INF1010

Programmation Orientée-Objet

Travail pratique #5
Classes génériques et la STL

Objectifs :	Permettre à l'étudiant de se familiariser avec le concept de fonctions et des classes génériques, aux foncteurs, aux conteneurs et algorithmes de la STL
Remise du travail :	Mardi 19 Novembre 2019, 8h
Références :	Notes de cours sur Moodle & Chapitres 11 à 14, 16 et 20 du livre Big C++ 2e éd.
Documents à remettre :	Les fichiers .cpp et .h complétés, les questions en pdf le tout sous la forme d'une archive au format .zip.
Directives :	Directives de remise des Travaux pratiques sur Moodle Les travaux dirigés s'effectuent obligatoirement en équipe de deux personnes faisant partie du même groupe. Veuillez suivre le guide de codage

Veuillez lire attentivement tout l'énoncé avant de commencer à écrire du code, afin de vous assurer de bien comprendre l'interaction entre les classes. Cela vous permettra aussi de commencer par ce qui vous semble le plus pertinent, car en effet l'ordre de description des classes dans le TP n'est pas forcément l'ordre à suivre. Il pourrait par exemple être plus intéressant de faire toutes les méthodes liées à l'insertion d'une donnée pour pouvoir les tester et continuer par la suite.

Notez que ce TP est sans doute le plus difficile de la session, vous avez 3 séances pour le réaliser, attention à ne pas prendre de retard et de bien répartir la charge de travail jusqu'au 19 Novembre. Assurez-vous aussi de bien comprendre le fonctionnement des templates et leur utilité pour créer des classes génériques. En complément au cours et la documentation de la STL, vous pouvez consulter ce lien.

Le travail effectué dans ce TP continue celui amorcé par les TP 1, 2, 3 et 4, soit l'application AirPoly en y intégrant les notions de foncteurs, classes génériques, conteneurs et algorithmes de la STL.

Les foncteurs sont des objets qui peuvent agir comme des fonctions grâce à la surcharge de l'opérateur « () ». Les foncteurs sont très utiles lorsqu'on travaille avec des structures de données (conteneurs) provenant de la bibliothèque STL comme une list ou une map... Les foncteurs peuvent avoir aucun ou plusieurs attributs. Généralement, un foncteur sans attribut permet de manipuler les objets sur lesquels il est appliqué. Les foncteurs possédant un attribut sont souvent utilisés à des fins de comparaison, mais ont aussi d'autres utilités.

L'utilisation du **static_cast** est **INTERDITE**. Tout le TP est réalisable sans cette fonction.

Pour vous aider, les fichiers du TP précédent avec certaines modifications vous sont fournis. Vous n'avez qu'à implémenter les nouvelles méthodes décrites plus bas. Les attributs ou méthodes qui ne sont plus nécessaires ont été supprimés. Et les méthodes à modifier vous ont été indiquées.

ATTENTION : Vous serez pénalisés pour les utilisations inutiles du mot-clé *this*. Utilisez-le seulement où nécessaire.

ATTENTION : Il est fortement recommandé d'utiliser les fichiers fournis, plutôt que de continuer avec vos fichiers du TP4.

Remarque : Pour plus de précision sur le travail à faire et les changements à effectuer, veuillez-vous référer aux fichiers .h.

Remarque : Vous avez le droit d'utiliser le mot clé « auto » pour simplifier votre code.

- **Vous serez pénalisés pour les utilisations inutiles du mot-clé *this*. Utilisez-le seulement où nécessaire.**
- **Vous devez tirer parti du polymorphisme. L'utilisation de solutions alternatives (attribut type_, typeof, static_cast, etc.) sera jugée hors-sujet et sanctionnée.**
- Vous devez tirer parti des fonctions proposées par la STL.
- **Vous devez ajouter des destructeurs, surcharger l'opérateur = et ajouter un constructeur par copie chaque fois que cela vous semble pertinent. N'en ajoutez pas si cela ne vous semble pas pertinent.**
- **Il faut utiliser le mot clé const chaque fois que cela vous semble pertinent.**
- **L'affichage du programme doit être aussi proche que possible que celui présenté en annexe.**
- **Il faut ajouter des commentaires aux méthodes jugées complexes**
- Vous ne devez pas modifier les signatures des méthodes fournies.

ATTENTION : L'utilisation de boucles for ou while de la forme for (int i; i <vec.size(); i++) est interdit pour les nouvelles méthodes que vous devez implémenter. Vous devez soit utiliser les algorithmes lorsque possible, soit utiliser les boucles for/while en utilisant les itérazateurs.

