C++

Compte rendu de travaux pratiques

GINHAC Dominique

Table des matières

- Avant-propos

- TP 1 : Un peu de lecture
    - Choix du TP
    - Répartition des tâches
    - Présentation du code
    - Conclusion

- TP 3 : EasyStore
    - Choix du TP
    - Répartition des tâches
    - Présentation du code
    - Conclusion

Avant-propos

Ce compte rendu s'inscrit dans le cadre de six séances de travaux pratiques en première année de cycle ingénieur à l'ESIREM. L'ensemble de ces séances réalisées avait pour objectif de nous permettre de nous initier au langage de programmation C++, et d'approfondir les différentes notions vues en cours magistral.

TP 1 : Un peu de lecture

Choix du TP

Ce sujet porte sur la création d'une bibliothèque. Étant considéré "débutant", nous l'avons choisi afin d'appliquer de façon concrète les différentes notions abordées dans les cours magistraux et travaux dirigés.

La première partie de ce TP a été consacrée à la découverte et à la prise en main de l'outil Github, service de cloud permettant aux développeurs de gérer et stocker leurs différents codes, et d'avoir un accès sur l'évolution de ces derniers au cours du temps.
Après avoir compris les bases du site web, nous avons créé notre premier dépôt sur Github, ainsi que nos dépôts locaux sur nos machines, en utilisant pour cela des clés SSH, permettant d'avoir un accès permanent aux dépôts depuis la machine possédant la clé.

Avant même de commencer à coder sur ordinateur, nous avons d'abord analysé l'énoncé et cherché comment créer les différentes de la façon la plus optimale.

Répartition du travail

Salim - Question 3 à 5 et question 7
Adrien - Question 1, 2, 6 et 7

Présentation du code

Lors de ce TP, nous avons utilisé 6 classes :

- Date
- Auteur
- Lecteur
- Livre
- Emprunt
- Bibliothèque

Date

Dans cette classe, nous avons repris l'exemple de la classe Date vue en cours en ajoutant la variable membre qui correspond aux années ainsi que sa fonction getter et setter.

Nous avons également ajouté la surcharge de l'opérateur "==" pour comparer 2 objets Date ainsi qu'une fonction helper "toString(Date d)" qui à l'aide de l'outil "ostrsingstream" permet d'afficher la date sous le format : jour/mois/année.

Auteur

La classe Auteur possède 5 variables : Un identifiant, un nom, un prénom, une date de naissance et un *std::vector* contenant la liste des livres écrit par l'auteur et une liste vide permettant d'initialiser celle des livres. Nous avons ensuite créé les getters pour chaque variable, une fonction affichant les livres écrits par l'auteur, et les surcharges d'opérateurs "<<" et "==" pour afficher et comparer les caractéristiques d'un auteur.

Lecteur

La classe Lecteur, très proche de la classe auteur dans son idée première, comporte 5 variables membres : Un identifiant, un prénom, un nom, une liste des ISBN des livres empruntés et une liste vide qui permet d'initialiser la première car celle-ci a une référence. Cette proximité entre cette classe et la classe précédente nous a d'abord donné l'idée de créer une classe "Personne", possédant un identifiant, un nom et un prénom, dont hériteront les classes Auteur et Lecteur. Nous avons par la suite estimé que cela ajoutait beaucoup de complications pour un problème initialement simple, et avons abandonné cette idée.
Par la suite on a créé ses getters et setters puis on ajouté 2 fonctions permettant la surcharge des opérateurs "<<" et "==". Enfin, nous avons mis en place la méthode "displayLoan()" qui permet d'afficher notre liste et de vérifier son contenu ainsi qu'une fonction amie qui retourne notre liste.

Livre

La classe Livre comprend plusieurs variables basiques, telles que le titre, l'identifiant ISBN, un std::vector contenant le nom des personnes ayant empruntés le livre, un booléen permettant de savoir si le livre est emprunté, ainsi que des objets Date et Auteur pour la date de publication du livre et la personne qui l'a écrit. La nouveauté par rapport aux classes précédentes est l'ajout de deux énumérations, Langue et Genre. Les énumérations nous permettant de choisir un élément parmi un ensemble d'éléments défini au préalable, nous avons pensé que cette solution était la plus adaptée pour les choix de la langue et du genre du livre.
Nous avons ensuite créé toutes les fonctions getter et setter, les surcharges d'opérateur "<<" et "==", et des fonctions permettant d'afficher, ou d'ajouter des emprunts dans notre std::vector "_listLoaner".

