

Support de cours

Cours:

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

Vidéo:

### W12-04-destructeurs-CPP-pt4

Concepts (extraits des sous-titres générés automatiquement) :

**Nombre d'instances d'une classe. Moyen d'un constructeur. Version minimale du destructeur. Exemple concret. Nouvelle version de la classe. Nombre d'instances. Constructeurs de copie. Instance active. Opérateur d'affectation. Variable compteur. Ouverture d'un bloc. Fin du programme. Bon programmeur c. Compteur. Bonne solution.**



[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/>

# Destructeurs

## (Partie 4)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s



La syntaxe de déclaration d'un destructeur pour une classe `NomClasse` est :

```
~NomClasse() { // opérations (de libération) }
```

- ▶ Le destructeur d'une classe est une méthode *sans paramètre*  
⚠ **pas de surcharge possible**
- ▶ Son nom est celui de la classe, précédé du signe `~` (tilda).
- ▶ Si le destructeur n'est pas défini explicitement par le programmeur, le compilateur en génère automatiquement une version minimale.

La question que nous nous posons maintenant est :

notes

résumé

0m 1s



## Exemple

Supposons que l'on souhaite compter le nombre d'instances d'une classe actives à un moment donné dans un programme.

Faut-il toujours se contenter de cette version minimale du destructeur par défaut, lorsque dans la classe, nous ne faisons aucune allocation de ressource. La réponse dans ce qui suit.

### notes

### résumé

0m 5s



## Exemple

Supposons que l'on souhaite compter le nombre d'instances d'une classe actives à un moment donné dans un programme.

```
int main()
{
    // compteur = 0
    Rectangle r1;
    // compteur = 1
    {
        Rectangle r2;
        // compteur = 2
        // ...
    }
    // compteur = 1
    return 0;
} // compteur = 0
```

Partons à nouveau d'un exemple concret, supposons que nous souhaitions compter le nombre d'instances d'une classe, qui sont actives à un moment donné dans un programme. Par exemple, imaginons que nous souhaitions compter le nombre d'instances de « Rectangle » actives dans un programme, on peut imaginer que l'on utilise pour le comptage une variable « compteur » dont nous nous préoccupons de la définition un peu plus tard, et qui fonctionnerait comme ceci. Au départ le compteur est à zéro, nous n'avons aucune instance active. Ensuite ici, nous créons une première instance de « Rectangle » au moyen d'un constructeur par défaut,

### notes

### résumé

0m 16s



## Exemple

Supposons que l'on souhaite compter le nombre d'instances d'une classe actives à un moment donné dans un programme.

Utilisons comme compteur une variable globale de type entier :

- le constructeur incrémente le compteur

```
long compteur(0); /* Hmm.... On y reviendra
                  dans une autre séquence vidéo ! */

class Rectangle {
    //...
    Rectangle(): hauteur(0.0), largeur(0.0) { //constructeur
        ++compteur; }
    // ...
}
```

et nous voudrions que le compteur se mette automatiquement à jour et comptabilise un « Rectangle » actif. Nous avons ici l'ouverture d'un bloc, dans lequel nous déclarons un second « Rectangle », « r2 », et nous aimerions qu'à ce stade le compteur vaille 2, puisque nous aurions à ce stade 2 « Rectangles » actifs, « r2 » et « r1 ». Lorsque nous terminons l'exécution de ce bloc, la variable « r2 » cesse d'exister, donc il n'y a plus qu'une seule instance active de « Rectangle », qui est l'instance « r1 ». Et nous aimerions que le compteur soit mis proprement à jour, et comptabilise une seule instance active. À la fin du programme, nous aimerions à nouveau avoir un comptage à zéro, étant donné que ni « r1 » ni « r2 » ne sont plus désormais 2 instances actives. Comment procéder ? Pour l'instant, à ce stade de votre apprentissage, pour ce qui est de la variable compteur, nous ne savons guère faire mieux que déclarer une variable globale en dehors du « main », ce qui n'est pas une bonne solution et nous aurons l'occasion d'y revenir dans une séquence prochaine.

