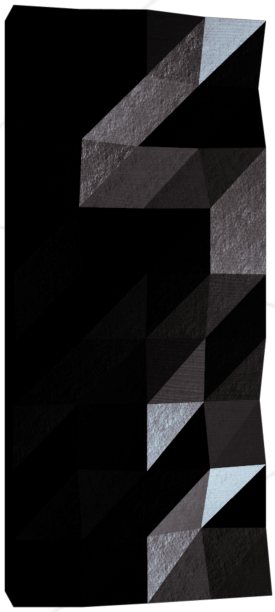

C++ - Module 02

Polymorphisme ad-hoc, overloads et classes canoniques

Résumé: Ce document contient le module 02 des modules de 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

Bonus rules

- A partir de maintenant, chaque classe que vous écrivez DOIT être canonique Au moins un constructeur par défaut, un constructeur par copie, un overload d'opérateur d'assignation et un destucteur.

Chapitre III

Exercice 00 : Mon premier canon

	Exercice : 00
Exercice 00 : Mon premier canon	
Dossier de rendu : *ex*00/	
Fichiers à rendre : `Fixed.class.hpp and Fixed.class.cpp, or Fixed.hpp and Fixed.cpp, or Fixed.h and Fixed.cc. Pick one, I don't care unless it's stupid.`	
Fonctions interdites : `Aucune`	

Vous connaissez les int et les floats.
Vous êtes mignons.

Veuillez lire cet article de 3 pages (en anglais, ndlr) :
(1,
2,
3)
pour découvrir que ca n'est pas le cas. Allez-y.

Jusqu'à aujourd'hui, tous les nombres que vous avez utilisés dans vos programmes étaient essentiellement des nombres entiers ou à virgule, ou l'une de leurs variantes (`short`, `char`, `long`, `double`, etc.). D'après vos lectures précédentes, il est prudent de supposer que les nombres entiers et les nombres à virgule flottante ont des caractéristiques opposées.

Mais aujourd'hui, cela va changer. Vous allez découvrir un nouveau type de nombre génial : les nombres à point fixe ! Toujours absents de la plupart des langages typés, les nombres à point fixe offrent un équilibre précieux entre performances, précision, portée et précision, ce qui explique pourquoi ces nombres sont largement utilisés dans les programmes graphiques, sonores ou scientifiques, pour n'en nommer que quelques-uns.

Vu que le `C++` n'a pas de nombre à point fixe, vous allez les ajouter aujourd'hui. Je recommanderai cet article de `Berkeley` pour bien démarrer. Si ca leur convient, ca nous convient. Si vous ne savez pas ce qu'est `Berkeley`, lisez cette section de leur page wikipedia.

Écrivez une classe canonique qui représente les nombres à point fixe :

- Membre privés :
  - Un int pour stocker la valeur a point fixe
  - Une `constante statique` Pour stocker le nombre de bits. Cette constante vaudra toujours `8`.
- Membres publics :
  - Un constructeur par défaut qui initialise la valeur a point fixe à `0`
  - Un destructeur.
  - Un constructeur par copie.
  - Un overload d'opérateur d'assignation.
  - Une fonction membre `int getRawBits(void) const;` qui renvoie la valeur brute du nombre à point fixe.
  - Une fonction membre `void setRawBits(int const raw);` qui set la valeur du nombre à point fixe.

Le code :

```
#include <iostream>

int		main(void) {

	Fixed a;
	Fixed b(a);
	Fixed c;

	c = b;

	std::cout << a.getRawBits() << std::endl;
	std::cout << b.getRawBits() << std::endl;
	std::cout << c.getRawBits() << std::endl;

	return 0;
}
```

Devrait renvoyer quelque chose comme :

```
> clang++ -Wall -Wextra -Werror Fixed.class.cpp main.cpp
> ./a.out
Default constructor called
Copy constructor called
Assignation operator called // <-- Cette ligne est peut-etre absente
getRawBits member function called
Default constructor called
Assignation operator called
getRawBits member function called
getRawBits member function called
0
getRawBits member function called
0
getRawBits member function called
0
Destructor called
Destructor called
Destructor called
```

Chapitre IV

Exercice 01 : Vers une classe plus utile

	Exercice : 01
Exercice 01 : Vers une classe plus utile	
Dossier de rendu : *ex*01/	
Fichiers à rendre : `Fixed.class.hpp and Fixed.class.cpp, or Fixed.hpp and Fixed.cpp, or Fixed.h and Fixed.cc.`	
Fonctions Autorisées : `roundf (from <cmath>)`	

Bon, `ex00` était un bon départ, mais notre classe ne sert littéralement à rien pour l'instant, car elle ne peut représenter que la valeur `0.0`. Ajoutez donc les constructeurs et fonctions membres publiques suivantes à votre classe :

- Un constructeur qui prend un const int en paramètre et qui le converti à sa valeur fixe(8) correspondante. La partie fractionnelle doit être initialisée comme dans l'ex00
- Un constructeur qui prend un const float en paramètre et et le convertit à sa valeur fixe(8) correspondante. La partie fractionnelle doit être initialisée comme dans l'ex00
- Une fonction membre `float toFloat(void) const;` that converts the fixed point value to a floating point value.
- Une fonction membre `int toInt(void) const;` that converts the fixed point value to an integer value.

