

C++ - Module 01

Allocation mémoire, réference, pointeur membres, file streams

 $R\'esum\'e: \ This \ document \ contains \ the \ subject \ for \ Module \ 01 \ of \ 42's \ C++ \ piscine.$

Table des matières

| Ι | Règles Générales | 2 |
|---------------|--|----|
| II | Exercice 00 : Des variétés de quadripèdes | 4 |
| III | Exercise 01 : Problème de plomberie | 5 |
| IV | Exercise 02 : Triturage de cerveaux | 6 |
| \mathbf{V} | Exercise 03 : Encore plus de cerveaux! | 7 |
| VI | Exercise 04 : BONJOUR ICI LE CERVEAU | 8 |
| VII | Exercise 05 : HI BRAIN THIS IS HUMAN | 9 |
| VIII | Exercise 06 : Unnecessary violence | 10 |
| \mathbf{IX} | Exercise 07 : Sed, c'est pour les perdants | 12 |
| \mathbf{X} | Exercise 08: Les switchs ne passeront pas! | 13 |
| XI | Exercise 09 : Trop de log tue le log | 14 |
| XII | Exercise 10 : Cat o' nine tails | 15 |

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
 - o Vous avez l'autorisation d'utiliser à peu près toute la librairie standard. CE-PENDANT, 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). Ca 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 a scripter ca dans votre éditeur de code préferé.

- 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 préferez. 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 : Des variétés de quadripèdes

| | Exercice: 00 | |
|----------------------------|-----------------------------|--|
| | Des variétés de quadripèdes | |
| Dossier de rendu : $ex00/$ | | |
| Fichiers à rendre : Pony. | cpp Pony.hpp main.cpp | |
| Fonctions interdites : Auc | une | |

An easy one, to start with. Pour commencer, un exercice facile.

Faites une classe Pony, qui contient le nécessaire pour décrire un poney (Mettez ce que souhaitez).

Ensuite, créez deux fonction ponyOnTheStack et ponyOnTheHeap, dans lesquelles vous allouez un Pony, et faites lui faire des choses.

Bien entneud, le premier Pony doit etre alloué sur la stack, et le second sur la heap.

Vous devez rendre un main avec des tests prouvant que vous avez rendu quelque chose qui marche. Dans les deux cas, l'objet Pony ne doit plus exister quand vous sortez de la fonction. Votre main doit le prouver.

Chapitre III

Exercise 01 : Problème de plomberie

| 4 | D : 01 | |
|-------------------------------|-----------------------|--|
| | Exercice: 01 | |
| 7 | Problème de plomberie | |
| Dossier de rendu : $ex01/$ | | |
| Fichiers à rendre : ex01.cpp | | |
| Fonctions interdites : Aucune | | |

Encore un exercice facile.

Vous devez rendre les fonctions données avec le sujet, une fois que la fuite mémoire est corrigée.

Bien entendu, nous attendons de vous que ous jouiez avec l'allocation/libération mémoire ici. Enlever juste une variable ou autre sans résoudre le problème comptera comme une mauvaise réponse.

```
void     memoryLeak()
{
     std::string*     panthere = new std::string("String panthere");
     std::cout << *panthere << std::endl;
}</pre>
```

Chapitre IV

Exercise 02 : Triturage de cerveaux

| 2 | Exercice: 02 | |
|---------------------------|--|----------|
| / | Triturage de cerveaux | |
| Dossier de rendu : $ex02$ | 1 | / |
| Fichiers à rendre : Zomb | ie.cpp Zombie.hpp ZombieEvent.cpp ZombieEv | rent.hpp |
| main.cpp | | |
| Fonctions interdites : Au | cune | / |

Commencez par créer une classe Zombie. Rajoutez lui un type et un nom (au moins) et ajoutez une fonction membre advert(), qui affichera quelque chose comme :

<nom (type) > Braiiiiiiinnnssss ...

Quoi que vous voulez, vraiment, tant que vous indiquez le nom et le type de Zombie.

