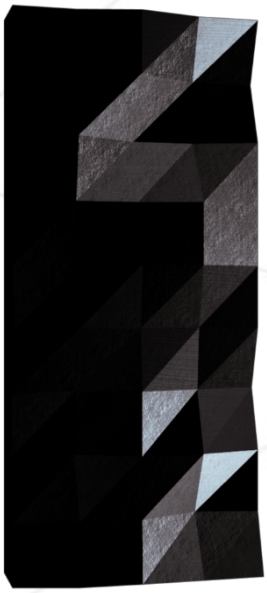

C++ - Module 01

Allocation de mémoire, pointeurs sur les membres, références, instructions de commutation

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

Version : 10

Contenu

Chapitre I

Introduction

Le C++ est un langage de programmation généraliste créé par Bjarne Stroustrup en tant qu'ex tension du langage de programmation C, ou "C avec des classes" (source : Wikipedia).

L'objectif de ces modules est de vous initier à la **programmation orientée objet**. Ce sera le point de départ de votre voyage en C++. De nombreux langages sont recommandés pour apprendre la POO. Nous avons décidé de choisir le C++ car il est dérivé de votre vieil ami le C. Comme 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 voulez devenir un développeur C++ compétent, il ne tient qu'à vous d'aller plus loin après le 42e tronc commun !

Chapitre II Règles générales

Compilation

- Compilez votre code avec c++ et les options -Wall -Wextra -Werror
- Votre code devrait toujours être compilé si vous ajoutez le drapeau -std=c++98

Conventions de formatage et de dénomination

- Les répertoires d'exercices seront nommés de la manière suivante : ex00, ex01, ... , exn
- Nommez vos fichiers, classes, fonctions, fonctions membres et attributs comme l'exigent les lignes directrices.
- Les noms des classes sont écrits en **majuscules**. Les fichiers contenant du code de classe seront toujours nommés en fonction du nom de la classe. Par exemple : Nomdeclasse.hpp/Nomdeclasse.h, Nomdeclasse.cpp, ou Nomdeclasse.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, tous les messages de sortie doivent être terminés par un caractère de nouvelle ligne et affichés sur la sortie standard.
- *Au revoir Norminette !* Aucun style de codage n'est imposé dans les modules C++. Vous pouvez suivre votre style préféré. Mais gardez à l'esprit qu'un code que vos pairs évaluateurs ne peuvent pas comprendre est un code qu'ils ne peuvent pas noter. Faites de votre mieux pour écrire un code propre et lisible.

Autorisé/interdit

Vous ne codez plus en C. Il est temps de passer au C++ ! C'est pourquoi :

- Vous êtes autorisé à utiliser presque tout ce qui se trouve dans la bibliothèque standard. Ainsi, au lieu 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 pas utiliser d'autres bibliothèques externes. Cela signifie que les bibliothèques C++11 (et formes dérivées) et Boost sont interdites. Les fonctions suivantes sont également interdites : *printf(), *alloc() et free(). Si vous les utilisez, votre note sera de 0 et c'est tout.

- Notez que, sauf indication contraire, les espaces de noms d'utilisation <ns_name> et les mots-clés amis sont interdits. Dans le cas contraire, votre note sera de -42.
- **Vous n'êtes autorisé à utiliser le STL que dans les modules 08 et 09.** Cela signifie : pas de **conteneurs** (vector/list/map/etc.) et pas d'**algorithmes** (tout ce qui nécessite d'inclure l'en-tête <algorithm>) jusqu'à cette date. Sinon, votre note sera de -42.

Quelques exigences en matière de conception

- Les fuites de mémoire se produisent également en C++. Lorsque vous allouez de la mémoire (en utilisant la fonction new
), vous devez éviter les **fuites de mémoire.**
- Du module 02 au module 09, vos cours doivent être conçus selon la **forme canonique orthodoxe, 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 dont ils ont besoin. Cependant, vous devez éviter le problème de la double inclusion en ajoutant des **gardes d'inclusion**. Dans le cas contraire, votre note sera de 0.

Lisez-moi

- Vous pouvez ajouter des fichiers supplémentaires si nécessaire (par exemple, pour diviser votre code). Comme ces travaux ne sont pas vérifiés par un programme, vous êtes libre de le faire tant que vous rendez les fichiers obligatoires.
- Parfois, les lignes directrices d'un exercice semblent courtes, mais les exemples peuvent montrer des exigences qui ne sont pas explicitement écrites dans les instructions.
- Lisez entièrement chaque module avant de commencer ! Vraiment, faites-le.
- Par Odin, par Thor ! Utilisez votre cerveau ! !!

