

Support de cours

Cours:

Introduction à la programmation orientée objet (en C++)

Vidéo:

W13-03-surchppexterne-CPP-pt1

Concepts (extraits des sous-titres générés automatiquement) :

Nombres complexes. Concept de surcharge des opérateurs. Séquence vidéo précédente. Copies d'un passage. Troisième séquence vidéo. Classe des nombres complexes. Prototype de la fonction operator. Fonction operator. Valeur de retour. Surcharge externe. Classe complexe. Référence constante. Troisième solution. Surcharge interne. Séquence vidéo.



[vers la recherche de séquences vidéo](#)

(dans Introduction à la programmation orientée objet (en C++).)



[vers la vidéo](#)

Center for Digital Education. Plus de matériel de soutien pédagogique ici :

<https://www.epfl.ch/education/educational-initiatives/cede/educational-technologies-gallery/boocs-en/>

Surcharge d'opérateurs : surcharge externe

(Partie 1)

Introduction à la programmation orientée objet (en C++)

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s





Dans la séquence vidéo précédente,

notes

résumé

0m 1s



Surcharge interne et surcharge externe

Presque tous les opérateurs sont surchargeables
(sauf, parmi ceux que vous connaissez, :: et .)

La surcharge des opérateurs peut être réalisée

- ▶ soit à l'extérieur,
- ▶ soit à l'intérieur

de la classe à laquelle ils s'appliquent.

```
Complexe operator+(Complexe, Complexe);
```

```
class Complexe {
public:
    Complexe operator+(Complexe) const;
};
```

- Les opérateurs externes sont des fonctions ;
les opérateurs internes sont des méthodes.

nous avons présenté d'un point de vue très général ce qu'est le concept de surcharge des opérateurs, et nous avons vu qu'il existe la surcharge interne et de la surcharge externe. Commençons dans cette séquence vidéo-ci par regarder en détail ce qu'est la surcharge externe.

notes

résumé

0m 5s



Exemple de surcharge externe

```
Complexe z1;  
Complexe z2;  
Complexe z3;  
// ...  
z3 = z1 + z2;
```

La surcharge externe consiste à définir les opérateurs comme des fonctions. Par exemple, cette fonction `operator+` qui prend ici 2 nombres complexes pour ajouter l'opérateur d'addition `+` à la classe des nombres complexes. Regardons cet exemple en détail, supposons que l'on ait défini une classe `Complexe`

notes

résumé

0m 21s



Exemple de surcharge externe

```
Complexe z1;
Complexe z2;
Complexe z3;
// ...
z3 = z1 + z2;
```

$z3 = \text{operator} + (z1, z2);$

$\text{operator} + (\text{Complexe}, \text{Complexe})$

et que l'on ait ici déclaré 3 instances, z1, z2 et z3 de cette classe, et que l'on souhaite faire l'addition " z3 = z1 + z2 ". Dans le cas de surcharge externe, c'est-à-dire où l'opérateur est une fonction, cela correspond, je vous rappelle, à l'appel " z3 = operator+(...)" avec z1 et z2 comme argument. D'où l'on déduit que le prototype de la fonction operator+ va prendre deux arguments de type Complexe

notes

résumé

0m 37s



Exemple de surcharge externe

```
Complexe z1;
Complexe z2;
Complexe z3;
// ...
z3 = z1 + z2;
```

$z3 = \text{operator} + (z1, z2);$

$\text{Complexe operator} + (\text{Complexe const\&, Complexe i});$

et va retourner ici un Complexe. Voilà donc un prototype possible pour cette surcharge de l'operator+. On peut vouloir optimiser cet appel, et ici éviter les copies d'un passage par valeur, et faire un passage par référence constante,

notes

résumé

1m 13s



Exemple de surcharge externe

```
Complexe z1;
Complexe z2;
Complexe z3;
// ...
z3 = z1 + z2;
```

$z3 = \text{operator} + (z1, z2);$
`const Complexe operator+(Complexe const&, Complexe const&);`

et on peut aussi vouloir, cela sera expliqué dans la troisième séquence vidéo sur la surcharge des opérateurs, ici, rajouter un type de retour qui serait `const Complexe`. Pour l'instant, on le prend comme acquis et cela sera expliqué

notes

résumé

1m 33s



Exemple de surcharge externe

```
Complexe z1;
Complexe z2;
Complexe z3;
// ...
z3 = z1 + z2;
```

