

Support de cours

Cours:

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

Vidéo:

INIT-CPP-7_6-PointeursIntelligents-pt2

Concepts (extraits des sous-titres générés automatiquement) :

Message essentiel. Zone mémoire. Portions de code différentes. Types de pointeurs intelligents. Façon unique. Biais d'un pointeur de type. Portion de code. Vaste sujet. Cadre simple. Situations de dépendance cyclique. Existence de ce genre d'outils. Emplacement mémoire. Comptabilité des différentes zones. Pointeur classique. Pointeur.



[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/>

Pointeurs «intelligents» (C++11)

(Partie 2)

Initiation à la programmation (C++)

Vincent Lepetit, Jean-Cédric Chappelier et Jamila Sam

...

notes

résumé

0m 0s

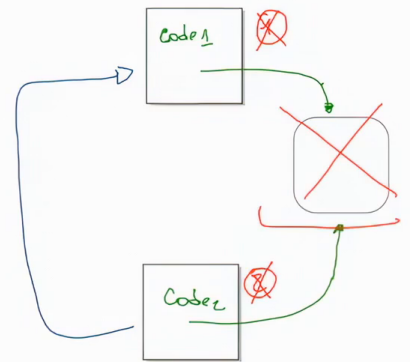


Les `unique_ptr` ne conviennent pas à toutes les situations.

Plus avancé :

- ▶ `shared_ptr` : zone mémoire partagée par plusieurs endroits du code
(sans qu'aucun ne sache quand elle n'est plus utile aux autres)
- ▶ `weak_ptr` : presque comme un `shared_ptr`, mais peut avoir été détruit par ailleurs.
(c'est-à-dire qu'on n'est pas compté dans les utilisateurs de cette zone mémoire).

👉 utile pour «casser les cycles» de `shared_ptr`



Les `unique_ptr` sont très sécurisés, nous venons de voir qu'ils garantissent que chaque pointeur pointe de façon unique sur une zone mémoire et donc, ça implique une utilisation très sûre et donc, leur usage est recommandé dans toutes les situations où on peut le faire. Mais par contre, il est clair qu'ils ne vont pas couvrir tous les besoins, et dans certains cas, il est absolument inévitable de garantir que deux portions de code différentes pointent, utilisent la même zone mémoire. Et donc à ce moment-là, il faudra avoir recours à d'autres types de pointeurs intelligents, comme les `shared_ptr` par exemple. Donc un `shared_ptr` est tel que il peut être utilisé pour pointer sur la même zone mémoire. Donc ici on peut avoir, on peut imaginer la situation où on a une portion de code `code1` et une portion de code `code2` et toutes les deux en fait pointent par le biais d'un pointeur de type `shared_ptr` sur la même zone dynamiquement allouée, qui sera celle-ci. Techniquement, pour qu'il puisse y avoir restitution automatique d'une zone mémoire référencée par des `shared_ptr`, il faut donc qu'il y est une comptabilité des différentes zones, portions de code, qui référencent cet emplacement mémoire. Donc, on ne peut désallouer automatiquement cet emplacement que lorsque plus personne ne l'utilise. Et il peut y avoir des situations de dépendance cyclique qui fait que l'on n'arrive plus à restituer proprement la mémoire en utilisant des `shared_ptr`. À ce moment-là, en combinaison, il va falloir utiliser des `weak_ptr` qui vont permettre de casser les cycles éventuellement induits par les `shared_ptr`. Donc ceci est évidemment très avancé par rapport au contenu du cours. Le message essentiel et que je souhaitais faire passer est l'existence de ce genre d'outils sans entrer en profondeur dans leur utilisation. Ce qu'il faut retenir essentiellement, c'est que si l'on veut utiliser, dans

notes

résumé

0m 1s



notes

1m 49s

