GAYET Lilian

FILLEUL Lucas

**Rapport Projet C++**

**Partie 1 - Les matrices**

# Sommaire

[**Sommaire**](#_61xa3i75sld8) **1**

[**Modélisation**](#_ghq3b2cquqtr) **2**

[Diagramme de classes](#_ms4but72800x) 2

[Classe CMatrice](#_up1lqg4ld6r5) 3

[Classe CException](#_swhrlcx7z1x) 3

[Classe COperations](#_yk4kiverjloa) 4

[**Utilisation de la librairie**](#_4h34rlnhodp2) **5**

[CMatrice](#_kjd8knhw3df5) 5

[COperation](#_15qtxpphpoy0) 6

[CParseur](#_wqb0ean3xsaj) 6

[**Annexes**](#_6wqtp5c5oxqq) **7**

# 

# Modélisation

## Diagramme de classes

Dans un premier temps, suite à la lecture du sujet, nous avons listé les classes qui nous semblaient importantes à la réalisation de ce projet ainsi que les attributs et méthodes de chacune. Après discussion et modification, nous avons finalisé le diagramme de classe et avons obtenus ceci : (En annexe dans un format plus grand et dans le dossier de rendus).

### 

### 

### Classe CMatrice

La classe **CMatrice** nous permettra de créer, initialiser et gérer un objet **CMatrice** où les composantes de celle-ci seront typées selon le choix de l’utilisateur. Par conséquent, cette classe sera un patron de classe. Ses attributs sont deux unsigned int renseignant le nombre de lignes et de colonnes de la matrice ainsi qu’un tableau de tableau du type donné par l’utilisateur.

Nous avons décidé d'implémenter 5 constructeurs au total. Le premier, le constructeur par défaut que l’utilisateur ne pourra pas utiliser. Le second, le constructeur de recopie. Enfin les trois derniers sont des constructeurs de confort. Par rapport au premier constructeur, ayant décidé d’implémenter la classe **CMatrice** de manière à ne pas accepter la re-dimensions d’une matrice, il nous fallait bloquer son utilisation. Initialisée à une valeur par défaut et ne pouvant être redimensionnable, la matrice serait inutile à l’utilisation et pourrait être bloquante. C’est pourquoi nous avons décidé de bloquer l’appel au constructeur par défaut. Enfin nous avons implémenté un destructeur spécifiant en précondition que l’attribut contenant les données de la matrice ait été supprimé auparavant.

Afin d’améliorer l’initialisation et l’utilisation des objets **CMatrice**, nous avons implémenté plusieurs méthodes :

* MATLireNombreLigne : Qui renvoie le nombre de ligne de la matrice ;
* MATLireNombreColonne : Qui renvoie le nombre de colonne de la matrice ;
* MATLireElement: Qui renvoie l’élément à la position (x,y) passée en paramètre ;
* MATAffecterDonnees: Qui affecte aux données un tableau passée en paramètre ;
* MATAffecterElement: Qui affecte à la position (x,y) passée en paramètre l’élément lui aussi passé en paramètre ;
* MATAfficherMatrice: Qui affiche dans la console la matrice ;
* MATSupprimerMatrice: Qui supprime les données de la matrice.

### Classe CException

La classe **CException** permettra de lever et gérer des exceptions afin de prévenir des cas d’utilisations prévisibles mais anormaux.

Elle comporte les méthodes suivantes :

* EXCLireCodeErreur
* EXCLireMessageErreur
* EXCModifierErreur

EXCLireCodeErreur permet, comme son nom l’indique, de renvoyer le code d’erreur levé.

EXCLireMessageErreur, quand à elle, “traduit” le code d’erreur en un message d’erreur afin d’expliquer à l’utilisateur la raison de la levée d’exception.

EXCModifierErreur est un accesseur d’écriture du code d’erreur mais il n’est pas utilisé au sein de notre code. Cependant, il pourrait s’avérer utile lors d’une nouvelle implémentation du code.

La classe **CException** étant très généraliste, il n’y aura qu’à changer EXCLireMessageErreur dans le .cpp pour pouvoir réutiliser cette classe dans d’autres projets.

Les exceptions gérer dans ce projet sont :

Dans l’utilisation de la classe **COperation** :

***Taile\_Diff (code 100)*** : "Addition de matrice de tailles différentes !"

***Mul\_Taille\_Diff (code 101)*** : “Multiplication de matrice de tailles invalides !"

***Division\_Par\_Zero (code 102)*** : “Division de matrice par zéro !"

***Parametre\_Invalide (code 103)*** : “Paramètres invalides !"

Dans l’utilisation de la classe **CParseur**:

***Fichier\_Introuvable (code 200)*** : "Fichier introuvable !"

***Format\_Fichier\_Errone (code 201)*** : “Format de fichier erroné !"

***Type\_Matrice\_Non\_Gerer (code 202)*** : “Type de matrice non géré (Type != double) !"

***Nombre\_Ligne\_Incompatible (code 203)*** : “Nombre de lignes incompatible !"

***Nombre\_Colonnes\_Incompatible (code 204)*** : "Nombre de colonnes incompatible !"

***Donnes\_Matrice\_Errone (code 205)*** : “Données de matrice erronées!"

***Format\_Fichier\_Errone\_Type (code 206)*** : “Format de fichier erroné (type)!"

***Format\_Fichier\_Errone\_Lignes (code 207)*** : “Format de fichier erroné (lignes)!"

***Format\_Fichier\_Errone\_Colonnes (code 208)*** : “Format de fichier erroné (colonnes)!"

***Format\_Fichier\_Errone\_Donnees (code 209)*** : “Format de fichier erroné (données)!"

***Format\_Fichier\_Errone\_Donnees\_NBLignes (code 210)*** : “Format de fichier erroné (nbligne ne correspond pas)!"

***Format\_Fichier\_Errone\_Donnees\_NBColonnes (code 211)*** : “Format de fichier erroné (nbcolonne ne correspond pas)!"

