INF1010

Programmation Orientée-Objet

Travail pratique #1
Allocation dynamique, composition et agrégation

Objectifs : Permettre à l'étudiant de se familiariser avec les notions de base de la programmation orientée objet, l'allocation dynamique de la mémoire, le passage de paramètres, les méthodes constantes et les principes de relation de composition et d'agrégation.

Remise du travail : Lundi 24 septembre 2018, 8h00 (AM).

Références : Notes de cours sur Moodle & Chapitre 2-7 du livre Big C++ 2e éd.

Documents à remettre : Tous les fichiers .cpp et .h complétés 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

La directive de précompilation « #ifndef »

La directive de précompilation « #ifndef » signifie « if not defined » (si non défini). Comme le type de directive le laisse deviner, cette directive est évaluée avant la phase de compilation du code source. Dans les travaux pratiques, vous l'utiliserez dans les fichiers d'en-têtes (.h), pour éviter la double inclusion. Un fichier peut inclure deux fichiers d'entête, par exemple prenons deux fichiers a.h et b.h. Il se peut que a.h soit inclus dans le fichier b.h. On se retrouve alors à inclure deux fois le fichier a.h, ce qui entrainerait une erreur de compilation, car on ne peut définir deux fois la même classe. La directive « #ifndef » nous évite cette double inclusion. Pour utiliser la directive « #ifndef », il faut respecter la syntaxe suivante :

```
#ifndef NOMCLASSE_H

#define NOMCLASSE_H

// Définir la classe NomClasse ici

#endif
```

La directive de précompilation « #include»

La directive de précompilation pour l'inclusion de fichiers « #include »

1. #include <nom_fichier>
2. #include "nom_fichier"

Ce qui différencie ces deux expressions est l'emplacement où le fichier spécifié est recherché. Pour la seconde forme, le précompilateur commence tout d'abord par rechercher dans le même répertoire que le fichier compilé. Par la suite, il procède de la même manière que la première forme, c'est-à-dire dans des répertoires prédéfinis par l'environnement de développement intégré.

En résumé, lorsqu'on inclut un fichier source qui se trouve dans le projet, on utilise la seconde forme. Et lorsque l'on inclut un fichier qui provient d'une bibliothèque externe au projet, on utilise la première forme.

Mise en contexte

Après avoir passé un été durant lequel vous avez pris part à plusieurs évènements avec vos amis, vous remarquez qu'il est compliqué de tenir les comptes de dépenses effectués dans votre groupe d'amis. Voilà pourquoi vous décidez de créer un programme, PolyCount, qui vous permettra de lister les dépenses faites par chaque membre pour la totalité du groupe, une fois les dépenses connues, le programme vous indiquera comment réaliser un équilibre entre chaque membre avec le moins de transferts d'argent inter-utilisateur.

Travail à réaliser

Les fichiers depense.h, utilisateur.h, transfert.h et groupe.h sont fournis. On vous demande de compléter les fichiers *.cpp ainsi que de compléter le programme principal en suivant les directives du fichier main.cpp.

Classe Depense

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

Cette classe contient les attributs privés suivants :

- titre_ (chaine de caractères, string)
- montant_ (double)

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut qui initialise les attributs aux valeurs par défaut
- Un constructeur par paramètres qui initialise les attributs aux valeurs correspondantes
- Les méthodes d'accès aux attributs.
- Les méthodes de modification.
- Une méthode d'affichage.

Classe Utilisateur

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

Elle contient les attributs privés suivants :

- nom, (chaine de caractères)
- listeDepenses_, tableau dynamique contenant des pointeurs à des dépenses.
- tailleTabDepense_, qui représente la taille du tableau listeDepenses_ (entier non signé).
- nombreDepenses_, (entier non signé)
- totalDepense_, le montant total des dépenses effectuées par un utilisateur.

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut qui initialise les attributs aux valeurs par défaut (dont une taille de tableau de 5 dépenses).
- Un constructeur par paramètres qui initialise les attributs aux valeurs correspondantes.
- Les méthodes d'accès aux attributs.
- Les méthodes de modifications.
- Une méthode d'ajout de dépense, (attention : vérifier si la taille du tableau le permet, et réagir dans le cas contraire). On doit toujours pouvoir ajouter une dépense.
- Une méthode calculerTotal(), qui calcule le montant total des dépenses effectuées.
- Une méthode d'affichage.

Classe transfert

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

Elle contient les attributs privés suivants :

- montant_, (double)
- donneur_, (pointeur vers l'utilisateur qui doit de l'argent)
- receveur_, (pointeur vers l'utilisateur qui reçoit l'argent)

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut qui initialise les attributs aux valeurs par défaut.
- Un constructeur par paramètres qui initialise les attributs aux valeurs correspondantes.
- Les méthodes d'accès.
- Les méthodes de modification.
- Une méthode d'affichage

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 attributs privés suivants :

- nom_, (chaîne de caractères)
- totalDepenses_, le total des dépenses de l'évènement (double)
- listeUtilisateurs_, un tableau dynamique contenant des pointeurs vers les utilisateurs participants à l'évènement.

