



rrouille

SCALE FOR PROJECT CPP MODULE 03

You should evaluate 1 student in this team



Git repository

git@vogsphere.42lausanne.ch



Introduction

Merci de respecter les règles suivantes:

- Restez polis, courtois, respectueux et constructifs pendant le processus

d'évaluation. Le bien-être de la communauté repose là-dessus.

- Identifiez avec la personne évaluée ou le groupe évalué les éventuels

dysfonctionnements de son travail. Prenez le temps d'en discuter et

débattez des problèmes identifiés.

- Vous devez prendre en compte qu'il peut y avoir de légères différences

d'interprétation entre les instructions du projet, son scope et ses

fonctionnalités. Gardez un esprit ouvert et notez de la manière la plus honnête possible. La pédagogie n'est valide que si la peer-évaluation est faite sérieusement.

Guidelines

- Ne notez que ce qui est contenu dans le dépôt Git cloné de l'étudiant(e) ou du groupe.
- Vérifiez que le dépôt Git appartient bien à l'étudiant(e) ou au groupe, que le projet est bien celui attendu, et que "git clone" est utilisé dans un dossier vide.
- Vérifiez scrupuleusement qu'aucun alias n'a été utilisé pour vous tromper et assurez-vous que vous évaluez bien le rendu officiel.
- Afin d'éviter toute surprise, vérifiez avec l'étudiant(e) ou le groupe les potentiels scripts utilisés pour faciliter l'évaluation (par exemple, des scripts de tests ou d'automatisation).
- Si vous n'avez pas fait le projet que vous allez évaluer, vous devez lire le sujet en entier avant de commencer l'évaluation.
- Utilisez les flags disponibles pour signaler un rendu vide, un programme ne fonctionnant pas, une erreur de Norme, de la triche... Dans ces situations, l'évaluation est terminée et la note est 0, ou -42 en cas de triche. Cependant, à l'exception des cas de triche, vous

êtes encouragé(e)s
à continuer la discussion sur le travail rendu, même si ce dernier est incomplet. Ceci afin d'identifier les erreurs à ne pas reproduire dans le futur.

- Si le sujet requiert un fichier de configuration, vous ne devriez jamais avoir à le modifier. Si vous souhaitez éditer un fichier, prenez le temps d'expliquer pourquoi à la personne évaluée et de vous assurer que vous avez son accord.

- Vous devez aussi vérifier l'absence de fuites mémoire. Toute mémoire allouée sur le tas doit être libérée proprement avant la fin de l'exécution du programme.
Vous avez le droit d'utiliser tout outil disponible sur la machine tel que leaks, valgrind ou e_fence. En cas de fuites mémoire, cochez le flag approprié.

Attachments



subject.pdf

Tests préliminaires

Si un cas de triche est suspecté, la notation et l'évaluation prennent fin immédiatement. Pour le signaler, sélectionnez le flag "Cheat". Faites attention à l'utiliser avec calme, précaution et discernement.

Prérequis

Le code doit compiler avec `c++` et les flags `-Wall -Wextra -Werror`

Pour rappel, ce projet doit suivre le standard C++98. Par conséquent,

des fonctions C++11 (ou autre standard) et les containers ne sont PAS attendus.

Ne notez pas l'exercice si vous trouvez :

- Une fonction implémentée dans un fichier d'en-tête (sauf pour les fonctions templates).
- Un Makefile compilant sans les flags demandés et/ou avec autre chose que `c++`.

Sélectionnez le flag "Fonction interdite" (Forbidden function) si

vous rencontrez :

- L'utilisation d'une fonction "C" (`* alloc, * printf, free`).
- L'utilisation d'une fonction interdite dans le projet.
- L'utilisation de "using namespace <ns_name>" ou du mot-clé "friend".
- L'utilisation d'une bibliothèque externe, ou de fonctionnalités propres aux versions postérieures à C++98.

Yes

No

Exercice 00 : Eeeeet... ACTION !

Comme d'habitude, il doit y avoir assez de tests pour prouver que le programme fonctionne comme demandé. S'il n'y en a pas, ne notez pas cet exercice.

Classe et attributs

Il y a une classe ClapTrap.

