

Support de cours

Cours:

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

Vidéo:

### W12-01-constrintro-CPP-pt4

Concepts (extraits des sous-titres générés automatiquement) :

**Attributs d'une classe. Paramètres hauteur. Initialisation de son attribut. Liste d'initialisation. Class rectanglecouleur. Objet de la classe. Milieu de l'appel du constructeur. But de cette liste d'initialisation. Nom de variable. Instance anonyme. Attributs. Rectangle. Constructeur. Objet d'une classe. Façon anonyme.**



[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/>

# Constructeurs (introduction)

## (Partie 4)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s



## Construction des attributs

Que se passe-t-il si les attributs sont eux-mêmes des objets ?

Exemple :  
(à améliorer dans  
une prochaine leçon)

```
class RectangleColore {  
private:  
    Rectangle rectangle;  
    Couleur couleur;  
    //...  
};
```

Le constructeur sert donc à initialiser les attributs. Mais qu'en est-il si les attributs d'une classe sont eux-mêmes des objets ?

notes

résumé

0m 1s



## Construction des attributs

Que se passe-t-il si les attributs sont eux-mêmes des objets ?

Exemple :  
(à améliorer dans  
une prochaine leçon)

```
class RectangleColore {
private:
    Rectangle rectangle;
    Couleur couleur;
    //...
};
```

**mauvaise** solution :

```
RectangleColore(double h, double L, Couleur c)
{
    rectangle = Rectangle(h, L);
    couleur = c;
}
```

Prenons un exemple qui devra être amélioré plus tard quand on aura vu la notion d'héritage où on suppose qu'on a une « class RectangleColore » qui contiendrait comme attribut, dans sa partie privée, un « rectangle » qui est donc lui-même une instance, un objet de la classe « Rectangle » et qui contiendrait par exemple une « Couleur, » pour simplifier ici, on pourrait supposer que « Couleur » est un type énuméré ou est un entier. Comment initialiser ici cet attribut « rectangle » de la classe « RectangleColore » puisque c'est lui même un objet d'une classe et il faudrait appeler le constructeur. Pour ça, on pourrait imaginer, c'est une très mauvaise solution, écrire le constructeur de « RectangleColore » on a donc le même nom ici que la classe « RectangleColore » qui prendrait donc des paramètres hauteur et largeur pour pouvoir initialiser le rectangle et un paramètre « Couleur » pour pouvoir initialiser l'attribut « couleur » et qui dans son corps, dans sa définition, copierait un « Rectangle »

notes

résumé

0m 12s





qu'on aurait créé de façon anonyme qui reçoit le paramètre « h » et le paramètre « L » à son constructeur et qui serait copié dans l'attribut « rectangle » de la classe « RectangleColore » et puis, on pourrait initialiser l'attribut « couleur » en copiant la valeur de la « Couleur » reçue ici. Cette syntaxe, comme ceci, « Rectangle » sans nom de variable ici au milieu de l'appel du constructeur, est tout à fait licite. Cela crée ce que l'on appelle une « instance anonyme », -- elle n'est pas nommée -- de la classe « Rectangle ». Mais ça crée bien effectivement un « Rectangle » qui existe avec sa hauteur et sa largeur lequel sera donc recopié dans l'attribut « rectangle » d'une instance de la classe « RectangleColore » comme ça qui serait créée et initialisée par le constructeur. On voit bien qu'on aura deux « Rectangles » : un premier « Rectangle » anonyme ici qui est initialisé par appel à son constructeur et un deuxième « Rectangle » ici qui est l'attribut « rectangle » de l'instance que l'on est en train d'initialiser dans le constructeur de « RectangleColore ». C'est donc une très mauvaise solution

## notes

## résumé

1m 13s



## Construction des attributs

Que se passe-t-il si les attributs sont eux-mêmes des objets ?

Exemple :  
(à améliorer dans  
une prochaine leçon)

```
class RectangleColore {
private:
    Rectangle rectangle;
    Couleur couleur;
    //...
};
```

**mauvaise** solution :

```
RectangleColore(double h, double L, Couleur c)
{
    rectangle = Rectangle(h, L);
    couleur = c;
}
```

🔊 Il faut initialiser *directement* les attributs en faisant appel à *leurs propres* constructeurs !

à laquelle il va falloir trouver une alternative. Il faudrait que les classes qui ont des objets comme attributs puissent appeler directement le constructeur de leurs attributs. Ici, par exemple, que « RectangleColore »

notes

résumé

2m 24s





appelle directement le constructeur de la classe « Rectangle » lors de l'initialisation de son attribut « rectangle » et ne pas passer comme ça par une copie intermédiaire.

notes

---

---

---

---

---

---

---

---

---

---

résumé

2m 37s



---

---

---

---

---

---

---

---

---

---

Un constructeur devrait normalement contenir une section d'appel aux constructeurs des attributs....

...ainsi que l'initialisation des attributs de type de base.

C'est ce qu'on appelle la « **liste d'initialisation** » du constructeur.

