DeepL

Abonnez-vous à DeepL Pro pour traduire des fichiers plus volumineux.
Visitez www.DeepL.com/pro pour en savoir plus.

ÉCHELLE POUR LE PROJET CPP MODULE 01 (/PROJECTS/CPP-MODULE-01)

Vous devez évaluer 1 étudiant dans cette équipe

Dépôt Git

```
git@vogsphere.msk.21-school.ru:vogsphere/intra-uuid-d5bec79
```

Introduction

Veuillez respecter les règles suivantes :

- Restez poli, courtois, respectueux et constructif tout au long du processus d'évaluation. Le bien-être de la communauté en dépend.

- Identifiez avec l'élève ou le groupe dont le travail est évalué les éventuels dysfonctionnements de leur projet. Prenez le temps de discuter et de débattre des problèmes qui ont pu être identifiés.

- Vous devez tenir compte du fait qu'il peut y avoir des différences dans la manière dont vos pairs ont compris les instructions du projet et l'étendue de ses fonctionnalités. Gardez toujours l'esprit ouvert et évaluez-les aussi honnêtement que possible. La pédagogie n'est utile que si l'évaluation par les pairs est faite sérieusement.

Lignes directrices

- Ne notez que le travail qui a été remis dans le dépôt Git de l'étudiant ou du groupe évalué.

- Vérifier que le dépôt Git appartient bien à l'étudiant ou aux étudiants. Assurez-vous que le projet est bien celui attendu. Vérifiez également que l'option 'git clone' est utilisée dans un dossier vide.

- Vérifiez soigneusement qu'aucun alias malveillant n'a été utilisé pour vous tromper et vous faire évaluer quelque chose qui n'est pas le contenu du dépôt official.

- Pour éviter toute surprise et, le cas , revoir ensemble les scripts utilisés

pour faciliter la notation (scripts pour les tests ou l'automatisation).

- Si vous n'avez pas terminé le travail que vous allez évaluer, vous devez lire l'intégralité du sujet avant de commencer le processus d'évaluation.

- Utilisez les flags disponibles pour signaler un dépôt vide, un programme qui ne fonctionne pas, une erreur de norme, une tricherie, etc.
Dans ces , le processus d'évaluation se termine et la note finale est 0, ou -42 en cas de tricherie. Cependant, à l'exception de la tricherie, les étudiants sont sont fortement encouragés à revoir ensemble le travail , afin d'identifier les erreurs à ne pas répéter à l'avenir.

- Vous ne devriez jamais avoir à éditer un file autre que le file de configuration, s'il existe. Si vous souhaitez modifier un file, prenez le temps d'en expliquer les raisons.
avec l'élève évalué et assurez-vous que vous êtes tous les deux d'accord.

- Vous devez également vérifier l'absence de fuites de mémoire. Toute mémoire allouée sur le tas doit être correctement libérée avant la fin de l'exécution.
Vous êtes autorisé à utiliser les différents outils disponibles sur l'ordinateur, tels que , valgrind ou e_fence. En cas de fuites de mémoire, cochez la case correspondante.

Pièces jointes

- sujet.pdf (https://cdn.intra.42.fr/pdf/pdf/40222/en.subject.pdf)

Tests préliminaires

En cas de suspicion de tricherie, l'évaluation s'arrête ici. Utilisez le fiag "Cheat" pour signaler. Prenez cette décision avec calme et sagesse, et utilisez ce bouton avec précaution.

Conditions préalables

Le code doit se compiler avec c++ et les flags -Wall -Wextra -Werror N'oubliez pas que ce projet doit suivre le standard C++98. Par conséquent, les fonctions ou conteneurs C++11 (et ultérieurs) ne sont PAS attendus.

Dans tous les cas, vous ne devez pas noter l'exercice en question :
- Une fonction est implémentée dans un file d'en-tête (à l'exception des fonctions modèles).
- Un Makefile compile sans les flags requis et/ou un autre compilateur que c++.

Dans tous les cas, vous devez marquer le projet avec la mention "Fonction interdite" :
- Utilisation d'une fonction "C" (*alloc, *printf, free).
- Utilisation d'une fonction non autorisée dans les lignes directrices de l'exercice.

- Utilisation de "using namespace" ou du mot-clé "friend".
- Utilisation d'une bibliothèque externe ou de fonctionnalités provenant de versions autres que C++98.

