INF1010

Programmation Orientée-Objet

Travail pratique #1
Concepts de base, Méthodes constantes, passage de paramètres et Pointeurs intelligents

---

Objectifs : Permettre à l'étudiant de se familiariser avec les notions de base de la programmation orientée objet, l'allocation dynamique de la mémoire en manipulant les pointeurs intelligents, le passage de paramètres et les méthodes constantes.

Remise du travail : Dimanche 17 Mai 2020, 23h55.

Références : Notes de cours sur Moodle & Chapitre 2-7 du 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**.

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

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'entêtes (.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.

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 premier TP, on vous demande d'implémenter le système de gestion des patients et des médecins de l'hôpital virtuel.

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 et main.cpp vous sont fournis. On vous demande de compléter les fichiers *.cpp en suivant les indications qui vont suivre.

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.

Cette classe contient les attributs privés suivants :

- nom_(*std::string*) : le nom complet du patient (prénom + nom)
- dateDeNaissance_(*std::string*) : la date de naissance du patient
- numeroAssuranceMaladie_(*std::string*) : le numéro de la carte d'assurance maladie du patient. Il est utilisé comme ID du patient.

Les méthodes suivantes doivent être implémentées :

- Un constructeur par paramètres qui initialise les attributs aux valeurs correspondantes.
- Les méthodes d'accès aux attributs (getters).
- Les méthodes de modification des attributs (setters).

Classe Medecin

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

Elle contient les attributs privés suivants :

- *nom_* (*std::string*) : le nom complet du médecin (prénom + nom)
- *numeroLicence_* (*std::string*) : le numéro de la licence de pratique du médecin. Il est utilisé comme ID du médecin
- *specialite_* (Specialite) : la spécialité du médecin
- *estActif_* (bool) : l'état du médecin dans la clinique. Cet attribut sert à distinguer un médecin présentement actif dans l'hôpital, d'un médecin qui ne l'est pas.
- *patientAssociés_* (*std::shared_ptr<shared_ptr*<Patient>[]>) : un tableau des patients du médecin
- *nbPatientsAssocies_* (*size_t*) : le nombre actuel des patients gérés par le médecin
- *capacitePatientsAssocies_* (*size_t*) : La capacité maximale d'un médecin à gérer des patients.

Les méthodes suivantes doivent être implémentées :

- Un constructeur par paramètres qui initialise les attributs aux valeurs correspondantes. Utilisez la constante *CAPACITE_PATIENTS_INITIALE* pour initialiser la capacité du tableau des patients associés au médecin.
- Les méthodes d'accès aux attributs (getters).
- Les méthodes de modifications des attributs (setters).
- *ajouterPatient* (`Patient`) : ajoute un patient au tableau *patientAssocies_*. (Attention : vérifier si la taille du tableau le permet, et réagir dans le cas contraire en doublant la capacité du tableau). On doit pouvoir toujours ajouter un patient.
- *afficher* (`std::ostream&`) : affiche toutes les informations concernant un médecin et ses patients associés (voir exemple d'affichage).
- *supprimerPatient* (`const std::string&`) : supprime le patient par son ID du tableau *patientAssocies_* d'un médecin.

Classe GestionnairePatients

Cette classe permet de gérer les patients

Elle contient les attributs privés suivants :

- *NB_PATIENT_MAX* (`static constexpr size_t`) : nombre maximal de patients dans la liste.
- *patients_* (`Patient[]`) : tableau d'une taille fixe (NB_PATIENT_MAX) contenant tous les patients de l'hôpital.
- *nbPatients_* (`size_t`) : nombre de patients dans le tableau *patients_*

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut qui initialise les attributs aux valeurs par défaut.
- *ajouterPatient* (`const Patient&`) : Cette méthode ajoute un patient à la liste du patient. Elle retourne un booléen pour indiquer le succès de l'ajout.
- *chercherPatient* (`const std::string&`) : Cette méthode cherche dans la liste le patient dont le numéro d'assurance maladie est passé en paramètre. Elle retourne un pointeur vers le patient du même numéro. Si aucun patient n'est trouvé, elle retourne un pointeur nul.
- *chargerDepuisFichier* (`const std::string&`) : Cette méthode prend en paramètre le nom de fichier à charger. Elle doit ouvrir le fichier, et envoyer chaque ligne du fichier à la fonction *lireLignePatient*. Si une erreur se produit dans *lireLignePatient*, elle retourne false. Retourner true si la lecture du fichier est bien réussie.

- *lireLignePatient* (const std::string&) : Cette méthode privée prend un string en paramètre et doit ajouter un patient à partir de ce string. Le format du string est "Nom Patient" "Date de naissance" "Numéro assurance maladie"
  Exemple : "Michelle Trembley" "12/01/1999" "MICHT120199"
  Hint : utiliser std ::quoted pour extraire les informations entre les apostrophes.
- *afficher* (std::ostream&) : Cette méthode permet d'afficher, dans le stream passé en paramètres, les informations de tous les patients dans la liste des patients.
- *getNbPatients*() : Cette méthode retourne le nombre de patients dans la liste.

