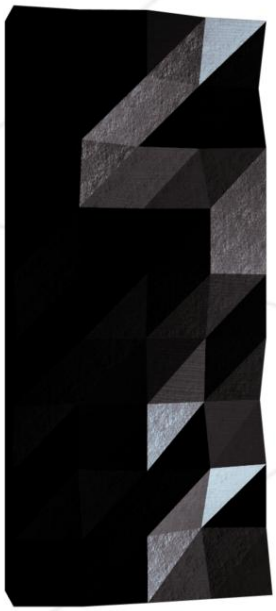

C++-Module 01

Allocation de mémoire, références, pointeurs vers les membres, commutateur

Résumé : Ce document contient le sujet du module 01 des modules C++.

Versionÿ: 8

Contenu

je Règles générales 2

II Exercice 00 : BraiiiiiiinnnzzzZ 4

III Exercice 01 : Moar brainz ! 5

IV Exercice 02 : SALUT C'EST LE CERVEAU 6

V Exercice 03 : Violence inutile sept

VI Exercice 04 : Sed est pour les losers 9

VII Exercice 05 : Karen 2.0 dix

VIII Exercice 06 : Filtre Karen 12

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 00 : BraiiiiiiinnnzzzZ

	Exercice : 00
BraiiiiiiinnnzzzZ	
Répertoire de rendu :	
ex00/ Fichiers à rendre : Makefile, main.cpp, Zombie.cpp, Zombie.hpp, newZombie.cpp, randomChump.cpp Fonctions interdites : Aucune	

Tout d'abord, créez une classe Zombie. Les zombies ont un nom privé et sont capables de s'annoncer comme :

<nom> BraiiiiiiinnnzzzZ...

Oui, annoncer(void) est une fonction membre. Ajoutez également un message de débogage dans le destructeur incluant le nom du Zombie.

Après cela, écrivez une fonction qui créera un zombie, nommez-le et renvoyez-le pour qu'il soit utilisé ailleurs dans votre code. Le prototype de la fonction est :

Zombie* newZombie(std::string name);

Vous devrez également écrire une autre fonction qui créera un zombie et le fera s'annoncer. Le prototype de la fonction est :

void randomChump(std::string name);

Maintenant, le vrai but de l'exercice : vos Zombies doivent être détruits au moment opportun (lorsqu'ils ne sont plus nécessaires). Ils doivent être alloués sur la pile ou le tas en fonction de son utilisationÿ: parfois, il est approprié de les avoir sur la pile, à d'autres moments, le tas peut être un meilleur choix.

Chapitre III

Exercice 01 : Moar brainz !

	Exercice : 01
Moar brainz!	
Répertoire de restitution :	
ex01/ Fichiers à rendre : Makefile, main.cpp, Zombie.cpp, Zombie.hpp, ZombieHorde.cpp	
Fonctions interdites : Aucune	

En réutilisant la classe Zombie, nous allons maintenant créer une horde de zombies !

Écrivez une fonction qui prend un entier N. Lorsqu'elle est appelée, elle alloue N objets Zombie. Il doit allouer tous les N objets Zombie en une seule allocation. Ensuite, il faut initialiser chaque Zombie en donnant à chacun un nom. Enfin, il devrait renvoyer le pointeur vers le premier zombie. La fonction est prototypée comme suitÿ:

Zombie* zombieHorde(int N, std::string name);

Soumettez un main pour tester que votre fonction zombieHorde fonctionne comme prévu. Vous voudrez peut-être le faire en appelant annoncer () sur chacun des zombies. N'oubliez pas de supprimer TOUS les zombies lorsque vous n'en avez plus besoin.

Chapitre IV

Exercice 02 : SALUT C'EST CERVEAU

	Exercice : 02
SALUT C'EST CERVEAU	
Répertoire de restitution : ex02/	
Fichiers à rendre : Makefile, main.cpp Fonctions	
interdites : Aucune	

Créez un programme dans lequel vous allez créer une chaîne contenant "HI THIS IS BRAIN".
Créez un stringPTR qui est un pointeur vers la chaîneÿ; et un stringREF qui est une référence à la chaîne.

Maintenant, affichez l'adresse en mémoire de la chaîne. Ensuite, affichez l'adresse du
chaîne en utilisant stringPTR et stringREF.

Enfin, affichez la chaîne à l'aide du pointeur, et enfin affichez-la à l'aide de la référence.

C'est tout, pas de trucs. Le but de cet exercice est de vous faire démystifier les références. Ce n'est pas quelque chose de complètement nouveau, c'est juste une autre syntaxe pour quelque chose que vous connaissez déjà : les adresses. Même il y a quelques détails minuscules-petits-minuscules.

Chapitre V

Exercice 03 : Violence inutile

	Exercice : 03
Violence inutile	
Répertoire de rendu :	
ex03/ Fichiers à rendre : Makefile, main.cpp, Weapon.cpp, Weapon.hpp, HumanA.cpp, HumanA.hpp, HumanB.cpp, HumanB.hpp Fonctions interdites : Aucune	

Créez une classe Weapon, qui a une chaîne de type et une méthode getType qui renvoie une référence const à cette chaîne. Il a aussi un setType, bien sûr !

Maintenant, créez deux classes, HumanA et HumanB, qui ont toutes deux une arme, un nom et une fonction attack() qui affiche :

NAME attaque avec son WEAPON_TYPE

HumanA et HumanB sont presque identiquesÿ; il n'y a que deux petits détails minusculesÿ:

- Alors que HumanA prend l'Arme dans son constructeur, HumanB ne le fait pas.
- HumanB n'a pas toujours d'Arme, mais HumanA sera TOUJOURS armé.