☐ Oui ☐ Non

Ex00 : BraiiiiiiinnnzzzZ

L'objectif de cet exercice est de comprendre comment allouer de la mémoire en C++.

Makefile et tests

Il y a un Makefile qui compile en utilisant les flags appropriés. Il y a au moins un main pour tester l'exercice.

☐ Oui ☐ Non

Classe Zombie

Il existe une classe de zombies.
Il possède un attribut de nom privé.
Il possède au moins un constructeur.
Il possède une fonction membre announce(void) qui imprime : " : BraiiiiinnnzzzZ..." Le destructeur affiche un message de débogage qui inclut le nom du zombie.

☐ Oui ☐ Non

nouveauZombie

Il existe une fonction newZombie() dont le prototype est le suivant : [Zombie* newZombie(std::string name) ;] : [Zombie* newZombie(std::string name) ;] Elle devrait allouer un Zombie sur le tas et le retourner.
Idéalement, il devrait appeler le constructeur qui prend une chaîne et initialise le nom.
L'exercice doit être noté comme correct si le Zombie peut s'annoncer avec le nom passé à la fonction.
Il existe des tests qui prouvent que tout fonctionne.
Le zombie est supprimé correctement avant la fin du programme.

☐ Oui ☐ Non

randomChump

Il y a une fonction randomChump() dont le prototype est le suivant : [void randomChump(std::string name) ;] : [void randomChump(std::string name) ;] Elle devrait créer un Zombie sur la pile, et le faire s'annoncer.
Idéalement, le zombie devrait être alloué sur la pile (et donc implicitement supprimé à la fin de la fonction). Il peut également être alloué sur le tas et ensuite

explicitement supprimée.
L'étudiant doit justifier ses choix. Il y a des tests
pour prouver que tout fonctionne.

☐ Oui ☐ Non

Ex01 : Plus de cerveaux !

Le but de cet exercice est d'allouer un certain nombre d'objets en même temps en utilisant new[], de les initialiser et de les supprimer correctement.

Makefile et tests

Il y a un Makefile qui compile en utilisant les flags appropriés. Il y a au
moins un main pour tester l'exercice.

☐ Oui ☐ Non

zombieHorde

La classe Zombie possède un constructeur par défaut.
Il existe une fonction zombieHorde() dont le prototype est le suivant : [Zombie* zombieHorde(int N, std::string name) ;] : [Zombie* zombieHorde(int N, std::string name) ;] Elle alloue N zombies sur le tas en utilisant explicitement new[].
Après l'allocation, il y a une initialisation des objets pour définir leur nom. Il renvoie un
pointeur sur le first zombie.
Il y a suffisamment de tests dans le main pour prouver les points précédents.
Ex : appeler announce() sur tous les zombies.
Enfin, tous les zombies doivent être supprimés en même temps dans la partie principale.

☐ Oui ☐ Non

Ex02 : HI THIS IS BRAIN

Démystifier les références ! Démystifier les références ! Démystifier les références ! Démystifier les références ! Démystifier les références ! Démystifier les références ! Démystifier les références ! Démystifier les références ! Démystifier les références ! Démystifier les références ! Démystifier les références ! Démystifier les références !

Makefile et tests

Il y a un Makefile qui compile en utilisant les flags appropriés. Il y a au
moins un main pour tester l'exercice.

☐ Oui ☐ Non

BONJOUR, JE M'APPELLE BRAIN

Il existe une chaîne contenant "HI THIS IS BRAIN". stringPTR est un pointeur sur la chaîne.
stringREF est une référence à la chaîne de caractères.
L'adresse de la chaîne est affichée à l'aide de la variable string, de stringPTR et de stringREF.
La chaîne est affichée à l'aide de stringPTR et de stringREF.

☐ Oui ☐ Non

Ex03 : Violence inutile

L'objectif de cet exercice est de comprendre que les pointeurs et les références présentent quelques petites différences qui les rendent plus appropriés en fonction de l'utilisation et du cycle de vie de l'objet utilisé.

Makefile et tests

Il y a un Makefile qui compile en utilisant les flags appropriés. Il y a au moins un main pour tester l'exercice.

☐ Oui ☐ Non

Arme

Il existe une classe Weapon qui possède un type string, une fonction getType() et une fonction setType(). La fonction getType() renvoie une référence constante à la chaîne de caractères.

