



C++ - Module 05

Répétition et Exceptions

Résumé: Ce document contient le sujet pour le module 05 des modules C++ de 42.

Table des matières

I	Règles Générales	2
II	Exercice 00 : Maman, quand je serai grand, je veux être bureau- crate !	4
III	Exercice 01 : En rang, les larves !	6
IV	Exercice 02 : Vous avez besoin du Form 28B, pas 28C ...	7
V	Exercice 03 : Mieux que d'apporter le café	9
VI	Exercice 04 : C'est comme ça qu'on les aime, simple et ennuyant	10
VII	Exercice 05 : Générateur infini de signatures	13

Chapitre I


Règles Générales

- Toute fonction implémentée dans une header (sauf pour les templates) ou tout header non-protégé, signifie 0 à l'exercice.
- Tout output doit être affiché sur stdout et terminé par une newline, sauf si autre chose est précisé.
- Les noms de fichiers imposés doivent être suivis à la lettre, tout comme les noms de classe, les noms de fonction, et les noms de méthodes.
- Rappel : vous codez maintenant en C++, et plus en C. C'est pourquoi :
 - Les fonctions suivantes sont **INTERDITES**, et leur usage se soldera par un 0 : `*alloc`, `*printf` et `free`
 - Vous avez l'autorisation d'utiliser à peu près toute la librairie standard. CÉPENDANT, il serait intelligent d'essayer d'utiliser la version C++ de ce à quoi vous êtes habitués en C, plutôt que de vous reposer sur vos acquis. Et vous n'êtes pas autorisés à utiliser la STL jusqu'au moment où vous commencez à travailler dessus (module 08). Ça signifie pas de Vector/List/Map/etc... ou quoi que ce soit qui requiert une include `<algorithm>` jusque là.
- L'utilisation d'une fonction ou mécanique explicitement interdite sera sanctionnée par un 0
- Notez également que sauf si la consigne l'autorise, les mot-clés `using namespace` et `friend` sont interdits. Leur utilisation sera punie d'un 0.
- Les fichiers associés à une classe seront toujours nommés `ClassName.cpp` et `ClassName.hpp`, sauf si la consigne demande autre chose.
- Vous devez lire les exemples minutieusement. Ils peuvent contenir des prérequis qui ne sont pas précisés dans les consignes.
- Vous n'êtes pas autorisés à utiliser des librairies externes, incluant C++11, Boost, et tous les autres outils que votre ami super fort vous a recommandé.
- Vous allez surement devoir rendre beaucoup de fichiers de classe, ce qui peut paraître répétitif jusqu'à ce que vous appreniez à scripter ça dans votre éditeur de code préféré.

- Lisez complètement chaque exercice avant de le commencer.
- Le compilateur est `clang++`
- Votre code sera compilé avec les flags `-Wall -Wextra -Werror -std=c++98`
- Chaque include doit pouvoir être incluse indépendamment des autres includes. Un include doit donc inclure toutes ses dépendances.
- Il n'y a pas de norme à respecter en C++. Vous pouvez utiliser le style que vous préférez. Cependant, un code illisible est un code que l'on ne peut pas noter.
- Important : vous ne serez pas noté par un programme (sauf si précisé dans le sujet). Cela signifie que vous avez un degré de liberté dans votre méthode de résolution des exercices.
- Faites attention aux contraintes, et ne soyez pas fainéant, vous pourriez manquer beaucoup de ce que les exercices ont à offrir
- Ce n'est pas un problème si vous avez des fichiers additionnels. Vous pouvez choisir de séparer votre code dans plus de fichiers que ce qui est demandé, tant qu'il n'y a pas de moulinette.
- Même si un sujet est court, cela vaut la peine de passer un peu de temps dessus afin d'être sûr que vous comprenez bien ce qui est attendu de vous, et que vous l'avez bien fait de la meilleure manière possible.

Chapitre II

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

	Exercice : 00
Maman, quand je serai grand, je veux être bureaucrate !	
Dossier de rendu : <i>ex00/</i>	
Fichiers à rendre : <code>Makefile</code> <code>Bureaucrat.hpp</code> <code>Bureaucrat.cpp</code> <code>main.cpp</code>	
Fonctions interdites : Aucune	

Notez bien que les classes d'exceptions ne doivent pas être sous forme copienne. Les autres classes doivent l'être.

