

Support de cours

Cours:

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

Vidéo:

W15-06-polymcoll2-CPP-pt2

Concepts (extraits des sous-titres générés automatiquement) :

Constructeur de copie. Général indésirable. Unique pointers. Copie de surface. Tableau de pointeurs. Différents éléments. Exemple des jeux. Copie du tableau. Copie de pointeurs. Règle d'or. Pointeurs multiples. Pointeurs. Trace de ses pointeurs. Jeu. Exemple.



[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/>

Collections hétérogènes : compléments avancés

(Partie 2)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s



Gare aux pointeurs !

Qui dit «pointeurs», dit aussi « **bonne gestion** » et « **programmation rigoureuse** »...

En particulier pensez, si nécessaire, à la *copie profonde* et au *destructeur* pour libérer la mémoire allouée

lorsqu'on a des pointeurs dans une classe, il faut penser au constructeur de copie, et au destructeur, ainsi qu'à l'opérateur d'affectation.

notes

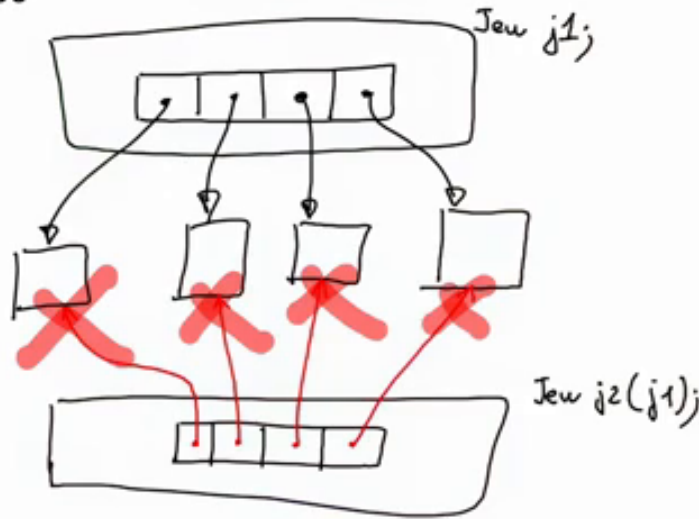
résumé

0m 1s



Qui dit « pointeurs », dit aussi « **bonne gestion** » et « **programmation rigoureuse** »...

En particulier pensez, si nécessaire, à la *copie profonde* et au *destructeur* pour libérer la mémoire allouée



En effet, si on a des pointeurs dans une collection, par exemple, ici dans notre jeu, nous avons un tableau de pointeurs sur des « Personnages ». La question qui se pose c'est : qu'est-ce que ça veut dire de copier un jeu ? Est-ce que l'on fait une copie de surface ? Si ici j'ai un jeu « j2 » qui est une copie de « j1 », il aura donc une copie du tableau de « Personnages », mais si on ne fait qu'une copie de surface, si on n'utilise que le constructeur de copie par défaut, alors on va copier des pointeurs. On va copier les différents éléments uniquement, terme à terme, donc une copie de pointeurs, cela veut dire que le jeu 2 pointera sur les mêmes « Personnages » que le jeu 1. Et ceci est en général indésirable, puisque l'on souhaite en général que les 2 collections aient des éléments qui vivent séparément, ici dans l'exemple des jeux, on souhaiterait pouvoir jouer séparément au jeu 1 et au jeu 2, sans que les personnages de l'un influent les personnages de l'autre. Ce problème n'existe pas avec les « unique pointers » puisqu'avec un « unique pointers », on n'a pas le droit d'avoir des pointeurs multiples sur les mêmes « Personnages »,

notes

résumé

0m 13s





et donc on n'aurait pas du tout le droit de faire une copie. Donc la copie est, de fait, interdite avec un jeu qui contient des « unique pointers ».

notes

résumé

1m 37s



Qui dit «pointeurs», dit aussi « **bonne gestion** » et « **programmation rigoureuse** »...

En particulier pensez, si nécessaire, à la *copie profonde* et au *destructeur* pour libérer la mémoire allouée

Et n'oubliez pas la règle d'or :

c'est celui qui a alloué la mémoire (new) qui est chargé de la libérer (delete)

Par exemple ici, fournir une fonction

```
void Jeu::destruire_personnage(Personnage* qui);
```

ou

```
void Jeu::destruire_personnage(size_t index);
```

ou alors (voire les deux) :

```
void Jeu::destruire_tout();
```

(Rappel : `jeu.ajouter_personnage(new Magicien(●...))`);

Le compilateur aurait en effet interdit l'appel au constructeur de copie, qui cherche à faire des copies de « unique pointers », donc le programmeur se serait rendu compte qu'il y a ici un problème. Par ailleurs, avec des pointeurs « à la C », il ne faut pas oublier la désallocation, et surtout, la règle d'or, que c'est celui qui a alloué la mémoire, celui qui a fait le « new », qui est chargé de la libérer, donc de faire le « delete ». Je vous rappelle qu'on utilisait, par exemple, typiquement comme ceci, ajouter un personnage au jeu, et le personnage est créé par appel à « new », donc par exemple « new Magicien », avec les paramètres pour son constructeur. C'est donc celui qui a fait cet appel ici, qui devra faire la destruction. Ici en l'occurrence, il n'a pas moyen de récupérer le pointeur qu'il a alloué, puisqu'il n'a pas sauvé cette valeur quelque part, et donc il faut offrir le moyen de détruire les « Personnages » que l'on a mis dans notre collection. Par exemple avec une méthode « `destruire_personnage` » où on indiquerait l'adresse du « Personnage » que l'on veut détruire, mais cela supposerait bien sûr que celui qui alloue garde trace de ses pointeurs pour pouvoir les utiliser ici. pour pouvoir les utiliser ici.

notes

résumé

1m 46s

