# INF1015 - Programmation orientée objet avancée
# Travail dirigé No. 4

11-Méthodes virtuelles et classes abstraites, 12-Conversion d'objets,
13-Héritage multiple

---

| | |
|---|---|
| **Objectifs :** | Permettre à l'étudiant de se familiariser avec les méthodes virtuelles, les objets polymorphes, l'héritage simple et multiple. |
| **Durée :** | Deux semaines de laboratoire. |
| **Remise du travail :** | Avant 23h30 le dimanche 20 mars 2022. |
| **Travail préparatoire :** | Avoir un TD3 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 TD3, il reprend votre solution finale du TD3. Après la date de remise du TD3, nous pourrons fournir un solutionnaire pour que vous puissiez corriger votre TD3 ou prendre notre solutionnaire du TD3 comme point de départ s'il est plus difficile de rendre fonctionnel votre code de TD3.
- 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 pouvez maintenant utiliser std::vector.
- 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).

---

`vector`

Maintenant qu'on sait faire l'allocation dynamique... vous pouvez utiliser vector, qui s'occupe de la réallocation du tableau. vecteur.push_back(element) est l'équivalent C++ de liste.append(element) en Python; on peut aussi utiliser vecteur.resize(taille) pour changer la taille (met des objets initialisés avec le constructeur par défaut dans les cases supplémentaires); le pointeur brut du début du tableau est vecteur.data(). La documentation sur cppreference.com est toujours utile.
(Noter qu'on ne vous demande pas de tout remplacer par de vector.)

**Travail à effectuer :**

1. Nouvelles structures/classes

On veut mettre dans une même liste des Film et des Livre, pour faire une bibliothèque.
Item, contient : Titre, Année
Film est un Item + Réalisateur, Recettes M$, acteurs
Livre est un Item + Auteur, Millions de copies vendues, nombre de pages

Vous pouvez commencer par faire l'héritage avec les champs publics pour voir que le programme fonctionne encore alors que la déclaration de Film a changée.

2. Construction de la bibliothèque

Faites un vecteur (vector) pour y placer les items de la bibliothèque; transférez-y les films de la listeFilms (listeFilms ne sera pas utile après et vous pouvez enlever tout ce qu'il y a dans le « main » après, en vous assurant qu'il n'y a pas de fuite de mémoire) et ajoutez-y les livres.

Les livres se trouvent dans un fichier **livres.txt** où les champs sont dans le même ordre que dans la description des attributs de Livre ci-dessus. Pour lire un champ texte entre guillemets qui contient des espaces, on utilise : `fichier >> quoted(uneVariableString);`

3. Affichage de la bibliothèque

On veut pouvoir afficher les items; on aimerait que Affichable soit une interface que supportent les Item.
Vous avez le droit de déplacer l'année dans le format d'affichage pour que les Livre et Film ait le même format d'affichage de base (pouvoir étendre la méthode en utilisant celle de la classe de base). On veut que l'affichage des items fonctionne avec `cout << unItem` (ou `*unItem`, selon).
Changez `afficherListeFilms` (ou votre fonction équivalente) pour afficher une liste d'items (le vecteur).

4. Un item combo FilmLivre

Prenez le film Le Hobbit dans le vecteur et le livre qui correspond. Prenez ces items comme un Film et un Livre, en vous assurant qu'ils ont bien les bons types, pour appeler le constructeur de FilmLivre en lui passant ces deux parties, et placez ce nouvel item dans la liste. FilmLivre hérite à la fois de Film et Livre, utilisant le titre et année du Film.

Tous les items devraient pouvoir bien s'afficher.

# 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