



## C++ - Module 01

Allocation mémoire, référence, pointeur membres,  
switch

*Résumé: Ce document contient le sujet du module 01 de la piscine C++ de 42*

# Table des matières

<b>I</b>	<b>Règles Générales</b>	<b>2</b>
<b>II</b>	<b>Exercice 00 : BraiiiiiiinnnzzzzZ</b>	<b>4</b>
<b>III</b>	<b>Exercice 01 : Encore plus de cerveaux !</b>	<b>5</b>
<b>IV</b>	<b>Exercice 02 : BONJOUR ICI LE CERVEAU</b>	<b>6</b>
<b>V</b>	<b>Exercice 03 : Violence inutile</b>	<b>7</b>
<b>VI</b>	<b>Exercice 04 : Sed, c'est pour les perdants</b>	<b>9</b>
<b>VII</b>	<b>Exercice 05 : Karen 2.0</b>	<b>10</b>
<b>VIII</b>	<b>Exercice 06 : Filtre de Karen</b>	<b>12</b>

# Chapitre I


## Règles Générales

- Toute fonction déclarée dans une header (sans pour les templates) ou tout header non-protégé, signifie 0 à l'exercice.
- Tout output doit être affiché sur stdout et terminé par une newline, sauf autre chose est précisé.
- Les noms de fichiers imposés doivent être suivis à la lettre, tout comme les noms de classe, les noms de fonction, et les noms de méthodes.
- Rappel : vous codez maintenant en C++, et plus en C. C'est pourquoi :
  - Les fonctions suivantes sont **INTERDITES**, et leur usage se soldera par un 0 : `*alloc`, `*printf` et `free`
  - Vous avez l'autorisation d'utiliser à peu près toute la librairie standard. CÉPENDANT, il serait intelligent d'essayer d'utiliser la version C++ de ce à quoi vous êtes habitués en C, plutôt que de vous reposer sur vos acquis. Et vous n'êtes pas autorisés à utiliser la STL jusqu'au moment où vous commencez à travailler dessus (module 08). Ça signifie pas de Vector/List/Map/etc... ou quoi que ce soit qui requiert une include `<algorithm>` jusque là.
- L'utilisation d'une fonction ou mécanique explicitement interdite sera sanctionnée par un 0
- Notez également que sauf si la consigne l'autorise, les mot-clés `using namespace` et `friend` sont interdits. Leur utilisation sera punie d'un 0.
- Les fichiers associés à une classe seront toujours nommés `ClassName.cpp` et `ClassName.hpp`, sauf si la consigne demande autre chose.
- Vous devez lire les exemples minutieusement. Ils peuvent contenir des prérequis qui ne sont pas précisés dans les consignes.
- Vous n'êtes pas autorisés à utiliser des librairies externes, incluant C++11, Boost, et tous les autres outils que votre ami super fort vous a recommandé
- Vous allez surement devoir rendre beaucoup de fichiers de classe, ce qui peut paraître répétitif jusqu'à ce que vous appreniez à scripter ça dans votre éditeur de code préféré.

- Lisez complètement chaque exercice avant de le commencer.
- Le compilateur est `clang++`
- Votre code sera compilé avec les flags `-Wall -Wextra -Werror`
- Chaque include doit pouvoir être incluse indépendamment des autres includes. Un include doit donc inclure toutes ses dépendances.
- Il n'y a pas de norme à respecter en C++. Vous pouvez utiliser le style que vous préférez. Cependant, un code illisible est un code que l'on ne peut pas noter.
- Important : vous ne serez pas noté par un programme (sauf si précisé dans le sujet). Cela signifie que vous avez un degré de liberté dans votre méthode de résolution des exercices.
- Faites attention aux contraintes, et ne soyez pas fainéant, vous pourriez manquer beaucoup de ce que les exercices ont à offrir
- Ce n'est pas un problème si vous avez des fichiers additionnels. Vous pouvez choisir de séparer votre code dans plus de fichiers que ce qui est demandé, tant qu'il n'y a pas de moulinette.
- Même si un sujet est court, cela vaut la peine de passer un peu de temps dessus afin d'être sûr que vous comprenez bien ce qui est attendu de vous, et que vous l'avez bien fait de la meilleure manière possible.

# Chapitre II

## Exercice 00 : BraiiiiiiinnnzzzzZ

	Exercice : 00
BraiiiiiiinnnzzzzZ	
Dossier de rendu : <i>ex00/</i>	
Fichiers à rendre : <i>Makefile</i> , <i>main.cpp</i> , <i>Zombie.cpp</i> , <i>Zombie.hpp</i> , <i>newZombie.cpp</i> , <i>randomChump.cpp</i>	
Fonctions interdites : <i>Aucune</i>	

Faites une classe **Zombie**. Les zombies ont un nom et sont capables de s'annoncer comme ça :

```
<name> BraiiiiiiinnnzzzzZ...
```

Oui, `announce( void )` est une fonction membre. De plus, ajoutez un message de débogage dans le destructeur incluant le nom du **Zombie**.

