction aurait pu également s'écrire :

```
void affiche (point * courbe, int np)
{  point * adp ;
   int i ;
   for (i=0, adp=courbe ; i<np ; i++, adp++)
      cout << "point " << (courbe+i)-> c << " de coordonnées "
           << (courbe+i)->x << (courbe+i)->y) ;
}
```

3. Voici ce que pourrait donner le programme demandé :

```
#include <iostream>
using namespace std ;
const int NP = 4 ; // nombre de points d'une courbe
struct point
   { char c ;
     int x, y ;
   } ;
void affiche (point courbe [], int np)
{
   int i ;
   for (i=0 ; i<np ; i++)
      cout << "point " << courbe[i].c << " de coordonnées "
           << courbe[i].x << " " << courbe[i].x << "\n" ;
}
main()
{  point courbe [NP] ;
   int i ;
   void affiche (point [], int) ;
        /* lecture des différents points de la courbe */
   for (i=0 ; i<NP ; i++)
    {  cout << "nom (1 caractère) et coordonnées point " << i+1 << "\n" ;
       cin >> courbe[i].c >> courbe[i].x >> courbe[i].y ;
    }
    affiche (courbe, NP) ;
}
```

Exercice 51

Énoncé

Écrire le programme de la question 3 de l'exercice précédent, **sans utiliser de structures**. On prévoira toujours une fonction pour lire les informations relatives à un point.

Solution

Ici, il nous faut obligatoirement prévoir 3 tableaux différents de même taille : un pour les noms de points, un pour leurs abscisses et un pour leurs ordonnées. Le programme ne présente pas de difficultés particulières (son principal intérêt est de pouvoir être comparé au précédent !).

```
#include <iostream>
using namespace std ;

const int NP = 4 ;            // nombre de points d'une courbe
main()
{  char c [NP] ;               // noms des différents points
   int  x [NP] ;               // abscisses des différents points
   int y  [NP] ;               // ordonnées des différents points
   int i ;

   void affiche (char [], int[], int[], int) ;
        /* lecture des différents points de la courbe */
   for (i=0 ; i<NP ; i++)
   {  cout << "nom (1 caractère) et coordonnées point "
           <<  i+1 << " :\n" ;
      cin >> c[i] >> x[i] >> y[i] ;
   }
   affiche (c, x, y, NP) ;
}

void affiche (char c[], int x[], int y[], int np)
{    for (int i=0 ; i<np ; i++)
       cout << "point " << c[i] << " de coordonnées "
            << x[i] << " " <<  y[i] << "\n";
}
```

Exercice 52

Énoncé

Soient les deux modèles de structure `date` et `personne` déclarés ainsi :

```
cont int LG_NOM = 30 ;
struct date
    { int jour ;
      int mois ;
      int annee ;
    } ;
struct personne
 { char nom [LG_NOM+1] ;  // chaîne de caractères (de style C)
                          //  représentant le nom
   struct date date_embauche ;
   struct date date_poste ;
 } ;
```

Écrire une fonction qui reçoit en argument une structure de type `personne` et qui en remplit les différents champs avec un dialogue se présentant sous l'une des 2 formes suivantes :

```
nom : DUPONT
date embauche (jj mm aa) : 16 1 75
date poste = date embauche ? (O/N) : O


nom : DUPONT
date embauche (jj mm aa) : 10 3 81
date poste = date embauche ? (O/N) : N
date poste (jj mm aa) : 23 8 91
```

Solution

Notre fonction doit modifier le contenu d'une structure de type `personne` ; il est donc nécessaire qu'elle en reçoive la référence ou l'adresse en argument. Ici, l'énoncé n'imposant rien de particulier, nous choisirons une transmission par référence. Voici ce que pourrait être la fonction demandée :

```
void remplit (personne & p)
{   char rep ;           // pour lire une réponse de type O/N
    cout << "nom : " ;
    cin >> p.nom ;       // attention, pas de contrôle de longueur

    cout << "date embauche (jj mm aa) : " ;
    cin >> p.date_embauche.jour
        >> p.date_embauche.mois
        >> p.date_embauche.annee ;

    cout << "date poste = date embauche ? (O/N) : " ;
    cin >> rep ;
```

```
     if (rep == 'O') p.date_poste = p.date_embauche ;
        else  { cout << "date poste (jj mm aa) : " ;
                cin >> p.date_poste.jour
                    >> p.date_poste.mois
                    >> p.date_poste.annee ;
              }
}
```

Voici, à titre indicatif, un petit programme d'essai de notre fonction (sa compilation nécessite les déclarations des structures `date` et `personne`) :

```
main()
{  void remplit (personne &) ; // déclaration remplit
   personne bloc ;
   remplit (bloc) ;
   cout << "nom : " << bloc.nom << "\ndate embauche : "
        << bloc.date_embauche.jour << " "
        << bloc.date_embauche.mois << " "
        << bloc.date_embauche.annee << "\n"
        <<"date poste  : "
        << bloc.date_poste.jour << " "
        << bloc.date_poste.mois << " "
        << bloc.date_poste.annee  ;
}
```