### notes

### résumé

0m 49s



### Example

```
int main()
{
    // compteur = 0
    Rectangle r1;
    // compteur = 1
    {
        Rectangle r2;
        // compteur = 2
        // ...
    }
    // compteur = 2
    return 0;
} // compteur = 2
```

Pour le reste, l'idée est simple, le constructeur sera en charge d'incrémenter le compteur : à chaque fois que l'on construit un nouveau « Rectangle », on incrémente le compteur. Si l'on se contente de cela,

notes

---

résumé

1m 49s



## Exemple

```
int main()
{
    // compteur = 0
    Rectangle r1;
    // compteur = 1
    {
        Rectangle r2;
        // compteur = 2
        // ...
    }
    // compteur = 2
    return 0;
} // compteur = 2
```

☞ on est obligé ici de **définir explicitement le destructeur**

le comptage ne va pas se faire de façon appropriée, parce qu'on se contente d'incrémenter le compteur à chaque fois qu'une instance commence son existence, mais à nul endroit, nous ne décrétons le compteur. Ici, typiquement, après la création de la première instance, le compteur vaudra 1, après la création de la seconde instance, le compteur vaudra 2, et rien ne viendra décrétement ce compteur ce qui veut dire qu'à la fin du bloc, on aura un comptage erroné de 2 instances au lieu d'une seule, et qu'à la fin du programme, nous aurons toujours un comptage erroné de 2 instances au lieu de zéro.

### notes

### résumé

2m 1s





- ▶ le constructeur incrémente le compteur
- ▶ le destructeur le décrémente

```
long compteur(0); /* Hmm.... On y reviendra
                  dans une autre séquence vidéo ! */

class Rectangle {
    //...
    Rectangle(): hauteur(0.0), largeur(0.0) { //constructeur
        ++compteur; }
    ~Rectangle() { --compteur; } // destructeur
    // ...
}
```

On est donc ici obligés de définir explicitement le destructeur pour faire en sorte que le comptage se fasse de façon appropriée, et que la variable compteur soit décrémentée en fin de vie d'un objet.

notes

résumé

2m 37s



## Exemple

```
int main()
{
    // compteur = 0
    Rectangle r1;
    // compteur = 1
    {
        Rectangle r2;
        // compteur = 2
        // ...
    }
    // compteur = 1
    return 0;
} // compteur = 0
```

Concrètement, ici, en plus du constructeur qui va incrémenter le compteur à chaque fois qu'un « Rectangle » est construit, il faudra, dans le destructeur, décrémenter le compteur à chaque fois qu'une instance termine son existence. C'est donc typiquement une situation où la version par défaut du destructeur, qui aurait un corps vide n'est pas satisfaisante, même si la classe ne fait aucune allocation explicite de ressource. Avec cette nouvelle version de la classe « Rectangle », qui incrémente le compteur dans le constructeur et le décrémenté dans le destructeur, nous avons désormais un comptage qui est fait de façon appropriée. Par exemple, ici, lorsque nous atteignons ce stade de l'exécution, la variable « r2 » n'est plus définie, donc l'objet « r2 » cesse d'exister. Le destructeur va être invoqué automatiquement,

### notes

### résumé

2m 47s



## Exemple

Que se passe-t-il si l'on souhaite utiliser la copie d'objet ?

```
int main()
{
    // compteur = 0
    Rectangle r1;
    // compteur = 1
    {
        Rectangle r2;
        // compteur = 2

        Rectangle r3(r2);
        // compteur = ?? ?
    }
    // compteur = ?
    return 0;
} // compteur = ?
```

et donc le compteur va bien être amené à comptabiliser une seule instance. De même, ici, la variable « r1 » cesse d'exister, n'est plus définie. Le destructeur est automatiquement invoqué et va ramener le compteur à zéro. Qu'en est-il maintenant de cet exemple si nous faisons intervenir aussi les constructeurs de copie ? Par exemple, dans ce bloc, au lieu d'avoir uniquement la déclaration d'une instance « r2 » construite au moyen du constructeur par défaut, j'aurais en plus la déclaration d'une instance « r3 » construite cette fois au moyen du constructeur de copie. Je le vois parce que l'argument passé au constructeur est ici un autre « Rectangle ». Qu'en est-il du comptage ? J'aimerais évidemment ici comptabiliser 3 instances : « r3 », « r2 », « r1 ». Qu'en est-il concrètement de la valeur de ce compteur à ces 3 endroits ? Sauriez-vous dire quelle est la valeur du compteur

### notes

### résumé

3m 37s



## Exemple

Que se passe-t-il si l'on souhaite utiliser la copie d'objet ?

```
int main()
{
    // compteur = 0
    Rectangle r1;
    // compteur = 1
    {
        Rectangle r2;
        // compteur = 2

        Rectangle r3(r2);
        // compteur = ?? 2
    }
    // compteur = 0
    return 0;
} // compteur = -1
```

oops... la copie d'un rectangle échappe au compteur d'instances car il n'y a pas de définition explicite du constructeur de copie