Aujourd'hui, nous allons créer un cauchemar artificiel de bureaux, couloirs, Forms, et lignes d'attente. Ca a l'air fun, hein ? Non ? Dommage.

Tout d'abord, nous commençons par le plus petit rouage de la vaste machine bureaucratique : le **Bureaucrat**.

Il doit avoir un nom constant et une note, qui va de 1 (la plus élevée possible) à 150 (la plus basse possible). Toute tentative de création d'un **Bureaucrat** avec une note invalide doit déclencher une exception, qui sera soit une **Bureaucrat::GradeTooHighException** ou une **Bureaucrat::GradeTooLowException**.

Vous créerez aussi des getters pour ces deux attributs (**getName** et **getGrade**), et deux fonctions pour incrémenter/décroître le grade. Attention : le grade 1 est le plus haut, donc le décroître vous donnera un grade 2, etc...

Les exceptions doivent être attrapables par un bloc de code du type : Les exceptions doivent fonctionner avec ces blocs :

```
try
{
    /* do some stuff with bureaucrats */
}
catch (std::exception & e)
{
    /* handle exception */
}
```


Vous devez également overloader l'opérateur « qui affichera quelque chose du genre : `<name>, bureaucrat grade <grade>`.

Vous allez fournir une surcharge de l'opérateur `<<` à `ostream` qui produit quelque chose comme : `<name>, bureaucrate grade <grade>`.

Bien sûr, vous fournirez un `main` pour prouver que tout fonctionne.

Chapitre III

Exercice 01 : En rang, les larves !

	Exercice : 01
En rang, les larves !	
Dossier de rendu : <i>ex01/</i>	
Fichiers à rendre : Pareil qu'avant + Form.cpp Form.hpp	
Fonctions interdites : Aucune	

Maintenant que nous avons des bureaucrates, il vaut mieux leur donner quelque chose à faire avec leur temps. Quoi de mieux qu'une pile de Forms à remplir ?

Créez une classe `Form`. Elle porte un nom, un booléen indiquant si elle est signée (au début, ce n'est pas le cas), un grade requis pour la signer et un grade requis pour l'exécuter. Le nom et le grade sont constants et tous ces attributs sont privés (et non protégés). Les grades sont sujets aux mêmes contraintes que dans `Bureaucrat`, et des exceptions seront levées si l'un d'eux est en dehors des limites, via `Form::GradeTooHighException` et `Form::GradeTooLowException`.

Comme auparavant, créez des getters pour tous les attributs et une surcharge de l'opérateur `<<` vers `ostream` qui décrit complètement l'état du `Form`.


Vous ajouterez également une fonction `beSigned` qui prend un `Bureaucrat`, et rend le formulaire signé si la note du bureaucrate est suffisamment élevée. Rappelez-vous toujours que la note 1 est meilleure que la note 2. Si la note est trop basse, elle déclenche une `Form::GradeTooLowException`.

Ajoutez également une fonction `signForm` à `Bureaucrat`. Si la signature est réussie, elle imprimera quelque chose comme "`<bureaucrat> signs <form>`", sinon elle imprimera quelque chose comme "`<bureaucrat> cannot sign because <raison>`".

Ajoutez tout ce qui est nécessaire pour tester ceci à votre `main`.

Chapitre IV

Exercice 02 : Vous avez besoin du Form 28B, pas 28C ...

	Exercice : 02
Vous avez besoin du Form 28B, pas 28C ...	
Dossier de rendu : <i>ex02/</i>	
Fichiers à rendre : Pareil qu'avant + <code>ShrubberyCreationForm.[hpp,cpp]</code> <code>RobotomyRequestForm.[hpp,cpp]</code> <code>PresidentialPardonForm.[hpp,cpp]</code>	
Fonctions interdites : Aucune	

Maintenant que nous avons des Forms basiques, nous allons faire quelques Forms qui font vraiment quelque chose.