Après cela, écrivez une fonction qui créera un **Zombie**, le nommera et le renverra pour être utilisé ailleurs dans votre code. Le prototype de la fonction est :

```
Zombie* newZombie( std::string name );
```


Vous devrez également écrire une autre fonction qui créera un zombie, et le fera s'annoncer. Le prototype de la fonction est :

```
void randomChump( std::string name );
```

Maintenant, le point réel de l'exercice : vos **Zombies** doivent être détruits au moment opportun (quand ils ne sont plus nécessaires). Ils doivent être alloués sur la pile ou le tas en fonction de leur utilisation : parfois il est approprié de les avoir sur la pile, à d'autres moments, le tas peut être un meilleur choix.

## Chapitre III

### Exercice 01 : Encore plus de cerveaux !

	Exercice : 01
Encore plus de cerveaux !	
Dossier de rendu : <i>ex01/</i>	
Fichiers à rendre : <code>Makefile</code> , <code>main.cpp</code> , <code>Zombie.cpp</code> , <code>Zombie.hpp</code> , <code>ZombieHorde.cpp</code>	
Fonctions interdites : <b>Aucune</b>	

En réutilisant la classe `Zombie`, nous allons maintenant créer une horde de zombies !


Écrivez une fonction qui prend un nombre entier `N`. Lorsqu'elle est appelée, elle alloue `N` objets `Zombie`. Elle doit allouer tous les `N` objets `Zombie` en une seule allocation. Ensuite, elle doit initialiser chaque `Zombie` en lui donnant un nom. Enfin, elle doit retourner le pointeur vers le premier `Zombie`. La fonction est prototypée comme suit :

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

Rendez un `main` pour tester que votre fonction `zombieHorde` fonctionne comme prévu. Vous pouvez vouloir le faire en appelant `announce()` sur chacun des `Zombies`. N'oubliez pas de supprimer TOUS les `Zombies` lorsque vous n'en avez plus besoin.

# Chapitre IV

## Exercice 02 : BONJOUR ICI LE CERVEAU

	Exercice : 02
BONJOUR ICI LE CERVEAU	
Dossier de rendu : <i>ex02/</i>	
Fichiers à rendre : <code>Makefile</code> , <code>main.cpp</code>	
Fonctions interdites : <code>Aucune</code>	


Faites un programme dans lequel vous créez une string contenant "HI THIS IS BRAIN". Créez un `stringPTR` qui est un pointeur vers la string et un `stringREF` qui est une référence à la string.

Vous devez afficher maintenant cette string en utilisant le pointeur puis en utilisant la référence.

C'est tout, pas de piège. Le but de cet exercice est de vous démystifier les références. Ce n'est pas quelque chose de complètement nouveau, c'est juste une autre syntaxe pour quelque chose que vous que vous connaissez déjà : les adresses. Même s'il y a quelques petits détails.

# Chapitre V

## Exercice 03 : Violence inutile

	Exercice : 03
Violence inutile	
Dossier de rendu : <i>ex03/</i>	
Fichiers à rendre : <code>Makefile</code> , <code>main.cpp</code> , <code>Weapon.cpp</code> , <code>Weapon.hpp</code> , <code>HumanA.cpp</code> , <code>HumanA.hpp</code> , <code>HumanB.cpp</code> , <code>HumanB.hpp</code>	
Fonctions interdites : Aucune	

Créez une classe `Weapon`, comportant une chaîne `type`, et une méthode `getType` qui renvoie une référence `const` à cette chaîne. Elle dispose également d'un `setType`.

Maintenant, créez deux classes, `HumanA` et `HumanB`, qui ont toutes les deux une `Weapon`, un nom et une fonction `attack()` qui affiche quelque chose comme :

```
NAME attacks with his WEAPON_TYPE
```

`HumanA` et `HumanB` sont presque identiques ; il n'y a que deux petits détails qui les distinguent :

- Alors que `HumanA` prend l'arme dans son constructeur, `HumanB` ne le fait pas.
- `HumanB` n'a pas toujours une arme, mais `HumanA` sera TOUJOURS armé.



Faites en sorte que le code suivant génère des attaques avec "crude spiked club" PUIS "some othe type of club", dans les deux cas de test :

```
int main()
{
    {
        Weapon      club = Weapon("crude spiked club");


        HumanA bob("Bob", club);
        bob.attack();
        club.setType("some other type of club");
        bob.attack();
    }
    {
        Weapon      club = Weapon("crude spiked club");

        HumanB jim("Jim");
        jim.setWeapon(club);
        jim.attack();
        club.setType("some other type of club");
        jim.attack();
    }
}
```

Dans quel cas est-il approprié de stocker la `Weapon` en tant que pointeur ? en référence ? Pourquoi ? Est-ce le meilleur choix compte tenu de ce qui est demandé ? Telles sont les questions que vous devriez vous poser avant de vous lancer dans cet exercice.

