INF1010

Programmation Orientée-Objet

Travail pratique #2

Vecteurs, Composition et agrégation, Constructeur de copie et affectation,

et Surcharge d'opérateurs

Objectifs : Permettre à l'étudiant de se familiariser avec la surcharge d'opérateurs, les vecteurs de la librairie STL, la composition et l'agrégation, les constructeurs de copie et l'opérateur d'affectation.

Remise du travail : Dimanche 24 Mai 2020, 23h55.

Références : Notes de cours sur Moodle & livre Big C++ 2e éd.

Documents à remettre : Les fichiers .cpp et .h **uniquement**, complétés et réunis sous la forme d'une archive au format **.zip**. Le nom du fichier devrait être : matricule1_matricule2_X.zip avec X = L1 ou L2 selon votre groupe.

Directives : Directives de remise des Travaux pratiques sur Moodle

Les entêtes (fichiers, fonctions) et les commentaires sont obligatoires.

Les travaux dirigés s'effectuent obligatoirement en équipe de deux personnes faisant partie du même groupe.

Pas de remise possible sans être dans un groupe.

Veuillez suivre le guide de codage

Mise en contexte

Pour faire face à la crise sanitaire actuelle, le Centre hospitalier de l'Université de Montréal a besoin de mettre en place un système de consultation en ligne afin d'évaluer l'état des patients avant leur arrivée sur place et d'appliquer davantage les mesures de la distanciation sociale et limiter les risques de contamination entre les personnes.

Pour ce faire, le CHUM fait appel aux étudiants de Polytechnique Montréal pour implémenter un système capable de répondre à leurs besoins. Ayant une grande confiance au niveau des étudiants du cours INF1010, l'école a décidé de vous confier ce projet qui s'étalerait sur 5 travaux pratiques.

Pour ce deuxième TP, on vous demande d'intégrer les systèmes de gestion des patients et des médecins précédemment développés dans un Hôpital virtuel en effectuant des modifications dans le code pour intégrer les notions vues en cours. On vous demande aussi d'ajouter la gestion de consultation virtuelle à l'hôpital.

Travail à réaliser

On vous demande de compléter les fichiers qui vous sont fournis pour pouvoir implémenter le système décrit ci-dessus.

Les fichiers Patient.h, Medecin.h, GestionnairePatients.h, GestionnaireMedecins.h, Consultation.h, Hopital.h et main.cpp vous sont fournis. Vous devez apporter des modifications dans les fichiers .h pour ajouter les opérateurs à surcharger, utiliser des vecteurs à la place des tableaux dynamiques et compléter les fichiers .cpp selon les directives ci-dessous.

Notes importantes :

- **Attention à bien lire les spécifications générales plus bas.**
- **On ne vous demande pas le contrôle des entrées (par exemple l'unicité des ID, la validité des dates …)**
- **Les fonctions doivent être implémentées dans le même ordre que la définition de la classe**
- **Utilisez la liste d'initialisation pour l'implémentation de vos constructeurs. L'ordre dans la liste doit être le même ordre que la définition des attributs de la classe.**
- **N'utilisez pas "using namespace std". Exemple d'usage accepté : std ::string**

Classe Patient

Cette classe représente un patient avec son nom, sa date de naissance et son numéro d'assurance maladie.

La méthode suivante doit être retirée :

- La méthode ***afficher*** doit être retirée. Elle sera remplacée par l'opérateur <<.

Les méthodes suivantes doivent être implémentées :

- L'opérateur << doit être implémenté pour remplacer la méthode ***afficher***.
- L'opérateur == qui compare un Patient avec un string qui représente le numéro d'assurance maladie d'un patient.
- Exemple: patient == "TREMR124520". **L'opération doit pouvoir se faire dans les deux sens**. C'est-à-dire : patient == " TREMR124520" et " TREMR124520" == patient doivent tous les deux fonctionner. Cet opérateur doit retourner ***true*** lorsque le numéro d'assurance maladie du patient est égal au string.

Classe Medecin

Cette classe représente un médecin qui exerce à l'hôpital.