Faites en sorte que le code suivant produise des attaques avec "un club à pointes brutes" PUIS "certains autre type de club", dans les deux cas typesÿ:

```
int principal()
{
	{
		Arme		gourdin = Weapon(" gros gourdin à pointes");

		HumainA bob("Bob", massue);
		bob.attaque(); club.setType("un
		autre type de club"); bob.attaque();

	}
	{
		Arme		gourdin = Weapon(" gros gourdin à pointes");

		HumainB jim("Jim");
		jim.setWeapon(club);
		jim.attaque();
		club.setType("un autre type de club"); jim.attaque();

	}
}
```

Dans quel cas est-il approprié de stocker l'Arme sous forme de pointeurÿ? Et comme référence ?
Pourquoi?
Telles sont les questions que vous devez vous poser avant de vous lancer dans cet exercice.

Chapitre VI

Exercice 04 : Sed est pour les nuls

	Exercice : 04
Sed est pour les perdants	
Répertoire de rendu :	
ex04/ Fichiers à rendre : Makefile, main.cpp, et tout ce dont vous avez	
besoin Fonctions interdites : Aucune	

Créez un programme appelé replace qui prend un nom de fichier et deux chaînes, appelons-les s1 et s2, qui ne sont PAS vides.

Il ouvrira le fichier et écrira son contenu dans FILENAME.replace, après avoir remplacé chaque occurrence de s1 par s2.

Toutes les fonctions membres de la classe std::string sont autorisées, sauf replace. Utilisez-les à bon escient!

Bien sûr, vous gérerez les erreurs du mieux que vous pourrez. Ne pas utiliser la manipulation de fichier C fonctions, parce que ce serait tricher, et tricher c'est mal, d'accordÿ?

Vous rendrez quelques fichiers de test pour montrer que votre programme fonctionne.

Chapitre VII

Exercice 05 : Karen 2.0

	Exercice : 05
Karine 2.0	
Répertoire de restitution :	
ex05/ Fichiers à rendre : Makefile, main.cpp, Karen.hpp, et Karen.cpp	
Fonctions interdites : Aucune	

Connaissez-vous Karine ? Nous le faisons tous, nonÿ? Au cas où vous ne le feriez pas, voici le genre de commentaires que Karen faitÿ:

- Niveau "DEBUG" : les messages de ce niveau contiennent des informations contextuelles détaillées. Ils sont principalement utilisés pour le diagnostic des problèmes. Exempleÿ: "J'adore avoir du bacon supplémentaire pour mon burger 7XL-double-fromage-triple-cornichon-spécial-ketchup. J'adore çaÿ!"

- Niveau "INFO" : Ces messages contiennent des informations contextuelles pour aider à tracer l'exécution dans un environnement de production. Exempleÿ: "Je ne peux pas croire que l'ajout de bacon supplémentaire coûte plus cher. Vous n'en mettez pas assezÿ! Si vous le faisiez, je n'aurais pas à le demanderÿ!"

- Niveau "WARNING" : Un message d'avertissement indique un problème potentiel dans le système. Le système est capable de gérer le problème par lui-même ou de traiter ce problème de toute façon. Exempleÿ: "Je pense que je mérite d'avoir du bacon supplémentaire gratuitement. Je viens ici depuis des années et vous venez de commencer à travailler ici le mois dernier."

- Niveau « ERREUR » : Un message d'erreur indique un problème sérieux dans le système. Le problème est généralement irrécupérable et nécessite une intervention manuelle. Exempleÿ: "C'est inacceptable, je veux parler au responsable maintenant."

On va automatiser Karen, elle dit toujours les mêmes choses. Vous devez créer une classe nommée Karen qui contiendra les fonctions membres privées suivantesÿ:

- void debug(void);
- annuler les informations(void);
- avertissement d'annulation(void);
- erreur d'annulation(void);

Karen doit également avoir une fonction publique qui appelle les fonctions privées en fonction au niveau passé en paramètre. Le prototype de la fonction est :

void plaint(std::string level);

Le but de cet exercice est d'utiliser des pointeurs vers des fonctions membres. Ceci n'est pas une suggestion, Karen doit se plaindre sans utiliser une forêt de if/elseif/else, elle n'hésite pas et ne réfléchit pas à deux fois !

Soumettez une main pour tester que Karen se plaint beaucoup. C'est bien si vous voulez utiliser le plaintes que nous donnons en exemple.

Chapitre VIII

Exercice 06 : Filtre Karen

	Exercice : 06
Karen-filtre	
Répertoire de restitution :	
ex06/ Fichiers à rendre : Makefile, main.cpp, Karen.hpp, et Karen.cpp	
Fonctions interdites : Aucune	

Nous allons mettre en place un système pour filtrer si ce que dit Karen est important ou non, parce que parfois nous ne voulons pas prêter attention à tout ce que dit Karen.

Il faut écrire le programme karenFilter qui recevra en paramètre le niveau de log que l'on souhaite écouter et afficher toutes les infos qui se trouvent à ce niveau ou au-dessus. Par example:

$> ./karenFilter "ATTENTION"
[AVERTISSEMENT]
Je pense que je mérite d'avoir du bacon supplémentaire gratuitement.
Je viens ici depuis des années et tu viens de commencer à travailler ici le mois dernier.

[ERREUR]
C'est inacceptable, je veux parler au directeur maintenant.

$> ./karenFilter "Je ne sais pas à quel point je suis fatigué aujourd'hui..."
[Se plaindre probablement de problèmes insignifiants]

Il existe de nombreuses façons de filtrer Karen, mais l'une des meilleures consiste à la DÉSACTIVER ;)
Vous devez utiliser, et peut-être découvrir, l'instruction switch dans cet exercice.