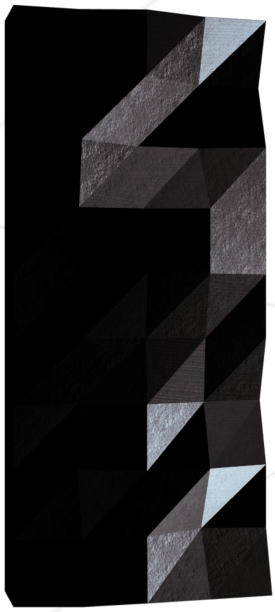

C++ - Module 07

c++ Templates

Résumé: Ce document contient le sujet du module 07 des modules de C++ de 42.

Table des matières

Chapitre I

Règles Générales

- Toute fonction déclarée dans une header (sauf 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. CEPENDANT, 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 paraitre 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 -std=c++98`
- 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 : Quelques fonctions

	Exercice : 00
Exercice 00 : Quelques fonctions	
Dossier de rendu : $ex00/$	
Fichiers à rendre : `whatever.hpp, main.cpp et un Makefile`	
Fonctions Autorisées : `Aucune`	

Écrivez les templates de fonction suivants :

- `swap` : échange les valeurs de deux arguments. ne renvoie rien.
- `min` : compare les valeurs de deux arguments et renvoie le plus petit. Si les deux arguments sont égaux, renvoie le second argument.
- `max` : compare les valeurs de deux arguments et renvoie le plus grand. Si les deux arguments sont égaux, renvoie le second argument.

Ces fonctions peuvent être appelées avec n'importe quel type d'argument (les deux doivent quand même être du même type et doivent supporter les opérateurs de comparaison).
Rendez un `main` qui prouve que votre code fonctionne.

Le code qui suit :

```
int     main( void ) {

    int a = 2;
    int b = 3;

    ::swap( a, b );
    std::cout << "a = " << a << ", b = " << b << std::endl;
    std::cout << "min( a, b ) = " << ::min( a, b ) << std::endl;
    std::cout << "max( a, b ) = " << ::max( a, b ) << std::endl;

    std::string c = "chaine1";
    std::string d = "chaine2";

    ::swap(c, d);
    std::cout << "c = " << c << ", d = " << d << std::endl;
    std::cout << "min( c, d ) = " << ::min( c, d ) << std::endl;
    std::cout << "max( c, d ) = " << ::max( c, d ) << std::endl;

    return 0;
}
```

Doit renvoyer (si vous avez tout juste) :

```
a = 3, b = 2
min(a, b) = 2
max(a, b) = 3
c = chaine2, d = chaine1
min(c, d) = chaine1
max(c, d) = chaine2
```

Chapitre III

Exercice 01 : Iter

	Exercice : 01
Exercice 01 : Iter	
Dossier de rendu : $ex01/$	
Fichiers à rendre : `iter.hpp, main.cpp et un Makefile`	
Fonctions Autorisées : `Aucune`	

Écrivez un fonction template `iter` qui prend 3 paramètres et ne renvoie rien. Le premier paramètre correspond à l'adresse d'un array, le second sa taille, et la troisième une fonction de type `void`, à appeler sur chaque élément de l'array.
Rendez un `main` prouvant que votre `iter` marche avec n'importe quel type d'array et/ou avec une instance de fonction template en 3ème paramètre.

Chapitre IV

Exercice 02 : Array

	Exercice : 02
Exercice 02 : Array	
Dossier de rendu : $ex02/$	
Fichiers à rendre : `Array.hpp, main.cpp et un Makefile`	
Fonctions Autorisées : `Aucune`	

Écrivez une classe template `Array` qui contient des éléments de type `T` et qui autorise les comportements suivants :

- Construction sans paramètre : crée un array vide
- Construction avec un `unsigned int n` en paramètre : Crée un array de `n` éléments, initialisés par défaut. (Astuce : Essayez de compiler `int * a = new int();`, puis affichez `a`)
- Construction par copie et assignation. Dans les deux cas, modifier l'un des array après copie/assignation ne doit pas affecter l'autre.
- Vous DEVEZ utiliser l'opérateur `new[]` pour votre allocation. Vous ne devez pas faire d'allocation en prévision. Votre code ne dois jamais accéder de la mémoire non allouée.
- Les éléments doivent être accessibles via l'operateur [].
- En accédant à un élement avec l'opérateur [], si cet élément est hors-limites, vous devez utiliser une `std::exception`.
- Une fonction membre `size` qui renvoie le nombre d'élements de l'array. Cette fonction ne prend pas d'argument ni ne modifie l'instance d'une quelconque manière.

Rendez un `main` qui prouve que votre code fonctionne.