|  |  |  |  |
| --- | --- | --- | --- |
|  | 01/06/2019 |  | |
| |  |  | | --- | --- | |  |  | |  |  | |  |  | |  |  | | |  | |
| Rapport de projet  *Librairie de classes et fonctions permettant de manipuler des matrices d’éléments* | | | |
|  |  | | |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |
|  |  | | Yoann DUPAS - Alexandre TURPIN |

Table des matières

[I. Modélisation UML 3](#_Toc10377850)

[A. Diagramme de classe 3](#_Toc10377851)

[B. CMatrice<T> 3](#_Toc10377852)

[1. Attributs 3](#_Toc10377853)

[2. Méthodes 4](#_Toc10377854)

[C. CMatriceOperationComplexe 4](#_Toc10377855)

[1. Méthodes 4](#_Toc10377856)

[D. CParseur 4](#_Toc10377857)

[E. CException 5](#_Toc10377858)

[1. Attributs 5](#_Toc10377859)

[2. Méthodes 5](#_Toc10377860)

[II. Evolution du projet 6](#_Toc10377861)

[A. Méthodes modifiées 6](#_Toc10377862)

[1. Classe CMatrice 6](#_Toc10377863)

[B. Méthodes ajoutées 6](#_Toc10377864)

[1. Classe CMatrice 6](#_Toc10377865)

[C. Méthodes supprimées 6](#_Toc10377866)

[1. Classe CMatrice 6](#_Toc10377867)

[2. Classe CMatriceOperationComplexe 6](#_Toc10377868)

[D. Organisation du projet 7](#_Toc10377869)

[E. Les tests 7](#_Toc10377870)

[III. Question Individuelle 8](#_Toc10377871)

[A. Méthodes 8](#_Toc10377872)

[B. Evolution du projet 8](#_Toc10377873)

[IV. Notices d’utilisation de la bibliothèque 9](#_Toc10377874)

[A. Créer une matrice 9](#_Toc10377875)

[B. Interaction avec les matrices 9](#_Toc10377876)

[1. Lecture 9](#_Toc10377877)

[2. Ecriture 9](#_Toc10377878)

[3. Opération 9](#_Toc10377879)

[C. Extraction de matrice 10](#_Toc10377880)

[1. Extraction à partir d’un point d’origine 10](#_Toc10377881)

[2. Extraction à partir d’un point central 10](#_Toc10377882)

[3. Extraction de toutes les sous matrices 10](#_Toc10377883)

[D. Utiliser la classe CParser 10](#_Toc10377884)

[1. Initialiser le parser 10](#_Toc10377885)

[2. Se balader dans le fichier 10](#_Toc10377886)

[3. Lire le fichier texte 10](#_Toc10377887)

