

POA Labo 1 : Matrix

Introduction

Ce laboratoire a pour but de développer un programme permettant de créer des matrices en 2 dimensions qui possèdent comme éléments des entiers. Il est possible d'effectuer des opérations arithmétiques entre deux matrices. Chacune des opérations possède plusieurs signatures de fonction, l'utilisateur choisi à sa convenance une d'entre elle selon la signature qui lui convient.

Choix de conception

Gestion de l'allocation mémoire

Pour la création des matrices, nous avons défini deux méthodes privées qui sont *allocate()* et *deleteTab()*. La première alloue dynamiquement l'attribut *tab* qui a eu sa taille spécifiée en paramètre lors de l'appel au constructeur de la matrice. Nous utilisons un tableau classique à 2 dimensions pour *tab*. Cette fonction d'allocation est utilisée dans les constructeurs d'une matrice. La méthode *deleteTab* désalloue la mémoire en restituant la taille allouée pour l'attribut *tab* d'une matrice. C'est dans le destructeur qu'elle est appelée.

Construction d'une matrice

Il y a trois constructeurs de matrice définis, deux sont publiques et un privé. Une matrice peut être créée en spécifiant le nombre de ligne, le nombre de colonne et un modulo correspondant à la valeur maximale possible qui sera attribuée à un élément de cette matrice. Ce constructeur initialise aléatoirement les éléments de la matrice, avec des valeurs entre 0 et n-1 si n est la valeur du modulo. Si un des trois paramètres vaut zéro (un ou plus), une exception de type *runtime_error* est levée. En effet la matrice n'aurait plus de sens à être créée avec une telle valeur.

Le deuxième constructeur publique est celui de copie à partir d'une matrice existante. Ce constructeur exploite le constructeur privé qui est défini de façon à pouvoir copier le contenu d'une matrice, mais en donnant la possibilité que la matrice résultante soit plus grande que la matrice copiée. Ce cas est utile lors des opérations sur matrices, en particulier lorsqu'il faut créer une nouvelle matrice pour le résultat.

Les constructeurs appellent *allocate()* pour l'allocation mémoire au moment de la création d'une instance. Le constructeur de copie fait appel à la méthode privée *copyTab()* qui effectue une copie des éléments de la matrice à copier, c'est-à-dire l'attribut *tab*. *CopyTab()* prévoit le cas où la nouvelle matrice peut avoir une taille plus grande que celle copiée, dans ce cas les éléments non copiable sont mis à 0.

Il n'est possible de créer une matrice en lui donnant des tailles négatives, les tailles étant des *size_t*, la valeur négatives est alors converties en non signé d'une valeur trop grande pour créer un tableau et une erreur est générée. Un modulo négatif est autorisé et sans erreur mais donne comme valeur 2^{32} - valeur.

Opérations sur matrices

Les opérations disponibles sont l'addition, la soustraction et la multiplication entre deux matrices. Chacune de ces opérations est définie en trois versions différentes :

1. `Matrix& operation(const Matrix& rhs)`

L'opération s'effectue sur la matrice implicite `this` et `rhs`, modifie `this` puis retourne une référence.

2. `Matrix operationToCpy(const Matrix& rhs) const`

Une nouvelle matrice temporaire est créée pour y stocker le résultat sans modifier les deux matrices.

Le retour de la fonction renvoie cette matrice temporaire.

3. `Matrix* operationDynamic(const Matrix& rhs) const`

Une nouvelle matrice est créée dynamiquement pour y stocker le résultat sans modifier les deux matrices.

Le retour de la fonction renvoie un pointeur sur la matrice résultante qui vient d'être allouée.

Pour la mise en oeuvre de ces opérations, la méthode privée *for_each* est définie pour factoriser le code semblable à chaque opération. Cette méthode s'effectue donc sur deux matrices, l'une étant passée implicitement en paramètre et l'autre explicitement. Elle attend également en paramètre un pointeur sur une fonction *f* prenant deux entiers en paramètre. C'est cette fonction *f* qui définira le type d'opération à effectuer entre deux éléments des matrices.

