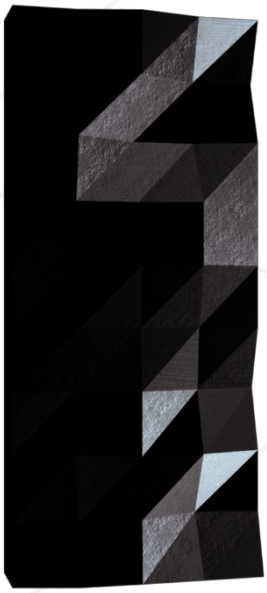

C++ - Module 05

Répétition et exceptions

Résumé :
Ce document contient les exercices du module 05 des modules C++.

Version : 10

Contenu

Chapitre I

Introduction

Le C++ est un langage de programmation généraliste créé par Bjarne Stroustrup en tant qu'ex tension du langage de programmation C, ou "C avec des classes" (source : Wikipedia).

L'objectif de ces modules est de vous initier à la **programmation orientée objet**. Ce sera le point de départ de votre voyage en C++. De nombreux langages sont recommandés pour apprendre la POO. Nous avons décidé de choisir le C++ car il est dérivé de votre vieil ami le C. Comme il s'agit d'un langage complexe, et afin de garder les choses simples, 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 voulez devenir un développeur C++ compétent, il ne tient qu'à vous d'aller plus loin après le 42e tronc commun !

Chapitre II Règles générales

Compilation

- Compilez votre code avec c++ et les options -Wall -Wextra -Werror
- Votre code devrait toujours être compilé si vous ajoutez le drapeau -std=c++98

Conventions de formatage et de dénomination

- Les répertoires d'exercices seront nommés de la manière suivante : ex00, ex01, ... , exn
- Nommez vos fichiers, classes, fonctions, fonctions membres et attributs comme l'exigent les lignes directrices.
- Les noms des classes sont écrits en **majuscules**. Les fichiers contenant du code de classe seront toujours nommés en fonction du nom de la classe. Par exemple : Nomdeclasse.hpp/Nomdeclasse.h, Nomdeclasse.cpp, ou Nomdeclasse.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, tous les messages de sortie doivent être terminés par un caractère de nouvelle ligne et affichés sur la sortie standard.
- *Au revoir Norminette !* Aucun style de codage n'est imposé dans les modules C++. Vous pouvez suivre votre style préféré. Mais gardez à l'esprit qu'un code que vos pairs évaluateurs ne peuvent pas comprendre est un code qu'ils ne peuvent pas noter. Faites de votre mieux pour écrire un code propre et lisible.

Autorisé/interdit

Vous ne codez plus en C. Il est temps de passer au C++ ! C'est pourquoi :

- Vous êtes autorisé à utiliser presque tout ce qui se trouve dans la bibliothèque standard. Ainsi, au lieu 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 pas utiliser d'autres bibliothèques externes. Cela signifie que les bibliothèques C++11 (et formes dérivées) et Boost sont interdites. Les fonctions suivantes sont également interdites : *printf(), *alloc() et free(). Si vous les utilisez, votre note sera de 0 et c'est tout.

- Notez que, sauf indication contraire, les espaces de noms d'utilisation <ns_name> et les mots-clés amis sont interdits. Dans le cas contraire, votre note sera de -42.
- **Vous n'êtes autorisé à utiliser le STL que dans les modules 08 et 09.** Cela signifie : pas de **conteneurs** (vector/list/map/etc.) et pas d'**algorithmes** (tout ce qui nécessite d'inclure l'en-tête <algorithm>) jusqu'à cette date. Sinon, votre note sera de -42.

Quelques exigences en matière de conception

- Les fuites de mémoire se produisent également en C++. Lorsque vous allouez de la mémoire (en utilisant la fonction new
), vous devez éviter les **fuites de mémoire.**
- Du module 02 au module 09, vos cours doivent être conçus selon la **forme canonique orthodoxe, 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 dont ils ont besoin. Cependant, vous devez éviter le problème de la double inclusion en ajoutant des **gardes d'inclusion**. Dans le cas contraire, votre note sera de 0.

Lisez-moi

- Vous pouvez ajouter des fichiers supplémentaires si nécessaire (par exemple, pour diviser votre code). Comme ces travaux ne sont pas vérifiés par un programme, vous êtes libre de le faire tant que vous rendez les fichiers obligatoires.
- Parfois, les lignes directrices d'un exercice semblent courtes, mais les exemples peuvent montrer des exigences qui ne sont pas explicitement écrites dans les instructions.
- Lisez entièrement chaque module avant de commencer ! Vraiment, faites-le.
- Par Odin, par Thor ! Utilisez votre cerveau ! !!

Vous devrez implémenter un grand nombre de classes. Cela peut sembler fastidieux, à moins que vous ne soyez capable d'utiliser votre éditeur de texte favori.

