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 :	Lundi 19 Novembre 2018, 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 2 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 une simulation de partage de dépense 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 à 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.

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 itérateurs.

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

Remarque : Les modifications doivent utiliser les méthodes de la classe set de la STL ainsi que les itérateurs de set. Vous avez le droit d'utiliser le mot clé « auto » pour simplifier votre code.

Directives

- 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.

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 T` (dans notre cas des `Utilisateur*` ou des `Depense*`) à l'aide d'un conteneur de `typename C` (qui pourra par exemple soit être une `map<Utilisateur*, double>` ou un `vector<Depense*>`). Le `typename D` permet de spécifier le type de retour de la méthode `getElementParIndex()` étant donné qu'elle peut varier en fonction du type de conteneur. Enfin, comme la méthode d'ajout est différente pour chaque conteneur nous allons utiliser des foncteurs d'ajout de `typename FoncteurAjouter`.

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 ce pointeur 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` : Cette méthode retourne le nombre d'éléments contenus dans le conteneur_.
- `D getElementParIndex(int i) const` : Cette méthode retourne l'élément du conteneur se situant à l'index i passé par paramètre. Cette méthode retourne un objet de type D qui peut être soit `Depense*` soit `pair<Utilisateur*, double>`.

Classe *GestionnaireDepenses*

La classe GestionnaireDepenses permet d'abstraire la gestion des dépenses dans la classe Groupe. Cette classe vous est fournie. Il vous faudra la modifier afin qu'elle hérite de la classe GestionnaireGenerique.

Attributs :

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

Méthodes :

Toutes les méthodes devront être retirées à l'exception de getTotalDepenses() qui devra être modifiée pour fonctionner avec sa classe mère GestionnaireGenerique.

Méthodes à modifier :

- `double getTotalDepenses() const` : Retourne le total des dépenses contenues dans le vecteur. (le conteneur à utiliser n'est plus depenses_ mais, conteneur_)

Classe *GestionnaireUtilisateurs*

La classe GestionnaireUtilisateurs permet d'abstraire la gestion des utilisateurs dans la classe Groupe. Cette classe vous est fournie. Il vous faudra la modifier afin qu'elle hérite de la classe GestionnaireGenerique et qu'elle puisse gérer les comptes aussi.

La classe GestionnaireUtilisateurs devra gérer un conteneur de type map de la STL. Sa clé sera un Utilisateur* et l'élément qu'elle permettra d'accéder sera un double (`map<Utilisateur*, double>`).

Attributs :

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

Méthodes :

Toutes les méthodes devront être retirées à l'exception de estExistant().

- `vector<double> getComptes() const` : Retourne un vector contenant les comptes des utilisateurs.
- `int estExistant(Utilisateur* utilisateur) const` : Retourne true si l'utilisateur existe, sinon faux. (Cette méthode vous est déjà fournie)
- `void mettreAJourComptes(Utilisateur* payePar, double montant)` : Met à jour les comptes avec le montant passé en paramètre. (Référez-vous à la méthode ajouterDepense de la classe Groupe)

- `pair<Utilisateur*, double>& getMax() const` : Retourne l'élément de la map ayant la valeur de compte la plus élevée.
- `pair<Utilisateur*, double>& getMin() const` : Retourne l'élément de la map ayant la valeur de compte la plus faible.
- `Utilisateur* getUtilisateurSuivant(Utilisateur* utilisateur, double montant) const` : Retourne l'utilisateur suivant l'utilisateur passé par paramètre. Cette méthode doit utiliser la fonction find_if de la STL avec un bind et un placeholder.
- `vector<pair<Utilisateur*, double>> getUtilisateursEntre(double borneInf, double borneSup) const` : Retourne un vector contenant les utilisateur compris entre la borne inférieure et la borne supérieure. Dans cette méthode, il faut utiliser le FoncteurIntervalle que vous allez implémenter, la fonction copy_if de la STL et un back_inserter.
- `GestionnaireUtilisateurs& setCompte(pair<Utilisateur*, double> p)` : Met à jour la `pair<Utilisateur*, double>` dans la `map<Utilisateur*, double>`.

Classe *Groupe*

Dans cette classe, il vous suffit d'adapter l'implémentation des méthodes en fonction des changements dans les classes GestionnaireUtilisateurs, GestionnaireDepenses et GestionnaireGenerique.

