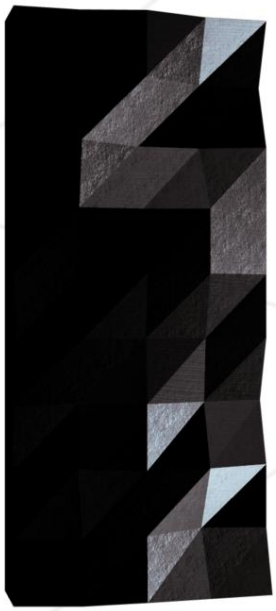

C++ - Module 01

Allocation de mémoire, pointeurs vers les membres, références et instructions switch

Résumé :
Ce document contient les exercices du Module 01 des modules C++.

Version : 10.1

Contenu

je	Introduction	2
II	Règles générales	3
III	Exercice 00 : Braa ...	6
IV	Exercice 01 : Plus de cerveaux !	7
	Exercice 02 : Salut, c'est le cerveau	8
VI	Exercice 03 : Violence inutile	9
VII	Exercice 04 : Sed est pour les perdants	11
VIII	Exercice 05 : Harl 2.0	12
IX	Exercice 06 : Filtre Harl	14
X	Soumission et évaluation par les pairs	15

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 initier à la programmation orientée objet.
Ce sera le point de départ de votre apprentissage du C++. De nombreux langages sont recommandés pour l'apprentissage de la POO. Nous avons choisi le C++ car il est dérivé de votre ancien ami, le C.
Parce qu'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 à bien des égards. Si vous souhaitez devenir un développeur C++ compétent, il vous appartient de poursuivre vos études après le Common Core 42 !

Chapitre II

Règles générales

Compilation

- Compilez votre code avec c++ et les indicateurs -Wall -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, ... , exn
- Nommez vos fichiers, classes, fonctions, fonctions membres et attributs comme requis dans les lignes directrices.
- Écrivez les noms de classe en majuscules (UpperCamelCase) . Les fichiers contenant du code de classe seront toujours nommés selon le nom de la classe. Par exemple : ClassName.hpp/ClassName.h, ClassName.cpp ou ClassName.tpp. Ainsi, si vous avez un fichier d'en-tête contenant la définition d'une classe « BrickWall » représentant un mur de briques, son nom sera BrickWall.hpp.
- Sauf indication contraire, chaque message de sortie doit se terminer par un caractère de nouvelle ligne et être affiché sur la sortie standard.
- Adieu Norminette ! Aucun style de codage n'est imposé dans les modules C++. Vous pouvez suivre votre style préféré. Mais gardez à l'esprit que le code que vos pairs ne comprennent pas est du code qu'ils ne peuvent pas noter. Faites de votre mieux pour écrire un code clair et lisible.

Autorisé/Interdit

Vous ne codez plus en C. Passez au C++ ! Par conséquent :

- Vous pouvez utiliser presque tout ce qui est proposé dans la bibliothèque standard. Ainsi, plutôt que 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 utiliser aucune autre bibliothèque externe. Cela signifie que C++11 (et ses dérivés) et les bibliothèques Boost sont interdits. Les fonctions suivantes sont également interdites : *printf(), *alloc() et free(). Si vous les utilisez, votre note sera de 0, point final.

- Notez que, sauf indication contraire explicite, l'espace de noms d'utilisation <ns_name> et Les mots-clés amis sont interdits. Sinon, votre note sera de -42.
- Vous êtes autorisé à utiliser le STL uniquement dans les modules 08 et 09. Cela signifie : pas de conteneurs (vecteur/liste/carte, etc.) ni d'algorithmes (tout ce qui nécessite l'inclusion de l'en-tête <algorithm>) jusqu'à cette date. 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 les fuites de mémoire.
- Du module 02 au module 09, vos cours doivent être conçus dans le style orthodoxe Forme canonique, sauf indication contraire explicite.
- 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. Ils doivent donc inclure toutes les dépendances nécessaires. Cependant, vous devez éviter le problème de double inclusion en ajoutant des gardes d'inclusion. Sinon, votre note sera de 0.

