

Support de cours

Cours:

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

Vidéo:

W12-02-constrdefaut-CPP-pt5

Concepts (extraits des sous-titres générés automatiquement) :

Constructeur explicite. Travers de la variante d. Valeur de type double. Fait de la déclaration d'un constructeur. Attributs de type de base. Réalité généralisable. Défaut. Constructeur. Cas de notre variante. Défaut de la classe rectangle. Fait préférable. Suppression d'une méthode. Delete de la méthode. Valeur entière. Simple lecture des constructeurs.



[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 par défaut en C++ (Partie 5)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s



Dès qu'au moins un constructeur a été spécifié, ce constructeur par défaut par défaut n'est plus fourni.

C'est très bien si c'est vraiment ce que l'on veut

(c'est-à-dire forcer les utilisateurs de la classe à utiliser nos constructeurs).

Mais si l'on veut quand même avoir le constructeur par défaut par défaut, on peut le re-demander en écrivant dans la définition de la classe :

```
NomClasse() = default;
```

Exemple (modification du cas (D) précédent) :

```
class Rectangle {  
public:  
    Rectangle() = default; // mais peu pertinent ici  
    Rectangle(double h, double L) : h(h), L(L) {}  
    // suite ...  
};
```

Nous avons illustré au travers de la variante D qu'un constructeur par défaut par défaut n'existe plus dès lors que l'on définit un constructeur explicite dans la classe qu'il soit par défaut ou non. Si l'on souhaite réactiver un constructeur par défaut par défaut c'est possible en C++2011 et à ce moment là il faut adhérer à la syntaxe suivante. On déclare dans la classe, le fait que le constructeur par défaut est la variante par défaut fournie par le compilateur. Si dans le cas de notre variante D vue précédemment on souhaitait faire en sorte que le constructeur par défaut par défaut existe quand même alors qu'il avait disparu du fait de la déclaration d'un constructeur explicite, eh bien il faut spécifier que l'on souhaite le réactiver. Notez bien que dans ce cas, eh bien c'est peu pertinent Pourquoi ? Parce que le constructeur par défaut par défaut, nous l'avons vu, n'initialisait pas les attributs de type de base. Il est toujours non souhaitable d'avoir la possibilité de construire un objet

notes

résumé

0m 1s



Ce que l'on a fait précédemment (= `default`) pour le constructeur par défaut se généralise :

- ▶ à toute méthode (pour laquelle cela est pertinent)
- ▶ à la suppression de méthode, via la syntaxe « = `delete` »

Exemple :

```
class Demo {  
public:  
    double pas_d_int(double x) { ... }  
    double pas_d_int(int) = delete;  
};
```

Un autre exemple sera donné dans la séquence suivante.

où certains attributs sont non-initialisés évidemment. Cela étant, dans toutes les situations où il est pertinent de réactiver le constructeur par défaut par défaut typiquement lorsque les champs sont des objets, nous aurons recours à cette syntaxe. Réactiver une variante par défaut est en réalité généralisable à d'autres méthodes. Nous allons voir d'autres exemples dans les séquences suivantes. C'est aussi généralisable à la suppression d'une méthode auquel cas on utilisera la syntaxe = delete. Par exemple, supposez que dans une classe donnée il existe une méthode particulière qui attend comme paramètre une valeur de type double.

notes

résumé

1m 1s



C++11 autorise les constructeurs d'une classe à appeler n'importe quel autre constructeur de cette même classe

Exemple :

```
class Rectangle {  
private:  
    double hauteur; double largeur;  
public:  
    Rectangle(double h, double L) : hauteur(h), largeur(L) {}  
  
    Rectangle() : Rectangle(0.0, 0.0) {}  
    // bien mieux que le =default précédent  
  
    // suite ...  
};
```

