TP n°7 : Structures autoréférentes en C++

En cas de difficulté, demandez un indice ou une explication à l'enseignant. Chaque exercice doit faire l'objet d'une **validation expérimentale poussée**.

Exercice 1 (acquisition)

Reprenez le code de la classe **ArbreBin** présentée en cours (ce code est disponible en ligne), et ajoutez les méthodes suivantes :

- **void affichePrefixe()** : affiche les chaînes contenues dans l'arbre en utilisant un parcours en ordre préfixe.
- **int taille()** : retourne le nombre de chaînes contenues dans l'arbre.
- **string[] getStrings()** : retourne un tableau contenant toutes les chaînes stockées dans l'arbre en ordre lexicographique.

Compétences : Structures chaînées et autoréférentes en C++, tableaux 1D.

Exercice 2 (consolidation)

Reprenez l'exercice 1 en ajoutant une méthode :

void retire(ArbreBin* p) : retire la chaîne située à la racine de l'arbre pointé par **p** en maintenant l'arbre trié.

Compétences : Structures chaînées et autoréférentes en C++.

Exercice 3 (acquisition)

Réalisez une classe **ListCh** représentant une liste chaînée de chaînes de caractères. Une classe supplémentaire devra être définie pour représenter les maillon. Le chaînage sera réalisé dans un seul sens (chaînage simple).

Cette classe devra être pourvue d'un constructeur produisant une liste vide, d'une méthode d'ajout en tête, d'une méthode d'ajout en fin de liste (essayez de trouver une solution pour que l'ajout en fin de liste soit rapide), et d'une méthode **toString** retournant toutes les chaînes contenues dans la liste rassemblées en une seule chaînes avec espace de séparation.

Compétences : Structures chaînées et autoréférentes en C++.

Exercice 4 (consolidation)

Ajoutez à la classe de l'exercice 3 un destructeur, un constructeur en copie et une surcharge de l'opérateur d'affectation.

Ajoutez également un curseur (variable d'instance privée) désignant une des chaînes de la liste, une méthode permettant de remettre le curseur en début de liste, une méthode permettant de lire la chaîne désignée par le curseur et une méthode permettant de tester s'il reste encore des chaînes à lire.

Ajoutez une méthode permettant de trier la liste par ordre lexicographique en insérant toutes les chaînes dans un arbre binaire trié puis en récupérant ensuite les chaînes par un parcours de l'arbre en ordre infixe.

Pour faciliter la communication entre les classes **ArbreBin** et **ListeCh**, vous devez ajouter à la classe **ArbreBin** les méthodes suivantes :

- Un constructeur **ArbreBin(ListeCh w)** qui produit un arbre contenant les chaînes de la liste w.
- Une méthode **ListeCh toList()** qui parcours l'arbre en ordre préfixe et retourne la liste des chaînes stockées.

Attention, l'arbre produit et la liste initiale devront être détruit proprement de manière à éviter toute fuite mémoire.

Compétences : Structures chaînées et autoréférentes en C++.