



C++ - Module 05

Répétition et exceptions

Résumé :

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

Version : 10.2

Contenu

je	Introduction	2
II	Règles générales	3
III	Exercice 00 : Maman, quand je serai grande, je veux être bureaucrate !	
IV	Exercice 01 : Formez-vous, asticots !	8
V	Exercice 02 : Non, vous avez besoin du formulaire 28B, pas 28C...	10
VI	Exercice 03 : Au moins, c'est mieux que de faire du café	12
VII	Soumission et évaluation par les pairs	14

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 parcours C++. De nombreux langages sont recommandés pour apprendre la programmation orientée objet. Nous avons décidé de choisir C++ car il est dérivé de votre vieil ami C.

Parce qu'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 sur de nombreux aspects. Donc si vous voulez devenir un développeur C++ compétent, c'est à vous d'aller plus loin après le 42 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 ainsi : `ex00`, `ex01`, ... , `exn`
- Nommez vos fichiers, classes, fonctions, fonctions membres et attributs comme requis dans les lignes directrices.
- Écrivez les noms de classe au format UpperCamelCase . Les fichiers contenant le code de classe doit toujours être nommé en fonction du nom de la classe. Par exemple : `ClassName.hpp/ClassName.h`, `ClassName.cpp` ou `ClassName.hpp`. 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 une nouvelle ligne caractère et affiché 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++ ! Par conséquent :

- 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 utiliser aucune autre bibliothèque externe. Cela signifie que les bibliothèques C++11 (et les 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 explicite contraire, 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.) et pas d'algorithmes (tout ce qui nécessite d'inclure l'en-tête `<algorithm>`) jusqu'à ce moment-là. 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 explicite contraire.
- 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. Ainsi, ils doivent inclure toutes les dépendances dont ils ont besoin. 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 vous en avez besoin (par exemple pour diviser votre code). Comme ces tâches ne sont pas vérifiées par un programme, n'hésitez pas à le faire à condition de rendre les fichiers obligatoires.
- Parfois, les directives d'un exercice semblent courtes, mais les exemples peuvent le montrer. exigences qui ne sont pas explicitement écrites dans les instructions.
- Lisez chaque module dans son intégralité avant de commencer ! Vraiment, faites-le.
- Par Odin, par Thor ! Utilise ton 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 soyez capable de créer un script 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
manque beaucoup d'informations utiles ! N'hésitez pas à lire à propos
concepts théoriques.

Chapitre III

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

	Exercice : 00
Maman, quand je serai grande, je veux être bureaucrate !	
Répertoire de remise : ex00/	
Fichiers à rendre : Makefile, main.cpp, Bureaucrat.{h, cpp}, Bureaucrat.cpp Fonctions interdites : Aucune	



Veuillez noter que les classes d'exception ne doivent pas nécessairement être conçues selon la forme canonique orthodoxe. Mais 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 amusant ? Non ? Dommage.

Commençons d'abord par le plus petit rouage de cette vaste machine bureaucratique : le bureaucrate.

Un bureaucrate doit avoir :

- Un nom constant.
- Et une note qui va de 1 (note la plus élevée possible) à 150 (note la plus basse possible) grade).

Toute tentative d'instancier un Bureaucrate en utilisant une note non valide doit générer une exception : soit une `Bureaucrat::GradeTooHighException`, soit une `Bureaucrat::GradeTooLowException`.

Vous fournirez des getters pour ces deux attributs : `getName()` et `getGrade()`. Implémentez également deux fonctions membres pour incrémenter ou décrémenter la note du bureaucrate. Si la note est hors de portée, les deux généreront les mêmes exceptions que le constructeur.



N'oubliez pas. Étant donné que la note 1 est la plus élevée et 150 la plus basse, une augmentation de la note 3 devrait donner la note 2 au bureaucrate.

Les exceptions levées doivent pouvoir être capturées à l'aide des blocs `try` et `catch` :

```
essayer
{
    /* faire des trucs avec les bureaucrates */
}
attraper (std::exception & e) {

    /* gérer l'exception */
}
```


Vous allez implémenter une surcharge de l'opérateur d'insertion (`<<`) pour imprimer quelque chose comme (sans les chevrons) :

`<nom>, grade de bureaucrate <grade>.`

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

Chapitre IV

Exercice 01 : Formez-vous, asticots !

	Exercice : 01
Formez-vous, asticots !	
Répertoire de remise : ex01/	
Fichiers à rendre : Fichiers de l'exercice précédent + Form.{h, cpp}, Form.cpp Fonctions interdites : Aucune	

Maintenant que vous avez des bureaucrates, donnons-leur quelque chose à faire. Quelle meilleure activité pourrait-il y avoir autre chose que de remplir une pile de formulaires ?

