



C++ - Module 06

C++ casts

Résumé:

Ce document contient les exercices du Module 06 des C++ modules.

Version: 5

Table des matières

I	Introduction	2
II	Consignes générales	3
III	Consigne supplémentaire	5
IV	Exercice 00 : Conversion scalaire	6
V	Exercice 01 : Sérialisation	8
VI	Exercice 02 : Identifiez le véritable type	9

Chapitre I

Introduction

C++ is a general-purpose programming language created by Bjarne Stroustrup as an extension of the C programming language, or "C with Classes" (source : [Wikipedia](#)).

C++ est un langage de programmation compilé permettant la programmation sous de multiples paradigmes, dont la programmation procédurale, la programmation orientée objet et la programmation générique. Ses bonnes performances, et sa compatibilité avec le C en font un des langages de programmation les plus utilisés dans les applications où la performance est critique (source : [Wikipedia](#)).

Ces modules ont pour but de vous introduire à la **Programmation Orientée Objet**. Plusieurs langages sont recommandés pour l'apprentissage de l'OOP. Du fait qu'il soit dérivé de votre bon vieil ami le C, nous avons choisi le langage C++. Toutefois, étant un langage complexe et afin de ne pas vous compliquer la tâche, vous vous conformerez au standard C++98.

Nous avons conscience que le C++ moderne est différent sur bien des aspects. Si vous souhaitez pousser votre maîtrise du C++, c'est à vous de creuser après le tronc commun de 42!

Chapitre II

Consignes générales

Compilation

- Compilez votre code avec `c++` et les flags `-Wall -Wextra -Werror`
- Votre code doit compiler si vous ajoutez le flag `-std=c++98`

Format et conventions de nommage

- Les dossiers des exercices seront nommés ainsi : `ex00`, `ex01`, ... , `exn`
- Nommez vos fichiers, vos classes, vos fonctions, vos fonctions membres et vos attributs comme spécifié dans les consignes.
- Rédigez vos noms de classe au format **UpperCamelCase**. Les fichiers contenant le code d'une classe porteront le nom de cette dernière. Par exemple : `NomDeClasse.hpp/NomDeClasse.h`, `NomDeClasse.cpp`, ou `NomDeClasse.hpp`. Ainsi, si un fichier d'en-tête contient la définition d'une classe "BrickWall", son nom sera `BrickWall.hpp`.
- Sauf si spécifié autrement, tous les messages doivent être terminés par un retour à la ligne et être affichés sur la sortie standard.
- *Ciao Norminette !* Aucune norme n'est imposée durant les modules C++. Vous pouvez suivre le style de votre choix. Mais ayez à l'esprit qu'un code que vos pairs ne peuvent comprendre est un code que vos pairs ne peuvent évaluer. Faites donc de votre mieux pour produire un code propre et lisible.

Ce qui est autorisé et ce qui ne l'est pas

Le langage C, c'est fini pour l'instant. Voici l'heure de se mettre au C++ ! Par conséquent :

- Vous pouvez avoir recours à quasi l'ensemble de la bibliothèque standard. Donc plutôt que de rester en terrain connu, essayez d'utiliser le plus possible les versions C++ des fonctions C dont vous avez l'habitude.
- Cependant, vous ne pouvez avoir recours à aucune autre bibliothèque externe. Ce qui signifie que C++11 (et dérivés) et l'ensemble **Boost** sont interdits. Aussi, certaines fonctions demeurent interdites. Utiliser les fonctions suivantes résultera

en la note de 0 : `*printf()`, `*alloc()` et `free()`.

- Sauf si explicitement indiqué autrement, les mots-clés `using namespace <ns_name>` et `friend` sont interdits. Leur usage résultera en la note de -42.
- **Vous n'avez le droit à la STL que dans les Modules 08 et 09.** D'ici là, l'usage des **Containers** (`vector`/`list`/`map`/etc.) et des **Algorithmes** (tout ce qui requiert d'inclure `<algorithm>`) est interdit. Dans le cas contraire, vous obtiendrez la note de -42.

Quelques obligations côté conception

- Les fuites de mémoires existent aussi en C++. Quand vous allouez de la mémoire (en utilisant le mot-clé `new`), vous ne **devez pas avoir de memory leaks**.
- Du Module 02 au Module 09, vos classes devront se conformer à la forme **canonique, dite de Coplien, sauf si explicitement spécifié autrement**.
- Une fonction implémentée dans un fichier d'en-tête (hormis dans le cas de fonction template) équivaldra à la note de 0.
- Vous devez pouvoir utiliser vos fichiers d'en-tête séparément les uns des autres. C'est pourquoi ils devront inclure toutes les dépendances qui leur seront nécessaires. Cependant, vous devez éviter le problème de la double inclusion en les protégeant avec des **include guards**. Dans le cas contraire, votre note sera de 0.

