Rapport - Projet Matrice

COUCHOUD Thomas COLEAU Victor

29 mars 2017

Table des matières

1 Présentation du sujet 2

2 Architecture 3

3 Choix de codage 5

4 Tests effectués 6

5 Conseils d'utilisation 8

Chapitre 1

Présentation du sujet

L'objectif de ce projet est de réaliser une librairie de classes et de fonctions permettant de manipuler des matrices. Celle-ci devra nous permettre de stocker des matrices en mémoire ainsi que d'effectuer des opérations mathématiques simples sur celles-ci.

De plus, il devra être possible d'importer des matrices depuis des fichiers texte externes formatés comme indiqué dans le sujet.

Chapitre 2

Architecture

La classe centrale de notre projet est CMatrix. Celle-ci représente une matrice. Y sont stockées les informations suivantes :

- La hauteur représentant le nombre de lignes de la matrice.
- La largeur représentant le nombre de colonnes de la matrice.
- Un tableau 2D contenant l'ensemble des valeurs de la matrice.

De plus, cette classe contient l'ensemble des méthodes demandées telles que :

- L'opérateur d'addition avec une deuxième matrice en renvoyant une nouvelle.
- L'opérateur de soustraction avec une deuxième matrice en renvoyant une nouvelle.
- L'opérateur de multiplication par un scalaire (à droite et à gauche) renvoyant une nouvelle matrice.
- L'opérateur de division par un scalaire renvoyant une nouvelle matrice.
- L'opérateur de multiplication par une autre matrice en renvoyant une nouvelle.
- L'affichage d'une matrice dans la console.
- Le calcul et la création de la transposée d'une matrice sans changer l'originale.
- L'opérateur d'affection copiant le contenu d'une matrice dans une autre.
- L'opérateur de comparaison.
- Les opérateurs *= et /= avec un scalaire ou une matrice. Ceux-ci modifient la matrice en cours.
- L'opérateur parenthèse prenant deux paramètres et renvoyant la valeur contenue dans la case demandée.

Nous avons ensuite fait hériter à cette classe une autre classe appelée CSquareMatrix représentant elle aussi une matrice mais carrée. Celle-ci contient les mêmes informations que sa classe mère mais propose des méthodes supplémentaires spécifiques aux calculs mathématiques réservés aux matrices carrées :

- Calcul du déterminant.
- Calcul de l'inverse.
- Calcul de la comatrice.
- Calcul des puissances d'une matrice.

Afin de pouvoir lire des matrices externes au programme, nous avons implémentée une classe statique CMatrixParser. Cette dernière lira un fichier texte du format indiqué et renverra au choix, une matrice ou une matrice carrée. Pour réaliser cela, cette dernière s'appuie sur une énumération et une structure. L'énumération eMatrixType permet de faire la transition entre le type de la matrice écrite dans le fichier et le type du language. La structure sMatrixInfo stocke de manière temporaire les informations de la matrice lues dans le fichier.

