C++ - Module 02

Polymorphisme ad-hoc, surcharge d'opérateurs et forme canonique

Résumé:
Ce document contient les exercices du Module 02 des C++ modules.

Version: 8

Table des matières

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.tpp`. 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 avec 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) équivaudra 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

Nouvelle consigne

À partir de maintenant, vos classes devront impérativement se conformer à la **forme canonique de Coplien**, sauf en cas d'indication contraire. Cela signifie qu'elles devront comporter les quatre fonctions membres suivantes :

- Constructeur par défaut
- Constructeur de recopie
- Opérateur d'affectation
- Destructeur

Séparer le code de vos classes dans deux fichiers. Le fichier d'en-tête (.hpp/.h) contient la définition de la classe, tandis que le fichier source (.cpp) contient son implémentation.

Chapitre IV

Exercice 00 : Mon premier canon

	Exercice : 00
Mon premier canon	
Dossier de rendu : *ex*00/	
Fichiers à rendre : `Makefile, main.cpp, Fixed.{h, hpp}, Fixed.cpp`	
Fonctions interdites : `Aucune`	

Si vous pensiez connaître les entiers et les flottants, cet article de 3 pages (1, 2, 3) vous montrera que ce n'est sans doute pas le cas. Allez, lisez-le.

Jusqu'à aujourd'hui, tous les nombres utilisés dans votre code étaient soit des entiers, soit des flottants, soit éventuellement leurs dérivés (`short`, `char`, `long`, `double`, etc.). Après avoir lu l'article ci-dessus, il paraît évident d'affirmer que les caractéristiques des entiers et des nombres à virgule flottante sont opposées.

Mais aujourd'hui, les choses vont changer. Vous allez découvrir une notion inédite et passionnante : la **représentation des nombres en virgule fixe** ! À jamais absents des types scalaires de la plupart des langages, les nombres à virgule fixe offrent un équilibre intéressant entre performance, exactitude, portée et précision. Cela explique pourquoi ces nombres sont largement utilisés dans l'imagerie numérique, le domaine du son ou la programmation scientifique, pour n'en citer que trois.

Étant donné que le C++ ne possède pas de nombres à virgule fixe, vous allez les ajouter. Cet article de Berkeley est un bon point de départ. Si vous ne savez pas ce qu'est l'Université de Berkeley, lisez cette partie de sa page Wikipédia.

Créez une classe sous forme canonique pour représenter un nombre à virgule fixe.

- Membres privés :
  - Un **entier** pour stocker la valeur du nombre en virgule fixe.
  - Un **entier constant statique** pour stocker le nombre de bits de la partie fractionnaire, et dont la valeur sera toujours le littéral entier `8`.
- Membres publics :
  - Un constructeur par défaut qui initialisera la valeur du nombre à virgule fixe à `0`.
  - Un constructeur de recopie.
  - Une surcharge de l'opérateur d'affectation.
  - Un destructeur.
  - Une fonction membre `int getRawBits( void ) const;` qui retourne la valeur du nombre à virgule fixe sans la convertir.
  - Une fonction membre `void setRawBits( int const raw );` qui initialise la valeur du nombre à virgule fixe avec celle passée en paramètre.

Exécuter ce 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 afficher ce résultat :

```
$> ./a.out
Default constructor called
Copy constructor called
Copy assignment operator called // <-- This line may be missing depending on your implementation
getRawBits member function called
Default constructor called
Copy assignment 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 V

Exercice 01 : Premiers pas vers une classe utile

	Exercice : 01
Premiers pas vers une classe utile	
Dossier de rendu : $ex01/$	
Fichiers à rendre : `Makefile, main.cpp, Fixed.{h, hpp}, Fixed.cpp`	
Fonctions Autorisées : `roundf (from <cmath>)`	

L'exercice précédent était un bon point de départ mais notre classe n'a pour l'instant pas beaucoup d'intérêt. Elle peut seulement représenter la valeur `0.0`.

Ajoutez à votre classe les constructeurs et les fonctions membres suivants en public :

- Un constructeur prenant un **entier constant** en paramètre et qui convertit celui-ci en virgule fixe. Le nombre de bits de la partie fractionnaire est initialisé à 8 comme dans l'exercice 00.
- Un constructeur prenant un **flottant constant** en paramètre et qui convertit celui-ci en virgule fixe. Le nombre de bits de la partie fractionnaire est initialisé à 8 comme dans l'exercice 00.
- Une fonction membre `float toFloat( void ) const;`
  qui convertit la valeur en virgule fixe en nombre à virgule flottante.
- Une fonction membre `int toInt( void ) const;`
  qui convertit la valeur en virgule fixe en nombre entier.

Ajoutez également la fonction suivante à vos fichiers de la classe **Fixed** :

- Une surcharge de l'opérateur d'insertion («) qui insère une représentation en virgule flottante du nombre à virgule fixe dans le flux de sortie (objet output stream) passé en paramètre.

Exécuter ce code :

