# TP n° 4 : Manipulation des patrons et de la STL

Ce TP est divisé en deux phases qui doivent, dans la mesure du possible, être traitées simultanément. La première phase consiste en l'implémentation pure de la classe et de ses méthodes. La seconde phase s'attachera à l'analyse et à l'amélioration du code.

---

L'objectif de ce TP est de travailler sur la notion de patron et de manipuler différents éléments de la STL. Vous devrez en particulier prêter attention à ne pas rendre votre code trop dépendant de la nature du conteneur utilisé. **Une attention toute particulière sera prêter à la complétude des classes : les classes ne doivent contenir que les opérateurs et fonctions qui ont un sens de le contexte étudié.**

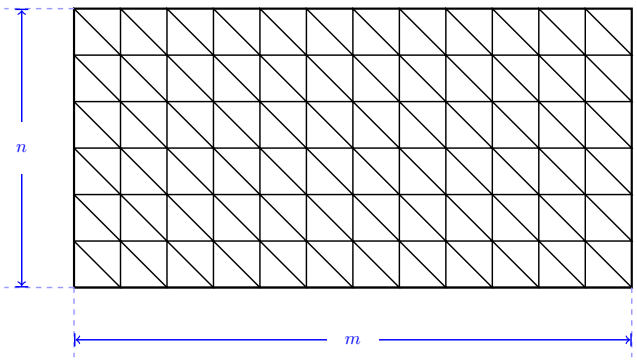
La gestion d'erreurs doit se faire via des exceptions (utilisez `throw`).

### Phase 1 : Implémentation

1. * Implémenter une classe `Point` bidimensionnel :
   (a) Dont le type sera soit le `double`, soit le `float`, et dont le constructeur prendra l'abscisse en premier paramètre et l'ordonnée en second.
   (b) Proposant les méthodes `x()` et `y()` pour l'accès en lecture aux coordonnées précisées à la création.

   Une fois la question 1 traitée, vérifier que la structure (dossiers, fichiers, nom des classes et fonctions) répond bien aux instructions données : Lancer le script `verifier.py`, celui-ci doit confirmer que la question 1 a été traitée correctement. Relancer `verifier.py` régulièrement (pour chaque question), afin de vérifier que vous respectez bien l'énoncé. **Enfin, le relancer sur l'archive compressée (.tar.gz) avant de l'envoyer.**
2. * Implémenter une classe `Triangle` constituée de trois points :
   (a) Dont le constructeur prendra trois points en paramètre
   (b) Proposant les méthodes `p1()`, `p2()` et `p3()` permettant d'accéder aux trois points.
3. * Implémenter une classe `Maillage` constituée de triangles en utilisant un conteneur STL :
   (a) Utilisant deux paramètres "template". Le premier pour indiquer le type d'objet (`double`, ou `float`), et le second pour indiquer le type de conteneur (`std::vector`, `std::list`, ...). Un constructeur permettra de générer un maillage à partir des nombres $m$ et $n$ de segments selon les directions $x$ et $y$, à partir d'un point d'origine. Chaque carré constituant le maillage aura des côtés de taille 1 et sera formé de deux triangles orientés comme dans la figure ci-dessous :

```
Maillage<T,C>::Maillage( int m, int n, const Point<T>& origine )
```



(b) Des méthodes `beginiter()` et `enditer()` permettant de récupérer un itérateur constant sur le début et la fin du conteneur

4. Définir l'opérateur de redirection ($<<$) d'un maillage vers un flux : l'enregistrement est réalisé par triangle en écrivant sur 4 lignes les coordonnées des points du triangle, le premier point étant répété en 4ème position, chaque triangle étant séparé par une ligne blanche.

   Le fichier résultant du maillage d'un carré par 2 triangles (`Maillage m( 1, 1, Point<T>(0,0) )`) aura donc la structure suivante.

```
0. 0.
0. 1.
1. 0.
0. 0.

1. 1.
0. 1.
1. 0.
1. 1.
```

   On pourra par exemple l'afficher en utilisant `gnuplot`, avec la commande suivante :

```
plot [-0.2:1.2][-0.2:1.2] "carre.dat" w l
```

5. Ecrire la fonction `fusionner( const Maillage<T,C>& m )` permettant la fusion de deux maillages qui ne se recouvrent pas.

6. Ecrire des fonctions permettant les transformations "affines" d'un maillage :

   (a) `transformer( double m11, double m12, double m21, double m22 )` pour la transformation affine d'un point, d'un triangle ou d'un maillage utilisant la matrice de transformation affine tel que : $\begin{pmatrix} x' & y' \end{pmatrix} = \begin{pmatrix} x & y \end{pmatrix} \begin{pmatrix} m11 & m12 \\ m21 & m22 \end{pmatrix} = \begin{pmatrix} x*m11+y*m21 & x*m12+y*m22 \end{pmatrix}$

   (b) `deplacer( double dx, double dy )` pour le déplacement d'un point, d'un triangle ou d'un maillage tel que : $\begin{pmatrix} x' & y' \end{pmatrix} = \begin{pmatrix} x+dx & y+dy \end{pmatrix}$

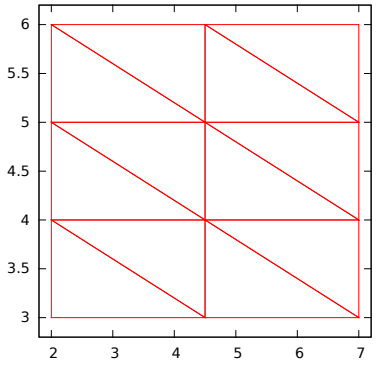
   (c) `tourner( double angle, const Point<T>& pt )` pour la rotation d'un maillage de $angle$ radians autour du point $pt$.

7. Ecrire un constructeur permettant le maillage d'un rectangle quelconque en s'appuyant sur les fonctions précédentes. Ce constructeur prendra en paramètre les 4 points formant le rectangle ainsi que les nombres de $m$ et $n$ de segments :

   `Maillage<T,C>::Maillage( const Point<T>& p1, const Point<T>& p2, const Point<T>& p3, const Point<T>& p4, int m, int n )`

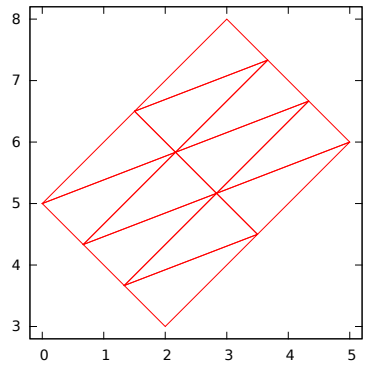
   Le rectangle généré sera constitué de $m$ segments entre $p1$ et $p2$ et $n$ segments entre $p2$ et $p3$.
   Par exemple :

   `Maillage( Point(2,3), Point(7,3), Point(7,6), Point(2,6), 2, 3 )` affichera :




Ou encore :

`Maillage( Point(2,3), Point(5,6), Point(3,8), Point(0,5), 2, 3 )` affichera :




Note : Votre code devra lever une exception pour les cas invalides.

**Phase 2 : Analyse**

1. * Etudier les performances de votre code en fonction de la taille du maillage.
2. * Etudier les performances de votre code en fonction de la nature des conteneurs STL utilisés.
3. * Etudier les performances de votre code en fonction de la nature des algorithmes STL utilisés.