

Support de cours

Cours:

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

Vidéo:

### W13-04-surchopinterne-CPP-pt2

Concepts (extraits des sous-titres générés automatiquement) :

**Nouveaux complexes. Façon suivante. Opérateur +. Valeur de z2. Valeur du complexe c1. Exemple concret. Opérateur. Valeur de l'attribut. Premier opérande. Sémantique suivante. Lien sémantique. Valeur de c2. Petit exemple simple. Certaine façon. Entête de l'opérateur.**



[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 interne

## (Partie 2)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s



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

→ ajouter à  $z_1$  la valeur de  $z_2$   
↑  
modifié

Voyons maintenant sur un exemple concret comment un opérateur par exemple « += » peut-être surchargé en interne dans la « class » « Complexe » ; l'opérateur « += » a usuellement la sémantique suivante : « ajouter à  $z_1$  la valeur de  $z_2$  ». Contrairement à « opérateur + » que nous avons eu l'occasion d'étudier précédemment qui avait pour vocation de construire un nouveau « Complexe » à partir de deux « Complexes » existants en les sommant, l'opérateur « += », lui, va modifier son premier opérande, on voit donc que l'opérateur « += », que l'on a choisi en guise d'exemple ici, est un opérateur qui est très proche des objets de la « class » « Complexe »

notes

résumé

0m 1s



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

```
class Complexe {  
public:  
    // ...  
    void operator+=(Complexe const& z2); // z1 += z2;  
    // ...  
};  
  
void Complexe::operator+=(Complexe const& z2) {  
    x += z2.x;  
    y += z2.y;  
}
```

puisque'il doit pouvoir en modifier le contenu et dans ce cas on a usuellement recours à une surcharge interne ce qui va se traduire par le code suivant,

notes

résumé

0m 49s

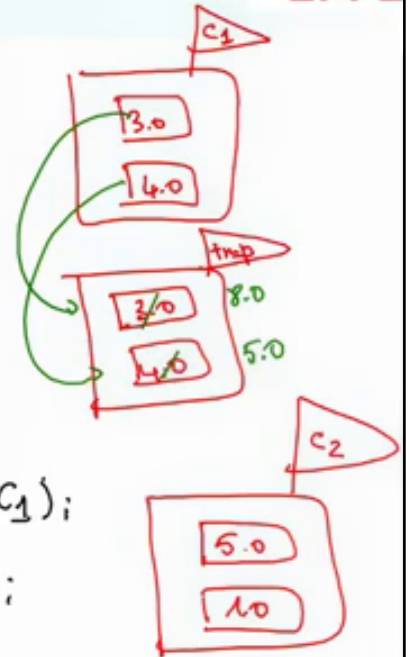




## Retour sur l'addition externe

```
class Complexe {
public:
    // ...
    void operator+=(Complexe const& z2); // z1 += z2;
    // ...
};

const Complexe operator+(Complexe z1, Complexe const& z2) {
    z1 += z2; // utilise l'opérateur += redéfini précédemment
    return z1;
}
```



→ Complexe c1 ( 3.0, 4.0);      → Complexe tmp (c1);

→ Complexe c2 ( 5.0, 1.0);      → tmp += c2;