Lis-moi

- Vous pouvez ajouter des fichiers supplémentaires si nécessaire (par exemple, pour fractionner votre code). Ces devoirs n'étant pas vérifiés par un programme, n'hésitez pas à le faire, à condition de fournir les fichiers obligatoires.
- Parfois, les lignes directrices d'un exercice semblent courtes, mais les exemples peuvent montrer exigences qui ne sont pas explicitement écrites dans les instructions.
- Lisez chaque module en entier avant de commencer ! Vraiment, foncez.
- Par Odin, par Thor ! Utilisez votre cerveau !!!

Concernant le Makefile pour les projets C++, les mêmes règles qu'en C s'appliquent (voir le chapitre Norme sur le Makefile).

Vous devrez implémenter de nombreuses classes. Cela peut paraître fastidieux, à moins que vous ne puissiez écrire des scripts dans votre éditeur de texte préféré.

Vous disposez d'une certaine liberté pour réaliser les exercices.

Cependant, suivez les règles obligatoires et ne soyez pas paresseux. Vous
Vous manquez beaucoup d'informations utiles ! N'hésitez pas à lire

concepts théoriques.

Chapitre III

Exercice 00 : Braa ...

	Exercice : 00
Braa ...	
Répertoire de remise : ex00/	
Fichiers à rendre : Makefile, main.cpp, Zombie.{h, hpp}, Zombie.cpp, newZombie.cpp, randomChump.cpp Fonctions interdites :	
Aucune	

Tout d'abord, implémentez une classe Zombie . Son attribut est une chaîne privée.
Ajoutez une fonction membre void announce(void); à la classe Zombie.
s'annoncent comme suit :

<nom> : BraiiiiiiinnnzzzzZ...

N'imprimez pas les chevrons (< et >). Pour un zombie nommé Foo, le message serait :

Foo : BraiiiiiiinnnzzzzZ...

Ensuite, implémentez les deux fonctions suivantes :

- Zombie* newZombie(std::string name); Cette fonction crée un zombie, le nomme et le renvoie afin que vous puissiez l'utiliser en dehors de la portée de la fonction.

- void randomChump(std::string nom);
  Cette fonction crée un zombie, le nomme et le fait s'annoncer.

Quel est donc l'intérêt de l'exercice ? Il faut déterminer dans quel cas
il est préférable d'allouer les zombies sur la pile ou le tas.

Les zombies doivent être détruits dès qu'ils ne sont plus nécessaires. Le destructeur doit imprimer
un message avec le nom du zombie à des fins de débogage.

Chapitre IV

Exercice 01 : Plus de cerveaux !

	Exercice : 01
Plus de cerveaux !	
Répertoire de remise : ex01/	
Fichiers à rendre : Makefile, main.cpp, Zombie.{h, hpp}, Zombie.cpp, zombieHorde.cpp Fonctions interdites : Aucune	

Il est temps de créer une horde de zombies !

Implémentez la fonction suivante dans le fichier approprié :

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

Elle doit allouer N objets Zombies en une seule allocation. Elle doit ensuite initialiser les zombies en leur donnant à chacun le nom passé en paramètre. La fonction renvoie un pointeur vers le premier zombie.

Implémentez vos propres tests pour vous assurer que votre fonction zombieHorde() fonctionne comme prévu. attendu. Essayez d'appeler announce() pour chacun des zombies.

N'oubliez pas d'utiliser delete pour désallouer tous les zombies et vérifier les fuites de mémoire.

Chapitre V

Exercice 02 : Salut, c'est le cerveau

	Exercice : 02
Salut, c'est le cerveau	
Répertoire de remise : ex02/	
Fichiers à rendre : Makefile, main.cpp Fonctions	
interdites : Aucune	

Écrivez un programme qui contient :

• Une variable de chaîne initialisée à « HI THIS IS BRAIN ».