Vous devez modifier les attributs afin de remplacer le tableau par un vecteur. Retirez tous les attributs que vous jugez dorénavant inutiles avec le remplacement du tableau par un vecteur.

Les méthodes suivantes doivent être retirées :

- La méthode ***afficher*** doit être retirée. Elle sera remplacée par l'opérateur <<.
- La méthode ***ajouterPatient*** doit être retirée. Elle sera remplacée par l'opérateur +=.
- La méthode ***supprimerPatient*** doit être retirée. Elle sera remplacée par l'opérateur -=.

Les méthodes suivantes doivent être implémentées :

- L'opérateur << doit être implémenté pour remplacer la méthode ***afficher***.
- L'opérateur == qui compare un médecin avec un string qui représente le numéro de la licence du médecin.

- Exemple: medecin == "158795". **L'opération doit pouvoir se faire dans les deux sens**. C'est-à-dire : medecin == " 158795" et " 158795" == patient doivent tous les deux fonctionner. Cet opérateur doit retourner true lorsque le numéro de la licence du médecin est égal au string.
- L'pérateur += qui permet d'ajouter un patient dans le vecteur ***patientsAssocies_*** s'il n'existe pas parmi les patients associés au médecin. Il retourne ***true*** si l'opération d'ajout est réussie. Il prend en paramètre une référence vers le patient à ajouter. Cette méthode remplace la méthode ***ajouterPatient***.
- L'opérateur -= permet de supprimer un patient du vecteur ***patientsAssocies_***. L'opérateur prend en paramètre le numéro d'assurance maladie et retourne ***true*** si l'opération est réussie. Cette méthode remplace la méthode ***supprimerPatient***.
- La méthode ***chercherPatient***(*const std::string&*) : cette méthode permet de chercher un patient en utilisant son numéro d'assurance social. Elle retourne un pointeur vers le patient si on le trouve, si non elle retourne nullptr.

Classe GestionnairePatients

Cette classe permet de gérer les patients.

Vous devez modifier les attributs afin de remplacer le tableau par un vecteur. Retirez tous les attributs que vous jugez dorénavant inutiles avec le remplacement du tableau par un vecteur.

Les méthodes suivantes doivent être retirées :

- La méthode ***afficher*** doit être retirée. Elle sera remplacée par l'opérateur <<
- La méthode ***ajouterPatient*** doit être retirée. Elle sera remplacée par l'opérateur +=.
-

Les méthodes suivantes doivent être implémentées :

- Constructeur de copie
- L'opérateur d'affectation
- L'opérateur << doit être implémenté pour remplacer la méthode ***afficher***.
- L'opérateur += qui permet d'ajouter un patient dans le vecteur ***patient_*** s'il n'y existe pas déjà et si le maximum ***NB_PATIENT_MAX*** n'est pas dépassé. Il retourne ***true*** si l'opération d'ajout est réussie. Il prend en paramètre une référence vers le patient à ajouter. Cette méthode remplace la méthode ***ajouterPatient***.
- ***getPatients*** qui retourne une référence constante vers le vecteur ***patients_***.

Les méthodes suivantes doivent être modifiées :

- La méthode ***lireLignePatient*** doit utiliser l'opérateur += pour ajouter les patients
- ***getNbPatients*** doit être adaptée au vecteur.
- ***chercherPatient*** doit être adaptée au vecteur. Elle doit utiliser l'opérateur == de la classe Patient.
- ***chargerDepuisFichier*** doit être adaptée au vecteur. Il faut vider le vecteur avant de charger les patients.

Classe GestionnaireMedecin

Cette classe permet la gestion des médecins de l'hôpital.

Vous devez modifier les attributs afin de remplacer le tableau par un vecteur. Retirez tous les attributs que vous jugez dorénavant inutiles avec le remplacement du tableau par un vecteur.

Les méthodes suivantes doivent être retirées :

- La méthode ***afficher*** doit être retirée. Elle sera remplacée par l'opérateur <<.
- La méthode ***ajouterMedecin*** doit être retirée. Elle sera remplacée par l'opérateur +=.
- La méthode ***supprimerMedecin*** doit être retirée. Elle sera remplacée par l'opérateur -=.

