INF1010

Programmation Orientée-Objet

Travail pratique #1
Allocation dynamique, composition et agrégation et Vector STL

À remettre un fichier . zip contenant les fichiers .h et .cpp des classes avant le 21 janvier 23h 30

À la fin de ce laboratoire, vous devrez être capable de :

- Comprendre le concept de classe et objet
- Faire des allocations dynamiques
- Passer des paramètres à des méthodes
- Comprendre les concepts de composition et d'agrégation
- Utiliser des vecteurs de la STL C++

Directives :

- Les travaux s'effectuent obligatoirement en équipe de deux personnes faisant partie du même groupe.
- Ne touchez pas aux entêtes des fichiers fournis sauf si on vous le demande.
- Les fonctions que vous décidez d'ajouter au programme doivent être documentées.

Conseils :

- Lisez les conseils, l'aperçu et les spécifications avant de commencer!
- Ayez lu vos notes de cours!
- Si vous avez des difficultés au cours du TP, rappelez-vous que de nombreux problèmes demandés sont assez couramment rencontrés en C++.
- Consulter la documentation sur cppreference, les notes de cours, et les forums sur Google peuvent vous donner de bonnes pistes de résolution!
- Si votre programme ne compile pas, veuillez mettre en commentaires les instructions qui ne compilent pas.
- Relisez votre code après l'avoir écrit pour éviter les erreurs de syntaxe.

Spécifications générales

- Tout warning à la compilation sera pénalisé. Si vous utilisez Visual Studio de Microsoft, vous devez activer l'option /W4. Sur Visual Studio, assurez-vous de compiler avec x64, certains warnings pourraient ne pas s'afficher sinon.
- Des TODOs sont présents dans le code pour vous guider. Vous devez les compléter.
- Toutes les méthodes doivent être définies dans le fichier d'implémentation (.cpp) dans le même ordre que leur déclaration dans le fichier d'en-tête (.h). Le non-respect de cette règle entraînera une pénalité au niveau du style.

- Utilisez le plus possible la liste d'initialisation des constructeurs. L'utilisation du corps des constructeurs à la place de la liste d'initialisation entraînera une pénalité de style.
- L'ordre des variables (attributs) dans la liste d'initialisation doit être la même que celle dans la liste des attributs de la classe dans le fichier d'en-tête
- Suivez le guide de codage sur Moodle.

Mise en contexte

En tant que développeur stagiaire dans un hôpital, votre tâche sera de mettre en place un système de gestion du personnel. Dans un premier temps, vous serez amené à gérer les listes de médecins et d'infirmiers ainsi que leur salaire. Il est à noter que chaque médecin a une spécialité ou non.

Le projet contient les classes suivantes :
Medecin : Représente un médecin ayant une spécialité ou aucune.
Infirmier : Représente un infirmier.
Specialité : représente une spécialité d'un médecin
HopitalPoly : Représente un hôpital qui conserve les listes de médecins et d'infirmiers.

Travail à réaliser

Les fichiers Medecin.h, Infirmier.h Specialite.h et HopitalPoly.h vous sont fournis, vous devez compléter les implémentation des classes.

Classe *Infirmier*

Cette classe est caractérisée par un nom, un prénom et un infirmier s'occupe d'une liste de chambres.

Cette classe contient les attributs privés suivants :

- Un nom (string)
- Un prénom (string)
- Un taux horaire (float)
- Un tableau de chambres (vector de string)
- Le total des chambres est le nombre total des chambres dont l'infirmier a terminé de s'occuper de la chambre. Cela représente son expertise.
- Le nombre d'heures travaillées.

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut et paramètres qui initialise les attributs aux valeurs correspondantes. De plus le total des chambres est initialisé à 0 ainsi que les heures travaillées.
- Les méthodes d'accès aux attributs.
- Les méthodes de modifications des attributs.
- La méthode chercherChambre.
- La méthode qui ajoute une chambre dans la liste des chambres si elle n'existe pas.
- La méthode qui retire une chambre de la liste des chambres et incrémente le total des chambres
- La méthode calculerSalaire. Le salaire correspond à heuresTravaillees_ * tauxHoraire_ + totalChambres_ * tauxHoraire_/20. Les heures travaillées sont mises à 0.
- La méthode afficher()

Classe Specialite

Cette classe est caractérisée par un domaine et un niveau.

Cette classe contient les attributs privés suivants :

- Un domaine (string)
- Un niveau (entier)

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut qui initialise les attributs aux valeurs par défaut : domaine = " " et niveau à 0,
- Un constructeur par paramètres qui initialise les attributs aux valeurs correspondantes.
- Les méthodes d'accès aux attributs.
- Les méthodes de modifications des attributs.
- La méthode afficher().

Classe *Medecin*

Cette classe est caractérisée par un nom, un salaire, et une spécialité.

Cette classe contient les attributs privés suivants :

- Un nom (string).
- Une spécialité (Specialite).
- Un salaire.

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut qui initialise le nom et les autres attributs aux valeurs par défaut.

- Un constructeur par paramètres qui initialise les attributs aux valeurs correspondantes.
- Les méthodes d'accès aux attributs.
- Les méthodes de modifications des attributs.
- La méthode calculerSalaire qui correspond à salaire_ * specialite_->getNiveau().
- La méthode afficher()

Classe *HopitalPoly*

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

Cette classe contient les attributs privés suivants :

- Le nom de l'hôpital (string).
- Un vector de pointeurs à des médecins, qui contiendra les différents médecins.
- Un vector de pointeurs à des infirmiers, qui contiendra les différents infirmiers.

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut qui initialise le nom de l'hôpital.
- Les méthodes d'accès aux attributs.
- Les méthodes de modifications des attributs.
- Une méthode chercherMedecin qui vérifie si le médecin est dans le vector.
- Une méthode ajouterMedecin() qui prend en paramètre un pointeur de médecin et l'ajoute au vector de médecins s'il n'existe pas.
- Une méthode supprimerMedecin() qui prend en paramètre un pointeur de médecin et le supprime du vector de médecins s'il existe.
- Idem pour les méthodes chercherInfirmier, ajouterInfirmier et supprimerInfirmier.
- Une méthode afficherMedecins() qui affiche la liste des médecins présents dans le tableau de médecins.
- Une méthode afficherInfirmier() qui affiche la liste des infirmier présents dans le tableau d'infirmiers.
- Une méthode afficher().

Main.cpp

Les tests unitaires pourront être exécutés quand vous aurez des implémentations pour les classes. Pour les exécuter, vous devrez compiler le projet et exécuter le fichier main.cpp. Il est interdit de modifier le fichier main.cpp, Testsuite.h et TestSuite.cpp. Pour le format des affichages, il faut vous fier au format dans le fichier main.cpp. Vous avez un makefile pour compiler et créer l'exécutable.

Correction

La correction du TP1 se fera sur 20 points.

Voici les détails de la correction :

- (4 points) Compilation du programme ;
- (4 points) Exécution du programme ;
- (2 points) Qualité du code;
- (10 points) Comportement exact de l'application