Travail à réaliser

Lisez la description des classes ci-dessous et suivez aussi les indications des **TODO** insérés dans le code pour apporter vos modifications au code fourni.

Si une méthode d'affichage ne respecte pas le format de la capture donnée en annexe, vous devez modifier les méthodes en question.

Classes *GestionnaireGenerique*

Cette classe permet la création d'un gestionnaire générique qui permet de manipuler plusieurs données d'un même `typename.` Pour chaque type générique, on a :

- `T` : (`pair<string, Membre*>` ou `Coupon*`) où la paire représenterait le nom du membre, et un pointeur vers ce membre.
- `C` (`map<String, Membre*>` ou `vector<Coupon*>`).
- `typename FoncteurAjouter : (AjouterCoupon ou AjouterMembre)`. La méthode d'ajout est différente pour chaque conteneur nous allons utiliser des foncteurs d'ajout de

Attributs :

- (protected) `C conteneur_` : Le conteneur qui contiendra les éléments du gestionnaire de type C.

Méthodes :

- `C getConteneur() const`**:** retourne le conteneur_ du gestionnaire.
- `void ajouter(T t)` **:** prend en paramètre un `typename T` qui devra ajouter cet objet dans le conteneur à l'aide d'un foncteur d'ajout `FoncteurAjouter` qui changera en fonction du type de conteneur. Cette méthode n'a pas de valeur de retour.
- `int` getNombreElements() `const :` retourne le nombre d'éléments dans conteneur_.

Classe *GestionnaireCoupons*

La classe GestionnaireCoupons est extraite de la classe Gestionnaire du TP4. Cette classe est fournie.

Il vous faudra la modifier afin qu'elle hérite de la classe GestionnaireGenerique et de définir les types génériques correspondant à la classe.

Attributs :

La classe GestionnaireCoupons ne doit plus contenir d'attributs puisque ceux-ci sont dans la classe GestionnaireGenerique.

Méthodes :

La modification apportée par rapport au TP4 est que la méthode prend désormais en paramètre un `Membre*` et non plus le nom du Membre.

Méthodes à modifier :

AUCUNES.

Classe *GestionnaireMembres*

La classe GestionnaireMembres est extraite de la classe Gestionnaire du TP4.

Il vous faudra la modifier afin qu'elle hérite de la classe GestionnaireGenerique afin de définir les types génériques correspondant à la classe.

La classe GestionnaireMembres devra gérer un conteneur de type map de la STL. Sa clé sera un nom d'utilisateur et l'élément qu'elle permettra d'accéder sera un Membre* (`map<string, Membre*>`).

Attributs :

La classe GestionnaireMembres ne doit plus contenir d'attributs puisque ceux-ci sont dans la classe GestionnaireGenerique.

Méthodes :

- `void assignerBillet()` : Elle vous est donnée. Contrairement aux précédents TP, la méthode ne prend plus en paramètre un booléen pour indiquer s'il faut utiliser un coupon. Mais reçoit directement le montant du rabais (s'il y a lieu, 0 sinon)

Méthodes à modifier:

Les méthodes à extraire de la classe Gestionnaire sont :

- `double calculerRevenu() const` : À l'aide de la STL, changer l'implémentation pour optimiser la fonction et supprimer la forme `for(int i =0; i<size();++i)`.
- `int calculerNombreBilletsEnSolde() const` : À l'aide de la STL, changer l'implémentation pour optimiser la fonction et supprimer la forme `for(int i =0; i<size();++i)`.
- `Billet* getBilletMin() const` : À l'aide de la STL, retourne le billet avec le prix le moins élevé pour un client.

- `Billet* getBilletMax() const` : À l'aide de la STL, retourne le billet avec le prix le plus élevé pour un client.
- `vector<Billet*> trouverBilletParIntervalle(Membre* membre, double prixInf, double prixSup) const` : À l'aide de la STL, retourne le(s) billet(s) avec un prix compris dans l'intervalle
- `void afficher() const` : À l'aide de la STL, changer l'implémentation pour optimiser la fonction et supprimer la forme `for(int i =0; i<size();++i).`

Classe *Billet*

Dans cette classe, il vous suffit d'adapter l'implémentation en ajoutant `operator<<` afin de permettre d'utiliser la std ::copy dans un steam plus tard dans la classe Membre.