Créez les quelques Forms concrètes suivantes :

- **ShrubberyCreationForm** (Grades requis : signature 145, execution 137). Action : Crée un fichier nommé `<target>_shrubbery`, et dessines des arbres en ASCII dedans, dans le dossier courant.
- **RobotomyRequestForm** (Grades requis : signature 72, execution 45). Action : Fait des bruits de perceuses, et annonce que `<target>` a bien été robotomisée dans 50% des cas, ou son echec le reste du temps.
- **PresidentialPardonForm** (Grades requis : signature 25, execution 5). Action : Nous annonce que `<target>` a été pardonnée par Zafod Beeblebrox.

Tous ces éléments devront prendre un seul paramètre dans leur constructeur, qui représentera la cible du Form. Par exemple, "maison" si vous souhaitez planter un arbuste à la maison. N'oubliez pas que les attributs du Form doivent rester privés et dans la classe de base.

Maintenant, vous devez ajouter une méthode `execute` (`Bureaucrat const & executor`) `const` au Form de base et implémenter une méthode exécutant en réalité l'action du Form dans tous les Forms concrets. Vous devez vérifier que le Form est signé et que le bureaucrate qui tente de l'exécuter a un niveau assez élevé, sinon vous lancerez l'exception


appropriée. Que vous souhaitiez effectuer ces contrôles dans chaque classe concrète ou dans la classe de base, puis appeler une autre méthode pour exécuter l'action dépend de vous, mais une méthode est évidemment plus jolie que l'autre. Dans tous les cas, le Form de base doit être une classe abstraite.

Terminez en ajoutant une fonction `executeForm (Form Const & form)` au `bureau-`
`crate`. Il doit essayer d'exécuter le Form et, en cas de succès, affichera quelque chose
comme `<bureaucrat> executs <form>`. Sinon, affiche un message d'erreur explicite.

Ajoutez le nécessaire dans votre `main` pour tester tout cela.

Chapitre V

Exercice 03 : Mieux que d'apporter le café

	Exercice : 03
Mieux que d'apporter le café	
Dossier de rendu : <i>ex03/</i>	
Fichiers à rendre : Pareil qu'avant + Intern.hpp Intern.cpp	
Fonctions interdites : Aucune	

Comme remplir des Forms est assez ennuyant, il serait cruel de demander à nos bureaucrates de les écrire entièrement par eux-mêmes. Non, nous allons simplement demander à un Intern de le faire.

Vous allez créer la classe `Intern`. Le stagiaire n'a pas de nom, pas de grade, pas de caractéristique déterminante, nous nous soucions seulement qu'il fasse son travail.

Le stagiaire a une chose importante, la fonction `makeForm`. Il lui faut deux chaînes de caractères, la première représentant le nom d'un Form et la seconde étant la cible du Form. Il renverra, en tant que pointeur sur `Form`, un pointeur sur la classe de Form concret représentée par le premier paramètre, initialisée avec le deuxième paramètre. Il va imprimer quelque chose comme "`Intern creates <form>` " sur la sortie standard. Une autre méthode du type `if/elseif/elseif/else`, ou autre méthode pas très jolie ne sera pas acceptée en évaluation. Si le Form demandé n'est pas connu, affichez un message d'erreur explicite.

