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 :	Lundi 12 Février 2018, 8h
Références :	Notes de cours sur Moodle & Chapitre 14 du livre Big C++ 2e éd.
Documents à remettre :	Seulement 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. Veuillez suivre le guide de codage.

Travail à réaliser

Le travail consiste à continuer l'application de commerce en ligne commencé au TP précédent en y intégrant les notions de vecteurs, de surcharge d'opérateurs, constructeur de copie et opérateur d'affectation.

Pour remplacer les tableaux dynamiques qui rendaient la gestion des produits difficile, ce TP fait appel aux vecteurs de la librairie STL, soit std::vector. Et, 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 du TP précédent vous sont fournis. Vous n'avez qu'à implémenter les nouvelles méthodes décrites plus bas. Les attributs qui ne sont plus nécessaires ont été supprimés. Et 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 *Produit*

Cette classe Produit est caractérisée par un nom, une référence, et un prix.

Les attributs et méthodes liés au TP1 restent inchangés, sauf si spécifié.

Les méthodes suivantes doivent être implémentées :

- ~~L'opérateur << (remplace la méthode d'affichage), qui affiche les caractéristiques du produit.~~
- ~~L'opérateur == qui prend un *Produit* en paramètre et permet de comparer 2 produits (mêmes attributs). Cet opérateur va pouvoir faire l'opération *produit1* == *produit2*.~~
- ~~Les opérateurs < et > qui compare deux produits, l'un est plus grand que l'autre lorsque son prix est supérieur (strictement) au prix du deuxième.~~
- ~~L'opérateur >> qui permet de saisir en entrée les attributs (nom, ref, prix) d'un Produit.~~

Classe *Rayon*

Cette classe est caractérisée par une catégorie et un tableau dynamique de Produit.

Les attributs et méthodes liés au TP1 restent inchangés, sauf si spécifié.

Cette classe contient l'attribut privé suivant :

- tousProduits_ : Un vecteur de pointeurs Produit, qui contiendra les différents produits.

Les méthodes suivantes doivent être implémentées :

- ~~L'opérateur << (remplace la méthode de l'affichage), qui affiche la catégorie du rayon et tous les produits qu'il contient.~~ Référez-vous à la capture d'écran pour la forme de l'affichage.
- ~~L'opérateur += qui prend en paramètre un produit, qui l'ajoute au vecteur tousProduits_ et qui retourne le rayon après l'ajout de ce produit.~~
- La méthode compterDoublons (Produit* produit) qui retourne le nombre de fois qu'un produit identique au produit passé en paramètre apparaisse dans le rayon. Ici on doit comparer les produits pointés et non pas les pointeurs.

Classe *Panier*

Cette classe sert à regrouper les produits achetés par un client, à calculer le nombre de produits et le total à payer.

Les attributs et méthodes liés au TP1 restent inchangés, sauf si spécifié.

Cette classe contient l'attribut privé suivant :

- contenuPanier_ : Un vecteur de pointeurs de Produit, qui contiendra les produits contenus dans le panier.

Les méthodes suivantes doivent être implémentées :

- L'opérateur << (remplace l'affichage), qui affiche un panier selon suivant la forme du TP1.
- La méthode trouverProduitPlusCher() qui retourne le pointeur vers le produit le plus cher du panier. S'il y a plusieurs produits avec le même prix max, choisissez

celui qui vous arrange. (Astuce : la surcharge d'un opérateur de la classe Produit pourrait vous être utile)

Classe *Client*

Cette classe Client permet de créer un client qui va acheter des produits et les ajouter dans son panier.

Les attributs et méthodes liés au TP1 restent inchangés, sauf si spécifié.

Les méthodes suivantes doivent être implémentées :

- ~~Un constructeur par copie (si nécessaire)~~.
- ~~L'opérateur = qui écrase les attributs de l'objet de gauche par les attributs de l'objet passé en paramètre.~~
- ~~L'opérateur << (remplace l'affichage), qui affiche les informations qui concerne un groupe d'images suivant l'exemple présenté à la fin du document.~~
- ~~L'opérateur == qui prend un entier en paramètre et permet de comparer un client avec son identifiant. Exemple (*client1*== 14183945). L'opérateur retourne true si l'identifiant du client est le même que celui passé en paramètre de l'opérateur.~~

La fonction globale : ~~l'opérateur == qui permet d'inverser l'ordre de l'opérateur == précédent. Ainsi cet opérateur va pouvoir faire l'opération (identifiant==client)~~

Main.cpp

Le programme principal contient des directives à suivre pour instancier différents objets et essayer les différentes méthodes implémentées. Votre affichage devrait avoir une apparence semblable à celle ci-dessous :

```
Saisissez les attributs pour un produit : p15 15 20.54

Le produit saisie est nom : p15  ref : 15        prix : 20.54

Le produit p20 est moins cher que le produit p1 ? false

Le rayon sport:
----> nom : p0   ref : 0         prix : 12.56
----> nom : p1   ref : 1         prix : 50
----> nom : p20  ref : 31        prix : 100
----> nom : p3   ref : 3         prix : 56
----> nom : p4   ref : 4         prix : 77
----> nom : p5   ref : 5         prix : 91
----> nom : p6   ref : 6         prix : 21
----> nom : p7   ref : 7         prix : 34
----> nom : p8   ref : 8         prix : 88
----> nom : p9   ref : 9         prix : 65
----> nom : p0   ref : 0         prix : 12.56
Nombre de doublons du produit p0 : 2

Test identifiant paul: true

Le panier de Paul est vide !

Le panier de Martine:
nom : p14        ref : 14        prix : 68
nom : p13        ref : 13        prix : 53
nom : p12        ref : 12        prix : 82
nom : p11        ref : 11        prix : 72
nom : p10        ref : 10        prix : 42
----> total a payer : 317

Le produit le plus cher que Martine ait achete est :
nom : p12        ref : 12        prix : 82
```

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.
• Note : Lorsque l'on fait référence aux valeurs par défaut, pour un string cela équivaut à la chaine vide, et pour un entier au nombre 0.

Questions

Répondez aux questions au début du main.

1. Quelle est l'utilité de l'opérateur = et du constructeur par copie ?
2. Dans quel cas est-il absolument nécessaire de les implémenter ?
3. Qu'est-ce qui différencie l'opérateur = du constructeur par copie ?

Correction

La correction du TP 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 ;
- (1.5 points) Documentation du code ;
- (1 point) Utilisation correcte du mot-clé this pour les opérateurs ;
- (1 point) Utilisation correcte du mot-clé const ;
- (1.5 points) Réponse aux questions.