Dans l’utilisation de la classe **CMatrice** :

***Parametre\_Matrice\_Invalide(code 300)*** : "Matrice passée en paramètre invalide !"

***Position\_Invalide(code 301)*** : “Position recherchée invalide !"

***Matrice\_Vide(code 302)*** : “Matrice vide !"

***Matrice\_Deja\_Allouee(code 303)*** : “ Matrice déjà allouée !"

***Parametre\_Tableau\_Invalide(code 304)*** : "Tableau passé en paramètre invalide !"

***Parametre\_Element\_Invalide(code 305)*** : “Element passe en paramètre invalide !"

***Matrice\_Nombre\_Ligne\_Incompatible(code 306)*** : “Nombre de lignes incompatible !"

***Matrice\_Nombre\_Colonnes\_Incompatible(code 307)***:“Nombre de colonnes incompatible!"

### Classe COperations

La classe **COperations** permettra d’effectuer des opérations plus complexe sur des matrices (Addition / Multiplication / Soustraction / Transposée / etc…). Cette classe statique contient les patrons des méthodes de division et de multiplication d’une matrice avec une valeur (entier, double, etc …) définie par l’utilisateur.

Les opérations implémentées sont :

* OPEMultiplication<Type> : Qui multiplie la matrice par un nombre ;
* OPEDivision<Type> : Qui multiplie la matrice par un nombre ;
* OPETransposee: Qui calcul la transposée de la matrice ;
* OPEAddition: Qui additionne deux matrice ;
* OPESoustraction: Qui soustrait une matrice par une autre ;
* OPEMultiplication: Qui multiplie deux matrice ;

# Utilisation de la librairie

## CMatrice

* Exemple de déclaration et instanciation d’un objet **CMatrice** :

*CMatrice<Type> matrice = CMatrice<Type>(NbLignes, NbColonnes);*

*CMatrice<Type> matrice = CMatrice<Type>(NbLignes, NbColonnes, Tableau);*

*CMatrice<Type> matrice = CMatrice<Type>(AutreMatrice);*

* Exemple de déclaration et instanciation d’un pointeur d’un objet **CMatrice** :

*CMatrice<Type> \* pmatrice = new CMatrice<Type>(NbLignes, NbColonnes);*

*CMatrice<Type> \* pmatrice = new CMatrice<Type>(NbLignes, NbColonnes, Tableau);*

*CMatrice<Type> \* pmatrice = new CMatrice<Type>(AutreMatrice);*

* Exemple d’utilisation des méthodes sur un objet ou un pointeur d’objet **CMatrice** :

1) Récupération du nombre de lignes d’une matrice :

*matrice.MATLireNombreLigne();*

*pmatrice->MATLireNombreLigne();*

2) Récupération du nombre de colonnes d’une matrice :

*matrice.MATLireNombreColonne();*

*pmatrice->MATLireNombreColonne();*

3) Affecter un élément à une position donnée d’une matrice :

*matrice.MATAffecterElement(positionLigne, positionColonne, Element);*

*pmatrice->MATAffecterElement(positionLigne, positionColonne, Element);*

4) Affecter à une matrice ses éléments :

*matrice.MATAffecterDonnees(Tableau);*

*pmatrice->MATAffecterDonnees(Tableau);*

5) Afficher la matrice :

*matrice.MATAfficherMatrice();*

*pmatrice->MATAfficherMatrice();*

### 

## COperation

* Exemple d’utilisation des méthodes de la classe **COperation** :

1) Addition de deux matrices :

*CMatrice<Type> matrice = COperation::OPEAddition(matrice1, matrice2);*

*CMatrice<Type> \* pmatrice = &COperation::OPEAddition(matrice1, matrice2);*

2) Soustraction de deux matrices :

*CMatrice<Type> matrice = COperation::OPESoustraction(matrice1, matrice2);*

*CMatrice<Type> \* pmatrice = &COperation::OPESoustraction(matrice1, matrice2);*

3) Multiplication de deux matrices :

*CMatrice<Type> matrice = COperation::OPEMultiplication(matrice1, matrice2);*

*CMatrice<Type> \* pmatrice = &COperation::OPEMultiplication(matrice1, matrice2);*

4) Multiplication d’une matrice par une valeur numérique :

*CMatrice<Type> matrice = COperation::OPEMultiplication(matrice1, ValeurNum);*

*CMatrice<Type> \* pmatrice = &COperation::OPEMultiplication(matrice1, ValeurNum);*

*CMatrice<Type> matrice = COperation::OPEMultiplication(ValeurNum, matrice1);*

*CMatrice<Type> \* pmatrice = &COperation::OPEMultiplication(ValeurNum, matrice1);*

5) Division d’une matrice par une valeur numérique :

*CMatrice<Type> matrice = COperation::OPEDivision(matrice1, ValeurNumerique);*

*CMatrice<Type> \* pmatrice = &COperation::OPEDivision(matrice1, ValeurNumerique);*

6) Calcul de la transposée d’un matrice :

*CMatrice<Type> matrice = COperation::OPETransposee(matrice1);*

*CMatrice<Type> \* pmatrice = &COperation::OPETransposee(matrice1);*

## CParseur

* Exemple d’utilisation de la méthode de parsing d’une matrice :

*CMatrice<Type> matrice = CParseur::PARParseurMatrice(CheminFichier);*

*CMatrice<Type> \* pmatrice = &CParseur::PARParseurMatrice(CheminFichier);*

# Annexes

