



# C++ - Module 06

## Castings C++

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

Version : 7.2

# Contenu

|     |   |    |
|-----|---|----|
| je  | Introduction                                | 2  |
| II  | Règles générales                            | 3  |
| III | Règle supplémentaire                        | 6  |
| IV  | Exercice 00 : Conversion de types scalaires | 7  |
| V   | Exercice 01 : Sérialisation                 | 10 |
| VI  | Exercice 02 : Identifier le type réel       | 11 |
| VII | Soumission et évaluation par les pairs      | 12 |

# Chapitre I

## Introduction

C++ est un langage de programmation à usage général créé par Bjarne Stroustrup comme une extension du langage de programmation C, souvent appelé « 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 apprentissage du C++. De nombreux langages sont recommandés pour l'apprentissage de la POO, mais nous avons choisi le C++ car il est dérivé de votre ancien ami, le C. Le C++ étant un langage complexe, votre code respectera la norme C++98 pour plus de simplicité.

Nous reconnaissons que le C++ moderne diffère considérablement sur de nombreux aspects. Si vous souhaitez devenir un développeur C++ compétent, il vous appartiendra d'explorer plus avant les 42 éléments du 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 de cette façon : `ex00`, `ex01`, ... , `exn`
- Nommez vos fichiers, classes, fonctions, fonctions membres et attributs comme requis dans les lignes directrices.
- Écrivez les noms de classe en majuscules (UpperCamelCase) . Les fichiers contenant du code de classe seront toujours nommés selon le 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, chaque message de sortie doit se terminer par un caractère de nouvelle ligne et être affiché sur la sortie standard.
- Adieu Norminette ! Aucun style de codage n'est imposé dans les modules C++. Vous pouvez suivre votre style préféré. Mais gardez à l'esprit que le code que vos pairs ne comprennent pas est du code qu'ils ne peuvent pas noter. Faites de votre mieux pour écrire un code clair et lisible.

#### Autorisé/Interdit

Vous ne codez plus en C. Passez au C++ ! Par conséquent :

- Vous pouvez utiliser presque tout ce qui est proposé dans la bibliothèque standard. Ainsi, plutôt que 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 C++11 (et ses dérivés) et les bibliothèques Boost sont interdits. Les fonctions suivantes sont également interdites : `*printf()`, `*alloc()` et `free()`. Si vous les utilisez, votre note sera de 0, point final.

- Notez que, sauf indication contraire explicite, 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.) ni d'algorithmes (tout ce qui nécessite l'inclusion de l'en-tête <algorithm>) jusqu'à cette date. 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 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 nécessaires. 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 nécessaire (par exemple, pour fractionner votre code). Ces devoirs n'étant pas vérifiés par un programme, n'hésitez pas à le faire, à condition de fournir les fichiers obligatoires.
- Parfois, les lignes directrices d'un exercice semblent courtes, mais les exemples peuvent montrer exigences qui ne sont pas explicitement écrites dans les instructions.
- Lisez chaque module en entier avant de commencer ! Vraiment, forcez.
- Par Odin, par Thor ! Utilisez votre cerveau !!!



Concernant le Makefile pour les projets C++, les mêmes règles qu'en C s'appliquent (voir le chapitre Norme sur le Makefile).



Vous devrez implémenter de nombreuses classes. Cela peut paraître fastidieux, à moins que vous ne puissiez écrire des scripts 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

manquez beaucoup d'informations utiles ! N'hésitez pas à lire

concepts théoriques.

## Chapitre III


### Règle supplémentaire

La règle suivante s'applique à l'ensemble du module et est obligatoire.

Pour chaque exercice, la conversion de type doit être gérée à l'aide d'un type de casting spécifique.  
Votre choix sera revu lors de la soutenance.

# Chapitre IV

## Exercice 00 : Conversion de scalaires types

|   |             |
|---|-------------|
|    | Exercice 00 |
| Conversion de types scalaires   |             |
| Répertoire de remise : ex00/  |             |
| Fichiers à remettre : Makefile, *.cpp, *.h, *.hpp   |             |
| Fonctions autorisées : toute fonction permettant de convertir une chaîne en entier, en flottant ou en double. Cela peut être utile, mais ne suffit pas. |             |

Écrivez une classe `ScalarConverter` qui ne contiendra qu'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 :

- charbon
- int
- flotter
- double

Comme cette classe n'a pas besoin de stocker quoi que ce soit, elle ne doit pas être instanciable par les 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 caractères n'est pas affichable, affichez 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 dans les trois autres types de données. Enfin, affichez les résultats comme indiqué ci-dessous.

Si une conversion est incompréhensible ou déborde, affichez un message informant 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 : S rialisation

|   |               |
|---|---------------|
|  | Exercice : 01 |
| S rialisation   |               |
| R pertoire de remise :  |               |
| ex01/ Fichiers   remettre : Makefile, *.cpp, *.{h, hpp}                           |               |
| Fonctions interdites : Aucune   |               |

Impl menter une classe Serializer, qui ne sera en aucun cas initialisable par l'utilisateur, avec les m thodes statiques suivantes :

```
uintptr_t s rialiser(Donn es* ptr);
```

Il prend un pointeur et le convertit en type entier non sign  uintptr\_t.

```
Donn es* d s rialiser(uintptr_t raw);
```

Il prend un param tre entier non sign  et le convertit en un pointeur vers Data.

 crivez un programme pour tester que votre classe fonctionne comme pr vu.


Vous devez cr er une structure de donn es non vide (ce qui signifie qu'elle contient des membres de donn es).

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

N'oubliez pas de rendre les fichiers de votre structure de donn es.

# Chapitre VI

## Exercice 02 : Identifier le type réel

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

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



Ces quatre classes n'ont pas besoin d'être conçues dans l'Église orthodoxe.  
Forme canonique.

Implémentez les fonctions suivantes :

Base \* generate(void); Il instancie aléatoirement A, B ou C et renvoie l'instance sous forme de pointeur Base. N'hésitez pas à utiliser ce que vous souhaitez 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); Cette fonction affiche le type réel de l'objet référencé par p : « A », « B » ou « C ». L'utilisation d'un pointeur dans 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

Soumettez 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