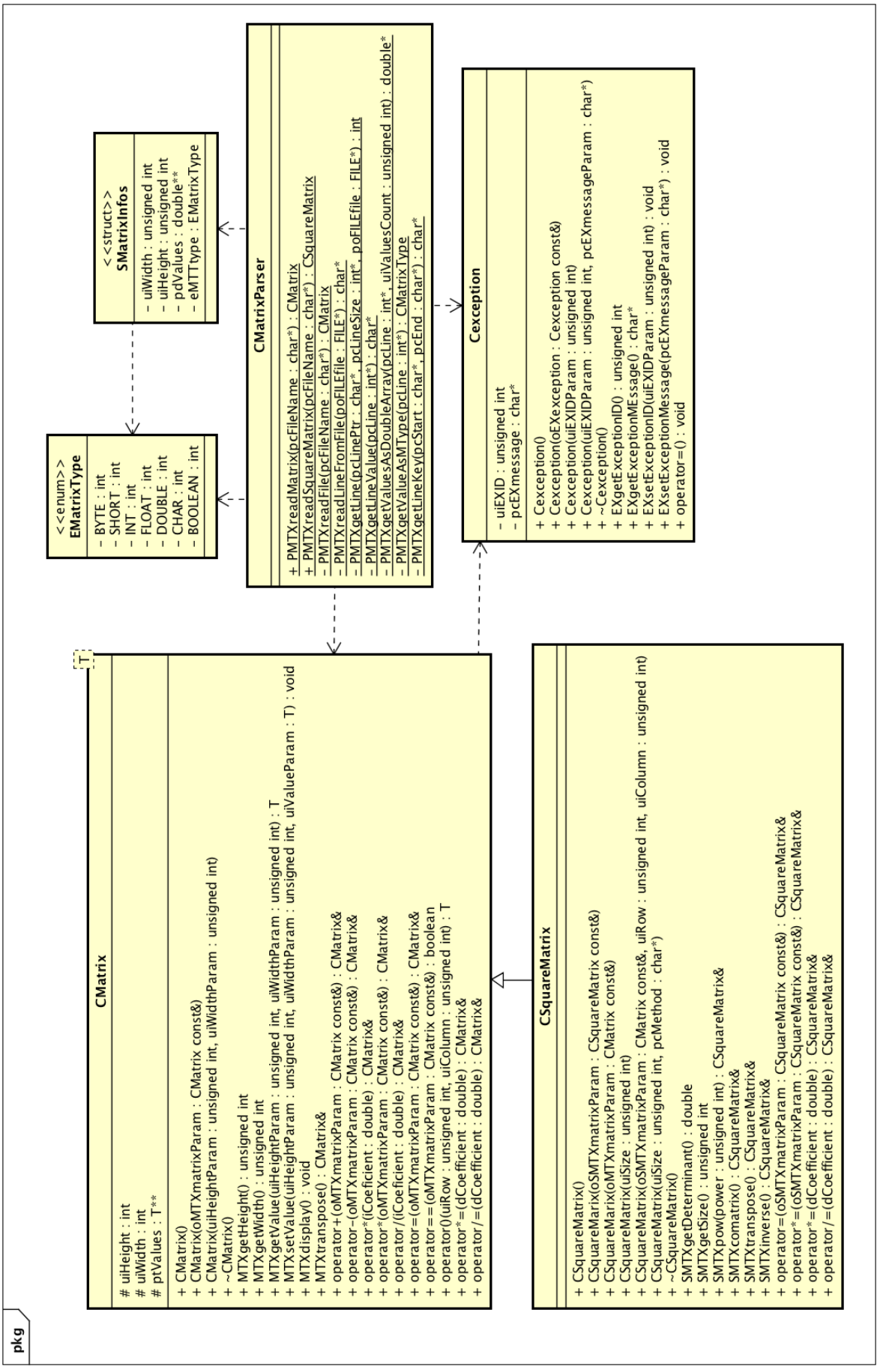

FIGURE 2.1 – Diagramme de classes

Chapitre 3

Choix de codage

Avant tout chose, nous avons prit la décision de créer un fichier nommé "utils.h". Celui-ci contient des macros permettant de compiler sous Visual Studio et un système Unix (utilisation des méthodes propres à chaque compilateur). De plus, il contient un certain nombre de macros utilitaires, notamment MMALLOC et RREALLOC allouant l'espace mémoire demandé tout en vérifiant le bon déroulement.

Le choix de créer une classe CSquareMatrix héritée de CMatrix vient des nombreuses opérations mathématiques propres aux matrices carrées. Nous aurions put implémenter celles-ci directement dans la classe des matrices quelconques mais nous aurions alors du vérifier à chaque appel si la matrice avait le même nombre de lignes et de colonnes et lever une exception en cas d'échec. Grâce à ce choix, nous évitons un grand nombre de vérifications sans pour autant géner les calculs normaux puisque des matrices carrées sont aussi des matrices simples et peuvent donc être multipliées, transposées, etc..

Dans le parser, nous utilisons une structure SMatrixInfo. Celle-ci contient temporairement les informations d'une matrice. Nous en avons l'utilité lors de la création de nos matrices après lecture. En effet, si nous stockions directement les informations lues dans un objet matrice, nous ne pourrions pas faire la différence entre une matrice quelconque et une matrice carrée. Cela est du au fait que l'objet doit être créé avant la lecture mais que nous avons besoin des informations lues pour définir le type d'objet à créer. La structure sert donc à vérifier si une matrice est carrée ou non pour ensuite instancier l'objet approprié.

Chapitre 4

Tests effectués

Afin de pouvoir valider le fonctionnement de notre code, nous avons réalisé de nombreux tests. Certains ont été réalisés manuellement mais un bon nombre d'entre eux ont été écrits sous forme de "test unitaires". Ceux-ci sont présents dans les fichiers CXXXTest.cpp qui vont respectivement tester leur classe XXX.

