



# C++ - Module 01

Allocation de mémoire, pointeurs vers des  
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 : CerviiiiiiiiinnzzzzZ	5
IV	Exercice 01 : Plus de cerveaux !	6
V	Exercice 02 : Salut, c'est le cerveau	7
VI	Exercice 03 : Violence inutile	8
VII	Exercice 04 : Sed est pour les perdants	10
VIII	Exercice 05 : Harl 2.0	11
IX	Exercice 06 : Filtre Harl	13
X	Soumission et évaluation par les pairs	14

# 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 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. 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 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 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 ainsi : ex00, ex01, ... , exn
- Nommez vos fichiers, classes, fonctions, fonctions membres et attributs comme requis dans les lignes directrices.
- Écrivez les noms de classe au format UpperCamelCase . Les fichiers contenant le code de classe doit toujours être nommé en fonction du 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, tous les messages de sortie doivent être terminés par une nouvelle ligne caractère et affiché 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++ ! 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 utiliser aucune autre bibliothèque externe. Cela signifie que les bibliothèques C++11 (et les 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 explicite contraire, 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.) et pas d'algorithmes (tout ce qui nécessite d'inclure l'en-tête <algorithm>) 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 les fuites de mémoire.
- Du Module 02 au Module 09, vos cours doivent être conçus dans le style orthodoxe  
Forme canonique, 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 des gardes d'inclusion. 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 le montrer.  
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 !!!




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

# Chapitre III

## Exercice 00 : CerviiiiinnnnzzzzZ

	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`. Elle possède un nom d'attribut privé de type chaîne. Ajoutez une fonction membre `void Announce( void );` à la classe `Zombie`. `Zombies` s'annoncent comme suit :

<nom>: BraiiiiinnnnzzzzZ...

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

`Foo` : BraiiiiinnnnzzzzZ...

Ensuite, implémentez les deux fonctions suivantes :


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

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


Il faut allouer N objets zombies 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 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 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, 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/	
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 const au type.
- Une fonction membre setType() qui définit le type en utilisant le nouveau type passé en tant que paramètre.éter.

Maintenant, créez deux classes : HumanA et HumanB. Elles ont toutes deux une arme et un nom. Elles ont également une fonction membre attack() qui affiche (bien sûr, sans les crochets angulaires) :

<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'a peut-être pas toujours d'arme, alors que l'Humain A sera toujours 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 main() {  
  
    {  
        Club d'armes = Arme(" club à pointes brut");  
  
        HumanA bob("Bob", club);  
        bob.attack();  
        club.setType("un autre type de club"); bob.attack(); } {  
  
        Club d'armes = Arme(" 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 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/ 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 cordes, s1 et s2.


Il ouvrira le fichier <nom de fichier> et copiera 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 sera considérée comme de la triche. Toutes les fonctions membres de la classe std::string sont autorisées, à l'exception de replace. Utilisez-les à bon escient !

Bien sûr, gérez les entrées et les erreurs inattendues. Vous devez créer et remettre 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, cpp}, Harl.cpp Fonctions interdites : Aucune	

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 :

- Niveau « DEBUG » : 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 hamburger 7XL-double-fromage-triple-cornichon-ketchup spécial. Vraiment ! »
- Niveau « INFO » : Ces messages contiennent des informations détaillées. Ils sont utiles pour traçage de l'exécution du 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 assez, 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 tu as 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 responsable maintenant. »

Vous allez 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 void( 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. 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  
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 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..."
[ Se plaignant 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 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.



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