- TP 1 : Intersections de segments -

Le but de ce TP est d'implémenter l'algorithme par balayage d'intersection de segments vu en cours.
Un code pré rempli en C++ est disponible sur le moodle du cours.

Les points du plan sont repérés par leurs coordonnées supposées entières et appartiennant au domaine $[0,800] \times [0,800]$. L'exécution du programme génére un fichier svg visualisable par un visionneur de document ou un navigateur. Par curiosité, la fonction d'affichage est en fin de code (après le main).

Pensez bien à tester vos différentes fonctions, on peut vite se tromper !

- Exercice 1 - Génération de points du plan -

Commencer par regarder les deux structures de données proposées *point* et *seg*.
Compléter la fonction *void SegmentsAuHasard(int n, seg segments[])* pour générer les segments au hasard dans le plan. Pensez bien à remplir tous les champs des éléments à générer. Bien se référer à l'en-tête de la fonction.

- Exercice 2 - Tests géométriques -

Compléter les fonctions suivantes. Des indications sont données dans les en-têtes.

- *int compareLex(const void * x, const void * y)*
- *bool Intersectent(seg s1, seg s2)*
- *bool compareSeg(seg s1,seg s2)*. Attention aux deux cas à traiter.

La fonction *compareLex* permet de faire tourner la fonction *TriLex* où *qsort* effectue un tri lexicographique des $2n$ points considérés en temps $O(n \log n)$.

- Exercice 3 - Intersection de segments -

Il s'agit maintenant de compléter le code de la fonction *bool Intersection(int n, seg segments[])* pour finir d'implémenter l'algorithme par balayage.

Le conteneur *set* de la STL permet d'obtenir l'implémentation d'un arbre binaire de recherche équilibré et les complexités associées. Un itérateur (un pointeur) permet de parcourir l'ensemble.

Les instructions *segPred=ordre.lower_bound(segCourant) ;* et *segSuiv=ordre.upper_bound (segCourant) ;* permettent d'avoir respectivement un pointeur sur *segCourant* et un sur le segment qui suit *segCourant* dans l'ordre. Pour avoir un pointeur sur l'élément précédant *segCourant*, on fera donc *segPred- -;*

Les bornes du conteneur *ordre* sont données par *ordre.begin()*, qui pointe sur le premier élément de *ordre* et *ordre.end()*, qui pointe sur ce qui suit le dernier élément de *ordre* (un dépassement du conteneur).

Exercices supplémentaires -

- Exercice 4 - Polygone simple -

Ecrire une fonction qui prend en entrée une suite de n segments et qui teste si ces segments forment un polygone simple ou non. Votre fonction devra fonctionner en temps $O(n \log n)$.

- Exercice 5 - Points d'intersection -

Ecrire une fonction qui prend en entrée une suite de n segments et qui imprime toutes les intersections formées par ces segments. Votre fonction devra fonctionner en temps $O((n+k)\log(n+k))$, où k est le nombre d'intersections formées par tous les segments.

- Exercice 6 - Intersection de disques -

Implémenter l'algorithme de recherche d'intersections de disques vu en TD. Dans un premier temps, on ne cherchera qu'à détecter une seule intersection, puis on les trouvera toutes.