Vous disposez d'une certaine liberté pour réaliser les exercices. Toutefois, respectez les règles obligatoires et ne soyez pas paresseux. Vous passeriez à côté d'un grand nombre d'informations utiles ! N'hésitez pas à vous documenter sur les concepts théoriques.

Chapitre III

Exercice 00 : Maman, quand je serai grand, je veux être un bureaucrate !

	Exercice : 00
Maman, quand je serai grand, je veux être un bureaucrate !	
Annuaire de retournement : *ex00/*	
Fichiers à rendre : `Makefile, main.cpp, Bureaucrat.{h, hpp}, Bureaucrat.cpp`	
Fonctions interdites : `Aucune`	

`Veuillez noter que les classes d'exception n'ont pas besoin d'être conçues dans la forme canonique orthodoxe. En revanche, toutes les autres classes doivent l'être.`

Concevons un cauchemar artificiel de bureaux, de couloirs, de formulaires et de files d'attente.
Ça a l'air sympa ? Non ? C'est dommage.

Commençons par le plus petit rouage de cette vaste machine bureaucratique : le

bureaucrate. Un **bureaucrate** doit avoir :

- Un nom constant.
- Et une note allant de **1** (note la plus élevée possible) à **150** (note la plus basse possible).

Toute tentative d'instanciation d'un Bureaucrat à l'aide d'un grade non valide doit générer une exception :
soit une exception Bureaucrat::GradeTooHighException, soit une exception Bureaucrat::GradeTooLowException.

Vous fournirez des getters pour ces deux attributs : getName() et getGrade(). Impliquez également deux fonctions membres pour incrémenter ou décrémenter le grade du bureaucrate. Si le grade est en dehors de la plage, les deux fonctions lèveront les mêmes exceptions que le constructeur.

Rappelez-vous. Le grade 1 étant le plus élevé et le grade 150 le plus bas, l'augmentation d'un grade 3 devrait donner un grade 2 au bureaucrate.

Les exceptions lancées doivent pouvoir être rattrapées à l'aide de blocs try et catch :

```
essayer
{
  /* faire des choses avec les bureaucrates */
}
catch (std::exception & e)
{
  /* gérer l'exception */
}
```

Vous allez implémenter une surcharge de l'opérateur d'insertion (") pour imprimer quelque chose comme (sans les crochets d'angle) :

<nom>, grade bureaucrate <grade>.

Comme d'habitude, effectuez quelques tests pour vérifier que tout fonctionne comme prévu.

Chapitre IV

Exercice 01 : En formation, les asticots !

	Exercice : 01
Formez les asticots !	
Répertoire de retournement : *ex01/*	
Fichiers à rendre : `Fichiers de l'exercice précédent + Form.{h, hpp}, Form.cpp`	
Fonctions interdites : `Aucune`	

Maintenant que vous avez des bureaucrates, donnons-leur quelque chose à faire. Quelle meilleure activité que celle qui consiste à remplir une pile de formulaires ?

Ensuite, créons une classe **Form.** Elle possède :

- Un nom constant.
- Un booléen indiquant s'il est signé (à la construction, il ne l'est pas).
- Un grade constant est requis pour le signer.
- Et un grade constant est nécessaire pour l'exécuter.

Tous ces attributs sont **privés** et non protégés.

Les notes du **formulaire** suivent les mêmes règles que celles qui s'appliquent au bureaucrate. Ainsi, les exceptions suivantes seront levées si la note d'un formulaire est hors limites : Form::GradeTooHighException et Form::GradeTooLowException.

Même chose que précédemment, écrire des getters pour tous les attributs et une surcharge de l'opérateur d'insertion (") qui imprime toutes les informations du formulaire.

Ajouter également une fonction membre beSigned() au formulaire qui prend un bureaucrate comme paramètre. Elle fait passer le formulaire à l'état signé si le grade du bureaucrate est suffisamment élevé (supérieur ou égal au grade requis). N'oubliez pas que le grade 1 est supérieur au grade 2.
Si la note est trop basse, une exception Form::GradeTooLowException est levée.

Enfin, ajoutez une fonction membre signForm() au Bureaucrate. Si le formulaire a été signé, il s'imprimera quelque chose comme :

<bureaucrat> a signé <form>.

Sinon, il imprimera quelque chose comme :

<bureaucrate> n'a pas pu signer <formulaire> pour <motif>.

Mettre en œuvre et rendre quelques tests pour s'assurer que tout fonctionne comme prévu.

Chapitre V

Exercice 02 : Non, vous avez besoin du formulaire 28B, pas du 28C...

