mod_05

C++ - Module 05

Répétition et exceptions

Résumé:

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

Version : 10.3

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. Nous avons choisi le C++ car il est dérivé de votre ancien ami, le C. Étant donné sa complexité, et afin de préserver la simplicité, 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 directives.

• Écrivez les noms de classe en majuscules (majuscules). 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. Si vous disposez d'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.

C++ - Module 05 Répétition et exceptions

• Sauf mention contraire explicite, l'utilisation de l'espace de noms <ns_name> et des mots-clés friend est interdite. Dans le cas contraire, 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++. Lors de l'allocation de mémoire (avec le mot-clé new), il est essentiel d'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 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 directives d'un exercice semblent courtes, mais les exemples peuvent montrer des 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, respectez les règles obligatoires et ne soyez pas paresseux. Vous passeriez à côté de nombreuses informations utiles ! N’hésitez pas à vous renseigner sur les concepts théoriques.

Chapitre III

Exercice 00 : Maman, quand je serai grand, 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, hpp}, Bureaucrat.cpp

Fonctions interdites : Aucune

Veuillez noter que les classes d’exception n’ont pas besoin d’être conçues dans

Forme canonique orthodoxe. Cependant, toutes les autres classes doivent la suivre.

Créons 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.

• Une note allant de 1 (note la plus élevée possible) à 150 (note la plus basse possible).

Toute tentative d’instancier un Bureaucrat avec une note non valide doit générer une exception : soit une Bureaucrat::GradeTooHighException, soit une Bureaucrat::GradeTooLowException.

Vous fournirez des accesseurs pour les deux attributs : getName() et getGrade(). Vous devrez également implémenter deux fonctions membres pour incrémenter ou décrémenter la note du fonctionnaire. Si la note dépasse la limite, les deux fonctions doivent lever les mêmes exceptions que le constructeur.

N’oubliez pas que, puisque la note 1 est la plus élevée et 150 la plus basse, l’augmentation d’une note 3 devrait donner lieu à une note 2 pour le bureaucrate.

Les exceptions levées doivent pouvoir être interceptées à l’aide des blocs try et catch :

essayer {

/* faire des trucs avec les bureaucrates */

} catch (std::exception & e)

{

/* gérer l’exception */

}

Vous devez implémenter une surcharge de l’opérateur d’insertion («) pour imprimer la sortie au format suivant (sans les chevrons) :

<nom>, grade de bureaucrate <grade>.

Comme d’habitude, soumettez 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, hpp}, Form.cpp

Fonctions interdites : Aucune

Maintenant que vous avez des bureaucrates, donnons-leur quelque chose à faire. Quoi de mieux que de remplir une pile de formulaires ?

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.

• Une note constante est nécessaire pour l’exécuter.

Tous ces attributs sont privés et non protégés.

Les notes du formulaire suivent les mêmes règles que celles du Bureaucrate. Ainsi, les exceptions suivantes seront levées si la note d’un formulaire dépasse les limites :

Form::GradeTooHighException et Form::GradeTooLowException.

Comme précédemment, écrivez des getters pour tous les attributs et surchargez l'opérateur d'insertion («) pour imprimer toutes les informations du formulaire.

Ajoutez également une fonction membre beSigned() au formulaire qui prend un bureaucrate en paramètre. Elle modifie le statut du formulaire en « signé » si la note du bureaucrate est suffisamment élevée (supérieure ou égale à la note requise). N'oubliez pas que la note 1 est supérieure à la note 2. Si la note est trop basse, lancez une exception Form::GradeTooLowException.

Modifiez ensuite la fonction membre signForm() de la classe Bureaucrat. Cette fonction doit appeler Form::beSigned() pour tenter de signer le formulaire. Si le formulaire est signé avec succès, il affichera un message similaire à :

<bureaucrate> a signé <formulaire>

Sinon, il imprimera quelque chose comme :

<bureaucrate> n'a pas pu signer <formulaire> à cause de <raison>.