- tailleTabUtilisateurs_, (entier non signé) .
- nombreUtilisateurs_, (entier non signé) ; le nombre d'utilisateurs ayant participé au groupe.
- listeDepenses_, tableau dynamique de pointeurs vers des dépenses.
- nombreDepenses_, (entier non signé).
- tailleTabDepenses_, (entier non signé).
- comptes_, tableau dynamique vers des double (de taille nombreUtilisateurs_), ils représentent les comptes entre utilisateurs du groupe.*
- listeTransferts_, un tableau dynamique (de taille nombreUtilisateurs_) contenant des pointeurs vers des transferts.
- nombreTransferts_, (entier non signé).

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut qui initialise les attributs aux valeurs par défaut. La taille des utilisateurs_ est 5 et la taille des dépenses_ est 10.
- Un constructeur par paramètres qui initialise les attributs aux valeurs correspondantes
- Les méthodes d'accès.
- Les méthodes de modifications.
- Une méthode d'ajout d'utilisateur, (attention : vérifier si la taille du tableau le permet, et réagir dans le cas contraire). On doit pouvoir toujours ajouter un utilisateur.
- Une méthode d'ajout de dépense, prend deux paramètres : un pointeur vers une dépense et un pointeur vers un utilisateur, il faudra ajouter la dépense à l'évènement mais également à l'utilisateur qui l'a payée (attention : vérifier si la taille du tableau le permet, et réagir dans le cas contraire). On doit pouvoir toujours ajouter une dépense.
- Une méthode calculerTotalDepenses(), qui met à jour le total de dépenses de l'évènement et fait les comptes (soit le montant, positif ou négatif, de nos dépenses par rapport à la moyenne de dépenses de l'évènement).
- Une méthode équilibrerComptes(), qui créé des transferts (de manière optimale, soit le nombre minimum de transferts) afin de remettre les comptes à zéro et les mets dans la liste. Il faut utiliser les attributs comptes_ et listeTransferts_. Il est de votre responsabilité de concevoir l'algorithme de cette méthode.
- Une méthode d'affichage, qui affiche toutes les dépenses par utilisateur, les comptes (s'ils ne sont pas tous nuls), les transferts (si comptes remis à 0),

*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.

```
        Bienvenue sur PolyCount
 ****************************************
L'évenement : Madrid a couté un total pour le groupe de :
620 voici les dépenses :
     Utilisteur  : u1 a dépensé pour un total de : 310
     Liste de dépenses :
Achat : d1 Prix : 200;

Achat : d6 Prix : 50;

Achat : d7 Prix : 60;

     Utilisteur  : u2 a dépensé pour un total de : 10
     Liste de dépenses :
Achat : d2 Prix : 10;

     Utilisteur  : u3 a dépensé pour un total de : 50
     Liste de dépenses :
Achat : d3 Prix : 50;

     Utilisteur  : u4 a dépensé pour un total de : 50
     Liste de dépenses :
Achat : d4 Prix : 50;

     Utilisteur  : u5 a dépensé pour un total de : 200
     Liste de dépenses :
Achat : d5 Prix : 200;

soit les comptes suivants :
u1 : 186
u2 : -114
u3 : -74
u4 : -74
u5 : 76
```

```
L'évenement : Madrid a couté un total pour le groupe de :
620 voici les dépenses :
     Utilisteur  : u1 a dépensé pour un total de : 310
     Liste de dépenses :
Achat : d1 Prix : 200;

Achat : d6 Prix : 50;

Achat : d7 Prix : 60;

     Utilisteur  : u2 a dépensé pour un total de : 10
     Liste de dépenses :
Achat : d2 Prix : 10;

     Utilisteur  : u3 a dépensé pour un total de : 50
     Liste de dépenses :
Achat : d3 Prix : 50;

     Utilisteur  : u4 a dépensé pour un total de : 50
     Liste de dépenses :
Achat : d4 Prix : 50;

     Utilisteur  : u5 a dépensé pour un total de : 200
     Liste de dépenses :
Achat : d5 Prix : 200;

pour équilibrer  :
Transfert fait part : u2 pour : u1 d'un montant de : 114
Transfert fait part : u3 pour : u5 d'un montant de : 74
Transfert fait part : u4 pour : u1 d'un montant de : 72
Transfert fait part : u4 pour : u5 d'un montant de : 2
```

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

1. Quel est le lien (agrégation ou composition) entre les classes Groupe et Transfert ?
2. Quel effet aura une méthode si elle a un const ?

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 ;
- (2 points) Documentation du code et bonne norme de codage ;
- (2 points) Utilisation correcte du mot-clé *const* et dans les endroits appropriés ;
- (2 points) Utilisation adéquate des directives de précompilation ;
- (2 points) Allocation et désallocation appropriées de la mémoire ;
- (2 points) Réponse aux questions ;