	Exercice : 02
Non, vous avez besoin du formulaire 28B, pas du 28C...	
Annuaire de retournement : *ex02/*	
Fichiers à rendre : `Makefile, main.cpp, Bureaucrat.[{h, hpp},cpp], Bureaucrat.cpp + AForm.[{h, hpp},cpp], ShrubberyCreationForm.[{h, hpp},cpp], + RobotomyRequestForm.[{h, hpp},cpp], PresidentialPardonForm.[{h, hpp},cpp]`	
Fonctions interdites : `Aucune`	

Puisque vous avez maintenant des formulaires de base, il est temps d'en créer d'autres qui servent à quelque chose.

Dans tous les cas, la classe de base Form doit être une classe abstraite et doit donc être renommée AForm. N'oubliez pas que les attributs du formulaire doivent rester privés et qu'ils se trouvent dans la classe de base.

Ajoutez les classes concrètes suivantes :

- **Formulaire de création d'arbustes** : Notes requises : signe 145, exec 137
  Crée un fichier <target>_shrubbery dans le répertoire de travail, et écrit des arbres ASCII à l'intérieur de celui-ci.
- **RobotomyRequestForm** : Notes requises : sign 72, exec 45
  Fait des bruits de forage. Puis, informe que la <cible> a été robotisée avec succès dans 50% des cas. Sinon, informe que la robotomie a échoué.
- **PresidentialPardonForm** : Notes requises : sign 25, exec 5
  Informe que <target> a été gracié par Zaphod Beeblebrox.

Tous ne prennent qu'un seul paramètre dans leur constructeur : la cible du formulaire. Par exemple, "home" si vous voulez planter des arbustes chez vous.

Maintenant, ajoutez la fonction membre execute(Bureaucrat const & executor) const au formulaire de base et mettez en œuvre une fonction pour exécuter l'action du formulaire des classes concrètes. Vous devez vérifier que le formulaire est signé et que le grade du bureaucrate qui tente d'exécuter le formulaire est suffisamment élevé. Dans le cas contraire, lancez une excep- tion appropriée.

C'est à vous de décider si vous voulez vérifier les exigences dans chaque classe concrète ou dans la classe de base (puis appeler une autre fonction pour exécuter le formulaire). Cependant, l'une des deux méthodes est plus jolie que l'autre.

Enfin, ajoutez la fonction membre executeForm(Form const & form) au Bureaucrat. Elle doit tenter d'exécuter le formulaire. Si elle réussit, elle imprime quelque chose comme :

<bureaucrat> exécuté <form>

Si ce n'est pas le cas, un message d'erreur explicite est imprimé.

Mettre en œuvre et rendre des tests pour s'assurer que tout fonctionne comme prévu.

Chapitre VI

Exercice 03 : au moins, c'est mieux que de faire du café

	Exercice : 03
Au moins, c'est mieux que de faire du café	
Annuaire de retournement : *ex03/*	
Fichiers à rendre : `Fichiers des exercices précédents + Intern.{h, hpp}, Intern.cpp`	
Fonctions interdites : `Aucune`	

Parce que remplir des formulaires est déjà suffisamment ennuyeux, il serait cruel de demander à nos bureaucrates de le faire toute la journée. Heureusement, les stagiaires existent. Dans cet exercice, vous devez implémenter la classe **Intern**. Le stagiaire n'a pas de nom, pas de grade, pas de caractéristiques uniques. La seule chose qui intéresse les bureaucrates, c'est qu'ils fassent leur travail.

Cependant, le stagiaire dispose d'une capacité importante : la fonction makeForm(). Elle prend deux chaînes de caractères. La première est le nom d'un formulaire et la seconde est la cible du formulaire. Elle retourne un pointeur vers un **objet Form** (dont le nom est celui passé en paramètre) dont la cible sera initialisée au second paramètre.

Il s'imprimera quelque chose comme :

Le stagiaire crée <form>

Si le nom du formulaire passé en paramètre n'existe pas, un message d'erreur explicite est affiché.

Vous devez éviter les solutions illisibles et laides comme l'utilisation d'une forêt de if/elseif/else. Ce genre de choses ne sera pas accepté lors du processus d'évaluation. Vous n'êtes plus dans la Piscine. Comme d'habitude, vous devez tester que tout fonctionne comme prévu.

Par exemple, le code ci-dessous crée un **RobotomyRequestForm** ciblé sur "Bender" :

```
{
    Intern someRandomIntern ;
    Form*   rrf ;

    rrf = someRandomIntern.makeForm("robotomy request", "Bender") ;
}
```

Chapitre VII

Soumission et évaluation par les pairs

Rendez votre travail dans votre dépôt Git comme d'habitude. Seul le travail contenu 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.

```
16D85ACC441674FBA2DF65190663F9373230CEAB1E4A0818611C0E39F5B26E4D774F1
74620A16827E1B16612137E59ECD492E468A92DCB17BF16988114B98587594D12810
E67D173222A
```