Emprunt

Cette classe est composée de 5 variables membres : une date, un livre, un lecteur, un booléen correspondant au statut de l'emprunt et un booléen égale à true permettant d'initialiser le statut car celui-ci à une référence.
Nous avons ensuite créé le constructeur qui permet d'initialiser les variables membres et affiche un message d'erreur si la date d'emprunt n'est pas correcte.
On a également mis en place ses getters ainsi qu'un setter pour la variable du statut de l'emprunt.
Enfin comme pour la classe Date, nous avons créé une fonction helper "toString(Loan l)" qui permet d'afficher les informations de l'emprunt.

Bibliothèque

La classe bibliothèque possède plusieurs std::vector, correspondant aux liste de livres, de lecteurs, d'auteurs et d'emprunts inclus dans la bibliothèque. Le constructeur prend en entrée un auteur, un livre et un lecteur à sa création, qui seront enregistrés à part en plus d'être ajoutés dans leurs tableaux dynamiques respectifs à l'initialisation de la bibliothèque.

Nous avons ensuite créé les méthodes permettant d'ajouter auteur, livre et lecteur à la bibliothèque, à l'aide de la fonction push_back. Nous avons également créé les méthodes d'affichage des différents éléments présents dans la bibliothèque, en utilisant des boucles for affichant un élément à chaque itération. Les méthodes emprunter et restituer ont été les plus compliquées à implémenter car il fallait penser à vérifier la disponibilité du livre ainsi que celle de l'emprunt en lui-même pour éviter d'emprunter plusieurs fois le même livre ou encore le faire restituer par une personne différente de celle de l'emprunt. C'est dans ces 2 méthodes qu'on été utilisé les booléens du livre et de l'emprunt afin d'éviter les situations décrites ci-dessus. C'est également dans ces fonctions que l'ajout des ISBN des livres, des identifiants des lecteurs ainsi que les titres des livres empruntés a été effectué dans leurs listes respectives. La méthode restituer permet de changer le statut de disponibilité du livre et de l'emprunt et supprime le livre restitué de la liste des livres empruntés.

Nous avons créé une méthode permettant d'afficher les prêts, ainsi qu'une méthode calculant le pourcentage de livres prêtés.
La dernière méthode à créer a été celle consistant à afficher le classement des lecteurs en fonction du nombre de livres empruntés. Cette méthode a nécessité la création de deux tableaux liés, un tableau contenant les noms des lecteurs et un autre contenant le nombre de livres empruntés par chacun d'entre eux. Nous avons ensuite réalisé un simple algorithme de tri par ordre croissant, puis nous affichons chaque ligne des deux tableaux une fois que ces derniers sont triés.

Conclusion

Les séances de travaux pratiques dédiées à ce sujet ont été essentielles pour appliquer les connaissances que nous avions vu jusqu'à présent, telles que la surcharge d'opérateur ou la création de différentes méthodes, et ainsi de mieux assimiler ces notions.
La découverte de Github est, et sera pour le futur essentielle également, autant pour le "versionning" que pour les différentes possibilités que l'outil offre pour les travaux de groupe.
La réalisation du code du TP1 s'est passée pour l'ensemble sans gros problème ou blocage, et à la fin de ce sujet, nous avons considéré être assez à l'aise avec le C++ pour passer à un sujet "intermédiaire".

TP 3 : EasyStore

Choix du TP

Ce TP consiste à créer une application de gestion de magasin en ligne appelée EasyStore. Nous avons choisi ce TP car il est situé dans le niveau intermédiaire et nous nous sentions confortable à l'idée d'élever la difficulté du TP par rapport au premier. De cette façon nous pouvons remarquer les différences entre ces 2 niveaux et ainsi rencontrer d'autres problèmes qui nous force à mieux comprendre le langage de programmation.

