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 :	Mardi 7 février 2017, 8h
Références :	Notes de cours sur Moodle & Chapitre 14 du livre Big C++ 2e éd.
Documents à remettre :	La solution ainsi que 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

Informations préalables

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.

Travail à réaliser

Le travail consiste à continuer le système de gestion du personnel d'hôpital de Montréal commencé au TP précédent en y intégrant les notions de vecteurs et de surcharge d'opérateurs.

Pour remplacer les tableaux dynamiques qui limitaient le nombre de médecins et d'infirmiers, 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.

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 nécessaires.

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 *Infirmier*

Cette classe est caractérisée par un nom, un prénom, un infirmier s'occupe d'un nombre de chambre (nbChambres).

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

Les méthodes suivantes doivent être implémentées :

- La méthode qui retourne le nom complet (obtenirNomComplet), avec un espace de séparation entre le prénom et le nom.
- L'opérateur << (remplace la méthode d'affichage), qui affiche les informations qui concernent un *Infirmier*, tel que présenté dans l'exemple à la fin du document.
- L'opérateur == qui prend un *Infirmier* en paramètre et permet de comparer 2 *infirmiers* en considérant le nom complet. Cet opérateur va pouvoir faire l'opération *infirmier1 == infirmier2*.
- L'opérateur == qui prend un nom complet en paramètre et permet de comparer 1 infirmier en considérant seulement son nom complet avec la chaîne de caractères passé en paramètre. L'opérateur retourne *true* si les noms complets sont identiques, *false* sinon. Ainsi, cet opérateur va pouvoir faire l'opération *infirmier == nom c*omplet.
- L'opérateur == de type *friend* qui permet d'inverser l'ordre de l'opérateur== précédent. Ainsi, cet opérateur va pouvoir faire l'opération *nom c*omplet *== infirmier*.

Classe Specialite

Cette classe est caractérisée par un domaine et un niveau.

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

Les méthodes suivantes doivent être implémentées :

- L'opérateur << (remplace la méthode d'affichage), qui affiche les informations qui concernant une Specialite, tel que présenté dans l'exemple à la fin du document.

Classe *Medecin*

Cette classe est caractérisée par un nom, les horaires de travail, et une spécialité.

Les attributs et méthodes liées au TP1 restent inchangées, 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 l'objet passés en paramètre.
- L'opérateur << (remplace la méthode d'affichage), qui affiche les informations qui concernent un *Medecin*, tel que présenté dans l'exemple à la fin du document.

- L'opérateur == qui prend un *Medecin* en paramètre et permet de comparer 2 *médecins* en considérant le nom. Cet opérateur va pouvoir faire l'opération *medecin1 == medecin2*.
- L'opérateur == qui prend un nom en paramètre et permet de comparer 1 médecin avec un chaine de caractère. L'opérateur retourne *true* si les noms sont identiques, *false* sinon. Ainsi, cet opérateur va pouvoir faire l'opération *medecin == nom*.
- L'opérateur == de type *friend* qui permet d'inverser l'ordre de l'opérateur== précédent. Ainsi, cet opérateur va pouvoir faire l'opération *nom == medecin*.

Classe *Personnel*

Cette classe sert à sauvegarder et présenter les objets de type Medecin et Infirmier.

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

Cette classe contient les attributs privés suivants :

- *medecins_* : Un Vecteur de pointeurs Medecin, qui contiendra les différents médecins
- *infirmiers_* : Un Vecteur de pointeurs Infirmier, qui contiendra les différents infirmiers

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut qui initialise les attributs aux valeurs par défaut.
- Un destructeur
- Une méthode *ajouterMedecin()* qui permet d'ajouter un ***médecin*** reçu en paramètre, deux médecins ne peuvent pas avoir le même nom. **Pensez à utiliser l'opérateur == surchargé pour un *Medecin.***
- La méthode *retirerMedecin()* qui permet de retirer le médecin en utilisant le nom reçu en paramètre. **Pensez à utiliser l'opérateur == surchargé pour un *Medecin.***
- L'opérateur += qui prend en paramètre un médecin, qui l'ajoute au vecteur de medecins et qui retourne la Personnel avec les nouvelles modifications. Le comportement de cette surcharge d'opérateur doit être similaire à *ajouterMedecin*. Évitez de recopier du code.
- L'opérateur -= qui prend en paramètre un médecin, qui l'enlève du vecteur de médecin et qui retourne le Personnel avec les nouvelles modifications. Le comportement de cette surcharge d'opérateur doit être similaire à *retirerMedecin*. Évitez de recopier du code.
- Une méthode *ajouterInfirmier()* qui permet d'ajouter un ***infirmier*** reçu en paramètre, deux médecins ne peuvent pas avoir le même nom. **Pensez à utiliser l'opérateur == surchargé pour un *Infirmier.***
- La méthode *retireInfirmier()* qui permet de retirer l'infirmier en utilisant le nom reçu en paramètre. **Pensez à utiliser l'opérateur == surchargé pour un Infirmier.**
- L'opérateur += qui prend en paramètre un infirmier, qui l'ajoute au vecteur d'infirmiers et qui retourne la Personnel avec les nouvelles modifications. Le comportement de cette surcharge d'opérateur doit être similaire à *ajouterInfirmier*. Évitez de recopier du code.

