Algorithmique avancée - TD4

—o000o——o000o—

Listes circulaires et doublement chaînées

1 Liste croissante

On considère dans cet exercice une liste contenant des éléments de type `Double`.

1. Écrire la méthode `croissante()` qui retourne `true` si et seulement si la liste est triée par ordre croissant au sens large. On implémentera la fonction itérativement.
2. Proposer maintenant une version récursive de la méthode précédente nommée `croissanteRec()`. Vous pourrez considérer les cas de base suivants :
   - une liste vide est considérée comme triée par ordre croissant,
   - une liste avec un seul élément est croissante également ;

2 Liste circulaire

On s'intéresse dans cet exercice à proposer des méthodes pour enrichir la barrière d'abstraction étudiée en cours.

1. Écrire un constructeur `LListCirc(LList L)` qui permet de construire une liste circulaire à partir d'une liste non circulaire.
2. Proposer une méthode `fusion(LListCirc<T> L)` qui permet de fusionner la liste circulaire `L` à la fin de la liste courante.
3. Écrire une méthode `remove(int index)` qui supprimer l'élément à l'indice passé en argument dans la liste circulaire. Tous les cas doivent pouvoir être traités.

3 Liste doublement chaînée

On travaille finalement sur les listes doublement chaînées.

1. On suppose disposer d'une liste doublement chaînée dont les éléments de type `Double` sont triés par ordre croissant. Écrire une méthode `insert(Double elem)` qui permet d'insérer la nouvelle valeur de telle sorte que la liste résultante soit toujours triée par ordre croissant.
2. Écrire une méthode `inverser()` qui produit une liste doublement chaînée dont les éléments sont en ordre inverse de la liste originale.
3. Proposer une méthode `unSurDeux()` qui produit une liste ne contenant qu'un élément sur deux de la liste originale.