Nous sommes conscients que les tests sont très minimalistes et ne couvrent pas tout, cependant cela est déjà un bon départ pour vérifier un fonctionnement "normal".

Commençons par CExceptionUnit :

- Tests des différents constructeur impliquant l'ID de l'exception ainsi que des getters et setters de celui-ci.
- Tests des différents constructeur impliquant le message de l'exception ainsi que des getters et setters de celui-ci.

Ceux-ci sont très généralistes mais couvrent relativement bien les différentes utilisations d'un objet de cette classe.

Intéressons-nous maintenant à CMatrixUnit :

- Tests des différents constructeurs ainsi que des getters concernant le nombre de lignes et le nombre de colonnes.
- Tests des getters et setters sur les valeurs de la matrice.
- Tests des différentes opérations :
  - Transposée d'une matrice.
  - Multiplication par un coefficient.
  - Division par un coefficient.
  - Division par le coefficient 0 renvoyant une exception.
  - Test d'égalité de matrices.
  - Test d'affectation de matrices.
  - Test de l'opérateur ()
  - Test de l'opérateur () avec coordonnées invalides renvoyant une exception.
  - Test de la somme de deux matrices.
  - Test de la soustraction de deux matrices.
  - Test du produit de deux matrices.
  - Test du produit de deux matrices incompatibles renvoyant une exception.
  - Test de l'opérateur *= avec un coefficient.
  - Test de l'opérateur *= avec une matrice.
  - Test de l'opérateur *= avec une matrice incompatible renvoyant une exception.
  - Test de l'opérateur /= avec un coefficient.
  - Test de l'opérateur /= avec un coefficient 0 renvoyant une exception.

Ces tests nous semblent assez complets sur les matrices. Cependant chaque test n'est effectué qu'une

seule fois ce qui peut mettre en douter l'efficacité de celui-ci.

Passons ensuite aux tests contenus dans CSquareMatrixUnit :

- Test des différents constructeurs ainsi que des getters concernant la dimension de la matrice.
- Tests des getters et setters sur les valeurs de la matrice.
- Tests des différentes opérations :
  - Test du calcul du déterminant.
  - Test du calcul d'une comatrice.
  - Test du calcul d'une matrice inverse.
  - Test du calcul d'une matrice inverse avec déterminant nul, renvoyant une exception.
  - Calcul de la puissance d'une matrice.

Enfin les derniers tests sont effectués dans CMatrixParserUnit :

- Test de la lecture d'un fichier.
- Test de la lecture d'un ficher avec un type non supporté, renvoyant une exception.
- Test de la lecture d'un fichier en tant que matrice carrée.
- Test de la lecture d'un fichier en tant que matrice carrée avec une matrice non carrée, renvoyant une exception.

Chapitre 5

Conseils d'utilisation

Le programme a été conçu et compilé de sorte que l'exécutable puisse prendre en arguments des fichiers sources de matrice formatés comme défini dans le sujet.

Suite à cela, il va exécuter les instructions suivantes :

- Demander à l'utilisateur un coefficient.
- Calculer et afficher les résultats de la multiplication de chaque matrice par le coefficient précédent.
- Calculer et afficher les résultats de la division de chaque matrice par le coefficient précédent.
- Calculer et afficher le résultat de la somme de toutes les matrices.
- Calculer et afficher le résultat de la somme alternée de toutes les matrices.
- Calculer et afficher le résultat du produit de toutes les matrices.

Si une exception est levée pour n'importe lequel de ces points, le message d'erreur est affiché et le programme s'arrête instantanément.

Pour toute autre utilisation, la classe Main devra être modifier par l'utilisateur, deux choix sont possibles.

Une matrice peut être lue d'un fichier grâce aux fonctions statiques "PMTXreadMatrix" et "PMTXreadSquareMatrix" prenant chacune en paramètre le chemin relatif d'un fichier source de matrice. La première renverra un objet de la classe CMatrix quelles que soient les réelles dimensions de la matrice lue alors que la seconde créera une matrice carrée (ou lèvera une exception si la matrice lue n'est pas carrée). Les deux lèveront des exceptions si un problème survient lors de la lecture.

D'une seconde manière, une matrice peut être créée directement depuis son constructeur mais celle-ci sera remplie de 0. Nous pouvons par la suite utiliser le setter adéquat pour modifier chaque valeur de la matrice a sa guise. A savoir qu'un constructeur identité est disponible pour les matrices carrées.

Pour toute information supplémentaire sur les méthodes, se référer aux cartouches d'entête présents dans les fichiers .h.