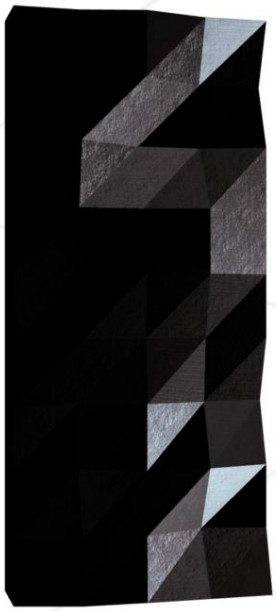

C++ - Module 06

Castings C++

Résumé :
Ce document contient les exercices du module 06 des modules C++.

Version: 6.2

Contenu

Chapitre I

Introduction

C++ est un langage de programmation à usage général créé par Bjarne Stroustrup comme une extension du langage de programmation C, ou « C avec classes » (source : Wikipédia).

L'objectif de ces modules est de vous initier à la programmation orientée objet.
Ce sera le point de départ de votre parcours C++. De nombreux langages sont recommandés pour apprendre la programmation orientée objet. Nous avons décidé de choisir C++ car il est dérivé de votre vieil ami C.
Parce qu'il s'agit d'un langage complexe, et afin de garder les choses simples, votre code sera conforme à la norme C++98.

Nous sommes conscients que le C++ moderne est très différent sur de nombreux aspects. Donc si vous voulez devenir un développeur C++ compétent, c'est à vous d'aller plus loin après le 42 Common Core !

Chapitre II

Règles générales

Compilation

- Compilez votre code avec c++ et les indicateurs -Wall -Wextra -Werror
- Votre code devrait toujours être compilé si vous ajoutez l'indicateur -std=c++98

Conventions de formatage et de dénomination

- Les répertoires d'exercices seront nommés ainsi : ex00, ex01, ... , exn
- Nommez vos fichiers, classes, fonctions, fonctions membres et attributs comme requis dans les lignes directrices.
- Écrivez les noms de classe au format UpperCamelCase . Les fichiers contenant le code de classe doit toujours être nommé en fonction du nom de la classe. Par exemple : ClassName.hpp/ClassName.h, ClassName.cpp ou ClassName.tpp. Ainsi, si vous avez un fichier d'en-tête contenant la définition d'une classe « BrickWall » représentant un mur de briques, son nom sera BrickWall.hpp.
- Sauf indication contraire, tous les messages de sortie doivent être terminés par une nouvelle ligne caractère et affiché sur la sortie standard.
- Au revoir Norminette ! Aucun style de codage n'est imposé dans les modules C++. Vous pouvez suivre votre style préféré. Mais gardez à l'esprit qu'un code que vos pairs évaluateurs ne peuvent pas comprendre est un code qu'ils ne peuvent pas noter. Faites de votre mieux pour écrire un code propre et lisible.

Autorisé/Interdit

Vous ne codez plus en C. Il est temps de passer au C++ ! Par conséquent :

- Vous êtes autorisé à utiliser presque tout ce qui se trouve dans la bibliothèque standard. Ainsi, au lieu de vous en tenir à ce que vous connaissez déjà, il serait judicieux d'utiliser autant que possible les versions C++ des fonctions C auxquelles vous êtes habitué.
- Cependant, vous ne pouvez utiliser aucune autre bibliothèque externe. Cela signifie que les bibliothèques C++11 (et les formes dérivées) et Boost sont interdites. Les fonctions suivantes sont également interdites : *printf(), *alloc() et free(). Si vous les utilisez, votre note sera de 0 et c'est tout.

- Notez que sauf indication explicite contraire, l'espace de noms d'utilisation <ns_name> et Les mots-clés amis sont interdits. Sinon, votre note sera de -42.
- Vous êtes autorisé à utiliser le STL uniquement dans les modules 08 et 09. Cela signifie : pas de conteneurs (vecteur/liste/carte/etc.) et pas d'algorithmes (tout ce qui nécessite d'inclure l'en-tête <algorithm>) jusqu'à ce moment-là. Sinon, votre note sera de -42.

Quelques exigences de conception

- Les fuites de mémoire se produisent également en C++. Lorsque vous allouez de la mémoire (en utilisant le nouveau (mot-clé), vous devez éviter les fuites de mémoire.
- Du Module 02 au Module 09, vos cours doivent être conçus dans le style orthodoxe Forme canonique, sauf indication explicite contraire.
- Toute implémentation de fonction placée dans un fichier d'en-tête (à l'exception des modèles de fonction) signifie 0 pour l'exercice.
- Vous devez pouvoir utiliser chacun de vos en-têtes indépendamment des autres. Ainsi, ils doivent inclure toutes les dépendances dont ils ont besoin. Cependant, vous devez éviter le problème de double inclusion en ajoutant des gardes d'inclusion. Sinon, votre note sera de 0.

Lis-moi

- Vous pouvez ajouter des fichiers supplémentaires si vous en avez besoin (par exemple pour diviser votre code). Comme ces tâches ne sont pas vérifiées par un programme, n'hésitez pas à le faire à condition de rendre les fichiers obligatoires.
- Parfois, les directives d'un exercice semblent courtes, mais les exemples peuvent le montrer. exigences qui ne sont pas explicitement écrites dans les instructions.
- Lisez chaque module dans son intégralité avant de commencer ! Vraiment, faites-le.
- Par Odin, par Thor ! Utilise ton cerveau !!!

Vous devrez implémenter de nombreuses classes. Cela peut paraître fastidieux, à moins que vous ne soyez capable de créer un script dans votre éditeur de texte préféré.

