Quentin Forestier, Herzig Melvyn

POO2 – 03.03.2021

Matrices binaires

Laboratoire 5



Table des matières

[Introduction 2](#_Toc54018509)

[Attributs 2](#_Toc54018510)

[Constructeurs 2](#_Toc54018511)

[Exceptions 2](#_Toc54018512)

[Affichage 3](#_Toc54018513)

[Opérations 3](#_Toc54018514)

[Exceptions 4](#_Toc54018515)

[Tests 4](#_Toc54018516)

[Résultats 4](#_Toc54018517)

[Annexes 6](#_Toc54018518)

# Introduction

Le but de ce laboratoire et de se familiariser avec la programmation orienté objet en C++. Nous avons pour but d’implémenter une classe « Matrix » qui modélise des matrices de taille H x L dont le contenu est défini modulo M. Entre autres, ces matrices pourront être additionnées, soustraites et multipliées.

Attributs

Une matrice est définie par son contenu que nous stockons sous formes de tableau à deux dimension alloué dynamiquement.

# Constructeurs

Une matrice peut être construite de deux manières :

* **Contenu aléatoire** : L’utilisateur passe la taille H x L et le modulo M en argument. Le contenu est généré aléatoirement modulo P. Une classe Utils permet d’appeler un générateur aléatoire.
* **Copie**: L’utilisateur passe en paramètre un objet Matrix qui sera copié.
* **Par défaut**: Initialise une matrice de taille 0x0, de modulo 1. Ce constructeur permet d’utiliser la classe avec les structures de la STL qui construisent des objets par défaut.

## Exceptions

Le constructeur de contenu aléatoire lève une runtime\_error si l’un de ses trois paramètre vaut 0.

# Surcharge

Différents opérateurs ont été surchargés :

* operator<<
* operator=
* operator+
* operator+=
* operator\*
* operator\*=
* operator-
* operator-=

Les opérateurs arithmétiques se basent sur des méthodes privées qui demandent un objet MatrixOperator.

# Opérations

Les opérations matricielles s’effectuent composante par composante.

Si les matrices opérantes sont de tailles différentes, la matrice résultante sera de taille  
 max(M1,M2) x max(N1,N2) où M1,N1,M2,N2 sont les dimensions des matrices opérantes.

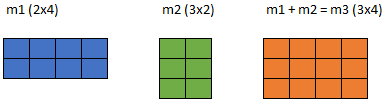
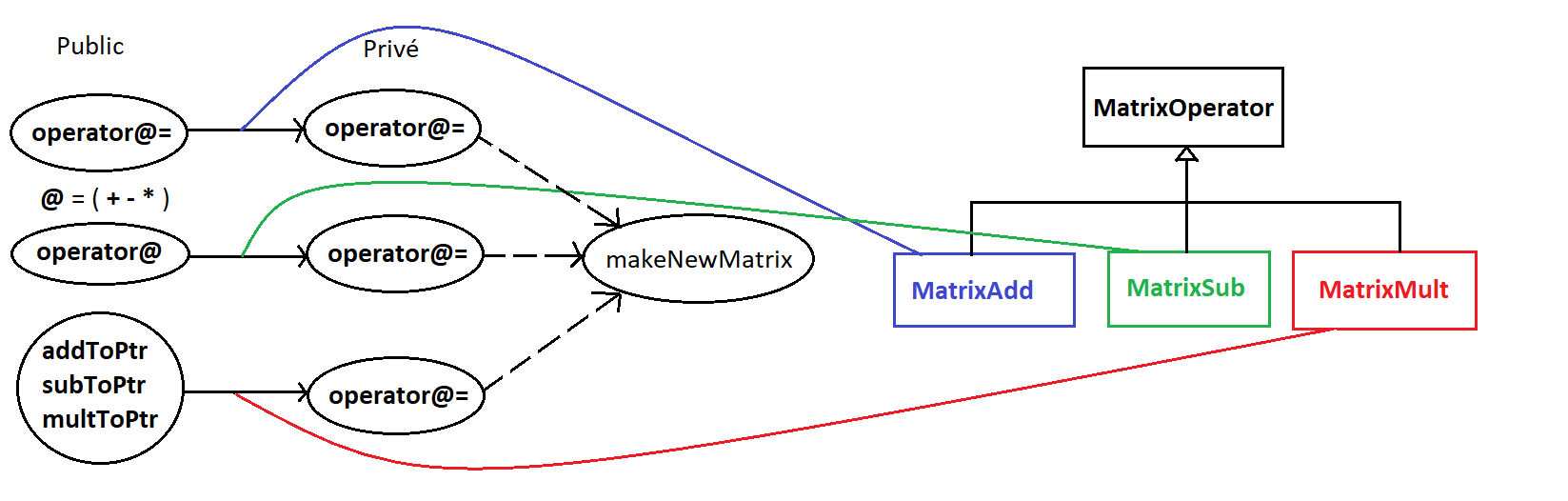