Les méthodes suivantes doivent être implémentées :

- Le constructeur de copie
- L'opérateur d'affectation
- L'opérateur << doit être implémenté pour remplacer la méthode ***afficher***.
- L'opérateur += qui permet d'ajouter un medecin dans le vecteur ***medecins_*** s'il n'existe pas déjà. Il retourne ***true*** si l'opération d'ajout est réussie. Il prend en paramètre une référence vers le medecin à ajouter. Cette méthode remplace la méthode ***ajouterMedecin***.
- L'opérateur -= permet de supprimer un médecin du vecteur ***médecins_***. L'opérateur prend en paramètre le numéro de licence de médecin à supprimer et retourne ***true*** si l'opération est réussie. Cette méthode remplace la méthode ***supprimerMedecin***.
- ***getMedecins*** qui retourne une référence constante vers le vecteur.

Les méthodes suivantes doivent être modifiées :

- La méthode ***lireLigneMedecin*** doit utiliser l'opérateur += pour ajouter les médecins.
- ***getNbMedecins*** doit être adaptée au vecteur.

- ***chercherMedecin*** doit être adaptée au vecteur. Elle doit utiliser l'opérateur == de la classe Medecin.
- ***chargerDepuisFichier*** doit être adaptée au vecteur. Il faut vider le vecteur avant de charger les médecins.
- ***trouverIndexMedecin*** doit être adaptée au vecteur. Elle doit utiliser l'opérateur == de la classe Medecin.

Classe Consultation

Cette classe permet de gérer les consultations de l'hôpital.

Elle contient les attributs suivants :

- medecin_ (`Medecin*`) : le médecin responsable de la consultation.
- patient_ (`Patient*`) : le patient qui demande la consultation.
- date_ (`std::string`) : la date de la consultation.

Les méthodes suivantes doivent être implémentées :

- Un constructeur par paramètres qui initialise les attributs aux valeurs correspondantes. Il permet aussi d'ajouter le patient au vecteur ***patientsAssocies***_ du médecin s'il n'y existe pas déjà.
- L'opérateur << pour afficher les informations d'une consultation. Exemple d'affichage :

```
Consultation:
        Medecin: Tamard
        Patient: Simon | Date de naissance: 25/10/92 | Numero d'assurance maladie: 1000000
        Date de consultation: 05/05/2020
```

Classe Hopital

Cette classe permet la gestion de l'hôpital.

Elle contient les attributs suivants :

- nom _ (`std::string`) : le nom de l'hôpital
- adresse_ (`std::string`) : l'adresse de l'hôpital
- gestionnaireMedecins_ (`GestionnaireMedecins`) : le gestionnaire des médecins.
- gestionnairePatients_ (`GestionnairePatients`) : le gestionnaire des patients.
- consultations_ un vecteur des consultations

Les méthodes suivantes doivent être implémentées :

- ***chargerBaseDeDonnees***(`const std::string&, const std::string&`) : Cette méthode prend en paramètre le nom de fichier des médecins et des patients et puis elle charge les deux fichiers. Elle retourne ***true*** si l'opération est réussie.
- ***getConsultations*** : retourne une référence constante vers le vecteur ***consultations_***
- L'opérateur += qui ajoute une consultation au vecteur ***consultations_***. Il prend en paramètre une référence vers une consultation. La consultation est ajoutée si le médecin est actif et il existe dans le gestionnaire des médecins et le patient dans le gestionnaire des patients. La méthode retourne ***true*** si l'opération de l'ajout est réussie.
- L'opérateur += qui ajoute un médecin au gestionnaire des médecins. Il prend en paramètre une référence vers le médecin à ajouter. La méthode retourne ***true*** si l'opération de l'ajout est réussie.
- L'opérateur += qui ajoute un patient au gestionnaire des patients. Il prend en paramètre une référence vers le patient à ajouter. La méthode retourne ***true*** si l'opération de l'ajout est réussie.

Main.cpp

Le fichier main.cpp vous est fourni et ne devrait pas avoir à être modifié pour la remise. N'hésitez pas à commenter certaines parties si vous avez besoin de compiler votre code. Vous pouvez aussi utiliser ce fichier pour mieux comprendre les requis des fonctions.

