ISIMA 3ème année - MODL/C++
TP 1 : Révisions

Exercice 1

1) Définir une classe **Point** proposant une méthode abstraite `afficher()`. Dériver deux classes, **Cartesien** et **Polaire**, représentant un point sous forme de coordonnées respectivement cartésiennes et polaires. Offrir un jeu de fonctionnalités minimal à ces classes.

2) Surcharger les opérateurs vers les flux d'entrée (`std::cin`) et de sortie (`std::cout`) de manière à ce qu'ils exploitent l'héritage défini précédemment.

3) Proposer des passerelles de conversion entre les deux classes concrètes. Deux approches sont possibles qu'il vous est demandé d'implémenter: (i) proposer des méthodes virtuelles dans la classe **Point** et ses filles pour convertir un point dans chacun des types **Cartesien** et **Polaire**; (ii) proposer un constructeur dans chacune des deux classes concrètes qui permette la conversion à partir de l'autre classe.

4) Dans le programme principal, stocker des instances de **Cartesien** et de **Polaire** dans un vecteur STL et afficher son contenu. Essayer `vector<Point>` et `vector<Point *>`.

5) Aux vues de la question précédente, définir une classe **Nuage** utilisant la classe `vector` et contenant un ensemble de points (qui peuvent être de types différents). Cette classe devra fournir des itérateurs, à la manière STL (méthodes `begin()` et `end()`, type `iterator`).

6) Proposer une fonction qui calcule le barycentre d'un nuage de points. Proposer une version foncteur de cette fonction.

Exercice 2

1) Définir une classe **Vecteur** conservant des entiers dans un tableau dynamique. Proposer les méthodes nécessaires au fonctionnement de cette classe, en particulier celles de la forme normale de Coplien (constructeur par défaut, constructeur par copie, destructeur et opérateur de copie).

2) Ajouter des opérateurs pour l'insertion sur le flux en sortie (`<<`), la concaténation (`+`), l'accès direct (`[]`, accès lecture ou lecture/écriture) et le produit scalaire (`*`).

3) Définir une classe **Iterateur** pour parcourir un vecteur avec les mécanismes de base:

   - itérateur au début du vecteur (fourni par le vecteur): `begin()`
   - itérateur à la fin du vecteur (fourni par le vecteur): `end()`
   - passage à l'élément suivant (préfixé): `Iterateur & operator ++ ()`
   - passage à l'élément suivant (postfixé): `Iterateur operator ++ (int)`
   - accès à l'élément pointé: `int operator * ()`
   - comparaison entre deux itérateurs: `bool operator == (const Iterateur &)`