INF1010

Programmation Orientée-Objet

Travail pratique #2
Vecteurs et surcharges 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 :	Dimanche 7 février 2016, 23h
Références :	Notes de cours sur Moodle & Chapitre 14 du livre Big C++ 2e éd.
Documents à remettre :	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

On veut dans ce travail continuer l'application du TP1 pour la rendre plus robuste et lui ajouter des fonctionnalités.

Pour remplacer les tableaux dynamiques qui limitaient le nombre d'abonnés, de livres et d'emprunts, 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.

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 des 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.

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 est fortement recommandé d'utiliser les fichiers fournis, plutôt que de continuer avec vos fichiers du TP1.

Classe *Abonne*

Créer une classe *Abonne* qui représente un abonné de la bibliothèque.

Les attributs et méthodes liées au TP1 restent inchangées, sauf si spécifié.

Cette classe contient les nouveaux attributs suivants :

- Vecteur d'emprunts (vecteur de pointeurs **Emprunt*).

Les nouvelles méthodes suivantes doivent être implémentées :

- Un destructeur (doit être modifié si requis).
- Une méthode d'accès (mais pas de modification) à la liste d'emprunts.
- Une méthode *obtenirNombreEmprunte()* qui retourne le nombre d'emprunts de l'abonné.
- Une méthode *estEmprunte()* qui prend en paramètre un *Livre* et vérifie si l'abonné possède un emprunt associé à ce livre. Si oui, la méthode retourne *true*, sinon *false.*
- Indice : Utilisez l'opérateur == de *Livre* pour vérifier si le livre n'est pas déjà emprunté.
- Une méthode *ajouterEmprunt()* qui prend en paramètre un pointeur *Emprunt*, et l'ajoute au vecteur.

- Une méthode *retirerEmprunt ()* qui prend en paramètre un *Livre.* La méthode enlève l'emprunt associé à ce livre du vecteur seulement s'il est présent. La méthode retourne un booléen *true* si l'emprunt est retiré et *false* s'il n'est pas retiré.
- L'opérateur << (remplace la méthode d'affichage), qui affiche les informations qui concernent un *Abonne*, tel que présenté dans l'exemple à la fin du document. Il doit appeler l'opérateur << de la classe *Emprunt*.
- L'opérateur == qui prend un *Abonne* en paramètre et permet de comparer 2 abonnés en considérant à la fois le matricule, le Nom, le Prénom et l'âge. L'opérateur retourne *true* si les abonnés sont identiques, *false* sinon. Ainsi, cet opérateur va pouvoir faire l'opération *abonne1 == abonne2*.
- L'opérateur == qui prend un matricule en paramètre et permet de comparer 2 abonnés en considérant seulement le matricule. L'opérateur retourne *true* si les matricules sont identiques, *false* sinon. Ainsi, cet opérateur va pouvoir faire l'opération *abonne == matricule*.
- L'opérateur == de type *friend* qui permet d'inverser l'ordre de l'opérateur== précédent. Ainsi, cet opérateur va pouvoir faire l'opération *matricule == abonne*.

Classe *Livre*

Créer une classe *Livre* qui représente un livre dans la bibliothèque.

Les attributs et méthodes liées au TP1 restent inchangées, sauf si spécifié.

Livre implémente les nouvelles méthodes suivantes :

- Un destructeur (doit être modifié si requis).
- Une méthode *convertirMinuscule()* qui permet de convertir une phrase en minuscule. Cette méthode **est déjà implémentée**.
- La méthode *recherche()* qui prend en paramètre un string et va ensuite vérifier s'il est contenu dans le livre. La méthode va donc retourner *true* si le string reçu est contenu dans le titre **ou** la cote; sinon la méthode retourne *false*.
    - Indice : Vous pouvez utiliser la méthode *find* de la classe std::string. http://www.cplusplus.com/reference/string/string/find/
    - Indice 2 : Vous pouvez utiliser la fonction *convertirMinuscule ()* pour éviter de prendre en compte les majuscules et minuscules.
- L'opérateur == qui prend un *Livre* en paramètre et permet de comparer 2 livres en considérant à la fois le titre et la cote d'un livre. L'opérateur retourne *true* si les livres sont identiques. Ainsi, cet opérateur va pouvoir faire l'opération *livre1 == livre2*.
- L'opérateur == qui prend une cote en paramètre et permet de comparer 2 livres en considérant seulement la cote. L'opérateur retourne *true* si les cotes sont identiques. Ainsi, cet opérateur va pouvoir faire l'opération *livre == cote*.
- L'opérateur == de type *friend* qui permet d'inverser l'ordre de l'opérateur== précédent. Ainsi, cet opérateur va pouvoir faire l'opération *cote == livre*.
- L'opérateur < qui prend en paramètre un *Livre* et permet de comparer les titres des livres en ordre alphabétique.