Méthode à ajouter :

- `operator<<` cette méthode doit appeler la méthode afficher de billet.

Classe *Membre*

Dans cette classe, il faut adapter les méthodes existantes à la STL.

Méthode à modifier :

- `Constructeur de copie :` Pour cette méthode, il faut utiliser un std::copy
- Destructeur : à adapter avec les itérateurs.
- `trouverBillet()` : cette méthode doit utiliser la méthode appropriée de la STL pour trouver le billet en paramètre. Une fois trouvé, retourner l'itérateur qui pointe sur le billet trouvé.
- `utiliserBillet() :` cette méthode doit utiliser la méthode `trouverBillet`. La suppression du billet doit être adapté avec la méthode approprié de la STL.
- `Opérateur=() :` cet opérateur adaptée avec les itérateurs et la fonction std::copy.
- `Afficher() :` cette méthode doit utiliser std ::copy afin de faire l'affichage des billets.

Classe *MembreRegulier*

Dans cette classe, il faut adapter les méthodes existantes à la STL.

Méthode à modifier :

- `operator-= :` Il faut adapter la suppression avec les itérateurs. Utilisé la bonne méthode de la STL pour la suppression.
- `Afficher()` : il faut utiliser std ::copy afin d'afficher les coupons

Classe *Coupon*

Dans cette classe, il faut ajouter l'opérateur << afin de permettre l'utilisation de std::copy pour l'affichage dans la classe MembreRegulier.

Méthode à ajouter :

- `operator<<` cette méthode doit appeler la méthode afficher de Coupon

Foncteur.h

Ce fichier comporte les différents foncteurs utilisés dans le TP. Les foncteurs doivent tous être implémentés uniquement dans ce fichier.

AjouterCoupon

Ce foncteur permet l'ajout d'un Coupon dans un conteneur de type **vector**.

Attribut :

- **conteneur_ :** de type `vector<Coupon*>` est une agrégation par référence et initialisé par paramètre (par référence) dans le constructeur.

Méthodes :

- **AjouterCoupon(vector<Coupon*>& conteneur)** : Constructeur par paramètres qui initialise le conteneur de la classe avec le vecteur passé en paramètres.
- **operator(Coupon* coupon) :** prend en paramètre un pointeur de Coupon, après avoir vérifié que le coupon n'est pas déjà dans le conteneur_, ajouter le coupon et retourner le vecteur avec le nouveau coupon.

AjouterMembre

Ce foncteur permet l'ajout d'un Membre dans un conteneur de type **map**.

Attribut :

- **conteneur_ :** de type `map<string, Membre*>` est une agrégation par référence et initialisé par paramètre (par référence) dans le constructeur.

Méthodes :

- **AjouterMembre(map<String, Membre*>& conteneur)** : Constructeur par paramètres qui initialise le conteneur de la classe avec le vecteur passé en paramètres.

- **operator(pair<string, Membre*> membre) :** prend en paramètre une `pair<string, Membre*>` qui correspond respectivement au nom du membre et au Membre en question

IntervallePrixBillet

Ce foncteur est utilisé par le GestionnaireMembre afin de trouver les billets d'un membre qui ont un prix dans un interval compris entre borneInf_ et borneSup_.

Attributs :

- **borneInf_ :** est un double initialisé par paramètre dans le constructeur.
- **borneSup_ :** est un double initialisé par paramètre dans le constructeur.

Méthodes :

- **IntervallePrixBillet(double borneInf, double borneSup)** : Constructeur par paramètres qui permet d'initialiser les attributs borne supérieure et inférieure de la classe.
- **operator(Billet* billet) :** prend en paramètre une `Billet*` et retourne vrai si le compte associé à le prix du billet est compris entre les bornes borneInf_ et borneSup_, faux sinon.

Main.cpp

Pour ce TP le *main* est entièrement fourni avec des tests comme pour le TP précédent, n'apportez **aucune modification à celui-ci (sauf demandé voir dans le fichier main.cpp avec les TODO)** et répondez aux questions cette fois-ci dans un document PDF que vous ajouterez à l'archive de votre remise.

Si certains tests ne fonctionnent pas, regardez attentivement dans le main celui qui ne passe pas et ajustez votre code en conséquence. Pendant la correction, nous remplacerons votre main par le nôtre afin de prévenir toutes modifications de votre part.