Implémentez et soumettez certains tests pour vous assurer que tout fonctionne comme prévu.

Chapitre V

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

Exercice : 02

Non, vous avez besoin du formulaire 28B, pas du 28C…

Répertoire de remise : ex02/

Fichiers à rendre : Makefile, main.cpp, Bureaucrat.[{h, hpp},cpp], +

AForm.[{h, hpp},cpp], ShrubberyCreationForm.[{h, hpp},cpp], +

Formulaire de demande de robotomy.[{h, hpp},cpp], Formulaire de pardon présidentiel.[{h, hpp},cpp]

Fonctions interdites : Aucune

Maintenant que vous disposez 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 appartiennent à la classe de base.

Ajoutez les classes concrètes suivantes :

• ShrubberyCreationForm : Niveaux requis : signe 145, exécutive 137

Crée un fichier <target>_shrubbery dans le répertoire de travail et écrit des arbres ASCII à l'intérieur.

• RobotomyRequestForm : Niveaux requis : signe 72, exécutive 45

Émet des bruits de forage, puis informe que la robotisation de <cible> a réussi dans 50 % des cas. Dans le cas contraire, il informe que la robotisation a échoué.

• Formulaire de grâce présidentielle : Niveaux requis : signature 25, exécutif 5

Informe que <cible> a été gracié par Zaphod Beeblebrox.

Tous ces types de formulaires n'acceptent qu'un seul paramètre dans leur constructeur : la cible du formulaire. Par exemple, « home » si vous souhaitez planter des arbustes chez vous.

Ajoutez maintenant la fonction membre execute(Bureaucrat const & executor) const au formulaire de base et implémentez une fonction pour exécuter l'action du formulaire dans les classes concrètes. Vérifiez que le

formulaire est signé et que le grade du bureaucrate qui tente de l’exécuter 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 élégante que l’autre.

Enfin, ajoutez la fonction membre executeForm(AForm const & form) const à la classe Bureaucrat. Elle doit tenter d’exécuter le formulaire. En cas de succès, affichez un message similaire à :

<bureaucrate> a exécuté <formulaire>

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

Implémentez et soumettez certains 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

Puisque remplir des formulaires toute la journée serait trop cruel pour nos bureaucrates, les stagiaires sont là pour se charger de cette tâche fastidieuse. Dans cet exercice, vous devez implémenter la classe Intern. Le stagiaire n’a ni nom, ni grade, ni caractéristiques uniques. La seule chose qui importe aux bureaucrates, c’est de faire leur travail.

Cependant, le stagiaire dispose d’une fonctionnalité clé : la fonction makeForm(). Cette fonction prend deux chaînes de caractères en paramètres : la première représente le nom d’un formulaire et la seconde sa cible. Elle renvoie un pointeur vers un objet AForm (correspondant au nom du formulaire passé en paramètre), dont la cible est initialisée au second paramètre.

Il devrait imprimer quelque chose comme :

Le stagiaire crée <formulaire>

Si le nom du formulaire fourni n’existe pas, imprimez un message d’erreur explicite.

Vous devez éviter les solutions illisibles et confuses, comme l’utilisation excessive de structures if/elseif/else. Ce type d’approche ne sera pas accepté lors du processus d’évaluation. Vous n’êtes plus dans la Piscine. Comme d’habitude, vous devez tout tester pour vous assurer que tout fonctionne comme prévu.

Par exemple, le code suivant crée un RobotomyRequestForm ciblant

“Cintreuse”:

{

Stagiaire someRandomIntern ; AFormulaire* rrf;

rrf = someRandomIntern.makeForm(“demande de robotique”, “Bender”); }

Chapitre VII

Soumission et évaluation par les pairs

Soumettez votre devoir dans votre dépôt Git comme d’habitude. Seul le travail effectué dans votre dépôt sera évalué lors de la soutenance. Vérifiez bien les noms de vos dossiers et fichiers.