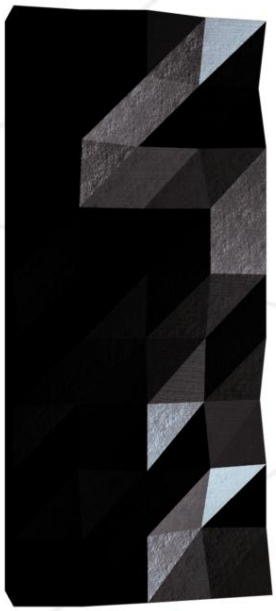

C++ - Module 02

Polymorphisme ad hoc, surcharge d'opérateurs et forme de classe canonique orthodoxe

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

Version : 8.2

Contenu

Chapitre I

Introduction

C++ est un langage de programmation à usage général créé par Bjarne Stroustrup comme une extension du langage de programmation C, ou « 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 du C. Comme il s'agit d'un langage complexe, et afin de simplifier les choses, votre code sera conforme à la norme C++98.

Nous sommes conscients que le C++ moderne est très différent à bien des égards. Si vous souhaitez devenir un développeur C++ compétent, il vous appartient de poursuivre vos études après le Common Core 42 !

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, foncez.
- 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
Vous manquez beaucoup d'informations utiles ! N'hésitez pas à lire
concepts théoriques.

Chapitre III

Nouvelles règles

Désormais, toutes vos classes doivent être conçues selon la forme canonique orthodoxe, sauf indication contraire explicite. Elles implémenteront alors les quatre fonctions membres requises ci-dessous :

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

Divisez votre code de classe en deux fichiers. Le fichier d'en-tête (.hpp/.h) contient la classe. définition, tandis que le fichier source (.cpp) contient l'implémentation.

Chapitre IV

Exercice 00 : Mon premier cours en Forme canonique orthodoxe

	Exercice : 00
Mon premier cours sur la forme canonique orthodoxe	
Répertoire de remise : ex00/	
Fichiers à rendre : Makefile, main.cpp, Fixed.{h, hpp}, Fixed.cpp Fonctions interdites : Aucune	

Tu crois connaître les nombres entiers et les nombres à virgule flottante. C'est mignon !

Veuillez lire cet article de 3 pages (1, 2, 3) pour découvrir que ce n'est pas le cas. Allez-y, lisez-le.

Jusqu'à aujourd'hui, chaque nombre que vous utilisiez dans votre code était essentiellement soit un entier, soit un nombre à virgule flottante, ou l'une de leurs variantes (court, char, long, double, etc.).
Après avoir lu l'article ci-dessus, on peut supposer en toute sécurité que les entiers et les nombres à virgule flottante ont des caractéristiques opposées.

Mais aujourd'hui, les choses vont changer. Vous allez découvrir un nouveau type de nombres impressionnant : les nombres à virgule fixe ! Toujours absents des types scalaires de la plupart des langages, les nombres à virgule fixe offrent un équilibre précieux entre performance, exactitude, portée et précision. C'est pourquoi ils sont particulièrement adaptés à l'infographie, au traitement du son ou à la programmation scientifique, pour n'en citer que quelques-uns.

Comme le C++ ne dispose pas de nombres à virgule fixe, vous allez les ajouter. Cet article Berkeley est un bon début. Si vous ne savez pas ce qu'est l'Université de Berkeley, lisez cette section. de sa page Wikipédia.

Créez une classe sous forme canonique orthodoxe qui représente un nombre à virgule fixe :

• Membres privés :

- Un entier pour stocker la valeur du nombre à virgule fixe.
- Un entier constant statique pour stocker le nombre de bits fractionnaires. Sa valeur sera toujours le littéral entier 8.

• Membres du public :

- Un constructeur par défaut qui initialise la valeur du nombre à virgule fixe à 0.
- Un constructeur de copie.
- Une surcharge d'opérateur d'affectation de copie.
- Un destructeur.
- Une fonction membre int getRawBits(void) const; qui renvoie la valeur brute de la valeur à virgule fixe.
- Une fonction membre void setRawBits(int const raw); qui définit la valeur brute du nombre à virgule fixe.

Exécution de ce code :

```
#include <iostream>

int                     principal( vide ) {

    Correction d'un;
    Fixe b( a );
    Fixe c;

    c = b;

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


    retourner 0;
}
```

Devrait afficher quelque chose de similaire à :

```
> ./a.out
Constructeur par défaut appelé
Constructeur de copie appelé
Opérateur d'affectation de copie appelé // <-- Cette ligne peut être manquante en fonction de votre implémentation getRawBits fonction membre appelée

Constructeur par défaut appelé
Opérateur d'affectation de copie appelé fonction membre
getRawBits appelée fonction membre getRawBits appelée
0 fonction membre getRawBits appelée 0



fonction membre getRawBits appelée 0

Destructeur appelé
Destructeur appelé
Destructeur appelé
$>
```

Chapitre V

Exercice 01 : Vers une classe de nombres à virgule fixe plus utile

	Exercice 01
Vers une classe de nombres à virgule fixe plus utile	
Répertoire de remise : ex01/	
Fichiers à rendre : Makefile, main.cpp, Fixed.{h, hpp}, Fixed.cpp Fonctions	
autorisées : roundf (de <cmath>)	

L'exercice précédent était un bon début, mais notre classe est plutôt inutile. Elle ne peut représenter que la valeur 0,0.

