Chapitre 4 : classes, surcharge de fonctions

Fondamentaux du C++

Master 2 Ingénierie mathématique
Sorbonne Université

Automne 2019

Plan

Classes

Les classes

- Elles constituent le principal apport du C++ par rapport au C ;
- Idée : regrouper sous un *même nom* plusieurs variables liées ;

Exemple : calculs dans $\mathbb{C}$, sans les classes

```
double x=1, y=3; // "Déclare" le nombre 1+3i
// Déclaration de la fonction "conjugué"
void conj(double x, double y, double& c_x, double& c_y);
conj(1,-1,x,y); // La "variable" x+iy prend la valeur 1+i
```

Syntaxe *très lourde*.

Classes

Déclaration de classe

```
class complexe{
double x, y;
};// Ne pas oublier le ";"
```

On a déclaré une “classe”, qui se manipulera comme un nouveau type :

```
complexe z; // Déclaration de variable
z.x = 3; // Deux champs de type double
z.y = 1;
complexe conj(complexe); // Conjugaison complexe
z2 = conj(z);
```

Syntaxe plus agréable.

Classes

- `x` et `y` sont appelés des *membres* de la classe `complexe` ;
- Une variable de type `complexe` est appelée une *instance* de cette classe ;
- On peut déclarer des fonctions membres (appelées méthodes).

Déclaration de classe

```
class complexe{
double x, y;
complexe conj(void);
};

z2 = z.conj(); // Conjugué de z
```

Classes : membres privés et publics

Déclaration de classe

```
class complexe{
private:
   double x, y;
public:
   complexe conj(void);
};
```

- Membres publics : accessibles par n'importe quelle fonction.
- Membres privés : accessibles seulement par les méthodes de la classe et par les fonctions "amies" (déclarées comme telles dans la classe).

Intérêt : faire en sorte que les données stockées dans la classe restent cohérentes.
Exemple : pour une classe implémentant les rationnels p/q, on stockera deux entiers p et q en faisant en sorte que les fractions soient irréductibles.

Classes : constructeurs, destructeurs

Toute classe va contenir plusieurs méthodes particulières :

- Les *constructeurs* sont des méthodes qui permettent de créer une instance de la classe à partir de certains paramètres.

```
ma_classe(int);
ma_classe(void);
ma_classe(double, int);
(etc...)
```

(même nom que la classe, pas de type retour).
Un constructeur est appelé à chaque déclaration de variable.

Exemples (complexes)

```
complexe z(x,y); // Le complexe x+iy
complexe z2(x); // Le complexe x (+ 0i)
```

Classes : constructeurs, destructeurs

Toute classe va contenir plusieurs méthodes particulières :

- Il est conseillé qu'au moins un des constructeurs n'ait pas d'argument (constructeur par défaut) :

```
complexe z; // Appel du constructeur par défaut
```

On a aussi un constructeur *par copie*, qui copie une variable dans une autre :

```
ma_classe(ma_classe const&);
```

(prend en argument une référence sur un objet constant, pas de type retour). Exemple :

```
complexe z2 = z; // Appel du constructeur par copie
```

Classes : constructeurs, destructeurs

Toute classe va contenir plusieurs méthodes particulières :

- Le *destructeur* est une méthode qui est appelé automatiquement à chaque fois qu'une instance de classe est effacée, par exemple lorsque l'on quitte une fonction dans laquelle une instance locale avait été déclarée.

```
~ma_classe (void);
```

(pas d'arguments, et pas de type de retour).

Classes : constructeurs, destructeurs

Toute classe va contenir plusieurs méthodes particulières :

- L'opérateur d'affectation, ou opérateur = : copie le contenu d'une instance dans une autre.

```
ma_classe& operator=(ma_classe const&);
x = y; // Copie de y dans x.
```

(prend en argument une référence constante sur un objet de la classe et renvoie une référence sur un objet de la classe).

Classes : constructeurs, destructeurs

- Si on ne définit pas de constructeur par copie, de destructeur et d'opérateur d'affectation, ces méthodes sont créées automatiquement.
- Dans certains cas, un des champs de la classe est un pointeur vers une zone mémoire hors de la classe. Dans ce cas, les méthodes créées automatiquement ne fonctionnent pas correctement (fuite de mémoire, ou objets différents partageant la même zone mémoire).
- Il faut alors redéfinir à la main un constructeur par copie, un destructeur et un opérateur d'affectation.

Plan

Surcharge de fonctions

En C++, il est possible de donner le même nom à plusieurs fonctions.

Exemple : somme de deux nombres

```
double somme(double, double);
int somme(int, int);
complexe somme(complexe, complexe);
```

Seule contrainte : pas d'ambiguïté sur l'identité de la fonction à partir des arguments.

Exemple non autorisé

```
double somme(int, int);
int somme(int, int);
somme(2,6); // Ambiguïté sur la fonction appelée
```

Arguments par défaut

Arguments par défaut : surcharge automatique des fonctions. Exemple :

```
double mafonction(int n, int m=2, double x=0.);
```

est équivalent aux *trois* déclarations :

```
double mafonction(int n);
double mafonction(int n, int m);
double mafonction(int n, int m, double x);
```

`mafonction(n)` correspond à `mafonction(n,2,0.)`
`mafonction(n,m)` correspond à `mafonction(n,m,0.)`

Surcharge d'opérateurs

La plupart des opérateurs du C++ (par exemple, +, *, -, /, etc.) sont en fait des fonctions et peuvent être surchargés.

Exemple

```
complexe operator+(complexe, complexe);
z1 + z2; // Équivalent à z1.operator+(z2);
```