```
#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;
}
```

Devrait afficher ce résultat :

```
$> ./a.out
Default constructor called
Int constructor called
Float constructor called
Copy constructor called
Copy assignment operator called
Float constructor called
Copy assignment operator called
Destructor called
a is 1234.43
b is 10
c is 42.4219
d is 10
a is 1234 as integer
b is 10 as integer
c is 42 as integer
d is 10 as integer
Destructor called
Destructor called
Destructor called
Destructor called
$>
```

Chapitre VI

Exercice 02 : Maintenant, on peut parler

	Exercice : 02
Maintenant, on peut parler	
Dossier de rendu : $ex02/$	
Fichiers à rendre : `Makefile, main.cpp, Fixed.{h, hpp}, Fixed.cpp`	
Fonctions Autorisées : `roundf (from <cmath>)`	

Ajoutez à votre classe des fonctions membres publiques afin de surcharger les opérateurs suivants :

- Les 6 opérateur de comparaison : `>`, `<`, `>=`, `<=`, `==` et `!=`.
- Les 4 opérateurs de arithmétiques : `+`, `-`, `*`, et `/`.
- Les 4 opérateurs d'incrémentation et de décrémentation (pré-incrémentation et post-incrémentation, pré-décrémentation et post-décrémentation) qui diminueront la valeur du nombre à virgule fixe d'unité ϵ tel que $1 + \epsilon > 1$.

Ajoutez à votre classe ces quatre fonctions membres publiques surchargées :

- Une fonction membre statique `min` prenant en paramètres deux références sur des nombres à virgule fixe et qui retourne le plus petit d'entre eux.
- Une fonction membre statique `min` prenant en paramètres deux références sur des nombres à virgule fixe **constants** et qui retourne le plus petit d'entre eux.
- Une fonction membre statique `max` prenant en paramètres deux références sur des nombres à virgule fixe et qui retourne le plus grand d'entre eux.
- Une fonction membre statique `max` prenant en paramètres deux références sur des nombres à virgule fixe **constants** et qui retourne le plus grand d'entre eux.

C'est à vous de tester chaque fonctionnalité de votre code. Mais exécuter ce code :

```
#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;
}
```

Devra afficher ce résultat (pour plus de lisibilité, les messages du constructeur et du destructeur ont été retirés) :

```
$> ./a.out
0
0.00390625
0.00390625
0.00390625
0.0078125
10.1016
10.1016
$>
```

Si vous effectuez une division par 0, il est acceptable que le programme crash.

Chapitre VII

Exercice 03 : BSP

	Exercice : 03
BSP	
Dossier de rendu : $ex03/$	
Fichiers à rendre : `Makefile, main.cpp, Fixed.{h, hpp}, Fixed.cpp, Point.{h, hpp}, Point.cpp, bsp.cpp`	
Fonctions Autorisées : `roundf (from <cmath>)`	

Maintenant que vous avez une classe **Fixed** fonctionnelle, ce serait sympa de l'utiliser non ?

Implémentez une fonction qui indique si un point donné est à l'intérieur d'un triangle. Super utile, n'est-ce pas ?

`BSP signifie Binary Space Partitioning. Ne me remerciez pas. :)`

`Vous pouvez valider ce module sans l'exercice 03.`

Premièrement, créez une classe **Point** sous forme canonique pour représenter un point 2D :

- Membres privés :
  - Un attribut Fixed constant `x`.
  - Un attribut Fixed constant `y`.
  - Et tout ce qui peut vous être utile.
- Membres publics :
  - Un constructeur par défaut qui initialise `x` et `y` à `0`.
  - Un constructeur prenant deux flottants constants en paramètres et initialisant `x` et `y` avec ces derniers.
  - Un constructeur de recopie.
  - Une surcharge de l'opérateur d'affectation.
  - Un destructeur.
  - Et tout ce qui peut vous être utile.

Pour conclure, implémentez la fonction suivante dans le fichier correspondant :

```
bool bsp( Point const a, Point const b, Point const c, Point const point);
```

- a, b, c : Les sommets de notre cher triangle.
- point : Le point à évaluer.
- Retourne : True si le point est à l'intérieur du triangle. False dans le cas contraire. Cela veut dire que, si le point est un sommet ou placé sur une arrête, la fonction retournera False.

Écrivez et rendez vos propres tests afin de démontrer que votre classe fonctionne comme demandé.

Chapitre VIII

Submission and peer-evaluation

Rendez votre travail dans votre dépôt `Git` comme d'habitude. Seul le travail présent sur votre dépôt sera évalué en soutenance. Vérifiez bien les noms de vos dossiers et de vos fichiers afin que ces derniers soient conformes aux demandes du sujet.

```
??????????? XXXXXXXXXX = $3$$d6f957a965f8361750a3ba6c97554e9f
```