Comme pour le premier TP, nous avons utilisé GitHub pour mettre en commun nos codes respectifs à l'aide de dépôts locaux sur nos machines.

Répartition du travail

Salim - Question 1 à 5 et question 8
Adrien - Question 6 et 7

Présentation du code

Dans ce TP, nous avons utilisé 4 classes et un menu créé à partir de fonctions :

- Produit
- Client
- Commande
- Magasin
- Menu

Produit

La classe Produit a été créée au commencement du TP, elle permet de créer un objet Produit qui comme son nom l'indique représente un produit du magasin. Cette classe comporte 4 variables membres : un titre, une description, une quantité et un prix.

Par la suite nous avons suivi les consignes du TP en créant son constructeur et ses fonctions getters qui permettent d'accéder aux variables membres de la classe ainsi que la méthode "setQuantity(int quantity)" qui nous est utile pour modifier la quantité du produit.

Nous avons également surcharger l'opérateur "<<" pour afficher les caractéristiques d'un produit comme demandé dans le TP. Nous avons ajouté une

surcharge de l'opérateur "==" qui initialement n'était pas demandé dans le TP afin de comparer plusieurs produits.

Client

Pour cette classe, on a inclus la classe Produit qui nous permettra de mettre en argument un objet de type Produit.
Elle comporte 4 variables membres : un identifiant, un prénom, un nom et un panier d'achat.

La dernière variable membre était initialement un tableau de type "vector<Product>" mais un problème est vite apparu : nous ne pouvions pas gérer les quantités simplement avec un simple std::vector. Si nous souhaitions ajouter un produit 5 fois au panier d'achat, il fallait faire 5 fois la commande "Ajouter le produit au panier" au lieu de simplement faire une seule commande "Ajouter le produit 5 fois au panier". Pour palier à ce problème, nous avons changé le tableau initial en un std::vector qui contient des paires de produit et d'un entier, représentant la quantité.

Nous avons ensuite suivi les consignes du TP en créant le constructeur, les fonction getters, ainsi que les méthodes : ajouter un produit au panier d'achat, vider le panier d'achat, modifier la quantité d'un produit du panier d'achat et enfin supprimer un produit du panier d'achat.

Pour l'ajout d'un produit au panier d'achat, on a utilisé la surcharge de l'opérateur "==" du produit pour s'assurer que lorsqu'on ajoute plusieurs fois le même produit dans le panier d'achat, la quantité de celui-ci augmente le nombre de fois ajouté. Pour ajouter une première fois un produit au panier, nous avons utilisé la fonction "make_pair" avec comme paramètre le produit et la quantité souhaités.

La fonction "clear" a été utilisée pour vider le panier d'achat.

Lors de la modification de la quantité, nous parcourons le panier d'achat en comparant à l'aide de la surcharge de "==" les produits du panier et celui passé en paramètre de la méthode. Si il y a une correspondance alors la quantité du produit du panier d'achat sera remplacée par celle passée en paramètre.
Même chose pour la suppression mise à part que lorsqu'il y a une correspondance entre les produits, nous utilisons la fonction "erase" pour effacer celui-ci du panier d'achat.

Pour finir nous avons comme pour la classe Produit, surcharger les opérateurs "<<" et "==" pour afficher et comparer les caractéristiques d'un client. Nous avons également ajouté une méthode appelée "displayShoppingCart()" permettant d'afficher seulement les produits du panier d'achat d'un client avec leurs quantités..

Commande

La classe commande a été la dernière classe créée de ce TP, elle a pour objectif de prendre en entrée le panier d'achat d'un client et de le transformer en commande, que l'utilisateur peut valider. Ses variables privées sont le client que le constructeur de la classe prend en paramètre, un std::vector<std::pair<Product,int>> reprenant les éléments du panier d'achat du client, et le statut, qui sera 0 ou 1.
Nous avons créé les getters des différentes variables privées, une méthode permettant d'afficher les éléments de la commande, une méthode permettant de modifier le statut de la commande, qui sera utilisée lors de la validation de cette dernière, et les surcharges d'opérateur "==" et "<<".

