17/03/2022

Alexandre Jaquier et Jonathan Friedli

Rapport Laboratoire 1 : Matrix Reloaded

# Introduction

Dans le cadre de ce laboratoire, nous devons implémenter une classe matrix, permettant de représenter des matrices de taille quelconque. Chaque matrice aura des valeurs qui seront modulo n, n étant une valeur passée en paramètre à la création de la matrice. Il sera également possible de d’effectuer des opérations sur ces dernières, telles que l’addition, la soustraction et la multiplication par composante. Chaque opération doit pouvoir être fait de 3 manières différentes. Soit en modifiant la première matrice, soit en retournant par valeur une nouvelle matrice résultat, soit en retournant par pointeur une nouvelle matrice résultat.

# Diagramme UML

# Choix de modélisation et d’implémentation

## Matrix

Les attributs de Matrix sont tous évidemment privés afin de préserver l’encapsulation. Ces derniers sont accessibles via des accesseurs (« getter »). Le modulo, ainsi que les dimension d’une matrice ne pouvant pas être négatif, nous avons choisi de leur mettre un type unsigned. Cependant, vu que les attributs col et row sont utilisés pour faire des comparaisons d’index maximal dans plusieurs boucles, leur type est « size\_t ». L’attribut mod quant à lui est simplement du type « unsigned int ».

Les matrices acceptent comme 0x0 comme taille minimale. Cela représente la matrice vide et toutes les opérations fonctionnent correctement sur la matrice vide. La taille maximale est quant à elle, limitée par la taille maximale qu’un tableau simple peut prendre en c++. Comme mentionné précédemment, la taille de la matrice ne peut pas être négative. Si nous tentons de mettre une valeur négative, il y aura un underflow et cela tentera de créer une matrice d’une taille de quelques milliards, ce qui résultera en un crash du programme.

Le modulo accepte 1 comme valeur minimale. En effet, appliquer l’opération modulo 0 sur un entier ne fait aucun sens. Nous aurions également pu prendre 2 comme valeur minimale, car si la valeur du modulo est égale à 1. Cela veut dire que toutes les valeurs de la matrice seront de 0.

Les opérations ne peuvent être effectuées que sur deux matrices ayant la même valeur de modulo. Si les deux matrices n’ont pas la même taille, cela n’est pas grave, nous remplaçons les valeurs manquantes de la plus petite matrice par 0.

Etant donné que chaque opération doit pouvoir être effectuée de 3 manières différentes, nous leurs avons donné des noms précis. L’opération modifiant la première matrice, se nomme « nomOpérationItself », cela donne « addItslef » pour l’addition. La méthode retournant une matrice par valeur se nomme « addStaticNew » et celle retournant un pointeur sur une matrice résultat, se nomme « addDynamicNew ».

TODO : Pourquoi par valeur et non ref (cf consigne)

## Opération

Afin de rendre l’ajout d’une autre opération (ex : la division) rapide et intuitif, nous avons créé une classe abstraire Opération. Chaque opération doit donc hériter de cette classe abstraite et réimplémenter la méthode apply. Cette dernière permet de calculer le résultat de l’opération entre deux éléments de matrice.

# Tests effectués

Nous avons créé une méthode de test en plus du petit programme demandé dans la consigne.

|  |  |
| --- | --- |
| **Test** | **Résultat attendu** |
| La création d’une matrice de taille 0x0 doit être possible. Si une des composantes de la taille est négative, une exception se lance. | OK |
| La création d’une matrice avec un modulo valant 0 ne doit pas être possible | OK |
| Les opérations entre deux matrices sont possibles uniquement si les deux matrices ont le même modulo. | OK |
| Les opérations entre deux matrices de taille différentes sont possibles si leur modulo est pareil. | OK |
| Les valeurs de la matrice ne sont jamais négatives. En particulier après une soustractions entre deux matrices. | OK |