Après cela, vous allez créer une classe ZombieEvent. Il y aura un setZombieType, qui stockera un type dans l'objet et un function Zombie * newZombie (std::string name) qui créera un Zombie avec le type choisi, nommez-le et retournez-le.

Vous ferez aussi une fonction randomChump, qui créera un Zombie avec un nom aléatoire et faites-le s'announce(). Vous êtes libre d'implémenter votre méthode d'"aléatoire", soit des noms véritablement aléatoires ou alors un choix aléatoire parmi un groupe de noms.

Vous devez rendre un programme complet, main inclus, avec suffisamment d'éléments pour prouver que ce que vous avez fait fonctionne comme vous le souhaitez. Par exemple, faites en sorte que votre zombie créé s'annonce.

Maintenant le véritable objectif de l'exercice : vos Zombies doivent être détruits aux moment approprié (donc, quand ils ne sont plus nécessaires). Ils doivent aussi être alloué de la manière appropriée : Parfois, il est approprié de les avoir sur la stack, à d'autres moments, la heap est un meilleur choix. Vous devrez justifier ce que vous avez fait pour obtenir une note positive.

Chapitre V

Exercise 03 : Encore plus de cerveaux!

| Exercice: 03 | |
|---|---------------|
| Encore plus de cerveaux! | |
| Dossier de rendu : $ex03/$ | |
| Fichiers à rendre : Zombie.cpp Zombie.hpp ZombieHorde.cpp Zom | mbieHorde.hpp |
| main.cpp | |
| Fonctions interdites: Aucune | |

En réutilisant la classe Zombie de l'exercice précedent, faites une classe ZombieHorde.

Cette classe aura un constructeur qui prend un entier n. À sa création, il doit allouer n objets zombies, avec un nom aléatoire (même chose que l'exercice précedent), et les stocker.

Vous implémenterez également une méthode announce() qui appellera la fonction announce() de chacun des zombies stockés.

Vous devez allouer tous les objets Zombie en une seule allocation, et les détruire quand ZombieHorde est détruite.

Rendez un main avec des tests qui justifient vos choix.

Chapitre VI

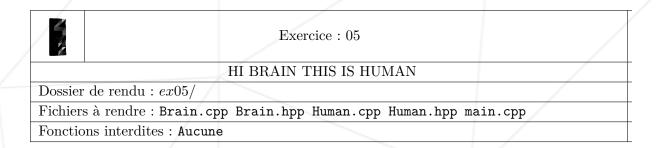
Exercise 04 : BONJOUR ICI LE CERVEAU

| | Exercice: 04 | |
|------------|------------------------|---|
| | BONJOUR ICI LE CERVEAU | |
| Dossier de | rendu: $ex04/$ | |
| Fichiers à | rendre: ex04.cpp | / |
| Fonctions | interdites · Aucune | |

Faites un programme dans lequel vous créérez une string contenant "HI THIS IS BRAIN", un pointeur vers cette string, et une reference. Vous devez l'afficher en utilisant le pointeur puis en utilisant la réference. C'est tout, pas de piège.

Chapitre VII

Exercise 05 : HI BRAIN THIS IS HUMAN



Créez une classe Brain, avec tout ce que vous croyez digne d'un cerveau. Vous devez avoir une fonction identifier(), qui retourne une chaîne contenant l'adresse du cerveau en mémoire, au format hexadécimal, préfixée par 0x (Par exemple, "0x194F87EA").

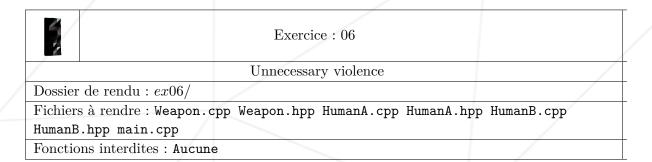
Ensuite, créez une classe Human, qui a un attribut constant Brain, avec la même vie. Il a une fonction identifier(), qui appelle simplement la fonction identifier() de son cerveau et renvoie son résultat.