Syntaxe générale :

```
NomClasse(liste_paramètres)
// liste d'initialisation
: attribut1(...), // appel au constructeur de attribut1
...
attributN(...) // appel au constructeur de attributN
{ // autres opérations }
```

Autant mettre les choses au bon endroit, plutôt que passer par une copie. Surtout si l'objet est très volumineux. Pour ça, le C++ offre ce que l'on appelle une « liste d'initialisation » dans les constructeurs. Le but de cette liste d'initialisation est justement d'appeler les constructeurs des attributs [ qui sont des objets ] ; et aussi d'initialiser les attributs qui ne sont pas des objets, qui sont, ce que l'on appelle, des « types de base » comme des « int » et des « double ». C'est une très bonne pratique que d'utiliser cette liste d'initialisation pour initialiser les attributs plutôt que de le faire dans le corps du constructeur. L'écriture générale d'une liste d'initialisation se fait de la façon suivante : entre l'entête du constructeur qui a donc le nom qui est le même nom que le nom de la classe et puis ici, une liste d'éventuels paramètres entre cet entête et puis, la définition du constructeur, nous allons mettre ici la liste d'initialisation qui commence par deux points (':') et puis ensuite, chaque fois

notes

résumé

2m 48s





# Appel aux constructeurs des attributs

## Exemple :

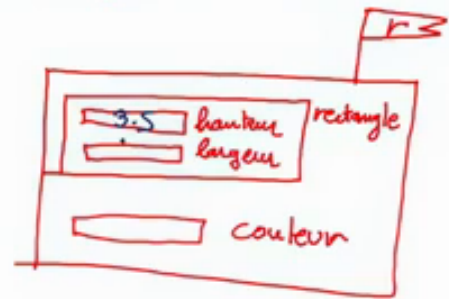
```
class Rectangle {
    Rectangle(double h, double L);
    // ...
};

class RectangleCouleur {

    RectangleCouleur(double h, double L, Couleur c)
    → : rectangle(h, L), couleur(c)
    {}

private:
    Rectangle rectangle;
    Couleur couleur;
};
```

RectangleCouleur r (3.5, 4.5, rouge)



pour chaque attribut, nous allons avoir ici le nom de l'attribut suivi de la syntaxe similaire à celle que l'on avait quand on déclare une instance, c'est-à-dire l'appel du constructeur correspondant à chacun des attributs ici entre parenthèses, suivi d'une virgule s'il y a plusieurs attributs. Et on recommence, comme ça, ainsi de suite pour tous les attributs à appeler les constructeurs ici avec la syntaxe de parenthèses. Donc si je reprends notre exemple de « RectangleCouleur », avec pour rappel « RectangleCouleur » qui a comme attribut un attribut qui s'appelle « rectangle », qui est une instance, un objet, de la classe « Rectangle » ; je vous rappelle aussi que la classe « Rectangle » avait un constructeur qui était ici à deux paramètres pour initialiser la hauteur et la largeur ; et puis donc notre classe « RectangleCouleur », en plus de l'attribut « rectangle » elle a aussi un attribut « couleur ». Donc le constructeur de « RectangleCouleur », c'est ce qui nous intéresse ici, va recevoir des paramètres pour pouvoir initialiser son attribut « rectangle » et puis des paramètres ici pour pouvoir initialiser son attribut « couleur ». Et cette initialisation, il va la faire dans ce qu'on appelle la « liste d'initialisation », ici, qui commence par deux points (':') en appelant le constructeur de la classe « Rectangle », qui prend deux paramètres ici, sur l'attribut « rectangle ». Ici, ce que l'on met c'est le nom de l'attribut « rectangle », c'est le même nom ici bien sûr c'est cet attribut qu'on initialise, donc on met ici le nom de l'attribut « rectangle » et avec une syntaxe comme si on avait déclaré une instance, comme si on avait initialisé une variable, ici l'appel au constructeur correspondant de la classe « Rectangle ». Ensuite, la virgule pour séparer les différentes parties

## notes

## résumé

3m 49s



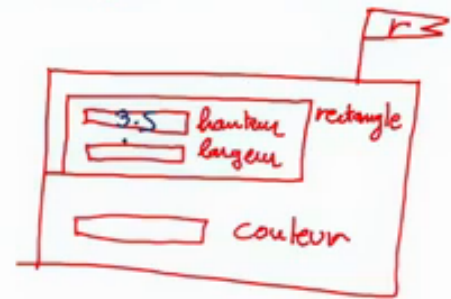
## Exemple :

```
class Rectangle {
    Rectangle(double h, double L);
    // ...
};

class RectangleCouleur {
    RectangleCouleur(double h, double L, Couleur c)
    : rectangle(h, L), couleur(c)
    {}

private:
    Rectangle rectangle;
    Couleur couleur;
};
```

