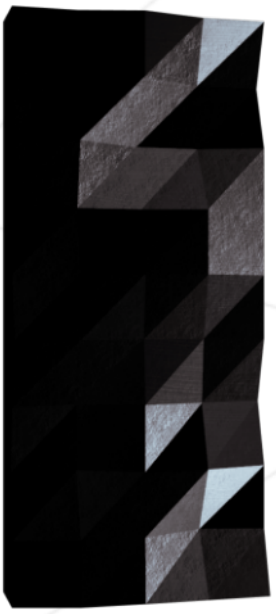

C++ - Module 07

C++ templates

Résumé:
Ce document contient les exercices du Module 07 des C++ modules.

Version: 6

Table des matières

Chapitre I

Introduction

C++ is a general-purpose programming language created by Bjarne Stroustrup as an extension of the C programming language, or "C with Classes" (source : Wikipedia).

C++ est un langage de programmation compilé permettant la programmation sous de multiples paradigmes, dont la programmation procédurale, la programmation orientée objet et la programmation générique. Ses bonnes performances, et sa compatibilité avec le C en font un des langages de programmation les plus utilisés dans les applications où la performance est critique (source : Wikipedia).

Ces modules ont pour but de vous introduire à la **Programmation Orientée Objet**. Plusieurs langages sont recommandés pour l'apprentissage de l'OOP. Du fait qu'il soit dérivé de votre bon vieil ami le C, nous avons choisi le langage C++. Toutefois, étant un langage complexe et afin de ne pas vous compliquer la tâche, vous vous conformerez au standard C++98.

Nous avons conscience que le C++ moderne est différent sur bien des aspects. Si vous souhaitez pousser votre maîtrise du C++, c'est à vous de creuser après le tronc commun de 42 !

Chapitre II

Consignes générales

Compilation

- Compilez votre code avec `c++` et les flags `-Wall -Wextra -Werror`
- Votre code doit compiler si vous ajoutez le flag `-std=c++98`

Format et conventions de nommage

- Les dossiers des exercices seront nommés ainsi : `ex00, ex01, ... , exn`
- Nommez vos fichiers, vos classes, vos fonctions, vos fonctions membres et vos attributs comme spécifié dans les consignes.
- Rédigez vos noms de classe au format **UpperCamelCase**. Les fichiers contenant le code d'une classe porteront le nom de cette dernière. Par exemple : `NomDeClasse.hpp`/`NomDeClasse.h`, `NomDeClasse.cpp`, ou `NomDeClasse.tpp`. Ainsi, si un fichier d'en-tête contient la définition d'une classe "BrickWall", son nom sera `BrickWall.hpp`.
- Sauf si spécifié autrement, tous les messages doivent être terminés par un retour à la ligne et être affichés sur la sortie standard.
- *Ciao Norminette !* Aucune norme n'est imposée durant les modules C++. Vous pouvez suivre le style de votre choix. Mais ayez à l'esprit qu'un code que vos pairs ne peuvent comprendre est un code que vos pairs ne peuvent évaluer. Faites donc de votre mieux pour produire un code propre et lisible.

Ce qui est autorisé et ce qui ne l'est pas

Le langage C, c'est fini pour l'instant. Voici l'heure de se mettre au C++ ! Par conséquent :

- Vous pouvez avoir recours à quasi l'ensemble de la bibliothèque standard. Donc plutôt que de rester en terrain connu, essayez d'utiliser le plus possible les versions C++ des fonctions C dont vous avec l'habitude.
- Cependant, vous ne pouvez avoir recours à aucune autre bibliothèque externe. Ce qui signifie que C++11 (et dérivés) et l'ensemble `Boost` sont interdits. Aussi, certaines fonctions demeurent interdites. Utiliser les fonctions suivantes résultera

en la note de 0 : `*printf()`, `*alloc()` et `free()`.

- Sauf si explicitement indiqué autrement, les mots-clés `using namespace <ns_name>` et `friend` sont interdits. Leur usage résultera en la note de -42.
- **Vous n'avez le droit à la STL que dans les Modules 08 et 09**. D'ici là, l'usage des **Containers** (vector/list/map/etc.) et des **Algorithmes** (tout ce qui requiert d'inclure `<algorithm>`) est interdit. Dans le cas contraire, vous obtiendrez la note de -42.

Quelques obligations côté conception

- Les fuites de mémoires existent aussi en C++. Quand vous allouez de la mémoire (en utilisant le mot-clé `new`), vous ne **devez pas avoir de memory leaks**.
- Du Module 02 au Module 09, vos classes devront se conformer à la forme **canonique, dite de Coplien, sauf si explicitement spécifié autrement**.
- Une fonction implémentée dans un fichier d'en-tête (hormis dans le cas de fonction template) équivaudra à la note de 0.
- Vous devez pouvoir utiliser vos fichiers d'en-tête séparément les uns des autres. C'est pourquoi ils devront inclure toutes les dépendances qui leur seront nécessaires. Cependant, vous devez éviter le problème de la double inclusion en les protégeant avec des **include guards**. Dans le cas contraire, votre note sera de 0.