$z3 = \text{operator} + (z1, z2);$

const Complexe operator+(Complexe const&, Complexe const&);

const Complexe operator+(Complexe, Complexe const&).

dans la dernière vidéo sur la surcharge des opérateurs. Voilà donc un autre prototype, un deuxième prototype possible pour cet `operator+` défini en surcharge externe. Une troisième solution, encore plus optimale en C++ 2011, et qui vous sera aussi expliquée dans la troisième séquence vidéo sur la surcharge des opérateurs, serait d'avoir le prototype suivant : `const Complexe` comme valeur de retour, `operator+`, un passage ici par valeur, et un passage par référence constante pour le deuxième argument.

notes

résumé

1m 49s



Exemple de surcharge externe

```
Complexe z1;
Complexe z2;
Complexe z3;
// ...
z3 = z1 + z2;
```

```
class Complexe {
public:
    Complexe(double abscisse, double ordonnee)
        : x(abscisse), y(ordonnee) {}
    // ...
    double get_x() const;
    double get_y() const;
    // ...
private:
    double x;
    double y;
};

const Complexe operator+(Complexe z1, Complexe const& z2)
{
    Complexe z3( z1.get_x() + z2.get_x(),
                 z1.get_y() + z2.get_y() );
    return z3;
}
```

Même si le passage par valeur, ici, peut sembler un peu sous-optimal et contre-intuitif, les spécificités de C++ 2011, en particulier la notion de déplacement au lieu de copie, qui sort largement du cadre de ce cours, en font finalement une meilleure option. Ce qui nous donne donc le code suivant pour regarder maintenant les détails. On aurait donc ici la classe `Complexe` qui définirait un complexe, comme par exemple ici 2 `double`, et puis qui aurait un constructeur qui permet d'initialiser ses 2 attributs `x` et `y`,

notes

résumé

2m 25s



Exemple de surcharge externe

```
Complexe z1;
Complexe z2;
Complexe z3;
// ...
z3 = z1 + z2;
```

Complexe z3(1.0, 2.0);

```
class Complexe {
public:
    Complexe(double abscisse, double ordonnee)
        : x(abscisse), y(ordonnee) {}
    // ...
    double get_x() const;
    double get_y() const;
    // ...
private:
    double x;
    double y;
};

const Complexe operator+(Complexe z1, Complexe const& z2)
{
    Complexe z3( z1.get_x() + z2.get_x(),
                 z1.get_y() + z2.get_y() );
    return z3;
}
```

ici en recevant 2 valeurs, par exemple on pourrait construire un complexe comme ceci. Par exemple en passant la valeur 1 et la valeur 2 pour y, ça, c'est le constructeur que l'on a ici. Puis on aurait un tas de méthodes nécessaires pour manipuler les nombres complexes, comme par exemple récupérer la valeur de x et récupérer la valeur de y. Et donc notre surcharge externe de l'operator+, ici, où l'on retrouve le prototype que je vous ai écrit précédemment, s'écrit alors de la façon suivante, il faut bien sûr quand on écrit " z1 + z2 " comme ceci, définir un nouveau complexe z1+z2, le résultat de l'addition de z1 et z2 est une nouvelle valeur qui n'est ni z1, ni z2. On va donc ici définir cette nouvelle valeur, que l'on va construire en utilisant le constructeur à 2 paramètres. Il se trouve que la formule pour l'addition des nombres complexes se fait en récupérant x de z1, et x de z2, et en faisant l'addition de ces 2 x. Et sur la composante y, on récupère les 2 valeurs y, et on fait leur addition. Donc on construit bien ici un nombre complexe qui correspond à la définition de l'addition de z1 et z2, et on retourne ce nombre complexe pour valeur de retour de l'operator+. Pour finir, les plus avancés d'entre vous auront bien sûr noté, mais c'est ici un détail, que l'on peut regrouper ces 2 lignes et par exemple écrire simplement " return Complexe(...) ", ici sans donner de nom, " return Complexe(...) " avec un Complexe anonyme, et ici appel direct au constructeur, mais encore une fois, ceci est avancé, et n'apporte pas grand chose. et n'apporte pas grand chose.

notes

résumé

3m 1s