Spécifications générales

• Ajoutez un destructeur pour chaque classe chaque fois que cela vous semble pertinent.
• Utilisez la liste d'initialisation pour l'implémentation de vos constructeurs
• Ajoutez le mot-clé const chaque fois que cela est pertinent
• Appliquez un affichage similaire à ce qui est présenté à la fin de ce document.
• Documentez votre code source.
• Répondez aux questions dans un PDF que vous joindrez à l'archive zip de votre remise. **Attention, les réponses rédigées dans le main ne seront pas corrigées.**

Questions

1. Quel est l'avantage d'utiliser une linked-list au lieu d'un vecteur si on veut supprimer ou rajouter un élément ? Quelle structure de données est plus avantageuse si on veut accéder à des éléments à des positions aléatoires ?

2. Pourquoi est-ce que l'implémentation des classes génériques est dans .h et non pas séparée en .h et .cpp comme les classes normales ?

Corrections

La correction du TP se fera sur 20 points.

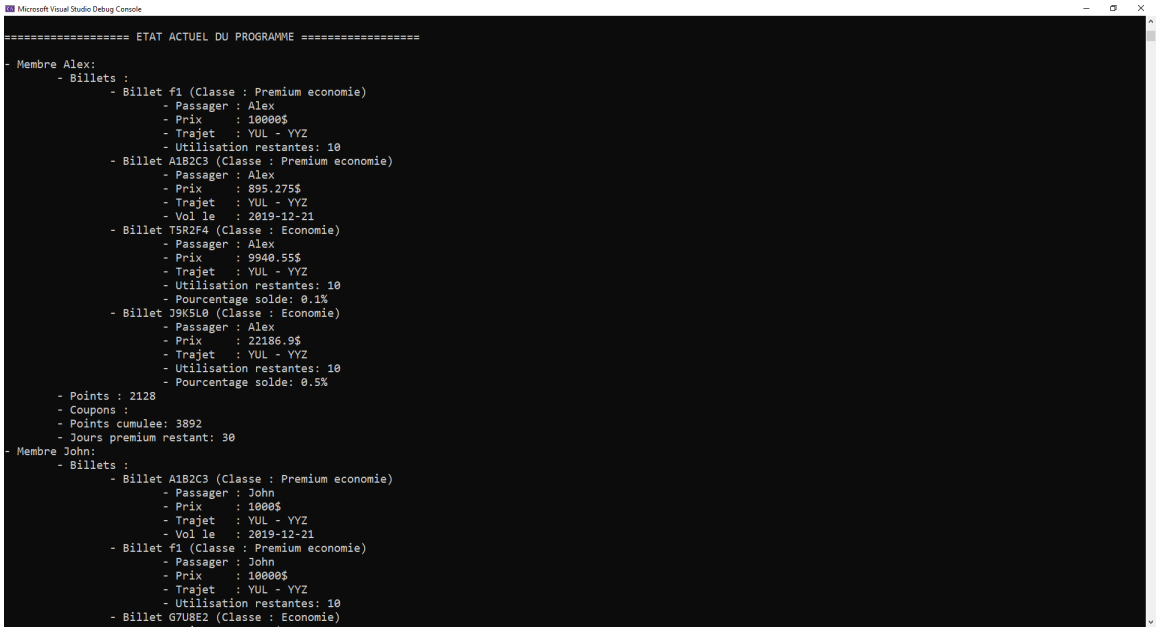
Voici les détails de la correction :

