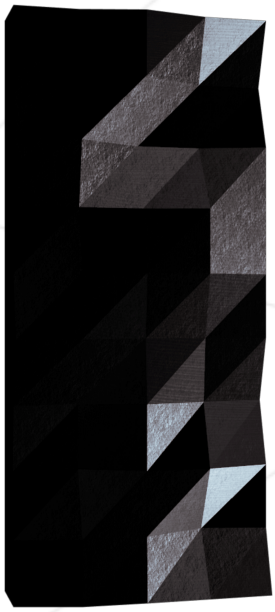

C++ - Module 03

Héritage

Résumé:
Ce document contient les exercices du module 03 des modules C++.

Version: 7

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 présenter**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. 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 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 avecc++et les drapeaux -Mur -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, ... , par exemple
- Nommez vos fichiers, classes, fonctions, fonctions membres et attributs comme requis dans les directives.
- Écrivez les noms des classes dans**UpperCamelCase**format. Les fichiers contenant le code de classe seront toujours nommés selon le nom de la classe. Par exemple : NomDeClasse.hpp/NomDeClasse.h, NomDeClasse.cpp,ouNom de classe.tpp.Ensuite, si vous avez un fichier d'en-tête contenant la définition d'une classe « BrickWall » représentant un mur de briques, son nom seraMur de briques.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 Nominette !*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 pas utiliser d'autres bibliothèques externes. Cela signifie C++11 (et les formes dérivées) etBoosterLes bibliothèques sont interdites. Les fonctions suivantes sont également interdites : *printf(), *alloc()etgratuit().Si vous les utilisez, votre note sera de 0 et c'est tout.

- Veuillez noter que, sauf indication contraire explicite, leen utilisant l'espace de noms <ns_name>et amiles mots-clés sont interdits. Sinon, votre note sera de -42.
- **Vous êtes autorisé à utiliser le STL dans les modules 08 et 09 uniquement.**Cela signifie : non**Conteneurs**(vecteur/liste/carte/et ainsi de suite) et non**Algorithmes**(tout ce qui nécessite d'inclure le <algorithme>(en-tête) 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**fuites de mémoire**.
- Du module 02 au module 09, vos cours doivent être conçus dans le**Forme canonique orthodoxe, 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**inclure des gardes**. 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 montrer des 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 !!!

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, respectez les règles obligatoires et ne soyez pas paresseux. Vous passeriez à côté de beaucoup d'informations utiles ! N'hésitez pas à lire les notions théoriques.

Chapitre III

Exercice 00 : Et... OUVERT !

	Exercice : 00
Et... OUVERT !	
Répertoire de remise :*ex*00*/*	
Dossiers à rendre :Makefile, main.cpp, ClapTrap.{h, hpp}, ClapTrap.cpp	
Fonctions interdites :Aucun	

Il faut d'abord implémenter une classe ! Quelle originalité !

On l'appellera**Boniment**et aura les attributs privés suivants initialisés aux valeurs spécifiées entre parenthèses :

- Nom, qui est passé en paramètre à un constructeur
- Les points de vie (10) représentent la santé du ClapTrap
- Points d'énergie (10)
- Dégâts d'attaque (0)

Ajoutez les fonctions membres publiques suivantes pour que le ClapTrap soit plus réaliste :

- void attaque(const std::string& cible);
- void takeDamage(unsigned int montant);
- void beRepaired(unsigned int montant);

Lorsque ClapTrack attaque, il fait perdre à sa cible <dégâts d'attaque>points de vie. Lorsque ClapTrap se répare, il obtient <montant>points de vie en retour. Attaquer et réparer coûtent 1 point d'énergie chacun. Bien sûr, ClapTrap ne peut rien faire s'il n'a plus de points de vie ou de points d'énergie.

Dans toutes ces fonctions membres, vous devez afficher un message pour décrire ce qui se passe. Par exemple,attaque()la fonction peut afficher quelque chose comme (bien sûr, sans les crochets angulaires) :

ClapTrap <nom> attaque <cible>, causant <dégâts> points de dégâts !

Les constructeurs et le destructeur doivent également afficher un message, afin que vos évaluateurs puissent facilement voir qu'ils ont été appelés.

Implémentez et remettez vos propres tests pour vous assurer que votre code fonctionne comme prévu.

Chapitre IV

Exercice 01 : Serena, mon amour !

