

## ISIMA 3<sup>ème</sup> année - MODL/C++

### TP 4 : C++11

On propose d'illustrer certaines fonctionnalités du C++11 en adaptant un code C++03 existant sur la modélisation d'un vecteur de nombres complexes : la classe **Vecteur** contient des objets du type **complexe\_t**. Les complexes sont manipulés à travers le type **complexe\_t**, et non pas directement par la classe **Complexe**, car on souhaite tracer les constructions et les copies d'objets de cette classe. Pour cela on manipule des objets de la classe **Mouchard<Complexe>** *a.k.a.* **complexe\_t**. Après avoir terminé ce TP, vous pourrez vous pencher sur le contenu de la classe **Mouchard**.

- 1) Etudier le code des fichiers `test_mouchard.cpp` et `test_vecteur03.cpp` et les résultats qu'ils produisent afin de bien comprendre d'où viennent toutes les copies et les "mouvements" d'objets **Complexe**. Attention : une même opération peut être comptabilisée plusieurs fois ; par exemple, si le constructeur de copie est appelé, l'opération est comptabilisée comme "construction" et comme "copie".
- 2) Remplacer les boucles `for` traditionnelles par des boucles équivalentes à l'algorithme `foreach` du C++03 pour tous les parcours de conteneur. Pour parcourir un vecteur notamment, des modifications au niveau de l'interface de la classe **Vecteur** sont nécessaires.
- 3) L'utilisation des opérateurs `+` et `*` sur les vecteurs créent de nombreuses copies. Pour limiter ces dernières, implémenter un opérateur d'affectation par mouvement pour la classe **Vecteur**. (Le constructeur de mouvement doit aussi être implémenté, même s'il n'est pas utilisé dans notre exemple).
- 4) Dans l'exemple, le vecteur `v5` est copié dans un conteneur STL et trié selon l'ordre défini par le foncteur comparateur **CompareurComplexe** (les nombres sont ordonnés d'abord sur leur partie réelle, puis, en cas d'égalité, sur leur partie imaginaire). Utiliser une expression lambda à la place du foncteur pour effectuer le tri.
- 5) Les opérateurs `+` et `*` ont été surchargés pour les vecteurs, mais on pourrait imaginer de nombreuses opérations binaires entre vecteurs. Proposer une fonction générique dont le paramètre est l'opération binaire à appliquer aux éléments des deux vecteurs passés en argument. Comme pour les opérateurs, le résultat sera renvoyé dans un nouveau vecteur.

L'opération passée en paramètre pourra être une lambda, un foncteur ou un pointeur de fonction. Pour l'exemple, utiliser la fonction avec une opération d'addition sous la forme d'une lambda, une opération de soustraction sous la forme d'un foncteur et une opération de multiplication sous la forme d'une fonction.

*Remarque : Selon le compilateur, la norme activée par défaut diffère. Avec `g++`, pour activer la norme C++**XX**, il faut ajouter l'option « `-std=c++XX` », où **XX**=03, 11, 14, 17...*