En ce qui concerne le Makefile pour les projets C++, les mêmes règles que pour le C s'appliquent (voir le chapitre de Norm sur le Makefile).

Vous devrez implémenter un grand nombre de classes. Cela peut sembler fastidieux, à moins que vous ne soyez capable d'utiliser votre éditeur de texte favori pour créer des scripts.

Vous disposez d'une certaine liberté pour réaliser les exercices.
Toutefois, respectez les règles obligatoires et ne soyez pas paresseux.
Vous passeriez à côté d'un grand nombre d'informations utiles !
N'hésitez pas à vous documenter sur les concepts théoriques.

Chapitre III

Exercice 00 : BraiiiiiiinnnzzzZ

	Exercice : 00
BraiiiiiiinnnzzzZ	
Annuaire de retournement : *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**. Elle possède un attribut privé string name.
Ajoutez une fonction membre void announce(void) ; à la classe Zombie. Les zombies s'annoncent comme suit :

<nom> : BraiiiiiiinnnzzzZ...

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

Foo : BraiiiiiiinnnzzzZ...

Ensuite, mettez en œuvre les deux fonctions suivantes :

- Zombie* newZombie(std::string name) ;
  Elle crée un zombie, lui donne un nom et le renvoie afin que vous puissiez l'utiliser en dehors de la portée de la fonction.
- void randomChump(std::string name) ;
  Il crée un zombie, donne un nom et le zombie s'annonce.

Maintenant, quel est le but de l'exercice ? Vous devez déterminer dans quel cas il est préférable d'allouer les zombies sur la pile ou sur le tas.

Les zombies doivent être détruits lorsque vous ' avez plus besoin. Le destructeur doit afficher 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 retournement : *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 name ) ;
```

Il doit allouer N objets Zombie en une seule fois. Ensuite, elle doit initialiser les zombies en donnant à chacun d'eux le nom passé en paramètre. La fonction renvoie un pointeur sur le premier zombie.

Mettez en place vos propres tests pour vous assurer que votre fonction zombieHorde() fonctionne comme prévu.

Essayez d'appeler announce() pour chacun des zombies.

N'oubliez pas de supprimer tous les zombies et de vérifier qu'il n'y a pas **de fuites de mémoire**.

Chapitre V

Exercice 02 : HI THIS IS BRAIN

	Exercice : 02
BONJOUR, JE M'APPELLE BRAIN	
Annuaire de retournement : *ex02/*	
Fichiers à rendre : `Makefile, main.cpp`	
Fonctions interdites : `Aucune`	

Écrire un programme qui contient :

- Une variable de type chaîne de caractères initialisée à "HI THIS IS BRAIN".
- stringPTR : pointeur sur la chaîne de caractères.
- stringREF : Une référence à la chaîne de caractères.

Votre programme doit être imprimé :

- L'adresse mémoire de la variable chaîne.
- L'adresse mémoire détenue par stringPTR.
- L'adresse mémoire détenue par stringREF. Et

ensuite :

- La valeur de la variable chaîne.
- La valeur indiquée par stringPTR.
- La valeur indiquée par stringREF.

C'est tout, pas d'astuces. Le but de cet exercice est de démystifier les références qui peuvent sembler complètement nouvelles. Bien qu'il y ait quelques petites différences, il s'agit d'une autre syntaxe pour quelque chose que vous faites déjà : la manipulation d'adresses.

Chapitre VI

Exercice 03 : violence inutile

	Exercice : 03
Violence inutile	
Annuaire de retournement : *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`	

Implémenter une classe d'arme qui a :

- Un type d'attribut privé, qui est une chaîne de caractères.
- Une fonction membre getType() qui renvoie une référence constante au type.
- Une fonction membre setType() qui définit le type en utilisant le nouveau type transmis en tant que paramétre.

Créez maintenant deux classes : **HumanA** et **HumanB**. Elles ont toutes deux une arme et un nom. Elles ont également une fonction membre attack() qui s'affiche (bien sûr, sans les crochets) :

<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'homme B n'**a pas toujours** d'arme, alors que l'homme A est **toujours** armé.

Si votre implémentation est correcte, l'exécution du code suivant imprimera une attaque avec "une massue à pointes brutes" puis une seconde attaque avec "un autre type de massue" pour les deux cas de test :

```
int main()
{
  {
    Arme massue= Arme("massue grossière à pointes") ;

