

Opérations de classe

Loïc Joly, Michel Simatic, Amina Guermouche

Télécom SudParis

22 Mai 2018

Les constructeurs : constructeur par défaut

- Déclaration :

```
A();  
A(int i = 0);
```

- Appel :

```
A a1;
```

- Un constructeur par défaut est facultatif.
- Il est généré automatiquement si aucun autre constructeur n'est fourni.
- **Remarque** : Il est généré automatiquement uniquement si c'est possible. Si une des données membre a besoin d'un constructeur pour être initialisée, mais que ce constructeur n'est pas fourni, le constructeur par défaut ne sera pas généré.

Les constructeurs : constructeur de copie

- Un constructeur de copie est un constructeur qui prend en paramètre une référence vers un objet de la même classe.
- Les champs de l'objet créé seront initialisés avec les champs de l'objet passé en paramètre.
- Déclaration :

```
A(A const &a)
```

- Appel :

```
void f(A a) {...}  
void g()  
{  
    A a2(a1);  
    A a3 = a1;  
    f(a1);  
}
```

Remarque sur les constructeurs de copie

- Chaque fois qu'un objet passé est passé à une fonction par valeur, le constructeur de copie est appelé.
- Chaque fois qu'un objet est retourné par valeur, le constructeur de copie est appelé.
- Lorsqu'on passe un paramètre par référence ou qu'on retourne une référence, aucun constructeur n'est appelé.
- Un constructeur de copie peut ne pas exister (par exemple pour `unique_ptr`).
- Si un constructeur de copie n'est pas défini, alors il est automatiquement généré (s'il est possible d'en définir un).

L'opérateur =

- L'opérateur = entre deux objets fonctionne de la même façon qu'un constructeur de copie : il y a une affectation membre à membre de l'objet.
- Déclaration :

```
A& operator=(A const &a)
```

- Appel :

```
A a2;  
A a2 = a1;
```

Destructeur

- Un destructeur est automatiquement généré si aucun n'est fourni par l'utilisateur.
- Un destructeur libère les données membres de l'objet puis appelle le destructeur de la classe de base dans le cas d'héritage.
- La destruction se fait dans l'ordre inverse de la création.
- Déclaration :

```
~A();
```

- Appel :

```
auto p = new A;  
delete p;
```

- Remarque : Le constructeur par défaut est défini comme suit :

```
~A() {};
```

Les constructeurs : constructeur par déplacement

- Un constructeur par déplacement fait une copie destructive : l'objet passé en paramètre voit ses membres remis à 0.
- Un constructeur par déplacement est utilisé si on sait que l'objet en paramètre ne sera plus utilisé. Par exemple, s'il s'agit d'une variable temporaire.
- Le constructeur par déplacement est généré automatiquement sauf si le destructeur, le constructeur par copie ou l'opérateur = ne sont pas définis.
- Déclaration :

```
A(A && a) noexcept
```

- Appel :

```
A a = f (); /* constructeur par déplacement */  
A a2 = a; /* constructeur par copie*/
```

Remarque sur les constructeurs par déplacement

- On utilise un constructeur par déplacement lorsque le paramètre est une donnée temporaire.
- Si on veut appeler le constructeur par déplacement sur une variable nommée, on utilise :

```
A a2 = std::move(a1);
```

- Lorsque l'on définit le constructeur par déplacement, il faut faire en sorte qu'il ne lance pas d'exception et ajouter `noexcept`.

Opérateur = par déplacement

- L'opérateur = par déplacement fonctionne comme le constructeur par déplacement.
- Appel :

```
A a ;  
a = f ( ) ;
```

Règles de programmation

- *Règle des 3* (Règle historique) : si on est obligé de définir une des 3 opérations (destructeur, constructeur par copie ou opérateur=) alors on est obligés de définir les 2 autres.
- *Règle des 5* (Règle moderne) : si on est obligé de définir une des 5 opérations (destructeur, constructeur par copie ou opérateur =, destructeur par déplacement, opérateur= par déplacement) alors on est obligés de définir les 4 autres.
- L'idéal est cependant d'utiliser la *règle des 0* en ne définissant aucun et en utilisant des classes prédéfinies où toutes ces opérations sont déjà définies (vector par exemple).