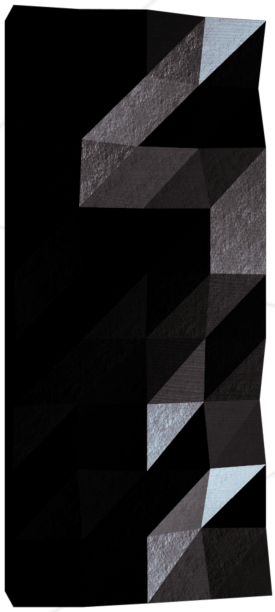

C++ - Module 06

C++ Casts

Résumé: Ce document contient le sujet du module 06 des modules C++ de 42.

Table des matières

Chapitre I

Règles Générales

- Toute fonction déclarée dans une header (sans pour les templates) ou tout header non-protégé, signifie 0 à l'exercice.
- Tout output doit être affiché sur stdout et terminé par une newline, sauf autre chose est précisé.
- Les noms de fichiers imposés doivent être suivis à la lettre, tout comme les noms de classe, les noms de fonction, et les noms de méthodes.
- Rappel : vous codez maintenant en `C++`, et plus en `C`. C'est pourquoi :
  - Les fonctions suivantes sont **INTERDITES**, et leur usage se soldera par un 0 : `*alloc`, `*printf` et `free`
  - Vous avez l'autorisation d'utiliser à peu près toute la librairie standard. CEPENDANT, il serait intelligent d'essayer d'utiliser la version C++ de ce à quoi vous êtes habitués en C, plutôt que de vous reposer sur vos acquis. Et vous n'êtes pas autorisés à utiliser la STL jusqu'au moment où vous commencez à travailler dessus (module 08). Ca signifie pas de Vector/List/Map/etc... ou quoi que ce soit qui requiert une include <algorithm> jusque là.
- L'utilisation d'une fonction ou mécanique explicitement interdite sera sanctionnée par un `0`
- Notez également que sauf si la consigne l'autorise, les mot-clés `using namespace` et `friend` sont interdits. Leur utilisation sera punie d'un `0`.
- Les fichiers associés à une classe seront toujours nommés `ClassName.cpp` et `ClassName.hpp`, sauf si la consigne demande autre chose.
- Vous devez lire les exemples minutieusement. Ils peuvent contenir des prérequis qui ne sont pas précisés dans les consignes.
- Vous n'êtes pas autorisés à utiliser des librairies externes, incluant `C++11`, `Boost`, et tous les autres outils que votre ami super fort vous a recommandé
- Vous allez surement devoir rendre beaucoup de fichiers de classe, ce qui peut paraitre répétitif jusqu'à ce que vous appreniez a scripter ca dans votre éditeur de code préferé.

- Lisez complètement chaque exercice avant de le commencer.
- Le compilateur est `clang++`
- Votre code sera compilé avec les flags `-Wall -Wextra -Werror`
- Chaque include doit pouvoir être incluse indépendamment des autres includes. Un include doit donc inclure toutes ses dépendances.
- Il n'y a pas de norme à respecter en `C++`. Vous pouvez utiliser le style que vous préferez. Cependant, un code illisible est un code que l'on ne peut pas noter.
- Important : vous ne serez pas noté par un programme (sauf si précisé dans le sujet). Cela signifie que vous avez un degré de liberté dans votre méthode de résolution des exercices.
- Faites attention aux contraintes, et ne soyez pas fainéant, vous pourriez manquer beaucoup de ce que les exercices ont à offrir
- Ce n'est pas un problème si vous avez des fichiers additionnels. Vous pouvez choisir de séparer votre code dans plus de fichiers que ce qui est demandé, tant qu'il n'y a pas de moulinette.
- Même si un sujet est court, cela vaut la peine de passer un peu de temps dessus afin d'être sûr que vous comprenez bien ce qui est attendu de vous, et que vous l'avez bien fait de la meilleure manière possible.

Chapitre II

Règles bonus

- Pour chaque exercice, la situation doit être résolue par un cast spécifique. L'évaluation vérifiera que votre choix de cast est correct.