Rapport de projet

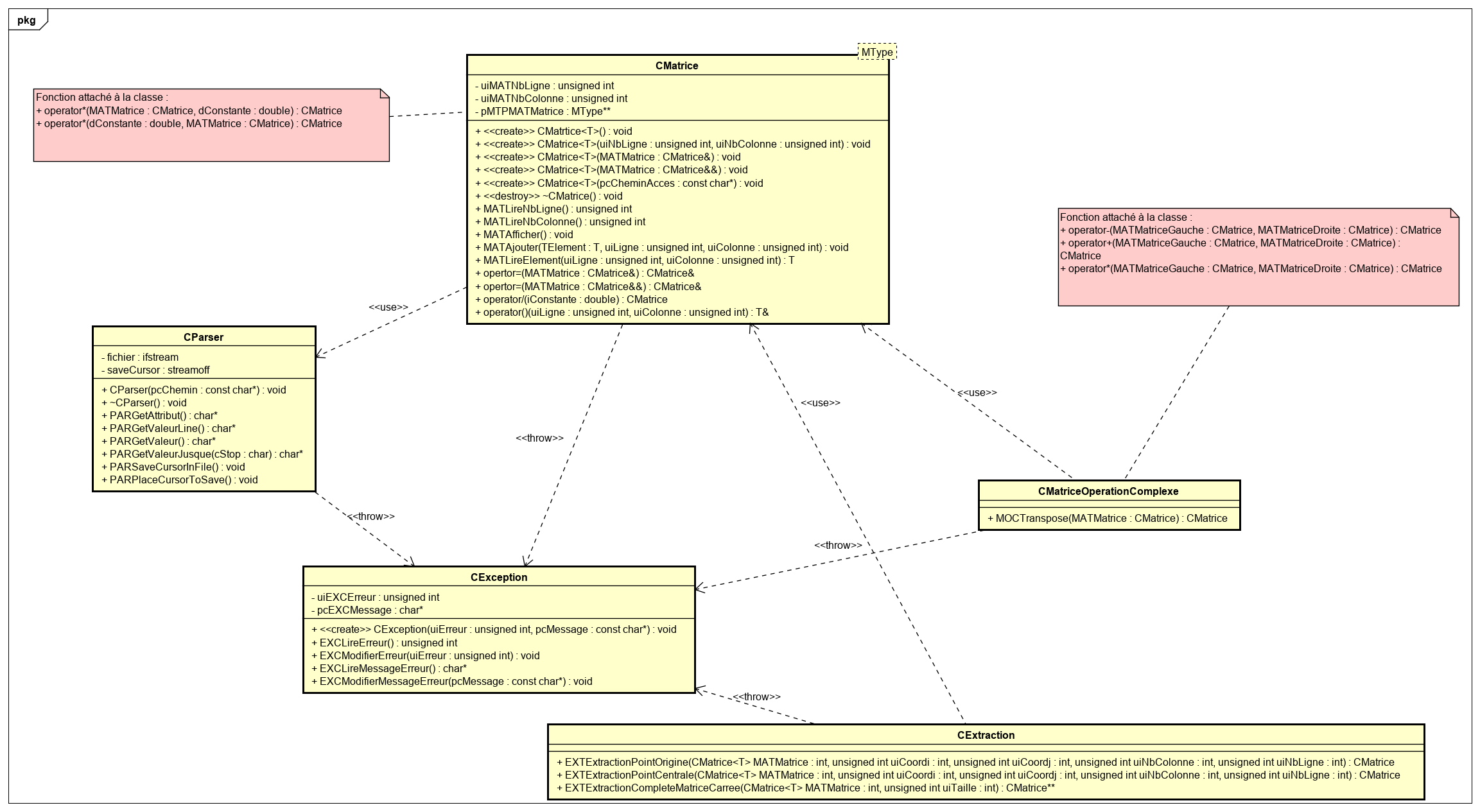
Librairie de classes et fonctions permettant de manipuler des matrices d’éléments

L’objectif de ce projet est de réaliser une libraire de classes et fonctions permettant de manipuler des matrices d’éléments.

# Modélisation UML

## Diagramme de classe

Voici le diagramme de classe final de notre projet :



Nous avons décidé de structurer le programme en cinq classes :

* CMatrice<T> : C’est la classe centrale du projet, elle permet de représenter les matrices dans notre programme.
* CMatriceOperationComplexe : C’est une classe qui traite les opérations complexes avec des matrices.
* CParseur : Cette classe est un analyseur syntaxique.
* CException : Cette classe permet de gérer les exceptions du projet.
* CExtraction : Cette classe traite les opérations d’extractions de sous-matrices.

