CHAPITRE 9

Identification et spécification des opérations - I

Différents types de fonctions et de méthodes en C++

Fonctions	**Méthodes**
Fonction globale	Méthode virtuelle
Fonction globale amie	Méthode virtuelle pure
Fonction globale statique	Méthode non-virtuelle
Fonction globale template	Méthode statique
Fonction lambda	Méthode template
Fonction membre: Méthode	Méthode const
	Méthode final
	Méthode override

Fonctions globales vs. méthodes

Les fonctions globales sont déclarées à l'extérieur de toute classe alors que les méthodes sont nécessairement déclarées à l'intérieur d'une classe.

Une méthode a accès aux attributs et méthodes de la classe dans laquelle elle est déclarée. Par défaut, une fonction globale n'a accès aux attributs et méthodes d'aucune classe ou objet.

Autant les fonctions globales que les méthodes peuvent recevoir des paramètres (arguments).

Notion de portée

La portée d'une opération spécifie l'ensemble du code source (instructions, méthodes, fonctions) où l'opération est définie et peut être utilisée.

La portée d'une méthode est liée à la visibilité de la méthode:

- Une méthode publique est visible partout où la classe est définie,
- Une méthode privée est définie uniquement dans la classe elle-même,
- Une méthode protégée est définie dans la classe et les classes dérivées.

Notion de portée

La portée d'une fonction non-membre est généralement globale :

- Une fonction est généralement accessible de n'importe quelle autre fonction ou méthode,
- En C/C++, le mot-clé **`static`** permet de restreindre la portée d'une fonction à un seul fichier (*translation unit*),
- Le mot-clé **`friend`** permet de d'augmenter la portée d'une fonction globale ou d'une méthode externe à une classe afin de donner accès à cette fonction ou à cette méthode aux attributs protégés et privés de la classe.

Méthodes de classe vs. méthodes d'instance

En C/C++, le mot-clé **`static`** appliqué à une méthode indique que la méthode est définie au niveau de la classe et non au niveau de chaque instance (objet).

Une méthode définie au niveau des instances reçoit un paramètre caché (**`this`**) donnant accès à l'objet sur lequel la méthode a été invoquée.

Une méthode définie au niveau de la classe ne reçoit pas de paramètre **`this`**.

Utilisation de const

Le mot clé const permet d'imposer une contrainte sémantique sur un objet que le compilateur va automatiquement faire respecter:

Il faut en profiter !

Rappel sur const

Le mot clé `const` indique qu'un objet ne peut être modifié:

```
char* p                = "Bonjour";  // rien de constant
const char* p          = "Bonjour";  // données constantes
char* const p          = "Bonjour";  // pointeur constant
const char* const p    = "Bonjour";  // tout est constant
```

Les utilisations les plus puissantes de `const` viennent de l'utilisation de `const` pour la déclaration d'opérations.

Utilisation de const dans les déclarations d'opérations

Lors de la déclaration d'une opération, on peut déclarer const

- La valeur de retour de l'opération*,
- Chaque paramètre individuellement,
- L'opération elle-même s'il s'agit d'une fonction membre.

** Non-recommandé depuis C++11*

Utilisation de const dans les déclarations d'opérations

Pour les paramètres d'une opération:

- Les paramètres déclarés const vont agir, dans la méthode, comme des objets locaux constants,
- Permet au compilateur d'identifier les occasions d'optimisation, et évite les problèmes d'*aliasing.*

Utilisation de const dans les déclarations d'opérations

Pour l'opération elle-même:

- Seules les méthodes déclarées const pourront être invoquées sur des objets constants,
- Deux méthodes peuvent être surchargées uniquement sur le fait que l'une est const et que l'autre ne l'est pas.

Utilisation de const dans les déclarations d'opérations

Exemple:

```
class Chaine
{
private:
   char* data;
public:
   ...
   // Opérateur pour les objets modifiables
   char& operator[](int pos) { return data[pos]; }
   // Opérateur pour les objets constants
   const char& operator[](int pos) const {return data[pos]; };
}
```

Utilisation de const dans les déclarations d'opérations

Les chaînes constantes et non constantes ne sont pas traitées de la même façon:

```
Chaine str       = "Bonjour";
const Chaine cStr = "le monde";
char c1          = str[0];      // Ok
char c2          = cStr[0];     // Ok
str[0]           = 'x';         // Ok
cStr[0]          = 'x';         // Erreur !
```