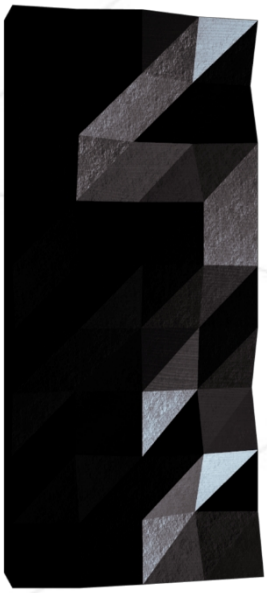

C++ - Module 06

Les casts en C++

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

Version : 6

Contenu

Chapitre I

Introduction

Le C++ est un langage de programmation généraliste créé par Bjarne Stroustrup en tant qu'ex tension du langage de programmation C, ou "C avec des classes" (source : Wikipedia).

L'objectif de ces modules est de vous initier à la **programmation orientée objet**. Ce sera le point de départ de votre voyage en C++. De nombreux langages sont recommandés pour apprendre la POO. Nous avons décidé de choisir le C++ car il est dérivé de votre vieil ami le C. Comme 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 à bien des égards. Si vous voulez devenir un développeur C++ compétent, il ne tient qu'à vous d'aller plus loin après le 42e tronc commun !

Chapitre II Règles générales

Compilation

- Compilez votre code avec c++ et les options -Wall -Wextra -Werror
- Votre code devrait toujours être compilé si vous ajoutez le drapeau -std=c++98

Conventions de formatage et de dénomination

- Les répertoires d'exercices seront nommés de la manière suivante : ex00, ex01, ... , exn
- Nommez vos fichiers, classes, fonctions, fonctions membres et attributs comme l'exigent les lignes directrices.
- Les noms des classes sont écrits en **majuscules**. Les fichiers contenant du code de classe seront toujours nommés en fonction du nom de la classe. Par exemple : Nomdeclasse.hpp/Nomdeclasse.h, Nomdeclasse.cpp, ou Nomdeclasse.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 un caractère de nouvelle ligne et affichés 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++ ! C'est pourquoi :

- 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 pas utiliser d'autres bibliothèques externes. Cela signifie que les bibliothèques C++11 (et 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 contraire, les espaces de noms d'utilisation <ns_name> et les mots-clés amis sont interdits. Dans le cas contraire, votre note sera de -42.
- **Vous n'êtes autorisé à utiliser le STL que dans les modules 08 et 09.** Cela signifie : pas de **conteneurs** (vector/list/map/etc.) et pas d'**algorithmes** (tout ce qui nécessite d'inclure l'en-tête <algorithm>) jusqu'à cette date. Sinon, votre note sera de -42.

Quelques exigences en matière de conception

- Les fuites de mémoire se produisent également en C++. Lorsque vous allouez de la mémoire (en utilisant la fonction new), vous devez éviter les **fuites de mémoire.**
- Du module 02 au module 09, vos cours doivent être conçus selon la **forme canonique orthodoxe, sauf indication contraire explicite.**
- 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. Ils doivent donc inclure toutes les dépendances dont ils ont besoin. Cependant, vous devez éviter le problème de la double inclusion en ajoutant des **gardes d'inclusion**. Dans le cas contraire, votre note sera de 0.

Lisez-moi

- Vous pouvez ajouter des fichiers supplémentaires si nécessaire (par exemple, pour diviser votre code). Comme ces travaux ne sont pas vérifiés par un programme, vous êtes libre de le faire tant que vous rendez les fichiers obligatoires.
- Parfois, les lignes directrices d'un exercice semblent courtes, mais les exemples peuvent montrer des exigences qui ne sont pas explicitement écrites dans les instructions.
- Lisez entièrement chaque module avant de commencer ! Vraiment, faites-le.
- Par Odin, par Thor ! Utilisez votre cerveau ! !!

Vous devrez implémenter un grand nombre de classes. Cela peut sembler fastidieux, à moins que vous ne soyez capable d'utiliser votre éditeur de texte favori.

Vous disposez d'une certaine liberté pour réaliser les exercices. Toutefois, respectez les règles obligatoires et ne soyez pas paresseux. Vous passeriez à côté d'un grand nombre d'informations utiles ! N'hésitez pas à vous documenter sur les concepts théoriques.

Chapitre III

Règle complémentaire

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 à l'aide d'un type de moulage spécifique.

Votre choix sera vérifié lors de la soutenance.

Chapitre IV

Exercice 00 : Conversion des types scalaires

	Exercice 00
Conversion des types scalaires	
Annuaire de retournement : *ex00/*	
Fichiers à rendre : `Makefile, *.cpp, *.{h, hpp}`	
Fonctions autorisées : `Toute fonction permettant de convertir une chaîne de caractères en un int, un float ou un double. Cela aidera, mais ne fera pas tout le travail.`	

Écrire une classe statique ScalarConverter qui contiendra une méthode "convert" prenant en paramètre une représentation sous forme de chaîne d'un littéral C++ dans sa forme la plus courante. Ce littéral doit appartenir à l'un des types scalaires suivants :

- char
- int
- flotteur
- double

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

Exemples de caractères littéraux : '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, un message d'information est affiché.

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

Exemples de littéraux float : 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 à son type réel, puis le convertir **explicitement** dans les trois autres types de données. Enfin, affichez les résultats comme indiqué ci-dessous.

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

```
./convert 0
char : Non affichable
int : 0
float : 0.0f
double :
0.0
./convert nan
char :
impossible int :
impossible
float : nanf
double : nan
./convert 42.0f
char : '*'
int : 42
float : 42.0f
double :
42.0
```

Chapitre V

Exercice 01 : sérialisation

	Exercice : 01
Sérialisation	
Répertoire de retournement : *ex01/*	
Fichiers à rendre : `Makefile, *.cpp, *.{h, hpp}`	

Fonctions interdites : `Aucune`

Implémentez une classe statique Serializer avec les méthodes suivantes :

uintptr_t serialize(Data* ptr) ;
Elle prend un pointeur et le convertit en un entier non signé de type uintptr_t.

Data* deserialize(uintptr_t raw) ;
Elle prend un paramètre entier non signé et le convertit en un pointeur sur Data.

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

Vous devez créer une structure de données non vide (ce qui signifie qu'elle possède des membres).

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

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

Chapitre VI

Exercice 02 : Identifier le type de réel

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

Implémentez une classe **Base** qui ne possède qu'un destructeur virtuel public. Créez trois classes vides **A**, **B** et **C**, qui héritent publiquement de la classe Base.

Ces quatre classes ne doivent pas nécessairement être conçues selon la forme canonique orthodoxe.

Mettre en œuvre les fonctions suivantes :

Base * generate(void) ;
Il instancie aléatoirement A, B ou C et renvoie l'instance sous la forme d'un 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 identify(Base& p) ;
Elle imprime 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.

Il est interdit d'inclure l'en-tête typeinfo.

Ecrivez un programme pour tester que tout fonctionne comme prévu.

Chapitre VII

Soumission et évaluation par les pairs

Rendez votre travail dans votre dépôt Git comme d'habitude. Seul le travail contenu 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