INF1010

Programmation Orientée-Objet

Travail pratique #2
Vecteurs et surcharge d'opérateurs

Objectifs : Permettre à l'étudiant de se familiariser avec la surcharge d'opérateurs, les vecteurs de la librairie STL et l'utilisation du pointeur ***this***.

Remise du travail : Mardi 24 septembre 2019, 8h00 (AM).

Références : Notes de cours sur Moodle & Chapitre 14 du livre Big C++ 2e éd.

Documents à remettre : Les fichiers .cpp et .h **uniquement**, complétés et réunis sous la forme d'une archive au format **.zip**.

Directives : Directives de remise des Travaux pratiques sur Moodle
Les en-têtes (fichiers, fonctions) et les commentaires sont obligatoires.
Les travaux dirigés s'effectuent obligatoirement en équipe de deux personnes faisant partie du même groupe. **Pas de remise possible sans être dans un groupe.**
Veuillez suivre le guide de codage

Travail à réaliser

Le travail consiste à continuer l'application pour le programme de fidélité d'Air Poly commencée au TP précédent en y intégrant les notions de vecteurs, de surcharge d'opérateurs, de constructeur par copie et d'opérateur d'affectation.

Pour remplacer les tableaux dynamiques qui rendaient la gestion des dépenses et des utilisateurs difficile, ce TP fait appel aux vecteurs de la librairie STL, soit *std::vector*. Par ailleurs, pour faciliter les interactions avec les différents objets, la surcharge d'opérateurs sera utilisée.
Les vecteurs implémentés en C++ (STL) sont très pratiques : ce sont des tableaux dont la taille d'adapte dynamiquement. On peut y ajouter des éléments sans se préoccuper de la taille de notre vecteur étant donné que la gestion de la mémoire est automatique. Lorsqu'il vous est demandé d'utiliser un vecteur de la STL plutôt qu'un tableau dynamique pensez à **bien supprimer tous les attributs qui n'ont plus lieu d'être et d'adapter les méthodes qui ont besoin d'être mises à jour**.

Le langage C++ est un langage avec lequel il est possible de redéfinir la manière dont fonctionnent la plupart des opérateurs (arithmétiques (+, -, *, /), d'affectation, etc..) pour de nouvelles classes. Nous pouvons donc redéfinir le comportement de ces opérateurs afin qu'ils effectuent une nouvelle opération ou englobent plusieurs opérations pour ces nouvelles classes.
Pour vous aider, la solution du TP précédent est fournie. Vous n'avez qu'à implémenter les nouvelles méthodes décrites plus bas et supprimer les attributs qui n'ont plus lieu d'être. Les méthodes à modifier vous ont été indiquées.
ATTENTION : Tout au long du TP, assurez-vous d'utiliser les opérateurs sur les objets et non sur leurs pointeurs ! Vous devez donc déréférencer les pointeurs si nécessaire.
ATTENTION : Vous serez pénalisés pour les utilisations inutiles du mot-clé *this*. Utilisez-le seulement où nécessaire.
ATTENTION : Il faut utiliser les fichiers fournis, plutôt que de continuer avec vos fichiers du TP1.
Remarque : Pour plus de précision sur le travail à faire et les changements à effectuer, veuillez-vous référer aux fichiers .h

Classe Coupon

Cette classe caractérise un coupon rabais par son code, son taux de rabais et son coût en points.

Les méthodes suivantes doivent être implémentées :

- L'opérateur << (remplace la méthode d'affichage), qui affiche les caractéristiques du coupon. Cet opérateur doit pouvoir être appelé en cascade.
- L'opérateur > qui compare deux coupons. Retourne vrai si le coupon de gauche a un rabais supérieur à celui de droite. Si les deux coupons ont le même rabais, retourne faux.
- L'opérateur < qui compare deux coupons. Retourne vrai si le coupon de gauche a un rabais inférieur à celui de droite. Si les deux coupons ont le même rabais, retourne faux.

Classe Billet

Cette classe représente un billet d'avion. Un billet est tout le temps associé à un membre.

La méthode suivante doit être implémentée :

- L'opérateur << (remplace la méthode d'affichage), qui affiche les caractéristiques du billet. Cet opérateur doit pouvoir être appelé en cascade.

Classe Membre

Cette classe caractérise un membre du programme. Un membre est en tout temps associé au gestionnaire du programme.

Les attributs suivants doivent être ajoutés :

- billets_ : Un vecteur (*vector*) de pointeurs à des billets d'avions associés à ce membre. Ce vecteur remplace le tableau dynamique existant.
- coupons_ : Un vecteur (*vector*) de coupons que possède ce membre. Un membre peut avoir plusieurs fois le même coupon. Ce vecteur remplace le tableau dynamique existant.

Pensez à supprimer les attributs obsolètes et d'ajuster les méthodes en conséquence.

Les méthodes suivantes doivent être implémentées :

- Un constructeur par copie.
- L'opérateur << (remplace la méthode d'affichage), qui affiche les caractéristiques du membre. Cet opérateur doit pouvoir être appelé en cascade.
- L'opérateur += (remplace la méthode ajouterCoupon(Coupon* coupon)). Cet opérateur doit pouvoir être appelé en cascade.
- L'opérateur -= (remplace la méthode retirerCoupon(Coupon* coupon)). Cet opérateur doit pouvoir être appelé en cascade.
- L'opérateur = qui écrase les attributs du membre par les attributs du membre passé en paramètre. Cet opérateur doit pouvoir être appelé en cascade.
- L'opérateur == qui compare un membre avec un string. Exemple : *membre* == "Thomas". Retourne vrai si le nom du membre est égal au string. L'opérande de gauche est le membre et celui de droite et le string.

