TP #4

POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

TP4

INF1010- Programmation orientée objet

Hiver 2025
Département de génie informatique
École Polytechnique de Montréal

À remettre le: 1 avril avant 23h30

Département de génie informatique — GIGL

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

- Utiliser la STL avec les conteneurs et les algorithmes
- Comprendre les foncteurs et les fonctions lambda
- Utiliser des itérateurs

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.
- Un Makefile est fourni pour vous aider à compiler votre programme si vous utilisez Linux et Visual Studio Code. Le Makefile est exclusif pour Linux.

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.

1 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 x86, certains warnings pourraient ne pas s'afficher sinon.
- Des TODOs sont présents dans le code pour vous guider. Vous devez les compléter et les enlever après avoir termine le TP.
- 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.

2 Aperçu des classes du projet

Vu le nombre de classe, une description de chaque classe est fournie pour vous aider à comprendre le projet qui est la même que dans le TP3, mais le travail à faire est indiqué dans les TODOs. Des restrictions sont aussi indiquées dans les TODOs, vous perdrez des points si vous ne les respectez pas.

2.1 ComparerPersonne

2.1.1 Introduction

La classe `ComparerPersonne` est un foncteur permettant de comparer deux objets de type `T*`, en se basant sur leur nom.

2.1.2 Méthodes

`bool operator()(const shared_ptr &person1, const shared_ptr &person2) const` Cette méthode compare les noms de deux objets pointés par des pointeurs intelligents de type `shared_ptr`. Elle retourne `true` si le nom du premier objet est lexicographiquement inférieur au nom du second.

2.2 ComparerSalaire

2.2.1 Introduction

La classe `ComparerSalaire` est un foncteur permettant de comparer les salaires de deux objets de type `T*`.

2.2.2 Méthodes

`bool operator()(const T* elem1, const T* elem2) const` Cette méthode compare les salaires des deux objets donnés. Elle retourne `true` si le salaire du premier objet est strictement inférieur à celui du second.

2.3 HashEmploye

2.3.1 Introduction

La classe `HashEmploye` est un foncteur permettant de générer une valeur de hachage pour un objet de type `Employe*`.

2.3.2 Méthodes

`size_t operator()(const Employe* a) const` Cette méthode calcule et retourne une valeur de hachage unique pour l'objet employé donné. Cette valeur est utilisée pour organiser efficacement les objets dans des conteneurs basés sur le hachage.

2.4 MedecinRegionEloigne

2.4.1 Introduction

La classe `MedecinRegionEloigne` représente un médecin spécialisé en médecine dans les régions éloignées. Cette classe hérite des classes `Medecin`, `Chirurgien`, `Pediatre`, et `Cardiologue`, ce qui lui permet d'exercer plusieurs spécialités simultanément.

2.4.2 Attributs

- `nbVoyagesEnHelicopteres_` : Un entier représentant le nombre de voyages en hélicoptère effectués pour soigner des patients en région éloignée.

2.4.3 Méthodes

`unsigned getNbVoyagesEnHelicopteres()` Retourne le nombre de voyages en hélicoptère effectués par le médecin.

`float calculerSalaire() const` Calcule et retourne le salaire du médecin. Celui-ci est déterminé en fonction de la formule suivante:

$$\begin{aligned}\text{Salaire} = \text{salaireBase} \times \text{niveauSpecialiteChirurgien} + 500 \times \text{nbVoyagesHelicopteres} \\ + 20 \times \text{nbPatients} + 30 \times \text{nbOperations} + 50 \times \text{nbEnfantsSoignes} \quad (1)\end{aligned}$$

`void visiterPatientEnHelicoptere(shared_ptr& p)` Incrémente le nombre de voyages en hélicoptère et ajoute un antécédent médical au patient pour indiquer qu'il a été transporté par le médecin.

`ostream& afficher(ostream& out) const` Affiche les informations du médecin, incluant ses niveaux de spécialisation, son nombre de voyages en hélicoptère et son salaire.

2.5 ListeMedecin

La classe `ListeMedecin` est une classe générique (template) qui gère une liste de médecins en utilisant une **structure de tas (heap)** pour assurer l'ordre des médecins en fonction de leur salaire. Cette classe repose sur les algorithmes de la **STL (Standard Template Library)** pour effectuer différentes opérations efficacement.

2.5.1 Attributs

- `heapMedecin_` : Un *vecteur* contenant des pointeurs vers des objets de type `T`, représentant des médecins. Ce vecteur est maintenu sous la forme d'un *tas* grâce aux fonctions `push_heap` et `pop_heap` de la STL.

2.5.2 Méthodes

`void ajouterMedecin(T *medecin)` Ajoute un médecin à la liste. Cette méthode utilise `push_heap` pour garantir que le tas respecte l'ordre défini par le foncteur `ComparerSalaire`, qui compare les médecins en fonction de leur salaire.

`template <class U, class Z> void ajouterMedecin(T *medecin)` Ajoute un médecin à la liste uniquement si ce dernier est de type `U` ou `Z` (via `dynamic_cast`). Comme pour la première méthode, `push_heap` est utilisé pour maintenir la propriété du tas.