	Exercice : 01
Serena, mon amour !	
Répertoire de remise :*ex*01/	
Dossiers à rendre :Fichiers de l'exercice précédent + ScavTrap.{h, hpp}, ScavTrap.cpp	
Fonctions interdites :Aucun	

Parce qu'on n'a jamais assez de ClapTraps, vous allez maintenant créer un robot dérivé. Il sera nommé**Piège à scav**et héritera des constructeurs et du destructeur de Clap-Trap. Cependant, ses constructeurs, son destructeur etattaque()imprimera des messages différents. Après tout, les ClapTraps sont conscients de leur individualité.

Notez que le chaînage de construction/destruction correct doit être montré dans vos tests. Lorsqu'un ScavTrap est créé, le programme commence par construire un ClapTrap. La destruction s'effectue dans l'ordre inverse. Pourquoi ?

Piège à scavutilisera les attributs de ClapTrap (mettra à jour ClapTrap en conséquence) et devra les initialiser à :

- Nom, qui est passé en paramètre à un constructeur
- Les points de vie (100) représentent la santé du ClapTrap
- Points d'énergie (50)
- Dégâts d'attaque (20)

ScavTrap aura également sa propre capacité spéciale :

```
void gardeGate();
```

Cette fonction membre affichera un message informant que ScavTrap est désormais en mode gardien de porte.

N'oubliez pas d'ajouter plus de tests à votre programme.

Chapitre V

Exercice 02 : Travail répétitif

	Exercice : 02
Travail répétitif	
Répertoire de remise :*ex*02*/*	
Dossiers à rendre :Fichiers des exercices précédents + FragTrap.{h, hpp}, FragTrap.cpp	
Fonctions interdites :Aucun	

Faire des ClapTraps commence probablement à vous énerver.

Maintenant, implémentez un**Piège à fragmentation**classe qui hérite de ClapTrap. Elle est très similaire à ScavTrap. Cependant, ses messages de construction et de destruction doivent être différents. Un enchaînement de construction/destruction correct doit être montré dans vos tests. Lorsqu'un FragTrap est créé, le programme commence par construire un ClapTrap. La destruction se fait dans l'ordre inverse.

Même chose pour les attributs, mais avec des valeurs différentes cette fois-ci :

- Nom, qui est passé en paramètre à un constructeur
- Les points de vie (100) représentent la santé du ClapTrap
- Points d'énergie (100)
- Dégâts d'attaque (30)

FragTrap a également une capacité spéciale :

vide highFivesGuys(void);

Cette fonction membre affiche une demande de high fives positive sur la sortie standard.

Encore une fois, ajoutez plus de tests à votre programme.

Chapitre VI

Exercice 03 : Maintenant c'est bizarre !

	Exercice : 03
Maintenant c'est bizarre !	
Répertoire de remise :*ex*03*/*	
Dossiers à rendre :Fichiers des exercices précédents + DiamondTrap.{h, hpp}, DiamondTrap.cpp	
Fonctions interdites :Aucun	

Dans cet exercice, vous allez créer un monstre : un ClapTrap qui est à moitié FragTrap, à moitié ScavTrap. Il sera nommé**Piège à diamants**, et il héritera à la fois du FragTrap et du ScavTrap. C'est tellement risqué !

La classe DiamondTrap aura unnomattribut privé. Donnez à cet attribut exactement le même nom de variable (je ne parle pas ici du nom du robot) que celui de la classe de base ClapTrap.

Pour être plus clair, voici deux exemples.
Si la variable de ClapTrap estnom,donne le nomnomà celle du DiamondTrap. Si la variable de ClapTrap est _nom,donne le nom _nomà celui du DiamondTrap.

Ses attributs et fonctions membres seront choisis parmi l'une de ses classes parentes :

- Nom, qui est passé en paramètre à un constructeur
- ClapTrap::nom (paramètre du constructeur + "_clap_name"suffixe)
- Points de vie (FragTrap)
- Points d'énergie (ScavTrap)
- Dégâts d'attaque (FragTrap)
- attaque() (Piège à scav)

En plus des fonctions spéciales de ses deux classes parentes, DiamondTrap aura sa propre capacité spéciale :

void qui suis-je();

Cette fonction membre affichera à la fois son nom et son nom ClapTrap.

Bien sûr, le sous-objet ClapTrap du DiamondTrap sera créé une fois, et une seule fois. Oui, il y a une astuce.

Encore une fois, ajoutez plus de tests à votre programme.

Connaissez-vous les indicateurs de compilateur -Wshadow et -Wno-shadow ?

Vous pouvez réussir ce module sans faire l'exercice 03.

Chapitre VII

Soumission et évaluation par les pairs

Remettez votre devoir dans votreGitcomme d'habitude. Seuls les travaux contenus dans votre dépôt seront évalués 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$$cf36316f07f871b4f14926007c37d388