## CMatrice<T>

### Attributs

* uiMATNbLigne (unsigned int)
* uiMATNbColonne (unsigned int)
  + Ces attributs permettent de définir la taille de la matrice
* pTMATMatrice (T\*\*)
  + C’est le tableau 2D où les éléments de la matrice seront stockés

### Méthodes

CMatrice comporte les méthodes permettant de manipuler des objets représentant les matrices. Un des attributs de cette classe est dynamique, il est donc essentiel qu’elle comporte un constructeur de recopie, ainsi qu’un destructeur afin d’éviter tout problème de mémoires. Nous avons ensuite décider pour des questions de conforts d’ajouter une méthode prenant en paramètre la taille du tableau souhaité et initialise chaque élément par défaut, ainsi qu’un chemin vers un fichier texte qui permet de créer une matrice à l’aide de la classe CParseur. En ce qui concerne les constructeurs de recopie, nous avons voulu qu’il soit possible de faire référence à des valeurs à gauche qu’à droite (C+11). Elle possède aussi d’autre méthodes élémentaires comme afficher une matrice, accéder par référence à un élément du tableau ou par lecture ainsi que des surcharges d’opérateur afin d’effectuer des opérations élémentaires avec des nombres, c’est-à-dire la multiplication et la division.

Les exceptions possibles :

* « Incohérence des arguments »
* « Lecture en dehors des dimensions de la matrice »
* « Division par 0 »

## CMatriceOperationComplexe

### Méthodes

CMatriceOperationComplexe contient les opérations mathématiques qualifié de complexe par le cahier des charges, c’est-à-dire la transposé et la multiplication, l’addition ou la soustraction de deux matrices, avec ou non des types différents.

## CParseur

La classe CParser est un analyseur syntaxique.

Un objet CParser contient deux attributs :

* Fichier : un objet ifstream correspondant au pointeur vers le fichier
* saveCursor : un objet streamoff qui permet d’enregistrer l’emplacement du curseur

La classe possède un constructeur de confort :

* CParser(const char\* pcChemin) : il permet d’ouvrir le fichier à partir de l’adresse passée en paramètre

Un destructeur qui permet de fermer proprement le fichier et une liste de méthode qui permet de récupérer les valeurs du fichier :

* PARGetAttribut() : permet de retourner l’attribut à l’emplacement du curseur
* PARGetValeurLine() : permet de retourner la chaine de caractère du curseur où l’on se trouve jusqu’à la fin de la ligne
* PARGetValeur() : permet de retourner la chaine de caractère du curseur où l’on se trouve jusqu’à un espace ou la fin de la ligne
* PARGetValeurJusque(char cStop) : permet de retourner la chaine de caractère du curseur où l’on se trouve jusqu’au caractère cStop passé en paramètre

Ainsi que des méthodes permettant de manipuler les curseurs :

* PARSaveCursorInFile() : permet d’enregistrer la position du curseur où l’on se trouve dans le fichier
* PARPlaceCursorToSave() : Cette méthode permet de remettre le curseur à la position sauvegardé au préalable

Les exceptions possibles :

* "Erreur dans l’ouverture du fichier"
* "Erreur de format du fichier"
* "Erreur de type"
* "Incohérence des arguments"