Attributs :

Les vector depenses_ et utilisateurs_ ont été déplacés et sont gérés par leur gestionnaire respectif.

Les attributs gesitonnaireDepenses_ et gestionnaireUtilisateurs_ ont été ajoutés (GestionnaireUtilisateurs* et GestionnaireDepenses*).

Méthode à ajouter :

- `GestionnaireUtilisateurs* getGestionnaireUtilisateurs()` : Cette méthode retourne le pointeur gestionnaireUtilisateurs_.
- `GestionnaireDepenses* getGestionnaireDepenses()` : Cette méthode retourne le pointeur gestionnaireDepenses_.

Méthodes à modifier :

- `vector<double> getComptes() const` : Utiliser la méthode appropriée de gestionnaireUtilisateurs_.
- `double getTotalDepenses() const.`
- `vector<Depense*> getDepenses() const` : Utiliser la méthode appropriée de gestionnaireDepenses_.
- `vector<Utilisateur*> getUtilisateurs() const` : Utiliser la méthode appropriée de gestionnaireUtilisateurs_.

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

FoncteurAjouterUtilisateur

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

Attribut :

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

Méthodes :

- **operator() :** prend en paramètre un pointeur d'Utilisateur et retourne la map par référence avec l'Utilisateur ajouté.

FoncteurAjouterDepense

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

Attribut :

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

Méthodes :

- **operator() :** prend en paramètre un pointeur de dépense et retourne le vector par référence avec la Depense ajouté.

FoncteurIntervalle

Ce foncteur est utilisé par le GestionnaireUtilisateurs afin de trouver l'utilisateur de la **map** dont le compte est 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 :

operator() : prend en paramètre une `pair<Utilisateur*, double>` et retourne vrai si le compte associé à la paire est compris entre les bornes borneInf_ et borneSup_, faux sinon.

Main.cpp

Pour ce TP le main est entièrement fournis 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. Aurait-il été possible de créer un autre gestionnaire ? Si oui, lequel et quel type de conteneur conseilleriez-vous ?

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 ?

3. Donner le diagramme des classes dans le fichier PDF.

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;
- (1 points) Documentation du code ;
- (2 points) Allocation / désallocation appropriée mémoire
- (1 point) Réponse aux questions.

Affichage attendu

```
Erreur : L'utilisateur Rebecca doit renouveler son abonnement premium
Erreur : L'utilisateur Axel n'est pas un utilisateur premium et est deja dans un groupe.
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!
```

```
Groupe Madrid.
Cout total: 2040$
Utilisateurs:

        - Utilisateur (regulier, dans un groupe) Yves :
                Total a payer:280.00 (7.58$ de frais)
                Depenses :
                        - Depense (a Montreal) : d4. Prix : 60.00$

        - Utilisateur (regulier, dans un groupe) Martine :
                Total a payer:-380.00 (0.00$ de frais)
                Depenses :
                        - Depense (a Montreal) : d5. Prix : 600.00$
                        - Depense (a Montreal) : d8. Prix : 120.00$

        - Utilisateur (regulier, dans un groupe) Samuel :
                Total a payer:40.00 (2.00$ de frais)
                Depenses :
                        - Depense (a Montreal) : d6. Prix : 300.00$

        - Utilisateur (premium) Ryan :
                Total a payer: -20.00$ (-0.00$ economises)
                Jours restants: 10
                Depenses :
                        - Depense (a Montreal) : d1. Prix : 180.00$
                        - Depense (a Montreal) : d9. Prix : 180.00$

        - Utilisateur (premium) David :
                Total a payer: 100.00$ (3.00$ economises)
                Jours restants: 10
                Depenses :
                        - Depense (a Montreal) : d3. Prix : 240.00$

        - Utilisateur (premium) Gaspard :
                Total a payer: -20.00$ (-0.00$ economises)
                Jours restants: 10
                Depenses :
                        - Depense (a Montreal) : d2. Prix : 360.00$


Transferts :
        - Yves     -> Martine  : 280.00$
        - David    -> Martine  : 100.00$
        - Samuel   -> Ryan     : 20.00$
        - Samuel   -> Gaspard  : 20.00$
```