

Support de cours

Cours:

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

Vidéo:

W12-03-constrcopie-CPP-pt2

Concepts (extraits des sous-titres générés automatiquement) :

Constructeur de copie de la classe. Exemple précédent. Copie de r1. Cas de passage. Constructeur de copie. Exemple. Paramètres de la fonction. Appel f de r1. Hauteur de l'instance. Paramètre de la fonction f. Classe rectangle. Référence constante. Cas du rectangle. Largeur de l'instance. Copie.



[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/>

Constructeur de copie

(Partie 2)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s



C++ offre un moyen de créer la **copie** d'une instance :
le *constructeur de copie*

```
Rectangle r1(12.3, 24.5);  
Rectangle r2(r1);
```

r1 et **r2** sont deux *instances distinctes*
mais ayant des mêmes valeurs pour leurs attributs
(au moins juste après la copie).

Autre exemple de copie (invocation du constructeur de copie) :

```
double f(Rectangle r);  
...  
x = f(r1);
```

r1 —

Oui, on a bien ici en effet une copie puisque nous avons ici un passage par valeur, je vous rappelle que dans les passages par valeur les arguments sont donc évalués et sont copiés dans les paramètres de la fonction.

notes

résumé

0m 1s





Donc ici on a bien une copie de r1 qui va donc être copiée dans r, paramètre de la fonction f, et il y a effectivement lors de l'appel f de r1, appel au constructeur de copie de la classe Rectangle.

notes

résumé

0m 25s



Constructeur de copie (2)

Le constructeur de copie permet d'initialiser une instance en *copiant* les attributs d'une *autre instance* du même type.

Syntaxe :

```
NomClasse(NomClasse const& autre) { ... }
```

Exemple :

```
Rectangle(Rectangle const& autre)
: hauteur(autre.hauteur), largeur(autre.largeur)
{}

```

Rectangle r2(1);

Il y a appel au constructeur de copie parce que ici on a un passage par valeur, donc une copie, évidemment en cas de passage par référence il n'y a pas de copie du tout et donc pas d'appel au constructeur de copie. Le constructeur de copie a donc pour but d'initialiser une instance avec une copie d'une autre instance de la même classe. Son prototype est donc parfaitement figé puisqu'il s'agit d'un constructeur, donc son nom est le même nom que le nom de la classe, mais comme il initialise avec une copie d'une autre instance de la même classe il doit recevoir un seul paramètre ici, qui est donc une autre instance de la même classe. On passera cette autre instance par référence constante pour éviter de faire un passage par valeur qui appellerait de nouveau une copie, auquel cas on n'en finirai pas, on ferait des copies de copies de copies, et donc ici on fait bien un passage par référence. Et puis le constructeur de copie ne modifiant pas l'instance qui est reçue, mais il modifie l'instance qu'il est entrain d'initialiser, donc comme il ne modifie pas l'instance qui est reçue, c'est une référence constante. Donc par exemple, dans le cas du Rectangle, on aurait ici un constructeur qui s'appellerait Rectangle pour la classe Rectangle qui recevrait une autre instance de la même classe Rectangle passée par référence constante et il utiliserait bien sûr la section, la liste d'initialisation, la section qui commence par deux points. Ici en initialisant la hauteur de l'instance qu'il est entrain de créer à partir de la valeur de la hauteur de l'autre instance qu'il a reçue comme paramètre, et puis donc en initialisant la largeur de l'instance qu'il est entrain de créer à partir de la copie de la largeur de l'autre instance qu'il a reçue en paramètre. Par exemple, si je reprends l'exemple précédent où j'avais une instance r2

notes

résumé

0m 38s



Le constructeur de copie permet d'initialiser une instance en *copiant* les attributs d'une *autre instance* du même type.

Syntaxe :

```
NomClasse(NomClasse const& autre) { ... }
```

Exemple :

```
Rectangle(Rectangle const& autre)  
: hauteur(autre.hauteur), largeur(autre.largeur)  
{ }
```

Rectangle r2(r1);

qui était créée par copie de l'instance r1, on aurait bien appel du constructeur

notes

résumé

Constructeur de copie (3)

- ▶ Un constructeur de copie est *automatiquement généré* par le compilateur s'il n'est pas explicitement défini (constructeur de copie par défaut)
- ▶ Ce constructeur opère une initialisation *membre à membre* des attributs (si l'attribut est un objet le constructeur de cet objet est invoqué)

copie de surface

Tout se passe comme si le constructeur précédent avait été écrit :

```
Rectangle(Rectangle const& autre)
: hauteur(autre.hauteur), largeur(autre.largeur)
{ }
```

mais il n'est *pas nécessaire* de l'écrire !

de copie. Dans ce cas-là r1 va donc être le paramètre autre ici et donc l'instance r2 va être initialisée et ici ce sera donc bien r2 point hauteur qui recevra r1 point hauteur et r2 point largeur qui recevra r1 point largeur. A noter qu'il est souvent pas du tout nécessaire d'écrire explicitement le constructeur de copie puisque le compilateur C++ nous en fournit un par défaut. Il y a toujours un constructeur de copie fourni par défaut, c'est ce qu'on appelle le constructeur de copie par défaut. Ce constructeur de copie effectue ce qu'on appelle une copie de surface. C'est-à-dire qu'il va recopier un à un les attributs du paramètre de l'autre instance qu'il a reçue pour initialiser l'instance courante. Par exemple dans le cas du rectangle, on n'aurait pas eu à écrire le constructeur précédent puisque dans ce cas simple il est automatiquement généré par le compilateur, il n'est donc pas nécessaire de l'écrire ce constructeur qui recopie l'attribut hauteur de l'autre instance que l'on a reçue comme paramètre dans la hauteur de l'instance que l'on est en train d'initialiser, la largeur de l'autre instance que l'on a reçue dans l'instance courante et et cetera si l'on avait plusieurs autres attributs, voilà comment est construit le constructeur de copie par défaut. le constructeur de copie par défaut.

notes

résumé

2m 37s