# Chapitre VI

## Exercice 04 : Sed, c'est pour les perdants

	Exercice : 04
Sed, c'est pour les perdants	
Dossier de rendu : <i>ex04/</i>	
Fichiers à rendre : <b>Makefile, et tout ce dont vous avez besoin</b>	
Fonctions interdites : <b>Aucune</b>	

Créez un programme appelé **replace** qui prend un nom de fichier et deux chaînes, appelons-les **s1** et **s2**, qui ne sont PAS vides.


Il ouvrira le fichier et écrira son contenu dans **FILENAME.replace**, après avoir remplacé chaque occurrence de **s1** par **s2**.

Bien sûr, vous gérerez les erreurs du mieux que vous pourrez et n'utiliserez pas les fonctions de manipulation du fichier C, car ce serait tricher, et tricher c'est mal, m'kay ?

Vous rendrez des fichiers de test pour montrer que votre programme fonctionne.

# Chapitre VII

## Exercice 05 : Karen 2.0

	Exercice : 05
Karen 2.0	
Dossier de rendu : <i>ex05/</i>	
Fichiers à rendre : <code>Makefile</code> , <code>main.cpp</code> , <code>Karen.hpp</code> , and <code>Karen.cpp</code>	
Fonctions interdites : <b>Aucune</b>	

Vous connaissez Karen ? Nous la connaissons tous, non ? Au cas où vous ne la connaissiez pas, voici le genre de commentaires que Karen fait :

- Niveau "DEBUG" : Les messages de ce niveau contiennent de nombreuses informations contextuelles. Ils sont principalement utilisés pour le diagnostic des problèmes. Exemple : "J'aime avoir du bacon supplémentaire pour mon burger 7XL-double-cheese-triple-pickle-special-ketchup. J'adore ça !"
- Niveau "INFO" : Ces messages contiennent des informations contextuelles pour aider à tracer l'exécution dans un environnement de production. Exemple : "Je ne peux pas croire que l'ajout de bacon supplémentaire coûte plus cher. Vous n'en mettez pas assez ! Si vous le faisiez, je n'aurais pas à le demander !"
- Niveau "WARNING" : Un message d'avertissement indique un problème potentiel dans le système. Le système est capable de gérer le problème par lui-même ou de le traiter quand même, de toute façon. Exemple : "Je pense que je mérite d'avoir du bacon supplémentaire gratuitement. Je viens ici depuis des années et vous n'avez commencé à travailler ici que le mois dernier."
- Niveau "ERROR" : Un message d'erreur indique un problème sérieux dans le système. Le problème est généralement irrécupérable et nécessite une intervention manuelle. Exemple : "C'est inacceptable, je veux parler au responsable maintenant".

Nous allons automatiser Karen, elle dit toujours la même chose. Vous devez créer une classe nommée `Karen` qui contiendra les fonctions membres privées suivantes :

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

Karen doit aussi avoir une fonction publique qui appelle les fonctions privées selon le niveau passé en paramètre. Le prototype de la fonction est :


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

Le but de cet exercice est d'utiliser des pointeurs vers des fonctions membres. Ce n'est pas une suggestion, Karen doit se plaindre sans utiliser une forêt de `if/elseif/else`, elle n'hésite pas ni ne réfléchit à deux fois !

Soumettre une main pour tester que Karen se plaint beaucoup. Il n'y a pas de problème si vous voulez utiliser les plaintes que nous donnons comme exemple.

# Chapitre VIII

## Exercice 06 : Filtre de Karen

	Exercice : 06
Filtre de Karen	
Dossier de rendu : <i>ex06/</i>	
Fichiers à rendre : <i>Makefile</i> , <i>main.cpp</i> , <i>Karen.hpp</i> , and <i>Karen.cpp</i>	
Fonctions interdites : <i>Aucune</i>	

Nous allons mettre en place un système pour filtrer si ce que dit Karen est vraiment important ou pas, car parfois nous ne voulons pas prêter attention à tout ce que Karen dit.

You have to write the program `karenFilter` that will receive as a parameter the log level you want to listen to and display all the info that is at this level or above it. For example :

Vous devez écrire le programme `karenFilter` qui recevra comme paramètre le niveau de log que vous voulez écouter et qui affichera toutes les informations qui se trouvent à ce niveau ou au-dessus. Par exemple :

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

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

$> ./karenFilter "I am not sure how tired I am today..."
[ Probablement plainte de problèmes insignifiants ]
```

Il y a plusieurs façons de filtrer Karen, mais l'une des meilleures est de la SWITCH en mode off;)