Figure 1 Exemple de redimensionnement sous opération

Les opérations devaient se faire selon trois valeurs de retour différentes, nous avons donc implémenter trois méthodes privées qui demande le second opérande ainsi qu’on objet MatrixOperator :

* Retour de l’objet courant qui est modifié -> opOnSelf
* Retour de la matrice résultante par valeur -> opToVal
* Retour de la matrice résultante par pointeur -> opToPtr

Au-dessus, nous avons interfacé 9 méthodes publiques qui appellent les méthodes privées qui elles travaillent avec makeNewMatrix qui se charchent des opérations matricielles. Le schéma suivant est simplifié mais montre le processus dans les grandes lignes



## Exceptions

makeNewMatrix peut lever deux exceptions :

* Si les matrices à traiter ont un modulo différent, une *invalid\_argument* exception est levée.
* Si la première matrice opérante n’est pas fournie, une runtime\_error estlevée.

# Tests

Les tests ont été effectuées une première fois avec deux matrices non constantes (Mat. N.C) puis avec deux matrices constantes (Mat. C.). Les résultats ont été analysés manuellement.

Parfois des distinctions sont faites pour nuancer les Mat. N.C et Mat. C. Lorsque le test n’est pas pris en compte, la colonne est remplie de « / ».

|  |  |  |  |
| --- | --- | --- | --- |
| Description | Résultat attendu | Mat. N.C. | Mat. C. |
| Constructeur avec valeurs aléatoires. | Les matrices sont créées conformément aux paramètres fournis. | validé | validé |
| Constructeur de copie | La matrice est correctement copiée. | validé | validé |
| operateur<< | La matrice s’affiche correctement. | validé | validé |
| operateur= | L’affectation s’effectue | validé | /////// |
| Ne compile pas. | /////// | validé |
| operateur += | L’addition s’effectue correctement sur la matrice gauche. | validé | ////// |
| Ne compile pas | /////// | validé |
| operateur \*= | La multiplication s’effectue correctement sur la matrice gauche. | validé | ////// |
| Ne compile pas | /////// | validé |
| operateur -= | La soustraction s’effectue correctement sur la matrice gauche. | validé | ////// |
| Ne compile pas | /////// | validé |
| operateur + | L’addition s’effectue correctement sur la matrice gauche. | validé | ////// |
| Ne compile pas | /////// | validé |
| operateur \* | La multiplication s’effectue correctement sur la matrice gauche. | validé | ////// |
| Ne compile pas | /////// | validé |
| operateur - | La soustraction s’effectue correctement sur la matrice gauche. | validé | ////// |
| Ne compile pas | /////// | validé |
| Fonction addToPtr | L’addition s’effectue correctement sur la matrice gauche. | validé | ////// |
| Ne compile pas | /////// | validé |
| operateur multToPtr | La multiplication s’effectue correctement sur la matrice gauche. | validé | ////// |
| Ne compile pas | /////// | validé |
| operateur subToPtr | La soustraction s’effectue correctement sur la matrice gauche. | validé | ////// |
| Ne compile pas | /////// | validé |
| Création d’une matrice avec un modulo nul. | Lève une runtime\_error | validé | validé |
| Opérations entre deux matrices de modulos différents | Lève une invalid\_argument exception | validé | validé |
| Création d’une matrice avec largeur nulle. | Lève une runtime\_error | validé | validé |
| Création d’une matrice avec hauteur nulle. | Lève une runtime\_error | validé | validé |

# Annexes

* Code source en pdf.