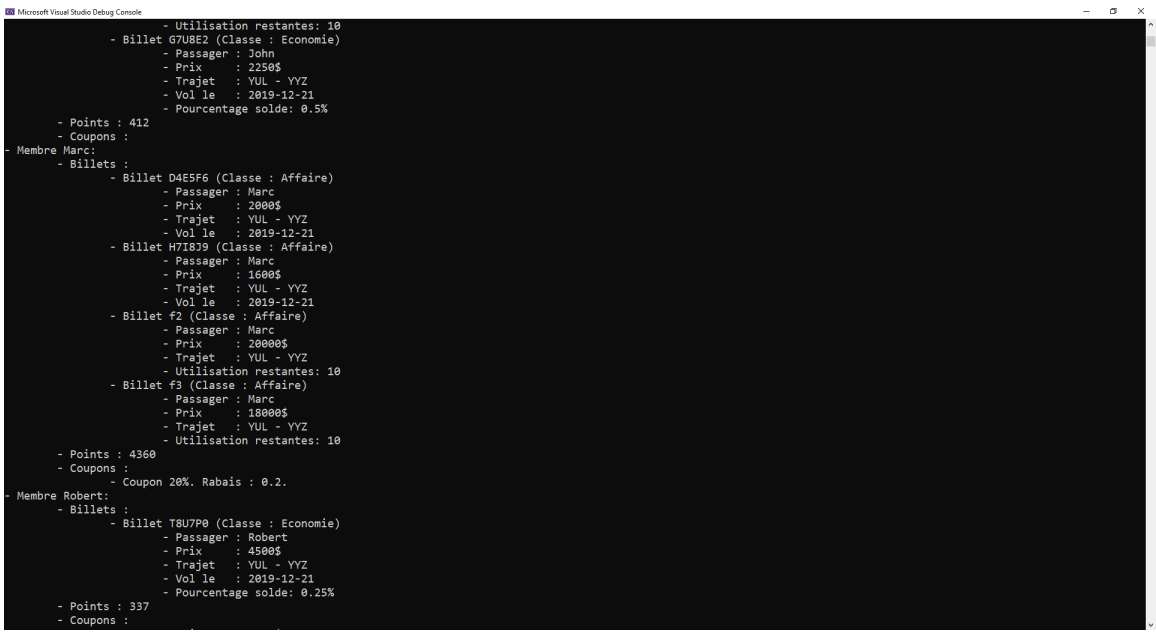
- (4 points) Compilation du programme ;
- (3 points) Exécution du programme ;
- (5 points) Comportement exact des méthodes du programme et l'utilisation de la STL;
- (4 points) Utilisation adéquate des foncteurs;
- (2 points) Documentation du code ;
- (2 point) Réponse aux questions.

Affichage attendu

Les tests fournis dans le main

et


Microsoft Visual Studio Debug Console
================== ETAT ACTUEL DU PROGRAMME ==================
- Membre Alex:
- Billets :
- Billet f1 (Classe : Premium economie)
- Passager : Alex
- Prix : 10000$
- Trajet : YUL - YYZ
- Utilisation restantes: 10
- Billet A1B2C3 (Classe : Premium economie)
- Passager : Alex
- Prix : 895.275$
- Trajet : YUL - YYZ
- Vol le : 2019-12-21
- Billet T5R2F4 (Classe : Economie)
- Passager : Alex
- Prix : 9940.55$
- Trajet : YUL - YYZ
- Utilisation restantes: 10
- Pourcentage solde: 0.1%
- Billet J9K5L0 (Classe : Economie)
- Passager : Alex
- Prix : 22186.9$
- Trajet : YUL - YYZ
- Utilisation restantes: 10
- Pourcentage solde: 0.5%
- Points : 2128
- Coupons :
- Points cumulee: 3892
- Jours premium restant: 30
- Membre John:
- Billets :
- Billet A1B2C3 (Classe : Premium economie)
- Passager : John
- Prix : 1000$
- Trajet : YUL - YYZ
- Vol le : 2019-12-21
- Billet f1 (Classe : Premium economie)
- Passager : John
- Prix : 10000$
- Trajet : YUL - YYZ
- Utilisation restantes: 10
- Billet G7U8E2 (Classe : Economie)


Microsoft Visual Studio Debug Console
- Utilisation restantes: 10
- Billet G7U8E2 (Classe : Economie)
- Passager : John
- Prix : 2250$
- Trajet : YUL - YYZ
- Vol le : 2019-12-21
- Pourcentage solde: 0.5%
- Points : 412
- Coupons :
- Membre Marc:
- Billets :
- Billet D4E5F6 (Classe : Affaire)
- Passager : Marc
- Prix : 2000$
- Trajet : YUL - YYZ
- Vol le : 2019-12-21
- Billet H7I8J9 (Classe : Affaire)
- Passager : Marc
- Prix : 1600$
- Trajet : YUL - YYZ
- Vol le : 2019-12-21
- Billet f2 (Classe : Affaire)
- Passager : Marc
- Prix : 20000$
- Trajet : YUL - YYZ
- Utilisation restantes: 10
- Billet f3 (Classe : Affaire)
- Passager : Marc
- Prix : 18000$
- Trajet : YUL - YYZ
- Utilisation restantes: 10
- Points : 4360
- Coupons :
- Coupon 20%. Rabais : 0.2.
- Membre Robert:
- Billets :
- Billet T8U7P0 (Classe : Economie)
- Passager : Robert
- Prix : 4500$
- Trajet : YUL - YYZ
- Vol le : 2019-12-21
- Pourcentage solde: 0.25%
- Points : 337
- Coupons :