    HumanA bob("Bob", club) ;
    bob.attack() ;
    club.setType("un autre type de club") ; bob.attack() ;
  }
  {
    Arme massue= Arme("massue grossière à pointes") ;

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

  retour 0 ;
}
```

N'oubliez pas de vérifier s'il y a **des fuites de mémoire**.

Dans ce cas, pensez-vous qu'il serait préférable d'utiliser un pointeur sur l'arme ? Et une référence à l'arme ? Pourquoi ? Réfléchissez-y avant de commencer cet exercice.

Chapitre VII

Exercice 04 : Sed est pour les perdants

	Exercice : 04
Le Sed est pour les perdants	
Annuaire de retournement : *ex04/*	
Fichiers à rendre : `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 chaînes de caractères, s1 et s2.

Il ouvre le fichier <nom du fichier> et copie son contenu dans un nouveau fichier. <fichier>.replace, remplaçant chaque occurrence de s1 par s2.

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

Bien entendu, il faut gérer les entrées et les erreurs inattendues. Vous devez créer et présenter vos propres tests pour vous assurer que votre programme fonctionne comme prévu.

Chapitre VIII

Exercice 05 : Harl 2.0

	Exercice : 05
Harl 2.0	
Annuaire de retournement : *ex05/*	
Fichiers à rendre : `Makefile, main.cpp, Harl.{h, hpp}, Harl.cpp`	
Fonctions interdites : `Aucune`	

Vous connaissez Harl ? le connaissons tous, n'est-ce pas ? Si ce n'est pas le cas, vous trouverez ci-dessous le genre de commentaires que fait Harl. Ils sont classés par niveau :

- Niveau "**DEBUG**" : Les messages de débogage contiennent des informations contextuelles. Ils sont principalement utilisés pour le diagnostic des problèmes.
  Exemple : *"J'aime avoir du bacon supplémentaire pour mon hamburger 7XL double fromage-triple cornichon-spécial ketchup. J'aime vraiment ça !"*

- Niveau "**INFO**" : Ces messages contiennent des informations détaillées. Ils sont utiles pour retracer l'exécution d'un programme dans un environnement de production.
  Exemple : *"Je ne peux pas croire que l'ajout de bacon coûte plus cher. Vous n'avez pas mis assez de bacon dans mon hamburger ! Si c'était le cas, je n'en demanderais pas plus !".*

- Niveau "**AVERTISSEMENT**" : Les messages d'avertissement indiquent un problème potentiel dans le système. Il peut toutefois être traité ou ignoré.
  Exemple : *"Je pense que je mérite d'avoir du bacon supplémentaire gratuitement. Je viens depuis des années alors que tu n'as commencé à travailler ici que 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".*

Vous allez automatiser Harl. Ce ne sera pas difficile puisqu'il dit toujours les mêmes choses. Vous devez créer une classe **Harl** avec les fonctions membres privées suivantes :

- void debug(void) ;
- void info(void) ;
- void warning(void) ;
- void error(void) ;

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

```
void    complain( std::string level ) ;
```

Le but de cet exercice est d'utiliser des **pointeurs sur les fonctions membres**. Il ne s'agit pas d'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 rendez des tests pour montrer que Harl se plaint beaucoup. Vous pouvez utiliser les exemples de commentaires énumérés ci-dessus dans le sujet ou choisir d'utiliser vos propres commentaires.

Chapitre IX

Exercice 06 : Filtre de Harl

	Exercice : 06
Filtre Harl	
Annuaire de retournement : *ex06/*	
Fichiers à rendre : `Makefile, main.cpp, Harl.{h, hpp}, Harl.cpp`	
Fonctions interdites : `Aucune`	

Parfois, vous voulez pas prêter attention à tout ce que dit Harl. Mettez en place un système permettant de filtrer ce que dit Harl en fonction des niveaux d'enregistrement que vous souhaitez écouter.

Créez un programme qui prend comme paramètre l'un des quatre niveaux. Il affichera tous les messages à partir de ce niveau. Il affichera tous les messages à partir de ce niveau :

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

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

$> ./harlFilter "Je ne suis pas sûr d'être fatigué
aujourd'hui..." [ Probablement en se plaignant de problèmes
insignifiants ]
```

Bien qu'il y ait plusieurs façons de faire face à Harl, l'une des plus efficaces est de léteindre.

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

Rendez votre travail dans votre dépôt Git comme d'habitude. Seul le travail contenu 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
```