INF1015 - Programmation orientée objet avancée
Travail dirigé No. 5

14-Complexité et conteneur non contigu,
15-Bibliothèque de structures de données et algorithmes

Objectifs :	Permettre à l'étudiant de se familiariser avec les différents conteneurs, algorithmes et leur complexité.
Durée :	Deux semaines de laboratoire.
Remise du travail :	Avant 23h30 le mercredi 2 avril 2025.
Travail préparatoire :	Avoir un TD4 fonctionnel, et lecture de l'énoncé.
Documents à remettre :	sur le site Moodle des travaux pratiques, vous remettrez l'ensemble des fichiers .cpp et .hpp compressés dans un fichier .zip en suivant la procédure de remise des TDs.

Directives particulières

- Ce TD est une suite du TD4, il reprend votre solution finale du TD4. Après la date de remise du TD4, nous pourrons fournir un solutionnaire pour que vous puissiez corriger votre TD4 ou prendre notre solutionnaire du TD4 comme point de départ s'il est plus difficile de rendre fonctionnel votre code de TD4.
- Vous pouvez ajouter d'autres fonctions/méthodes et structures/classes, pour améliorer la lisibilité et suivre le principe DRY (Don't Repeat Yourself).
- Il est interdit d'utiliser les variables globales; les constantes globales sont permises.
- Vous devez éliminer ou expliquer tout avertissement de « build » donné par le compilateur (avec /W4).
- Respecter le guide de codage, les points pertinents pour ce travail sont donnés en annexe à la fin.
- N'oubliez pas de mettre les entêtes de fichiers (guide point 33).

On ne demande pas d'implémenter la copie d'item polymorphe, les copies dans les différents conteneurs seront toujours des pointeurs non copropriétaires (pour pourvoir avoir des unique_ptr dans le conteneur original du TD4; ça fonctionne aussi si vous avez des shared_ptr).

Dans les algorithmes, on considère que la longueur des titres est bornée (est O(1)).

imap de cppitertools/imap.hpp permet la syntaxe « pipe » (la barre verticale | se lit « pipe » en anglais dans ce contexte) pour passer chaque élément d'un conteneur dans une fonction, à mesure qu'on itère sur intervalle, avant de conserver le résultat de cette fonction dans la variable d'itération. Exemple : `for (int v : range(10) | imap([](int x) { return x*x; })) cout << v;` affiche les carrés 0 1 4 9 16 25 36 49 64 81. Similairement avec la bibliothèque standard C++20 : `for (int v : views::iota(0, 10) | views::transform([](int x) { return x*x; })) cout << v;` Attention que cppitertools a été fait pour C++17 et n'est actuellement pas compatible avec les versions ranges:: C++20, donc on peut difficilement mélanger les deux dans une même expression (views::iota(…) | imap(…) fonctionne, mais range(…) | views::transform(…) ne compile pas).

Travail à effectuer :

0. Faire une version template de afficherListeItems pour qu'il fonctionne avec d'autres conteneurs comme forward_list en plus de vector, mais qui affiche uniquement le titre de l'item et le réalisateur et/ou l'auteur des films/livres, pour avoir des affichages plus courts mais qu'on voit que l'affichage polymorphe fonctionne encore (modifiez l'interface Affichable pour permettre un affichage court, cet affichage court n'a pas besoin de supporter l'opérateur<< ; il est aussi permis de simplement enlever les informations non pertinentes dans ce TD de l'affichage déjà présent, au lieu d'avoir deux modes d'affichage). Exemple :
   Alien, par Ridley Scott
   …

Le avventure di Pinocchio, de Carlo Collodi
Le Hobbit : La Bataille des Cinq Armées, par Peter Jackson, de J. R. R. Tolkien

Vous pouvez mettre public les attributs des classes de données sans invariant (Item, Film, Livre) qui avaient été forcés private pour vous aider à bien séparer le code entre les classes dans le TD précédent.

1. Liste liée et itérateurs

1.1 Copier les pointeurs du vecteur d'items final de la biliothèque dans une forward_list (liste liée simple similaire à celle du chap.14) dans l'ordre original (même ordre que dans le vecteur), en O(n). Attention que forward_list n'a pas de push_back (voir https://en.cppreference.com/w/cpp/container/forward_list).

1.2 Copier la liste qui est en ordre original à l'envers (le dernier item se retrouve en premier) en O(n) dans une autre forward_list sans passer par un conteneur intermédiaire ni avec une fonction récursive (fonction qui fait appel à elle-même directement ou indirectement) ni en utilisant « reverse » qui le fait automatiquement.

1.3 Copier la liste qui est en ordre original dans le même ordre qu'elle est, en O(n) dans une autre forward_list sans passer par un conteneur intermédiaire ni avec une fonction récursive (fonction qui fait appel à elle-même directement ou indirectement) ni en utilisant la copie intégrée à forward_list (constructeur de copie ou operator=).

1.4 Copier la liste qui est en ordre original à l'envers dans un vector avec les mêmes contraintes que ci-dessus (sans conteneur intermédiaire ni fonction récursive). Tentez d'optimiser l'ordre O(…) de votre algorithme et indiquez cet ordre en commentaire.

1.5 Ajoutez ce qu'il faut pour pouvoir itérer directement sur une liste : for (auto&& acteur : film.acteurs) (ou film.getActeurs() si vous avez un accesseur qui retourne les ListeActeurs, pas un span qui a déjà ce qu'il faut pour l'itérer de cette manière). Itérez en utilisant cette manière pour afficher les acteurs du premier film (Alien) .