• stringPTR : un pointeur vers la chaîne.

• stringREF : une référence à la chaîne.

Votre programme doit imprimer :

• L'adresse mémoire de la variable chaîne.

• L'adresse mémoire détenue par stringPTR.

• L'adresse mémoire détenue par stringREF.

Et puis:

• La valeur de la variable chaîne.

• La valeur pointée par stringPTR.

• La valeur pointée par stringREF.

C'est tout, sans artifice. Le but de cet exercice est de démystifier les références, qui peuvent paraître complètement nouvelles. Malgré quelques petites différences, il s'agit simplement d'une autre syntaxe pour une pratique courante : la manipulation d'adresses.

Chapitre VI

Exercice 03 : Violence inutile

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

Implémentez une classe d'arme qui a :

- Un type d'attribut privé, qui est une chaîne.
- Une fonction membre getType() qui renvoie une référence constante au type.
- Une fonction membre setType() qui définit le type en utilisant la nouvelle valeur passée en tant que paramètre.

Créez maintenant deux classes : HumainA et HumainB. Elles possèdent toutes deux une arme et un nom. Elles possèdent également une fonction membre attack() qui affiche (sans les chevrons) :

<nom> attaque avec son <type d'arme>

HumanA et HumanB sont presque identiques à l'exception de ces deux petits détails :

- Alors que HumanA prend l'Arme dans son constructeur, HumanB ne le fait pas.
- L'humain B n'aura peut-être pas toujours d'arme, alors que l'humain A en aura toujours une armé.

Si votre implémentation est correcte, l'exécution du code suivant imprimera une attaque avec « club à pointes brut » suivie d'une deuxième attaque avec « un autre type de club » pour les deux cas de test :

```
int main() {

  {
    Club d'armes = Weapon(" club à pointes brut");

    HumanA bob("Bob", club);
    bob.attack();
    club.setType("un autre type de club"); bob.attack(); } {



    Club d'armes = Weapon(" club à pointes brut");

    HumanB jim("Jim");
    jim.setWeapon(club);
    jim.attack();
    club.setType("un autre type de club"); jim.attack();

  }

  retourner 0;
}
```

N'oubliez pas de vérifier les fuites de mémoire.

Dans quel cas pensez-vous qu'il serait préférable d'utiliser un pointeur vers Weapon ? Et une référence à Weapon ? Pourquoi ? Réfléchissez-y avant de commencer cet exercice.

Chapitre VII

Exercice 04 : Sed est pour les perdants

	Exercice : 04
Sed est pour les perdants	
Répertoire de remise :	
ex04/ Fichiers à remettre : Makefile, main.cpp, *.cpp, *.{h, hpp}	
Fonctions interdites : std::string::replace	

Créez un programme qui prend trois paramètres dans l'ordre suivant : un nom de fichier et deux cordes, s1 et s2.

Il doit ouvrir le fichier <nom_fichier> et copier son contenu dans un nouveau fichier <filename>.replace, remplaçant chaque occurrence de s1 par s2.

L'utilisation de fonctions de manipulation de fichiers C est interdite et serait considérée comme de la triche. Toutes les fonctions membres de la classe std::string sont autorisées, sauf replace. Utilisez-les avec prudence !

Bien sûr, gérez les saisies et erreurs inattendues. Vous devez créer et rendre votre vos propres tests pour vous assurer que votre programme fonctionne comme prévu.