Classe GestionnaireMedecin

Cette classe permet la gestion des médecins de l'hôpital

Elle contient les attributs privés suivants :

- *medecins_* (std::unique_ptr<shared_ptr<Medecin>[]>) : tableau des médecins de l'hôpital
- *nbMedecins_* (size_t) : nombre de médecins de l'hôpital
- *capaciteMedecin* (size_t) : capacité maximale de l'hôpital pour embaucher des médecins

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut qui initialise les attributs aux valeurs par défaut. Utilisez la constante *CAPACITE_MEDECINS_INITIALE* comme pour celui de la classe Medecin.
- *ajouterMedecin* (std::unique_ptr<Medecin>): ajoute un médecin à l'hôpital. (Attention : vérifier si la taille du tableau le permet, et réagir dans le cas contraire en doublant la capacité du tableau). On doit pouvoir toujours ajouter un médecin.
- *chercherMedecin* (const std::string&) : cherche dans la liste le médecin dont le numéro de licence est passé en paramètre. Elle retourne un pointeur vers le médecin du même nom. Si aucun patient n'est trouvé, elle retourne un pointeur nul.
- *chargerDepuisFichier* (const std::string&) : Cette méthode prend en paramètre le nom de fichier à charger. Elle doit ouvrir le fichier, et envoyer chaque ligne du fichier à la fonction *lireLigneMedecin*. Si une erreur se produit dans lireLigneMedecin, elle retourne false. Retourner true si la lecture du fichier est bien réussie.
- *lireLigneMedecin* (const std::string&) : Cette méthode privée prend un string en paramètre et doit ajouter un médecin à partir de ce string. Le format du string est "Nom Médecin" "Numéro de Licence" indexSpécialité(entre 0 et 6)
  Exemple : "Michelle Trembley" "tt1234" 1
  Hint : utiliser std ::quoted pour extraire les informations entre les apostrophes

- *afficher* (*std::*ostream&) : Cette méthode permet d'afficher, dans le stream passé en paramètres, les informations de tous les médecins dans la liste des médecins.
- *supprimerMedecin* (const *std::*string&) : Cette méthode permet de supprimer un médecin de la liste *medecins_*. La suppression d'un médecin n'est pas permanente. En effet, il suffit de rendre le médecin inactif pour le supprimer, car on aurait toujours besoin de ses informations dans le futur. Elle prend en paramètre le numéro de licence du médecin à supprimer. Elle fait appel à la méthode *trouveIndexMedecin*() pour supprimer le médecin approprié.
- *trouverIndexMedecin* (const *std::string&*) : Cette méthode privée prend en paramètre le numéro de licence du médecin à supprimer. Elle cherche le médecin dans la liste *medecins_* et retourne son index. Elle retourne *MEDECIN_INEXSISTANT* si le médecin n'est pas trouvé.
- *getNbMedecins*() : Cette méthode retourne le nombre de médecins dans la liste des médecins.

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: 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.



Medecin: Robert Trembley
       Numero de licence: ROB1234
       Specialite: Cardiologue
       Statut: Actif
       Patients associes:
              Patient: Patient0 | Date de naissance: 12/12/2001 | Numero d'assurance maladie: SETT1212010
              Patient: Patient1 | Date de naissance: 12/12/2001 | Numero d'assurance maladie: SETT1212011
              Patient: Patient2 | Date de naissance: 12/12/2001 | Numero d'assurance maladie: SETT1212012

Test 1: OK!    0.5/0.5
Test 2: OK!    0.5/0.5
Test 3: OK!    1/1
Test 4: OK!    0.5/0.5
Test 5: OK!    1/1
Test 6: OK!    0.5/0.5
Test 7: OK!    0.5/0.5
Test 8: OK!    1/1
Test 9: OK!    1/1
Test 10: OK!   1.5/1.5
TOTAL:          8/8
```

Spécifications générales

- Ajoutez un destructeur pour chaque classe lorsque nécessaire
- Utilisez la liste d'initialisation pour l'implémentation de vos constructeurs.
- Ajoutez le mot-clé *const* chaque fois que cela est pertinent.
- Ne modifiez pas les signatures des méthodes, **sauf pour ajouter le mot-clé *const***
- Documentez votre code source.
- Ne remettez **que** les fichiers .h et .cpp lors de votre remise.
- Bien lire le barème de correction ci-dessous.

Correction

La correction du TP1 se fera sur 20 points.

Voici les détails de la correction :

- (3 points) Compilation du programme
- (2 points) Exécution du programme
- (8 points) Comportement exact des méthodes du programme
- (2 points) Documentation du code et bonne norme de codage
- (2 points) Utilisation correcte du mot-clé *const* et dans les endroits appropriés
- (3 points) Utilisation appropriée des pointeurs intelligents