Les structures en langage C++ (même en C)

Dernière modification le **mardi 14 octobre 2008 à 17:40** par Jean-François Pillou.

Différence entre une structure et un tableau

Un tableau permet de regrouper des éléments de même type, c'est-à-dire codés sur le même nombre de bits et de la même façon. Toutefois, il est généralement utile de pouvoir rassembler des éléments de type différent tels que des entiers et des chaînes de caractères.

Les structures permettent de remédier à cette lacune des tableaux, en regroupant des objets (des variables) au sein d'une entité repérée par un seul nom de variable.
Les objets contenus dans la structure sont appelés **champs de la structure**.

Déclaration d'une structure

Lors de la déclaration de la structure, on indique les champs de la structure, c'est-à-dire le type et le nom des variables qui la composent :

```
struct Nom_Structure {
        type_champ1 Nom_Champ1;

        type_champ2 Nom_Champ2;

        type_champ3 Nom_Champ3;

        type_champ4 Nom_Champ4;

        type_champ5 Nom_Champ5;

        ...
};
```

La dernière accolade doit être suivie d'un point-virgule ! Contrairement au langage C, le C++ n'impose pas le mot clé *struct*, ainsi les noms de structures peuvent être manipulés comme de simples noms de variables, ce qui allège l'écriture de telles définitions.

- Le nom des champs répond aux critères des noms de variable
- Deux champs ne peuvent avoir le même nom
- Les données peuvent être de n'importe quel type hormis le type de la structure dans laquelle elles se trouvent

Ainsi, la structure suivante est correcte :

```
struct MaStructure {
        int Age;

        char Sexe;

        char Nom[12];

        float MoyenneScolaire;

        struct AutreStructure StructBis;
/* en considérant que la structure AutreStructure est définie */
};
```


Par contre la structure suivante est incorrecte :

```
struct MaStructure {
        int Age;

        char Age;

        struct MaStructure StructBis;

};
```

Il y a deux raisons à cela :

- Le nom de variable *Age* n'est pas unique
- Le type de donnée *struct MaStructure* n'est pas autorisé

La déclaration d'une structure ne fait que donner l'allure de la structure, c'est-à-dire en quelque sorte une définition d'un type de variable complexe. La déclaration ne réserve donc pas d'espace mémoire pour une variable structurée (variable de type structure), il faut donc définir une (ou plusieurs) variable(s) structurée(s) après avoir déclaré la structure...

Définition d'une variable structurée

La définition d'une variable structurée est une opération qui consiste à créer une variable ayant comme type celui d'une structure que l'on a précédemment déclarée, c'est-à-dire la nommer et lui réserver un emplacement en mémoire.
La définition d'une variable structurée se fait comme suit :

```
struct Nom_Structure Nom_Variable_Structuree;
```

Nom_Structure représente le nom d'une structure que l'on aura préalablement déclarée.
Nom_Variable_Structuree est le nom que l'on donne à la variable structurée.

Il va de soi que, comme dans le cas des variables on peut définir plusieurs variables structurées en les séparant avec des virgules :

```
struct Nom_Structure Nom1, Nom2, Nom3, ...;
```

Soit la structure *Personne* :

```
struct Personne{
        int Age;

        char Sexe;

};
```

On peut définir plusieurs variables structurées :

```
struct Personne Pierre, Paul, Jacques;
```

Accès aux champs d'une variable structurée

Chaque variable de type structure possède des champs repérés avec des noms uniques. Toutefois le nom des champs ne suffit pas pour y accéder étant donné qu'ils n'ont de contexte qu'au sein de la variable structurée...
Pour accéder aux champs d'une structure on utilise l'opérateur de champ (un simple point) placé entre le nom de la variable structurée que l'on a définit et le nom du champ :

```
Nom_Variable.Nom_Champ;
```

Ainsi, pour affecter des valeurs à la variable *Pierre* (variable de type *struct Personne* définie précédemment), on pourra écrire :

```
Pierre.Age = 18;

Pierre.Sexe = 'M';
```

Tableaux de structures

Etant donné qu'une structure est composée d'éléments de taille fixe, il est possible de créer un tableau ne contenant que des éléments du type d'une structure donnée. Il suffit de créer un tableau dont le type est celui de la structure et de le repérer par un nom de variable :

```
Nom_Structure Nom_Tableau[Nb_Elements];
```

Chaque élément du tableau représente alors une structure du type que l'on a défini...
Le tableau suivant (nommé *Repertoire*) pourra par exemple contenir 8 variables structurées de type *struct Personne* :

```
Personne Repertoire[8];
```

De la même façon, il est possible de manipuler des structures dans les fonctions.

Utilité de typedef pour le type struct

(https://c.developpez.com/cours/bernard-cassagne/node116.php)

Lorsqu'on donne un nom à un type structure par `typedef`, l'utilisation est beaucoup plus aisée. En effet, si on déclare :

```
struct personne
   {
   ...
   }
```

les déclarations de variables se feront par :

```
struct personne p1,p2;
```

alors que si on déclare :

```
typedef struct
   {
   ...
   } PERSONNE;
```

les déclarations de variables se feront par :

```
PERSONNE p1,p2;
```

on voit que la seconde méthode permet d'éviter d'avoir à répéter `struct`.

De la même manière, en ce qui concerne les pointeurs, il est plus difficile d'écrire et de comprendre :

```
struct personne
   {
   ...
   };
struct personne *p1,*p2;       /*   p1 et p2 pointeurs vers des struct   */
```

que la version suivante qui donne un nom parlant au type pointeur vers struct :

```
typedef struct
   {
   ...
   } PERSONNE;

typedef PERSONNE *P_PERSONNE;    /*   P_PERSONNE type pointeur vers struct
*/

P_PERSONNE p1,p2;                /*   p1 et p2 pointeurs vers des struct
*/
```

Meilleure utilisation de typedef pour le type struct

Il faut noter que le mot clé typedef permet de créer d'alias pour plusieurs entités dont les structures. La meilleure manière d'utiliser typedef pour les structures est la suivante :

```
typedef struct Coordonnees Coordonnees;

struct Coordonnees { int x; int y; };
```

La première ligne déclare l'alias pour le type struct et les instructions suivantes définissent le contenu de la structure.