Vous disposez d'une certaine liberté pour réaliser les exercices. Cependant, suivez les règles obligatoires et ne soyez pas paresseux. Vous manque beaucoup d'informations utiles ! N'hésitez pas à lire à propos concepts théoriques.

Chapitre III

Règle supplémentaire

La règle suivante s’applique à l’ensemble du module et n’est pas facultative.

Pour chaque exercice, la conversion de type doit être résolue en utilisant un type de casting spécifique. Votre choix sera vérifié lors de la soutenance.

Chapitre IV

Exercice 00 : Conversion de scalaires types

	Exercice 00
Conversion de types scalaires	
Répertoire de remise : ex00/	
Fichiers à rendre : Makefile, *.cpp, *.{h, hpp}	
Fonctions autorisées : toute fonction permettant de convertir une chaîne en un entier, un flottant ou un double. Cela aidera, mais ne fera pas tout le travail.	

Écrivez une classe ScalarConverter qui contiendra une seule méthode statique « convert » qui prendra comme paramètre une représentation sous forme de chaîne d'un littéral C++ dans sa forme la plus courante et affichera sa valeur dans la série suivante de types scalaires :

- carboniser
- int
- flotter
- double

Comme cette classe n'a pas besoin de stocker quoi que ce soit, cette classe ne doit pas être instanciable par utilisateurs.

À l'exception des paramètres char, seule la notation décimale sera utilisée.

Exemples de littéraux de type char : « c », « a », ...
Pour simplifier les choses, veuillez noter que les caractères non affichables ne doivent pas être utilisés comme entrées. Si une conversion en char n'est pas affichable, affiche un message informatif.

Exemples de littéraux int : 0, -42, 42...

Exemples de littéraux flottants : 0,0f, -4,2f, 4,2f...
Vous devez également gérer ces pseudo-littéraux (vous savez, pour la science) : -inff, +inff

et nanf.

Exemples de littéraux doubles : 0,0, -4,2, 4,2...
Vous devez également gérer ces pseudo-littéraux (vous savez, pour le plaisir) : -inf, +inf et nan.

Écrivez un programme pour tester que votre classe fonctionne comme prévu.

Vous devez d'abord détecter le type du littéral passé en paramètre, le convertir de chaîne en son type réel, puis le convertir explicitement en trois autres types de données. Enfin, affichez les résultats comme indiqué ci-dessous.

Si une conversion n'a aucun sens ou déborde, affichez un message pour informer l'utilisateur que la conversion de type est impossible. Incluez tout en-tête dont vous avez besoin pour gérer les limites numériques et les valeurs spéciales.

```
./convert 0 char :
non affichable int : 0 float : 0,0f
double :
0,0

./convertir nan char:
impossible int: impossible
float: nanf

doublé : en
./convertir 42.0f
char: '*' int: 42

flottant : 42,0f double :
42,0
```

Chapitre V

Exercice 01 : Sérialisation

	Exercice : 01
Sérialisation	
Répertoire de remise :	
ex01/ Fichiers à rendre : Makefile, *.cpp, *.{h, hpp}	
Fonctions interdites : Aucune	

Implémenter une classe Serializer, qui ne sera en aucun cas initialisable par l'utilisateur, avec les méthodes statiques suivantes :

uintptr_t sérialiser(Données* ptr);
Il prend un pointeur et le convertit en type entier non signé uintptr_t.

Données* désérialiser(uintptr_t raw);
Il prend un paramètre entier non signé et le convertit en un pointeur vers Data.

Écrivez un programme pour tester que votre classe fonctionne comme prévu.

Vous devez créer une structure de données non vide (cela signifie qu'elle contient des membres de données).

Utilisez serialize() sur l'adresse de l'objet Data et transmettez sa valeur de retour à deserialize(). Ensuite, assurez-vous que la valeur de retour de deserialize() est égale au pointeur d'origine.

N'oubliez pas de rendre les fichiers de votre structure de données.

Chapitre VI

Exercice 02 : Identifier le type réel

	Exercice : 02
Identifier le type réel	
Répertoire de remise : ex02/	
Fichiers à rendre : Makefile, *.cpp, *.{h, hpp}	
Fonctions interdites : std::typeinfo	

Implémentez une classe de base qui n'a qu'un destructeur virtuel public. Créez trois vides classes A, B et C, qui héritent publiquement de Base.

Ces quatre classes n'ont pas besoin d'être conçues selon le modèle orthodoxe.
Forme canonique.

Implémenter les fonctions suivantes :

Base * generate(void); Il instancie aléatoirement A, B ou C et renvoie l'instance sous forme de pointeur de base. N'hésitez pas à utiliser ce que vous voulez pour l'implémentation du choix aléatoire.

void identify(Base* p); Il imprime le type réel de l'objet pointé par p : « A », « B » ou « C ».

void identifier(Base& p);
Elle affiche le type réel de l'objet pointé par p : "A", "B" ou "C". L'utilisation d'un pointeur à l'intérieur de cette fonction est interdite.

L'inclusion de l'en-tête typeinfo est interdite.

Écrivez un programme pour tester que tout fonctionne comme prévu.

Chapitre VII

Soumission et évaluation par les pairs

Remettez votre devoir dans votre dépôt Git comme d'habitude. Seul le travail effectué dans votre dépôt sera évalué lors de la soutenance. N'hésitez pas à vérifier les noms de vos dossiers et fichiers pour vous assurer qu'ils sont corrects.

16D85ACC441674FBA2DF65190663E136253996A5020347143B460E2CF3A3784D794B
104265933C3BE5B62C4E062601EC8DD1F82FEB73CB17AC57D49054A7C29B5A5C1D8
2027A997A3E24E387