2. Conteneurs

2.1 Utilisez un conteneur, pas un algorithme de tri (pas « sort », « qsort » …), pour avoir les items en ordre alphabétique. Affichez ces items. Encore, on utilise des pointeurs non propriétaires pour ne pas avoir à faire de copies d'items tel que dit au début de l'énoncé.

2.2 Utilisez un conteneur qui, après une étape initiale, va permettre de trouver des items par titre en O(1) en moyenne pour chaque item cherché. Affichez l'item « The Hobbit ».

3. Algorithmes

3.1 Utilisez l'algorithme copy_if ou copy pour copier les items qui « sont des » Film de la liste faite en 1.1 vers un vector, en gardant l'ordre, en une ligne de programme. Vous avez le droit d'utiliser les versions ranges:: C++20 de ces algorithmes ou d'écrire une fonction d'adaptation, d'une ligne, qui permet de passer un « range » au lieu de deux itérateurs à ces algorithmes (similairement à ce qu'il y a dans ranges::).
Aide : vous pouvez commencer par écrire les différentes parties de la ligne sur plusieurs lignes pour aider à trouver vos erreurs.

3.2 Faites la somme des recettes des films trouvés ci-dessus, en une ligne (pas de « for »).

ANNEXE 1 : Utilisation des outils de programmation et débogage.

Utilisation des avertissements :
Avec les TD précédents vous devriez déjà savoir comment utiliser la liste des avertissements. Pour voir la liste des erreurs et avertissements, sélectionner le menu Affichage > Liste d'erreurs et s'assurer de sélectionner les avertissements. Une recompilation (menu Générer > Compiler, ou Ctrl+F7) est nécessaire pour mettre à jour la liste des avertissements de « build ». Pour être certain de voir tous les avertissements, on peut « Regénérer la solution » (menu Générer > Regénérer la solution, ou Ctrl+Alt+F7), qui recompile tous les fichiers.

Votre programme ne devrait avoir aucun avertissement de « build » (les avertissements d'IntelliSense sont acceptés). Pour tout avertissement restant (s'il y en a) vous devez ajouter un commentaire dans votre code, à l'endroit concerné, pour indiquer pourquoi l'avertissement peut être ignoré.

Rapport sur les fuites de mémoire et la corruption autour des blocs alloués :
Le programme inclus des versions de débogage de « new » et « delete », qui permettent de détecter si un bloc n'a jamais été désalloué, et afficher à la fin de l'exécution la ligne du programme qui a fait l'allocation. L'allocation de mémoire est aussi configurée pour vérifier la corruption lors des désallocations, permettant d'intercepter des écritures hors bornes d'un tableau alloué.

Utilisation de la liste des choses à faire :
Le code contient des commentaires « TODO » que Visual Studio reconnaît. Pour afficher la liste, allez dans le menu Affichage, sous-menu Autres fenêtres, cliquez sur Liste des tâches (le raccourci devrait être « Ctrl \ t », les touches \ et t faites une après l'autre). Vous pouvez double-cliquer sur les « TODO » pour aller à l'endroit où il se trouve dans le code. Vous pouvez ajouter vos propres TODO en commentaire pendant que vous programmez, et les enlever lorsque la fonctionnalité est terminée.

Utilisation du débogueur :
Lorsqu'on a un pointeur « ptr » vers un tableau, et qu'on demande au débogueur d'afficher « ptr », lorsqu'on clique sur le + pour afficher les valeurs pointées il n'affiche qu'une valeur puisqu'il ne sait pas que c'est un tableau. Si on veut qu'il affiche par exemple 10 éléments, il faut lui demander d'afficher « ptr,10 » plutôt que « ptr ».

Utilisation de l'outil de vérification de couverture de code :
Suivez le document « Doc Couverture de code » sur le site Moodle.

Annexe 2 : Points du guide de codage à respecter

Les points du **guide de codage** à respecter **impérativement** pour ce TD sont :
(voir le guide de codage sur le site Moodle du cours pour la description détaillée de chacun de ces points)

Mêmes points que le TD3 :

2 : noms des types en `UpperCamelCase`
3 : noms des variables en `lowerCamelCase`
5 : noms des fonctions/méthodes en `lowerCamelCase`
7 : noms des types génériques, une lettre majuscule ou nom référant à un concept
8 : préférer le mot `typename` dans les template
15 : nom de classe ne devrait pas être dans le nom des méthodes
21 : pluriel pour les tableaux (`int nombres[];`)
22 : préfixe *n* pour désigner un nombre d'objets (`int nElements;`)
24 : variables d'itération `i`, `j`, `k` mais jamais `l`, pour les indexes
27 : éviter les abréviations (les acronymes communs doivent être gardés en acronymes)
29 : éviter la négation dans les noms
33 : entête de fichier
42 : `#include` au début
44,69 : ordonner les parties d'une classe public, protected, private
46 : initialiser à la déclaration
47 : pas plus d'une signification par variable
48 : aucune variable globale (les constantes globales sont tout à fait permises)
50 : mettre le & près du type
51 : test de 0 explicite (`if (nombre != 0)`)
52, 14 : variables vivantes le moins longtemps possible
53-54 : boucles `for` et `while`
58-61 : instructions conditionnelles
62 : pas de nombres magiques dans le code
67-78, 88 : indentation du code et commentaires
83-84 : aligner les variables lors des déclarations ainsi que les énoncés
85 : mieux écrire du code incompréhensible plutôt qu'y ajouter des commentaires