Ensuite, créons une classe Form . Elle contient :

- Un nom constant.
- Un booléen indiquant s'il est signé (à la construction, il ne l'est pas).
- Une note constante est requise pour le signer.
- Et une note constante est requise pour l'exécuter.

Tous ces attributs sont privés et non protégés.

Les grades de la classe de bureaucrate suivent les mêmes règles que celles qui s'appliquent à la classe de bureaucrate. Ainsi, les exceptions suivantes seront levées si une note de formulaire est hors limites :
Form::GradeTooHighException et Form::GradeTooLowException.

Comme précédemment, écrivez des getters pour tous les attributs et une surcharge de l'opérateur d'insertion (« ») qui imprime toutes les informations du formulaire.

Ajoutez également une fonction membre `beSigned()` au formulaire qui prend un `bureaucrate` comme paramètre. Elle modifie le statut du formulaire en signé si le grade du `bureaucrate` est suffisamment élevé (supérieur ou égal à celui requis). N'oubliez pas que le grade 1 est supérieur au grade 2. Si la note est trop basse, lancez une `Form::GradeTooLowException`.

Ensuite, modifiez la fonction membre `signForm()` dans la classe `Bureaucrat`. Cette fonction doit appeler `Form::beSigned()` pour tenter de signer le formulaire. Si le formulaire est signé avec succès, il affichera quelque chose comme : `<bureaucrat> signed`
`<form>`


Sinon, il imprimera quelque chose comme :

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

Implémentez et réalisez quelques tests pour vous 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...	
Répertoire de remise : ex02/	
Fichiers à remettre : Makefile, main.cpp, Bureaucrat.{h, hpp}, Bureaucrat.cpp + AForm.{h, hpp}, ShrubberyCreationForm.{h, hpp}, + RobotomyRequestForm.{h, hpp}, PresidentialPardonForm.{h, hpp}	
Fonctions interdites : Aucune	

Puisque vous disposez désormais de formulaires de base, il est temps d'en créer quelques autres qui font réellement quelque chose.

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

Ajoutez les classes concrètes suivantes :

- ShrubberyCreationForm : Niveaux requis : signe 145, exécutive 137
Créez un fichier <target>_shrubbery dans le répertoire de travail et écrivez des arbres ASCII à l'intérieur.
- RobotomyRequestForm : Notes requises : signe 72, exec 45 Émet des bruits de perçage. Informe ensuite que <target> a été robotisée avec succès dans 50 % des cas. Sinon, informe que la robotomy a échoué.
- Formulaire de grâce présidentielle : Grades requis : enseigne 25, exécutive 5
Informe que <cible> a été gracié par Zaphod Beeblebrox.

Tous ne prennent qu'un seul paramètre dans leur constructeur : la cible du formulaire. par exemple, « maison » si vous souhaitez planter des arbustes à la maison.

Ajoutez maintenant la fonction membre `execute(Bureaucrat const & executor) const` au formulaire de base et implémentez 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. Sinon, lancez une exception appropriée.

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

Enfin, ajoutez la fonction membre `executeForm(AForm const & form)` au Bureaucrat. Il doit tenter d'exécuter le formulaire. S'il réussit, affichez quelque chose comme :


`<bureaucrate> a exécuté <formulaire>`

Dans le cas contraire, imprimez un message d'erreur explicite.

Implémentez et réalisez quelques tests pour vous 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	
Répertoire de remise : ex03/	
Fichiers à rendre : Fichiers des exercices précédents + Intern.{h, hpp}, Intern.cpp Fonctions interdites : Aucune	

Parce que remplir des formulaires est déjà assez pénible, 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 importe aux bureaucrates, c'est qu'ils fassent leur travail.

Cependant, l'interne a une capacité importante : la fonction makeForm(). Elle prend deux chaînes. 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 imprimera quelque chose comme :

Le stagiaire crée <formulaire>

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

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

Par exemple, le code ci-dessous crée un RobotomyRequestForm ciblant « Ben-der » :

```
{  
    Stagiaire someRandomIntern;  
    Formulaire* rrf;  
  
    rrf = someRandomIntern.makeForm(" demande de robotique", "Bender");  
}
```

Chapitre VII

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.



16D85ACC441674FBA2DF65190663F9373230CEAB1E4A0818611C0E39F5B26E4D774F1
74620A16827E1B16612137E59ECD492E468A92DCB17BF16988114B98587594D12810
E67D173222A