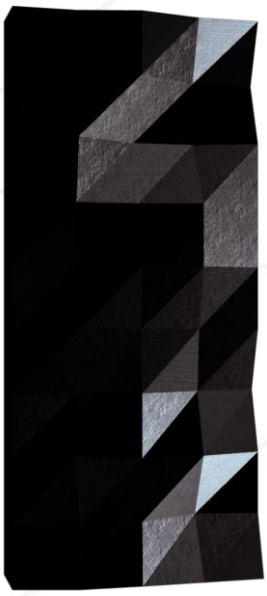

# C++ - Module 01

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

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

*Version : 9*

# 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 classes" (source : Wikipedia).*

L'objectif de ces modules est de vous présenter 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 C++ puisqu'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 au standard C++98.

Nous sommes conscients que le C++ moderne est très différent dans de nombreux aspects. Donc, si vous voulez devenir un développeur C++ compétent, c'est à vous d'aller plus loin après les 42 Troncs communs !

# Chapitre II

# Règles générales

**Compiler**

- Compilez votre code avec c++ et les drapeaux -Wall -Wextra -Werror
- Votre code devrait toujours compiler si vous ajoutez l'option -std=c++98.

**Conventions de formatage et d'appellation**

- Les répertoires d'exercices seront nommés de la façon suivante : ex00, ex01, . ,exn
- Nommez vos fichiers, classes, fonctions, fonctions membres et attributs comme l'exigent les directives.
- Écrivez les noms de classe au format **UpperCamelCase**. Les fichiers contenant du code de classe seront toujours nommés 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 un caractère de nouvelle ligne et affichés 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 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 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 explicite, l'espace de noms d'utilisation <ns_name> et l'espace de noms d'utilisation <ns_name> sont utilisés. les mots-clés amis sont interdits. Sinon, votre note sera de -42.
- **Vous êtes autorisé à utiliser la STL uniquement dans le module 08.** Cela signifie : pas de **Containers** (vector/list/map/etc.) et pas d'**Algorithmes** (tout ce qui nécessite d'inclure l'en-tête <algorithm>) jusque 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 la fonction new ), vous devez éviter les **fuites de mémoire**.
- Du module 02 au module 08, vos cours doivent être conçus selon la **forme canonique orthodoxe, sauf mention contraire explicite**.
- Toute implémentation de fonction mise dans un fichier d'en-tête (à l'exception des modèles de fonction) signifie 0 pour l'exercice.
- Vous devez être en mesure d'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 la double inclusion en ajoutant des **gardes d'inclusion**. Dans le cas contraire, votre note sera de 0.

**Lisez-moi**

- Vous pouvez ajouter quelques 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 directives 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 ! Utilise ton cerveau ! !!

Vous devrez implémenter un grand nombre de classes.Celapeut sembler fastidieux, à moins que vous ne soyez capable d'utiliser 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 manqueriez beaucoup d' informationsutiles !N'hésitez pas à vous documenter sur les concepts théoriques.

# Chapitre III

# Exercice 00 : BraiiiiiiinnnzzzZ

|  | Exercice : 00 |
|---|---|
| BraiiiiiiinnnzzzZ | |
| Répertoire d'entrée : *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. Les zombies s'annoncent comme suit :

<nom> : BraiiiiiiinnnzzzZ...

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

Foo : BraiiiiiiinnnzzzZ...

Ensuite, mettez en œuvre les deux fonctions suivantes :

- Zombie* newZombie( std::string name ) ;
  Elle 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 name ) ;
  Il crée un zombie, le nomme, et le zombie s'annonce.

Maintenant, quel est le but réel de l'exercice ? Vous devez déterminer dans quel cas il est préférable d'allouer 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 pour des raisons de débogage.

# Chapitre IV

# Exercice 01 : Moar brainz !

|  | Exercice : 01 |
|---|---|
| Moar brainz ! | |
| Répertoire d'entrée : *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'entre eux le nom passé en paramètre. La fonction retourne 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 les **fuites de mémoire**.

# Chapitre V

# Exercice 02 : C'EST LE CERVEAU

|  | Exercice : 02 |
|---|---|
| BONJOUR, C'EST BRAIN. | |
| Répertoire d'entrée : | |
| *ex02/* | |
| Fichiers à rendre : Makefile, main.cpp | |

Fonctions interdites : Aucune

Ecrivez un programme qui contient :

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

chaîne de caractères. Votre

programme doit imprimer :

- 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 de caractères.
- La valeur pointée par stringPTR.
- La valeur pointée 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 : la violence inutile

|  | Exercice : 03 |
|---|---|
| Violence inutile | |
| Répertoire d'entrée : *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 comme paramètre.

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) :

<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'humainB peut **ne pas toujours** avoir une arme, alors que l'humainA sera **toujours** armé.

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

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

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.

```
    Arme massue = Arme("massue à pointes grossières") ;

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

  retourner 0 ;
}
```

# Chapitre VII

# Exercice 04 : Sed est pour les perdants

|  | Exercice : 04 |
|---|---|
| Sed est pour les perdants | |
| Répertoire d'entrée : *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.

Elle ouvrira le fichier <nom du 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 en C est interdite et sera considérée comme une tricherie. Toutes les fonctions membres de la classe std::string sont autorisées, sauf replace. Utilisez-les à bon escient !

Bien sûr, il faut gérer 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 d'entrée : *ex05/* | |
| Fichiers à rendre : Makefile, main.cpp, Harl.{h, hpp}, Harl.cpp | |
| Fonctions interdites : Aucune | |

Tu connais Harl ? Nous le connaissons tous, n'est-ce pas ? Au cas où tu ne le saurais pas, tu trouveras ci-dessous le genre de commentaires que Harl fait. Ils sont classés par niveaux :

- Niveau "**DEBUG**" : Les messages de débogage contiennent des informations contextuelles. Ils sont surtout utilisés pour le diagnostic des problèmes.
  Exemple : *"J'adore avoir du bacon en plus pour mon burger 7XL-double-cheese-triple-pickle-special- ketchup. J'aime vraiment ça !"*

- Niveau "**INFO**" : 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 qu'ajouter du bacon supplémentaire 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 ne travailles ici que depuis le mois dernier."*

- niveau "**ERROR**" : 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 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** possède également une fonction membre publique qui appelle les quatre fonctions membres ci-dessus en fonction du niveau passé en paramètre :

```
voidcomplain(std::string level ) ;
```

Le but de cet exercice est d'utiliser des **pointeurs vers 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 ne réfléchit pas à deux fois !

Créer et rendre des tests pour montrer que Harl se plaint beaucoup. Vous pouvez utiliser les exemples de commentaires.

# Chapitre IX

# Exercice 06 : filtre Harl

|  | Exercice : 06 |
|---|---|
| Filtre Harl | |
| Répertoire d'entrée : *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 place un système pour filtrer ce que dit Harl en fonction des niveaux de log que vous voulez 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 "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 depuis le mois dernier.

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

$> ./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 faire face à 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 passer ce module sans faire l'exercice 06.