- L'opérateur == qui compare un string avec un membre. Attention, ici l'opérande de gauche est le string et celui de droite est le membre. Exemple : "Thomas" == *membre.*

La méthode suivante doit être modifiée :

- La méthode acheterCoupon doit utiliser l'opérateur += que vous avez surchargé dans la classe Membre pour ajouter un coupon au membre.

Voir le commentaire dans membre.cpp pour plus d'explications sur cette modification.

Classe Gestionnaire

Cette classe représente le cœur du programme. Elle permet de gérer les différents membres, la liste des coupons disponibles et de réaliser diverses autres opérations.

Les attributs suivants doivent être ajoutés :

- membres_ : Un vecteur (*vector*) de pointeurs à des billets d'avions associés à ce membre. Ce vecteur remplace le tableau dynamique existant.
- coupons_ : Un vecteur (*vector*) de pointeurs à des coupons de tous les coupons disponibles dans le programme. Ce vecteur remplace le tableau dynamique existant.

Pensez à supprimer les attributs obsolètes et d'ajuster les méthodes en conséquence.

La méthode suivante doit être implémentée :

- L'opérateur << (remplace la méthode d'affichage), qui affiche les caractéristiques du gestionnaire. Cet opérateur doit pouvoir être appelé en cascade.

Les méthodes suivantes doivent être modifiées :

- La méthode trouverMembre doit utiliser l'opérateur == que vous avez surchargé dans la classe Membre pour trouver le membre.
- La méthode appliquerCoupon doit utiliser l'opérateur > que vous avez surchargé dans la classe Coupon pour trouver le coupon avec le plus grand rabais. Elle doit aussi utiliser l'opérateur -= que vous avez surchargé dans la classe Membre pour retirer le coupon du membre.
- La méthode acheterCoupon doit utiliser l'opérateur > que vous avez surchargé dans la classe Coupon pour trouver le coupon avec le plus grand rabais.

Voir les commentaires dans gestionnaire.cpp pour plus d'explications sur ces modifications.

Main.cpp

Le fichier main.cpp vous est fourni et ne doit pas être modifié pour la remise, à l'exception de l'ajout des réponses aux questions. Initialement, ce fichier ne pourra pas compiler puisqu'il fait usage d'opérateurs qui n'ont pas encore été surchargés par vous. N'hésitez pas à commenter

certaines parties si vous avez besoin de compiler votre code. Vous pouvez aussi utiliser ce fichier pour mieux comprendre les requis des fonctions.

Une fois tous les fichiers complétés, tous les tests devraient passer. De plus, votre affichage devrait avoir une apparence semblable à l'exemple de la prochaine page.

Affichage attendu

```
Le membre ccc n'existe pas
Le membre ne peut acheter de coupon
Le membre n'a pas de coupon utilisable
TESTS
	Test 01... OK!
	Test 02... OK!
	Test 03... OK!
	Test 04... OK!
	Test 05... OK!
	Test 06... OK!
	Test 07... OK!
	Test 08... OK!
	Test 09... OK!
	Test 10... OK!
	Test 11... OK!
	Test 12... OK!
	Test 13... OK!
	Test 14... OK!
	Test 15... OK!
	Test 16... OK!
	Test 17... OK!
	Test 18... OK!
	Test 19... OK!
	Test 20... OK!
	Test 21... OK!
	Test 22... OK!
	Test 23... OK!
	Test 24... OK!
	Test 25... OK!
	Test 26... OK!
	Test 27... OK!
	Test 28... OK!
	Test 29... OK!
	Test 30... OK!
	Test 31... OK!
	Test 32... OK!
	Test 33... OK!
	Test 34... OK!
	Test 35... OK!
	Test 36... OK!
	Test 37... OK!

================== ETAT ACTUEL DU PROGRAMME ==================

- Membre Marc:
	- Points  : 1960
	- 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
	- Coupons :
		- Coupon 10%. Rabais : 0.1.

- Membre John:
	- Points  : 150
	- Billets :
		- Billet A1B2C3 (Classe : Premium economie)
			- Passager : John
			- Prix     : 1000$
			- Trajet   : YUL - YYZ
			- Vol le   : 2019-12-21
	- Coupons :

- Membre Robert:
	- Points  : 0
	- Billets :
	- Coupons :
```

Spécifications générales

- Ajouter un destructeur pour chaque classe chaque fois que cela vous semble pertinent.
- Utilisez la liste d'initialisation pour l'implémentation de vos constructeurs.
- Ajouter le mot-clé *const* chaque fois que cela est pertinent.
- Documenter votre code source.
- Ne remettez que les fichiers .h et .cpp lors de votre remise.
- **Bien lire le barème de correction ci-dessous.**

Questions

Répondez aux questions au début du main.

1. Quel est l'utilité de l'opérateur = et du constructeur par copie?
2. Dans quel cas est-il nécessaire de surcharger l'opérateur = et le constructeur par copie?
3. Pourquoi avons-nous dû surcharger l'opérateur = et le constructeur par copie pour la classe Membre?
4. Qu'est-ce qui différencie l'opérateur = du constructeur par copie?

Correction

La correction du TP2 se fera sur 20 points.

Voici les détails de la correction :

- (3 points) Compilation du programme
- (3 points) Exécution du programme
- (4 points) Comportement exact des méthodes du programme
- (3 points) Surcharge correcte des opérateurs
- (2 points) Utilisation correcte des vecteurs
- (2 points) Documentation du code et bonne norme de codage
- (1 point) Utilisation correcte du mot-clé *this* pour les opérateurs
- (2 points) Réponses aux questions