tmp → 8.0 5.0

valeur de « tmp » en utilisant l'opérateur « += » on se rend compte qu'il suffit de copier dans « tmp » la valeur du complexe c1 puis d'appliquer l'opérateur « += » à « tmp » pour lui ajouter la valeur de c2 ; on obtient bel et bien à l'intérieur du « tmp » la somme de c1 et de c2 en procédant de cette façon. Pour vous en convaincre faisons un petit schéma, la déclaration de ce « Complexe » va correspondre à cette situation mémoire donc on aura un objet de type « Complexe » c1 dont les champs 'x' et 'y' seront initialisés comme ceci. De façon analogue l'objet c2 aura cette allure et l'objet « tmp », qui est construit d'abord en cette ligne par copie de c1, va d'abord avoir dans ses champs 'x' et 'y' les valeurs copiées depuis c1, ensuite on va appliquer à « tmp » l'opérateur « += », donc en exécutant cette ligne on va appliquer l'opérateur « += » qui consiste à ajouter au 'x' de « tmp » la valeur du 'x' de c2 d'où la valeur 8.0 ici et à ajouter au 'y' de « tmp » la valeur du 'y' de c2 d'où la valeur 5.0 ici. On voit donc bien que le « Complexe » calculé ainsi au moyen de l'opérateur « += » correspond bel et bien à la somme de c1 et de c2. C'est exactement cet algorithme qui est appliqué dans la version que vous avez ici sous les yeux, à une subtilité près liée à ce passage par valeur ici ; en effet si l'on utilise cet entête de l'opérateur '+', qui est l'entête préconisé depuis c++ 11, lorsque l'on écrit quelque chose comme c1 + c2, donc on invoque cet opérateur '+', ce qui va se passer c'est que la valeur du

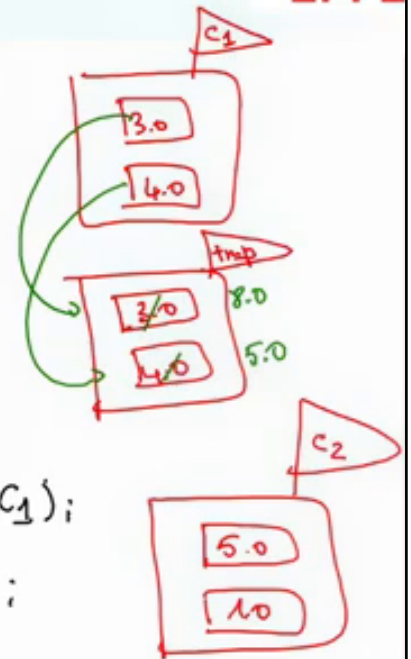
### notes

### résumé

## Retour sur l'addition externe

```
class Complexe {
public:
    // ...
    void operator+=(Complexe const& z2); // z1 += z2;
    // ...
};

const Complexe operator+(Complexe z1, Complexe const& z2) {
    z1 += z2; // utilise l'opérateur += redéfini précédemment
    return z1;
}
```

[illegible]

« Complexe » c1 va être copiée dans z1 qui est une zone locale à opérateur '+', à la fonction « opérateur + », sur laquelle on peut travailler et qui va être retournée au final donc on se rend compte que z1 joue ici exactement le rôle de « tmp » ; il est donc inutile de redéfinir à l'intérieur du corps de « opérateur + » une autre variable intermédiaire dans laquelle on aurait copié le premier opérande ici. Ce rôle est déjà rempli par le paramètre passé par valeur ici. En clair, la copie du premier opérande c1, faite dans une variable intermédiaire « tmp » est réalisée de facto lors du passage par valeur ici. Bien sûr l'algorithme décrit ici est placé à l'intérieur d'une fonction,

## notes

## résumé

## 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;
}
```

il faut qu'une fois le calcul effectué, la valeur calculée soit retournée et c'est exactement ce qui est fait ici. On obtient ainsi une définition de l'opérateur « opérateur + » faite au moyen de « += » qui est très concise, très élégante et qui a surtout le mérite d'établir un lien sémantique entre l'opérateur « += » et l'opérateur '+'. Ces deux opérateurs ne devraient pas être définis de façon découplée, notez aussi que cette façon de définir l'entête en utilisant le passage par valeur permet au compilateur moderne C++ 11 de faire de nombreuses optimisations. Notez pour terminer que la nouvelle version de l'opérateur '+', définie au moyen de « += », a l'avantage aussi de nous épargner l'utilisation des « getters » qui était nécessaire parce que la fonction externe n'avait pas accès aux attributs privés des « Complexes » donc ces « getters » ne sont plus nécessaires et c'est bon pour l'encapsulation on sait que la prolifération de « getters » peut être nuisible à une bonne encapsulation. à une bonne encapsulation.

notes

résumé

6m 25s