    - Ex : Si(Titre1 < Titre2 = true), donc Titre1 se positionne en 1er en ordre alphabétique.
    - Indice : Utilisez l'opérateur < qui est déjà implémenté pour les std::string.
- L'opérateur << (remplace la méthode d'affichage) qui affiche les informations qui concernent un *Livre*, tel que présenté dans l'exemple à la fin du document.
- Un constructeur par copie qui va copier tous les attributs.
- Un opérateur = qui écrase les attributs de l'objet de gauche par les attributs l'objet passés en paramètre.

Classe *Emprunt*

La classe *Emprunt* sert à représenter l'emprunt d'un livre par un abonné.

Les attributs et méthodes liées au TP1 restent inchangées, sauf si spécifié.

Emprunt possède les nouveaux attributs suivants :

- Le matricule de l'abonné (string), qui **remplace** le pointeur *Abonne*.

Emprunt implémente les méthodes suivantes :

- Les méthodes d'accès et de modification des nouveaux attributs.
- Modifier le constructeur par paramètre en fonction des nouveaux attributs.
- Un opérateur << (remplace la méthode d'affichage), qui affiche le matricule de l'abonné, puis appelle l'opérateur << de son *Livre*; tel que présenté dans l'exemple à la fin du document.

Classe *Bibliotheque*

La classe *Bibliotheque* est celle qui fait le lien entre toutes les classes précédentes.

Cette classe a subi beaucoup de changements. Il est recommandé de la recommencer à partir des fichiers fournis plutôt que de modifier le code du TP1.

Bibliotheque contient les attributs suivants :

- Vecteur de pointeurs d'abonnés.
- Vecteur de pointeurs de livres.
- Vecteur de pointeurs emprunt.

Bibliotheque implémente les méthodes suivantes :

Pour les méthodes d'ajout et de retrait, vous devez utiliser l'opérateur == approprié!

- Un constructeur par défaut.
- Un destructeur.
- La méthode *trouverAbonne()* qui prend un matricule (string) en paramètre, recherche le vecteur d'abonnés et retourne un pointeur *Abonne* associé à ce matricule. Si aucun abonné n'est retrouvé, un pointeur *nullptr* est retourné.