- L'opérateur -= qui prend en paramètre un infirmier, qui l'enlève du vecteur d'infirmiers et qui retourne la Personnel avec les nouvelles modifications. Le comportement de cette surcharge d'opérateur doit être similaire à *retirerInfirmier*. Évitez de recopier du code.
- L'opérateur << (remplace la méthode d'affichage), qui affiche les informations qui concernant *Personnel*, tel que présenté dans l'exemple à la fin du document.

Aide : Pensez à utiliser les différentes méthodes pour afficher les informations des différentes classes Medecin et Infirmier.

Classe *Hopital*

Cette classe représente un hôpital qui fait la gestion des personnels Medecin et Infirmier.

Cette classe contient les attributs privés suivants :

- nom_ : Une variable de type string
- personnel_ : Un pointeur de type Personnel

Les méthodes suivantes doivent être implémentées :

- Un constructeur par paramètres qui initialise les attributs aux valeurs correspondantes.
- Des méthodes d'accès et de modification des deux attributs (nom_, personnel_).
- L'opérateur << qui affiche les informations qui concernant *L'hôpital*, tel que présenté dans l'exemple à la fin du document.

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. Vous êtes libre de proposer un rendu plus ergonomique et plus agréable ainsi que de choisir vos propres exemples de noms, prénoms, spécialité, etc. :

```
Roger Lamarre a bien été ajouté !
Linda Laplante a bien été ajouté !
Hug Latour a bien été ajouté !
Emilie Lessard a bien été ajouté !
Emilie Lessard n'a pas été ajoute
Franc a bien été ajouté !
Sherlock a bien été ajouté !
Holmes a bien été ajouté !
Jean a bien été ajouté !
Jules a bien été ajouté !
Sherlock a bien été ajouté !
Holmes a bien été ajouté !
Kyle a bien été ajouté !
House a bien été ajouté !
Franc a bien été retire !
Franc a bien été ajouté !
Eric Langlais n'a pas été ajoute
Eric Langlais n'a pas été ajoute
                                    Hôpital Sacré-Coeur
================================================================================================
                                    Tableau Medecins
------------------------------------------------------------------------------------------------
| Nom                  | Horaires            | Domaine Specialite   |   Niveau Specialite  |
------------------------------------------------------------------------------------------------
| Jules                |          9          | Pediatrie            |          9           |
| Sherlock             |          5          | Demartologie         |         10           |
| Holmes               |          6          | Gastrologie          |         11           |
| Jean                 |          8          | Podologie            |          8           |
------------------------------------------------------------------------------------------------

                    Tableau Infimiers
------------------------------------------------------
| Nom Complet          | Nombre de Chambre    |
------------------------------------------------------
| John Doe             |          10          |
| Gilles Tremblay      |           2          |
| Sylvie Labe          |           3          |
| Amelie Labelle       |           4          |
| Maxime Lamontagne    |           5          |
| John Laflamme        |           6          |
| Julie Lamoureux      |           7          |
------------------------------------------------------
================================================================================================

                                    Hôpital Jean-Talon
================================================================================================
                                    Tableau Medecins
------------------------------------------------------------------------------------------------
| Nom                  | Horaires            | Domaine Specialite   |   Niveau Specialite  |
------------------------------------------------------------------------------------------------
| Sherlock             |          6          | Demartologie         |         10           |
| Holmes               |          3          | Gastrologie          |         11           |
| Kyle                 |          7          | Sport                |          7           |
| House                |         10          | Psychiatrie          |         45           |
| Franc                |          9          | Chirurgie            |         15           |
------------------------------------------------------------------------------------------------

                    Tableau Infimiers
------------------------------------------------------
| Nom Complet          | Nombre de Chambre    |
------------------------------------------------------
| Eric Langlais        |           8          |
| Roger Lamarre        |           9          |
| Linda Laplante       |          10          |
| Hug Latour           |          11          |
| Emilie Lessard       |          12          |
------------------------------------------------------
================================================================================================
```

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
- Note : Lorsqu'il est fait référence aux valeurs par défaut, pour un string cela équivaut à la chaine vide, et pour un entier au nombre 0

Questions

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 à la question.