Ajoutez les constructeurs publics et les fonctions membres publiques suivants à votre classe :

- Un constructeur qui prend un entier constant comme paramètre.
  Il le convertit en valeur à virgule fixe correspondante. La valeur des bits fractionnaires doit être initialisée à 8, comme dans l'exercice 00.
- Un constructeur qui prend un nombre à virgule flottante constant comme paramètre.
  Il le convertit en valeur à virgule fixe correspondante. La valeur des bits fractionnaires doit être initialisée à 8, comme dans l'exercice 00.
- Une fonction membre float toFloat(void) const;
  qui convertit la valeur à virgule fixe en une valeur à virgule flottante.
- Une fonction membre int toInt(void) const;
  qui convertit la valeur à virgule fixe en une valeur entière.

Et ajoutez la fonction suivante aux fichiers de classe Fixed :

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

Exécution de ce code :

```
#include <iostream>

int principal( vide ) {

    Fixé un;
    Constante fixe b( 10 );
    Constante fixe c( 42.42f );
    Constante fixe d( b );

    a = Fixe( 1234.4321f );

    std::cout << "a est " << un << std::endl;
    << "b est " << b << std::endl;
    std::cout << "c est " << c << std::endl;
    << "d est " << d << std::endl;

    std::cout << "a est " << a.toInt() << " comme entier" << std::endl;
    std::cout << "b est " << b.toInt() << " comme entier" << std::endl;
    << "c est " << c.toInt() << " comme entier" << std::endl;
    << "d est " << d.toInt() << " comme entier" << std::endl;

    retourner 0;
}
```

Devrait afficher quelque chose de similaire à :

```
> ./a.out
Constructeur par défaut appelé
Constructeur Int appelé
Constructeur Float appelé
Constructeur de copie appelé
Opérateur d'affectation de copie appelé
Constructeur Float appelé
Opérateur d'affectation de copie appelé
Destructeur appelé
a est 1234,43
b est 10
c est 42,4219
d est 10
a est 1234 en tant qu'entier
b est 10 en tant qu'entier
c est 42 en tant qu'entier
d est 10 en tant qu'entier
Destructeur appelé
Destructeur appelé
Destructeur appelé
Destructeur appelé
$>
```

Chapitre VI

Exercice 02 : Maintenant, nous parlons

	Exercice 02
Maintenant nous parlons	
Répertoire de remise : ex02/	
Fichiers à rendre : Makefile, main.cpp, Fixed.{h, hpp}, Fixed.cpp Fonctions autorisées : roundf (de <cmath>)	

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

- Les 6 opérateurs de comparaison : >, <, >=, <=, == et !=.
- Les 4 opérateurs arithmétiques : +, -, * et /.
- Les 4 opérateurs d'incrémentation/décrémentation (pré-incrémentation et post-incrémentation, pré-décrémentation et post-décrémentation), qui augmenteront ou diminueront la valeur à virgule fixe du plus petit représentable, tel que 1 + > 1.

Ajoutez ces quatre fonctions membres publiques surchargées à votre classe :

- Une fonction membre statique min qui prend deux références à des nombres à virgule fixe comme paramètres et renvoie une référence au plus petit.
- Une fonction membre statique min qui prend deux références à une virgule fixe constante des nombres comme paramètres et renvoie une référence au plus petit.
- Une fonction membre statique max qui prend deux références à des nombres à virgule fixe comme paramètres et renvoie une référence au plus grand.
- Une fonction membre statique max qui prend deux références à une virgule fixe constante nombres comme paramètres et renvoie une référence au plus grand.

C'est à vous de tester chaque fonctionnalité de votre classe. Cependant, exécutez le code ci-dessous :

```
#include <iostream>

int principal( vide ) {

    Fixé a;
    constante fixe b( Fixe( 5.05f ) * Fixe( 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;

    retourner 0;
}
```

Devrait afficher quelque chose comme (pour une meilleure lisibilité, le message constructeur/destructeur-les sages sont supprimés dans l'exemple ci-dessous) :

```
$> ./a.out 0

0,00390625
0,00390625
0,00390625
0,0078125
10,1016
10.1016 $>
```

Si jamais vous faites une division par 0, il est acceptable que le programme accidents

Chapitre VII

Exercice 03 : BSP

	Exercice 03
BSP	
Répertoire de remise : ex03/	
Fichiers à rendre : Makefile, main.cpp, Fixed.{h, hpp}, Fixed.cpp, Point.{h, hpp}, Point.cpp, bsp.cpp Fonctions autorisées : roundf	
(de <cmath>)	

Maintenant que vous avez une classe Fixed fonctionnelle , il serait intéressant de l'utiliser.

Implémenter une fonction indiquant si un point est à l'intérieur d'un triangle ou non. Très utile, n'est-ce pas ?

BSP signifie Partitionnement de l'Espace Binaire. De rien. :)

Vous pouvez réussir ce module sans avoir complété l'exercice 03.

Commençons par créer la classe Point sous forme canonique orthodoxe qui représente un point 2D :

• Membres privés :

- Un attribut constant fixe x.
- Un attribut constant fixe y.
- Toute autre chose utile.

• Membres du public :

- Un constructeur par défaut qui initialise x et y à 0.
- Un constructeur qui prend deux nombres à virgule flottante constants comme paramètres. Il initialise x et y avec ces paramètres.
- Un constructeur de copie.
- Une surcharge d'opérateur d'affectation de copie.
- Un destructeur.
- Toute autre chose utile.

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

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

• a, b, c : Les sommets de notre triangle bien-aimé.

• point : Le point à vérifier.

• Renvoie : Vrai si le point est à l'intérieur du triangle. Faux sinon.
Ainsi, si le point est un sommet ou sur une arête, il renverra False.

Implémentez et remettez vos propres tests pour vous assurer que votre classe se comporte comme prévu.

Chapitre VIII

Soumission et évaluation par les pairs

Remettez 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.

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