

Support de cours

Cours:

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

Vidéo:

W13-02-surchopintro-CPP-pt1

Concepts (extraits des sous-titres générés automatiquement) :

Surcharge des opérateurs. Fonction de votre niveau. Propres opérateurs. But de la surcharge des opérateurs. Classe complexe. Op b. Opérateur d'affichage. Niveau supérieur. Séquence vidéo. Auto-affectation. Fonction operator. Premier argument. Classes habituelles. Résultat de z1. Dernier niveau.



[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 : introduction

(Partie 1)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s





Dans cette séquence vidéo et dans les suivantes,

notes

résumé

0m 1s



Exemple avec les nombres complexes :

```
class Complexe { ... };  
Complexe z1, z2, z3, z4;
```



nous allons nous intéresser à la surcharge des opérateurs. C'est un sujet assez technique qui va nous permettre de définir nos propres opérateurs.

notes

résumé

0m 5s



Exemple avec les nombres complexes :

```
class Complexe { ... };  
Complexe z1, z2, z3, z4;
```

Commençons par un exemple. Voyons à quoi cela peut bien servir. Imaginons que, par exemple, vous ayez défini une classe `Complexe` pour additionner des nombres complexes. Peu importe ici ce dont il s'agit précisément. Ce qui compte de savoir, c'est qu'on peut additionner des nombres complexes. Supposons donc que vous ayez défini quelques nombres complexes : `z1`, `z2`, etc. Comment feriez-vous pour additionner ces nombres complexes ? Par exemple, pour mettre dans `z3` le résultat de `z1 + z2`.

notes

résumé

0m 15s



Exemple avec les nombres complexes :

```
class Complexe { ... };  
Complexe z1, z2, z3, z4;
```

$$z_3 = z_1 + z_2;$$

Naturellement je pense qu'on écrirait quelque chose comme : $z_3 = z_1 + z_2$ Si vous faites ceci avec vos classes habituelles, vous allez avoir une erreur de compilation parce que justement le $+$ n'est pas défini pour z_1 , z_2 , n'est pas défini pour les nombres complexes.

notes

résumé

0m 44s



Exemple avec les nombres complexes :

```
class Complexe { ... };  
Complexe z1, z2, z3, z4;
```

$z_3 = z_1 + z_2;$
 $z_3 = \text{add}(z_1, z_2);$
 $z_4 -$

Donc jusqu'à maintenant ce que vous auriez fait c'est plutôt d'avoir une fonction qui fait l'addition, par exemple : une fonction qui s'appelle « add » et on ferait comme ceci : $\text{add}(z_1, z_2)$ et $z_3 = \text{add}(z_1, z_2)$ pour mettre le résultat de l'addition de z_1 et z_2 dans z_3 . Et si on voulait faire par exemple z_4 qui est le résultat de l'addition

notes

résumé

1m 1s



Exemple avec les nombres complexes :

```
class Complexe { ... };  
Complexe z1, z2, z3, z4;
```

Il est quand même plus naturel d'écrire :

```
z4 = z1 + z2 + z3;  
que z3 = addition(addition(z1, z2), z3);
```

de z_1, z_2, z_3 on serait obligé d'écrire comme premier argument ici on additionne z_1 à z_2 et puis ensuite comme deuxième argument un deuxième appel de la fonction `add` on ajouterait ici z_3 . Avouez que donc cette façon d'écrire est assez peu naturelle et qu'il serait quand même plus naturel de pouvoir écrire $z_4 = z_1 + z_2 + z_3$ c'est plus clair que quelque chose comme ceci avec une fonction `addition`.

notes

résumé

1m 23s





Et bien justement, le but de la surcharge des opérateurs c'est de permettre d'écrire comme ceci d'utiliser le symbole, l'opérateur `+` avec une classe qui nous appartient, une classe que nous avons définie qui s'appelle la classe `Complexe`. De même on peut continuer à imaginer qu'on voudrait afficher des nombres complexes de façon homogène, comme ceci, dire par exemple : affiche le message `"z3 ="` suivi de la valeur du nombre complexe `z3` pour afficher `z3`. Eh bien cette écriture ici n'est pas possible si l'on ne surcharge pas l'opérateur d'affichage. Je pense que c'est bien préférable d'écrire ceci que d'avoir à découper ici un affichage du message `"z3 ="` puis d'appeler une fonction `"affiche"` qu'on aurait définie pour la classe complexe puis ensuite d'afficher comme ça le retour à la ligne.

notes

résumé

2m 1s



En pratique, quelle surcharge des opérateurs ?

Dans votre pratique du C++, vous pouvez, *en fonction de votre niveau* :



Voilà donc à quoi sert la surcharge des opérateurs. Cela nous permet d'étendre l'utilisation des opérateurs usuels, ceux que l'on trouve les plus utiles, aux classes auxquelles on trouve qu'il faut rajouter ces différents opérateurs : addition, affichage, multiplication... en fonction de nos besoins. Ceci dit, la surcharge des opérateurs est un sujet quand même assez technique, assez difficile, car on est vraiment au cœur du langage puisqu'on va redéfinir des opérations élémentaires : les opérateurs.

notes

résumé

2m 49s



Opérateur ?

Rappel : un opérateur est une opération sur un ou entre deux opérande(s) (variable(s)/expression(s)) :

opérateurs arithmétiques (+, -, *, /, ...), opérateurs logiques (**and**, **or**, **not**),
opérateurs de comparaison (==, >=, <=, ...), opérateur d'incrément (++), ...