Vous ajouterez aussi l'overload suivant dans votre header et votre fichier source :

- Overload de « qui insère une représentation de votre nombre à point fixe dans l'output demandé.

Vous pouvez utiliser le code suivant :

```cpp
#include <iostream>

int     main(void) {

    Fixed       a;
    Fixed const b(10);
    Fixed const c(42.42f);
    Fixed const d(b);

    a = Fixed(1234.4321f);

    std::cout << "a is " << a << std::endl;
    std::cout << "b is " << b << std::endl;
    std::cout << "c is " << c << std::endl;
    std::cout << "d is " << d << std::endl;

    std::cout << "a is " << a.toInt() << " as integer" << std::endl;
    std::cout << "b is " << b.toInt() << " as integer" << std::endl;
    std::cout << "c is " << c.toInt() << " as integer" << std::endl;
    std::cout << "d is " << d.toInt() << " as integer" << std::endl;

    return 0;
}
```

Chapitre V

Exercice 02 : Now we're talking

	Exercice : 02
Exercice 02 : Now we're talking	
Dossier de rendu : $ex02/$	
Fichiers à rendre : `Fixed.class.hpp and Fixed.class.cpp, or Fixed.hpp and Fixed.cpp, or Fixed.h and Fixed.cc.`	
Fonctions Autorisées : `roundf (from <cmath>)`	

On se rapproche. Ajoutez les fonctions publiques et overloads suivants à votre classe :

- Six opérateurs de comparaison : `>`, `<`, `>=`, `<=`, `==` and `!=`.
- Quatre opérateurs arithmétiques : `+`, `-`, `*`, et `/`.
- Les opérateurs de pre-incrément, post-incrément, pré-décrément et post-décrément, qui vont incrémenter et décrémenter la valeur du nombre à point fixe de la valeur représentable $\epsilon la plus petite telle que 1 + \epsilon > 1$.

Ajoutez les overloads de fonctions publiques non membres à votre classe :

- La fonction non-membre `min` qui prend une référence sur deux nombres a point fixe et qui renvoie une réference vers le plus petit, et un overload qui prend une reférence sur deux nombres à point fixe et qui renvoie une réference vers la plus petite valeur.
- La fonction non-membre `max` qui prend une référence sur deux nombres a point fixe et qui renvoie une réference vers le plus grand, et un overload qui prend une reférence sur deux nombres à point fixe et qui renvoie une réference vers la plus grande valeur.

C'est à vous de tester chaque feature de votre classe, mais ce court bout de code :

```cpp
#include <iostream>

int     main(void) {

    Fixed           a;
    Fixed const     b(Fixed(5.05f) * Fixed(2));

    std::cout << a << std::endl;
    std::cout << ++a << std::endl;
    std::cout << a << std::endl;
    std::cout << a++ << std::endl;
    std::cout << a << std::endl;

    std::cout << b << std::endl;

    std::cout << Fixed::max(a, b) << std::endl;

    return 0;
}
```

Doit ouput quelque chose de la sorte (nous avons supprimé les ctors/dtors logs) :

```
> clang++ -Wall -Wextra -Werror Fixed.class.cpp main.cpp
> ./a.out
0
0.00390625
0.00390625
0.00390625
0.0078125
10.1016
10.1016
```

Chapitre VI

Exercice 03 : Expressions

	Exercice : 03
Exercice 03 : Expressions	
Dossier de rendu : $ex03/$	
Fichiers à rendre : `Fixed.class.hpp and Fixed.class.cpp, et tout fichier nécessaire, et un Makefile. A vous de faire une convention de nommage.`	
Fonctions Autorisées : `roundf (from <cmath>)`	

`Cet exercice ne rapporte pas de points, mais demeure interessant. Vous n'êtes pas obligés de le faire.`

Écrivez un programme nommé `eval_expr` qui évalue des expressions arithmétiques simples sur des nombres à point fixe.

`Veillez à utiliser des strings, istreams, ostreams, istringstreams et ostringstreams autant que possible. Cela vous sauvera beaucoup de temps.`

Par exemple :

```
$> clang++ -Wall -Wextra -Werror -o eval_expr Fixed.class.cpp {your files}
$> ./eval_expr "(18.18 + 3.03) * 2"
42.4219
$>
```