TD1

Dans tout ce TD, on considère que quand on parle de rang, la numérotation commence à 0 comme c'est le cas pour les tableaux en langage C++.

Utilisez l'éditeur et le compilateur C++ de votre choix. Un exemple d'éditeur, compilateur et débogueur gratuit est Code::Blocks disponible à http://sourceforge.net/projects/codeblocks/files/Binaries/20.03/Windows/codeblocks-20.03mingw-setup.exe.

L'aide en ligne du langage C++, incluant la STL, est disponible ici : https://en.cppreference.com/w/cpp. Elle est très bien faite, et propose de nombreux exemples. Privilégiez la version anglaise qui est plus complète et propose plus d'exemples par rapport à la version française.

Exercice 1

Créer un nouveau projet.

Créer une classe "ListIntsV" ayant en donnée membre une instance du patron `std::vector` avec le type "int".

Cette classe devra définir les méthodes suivantes :

- Le constructeur par défaut.
- Le constructeur par recopie.
- Le destructeur.
- "insertBeg" : Insérer l'élément passé en paramètre au début.
- "getSize" : Retourne le nombre d'éléments.
- "removeAt" : Supprimer l'élément dont le rang est passé en paramètre.

Pour chaque méthode, préciser la complexité : « O (N) »…

Ecrire un programme principal qui :

- Crée une instance de la classe "ListIntsV".
- Ajoute 100 000 d'entiers de 0 à 99 999, l'ajout se faisant à chaque fois en début de liste. Afficher le temps passé pour cela. Selon vous, après ces ajouts, dans quel ordre seront ces valeurs dans la liste ?
- Supprime les 1000 éléments du rang 50000 au rang 50999. Afficher le temps passé pour cela.

Exemple pour mesurer le temps :

```
#include <time.h>   // clock ()
…
int iBeg = clock ();
// Partie du code pour laquelle mesurer le temps d'exécution.
int iEnd = clock ();
cout << "Time consumed (in seconds): " << (double) (iEnd - iBeg) / CLOCKS_PER_SEC << endl;
```

Exercice 2

Créer un nouveau projet et implémenter la classe "ListIntsL" dont le fonctionnement est identique à celle écrite à l'exercice précédent mais utilisant en interne le patron `std::list` au lieu de `std::vector`. Mettre à jour la complexité de chaque méthode.

Notez que les itérateurs sur `std::list` n'acceptent pas les opérateurs "+" ni "-". Si besoin, vous pouvez utiliser `std::distance`, `std::advance`.

Construisez un programme principal similaire à celui de l'exercice précédent.

Comparer les temps d'exécution. En déduire quelle est la meilleure structure de données pour chacune des étapes du programme principal.