$z = a + b$
 $(a == b) \text{ and } (x == y)$

ces opérateurs suivant le standard, à un niveau donc bien plus avancé. Le niveau visé par ce cours sera le niveau 2, c'est-à-dire de faire une surcharge des opérateurs relativement simple. Néanmoins, dans les vidéos qui vont suivre nous vous présenterons le niveau 3 et le niveau 4 afin que vous sachiez que ça existe, et si cela vous intéresse de pouvoir approfondir ce sujet. Entrons donc maintenant dans le vif du sujet de cette fameuse surcharge des opérateurs. D'abord qu'est-ce qu'un opérateur ? Je vous rappelle qu'un opérateur c'est les signes que l'on utilise pour représenter les opérations, comme par exemple les opérateurs arithmétiques : quand vous écrivez par exemple $z = a + b$ vous avez l'opérateur de l'addition. Vous avez les opérateurs logiques, par exemple si on écrit : $(a == b) \text{ and } (x == y)$ ça c'est un opérateur : **and**

notes

résumé

4m 13s



Opérateur ?

Rappel : un opérateur est une opération sur un ou entre deux opérande(s) (variable(s)/expression(s)) :

opérateurs arithmétiques (+, -, *, /, ...), opérateurs logiques (**and**, **or**, **not**),
opérateurs de comparaison (==, >=, <=, ...), opérateur d'incrément (++), ...

$z = a + b$
 $(a == b) \text{ and } (x == y)$
 $x <= y$
 $++i$

Vous avez les opérateurs de comparaison par exemple, lorsque vous écrivez : $x <= y$ vous avez encore un opérateur. Vous avez les opérateurs d'auto-incrément, par exemple quand on écrit : $++i$ dans les boucles for.

notes

résumé

5m 13s



Opérateur ?

Rappel : un opérateur est une opération sur un ou entre deux opérande(s) (variable(s)/expression(s)) :

opérateurs arithmétiques (+, -, *, /, ...), opérateurs logiques (and, or, not),
opérateurs de comparaison (==, >=, <=, ...), opérateur d'incrément (++), ...

Un appel à un opérateur est un appel à une fonction ou une méthode spécifique :

$a \text{ Op } b \rightarrow \text{operatorOp}(a, b) \text{ ou } a.\text{operatorOp}(b)$

$\text{Op } a \rightarrow \text{operatorOp}(a) \text{ ou } a.\text{operatorOp}()$

$a + b$

$\text{operator}+(a, b)$

$a.\text{operator}+(b)$

$-a$

$\text{operator}-(a)$

$a.\text{operator}-()$

$++a$

$\text{operator}++(a)$

$a.\text{operator}++()$

Et vous avez aussi l'opérateur d'affection dont je n'ai pas encore parlé mais qui est écrit ici. Quand vous avez $z =$ quelque chose c'est bien un opérateur, cet opérateur aussi est surchargeable, on en parlera tout à la fin de la dernière vidéo. Il faut savoir que quand vous écrivez une expression qui contient un opérateur, l'opérateur $==$ ici, l'opérateur and , etc., en fait cela correspond à l'appel d'une fonction ou l'appel d'une méthode. Chaque fois que vous écrivez $a \text{ Op } b$ c'est soit que vous avez la fonction $\text{operatorOp}(a, b)$ par exemple quand j'écris $a + b$ c'est que j'ai soit la fonction $\text{Operator}+(a, b)$ qui a été appelée soit, et on verra les distinctions plus tard, la méthode de la classe dont a est une instance $a.\text{operator}+(b)$, donc la méthode $\text{operator}+$ ici avec l'argument b . De la même façon, si vous avez des opérateurs unaires c'est-à-dire des opérateurs qu'utilise un seul opérand par exemple quand vous écrivez $-a$ pour prendre l'opposé de a ou quand vous écrivez $++a$ pour incrémenter a c'est soit un appel d'une fonction, par exemple dans le cas de $-a$ ce sera la fonction $\text{operator}-$ avec ici un argument a , ou dans le cas d'une méthode d'une instance de classe ce sera $a.\text{operator}-$ et ici sans argument. De même le $++a$ va soit appeler une fonction, en fonction de ce qui aura été défini, soit une fonction $\text{operator}++(a)$ soit la méthode $\text{operator}++$ de la classe dont a est une instance.

notes

résumé

5m 33s





Pour donner encore d'autres exemples quand vous écrivez `cout << a` c'est soit la fonction `operator <<` qui prend comme argument `cout` et `a` soit parce qu'il faut savoir que `cout` c'est une instance d'une classe qui s'appelle la classe `iostream`, donc soit la méthode `operator <<` de la classe dont `cout` est une instance c'est-à-dire la classe `ostream` avec comme paramètre `a`. Enfin pour finir, il faut savoir que quand vous appelez `a = b` alors on est effectivement dans ce cas-là mais ici vous n'avez pas de fonction `=` avec deux arguments. Là c'est forcément que c'est la méthode `operator =` de la classe dont `a` est une instance qui est appelée avec comme paramètre `b`. Voilà donc pour quelques exemples. L'ensemble des opérateurs que vous pouvez surcharger est donné en complément sous format d'un fichier pdf sur le site du cours. sur le site du cours.

notes

résumé

7m 25s