Read me

- Si vous en avez le besoin, vous pouvez rendre des fichiers supplémentaires (par exemple pour séparer votre code en plus de fichiers). Vu que votre travail ne sera pas évalué par un programme, faites ce qui vous semble le mieux du moment que vous rendez les fichiers obligatoires.
- Les consignes d'un exercice peuvent avoir l'air simple mais les exemples contiennent parfois des indications supplémentaires qui ne sont pas explicitement demandées.
- Lisez entièrement chaque module avant de commencer ! Vraiment.
- Par Odin, par Thor ! Utilisez votre cervelle!!!



Vous aurez à implémenter un bon nombre de classes, ce qui pourrait s'avérer ardu... ou pas ! Il y a peut-être moyen de vous simplifier la vie grâce à votre éditeur de texte préféré.



Vous êtes assez libre quant à la manière de résoudre les exercices. Toutefois, respectez les consignes et ne vous en tenez pas au strict minimum, vous pourriez passer à côté de notions intéressantes. N'hésitez pas à lire un peu de théorie.

Chapitre III


Consigne supplémentaire

La règle suivante s'applique à la totalité du module et n'est pas optionnelle.

Pour chaque exercice, la conversion de type devra être résolue en utilisant un type de cast spécifique. Votre choix sera évalué en soutenance.

Chapitre IV

Exercice 00 : Conversion scalaire

	Exercice : 00
Conversion scalaire	
Dossier de rendu : <i>ex00/</i>	
Fichiers à rendre : <i>Makefile, *.cpp, *.{h, hpp}</i>	
Fonctions Autorisées : Une fonction qui vous permet de convertir un string en int, float ou double, et qui vous aidera sans faire tout le travail.	

Écrivez un programme prenant en paramètre la représentation sous forme de chaîne de caractères d'un littéral C++ sous sa forme la plus répandue. Le littéral doit faire partie d'un des types scalaires suivants : char, int, float ou double. Hormis dans le cas de paramètre de type char, seule la notation décimale sera utilisée.

Exemples de littéraux de type char : `'c'`, `'a'`, ...

Pour simplifier les choses, notez que les caractères non affichables ne seront pas passés à votre programme. Si la conversion d'un char n'est affichable, mettez un message informatif.

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

Exemples de littéraux de type float : `0.0f`, `-4.2f`, `4.2f`...

Vous devez gérer ces pseudo littéraux aussi (pour l'amour de la science) : `-inff`, `+inff` et `nanf`.

Exemples de littéraux de type double : `0.0`, `-4.2`, `4.2`...

Vous devez gérer ces pseudo littéraux aussi (parce que c'est fun) : `-inf`, `+inf` et `nan`.


Vous devez premièrement détecter le type du littéral passé en paramètre, le convertir de sa représentation sous forme de chaîne de caractères vers son véritable type, et ensuite le convertir **explicitement** vers les trois autres types de données.

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

```
./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 V

Exercice 01 : Sérialisation

	Exercice : 01
Sérialisation	
Dossier de rendu : <i>ex01/</i>	
Fichiers à rendre : Makefile , *.cpp , *.{h, hpp}	
Fonctions interdites : Aucune	

Implémentez les fonctions suivantes :

```
uintptr_t serialize(Data* ptr);
```

Elle prend un pointeur et convertit celui-ci vers le type d'entier non-signé `uintptr_t`.

```
Data* deserialize(uintptr_t raw);
```

Elle prend un entier non-signé en paramètre et le convertit en pointeur sur `Data`.

Écrivez un programme pour tester vos fonctions et vous assurer que tout marche comme attendu.


Vous devez créer une structure non-vide (cela signifie qu'elle comporte des variables membres) `Data`.

Utilisez `serialize()` sur l'adresse de l'objet `Data` et passez la valeur de retour obtenue à `deserialize()`. Ensuite, assurez-vous que le retour de `deserialize()` est identique au pointeur d'origine.

N'oubliez pas de rendre les fichiers de votre structure `Data`.

Chapitre VI

Exercice 02 : Identifiez le véritable type

	Exercice : 02
Identifiez le véritable type	
Dossier de rendu : <i>ex02/</i>	
Fichiers à rendre : <code>Makefile</code> , <code>*.cpp</code> , <code>*.{h, hpp}</code>	
Fonctions interdites : <code>std::typeinfo</code>	

Implémentez une classe **Base** comportant seulement un destructeur public virtuel. Créez trois classes vides **A**, **B** et **C**, héritant publiquement de **Base**.



Ces quatre classes n'ont pas à se conformer à la forme canonique de Coplien.

Implémentez les fonctions suivantes :

```
Base * generate(void);
```

Elle crée aléatoirement une instance de A, B ou C et la retourne en tant que pointeur sur Base. Utilisez ce que vous souhaitez pour l'implémentation du choix aléatoire.

```
void identify(Base* p);
```

Elle affiche le véritable type de l'objet pointé par p : "A", "B" ou "C".

```
void identify(Base& p);
```

Elle affiche le véritable type de l'objet pointé par p : "A", "B" ou "C". Utiliser un pointeur dans cette fonction est interdit.

Le fichier d'en-tête `typeinfo` est interdit.

Écrivez un programme pour tester que tout fonctionne comme attendu.