TP ILO n° 4

Prérequis

- Ce TP se présente sous la forme d'une classe C++ à compléter dont vous pourrez trouver un squelette d'implémentation dans /pub/ILO_TP4.zip
- Référence C++ :
  - http://www.cplusplus.com/reference/

Coordonnée Générique

On souhaite réaliser un patron de classe permettant de représenter des coordonnées dans un espace n-dimensionnel (typiquement de taille 2, 3 ou 4). Ces coordonnées sont caractérisées par le type de valeurs qu'elles contiennent (`T`) ainsi que le nombre de valeurs qu'elles contiennent (`N` de type `size_t` qui est un entier non signé).
On veut pouvoir :

- Créer et copier tout type de coordonnées.
- Accéder aux valeurs stockées dans les coordonnées (en lecture et en écriture).
- Comparer des coordonnées entre elles (au sens d'un écart infinitésimal (epsilon) entre les valeurs de deux coordonnées).
- Convertir un type de coordonnée en un autre (par exemple `Coords<double,2>` en `Coords<float,4>`) lors d'une affectation.
- Réaliser des opérations arithmétiques simples sur les coordonnées (+, -, *, /) élément par élément.
- Réaliser quelques opérations spécifiques aux vecteurs comme le produit scalaire, la norme, la translation et l'homothétie.
- Et enfin afficher une représentation textuelle d'une coordonnée sous la forme : $(x_1, x_2, \ldots, x_n)$ dans un flux de sortie.

Coords	T
	size_t N

```
// Attributs
- data : T[]
- selfIndex : size_t

// Méthodes utilitaires
- setValues<U>(const U &)
- setValues<U, ...V>(const U &, const ...V &)
- hasInfinity() : bool

// Constructeurs / Destructeur
+ Coords()
+ Coords<U, ...V>(const U &, const ...V &)
+ Coords(std::initializer_list<T>)
+ Coords(const Coords<T, N> &)
+ Coords<U, M>(const Coords<U, M> &)
+ ~Coords()

// Opérations/Opérateurs d'accès aux valeurs
+ get(const size_t) const : T
+ get(const size_t) : T &
+ operator [](const size_t) const : T
+ operator [](const size_t) : T &
+ size() : size_t

// Opérateurs de comparaison
+ operator ==(const Coords<T, N> &) const : bool
+ operator !=(const Coords<T, N> &) const : bool

// Opérateur de copie/conversion
+ operator =<U, M>(const Coords<U, M> &) : Coords<T, N> &

// Opérateurs de self arithmétique
+ operator +=(const Coords<T, N> &) : Coords<T, N> &
+ operator -=(const Coords<T, N> &) : Coords<T, N> &
+ operator *=(const Coords<T, N> &) : Coords<T, N> &
+ operator /=(const Coords<T, N> &) : Coords<T, N> &

// Opérateurs arithmétiques
+ operator +(const Coords<T, N> &) : Coords<T, N>
+ operator -(const Coords<T, N> &) : Coords<T, N>
+ operator *(const Coords<T, N> &) : Coords<T, N>
+ operator /(const Coords<T, N> &) : Coords<T, N>

// Opérations
+ dotProduct() const : T
+ norm2() const : T
+ translate(const Coords<T, N> &) : Coords<T, N> &
+ scale(const T &) : Coords<T, N> &

// Opérateur global de sortie
operator <<(ostream &, const Coords<T, N> &) : ostream &
```

Travail à effectuer :

Complétez la classe Coords<T, N> et faites tourner le programme de test contenu dans TestCoords.cpp ou bien main.cpp (Voir l'annexe page 3).

Recommandations

- Lorsque vous avez fini d'implémenter une méthode, supprimez le commentaire contenant le TODO, cela nous permettra de corriger votre code plus rapidement.
- N'attendez pas la fin du TP pour compiler votre projet et lancer le programme de test : Faites-le dès le début (la plupart des tests échoueront) et régulièrement (jusqu'à ce que l'ensemble des tests réussissent).
- Ne modifiez en aucun cas les signatures des méthodes de Coords<T, N> ou les programmes de test (TestCoords.cpp, ou main.cpp)
- Respectez les mentions « Ne rien écrire ici » dans le code à compléter.
- Pour fabriquer votre archive de code lors du rendu, placez vous à la racine du projet et tapez la commande « make clean archive » qui fabriquera un fichier zip dans le sous-répertoire « archives » contenant tous les fichiers sources de votre projet.