Elle possède tous les attributs privés suivants :

- name (nom)
- hit points (points de vie)
- energy points (points d'énergie)
- attack damage (dommages d'attaque) Ces attributs sont initialisés aux valeurs demandées.

Yes

No

Fonctions membres

Les fonctions membres suivantes sont présentes et fonctionnent comme spécifié :

- attack()
- takeDamage()
- beRepaired()

Yes

No

Exercice 01 : Serena, mon amour !

Comme d'habitude, il doit y avoir assez de tests pour prouver que le programme fonctionne comme demandé. S'il n'y en a pas, ne notez pas cet exercice.

Classe et attributs

Il y a une classe ScavTrap.

Le ScavTrap hérite publiquement de la classe ClapTrap.

Elle ne redéclare pas les attributs.

Les attributs de la classe ClapTrap sont désormais protégés

et non plus

privés.

Les attributs sont initialisés aux valeurs demandées.

Yes

No

Fonctions membres

Les fonctions membres suivantes sont présentes et fonctionnelles :

- attack()
- takeDamage() (héritée)
- beRepaired() (héritée)

Les messages du constructeur, du destructeur, et de la fonction attack()
doivent être différents de ceux du ClapTrap.

Yes

No

Construction et destruction

Le ScavTrap doit avoir un constructeur et un destructeur avec des messages spécifiques.

Leur bonne implémentation doit être démontrée par un enchaînement des appels dans l'ordre attendu : si vous créez un ScavTrap, le message du constructeur du ClapTrap doit être affiché en premier, suivi de celui du ScavTrap.

À l'inverse, si vous supprimez un ScavTrap, le message du destructeur du

ScavTrap doit être affiché en premier, suivi de celui du ClapTrap.

Yes

No

Caractéristique spéciale

Le ScavTrap possède une fonction `guardGate()` qui affiche un message sur la sortie standard.

Le ScavTrap a également une fonction `attack()` qui affiche un message différent de celui du ClapTrap sur la sortie standard.

Yes

No

Exercice 02 : Travail à la chaîne

Comme d'habitude, il doit y avoir assez de tests pour prouver que le programme fonctionne comme demandé. S'il n'y en a pas, ne notez pas cet exercice.

Classe et attributs

Il existe une classe `FragTrap` qui hérite publiquement de `ClapTrap`.

Les attributs ne doivent pas être redéclarés sans raison.

Yes

No

Construction et destruction

Le `FragTrap` doit avoir un constructeur et un destructeur avec des messages spécifiques.

Leur bonne implémentation doit être démontrée par un enchaînement des appels dans l'ordre attendu : si vous créez un `FragTrap`, le message du constructeur du `ClapTrap` doit être affiché en premier, suivi de celui du

FragTrap.

À l'inverse, si vous supprimez un FragTrap, le message du destructeur du FragTrap doit être affiché en premier, suivi de celui du ClapTrap.

Yes

No

Caractéristique spéciale

Le FragTrap possède une fonction highFivesGuys() qui affiche un message sur la sortie standard.

Yes

No

Exercice 03 : Ok, ça devient bizarre

Comme d'habitude, il doit y avoir assez de tests pour prouver que le programme fonctionne comme demandé. S'il n'y en a pas, ne notez pas cet exercice.

L'ultime étrangeté du C++

Il y a une classe DiamondTrap.
Elle hérite à la fois du FragTrap et du ScavTrap.
Elle définit les attributs avec les valeurs demandées.
Elle utilise l'héritage virtuel pour éviter les pièges de l'héritage en diamant.

Yes

No

Choisissez judicieusement...

La classe DiamondTrap utilise la fonction attack() du Scavtrap.
Elle possède les fonctions spéciales de ses deux parents.
La DiamondTrap possède un membre privé std::string name.
La fonction whoAmI() a accès à la fois à name et à ClapTrap::name.

Yes

No

Ratings

Don't forget to check the flag corresponding to the defense

Ok

Outstanding project

Empty work

Incomplete work

Invalid compilation

Cheat

Crash

Concerning situation

Leaks

Forbidden function

Conclusion

Leave a comment on this evaluation

Finish evaluation