On sait qu'en C++, il est possible de mettre un int dans un double. Donc il est possible normalement d'appeler `pas_d_int` en lui fournissant en guise d'argument une valeur entière. Si l'on veut désactiver cette possibilité, c'est à dire forcer `pas_d_int` à être invoquée systématiquement avec un double, alors à ce moment là, on peut utiliser cette clause qui va faire un delete de la méthode invoquée avec un int et qui ne permettra plus l'invocation de la méthode `pas_d_int` avec une valeur entière en guise d'argument. Enfin notez qu'il est possible depuis C++2011 de faire en sorte qu'un constructeur invoque un autre constructeur de la même classe. Ceci se fait dans la section deux points. Donc ici le constructeur par défaut de la classe `Rectangle` appelle dans la section deux points le constructeur de la même classe qui a besoin de deux arguments. Ceci permettra au constructeur par défaut d'initialiser la hauteur et la largeur à 0. Notez que cette façon de procéder est bien meilleure que celle que nous avons vu précédemment qui consistait à réactiver le constructeur par défaut par défaut. Nous avons vu qu'en effet réactiver le constructeur par défaut par défaut,

notes

résumé

1m 37s



C++11 permet de donner directement une valeur par défaut aux attributs.

Si le constructeur appelé ne modifie pas la valeur de cet attribut, ce dernier aura alors la valeur indiquée.

Exemple :

Rectangle()

```
class Rectangle {  
    // ...  
private:  
    double hauteur = 0.0;  
    double largeur = 0.0;  
    // ...  
};
```

Conseil : préférez l'utilisation des constructeurs.

lorsque les attributs sont de type de base avait le désavantage de laisser ces attributs non initialisés. Ce qui ne sera pas le cas ici. Pour conclure sur l'initialisation, il faut savoir qu'en C++ 11, il est possible d'attribuer des valeurs par défaut aux attributs en le faisant directement à l'endroit où ils sont déclarés. C'est à dire en dehors de tout constructeur. S'il se trouve alors qu'un constructeur appelé ne modifie pas la valeur d'un attribut qui aurait une valeur par défaut ainsi spécifiée alors c'est cette valeur par défaut qui est utilisée. Imaginez par exemple que dans la classe Rectangle nous ayons déclaré un constructeur

notes

résumé

2m 49s



C++11 permet de donner directement une valeur par défaut aux attributs.

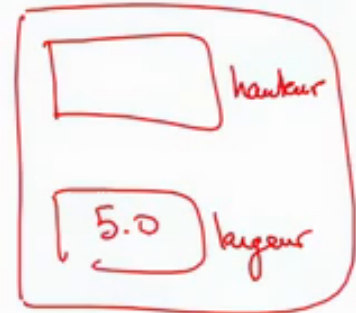
Si le constructeur appelé ne modifie pas la valeur de cet attribut, ce dernier aura alors la valeur indiquée.

Exemple :

```
class Rectangle {
    // ...
private:
    double hauteur = 0.0;
    double largeur = 0.0;
    // ...
};
```

Rectangle() : largeur(5.0) {}

Rectangle r:



Conseil : préférez l'utilisation des constructeurs.

qui initialise uniquement le champ largeur et qui ne fasse rien pour la hauteur. Si ce constructeur est invoqué par une tournure de cette nature, l'objet r ainsi construit aurait donc un champ largeur initialisé avec cette valeur par le constructeur par défaut, à savoir 5. Par contre, pour le champ hauteur puisque rien n'est spécifié dans le constructeur on irait chercher cette valeur-ci

notes

résumé

3m 25s



C++11 permet de donner directement une valeur par défaut aux attributs.

Si le constructeur appelé ne modifie pas la valeur de cet attribut, ce dernier aura alors la valeur indiquée.

Exemple :

```
class Rectangle {  
    // ...  
private:  
    double hauteur = 0.0;  
    double largeur = 0.0;  
    // ...  
};
```

Conseil : préférez l'utilisation des constructeurs.

spécifiée ici et donc la hauteur vaudrait 0. Il est en fait préférable de passer par une initialisation via des constructeurs plutôt que par ce genre de facilité. Pourquoi ? Parce que ceci permet à la simple lecture des constructeurs d'avoir une vision explicite de tout ce qui doit être initialisé. On n'a pas besoin d'aller chercher dans une section de déclaration des attributs si il y a éventuellement des valeurs par défaut qui auraient été attribuées par ailleurs. qui auraient été attribuées par ailleurs.

notes

résumé

4m 1s