Par exemple, pour créer un `RobotomyRequestForm` ciblé sur "Bender" :


```
{
    Intern  someRandomIntern;
    Form*   rrf;

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

Ajoutez le nécessaire dans votre `main` pour tester tout cela.

Chapitre VI

Exercice 04 : C'est comme ça qu'on les aime, simple et ennuyant

	Exercice : 04
C'est comme ça qu'on les aime, simple et ennuyant	
Dossier de rendu : <i>ex04/</i>	
Fichiers à rendre : Pareil qu'avant + <code>OfficeBlock.cpp</code> <code>OfficeBlock.hpp</code>	
Fonctions interdites : Aucune	



Cet exercice et ceux qui suivent ne rapportent pas de points, mais demeurent intéressant dans le cadre de votre piscine. Vous n'êtes pas obligés de les faire.

La Bureaucratie Centrale, étant le refuge de l'ordre et de l'organisation, est constituée d'immeubles de bureaux bien aménagés. Chacun de ces blocs nécessite un stagiaire et deux bureaucrates pour fonctionner, et est capable de créer, signer et exécuter des formulaires, le tout en lui donnant simplement un ordre. Cool, n'est-ce pas ?

Créez donc une classe `OfficeBlock`. Elle sera construite en passant des pointeurs (ou des références à, vous décidez en fonction de ce qui est approprié) un stagiaire, un bureaucrate signataire et un bureaucrate exécutant. Il peut également être construit vide. Aucune autre construction ne doit être possible (pas de copie, pas d'assignation).

Elle aura pour fonctions de désigner un nouveau stagiaire, un bureaucrate recruteur ou un bureaucrate exécutant.

Sa seule fonction "utile" sera `doBureaucracy`, elle prend un nom de formulaire et un nom de cible. Elle tentera de faire en sorte que le stagiaire crée le formulaire demandé, le fasse signer par le premier bureaucrate et le fasse exécuter par le second. Les messages imprimés par le stagiaire et les bureaucrates fourniront un log de ce qui se passe.

Lorsqu'une erreur survient, une exception doit être générée à partir de cette fonction : vous êtes libre de modifier ce que vous avez fait auparavant pour rendre cette fonction élégante. Rappelez-vous : les messages d'erreurs spécifiques sont toujours cool.

Bien sûr, si les trois places du bloc ne sont pas remplies, aucune bureaucratie ne peut être faite.

Ajoutez le nécessaire dans votre `main` pour tester tout cela.

Par exemple, le bloc de code qui suit pourrait proposer l'output suivant :

```
int main()
{
    Intern      idiotOne;
    Bureaucrat  hermes = Bureaucrat("Hermes Conrad", 37);
    Bureaucrat  bob = Bureaucrat("Bobby Bobson", 123);
    OfficeBlock ob;


    ob.setIntern(idiotOne);
    ob.setSigner(bob);
    ob.setExecutor(hermes);

    try
    {
        ob.doBureaucracy("mutant pig termination", "Pigley");
    }
    catch (Some::SpecificException & e)
    {
        /* specific known error happens, say something */
    }
    catch (std::exception & e)
    {
        /* oh god, unknown error, what to do ?! */
    }
}
```

```
$> ./ex04
Intern creates a Mutant Pig Termination Form (s.grade 130, ex.grade 50) targeted on Pigley (Unsigned)
Bureaucrat Bobby Bobson (Grade 123) signs a Mutant Pig Termination Form (s.grade 130, ex.grade 50)
    targeted on Pigley (Unsigned)
Bureaucrat Hermes Conrad (Grade 37) executes a Mutant Pig Termination Form (s.grade 130, ex.grade 50)
    targeted on Pigley (Signed)
That'll do, Pigley. That'll do ...
$>
```

Chapitre VII

Exercice 05 : Générateur infini de signatures

	Exercice : 05
Générateur infini de signatures	
Dossier de rendu : <i>ex05/</i>	
Fichiers à rendre : Pareil qu'avant + <code>CentralBureaucracy.cpp</code> <code>CentralBureaucracy.hpp</code>	
Fonctions interdites : Aucune	

Il ne reste plus qu'à emballer tout cela dans un joli petit paquet.

Créez la classe `CentralBureaucracy`. Elle sera créée sans paramètres, et à sa création aura 20 blocs de bureaux vides.

Il sera possible de "nourrir" les bureaucrates de l'objet. Les stagiaires seront générés automatiquement, sans intervention de l'utilisateur, car avouons-le, les stagiaires ne coûtent pas grand-chose.

Les bureaucrates qui sont "nourris" à l'objet seront utilisés pour occuper des sièges dans ses immeubles de bureaux. Si aucune place n'est disponible, vous pouvez soit les refuser, soit les stocker quelque part dans une salle d'attente.

Après cela, il sera possible de mettre les cibles en file d'attente dans l'objet, en utilisant un `queueUp` fonction qui prend une chaîne, le nom de la personne en file d'attente.

Enfin, quand une fonction `doBureaucracy` est appelée, faites un peu de Bureaucratie au hasard, premier arrivé, premier servi, dans l'ordre des blocs créés.

Enfin, voici à quoi pourrait ressembler votre `main` :

- Créez la Bureaucratie Centrale
- Créez 20 bureaucrates aléatoires et envoyez-les à la Bureaucratie Centrale

- Mettez en file d'attente un grand nombre de cibles dans la Bureaucratie Centrale
- Appelez la fonction `doBureaucracy()` et observez la magie s'effectuer