UNIVERSITÉ DE SHERBROOKE
DÉPARTEMENT D'INFORMATIQUE

IFT 339

Travail pratique#1 : Implémentation d'une matrice

Dans ce travail, vous allez implémenter une matrice, qui est une table à deux dimensions, en la représentant par un tableau de tableaux dynamiques. Le TP est une occasion de mettre en pratique les classes, l'encapsulation, les templates et la gestion de pointeurs. Le travail comporte deux parties pour l'implémentation de la matrice, puis pour son utilisation. Quelques modalités :

- Ce travail compte pour **5 %** de la session.
- **La date de remise de ce travail est le jeudi 19 septembre à 23h59. Vous devez remettre sur `https://turnin.dinf.usherbrooke.ca/` les fichiers générés au cours de ce travail.**
- Il n'y a **pas de script de correction automatique.** Vous êtes responsables de générer vos propres cas de test. Le fichier main.cpp donné contient quelques tests, mais il ne contient pas tous les cas possibles.

Partie 1 : implémentation d'une matrice

Une matrice de dimension $n \times m$ peut être vue comme un tableau avec n rangées et m colonnes. Vous devez implémenter une classe servant à stocker une matrice, dont une partie vous est donnée. La matrice doit réaliser les fonctionnalités suivantes :

construction(n, m) : permet de construire une nouvelle matrice avec n rangées et m colonnes. Après construction, la matrice contient $n \cdot m$ cellules, dont les valeurs ne sont pas initialisées.
construction(n, m, elements) : permet de construire une nouvelle matrice avec n rangées et m colonnes initialisée avec les élements donnés. Ici, **elements** est un vecteur dont les m premiers éléments sont les entrées de la première rangée, les m prochaines éléments ceux de la rangée 2, et ainsi de suite.

at(r, c) : permet d'obtenir *une référence* sur l'élément stocké à la rangée r et à la colonne c, où les rangées et colonnes sont indicées à 0. On doit vérifier que r est entre 0 et $n-1$ et c est entre 0 et $m-1$, et sinon on lance une exception.
Doit permettre d'obtenir *et* affecter les valeurs d'une cellule, par exemple

```
Matrice<int> mat(10, 10);
mat.at(2, 2) = 10;
```

doit modifier la matrice à la troisième rangée et troisième colonne.

operator(r, c) : permet d'obtenir *une référence* sur l'élément stocké à la rangée r et à la colonne c, comme le at. Par contre, on ne vérifie pas que r et c sont dans les bornes. Notez que la classe vector a le même comportement. Exemple :

```
Matrice<int> mat(10, 10);
mat(2, 2) = 10;
```

redimensionner(n, m) : permet d'attribuer les dimensions de la matrice, où n est le nouveau nombre de rangées et m le nouveau nombre de colonnes. Les valeurs des cellules encore présentes avant et après le redimensionnement doivent demeurer telles quelles dans la matrice. Si une dimension est rétrécie, les cellules en dehors des nouvelles bornes ne doivent plus être stockées par votre matrice. Si une dimension est aggrandie, les valeurs aux cellules nouvellement créées ne sont pas initialisées.

get_dimensions() : permet d'obtenir les dimensions n et m de la matrice, dans une objet pair de la librairie standard (déjà implémenté).

swap_rangees(r1, r2) : échange le contenu de la rangée $r1$ avec le contenu de la rangée $r2$.

swap_colonnes(c1, c2) : échange le contenu de la colonne $c1$ avec le contenu de la colonne $c2$.

copieur(src) : constructeur qui, en recevant une matrice src, copie ses dimensions et ses éléments.

operator= : surcharge de l'affectateur, qui copie la matrice passée et retourne une référence vers la matrice courante.

destructeur : nettoie la mémoire allouée dynamiquement.

Vous pouvez ajouter des méthodes **privées** à la classe fournie, au besoin.

Partie 2 : utilisation de votre matrice

Dans cette deuxième partie, vous devenez une personne qui utilise votre matrice pour vérifier si elle est *bien gardée*. Soit M une matrice booléenne, c'est-à-dire, dont les entrées sont $vrai$ ou $faux$. On dit que M est *bien gardée* si, pour chaque rangée i et chaque colonne j, il y a au moins une cellule à la rangée i ou à la colonne j dont la valeur est $vrai$.

En d'autres termes, *bien gardée* veut dire que si vous imaginez que cellules $vrai$ comme des gardes qui peuvent voir toute leur rangée et toute leur colonne, alors chaque cellule est visible par au moins un garde. Bref, s'il y a une cellule avec aucun garde sur sa rangée ni sur sa colonne, il faut retourner $faux$.

Fournissez une implémentation pour la fonction est_gardee dans main.cpp, qui reçoit une matrice booléenne M est retourne $true$ si et seulement si M est bien gardée.

Vous aurez *presque* tous vos points pour une implémentation qui retourne la bonne réponse sur toute entrée. Pour un maximum de points, votre méthode doit prendre un temps proportionnel à la taille de la matrice (en terme de complexité, un temps $O(n^2)$). Elle devrait être capable de terminer en un temps raisonnable sur une matrice de taille 1000×1000.

Un fichier exemple main.cpp vous est aussi fourni, ainsi que la sortie attendue en l'exécutant une fois votre implémentation complétée. Notez que le fait que vous obteniez la sortie attendue n'implique pas une note de 100% : vous êtes responsable de fournir un code libre de bugs, même

s'ils ne sont pas identifés par le main.cpp fourni.

Évaluation

Respect des spécifications (45 points) : votre code doit effectuer les opérations mentionnées ci-haut de façon correcte. Ceci inclut l'obtention de la sortie attendue ainsi que l'absence d'autres possibles bugs.

Qualité du code (45 points) : assurez-vous de bien nettoyer la mémoire allouée dynamiquement, de ne pas avoir de code inutile, surtout d'éviter la répétition de code lorsque possible, et de respecter le principe d'encapsulation.

Respect des normes de programmation (10 points) : utilisez soit les normes proposées sur le site du cours, ou bien celles des notes de cours, ou bien vos propres normes, mais soyez constant(e)s.

Remise du travail

Vous devez remettre deux fichiers matrice.h et main.cpp. Vous pouvez remettre les deux fichiers séparément ou bien dans un format compressé (.zip ou .tar.gz). Ne remettez pas d'autre fichier ni surtout d'exécutable. Il ne doit y avoir qu'une seule remise par équipe, à partir d'un seul CIP. Les noms des co-équipiers doivent être clairement indiqués en entête de chaque fichier soumis. Notez qu'il n'y a pas de script de correction automatique - votre code sera inspecté à la correction.