



C++ - Module 01

Allocation de mémoire, pointeurs vers les membres,
références, instruction switch

Résumé:

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

Version: 10

Contenu

je	Introduction	2
II	Règles générales	3
III	Exercice 00 : CerviiiiiiiiinnnzzzzZ	6
IV	Exercice 01 : Plus de cerveaux !	7
V	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 présenter **Programmation orientée objet**. Ce sera le point de départ de votre parcours C++. De nombreux langages sont recommandés pour apprendre la programmation orientée objet. Nous avons décidé de choisir C++ car il est dérivé de votre vieil ami 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 sur de nombreux aspects. Donc si vous voulez devenir un développeur C++ compétent, c'est à vous d'aller plus loin après le 42 Common Core !

Chapitre II

Règles générales

Compilation

- Compilez votre code avec `g++` et les drapeaux `-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`, ... , par exemple
- Nommez vos fichiers, classes, fonctions, fonctions membres et attributs comme requis dans les directives.
- Écrivez les noms des classes dans **UpperCamelCase** format. Les fichiers contenant le code de classe seront toujours nommés selon le nom de la classe. Par exemple : `NomDeClasse.hpp`/`NomDeClasse.h`, `NomDeClasse.cpp`, ou `Nom de classe.hpp`. Ensuite, si vous avez un fichier d'en-tête contenant la définition d'une classe « `BrickWall` » représentant un mur de briques, son nom sera `Mur de briques.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 Nominette !* 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++ ! Par conséquent :

- 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 C++11 (et les formes dérivées) et Boost. Les bibliothèques sont interdites. Les fonctions suivantes sont également interdites : `*printf()`, `*alloc()` et `getgratuit()`. Si vous les utilisez, votre note sera de 0 et c'est tout.

- Veuillez noter que, sauf indication contraire explicite, le `using` n'est pas autorisé. Les noms de namespace `<ns_name>` et les mots-clés sont interdits. Sinon, votre note sera de -42.
- **Vous êtes autorisé à utiliser le STL dans les modules 08 et 09 uniquement.** Cela signifie : non **Conteneurs** (vecteur/liste/carte/et ainsi de suite) et non **Algorithmes** (tout ce qui nécessite d'inclure le `<algorithm>` (en-tête) jusqu'à ce moment-là. 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 **fuites de mémoire**.
- Du module 02 au module 09, vos cours doivent être conçus dans le **Forme canonique orthodoxe, sauf indication explicite contraire**.
- 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. Ainsi, ils doivent inclure toutes les dépendances dont ils ont besoin. Cependant, vous devez éviter le problème de double inclusion en ajoutant **inclusion des gardes**. Sinon, votre note sera de 0.

Lis-moi

- Vous pouvez ajouter des fichiers supplémentaires si vous en avez besoin (par exemple pour diviser votre code). Comme ces tâches ne sont pas vérifiées par un programme, n'hésitez pas à le faire à condition de rendre les fichiers obligatoires.
- Parfois, les directives d'un exercice semblent courtes, mais les exemples peuvent montrer des exigences qui ne sont pas explicitement écrites dans les instructions.
- Lisez chaque module dans son intégralité avant de commencer ! Vraiment, faites-le.
- Par Odin, par Thor ! Utilise ton 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 soyez capable de créer un script dans votre éditeur de texte préféré.



Vous disposez d'une certaine liberté pour réaliser les exercices. Cependant, respectez les règles obligatoires et ne soyez pas paresseux. Vous passeriez à côté de beaucoup d'informations utiles ! N'hésitez pas à lire les notions théoriques.

Chapitre III

Exercice 00 : CerviiiiinnnnzzzzZ

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

Tout d'abord, implémentez un **Zombie** classe. Il a un attribut privé de chaîne nom. Ajouter une fonction membre void annoncer(void); à la classe Zombie. Les zombies s'annoncent comme suit :

<nom>: BraiiiiiiinnnnzzzzZ...

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

Foo : BraiiiiiiinnnnzzzzZ...

Ensuite, implémentez les deux fonctions suivantes :


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

Maintenant, quel est le véritable intérêt de l'exercice ? Il faut déterminer dans quel cas il est préférable de répartir les zombies sur la pile ou le tas.

Les zombies doivent être détruits lorsque vous n'en 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 remise : ex01/	
Dossiers à rendre : Makefile, main.cpp, Zombie.{h, hpp}, Zombie.cpp, zombieHorde.cpp	
Fonctions interdites : Aucun	

Il est temps de créer un **horde de zombies**!

Implémentez la fonction suivante dans le fichier approprié :

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


Il doit allouer N Les objets zombies sont répartis en une seule allocation. Il faut ensuite initialiser les zombies en donnant à chacun d'eux le nom passé en paramètre. La fonction retourne un pointeur vers le premier zombie.

Implémentez vos propres tests pour garantir votre Horde de zombies() la fonction fonctionne comme prévu. Essayez d'appeler annoncer() pour chacun des zombies.