à chacun des endroits où j'ai un point d'interrogation ? Ici, le problème est que dans la classe « Rectangle » nous n'avons fourni aucune définition explicite du constructeur de copie. Donc nous utilisons forcément ici la version par défaut de ce constructeur, qui évidemment, ne va rien faire par rapport au comptage des instances. Du coup, la copie d'un « Rectangle » échappe au compteur d'instances. Ici, typiquement, notre compteur qui valait 2, comme il ne va pas du tout être touché par le constructeur de copie, va rester à 2. Lorsque nous quittons ce bloc, il y a 2 instances qui ne sont plus utilisables, donc le destructeur va être invoqué une fois pour « r2 », une fois pour « r3 », et à chaque fois va décrémenter le compteur. Ce qui veut dire qu'on va décrémenter 2 fois le compteur, et sa valeur vaudra ici zéro ce qui est erroné puisque nous sommes censés comptabiliser toujours cette instance « r1 ». Ici, pour les mêmes raisons, nous allons nous retrouver avec un comptage à - 1,

notes

résumé

4m 25s



Il faudrait donc encore ajouter au code précédent, la définition *explicite* du constructeur de copie :

```
Rectangle(Rectangle const& r)
: hauteur(r.hauteur), largeur(r.largeur)
{ ++compteur; }
```

Règle générale : si on doit toucher à l'un des trois parmi destructeur, constructeur de copie et opérateur d'affectation (=), alors on doit certainement également toucher aux deux autres (ou alors au moins se poser la question !).

(En C++11, on peut ajouter le constructeur de déplacement et l'opérateur de déplacement à cette liste.... ...quand on s'en préoccupe [avancé])

La valeur de notre variable compteur vaudra - 1, au lieu de valoir zéro.

notes

résumé

5m 25s



## Exemple

Que se passe-t-il si l'on souhaite utiliser la copie d'objet ?

```
int main()
{
    // compteur = 0
    Rectangle r1;
    // compteur = 1
    {
        Rectangle r2;
        // compteur = 2

        Rectangle r3(r2);
        // compteur = ?? 2
    }
    // compteur = 0
    return 0;
} // compteur = -1
```

oops... la copie d'un rectangle échappe au compteur d'instances car il n'y a pas de définition explicite du constructeur de copie

Pour bien faire, il faudrait donc ajouter au code précédent la définition explicite du constructeur de copie. Une définition explicite, qui se chargerait de l'incrément du compteur lorsque l'on crée un nouveau « Rectangle » par copie.

notes

résumé

5m 31s



## Exemple

Il faudrait donc encore ajouter au code précédent, la **définition explicite du constructeur de copie** :

```
Rectangle(Rectangle const& r)
: hauteur(r.hauteur), largeur(r.largeur)
{ ++compteur; }
```

**Règle générale** : si on doit toucher à l'un des trois parmi destructeur, constructeur de copie et opérateur d'affectation (=), alors on doit certainement également toucher aux deux autres (ou alors au moins se poser la question !).

(En C++11, on peut ajouter le constructeur de déplacement et l'opérateur de déplacement à cette liste.... ...quand on s'en préoccupe [avancé])

À ce moment là, au moment où l'on invoque le constructeur de copie ici, on a bel et bien incrémentation de notre compteur, et on a un comptage fait de façon correcte, à 3 ici. Le compteur est décrémenté 2 fois, et donc vaudrait 1, ce qui est correct puisqu'on comptabilise désormais cette instance « r1 » correctement. Et de même, le comptage ici, en fin de course, serait correct. Il résulte de cette discussion une règle générale

### notes

### résumé

5m 46s





appliquée par tout bon programmeur C++ et qui est la suivante : si l'on est amené à toucher au destructeur explicitement, au constructeur de copie, ou à l'opérateur d'affectation, que nous aurons l'occasion d'aborder dans une séquence suivante, alors on doit certainement également toucher aux 2 autres, On doit prendre l'ensemble comme un tout ou alors en tout cas, au moins se poser la question de l'impact de la définition de l'un de ces éléments sur les autres. Notez finalement, qu'en C++11, et il s'agit là d'une notion avancée qui sort du cadre de ce cours, on peut ajouter à cette liste, destructeur, constructeur de copie et opérateur d'affectation, qu'il s'agit de considérer comme un tout, également, le constructeur de déplacement, et l'opérateur de déplacement, nous ne les citons ici que par soucis de complétude. Ceci termine notre chapitre sur la construction/destruction d'objet. Nous aurons l'occasion d'y revenir un peu lorsque nous parlerons d'héritage, en attendant, vous allez découvrir la notion de surcharge d'opérateur dans nos prochaines séquences. dans nos prochaines séquences.

#### notes

#### résumé

6m 13s