`size_t size() const` Retourne le nombre de médecins présents dans la liste.

`double getSalaireTotal() const` Calcule et retourne la somme des salaires de tous les médecins de la liste. Utilise `std::accumulate` pour additionner les salaires sans utiliser de boucle explicite.

`T* getMedecinAvecPlusHautSalaire() const` Retourne le médecin ayant le plus haut salaire. Comme le tas est structuré de manière à ce que l'élément ayant le plus grand salaire soit toujours en premier, cette méthode retourne simplement l'élément en tête du vecteur (`heapMedecin_.front()`).

`T* enleverMedecinAvecPlusHautSalaire()` Supprime le médecin ayant le salaire le plus élevé de la liste. Cette méthode :

- Utilise `pop_heap` pour déplacer l'élément ayant le plus grand salaire à la fin du vecteur.

- Supprime cet élément avec `pop_back()`.
- Retourne un pointeur vers le médecin supprimé.

2.6 HopitalPoly

2.6.1 Introduction

La classe `HopitalPoly` gère un hôpital en maintenant une liste d'employés et de patients. Elle utilise des structures de données modernes comme **vector**, **map**, **set**, et **unordered_map** pour organiser les médecins et patients de manière efficace. Elle repose fortement sur les algorithmes de la **STL (Standard Template Library)** pour optimiser les opérations courantes. Les TODOs dans cette classe ne concernent que les méthodes utilisant la librairie STL.

2.6.2 Attributs

- `nom_` : Une chaîne de caractères représentant le nom de l'hôpital.
- `chambreLibre_` : Un entier représentant le nombre de chambres libres dans l'hôpital.
- `employes_` : Un *vecteur* de pointeurs intelligents vers des objets `Employe`, représentant les employés de l'hôpital.
- `patients_` : Un *vecteur* de pointeurs intelligents vers des objets `Patient`, représentant les patients de l'hôpital.
- `medecinTousPatients_` : Une *unordered_ map* associant un `Medecin*` à un `set` de `shared_ptr`, permettant de gérer efficacement les patients attribués à chaque médecin.

2.6.3 Méthodes

`map<TypeSoins, vector<shared_ptr» trouverPatientParTypeSoins()` Retourne une `map` associant chaque `TypeSoins` à une liste de patients nécessitant ce type de soins.

`set<shared_ptr, ComparerPersonne> getMedecinParCritere(function<bool(const shared_ptr&)> critere) const` Retourne un ensemble de médecins respectant un critère donné.

`Medecin* listersi(const function<bool(const Employe&)> &critere) const` Retourne un pointeur sur le premier médecin respectant un critère donné.

`void ajouterPatientAuMedecin(string &id, const shared_ptr &patient)` Ajoute un patient à un médecin en recherchant ce dernier par son ID. Utilise `listersi()` et modifie la `unordered_map medecinTousPatients_`.

`unordered_map<Medecin*, set<shared_ptr, ComparerPersonne>, HashEmploye> getPatientsParMedecin()` Retourne la `unordered_map` associant chaque médecin à ses patients.

`float getSalaireTotal() const` Calcule et retourne la somme des salaires de tous les employés de l'hôpital.

`void afficherEmploye(ostream& out) const` Affiche la liste des employés de l'hôpital en utilisant `for_each`.

3 Tests unitaires

Les tests unitaires pourrons être exécutés quand vous aurez des implémentations pour toutes les classes. Pour les exécuter, vous devrez compiler le projet et exécuter le fichier main.cpp.

4 Compilation du projet avec le Makefile

Pour compiler le projet sur Linux exclusivement, vous pouvez utiliser le Makefile fourni. Vous pouvez compiler le projet en utilisant la commande make dans le terminal. Pour exécuter le projet, vous pouvez utiliser la commande ./int1010-tp.

5 Création d'une solution avec Visual Studio

Pour la solution avec Visual Studio, vous pouvez créer un nouveau projet C++ et ajouter le dossier include dans les includes supplémentaires du projet.

6 Grille d'évaluation

Total	**20 pt**
Compilation du programme sans avertissements	4 pt
Exécution du programme sans erreur	4 pt
Qualité du code	2 pt
Comportement exact de l'application	10 pt

⚠ Remise du travail avant le 1 avril avant 23h30 :

- Une note de 0 sera attribuée aux équipes qui remettent leur travail en retard.
- SVP ne mettez pas de commentaires dans le code
- Remettre tous les fichiers .cpp et .h SAUF le main.cpp et testSuite.cpp directement sous une archive .zip (sans dossier à l'intérieur et sans autres fichiers).
- Toute archive autre que zip ou remise ne respectant pas ces consignes sera refusée et se méritera une note de 0. Attention, une archive .7z ne compte pas comme une archive zip.
- Respectez le format de la remise! Nom de l'archive zip : matricule1_matricule2_groupe.zip Exemple : 1234567_1234568_1.zip