

TP n° 6 de C++

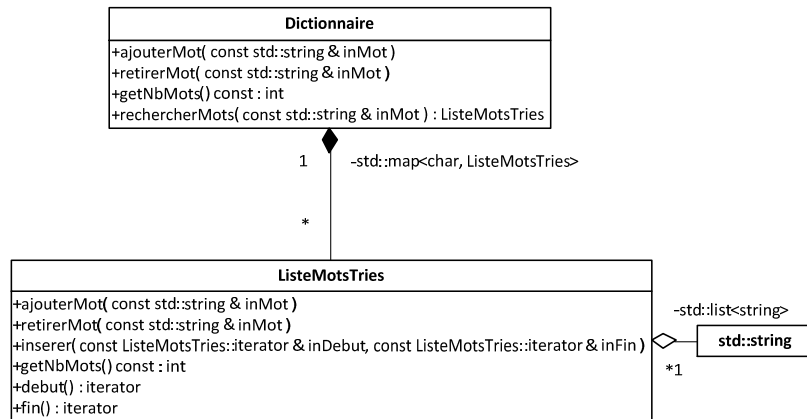
Introduction à la Standard Template Library (STL) Étude d'un conteneur séquentiel et d'un conteneur associatif Étude de foncteurs élémentaires

Consigne générale : Tout au long du TP, le but est d'employer au maximum les algorithmes et conteneurs proposés par la STL. Conservez un navigateur ouvert sur une des documentations de référence pour vous aider.

<http://www.sgi.com/tech/stl/>
<http://www.cplusplus.com/reference/>
<http://www.medini.org/download/stlqr/1.32/stlqr-a4.pdf>

Partie 1 : Dictionnaire

Vous implémenterez dans cet exercice une classe représentant un dictionnaire de manière simplifiée. Le diagramme de classe suivant vous donne la structure de l'application :



1. Implémentez en premier lieu la classe *ListeMotsTries*, il s'agit d'une encapsulation de la classe `std::list<std::string>` de la STL. Les méthodes ne dépasseront donc pas une à deux lignes de code.

Consignes :

- Les mots rajoutés à la classe *ListeMotsTries* sont triés après chaque insertion.
- La méthode *insérer* prend en paramètre un itérateur de début et un itérateur de fin sur une séquence de mots à ajouter.

Ex :

```
ListeMotsTries nomsAnimaux;
std::vector< std::string > nomsOiseaux;
[...] // remplissage du vecteur de noms d'oiseaux
nomsAnimaux.insérer( nomsOiseaux.begin(), nomsOiseaux.end() );
```

- La classe *ListeMotsTries* utilise l'algorithme `std::copy` et l'itérateur `std::ostream_iterator` pour surcharger l'opérateur de flux (`operator<<`).

- Vous veillerez à définir des itérateurs sur la classe *ListeMotsTries* en réutilisant ceux fournis dans la classe `std::list` de la STL (*typedef*).

2. La classe *Dictionnaire* est composée d'une instance de `std::map<char, ListeMotsTries>`. Le premier champ est la clé, il s'agit de la première lettre d'un mot. La valeur qui lui est associée est une liste de mots (instance de *ListeMotsTries*) qui stocke par ordre alphabétique tous les mots commençant par la dite lettre.

Consignes :

- Implémentez les méthodes de base telles que *ajouterMot* ou *supprimerMot*.
- La méthode *rechercherMots*, renvoie la liste de tous les mots commençant par le motif qu'on lui passe en paramètre. L'une des méthodes *compare* de la classe `std::string` vous aidera à fournir cette fonctionnalité.

3. Vérifiez à présent le bon fonctionnement de votre dictionnaire :

- Testez la recherche d'un mot qui n'existe pas.
- Vérifiez que vous avez compris la gestion d'une map en proposant les fonctions nécessaires pour utiliser cette surcharge :

```
std::ostream & operator<< ( std::ostream & inO, const Dictionnaire& inD );
```

Cet opérateur **devra obligatoirement utiliser l'algorithme `std::copy`** !

Partie 2 : Les Foncteurs

Le but de cette partie est de proposer une initiation aux foncteurs.

Foncteurs de base

1. En vous inspirant de ce que vous avez vu en cours avec la génération des nombres pairs, proposez un foncteur *Rand_0_100* qui génère des nombres aléatoires entre 0 et 100.

- Utiliser la fonction `std::generate` pour peupler un vecteur (`std::vector`).
- Proposer une autre solution à l'aide de la fonction `std::generate_n` et d'un `back_inserter` afin d'éviter l'initialisation inutile de tous les éléments du vecteur.

2. Utiliser la fonction `std::accumulate` pour calculer facilement la moyenne de l'échantillon.

3. Généraliser maintenant le foncteur pour qu'il génère des nombres aléatoires entre deux valeurs passées en paramètre du foncteur.

4. Nous désirons maintenant afficher la liste des nombres dits de « Fibonacci » (http://fr.wikipedia.org/wiki/Suite_de_Fibonacci). La valeur du $n^{\text{ième}}$ élément est égale à la somme des deux éléments précédent : $u_n = u_{n-1} + u_{n-2}$ avec $u_0 = 0$ et $u_1 = 1$. Pour cela, vous devrez créer un foncteur permettant de sauvegarder l'état de manière à ce qu'à chaque appel, un nouveau nombre de Fibonacci soit généré.

L'affichage sera réalisé, encore une fois, à l'aide de la fonction `std::copy` et d'un `std::ostream_iterator`.

Pour aller plus loin : utilisation de foncteurs avec des conteneurs

5. Créer un foncteur permettant de réaliser le tri sur un vecteur de string (`std::vector< std::string >`) à l'aide de la fonction `std::sort` en ne comparant pas l'ensemble de la chaîne mais seulement à partir du second caractère (on supposera que le vecteur contient uniquement des chaînes d'au moins deux caractères). Tester votre foncteur avec un jeu d'essai simple.

6. Créer un dernier foncteur permettant, à l'aide de la fonction `std::for_each`, de mettre en majuscule l'ensemble des chaînes de caractères contenu dans un vecteur de string. Tester votre foncteur.