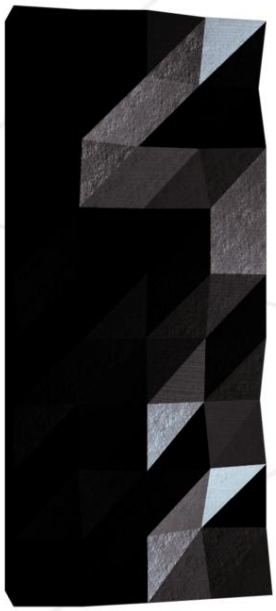

C++-Module 03

Héritage

Résumé : Ce document contient le sujet du module 03 des modules C++.

Modèle : 6

Contenu

Chapitre I

Règles générales

Pour les modules C++, vous n'utiliserez et n'apprendrez que C++98. L'objectif est que vous appreniez les bases d'un langage de programmation orienté objet. Nous savons que le C++ moderne est très différent à bien des égards, donc si vous voulez devenir un développeur C++ compétent, vous aurez besoin du C++ standard moderne plus tard. Ce sera le point de départ de votre parcours C++ à vous d'aller plus loin après le 42 Common Core !

- Toute fonction implémentée dans un en-tête (sauf dans le cas des modèles), et toute en-tête non protégé signifie 0 à l'exercice.
- Chaque sortie va à la sortie standard et se terminera par une nouvelle ligne, sauf indication contraire.
- Les noms de fichiers imposés doivent être suivis à la lettre, ainsi que les noms de classes, fonc les noms de fonction et les noms de méthode.
- N'oubliez pas : vous codez en C++ maintenant, et non plus en C. Donc:
  - ÿ Les fonctions suivantes sont INTERDITES, et leur utilisation sera punie par a 0, aucune question posée : *alloc, *printf et free.
  - ÿ Vous êtes autorisé à utiliser tout ce qui se trouve dans la bibliothèque standard. CEPENDANT, il serait judicieux d'essayer d'utiliser les versions C++-ish des fonctions auxquelles vous êtes habitué en C, au lieu de vous en tenir à ce que vous savez, c'est un nouveau langage après tout. Et NON, vous n'êtes pas autorisé à utiliser la STL tant que vous n'êtes pas censé le faire (c'est-à-dire jusqu'au module 08). Cela signifie pas de vecteurs/listes/cartes/etc... ou tout ce qui nécessite un <algorithme> d'inclusion jusque-là.
- En fait, l'utilisation de toute fonction ou mécanique explicitement interdite sera punie d'un 0, sans poser de questions.
- Notez également que sauf indication contraire, les mots clés C++ "using namespace" et "friend" sont interdits. Leur utilisation sera sanctionnée par un -42, sans poser de questions.
- Les fichiers associés à une classe seront toujours ClassName.hpp et ClassName.cpp, sauf indication contraire.
- Les répertoires de restitution sont ex00/, ex01/, . . . ,ex/.

- Vous devez lire attentivement les exemples. Ils peuvent contenir des exigences qui ne sont pas évidentes dans la description de l'exercice.
- Puisque vous êtes autorisé à utiliser les outils C++ que vous avez appris depuis le début, vous n'êtes pas autorisé à utiliser une bibliothèque externe. Et avant de demander, cela signifie également pas de C++11 et de dérivés, ni de Boost ou quoi que ce soit d'autre.
- Vous devrez peut-être rendre un nombre important de cours. Cela peut sembler fastidieux à moins que vous ne puissiez créer un script avec votre éditeur de texte préféré.
- Lisez ENTIÈREMENT chaque exercice avant de le commencerÿ! Fais le.
- Le compilateur à utiliser est c++.
- Votre code doit être compilé avec les drapeaux suivants : -Wall -Wextra -Werror.
- Chacune de vos inclusions doit pouvoir être incluse indépendamment des autres.
  Les inclusions doivent contenir toutes les autres inclusions dont elles dépendent.
- Au cas où vous vous poseriez la question, aucun style de codage n'est appliqué en C++. Vous pouvez utiliser n'importe quel style que vous aimez, sans aucune restriction. Mais rappelez-vous qu'un code que votre pair-évaluateur ne peut pas lire est un code qu'il ne peut pas noter.
- Choses importantes maintenantÿ: vous ne serez PAS noté par un programme, sauf si cela est explicitement indiqué dans le sujet. Par conséquent, vous bénéficiez d'une certaine liberté dans la façon dont vous choisissez de faire les exercices. Cependant, soyez conscient des contraintes de chaque exercice, et NE soyez PAS paresseux, vous manqueriez BEAUCOUP de ce qu'ils ont à offrir
- Ce n'est pas un problème d'avoir des fichiers superflus dans ce que vous remettez, vous pouvez choisir de séparer votre code dans plus de fichiers que ce qui vous est demandé. N'hésitez pas, tant que le résultat n'est pas noté par un programme.
- Même si le sujet d'un exercice est court, cela vaut la peine d'y consacrer du temps pour être sûr de bien comprendre ce qu'on attend de vous, et que vous l'avez fait de la meilleure façon possible.
- Par Odin, par Thor ! Utilise ton cerveau!!!