Chapitre III

Exercice 00 : Conversion scalaire

	Exercice : 00
Exercice 00 : Conversion scalaire	
Dossier de rendu : $ex00/$	
Fichiers à rendre : `Tous les fichiers dont vous avez besoin + Makefile`	
Fonctions Autorisées : `Une fonction qui vous permet de convertir un string en int, float ou double. Ca vous aidera sans faire tout le travail.`	

Ecrivez un programme qui prend en paramètre une chaîne de caractères réprésentant une valeur littérale de `C ++` (sous sa forme la plus courante). Cette valeur doit appartenir à l'un des types scalaires suivants : `char`, `int`, `float` ou `double`. Seule la notation décimale sera utilisée.

Exemples de valeurs littérales de `char` : `'c'`, `'a'`...
Pour simplifier, veuillez noter que : les caractères non affichables ne peuvent pas être passés en paramètre à votre programme et si une conversion en `char` ne peut pas être affichée, renvoyez un message d'erreur.

Exemples de valeurs littérales `int` : `0`, `-42`, `42`...

Exemples de valeurs littérales `float` : `0.0f`, `-4.2f`, `4.2f`... Vous accepterez également ces pseudo-littéraux, vous savez, pour la science : `-inff`, `+ inff` et `nanf`.

Exemples de valeurs littérales `double` : `0.0`, `-4.2`, `4.2`... Vous accepterez également ces pseudo-littéraux, vous savez, pour le plaisir ... : `-inf`, `+ inf` et `nan`.

Votre programme doit détecter le type du littéral, acquérir ce littéral dans le bon type (pour que ce ne soit plus une chaîne), puis le convertir **explicitement** en chacun des trois autres types et afficher les résultats en utilisant le même formatage que ci-dessous. Si une conversion n'a pas de sens ou qu'elle déborde, indiquez que la conversion est impossible. Vous pouvez inclure n'importe quel header pour gérer les limites et les valeurs spéciales.

Examples :

```
./convert 0
char: Non displayable
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 IV

Exercice 01 : Serialisation

	Exercice : 01
Exercice 01 : Serialisation	
Dossier de rendu : $ex01/$	
Fichiers à rendre : `Tous les fichiers dont vous avez besoin + Makefile`	
Fonctions Autorisées : `Aucune`	

Écrivez une fonction `"uintptr_t serialize(Data* ptr);"`. Cette fonction retournera le paramètre sous la forme d'un nombre entier.

Écrivez une fonction `"Data* deserialize(uintptr_t raw);"`. Cette fonction renvoie les données brutes que vous avez créées à l'aide de la fonction `"serialize"` à une structure `Data`.

Écrivez un programme avec ces deux fonctions pour prouver que tout fonctionne comme prévu.
Vous devez créer une structure de données valide.
Prenez une adresse "Data" et utilisez serialize sur celle-ci.
Envoyez la valeur de retour dans deserialize.
Vérifiez si la valeur de retour est égale au premier pointeur.
N'oubliez pas d'inclure la structure `Data` que vous avez utilisée.

Chapitre V

Exercice 02 : Identifiez le véritable type

	Exercice : 02
Exercice 02 : Identifiez le véritable type	
Dossier de rendu : *ex*02/	
Fichiers à rendre : `Tous les fichiers dont vous avez besoin + Makefile`	
Fonctions Autorisées : `Aucune`	

Créez une classe `Base` qui possède uniquement un destructeur public virtuel. Créez trois classes vides `A`, `B` et `C`, qui héritent publiquement de `Base`.

Ecrivez une fonction `"Base * generate(void);"` qui instancie de facon aléatoire `A`, `B` ou `C` et retourne un pointeur de l'instance `Base`. Vous pouvez utiliser n'importe quelle fonction pour la generation aléatoire.

Écrivez une fonction `"void identify_from_pointer(Base * p);"` qui affiche `"A"`, `"B"` ou `"C"` selon le type réel de `p`.

Écrivez une fonction `"void identify_from_reference( Base & p);"` qui affiche `"A"`, `"B"` ou `"C"` selon le type réel de `p`.

Ajoutez de quoi faire un programme, prouvant que tout fonctionne. L'inclusion de `<typeinfo>` est formellement interdite.