



Apprendre C++ avec Qt : Annexe 3

Membres statiques

Normalement, les membres d'une classe n'ont d'existence que dans le contexte d'une instance de cette classe. Il existe toutefois des membres pour lesquels cette règle ne s'applique pas, ce qui leur confère, comme nous allons le voir, des propriétés tout à fait particulières.

1 - Variables membre statiques

Si chaque instance possède son propre exemplaire de chacune des variables membre "ordinaires", une variable membre statique n'existe qu'en un seul exemplaire, quel que soit le nombre d'instances de la classe. Les variables membre statiques sont donc indépendantes du processus d'instanciation de la classe, ce qui complique légèrement leur création en exigeant une dissociation de leur déclaration et de leur définition.

Déclaration

Les variables membre statiques sont déclarées, comme les autres, dans la définition de la classe. Leur seule singularité est ici la présence du mot `static` qui indique leur statut.

```
1 class CAvecStatiques //définition DE LA CLASSE
2 {
3 public:
4     int m_entier; //DECLARATION d'un membre "ordinaire"
5     static int m_entierStatique; //DECLARATION d'un membre statique
6 };
```

Dans la plupart des cas, la définition d'un type (et, en particulier, d'une classe) est faite dans un fichier `.h` qui fera l'objet d'une directive `#include` dans tous les fichiers où la connaissance de cette définition est rendue nécessaire par le fait qu'un objet du type en question y est utilisé.

Définition

La définition d'un type (une classe, en l'occurrence) ne crée aucune variable. Les variables membre "ordinaires" ne sont effectivement créées que lorsque la classe est instanciée. En d'autres termes, leur définition est implicite, car elle est "contenue" dans la celle d'une instance de la classe. Il ne peut en aller de même dans le cas des variables membre statiques, puisque leur création ne doit pas être tributaire de l'instanciation (et ne doit pas être réitérée en cas d'instanciations multiples). Il est donc nécessaire de créer les variables membre statiques en les définissant explicitement.

Si un même objet peut être redéclaré sans inconvénients¹, il n'est en revanche pas question de redéfinir un objet qui existe déjà. Le fait que les fichiers `.h` sont amenés à faire l'objet de directives `#include` dans de multiples fichiers relevant du même projet interdit donc d'y faire figurer des définitions. Par conséquent,

Le lieu naturel de la définition des variables membre statiques est le fichier `.cpp` qui contient la définition des fonctions membre, et non le fichier `.h` définissant la classe elle-même.

La définition d'une variable membre statique exige la mention de son nom complet, mais le mot `static` ne doit pas être rappelé à cette occasion. Dans le cas de notre exemple, cette définition prendrait donc la forme suivante :

```
int CAvecStatiques::m_entierStatique; //définition sans mention du mot static
```

La définition d'une variable membre statique peut s'accompagner d'une initialisation :

```
int CAvecStatiques::m_entierStatique = 0;
```

¹ A condition, toutefois, que toutes les déclarations d'un même objet soient identiques.

Utilisation

On accède normalement à une variable membre statique en utilisant son nom complet :

```
CAvecStatiques::m_entierStatique = 18; //affectation d'une valeur
```

Etant donné que la variable n'est liée à aucune instance particulière, sa désignation se fera de la même façon, qu'elle figure ou non à l'intérieur d'une fonction membre de la classe.

Il est aussi possible d'utiliser une instance de la classe pour accéder à une variable membre statique, exactement comme s'il ne s'agissait pas d'une variable statique :

```
CAvecStatiques uneInstance;  
uneInstance.m_entierStatique = 18;
```

Cette méthode est cependant un peu paradoxale, puisque c'est toujours la seule et unique variable `m_entierStatique` qui est désignée, quelle que soit l'instance mentionnée avant l'opérateur de sélection. Le recours à une instance pour accéder à une variable qui n'a aucun lien particulier avec elle donne une apparence trompeuse à l'instruction, et un lecteur distrait risque fort de mal interpréter cette notation.

Cette façon de procéder crée même un danger encore plus pernicieux. L'expression utilisée à gauche du sélecteur de membre n'est, en réalité, qu'un moyen indirect de mentionner la classe concernée. Dans certains cas, l'évaluation de cette expression peut être incomplète, et les effets éventuellement attendus peuvent donc ne pas être obtenus.

2 - Fonctions membre statiques

Les fonctions membre peuvent, elles aussi, être rendues statiques. L'originalité fondamentale d'une fonction membre statique réside dans le fait que l'usage de son nom complet permet de l'appeler, même en l'absence de toute instance de la classe concernée.

Il est également possible d'appeler une fonction membre statique en utilisant une instance de la classe et un opérateur de sélection. Comme dans le cas des variables membre statiques, cette façon de procéder présente le défaut de ne rien révéler du fait que le membre sélectionné est statique, et que l'instance utilisée ne joue ici qu'un rôle purement syntaxique.

Déclaration

La déclaration d'une fonction membre statique se singularise uniquement par le fait qu'elle commence par le mot `static` :

```
1 class CAvecStatiques  
2 {  
3 public:  
4     int m_entier;  
5     static int m_entierStatique;  
6     static void fonctionMembreStatique();  
7 };
```

Définition

Comme dans le cas de la définition d'une variable membre statique, le mot `static` ne doit pas figurer dans la définition d'une fonction membre statique.

Le caractère statique d'une fonction membre est donc indiscernable à partir de la définition de cette fonction. Seul l'examen de la déclaration de la fonction le révèle.

```
1 void CAvecStatiques:: fonctionMembreStatique()  
2 {  
3     //code de la fonction...  
4 }
```

Le fait qu'une fonction membre statique n'est pas exécutée au titre d'une instance particulière a bien entendu une conséquence majeure : les fonctions statiques ne disposent pas du paramètre caché `this`, et ne peuvent donc pas accéder implicitement aux variables membre d'une instance particulière (de quelle instance pourrait-il bien s'agir, d'ailleurs ?). Les traitements effectués par une fonction membre statiques sont donc limités aux variables membre statiques et aux objets auxquels leurs paramètres leur donnent accès.

3 - A quoi ça sert ?

Les variables membre statiques procurent un espace de stockage commun à toutes les instances de la classe. Un exemple classique d'utilisation de cette possibilité est le comptage d'instances : un compteur qui augmente à chaque instanciation et diminue à chaque disparition d'une instance ne peut évidemment pas être stocké dans une instance (que deviendrait-il lorsque celle-ci disparaît ?). Stocker ce compteur en dehors de la classe n'est pas non plus très satisfaisant : de par sa nature même, la variable en question est très intimement liée à la classe. Une variable membre statique correspond donc idéalement à cette situation.

Les fonctions membre statiques peuvent assurer des traitements dont la signification a un rapport avec la classe, mais qui n'exigent pas la présence d'une instance. Un exemple que nous connaissons déjà est celui de la fonction `QString::number()`, dont le rôle est, justement, de créer "à la volée" une instance de `QString` qui représente la valeur de son paramètre sous la forme d'une suite de chiffres.

Si `QString::number()` était une fonction membre "normale", il faudrait écrire :

```
//il existe une variable x, de type double
QString leNombre; //création d'une instance pour appeler number()
leNombre.number(x);
QString message = "La variable x vaut " + leNombre;
```

Comme `QString::number()` est statique, il est possible de se contenter de :

```
QString message = "La variable x vaut " + QString::number(x);
```