Magasin

La classe Magasin a été la première classe créée lors de ce TP, nous l'avons modifiée au fur et à mesure des créations des classes précédentes. Elle comporte 3 variables membres : un tableau de produits, un tableau de clients et un tableau de commandes, tous de type "vector<Objet>".
Nous avons dû ajouter plusieurs méthodes au magasin après chaque création de classe comme l'ajout d'un produit, d'un client et d'une commande, l'affichage de tous les produits, clients et commandes présents dans le magasin, l'affichage d'un produit et d'un client sélectionné par son nom ou son identifiant ainsi que toutes les fonctionnalités présentes dans chacune des classes .

Nous avons également ajouté plusieurs fonctions non demandés comme les getters des variables membres et les fonctions "bool IsProduct(Product& p)" et "bool IsClient(Client& c)" utilisées lors de l'ajout au magasin qui permettent de vérifier si les objets ont déjà été ajouté.
Les méthodes "Client& getShopClient(Client c)" et "Order& getShopOrder(Order o)" permettent d'influer sur les objets qui sont dans les tableaux du magasin et non les objets en eux-mêmes, cela était problématique lors de l'ajout d'un produit au panier d'achat d'un client depuis le magasin.
Nous nous sommes assurés de plusieurs éventualités dans cette classe comme le fait de mettre la quantité d'un produit à 0, dans ce cas le produit sera supprimé du magasin. L'ajout d'un produit au panier d'achat en trop grande quantité par rapport à celle disponible ainsi que la modification de la quantité qui ne peut dépasser celle du produit au risque d'avoir un message d'erreur.
Il y a également le fait que seules les commandes ayant un statut confirmé pourront être affichées. Une commande étant confirmée, le panier d'achat se vide automatiquement et les quantités des produits sont mises à jour.

Nous avons vérifié lors de la validation d'une commande que celle-ci contrôle bien si la commande est éligible c'est-à-dire si la quantité des produits est supérieure à celle du panier d'achat.

Menu

Ce menu à été créé à l'aide de 5 fonctions : Menu, Gestion_Shop, Gestion_Clients, Gestion_Orders et Gestion_File.
Comme ses nom l'indiquent, ces fonctions permettent d'effectuer toutes les fonctionnalités du magasin lié à chaque classe. Par exemple, la fonction Gestion_Shop permet d'effectuer toutes les méthodes relatives à la classe Produit comme l'ajout d'un produit ou l'affichage de tous les produits présents dans le magasin ou encore la fonction Gestion_Clients qui elle s'occupe des méthodes liées à la classe Client.
De plus la dernière fonction permet quant à elle de créer 3 fichiers nommés : Produits, Clients, Commandes et d'écrire dans ces fichiers toutes les informations relatives aux produits, clients et commandes présents dans le magasin. Ces données sont ensuite écrasées lors d'une nouvelle exécution du code.
La fonction Menu est basée sur une boucle while dont la condition d'arrêt est d'entrer le chiffre 0 qui lui même permet de sortir des sous menus créés dans les 4 autres fonctions. Elles ont toutes en paramètres un objet de type Shop& ainsi qu'une variable de type int& qui permet d'effectuer notre choix aux niveaux des commandes proposées par les sous menus. Les références sont nécessaires pour garder en mémoire les modifications des valeurs des variables passées en paramètre.
Notre fonction main se résume donc aux 3 lignes de codes suivantes :

```
Shop s1;
Menu(s1);
return 0;
```

Conclusion

Ce TP nous a permis d'améliorer nos connaissances sur le langage de programmation C++ avec notamment la création de plusieurs objets par copie ou encore celle du menu de l'application.
La différence majeure entre les 2 TP était l'interaction entre la machine et l'utilisateur ainsi que la création et l'écriture de fichiers qui est totalement nouveau pour nous. Ce sujet nous a également permis de renforcer nos connaissances sur les listes d'objets, après avoir fait ce TP, nous avons une autre perspective d'amélioration du TP1.
La compréhension de l'utilisation des références et des "const" peut être encore améliorée et nous permettrait d'être plus à l'aise à l'avenir.