## CException

### Attributs

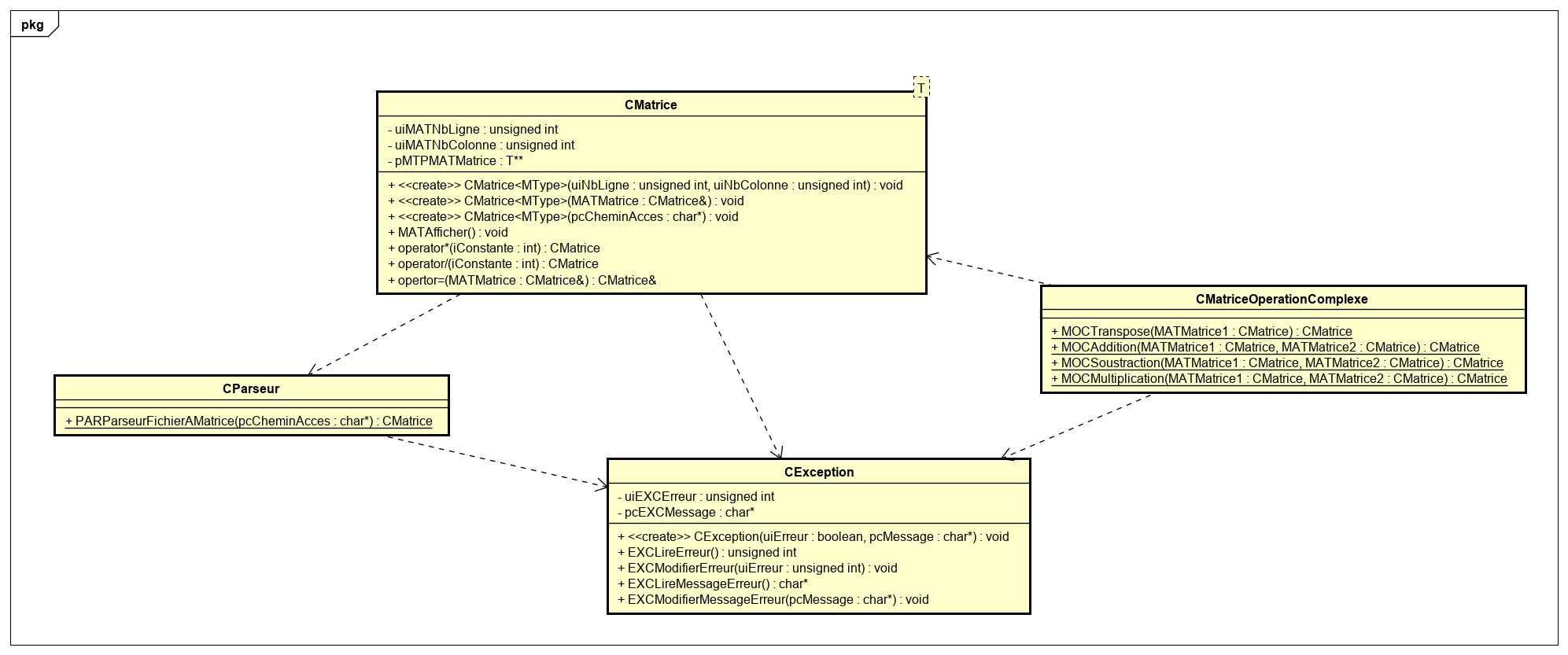
* CException
  + pcEXCMessage (const char \*)
    - C’est un message d’erreur qui permet de nous indiquer plus d’information sur les erreurs
  + uiEXCErreur (unsigned int)
    - C’est le numéro d’erreur défini par fichier pour chaque erreur potentielle.

### Méthodes

Dans cette classe, nous avons mis un constructeur de confort prenant en arguments un unsigned int pour le numéro de l’erreur et un char\* pour le message d’erreur. Ce constructeur nous permet de créer directement un objet CException tout en initialisant ses arguments. Le reste des méthodes de la classe sont des accesseurs. Ceux-ci permettent de lire et de modifier les attributs de la classe.

# Evolution du projet

Durant notre projet, nous avons passées les deux premières séances à déterminer le modèle UML suivant :



## Méthodes modifiées

### Classe CMatrice

* Constructeur de recopie et la surcharge de l’opérateur « = » avec en paramètre une Matrice par référencement par la droite. Lors de nos tests nous avons rencontrer une erreur qui nous empêchait de faire des opérations du style suivant : « matrice1 = matrice2 \* matrice3 », les opérations renvoyant une matrice (non référencé). Avec l’aide de google, nous avons trouvé que pour permettre ce genre d’opération, il fallait indiquer que la matrice en paramètre une référence par la droite avec le symbole (&&).
* La surcharge de l’opérateur \* qui se trouve en dehors de l’interface de la classe (mais dans le .h) afin de permette de qualifier les méthodes en tant que fonction et ainsi permettre la commutativité des opérations.

## Méthodes ajoutées

### Classe CMatrice

* Les accesseurs en lecture du nombre de ligne et de colonne de la matrice car on avait oublié les accesseurs.
* La méthode MATAjouter qui permet d’ajouter un élément. On avait remarqué quand on voulait ajouter des éléments, qu’on ne pouvait pas le faire. Dans la même occasion, on a surchargé les opérateurs () afin que ça soit plus rapide, plus simple et plus naturel pour modifier un élément de la matrice.