☐ Oui ☐ Non

HumanA et HumanB

HumanA peut avoir une référence ou un pointeur sur l'arme.
Idéalement, elle devrait être mise en œuvre sous la forme d'une référence, puisque l'arme existe depuis sa création jusqu'à sa destruction, et qu'elle ne change jamais.
HumanB doit avoir un pointeur sur une arme, car le champ n'est pas défini au moment de la création, et l'arme peut être NULL.

☐ Oui ☐ Non

Ex04 : Sed est pour les perdants

Grâce à cet exercice, l'étudiant devrait s'être familiarisé avec ifstream et ofstream.

Makefile et tests

Il y a un Makefile qui compile en utilisant les flags appropriés. Il y a au moins un main pour tester l'exercice.

☐ Oui ☐ Non

ex04

Il existe une fonction replace (ou autre nom) qui fonctionne comme spécifié dans le sujet.
La gestion des erreurs est efficient : essayer de passer un file qui n'existe pas, changer les permissions, le passer vide, etc.
Si vous pouvez trouver une erreur qui n'est pas gérée et qui n'est pas complètement ésotérique, aucun point pour cet exercice.
Le programme doit lire dans la file à l'aide d'un ifstream ou équivalent, et écrire à l'aide d'un ofstream ou équivalent.
La mise en œuvre de la fonction doit se faire à l'aide de fonctions de std::string, et non en lisant la chaîne caractère par caractère.
Ce n'est plus C !

☐ Oui ☐ Non

ex05 : Karen 2.0

Le but de cet exercice est d'utiliser des pointeurs vers des fonctions membres de la classe. C'est aussi l'occasion de découvrir les différents niveaux de log.

Makefile et tests

Il y a un Makefile qui compile en utilisant les flags appropriés. Il y a au moins un main pour tester l'exercice.

☐ Oui ☐ Non

Notre chère Karen

Il existe une classe Karen avec au moins les 5 fonctions requises dans le sujet. La fonction complain() exécute les autres fonctions en utilisant un pointeur vers elles.
Idéalement, l'étudiant aurait dû mettre en œuvre un moyen de faire correspondre les différentes chaînes de caractères correspondant au niveau d'enregistrement aux pointeurs de la fonction membre correspondante.
Si l'implémentation est différente mais que l'exercice fonctionne, vous devez le marquer comme valide. La seule chose qui n'est pas autorisée est d'avoir un if/elseif/else.
L'élève aurait pu choisir de modifier le message affiché par Karen ou d'afficher les exemples donnés dans le sujet, les deux étant valables.

 Oui Non

Ex06 : Karen-filter

Maintenant que vous êtes des codeurs expérimentés, vous devriez utiliser de nouveaux types d'instructions, des instructions, des boucles, etc. Le but de ce dernier exercice est de vous faire découvrir l'instruction switch.

Makefile et tests

Il y a un Makefile qui compile en utilisant les flags appropriés. Il y a au moins un main pour tester l'exercice.

 Oui Non

Désactivation de Karen

Le programme karenFilter prend en argument n'importe quel niveau de journal ("DEBUG", "INFO", "WARNING" ou "ERROR"). Il devrait alors afficher uniquement les messages
qui sont au même niveau ou à un niveau supérieur (DEBUG < INFO < WARNING < ERROR). Cela doit être mis en œuvre à l'aide d'une instruction de commutation avec un cas par défaut.
Une fois de plus, plus de if/elseif/else s'il vous plaît.

 Oui Non

Notations

N'oubliez pas de vérifier le flag correspondant à la défense

 Ok

 Travail vide Travail incomplet W Compilation non valide Tricherie d Crash

 Fonction interdite

Conclusion

Laisser un commentaire sur cette évaluation

Fin de l'évaluation

Politique de confidentialité (https://signin.intra.42.fr/legal/terms/5)
Conditions d'utilisation de la vidéosurveillance
(https://signin.intra.42.fr/legal/terms/1) Règlement intérieur
(https://signin.intra.42.fr/legal/terms/4)
Déclaration sur l'utilisation des cookies (https://signin.intra.42.fr/legal/terms/2)
Conditions générales d'utilisation du site (https://signin.intra.42.fr/legal/terms/6)
Mentions légales (https://signin.intra.42.fr/legal/terms/3)