Read me

- Si vous en avez le besoin, vous pouvez rendre des fichiers supplémentaires (par exemple pour séparer votre code en plus de fichiers). Vu que votre travail ne sera pas évalué par un programme, faites ce qui vous semble le mieux du moment que vous rendez les fichiers obligatoires.
- Les consignes d'un exercice peuvent avoir l'air simple mais les exemples contiennent parfois des indications supplémentaires qui ne sont pas explicitement demandées.
- Lisez entièrement chaque module avant de commencer ! Vraiment.
- Par Odin, par Thor ! Utilisez votre cervelle !!!

Vous aurez à implémenter un bon nombre de classes, ce qui pourrait s'avérer ardu... ou pas ! Il y a peut-être moyen de vous simplifier la vie grâce à votre éditeur de texte préféré.

Vous êtes assez libre quant à la manière de résoudre les exercices. Toutefois, respectez les consignes et ne vous en tenez pas au strict minimum, vous pourriez passer à côté de notions intéressantes. N'hésitez pas à lire un peu de théorie.

Chapitre III

Exercice 00 : Quelques fonctions pour commencer

	Exercice : 00
Quelques fonctions pour commencer	
Dossier de rendu : $ex00/$	
Fichiers à rendre : `Makefile, main.cpp, whatever.{h, hpp}`	
Fonctions interdites : `Aucune`	

Implémentez les fonctions templates suivantes :

- `swap` : Intervertit les valeurs de deux arguments donnés. Ne retourne rien.
- `min` : Compare les deux valeurs passées en argument et retourne la plus petite des deux. Si les deux sont équivalentes, alors retourne la seconde.
- `max` : Compare les deux valeurs passées en argument et retourne la plus grande des deux. Si les deux sont équivalentes, alors retourne la seconde.

Ces fonctions doivent pouvoir être appelées avec n'importe quel type d'argument à seule condition que ces derniers aient le même type et supportent les opérateurs de comparaison.

`Les templates doivent être définis dans les fichiers d'en-tête.`

Exécuter le code suivant :

```
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;
}
```

Devrait afficher ce résultat :

```
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 IV

Exercice 01 : Iter

	Exercice : 01
Iter	
Dossier de rendu : $ex01/$	
Fichiers à rendre : `Makefile, main.cpp, iter.{h, hpp}`	
Fonctions interdites : `Aucune`	

Implémentez une fonction template `iter` prenant 3 paramètres et ne retournant rien.

- Le premier paramètre est l'adresse d'un tableau.
- Le second est la taille du tableau.
- Le troisième est une fonction qui sera appelée sur chaque élément du tableau.

Rendez un fichier `main.cpp` contenant vos tests. Fournissez assez de code pour générer un exécutable de test.

Votre fonction template `iter` devra fonctionner avec n'importe quel type de tableau. Le troisième paramètre peut être une fonction template instanciée.

Chapitre V

Exercice 02 : Array

	Exercice : 02
Array	
Dossier de rendu : *ex*02/	
Fichiers à rendre : `Makefile, main.cpp, Array.{h, hpp}` `fichier optionnel : Array.tpp`	
Fonctions interdites : `Aucune`	

Créez une classe template **Array** (*tableau*) contenant des éléments de type `T` et qui implémente le comportement suivant et les fonctions suivantes :

- Construction sans paramètres : crée un *array* vide.
- Construction avec un paramètre de type unsigned int `n` : crée un *array* de `n` éléments initialisés par défaut.
  Conseil : Essayez de compiler `int * a = new int();` puis affichez `*a`.
- Construction par recopie et surcharge de l'opérateur d'affectation. Dans les deux cas, après une copie, toute modification de l'*array* original ou de sa copie ne doit pas impacter l'autre *array*.
- Vous DEVEZ utiliser l'opérateur `new[]` pour allouer de la mémoire. Toute allocation préventive (c'est-à-dire allouer de la mémoire en avance) est interdite. Votre programme ne doit pas pouvoir accéder à une zone non allouée.
- Les éléments doivent être accessibles grâce à l'opérateur d'indice : [ ].
- En cas d'index invalide lors d'une tentative d'accès d'un élément en utilisant l'opérateur [ ], une `std::exception` est jetée.
- Une fonction membre `size()` qui retourne le nombre d'éléments dans l'*array*. Cette fonction membre ne prend aucun paramètre et ne doit pas modifier l'instance courante.

Comme d'habitude, assurez-vous que tout fonctionne comme attendu et rendez un fichier `main.cpp` contenant vos tests.