## Méthodes supprimées

### Classe CMatrice

* Les surcharges de l’opérateur \* ont été enlevé de la classe et mis en tant que fonction afin de garantir la commutativité qui nous avait posé des problèmes lors des phases de tests de la classe, après relecture du cours de c++.

### Classe CMatriceOperationComplexe

* MOCMultiplication, MOCSoustraction et MOCAddition ont été supprimé pour être remplacer par des surcharges d’opérateur beaucoup plus naturel, et pour des raisons techniques du C++ placé en dehors de l’interface de la classe.

## Organisation du projet

Afin de mieux s’organiser dans le projet, nous avons mis en place un git, qui a été fastidieux à mettre en place au début. Nous avons utilisé l’IDE Visual Studio Enterprise 2017 et mis en place un Trello afin de suivre l’activité du projet (mais pas très mis à jour car on était souvent en présent ensemble lors des séances de travail).

## Les tests

Comme le projet était léger, nous avons uniquement tester nos méthodes directement dans le main.

# Question Individuelle

L’objectif de la question individuelle est de fournir une libraire pouvant extraire, d’une matrice donnée, un ensemble de sous-matrices. Pour répondre à cette question, nous avons décider de simplement créer une nouvelle classe CExtraction comportant trois méthodes. Chaque méthode répond aux trois demandes de la question :

## Méthodes

* EXTExtractionPointOrigine(CMatrice<T> MATMatrice,unsigned int uiCoordi, unsigned int uiCoordj, unsigned int uiNbColonne, unsigned int uiNbLigne) : La méthode permet d’extraire une sous matrice dont son indice (0,0) correspond à l’indice (uiCoordi,uiCoordj) de la matrice d’origine. uiNbColonne et uiNbLigne correspondent aux tailles de la matrice voulu, respectivement, le nombre de colonne et le nombre de ligne.
* EXTExtractionPointCentrale(CMatrice<T> MATMatrice, unsigned int uiCoordi, unsigned int uiCoordj, unsigned int uiNbColonne, unsigned int uiNbLigne) : La méthode permet d’extraire une sous matrice dont l’indice (uiCoordi,uiCoordj) correspond à l’élément centrale de la sous-matrice. . uiNbColonne et uiNbLigne correspondent aux tailles de la matrice voulu, respectivement, le nombre de colonne et le nombre de ligne à partir de l’élément centrale. Ce qui veut dire que sa taille finale correspond au nombre de colonne par deux plus un fois le nombre de ligne par deux plus un.
* EXTExtractionCompleteMatriceCarree(CMatrice<T> MATMatrice, unsigned int uiTaille) : cette méthode renvoi un tableau comprenant toute les sous matrice carrées de taille uiTaille que l’on peut extraire à partir d’une matrice

## Evolution du projet

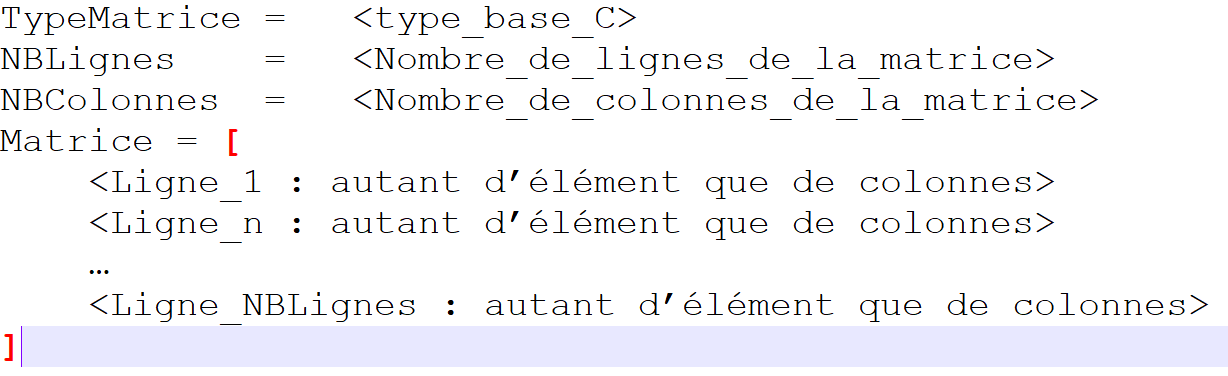
Les méthodes ont été réalisé les unes après les autres. Aucune difficultés majeurs n’a été reportés.