RectangleCouleur r (3.5, 4.5, rouge)



de la liste d'initialisation. Et puis de même, on peut initialiser des attributs, même si c'était un entier ou un double ici des attributs qui ne sont pas des objets avec donc toujours une syntaxe similaire ; ici, entre parenthèses, la valeur que l'on veut passer à l'attribut en conservant toujours le même nom que les noms des attributs. Par exemple, si je veux déclarer une instance « r » d'un « RectangleCouleur » en appelant ici son seul constructeur qui prend trois paramètres ; par exemple, avec une hauteur à 3.5, une largeur à 4.5 et puis une « Couleur » qui par exemple aurait pu être définie comme étant « rouge ». J'ai bien ici déclaration d'une instance avec appel du constructeur. Ce que ça va faire, c'est que ça va créer effectivement en mémoire une instance « r » de « RectangleCouleur » lequel contient un attribut « rectangle » ici de type « Rectangle » qui a lui-même sa hauteur et sa largeur. Puis, un deuxième attribut « couleur » de type « Couleur ». Lors de l'appel à ce constructeur, on va se brancher ici pour exécuter la « section deux points (':') » qui va appeler le constructeur de l'attribut « rectangle » qui est une instance de la classe « Rectangle » auquel on passe le paramètre « h », 3.5 ici, qui va passer au constructeur de « rectangle » pour donc initialiser la valeur de la « hauteur » de l'attribut « rectangle » de l'instance « r » de la class « RectangleCouleur ». De même ici, le 4.5, c'est-à-dire, le paramètre « L » de ce constructeur dans l'exécution de cette liste d'initialisation va passer au constructeur comme deuxième paramètre au constructeur de l'instance attribut « rectangle »

## notes

## résumé



de la classe « Rectangle ». Ici, on l'initialisera à la valeur 4.5 ; et puis le dernier paramètre ici « rouge » qui passe comme « Couleur » va ici, dans cette section d'une liste d'initialisation, initialiser l'attribut « couleur » directement.

notes

---

---

---

---

---

---

---

---

---

---

résumé

7m 49s



---

---

---

---

---

---

---

---

---

---

Cette section introduite par « : » est optionnelle mais **recommandée**.

Par ailleurs :

- ▶ les attributs non-initialisés dans cette section
  - ▶ prennent une valeur par défaut si ce sont des objets ;
  - ▶ restent indéfinis s'ils sont de type de base ;
- ▶ les attributs initialisés dans cette section peuvent (bien sûr) être changés dans le corps du constructeur.

Exemple :

```
Rectangle(double h, double L)
: hauteur(h) //initialisation
{
    // largeur a une valeur indéfinie jusqu'ici
    largeur = 2.0 * L + h; // par exemple...
    // la valeur de largeur est définie à partir d'ici
}
```

L'utilisation de cette section qui commence par « : » que l'on appelle la liste d'initialisation est extrêmement recommandée. Elle permet d'abord comme on l'a vu tout à l'heure d'éviter de faire des copies de tous les attributs qui sont des objets, des instances, d'autres classes. Ensuite, elle permet de regrouper de façon compacte et très lisible l'initialisation de l'ensemble des attributs. Enfin, elle donne une valeur initiale dès le départ avant même la construction de l'objet à chacun des attributs. Bien sûr, ces initialisations peuvent ensuite être réutilisées éventuellement modifiées dans le corps du constructeur mais ceci est extrêmement rare. Je l'illustre ici sur le transparent car c'est quelque chose qui est possible même si c'est quelque chose que vous allez certainement très rarement faire. Par exemple, dans le constructeur de la classe « Rectangle » ici où l'on reçoit les deux paramètres pour initialiser la hauteur et la largeur, on aurait dans la liste d'initialisation initialisé la hauteur ce serait mieux d'initialiser avec la virgule ici la largeur, mais on ne l'a pas fait dans cet exemple pour vous montrer que justement la largeur n'est pas initialisée dans cette portion

notes

résumé

8m 9s



## Notre programme (3/4)

```
// ...
class Rectangle {
public:
    Rectangle(double h, double L)
        : hauteur(h), largeur(L)
    {}
    double surface() const
    { return hauteur * largeur; }
    // accesseurs/modificateurs
    // ...
private:
    double hauteur;
    double largeur;
};

int main()
{
    Rectangle rect1(3.0, 4.0);
    // ...
}
```

du constructeur. A partir de cet endroit-là, la valeur est indéfinie, puis, dès que l'on affecte une valeur à la largeur, évidemment, elle va avoir cette valeur que l'on affecte mais il faut bien comprendre qu'on a une section du constructeur où la largeur n'est pas initialisée. C'est aussi une des raisons pour lesquelles on recommande d'utiliser la « section deux points (':') » qui est exécutée tout de suite, dès le départ avant même de rentrer dans le corps du constructeur. Et donc, à ce stade, tout de suite, dès le départ, avant que l'on commence le corps du constructeur ici, on a initialisé la hauteur avec la valeur « h ». Si l'on suit les conseils, notre programme ressemblerait à ceci dans la déclaration de la classe « Rectangle » dans son interface, on va offrir ici un constructeur qui prend deux paramètres « h » et « L » pour initialiser dans la section liste d'initialisation qui commence par « : » directement la hauteur avec la valeur « h » et la largeur avec la valeur « L », et le corps du constructeur se retrouve vide. C'est quelque chose qui est assez fréquent dans les constructeurs en C++ car on a cette section justement liste d'initialisation qu'il est recommandé d'utiliser. qu'il est recommandé d'utiliser.

### notes

### résumé

9m 13s