La méthode *for_each* lève une exception si la valeur des modules est différente dans les 2 matrices. Si les tailles de matrices diffèrent, le résultat sera de taille $\max(M1, M2) \times \max(N1, N2)$. Les éléments hors taille commune aux matrices seront simplement initialisés à 0. Il existe cependant un cas où une exception est levée si ces tailles diffèrent, lorsque l'opération choisie correspond à la 1ère déclaration qui est définie ci-dessus. En effet, la matrice implicite ne va pas voir sa taille modifiée pour satisfaire l'opération. Les deux autres déclarations ont l'avantage de créer sur le moment une matrice et la taille est adaptée selon la taille correcte au moment de la construction de la matrice qui fera l'objet du résultat.

Opérateur d'affectation

Une matrice existante peut se voir affecter les valeurs d'une autre matrice. Pour le réaliser, nous avons surchargé l'opérateur d'affectation. Si les dimensions ou le modulo ne coïncident pas avec la matrice à copier, cela n'a pas d'importance et tout sera semblable au retour de la fonction. Avec comme particularité que si les dimensions diffèrent, la matrice est désallouée puis réallouée avec la bonne taille.

Opérateur d'écriture sur un flux

Une fonction privée a été définie pour surcharger l'opérateur d'écriture sur un flux pour pouvoir afficher une matrice. L'affichage a pour format :

a1 a2 a3

b1 b2 b3

c1 c2 c3

Gestion de l'initialisation aléatoire

Pour avoir une utilisation cohérente du `rand` fourni par la librairie standard, et appeler un « seed » une seule fois durant le programme, nous avons créé une classe implémentant le design pattern de singleton.

Le constructeur privé va établir le « seed » une seul fois. Une méthode getInstance permet de récupérer une instance unique de cette classe. Finalement une méthode getUnsigned permet de récupérer un entier positif aléatoire. L'initialisation se fait donc sans nombre négatif, il peut toutefois en avoir dans une matrice en utilisant la soustraction.

Tests de fonctionnement

Test effectué	Résultat attendu	Résultat obtenu
Opération de même taille	Ok	Ok
Opération de taille différente	Ok	Ok
Constructeur avec nombre aléatoire entre 0 et n-1	Ok	Ok
Dupliquer une matrice (constructeur de copie)	Ok	Ok
Affectation de même taille	Ok	Ok
Affectation de taille différente	Ok	Ok
Operation de même modulo	Ok	Ok
Operation de modulus différents	Erreur	Erreur
Modulo mathématique (modulo avec nombre négatif et positif)	Ok	Ok
Afficher une matrice avec l'opérateur de flux	Ok	Ok
Les opérations sont effectuées modulo n	Ok	Ok
Chaque opération peut modifier la matrice sur laquelle est invoquée la méthode	Ok	Ok
Chaque opération peut retourner par valeur une nouvelle matrice résultat allouée statiquement	Ok	Ok
Chaque opération peut retourner un pointeur sur une nouvelle matrice résultat allouée dyn.	Ok	Ok
L'opération de taille différente donne en résultat une taille de $\max(M1, M2) \times \max(N1, N2)$	Ok	Ok
Lors d'une op. de taille différente, $A_{i,j}$ et $B_{i,j}$ manquants sont remplacés par des 0	Ok	Ok
Exception de type invalid_argument levée si les modulus sont différents	Ok	Ok
Exception de type runtime_error pour toute autre erreur	Ok	Ok

Test effectué	Résultat attendu	Résultat obtenu
Construction matrice N = 0, M et modulo quelconque	Erreur	Erreur
Construction matrice M = 0, N et modulo quelconque	Erreur	Erreur
Construction matrice modulo = 0, M et N quelconque	Erreur	Erreur
Construction matrice 1 seule ligne	Ok	Ok
Construction matrice 1 seule colonne	Ok	Ok
Construction matrice init. à 0 avec modulo = 1	Ok	Ok
Construction matrice avec taille négative	Erreur	Erreur
Matrice avec modulo négatif	Ok	Ok

UML