# Notices d’utilisation de la bibliothèque

## Créer une matrice

Vous avez deux choix pour créer une matrice :

* En indiquant le nombre de ligne et le nombre de colonne
* En indiquant un chemin vers un fichier texte avec la syntaxe suivante :



Vous pouvez aussi créer une matrice à l’aide de matrice existante ou le résultat de d’autre opération renvoyant une matrice

IMPORTANT : Vous devez toujours préciser entre <> le type que vous souhaité pour votre matrice.

## Interaction avec les matrices

### Lecture

* MATLireNbLigne
* MATLireNbColonne
  + Permet de lire la taille du tableau
* MATLireElement
  + Permet de lire un élément du tableau

### Ecriture

* MATAjouterElement (Numéro de la ligne, Numéro de la colonne)
  + Permet de lire et modifier l’élément de la ligne
    - Vous pouvez aussi directement modifier l’élément avec les parenthèses (Exemple : Matrice(1,3) = Elément en question)

### Opération

* Multiplication par une constance (symbole \*)
* Division à droite par une constance (symbole /)
* Multiplication par une matrice (symbole \*)
* Addition par une matrice (symbole +)
* Soustraction par une matrice (symbole -)
* MOCTranspose(Matrice)
  + Retourne une nouvelle matrice qui est la transposé de la matrice en paramètre

Chaque opération renvoie une matrice.

## Extraction de matrice

### Extraction à partir d’un point d’origine

* EXTExtractionPointOrigine
  + Permet d’extraire une sous matrice à partir d’un point d’origine et d’un nombre de ligne et de colonne.

### Extraction à partir d’un point central

* EXTExtractionPointCentrale
  + Permet d’extraire une sous matrice à partir d’un point central. La matrice renvoyé sera la sous-matrice allant du point donné moins le nombre de ligne et colonne donnés au point donné plus le nombre de ligne et colonne donnés.

### Extraction de toutes les sous matrices

* EXTExtractionCompleteMatriceCarree
  + Permet d’extraire toutes les sous-matrices carrées de la taille passée en paramètre. Les sous matrices sont renvoyées sous forme de tableau.

## Utiliser la classe CParser

### Initialiser le parser

* CParser(const char\* pcChemin)
  + Ce constructeur va initialiser le parser en ouvrant le fichier placé en paramètre. Si la méthode n’arrive pas à ouvrir le fichier alors une exception est levée.

### Se balader dans le fichier

* PARSaveCursorInFile()
  + Cette méthode permet d’enregistrer la position du curseur où l’on se trouve dans le fichier. Cette position est sauvegardée dans l’attribut saveCursor de l’objet CParser.
* PARPlaceCursorToSave()
  + Cette méthode permet de remettre le curseur à la position sauvegardé au préalable.

### Lire le fichier texte

* PARGetAttribut()
  + Cette méthode renvoi la chaine de caractère se trouvant à gauche du ‘=’ dans la ligne où l’on se trouve.
* PARGetValeurLine()
  + Cette méthode renvoi la chaine de caractère du curseur où l’on se trouve jusqu’à la fin de la ligne.
* PARGetValeur()
  + Cette méthode renvoi la chaine de caractère du curseur où l’on se trouve jusqu’à un espace ou la fin de la ligne.
* PARGetValeurJusque(char cStop)
  + Cette méthode renvoi la chaine de caractère du curseur où l’on se trouve jusqu’au caractère cStop passé en paramètre.