- La méthode *trouverLivre()* qui prend une cote (string) en paramètre, recherche le vecteur de livres et retourne un pointeur *Livre* associé à cette cote. Si aucun livre n'est retrouvé, un pointeur *nullptr* est retourné.
- La méthode *ajouterAbonne()* qui permet d'ajouter l'abonné reçu en paramètre, seulement s'il n'est pas déjà dans le vecteur.
- La méthode *retirerAbonne()* qui permet de retirer l'abonné en utilisant le matricule reçu en paramètre. Avant de le retirer, il faut **retourner tous les livres qu'il possède** en appelant la méthode appropriée.
- La méthode *ajouterLivre()* qui permet d'ajouter le livre reçu en paramètre seulement s'il n'est pas déjà dans le vecteur.
- La méthode *retirerLivre()* qui permet de retirer le livre en utilisant la cote reçue en paramètre. Le livre est retiré seulement s'il n'est pas emprunté. La méthode retourne donc un booléen *true* si le livre est retiré, sinon *false*.
- La méthode *rechercherLivre()* qui prend en paramètre une chaîne de caractères (string),
    - Cette méthode va rechercher les livres qui contiennent l'information désirée en **utilisant la méthode appropriée de la classe *Livre***. Pour chaque livre trouvé les informations seront affichées à l'aide de l'opérateur << de la classe *Livre*.
    - Si aucun livre n'est trouvé, affichez un message.
- Une méthode *estEmpruntable()* qui prend en paramètre un *Abonne* et un *Livre.*
    - Cette méthode vérifie s'il est possible pour l'abonné d'effectuer l'emprunt en question. Il est possible pour un abonné d'emprunter le livre si ce dernier est disponible, qu'il a l'âge minimal requis et qu'il n'a pas déjà emprunté ce livre.
    - Cette méthode doit retourner une valeur booléenne indiquant si l'emprunt est possible ou non.
- La méthode *emprunter()* qui prend en paramètres le matricule d'un abonné, la cote d'un livre et la date de retour. Cette méthode doit retourner une valeur booléenne indiquant si l'emprunt a été fait ou non.
    - Elle doit aussi appeler la méthode *estEmpruntable()*.
    - Elle doit s'assurer que le nombre d'emprunts ne dépasse pas la limite de 2 par abonné.
    - Elle doit diminuer le nombre de livres disponibles.
- La méthode *retourner()* qui prend en paramètres le matricule d'un abonné et la cote d'un livre. Si l'abonné avait bien emprunté ce livre, l'emprunt en question est retiré du vecteur d'emprunts.
    - Cette méthode doit retourner une valeur booléenne indiquant si le retour a été fait ou non.
    - Elle doit aussi appeler la méthode *retourner()* de la classe *Abonne*.
    - Si le livre est retourné, sa date de retour est mise à 0.
    - Elle doit augmenter le nombre de livres disponibles.
- La méthode *infoAbonne()* qui prend en paramètre un matricule d'abonné et affiche les informations qui le concerne en utilisant l'opérateur << approprié.
- Un opérateur >> qui permet d'entrer un mot-clé à chercher, puis qui appelle la méthode *rechercherLivre()*.
- L'opérateur += qui est défini 2 fois, avec un paramètre d'entrée différent :

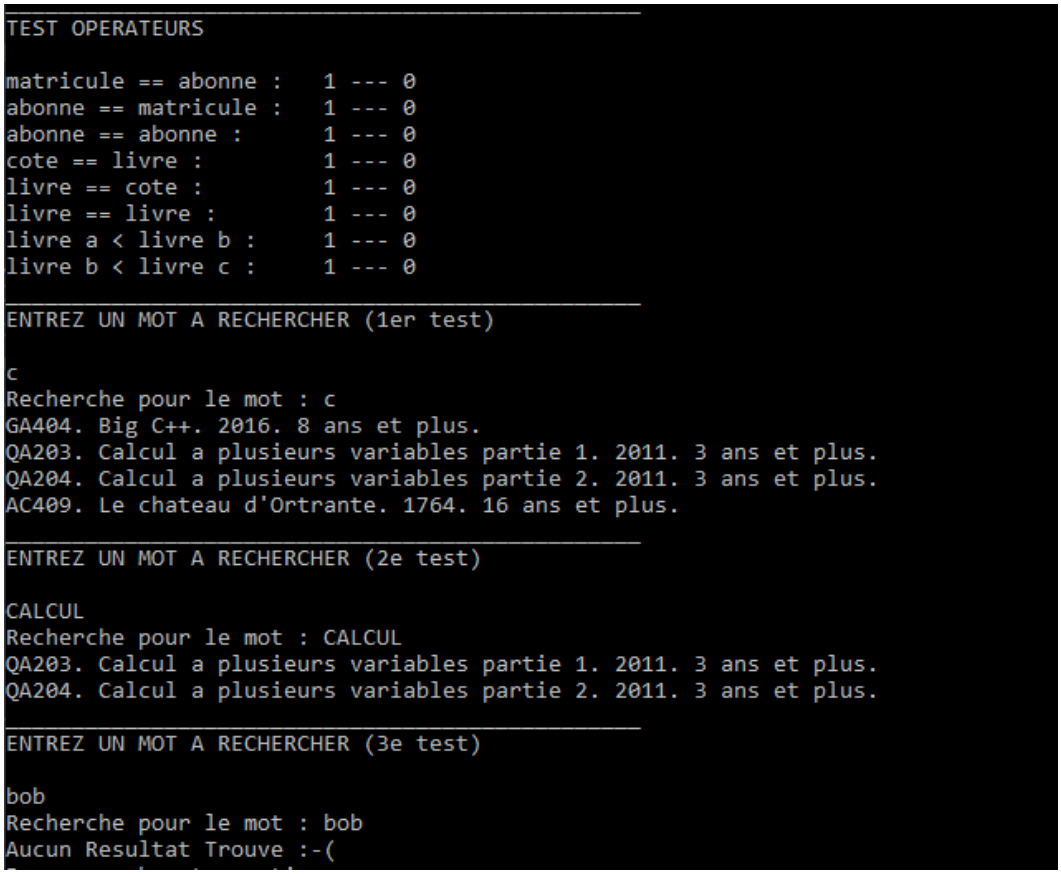
    - Une définition prend en paramètre un pointeur *Abonne*. Son comportement est similaire à *ajouterAbonne()*.
    - L'autre définition prend en paramètre un pointeur *Livre*. Son comportement est similaire à *ajouterLivre()*.
    - **Évitez de recopier des lignes de code!**
- L'opérateur -= qui est défini 2 fois, avec un paramètre d'entrée différent
    - Une définition prend en paramètre un pointeur *Abonne*. Son comportement est similaire à *retirerAbonne()*.
    - L'autre définition prend en paramètre un pointeur *Livre*. Son comportement est similaire à *retirerLivre()*.
    - **Évitez de recopier des lignes de code!**

