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 02 octobre 2018, 8h

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

Documents à remettre : Tous les fichiers .cpp et .h exclusivement 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 PolyCount 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 (ou vector si on utilise using namespace std). 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 est dynamique. 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 à 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, les fichiers corrigés du TP précédent vous sont fournis. 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 Depense

Cette classe caractérise une dépense par son montant et un nom (court mot se référant à la dépense).

Elle contient les nouveaux attributs privés suivants :

- lieu_ : un pointeur vers un string représentant le lieu où a été effectuée la dépense (pensez à modifier les constructeurs). C'est un attribut composite.

Les méthodes suivantes doivent être implémentées :

- L'opérateur = qui écrase les attributs de la dépense par les attributs de celle passée en paramètre et qui renvoie ensuite une référence à la dépense.
- L'opérateur << (remplace la méthode d'affichage), qui affiche les caractéristiques de la dépense.
- Un constructeur par copie.

Classe Utilisateur

Cette classe caractérise un utilisateur par son nom et les dépenses qu'il a effectuées.

Elle contient les nouveaux attributs privés suivants :

- depenses_ : un vecteur de pointeurs de dépenses contenant les dépenses réalisées par cet utilisateur (pensez à supprimer les attributs obsolètes)

Les méthodes suivantes doivent être implémentées :

- Une méthode d'accès au nouvel attribut.
- Un constructeur par copie
- L'opérateur += prenant en paramètre un pointeur vers une dépense et qui permet d'ajouter une dépense à l'utilisateur (remplace ajouter utilisateur). Il renvoie aussi une référence à l'utilisateur
- L'opérateur = qui écrase les attributs de l'utilisateur par les attributs de l'utilisateur passé en paramètre et qui renvoie ensuite une référence à l'utilisateur.
- L'opérateur << (remplace la méthode d'affichage), qui affiche les caractéristiques de l'utilisateur.

Classe Transfert

Cette classe caractérise un transfert d'argent, et permet de rééquilibrer les comptes entre chaque utilisateur (but final du programme).

Les méthodes suivantes doivent être implémentées :

- L'opérateur << (remplace la méthode d'affichage), qui affiche les caractéristiques du transfert.

Classe Groupe

Cette classe caractérise un groupe qui participe à un évènement, auquel sont liés des utilisateurs et une liste de dépenses (toutes les dépenses liées à l'évènement).

Elle contient les nouveaux attributs privés suivants :

- depenses_ : un vecteur de pointeurs de dépenses contenant les dépenses réalisées par les utilisateurs du groupe (pensez à supprimer les attributs obsolètes).
- utilisateurs_ : un vecteur de pointeurs d'utilisateurs contenant les utilisateurs appartenant à ce groupe (pensez à supprimer les attributs obsolètes).
- transferts_ : un vecteur de pointeurs de transferts contenant les transferts réalisés afin de permettre l'équilibre des comptes (pensez à supprimer les attributs obsolètes).
- comptes_ : un vecteur de double représentant les comptes entre les utilisateurs du groupe.*

Les méthodes suivantes doivent être implémentées :

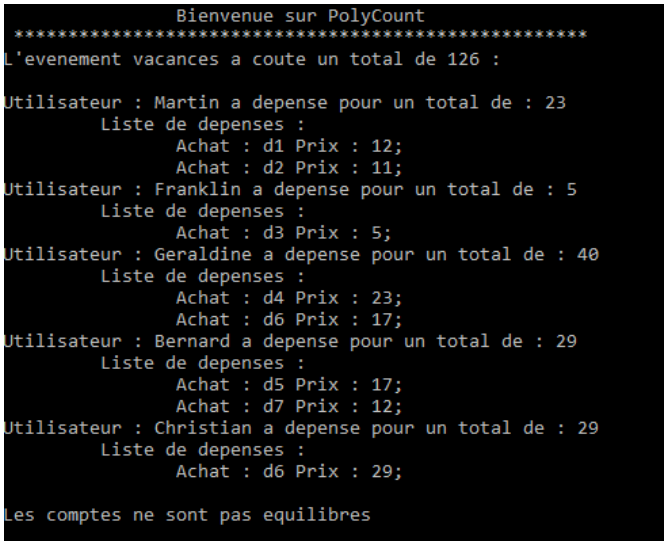
- L'opérateur += prenant en paramètre un utilisateur, permettant d'ajouter un utilisateur au groupe (remplace ajouterUtilisateur). Il renvoie aussi une référence vers le groupe.
- La méthode ajouterDepense doit être modifiée pour renvoyer le groupe après l'ajout de la dépense au groupe et à l'utilisateur.
- L'opérateur << (remplace la méthode d'affichage), qui affiche les caractéristiques de l'utilisateur.

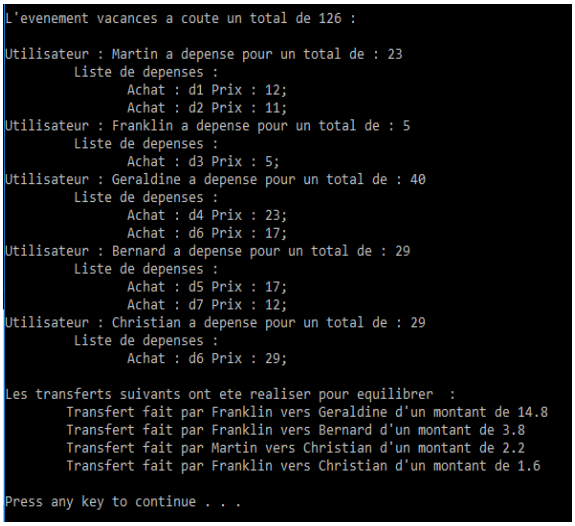
*Par exemple : Max dépense 10$ pour le groupe, Alex 30$ et Henri 50$. Les dépenses totales du groupe s'élèvent à 90$, soit 30$ par personne. Ainsi Max doit 20$, Alex est à 0 et Henri devrait recevoir 20$. Ce qui donne les comptes suivants :
Max : -20$
Alex : 0$
Henri : 20$

Main.cpp

Des directives vous sont fournies dans le fichier main.cpp et il vous est demandé de les suivre.

Votre affichage devrait avoir une apparence semblable à celle ci-dessous. Vous êtes libre de proposer un rendu plus ergonomique et plus agréable si vous le désirez.





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.
- Appliquez un affichage « user friendly » (ergonomique et joli) pour le rendu final.
- Documenter votre code source.
- **Bien lire le barème de correction ci-dessous.**

Questions

Répondez aux questions au début du main.

1. Quelle est l'utilité de l'opérateur = et du constructeur par copie ?
2. Qu'est-ce qui différencie l'opérateur = du constructeur par copie ?

Correction

La correction du TP1 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) Réponse aux questions.