Une fois tous les fichiers complétés, tous les tests devraient passer.

Fichiers complémentaires

En plus des fichiers ci-dessus, l'énoncé contient le fichier debogageMemoire.hpp. Ce dernier est présent pour vous aider à vérifier s'il y a des fuites de mémoire dans votre code. Exécutez la solution en débogage pour qu'il fonctionne. Si une fuite de mémoire est détectée, un message sera affiché dans la fenêtre Sortie (Output) de Visual studio.

Exemple d’affichage

Voici un exemple d’affichage du programme que vous pouvez utiliser pour vous inspirer. Pour votre travail, vous devez afficher les mêmes informations. Cependant, vous êtes libres au niveau de la forme et de l’esthétique.

```
Patient: Simon | Date de naissance: 25/10/92 | Numero d'assurance maladie: 1000000


Medecin: Tamard
        Numero de licence: 1000000
        Specialite: Cardiologue
        Statut: Actif
        Aucun patient n'est suivi par ce medecin.



Medecin: Tamard
        Numero de licence: 1000000
        Specialite: Cardiologue
        Statut: Actif
        Patients associes:
                Patient: Simon | Date de naissance: 25/10/92 | Numero d'assurance maladie: 1000000



Medecin: Tamard
        Numero de licence: 1000000
        Specialite: Cardiologue
        Statut: Actif
        Aucun patient n'est suivi par ce medecin.


Patient: John Lourdes | Date de naissance: 12/12/2001 | Numero d'assurance maladie: louj010304
Patient: George Lucas | Date de naissance: 01/04/1944 | Numero d'assurance maladie: LUCG441212



Medecin: Lourdes John
        Numero de licence: tt1234
        Specialite: Cardiologue
        Statut: Actif
        Aucun patient n'est suivi par ce medecin.


Medecin: George Lucas
        Numero de licence: hhjue2
        Specialite: Dermatologue
        Statut: Actif
        Aucun patient n'est suivi par ce medecin.
```

```
Consultation:
               Medecin: Tamard
               Patient: Simon | Date de naissance: 25/10/92 | Numero d'assurance maladie: 1000000
               Date de consultation: 05/05/2020

Test 1: OK!    0.25/0.25
Test 2: OK!    1/1
Test 3: OK!    0.25/0.25
Test 4: OK!    1/1
Test 5: OK!    0.5/0.5
Test 6: OK!    0.5/0.5
Test 7: OK!    0.25/0.25
Test 8: OK!    0.25/0.25
Test 9: OK!    0.5/0.5
Test 10: OK!   0.5/0.5
Test 11: OK!   0.5/0.5
Test 12: OK!   0.25/0.25
Test 13: OK!   0.25/0.25
Test 14: OK!   0.25/0.25
Test 15: OK!   0.5/0.5
Test 16: OK!   0.5/0.5
Test 17: OK!   0.5/0.5
Test 18: OK!   0.5/0.5
Test 19: OK!   0.25/0.25
Test 20: OK!   0.25/0.25
Test 21: OK!   0.25/0.25
Test 22: OK!   0.25/0.25
Test 23: OK!   0.25/0.25
Test 24: OK!   0.25/0.25
Test 25: OK!   0.25/0.25
TOTAL:          10/10
```

Spécifications générales

- Ajoutez un destructeur pour chaque classe lorsque nécessaire.
- Utilisez la liste d’initialisation pour l’implémentation de vos constructeurs.
- Documentez votre code source.
- Ne remettez que les fichiers .h et .cpp lors de votre remise.
- Bien lire le barème de correction ci-dessous.
- Ne modifier pas la signature des méthodes sauf c’est indiqué.

Correction
La correction du TP2 se fera sur 20 points.

Voici les détails de la correction :

- (2 points) Compilation du programme
- (2 points) Exécution du programme
- (10 points) Comportement exact des méthodes du programme
- (2 points) Documentation du code et bonne norme de codage
- (2 points) Manipulation correcte des vecteurs
- (2 points) Surcharge correcte des opérateurs