Main.cpp

Le programme principal contient des directives à suivre pour instancier différents objets et essayer les différentes méthodes implémentées.

Le résultat final devrait être similaire à ce qui suit :

```
_____________________________________________________
CREATION ET AFFICHAGE DES ABONNES

John, Doe. 23 ans. #1839456
LISTE DE LIVRES :
Nicolas, Gagnon. 8 ans. #1630236
LISTE DE LIVRES :
Martin, Tremblay. 18 ans. #1269348
LISTE DE LIVRES :

_____________________________________________________
CREATION ET AFFICHAGE DES LIVRES

GA403. Big C++. 2009. 8 ans et plus.
QA203. Calcul a plusieurs variables partie 1. 2011. 3 ans et plus.
QA204. Calcul a plusieurs variables partie 2. 2011. 3 ans et plus.
AC409. Le chateau d'Ortrante. 1764. 16 ans et plus.
BD302. Harry Potter et le prisionier d'Azkaban. 1999. 3 ans et plus.
GA404. Big C++. 2016. 8 ans et plus.

_____________________________________________________
Ajout des livres et abonnes a la bibliotheque


_____________________________________________________
TESTS D'EMPRUNTS

Echec emprunt
AC409 emprunte par 1630236
Echec emprunt
Echec emprunt
AC409 emprunte par 1630236
Echec emprunt

_____________________________________________________
INFO ABONNE AVANT RETOUR

Nicolas, Gagnon. 8 ans. #1630236
LISTE DE LIVRES :
1 - Usager #1630236. Livre BD302. Harry Potter et le prisionier d'Azkaban. 1999. 3 ans et plus.
Retour prevu : 160204

2 - Usager #1630236. Livre QA204. Calcul a plusieurs variables partie 2. 2011. 3 ans et plus.
Retour prevu : 160204

Abonne - 1269348 - non trouve

_____________________________________________________
TESTS RETOUR LIVRE

BD302 remis par 1630236
Echec remise

_____________________________________________________
INFO ABONNE APRES RETOUR

Nicolas, Gagnon. 8 ans. #1630236
LISTE DE LIVRES :
1 - Usager #1630236. Livre QA204. Calcul a plusieurs variables partie 2. 2011. 3 ans et plus.
Retour prevu : 160204
```



Question

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 TP2 se fera sur 20 points.

Voici les détails de la correction:

- (03 points) Compilation du programme;
- (03 points) Exécution du programme;
- (04 points) Comportement exact des méthodes du programme;
- (03 points) Surcharge correcte des opérateurs;
- (02 points) Utilisation correcte des vecteurs;
- (1.5 points) Documentation du code;
- (01 point) Utilisation correcte du mot-clé *this* pour les opérateurs;
- (01 point) Utilisation correcte du mot-clé *const*;
- (1.5 points) Réponse à la question.