Chapitre VIII

Exercice 05 : Harl 2.0

	Exercice : 05
Harl 2.0	
Répertoire de remise : ex05/	
Fichiers à rendre : Makefile, main.cpp, Harl.{h, hpp}, Harl.cpp Fonctions interdites : Aucune	

Connaissez-vous Harl ? Nous le connaissons tous, n'est-ce pas ? Si ce n'est pas le cas, découvrez ci-dessous les commentaires qu'il publie. Ils sont classés par niveaux :

- Niveau « DEBUG » : les messages de débogage contiennent des informations contextuelles. Ils sont principalement utilisés pour diagnostiquer les problèmes.
  Exemple : « J'adore avoir du bacon supplémentaire pour mon burger 7XL-double-fromage-triple-cornichons-ketchup spécial. Vraiment ! »

- Niveau « INFO » : Ces messages contiennent des informations détaillées. Ils sont utiles pour suivi de l'exécution du programme dans un environnement de production.
  Exemple : « Je n'arrive pas à croire que l'ajout de bacon coûte plus cher. Vous n'avez pas mis assez de bacon dans mon burger ! Si vous en aviez mis, je n'en demanderais pas plus ! »

- Niveau « AVERTISSEMENT » : les messages d'avertissement indiquent un problème potentiel dans le système. Cependant, cela peut être géré ou ignoré.
  Exemple : « Je pense que je mérite d'avoir du bacon supplémentaire gratuitement. Je viens depuis des années, alors que vous avez commencé à travailler ici le mois dernier. »

- Niveau « ERREUR » : Ces messages indiquent qu'une erreur irrécupérable s'est produite. Il s'agit généralement d'un problème critique qui nécessite une intervention manuelle.
  Exemple : « C'est inacceptable ! Je veux parler au directeur maintenant. »

Tu vas automatiser Harl. Ce ne sera pas difficile, car il dit toujours la même chose.
choses. Vous devez créer une classe Harl avec les fonctions membres privées suivantes :

- void debug(void);
- vide info(vide);
- avertissement nul(void);
- erreur vide(void);

Harl dispose également d'une fonction membre publique qui appelle les quatre fonctions membres ci-dessus en fonction du niveau passé en paramètre :

vide se plaindre(std::string niveau);

Le but de cet exercice est d'utiliser des pointeurs vers des fonctions membres. Ceci n'est pas une suggestion. Harl doit se plaindre sans utiliser une forêt de if/else if/else. Il n'y réfléchit pas à deux fois !

Créez et remettez des tests pour montrer que Harl se plaint beaucoup. Vous pouvez utiliser les exemples. des commentaires énumérés ci-dessus dans l'objet ou choisissez d'utiliser vos propres commentaires.

Chapitre IX

Exercice 06 : Filtre Harl

	Exercice : 06
Filtre Harl	
Répertoire de remise : ex06/	
Fichiers à rendre : Makefile, main.cpp, Harl.{h, hpp}, Harl.cpp Fonctions interdites : Aucune	

Parfois, vous ne voulez pas prêter attention à tout ce que dit Harl. Mettez en œuvre un système permettant de filtrer ce que dit Harl en fonction des niveaux de journal que vous souhaitez écouter.

Créez un programme prenant comme paramètre l'un des quatre niveaux. Il affichera tous les messages de ce niveau et des niveaux supérieurs. Par exemple :

```
> ./harlFilter "AVERTISSEMENT"
[ AVERTISSEMENT ]
Je pense que je mérite d'avoir du bacon supplémentaire gratuitement.
Je viens depuis des années, alors que vous avez commencé à travailler ici le mois dernier.

[ ERREUR ]
C'est inacceptable ! Je veux parler au directeur immédiatement.

$> ./harlFilter "Je ne sais pas à quel point je suis fatigué aujourd'hui..."
[ Probablement se plaignant de problèmes insignifiants ]
```

Bien qu'il existe plusieurs façons de gérer Harl, l'une des plus efficaces est de l'ÉTEINDRE.

Donnez le nom harlFilter à votre exécutable.

Vous devez utiliser, et peut-être découvrir, l'instruction switch dans cet exercice.

Vous pouvez réussir ce module sans faire l'exercice 06.

Chapitre X

Soumission et évaluation par les pairs

Remettez votre devoir dans votre dépôt Git comme d'habitude. Seul le travail effectué dans votre dépôt sera évalué 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$$4f1b9de5b5e60c03dcb4e8c7c7e4072c