N'oubliez pas de supprimer tous les zombies et vérifiez **fuites de mémoire**.

Chapitre V

Exercice 02 : Salut, c'est le cerveau

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

Écrivez un programme qui contient :

- Une variable de chaîne initialisée à « HI THIS IS BRAIN ».
- chaînePTR : Un pointeur vers la chaîne.
- chaîneREF : Une référence à la chaîne.

Votre programme doit imprimer :

- L'adresse mémoire de la variable chaîne.
- L'adresse mémoire détenue par chaînePTR.
- L'adresse mémoire détenue par chaîneREF.


Et puis:

- La valeur de la variable chaîne.
- La valeur pointée par chaînePTR.
- La valeur pointée par chaîneREF.

C'est tout, pas de trucage. Le but de cet exercice est de démystifier des 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	
Répertoire de remise : ex03/	
Dossiers à rendre : Makefile, main.cpp, Arme.{h, hpp}, Arme.cpp, HumainA.{h, hpp}, HumainA.cpp, HumainB.{h, hpp}, HumainB.cpp	
Fonctions interdites : Aucun	

Implémentez une classe d'arme qui a :

- Un attribut privé `taper`, qui est une chaîne.
- Une fonction membre `obtenirType()` qui renvoie une référence const à `taper`.
- Une fonction membre `définirType()` qui définit `taper` en utilisant le nouveau passé en paramètre.

Maintenant, créez deux classes : **HumainA** et **HumainB**. Ils ont tous les deux un `Arme` et un `nom`. Ils ont également une fonction membre `attaque()` qui affiche (bien sûr, 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.
- HumanB peut **pas toujours** avoir une arme, alors que HumanA aura **toujours** être armé.

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

```
int principal()
{
    {
        Arme club = Arme("club à pointes brut");

        HumainUn bob("Bob", club);
        bob.attaquer();
        club.setType("un autre type de club");
        bob.attaquer();
    }
    {
        Club d'armes = Arme("club à pointes brut");

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

return 0;
```


N'oubliez pas de vérifier **fuites de mémoire**.



Dans quel cas pensez-vous qu'il serait préférable d'utiliser un pointeur vers Weapon ? Et une référence vers 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/	
Dossiers à rendre : Makefile, main.cpp, *.cpp, *.{h, hpp}	
Fonctions interdites : std::string::remplacer	

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


Cela ouvrira le fichier `<nom de fichier>` et copie son contenu dans un nouveau fichier `<nom de fichier>.replace`, en remplaçant chaque occurrence de `s1` avec `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 autorisés, sauf `remplacer`. Utilisez-les à bon escient !

Bien sûr, gérez les entrées et les erreurs inattendues. Vous devez créer et rendre 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/
	Dossiers à rendre : Makefile, main.cpp, Harl.{h, hpp}, Harl.cpp
	Fonctions interdites : Aucun

Connaissez-vous Harl ? Nous le connaissons tous, n'est-ce pas ? Au cas où vous ne le sauriez pas, vous trouverez ci-dessous le genre de commentaires que fait Harl. Ils sont classés par niveaux :

- **"DÉBOGUER"** niveau : les messages de débogage contiennent des informations contextuelles. Ils sont principalement utilisés pour le diagnostic des problèmes.
Exemple: *"J'adore avoir du bacon supplémentaire pour mon burger 7XL double fromage triple cornichon ketchup spécial. Vraiment !"*
- **"INFO"** niveau : Ces messages contiennent des informations détaillées. Ils sont utiles pour suivre l'exécution d'un programme dans un environnement de production.
Exemple: *"Je ne peux pas croire que l'ajout de bacon supplémentaire 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 !"*
- **"AVERTISSEMENT"** niveau : les messages d'avertissement indiquent un problème potentiel dans le système. Cependant, il peut être traité ou ignoré.
Exemple: *"Je pense que je mérite d'avoir un peu de bacon supplémentaire gratuitement. Je viens depuis des années alors que tu as commencé à travailler ici le mois dernier."*
- **"ERREUR"** niveau : 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 car il dit toujours les mêmes choses. Vous devez créer un **Harl** classe avec les fonctions membres privées suivantes :

- vide débogage(vide);
- vide info(vide);
- avertissement vide(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 **pointeurs vers des fonctions membres**. Ce 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 de commentaires énumérés ci-dessus dans l'objet ou choisir d'utiliser vos propres commentaires.

Chapitre IX

Exercice 06 : Filtre Harl

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

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

Créez un programme qui prend 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 tu as commencé à travailler ici depuis le mois dernier.

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

> ./harlFilter "Je ne sais pas à quel point je suis fatigué aujourd'hui..." [Je
me plains probablement 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 votreGitcomme d'habitude. Seuls les travaux contenus dans votre dépôt seront évalués lors de la soutenance. N'hésitez pas à vérifier les noms de vos dossiers et fichiers pour vous assurer qu'ils sont corrects.



????????????? XXXXXXXXXXXX = \$3\$4f1b9de5b5e60c03dcb4e8c7c7e4072c