Chapitre II

Exercice 00ÿ: Aaaaand... OUVRIRÿ!

	Exercice : 00
Aaaaand... OUVERT !	
Répertoire de restitution :	
ex00/ Fichiers à rendre : Makefile ClapTrap.cpp ClapTrap.hpp main.cpp Fonctions	
interdites : Aucune	

Ici, vous devez faire une classe! Quelle originalité !

La classe s'appellera ClapTrap et aura les attributs privés suivants, initialisés aux valeurs spécifiéesÿ:

• Nom (Paramètre du constructeur)

• Points de vie (10)

• Points d'énergie (10)

• Dégâts d'attaque (0)

Vous lui donnerez également quelques fonctions publiques pour le rendre plus vivant :

• void attack(std::string const & target)

• annuler takeDamage (montant entier non signé)

• annuler beRepaired (montant entier non signé)

Dans toutes ces fonctions, vous devez afficher quelque chose pour décrire ce qui se passe.
Par exemple, la fonction d'attaque peut afficher quelque chose commeÿ:

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

Le constructeur et le destructeur doivent également afficher quelque chose, afin que les gens puissent voir qu'ils ont été appelés.

Vous fournirez une fonction principale, avec suffisamment de tests pour démontrer que votre code est fonctionnel.

Chapitre III

Exercice 01 : Serena, mon amour !

	Exercice : 01
Séréna, mon amour !	
Répertoire de restitution : ex01/	
Fichiers à rendre : Idem exercice précédent + ScavTrap.cpp ScavTrap.hpp Fonctions interdites : Aucune	

Parce que nous ne pouvons jamais avoir assez de Claptraps, maintenant vous allez en faire un autre.
La classe sera nommée ScavTrap et héritera de ClapTrap, le constructeur, le destructeur et l'attaque doivent utiliser des sorties différentes. Après tout, un Claptrap doit avoir une certaine individualité.

La classe ScavTrap aura ses messages de construction et de destruction. De plus, un chaînage de construction/destruction correct doit être présent (lorsque vous construisez un ScavTrap, vous devez commencer par construire un ClapTrap... La destruction est dans l'ordre inverse), et les tests doivent le montrer.

ScavTrap utilisera les attributs de Claptrap (vous devez modifier l'accord Claptrap simplement)ÿ! Et doit les initialiser àÿ:

- Nom (Paramètre du constructeur)
- Points de vie (100)
- Points d'énergie (50)
- dégâts d'attaque (20)

ScavTrap aura également une nouvelle fonction spécifique : void guardGate(); cette fonction affichera un message sur les sorties standard pour annoncer que ScavTrap est entré en mode Gate keeper.

Étendez votre fonction principale pour tout tester.

Chapitre IV

Exercice 02 : Travail répétitif

	Exercice : 02
Travail répétitif	
Répertoire de restitution :	
ex02/ Fichiers à rendre : Idem exercice précédent + FragTrap.cpp FragTrap.hpp	
Fonctions interdites : Aucune	

Faire des Claptraps commence probablement à vous énerver.

Maintenant, vous allez créer une classe FragTrap qui hérite de ClapTrap.

• Nom (Paramètre du constructeur)

• Points de vie (100)

• Points d'énergie (100)

• dégâts d'attaque (30)

La classe FragTrap aura ses messages de construction et de destruction. De plus, un chaînage de construction/destruction correct doit être présent (lorsque vous construisez un FragTrap, vous devez commencer par construire un ClapTrap... La destruction est dans l'ordre inverse), et les tests doivent le montrer.

La fonction spécifique pour FragTrap sera void highFivesGuys(void) et sera dis
jouer une demande positive de high fives sur la sortie standard.

Étendez votre fonction principale pour tout tester.

Chapitre V

Exercice 03 : Maintenant c'est bizarre !

	Exercice : 03
Maintenant c'est bizarre !	
Répertoire de restitution :	
ex03/ Fichiers à rendre : Idem exercice précédent + DiamondTrap.cpp DiamondTrap.hpp	
Fonctions interdites : Aucune	

Maintenant, vous allez créer un monstre en faisant un Claptrap qui est moitié Fragtrap, moitié Scav Trap.

Il s'appellera DiamondTrap, et il héritera à la fois du FragTrap ET du ScavTrap.

Acte très risqué et horrible, la classe DiamondTrap déclarera un attribut privé appelé nom comme celui à l'intérieur de ClapTrap.

Ses attributs et fonctions seront choisis parmi l'une ou l'autre de ses classes parentesÿ:

- Nom (Paramètre du constructeur)
- Claptrap::Name (Paramètre du constructeur + "_clap_name")
- Points de vie (Fragtrap)
- Points d'énergie (Scavtrap)
- Dégâts d'attaque (Fragtrap)
- attaque (Scavtrap)

Il aura les fonctions spéciales des deux.

Comme d'habitude, votre main sera étendue pour tester la nouvelle classe.

Bien sûr, la partie Claptrap du Diamondtrap sera créée une fois, et une seule... Oui, il y a une astuce.

DiamondTrap aura une nouvelle fonction void whoAmI(); qui affichera son nom et son nom clapTrap.

Drapeaux de compilation comme -Wshadow et -Wno-shadowÿ!