Maintenant, assurez-vous que ce code compile et affiche deux adresses identiques :

Ce code doit être rendu en tant que main, et tout ce que vous ajoutez aux classes Human ou Brain afin de le faire fonctionner doit être justifié (avec un autre argument que "Eh bien, je me suis amusé jusqu'à ce que cela fonctionne ").

Chapitre VIII

Exercise 06: Unnecessary violence



Créez une classe Weapon comportant une chaîne type et un getType renvoyant une référence const à cette chaîne. Ajoutez également un setType.

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

NAME attacks with his WEAPON_TYPE

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 :

Dans quel cas est-il approprié de stocker le 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 IX

Exercise 07: Sed, c'est pour les perdants

| | Exercice: 07 | |
|-----------------------------|-------------------------------------|---|
| · | Sed, c'est pour les perdants | / |
| Dossier de rendu : $ex07/$ | | / |
| Fichiers à rendre : Makefil | e, et tout ce dont vous avez besoin | / |
| Fonctions interdites: Aucur | e | / |

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

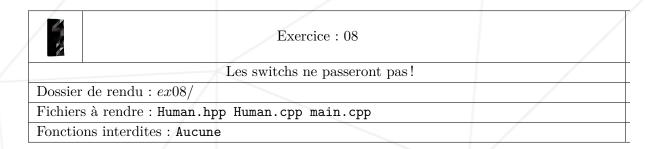
Il ouvrira le fichier et écrira son contenu dans FILENAME.replace, après avoir remplacé chaque occurrence de s1 par s2.

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 X

Exercise 08: Les switchs ne passeront pas!





Cet exercice et ceux qui suivent ne rapportent pas de points, mais demeurent interessant dans le cadre de votre piscine. Vous n'êtes pas obligés de les faire.

Utilisez la classe Human qui suit :

Implémentez toutes ces fonctions, les trois premières vont simplement sortir quelque chose sur la sortie standard afin que vous puissiez voir qu'elles ont été appelées, et la dernière devra appeler l'action appropriée sur la cible appropriée. Vous devez utiliser un tableau de pointeurs sur les membres pour sélectionner la fonction à appeler : l'utilisation de plusieurs instructions if ou switch est interdite.

Chapitre XI

Exercise 09: Trop de log tue le log

| | Exercice: 09 | |
|--|------------------------|--|
| / | Trop de log tue le log | |
| Dossier de rendu : ex09 | 1 | |
| Fichiers à rendre : Logger.cpp Logger.hpp main.cpp | | |
| Fonctions interdites : Au | cune | |

Créez une classe Logger qui devra, évidemment, écrire des logs.

Il aura deux fonctions privées, logToConsole et logToFile, qui prennent tous les deux une chaîne et l'écrivent respectivement dans la sortie standard et l'ajoutent à un fichier, dont le nom sera stocké dans le Logger au moment de la création.

Vous allez également créer une fonction privée appelée makeLogEntry qui prendra un message simple sous forme de chaîne de caractères et renverra une nouvelle chaîne contenant le message au format ressemblant à un log légitime. Au minimum, ajoutez la date du jour avant le message pour que nous puissions voir quand il a été enregistré.

Enfin, créez un log(std::string const &dest, std::string const &message), qui créera un log avec le message, et le transmettra à logToFile ou logToConsole, en fonction du paramètre dest. Comme dans l'exercice précédent, vous devez utiliser des pointeurs sur les membres pour sélectionner la fonction à appeler.

Chapitre XII

Exercise 10: Cat o' nine tails

| | Exercice: 10 | |
|------------------------------|---------------------|--|
| / | Cat o' nine tails | |
| Dossier de rendu : $ex10/$ | | |
| Fichiers à rendre : main.cpp | + Whatever you need | |
| Fonctions interdites : Aucun | e | |

Faites un programme nommé cato9tails qui fait la même chose que le cat système, sans options. Il pourra lire des fichiers et/ou depuis l'entrée standard. Faites attention pendant vos tests, ca n'est pas aussi simple que ça en a l'air.