# Développement objet C++
# Projet : Matrices

Etudiants :
ALHABAJ Mahmod
BELDA Tom

Professeur référent :
T'KINDT Vincent

Année universitaire :
2019-2020

# Sommaire

# A. Introduction

## 1. sujet

Ce projet a pour objectif de réaliser une librairie de classes et fonctions permettant de manipuler des matrices d'éléments. Ce projet a été réalisé dans le langage C++.

En plus de cette consigne de base, le client V.T'kindt, nous a donné une liste d'instructions supplémentaires :

- Une matrice doit pouvoir être de n'importe quel type. On doit autant pouvoir créer une matrice de double qu'une matrice d'objets d'une classe quelconque.

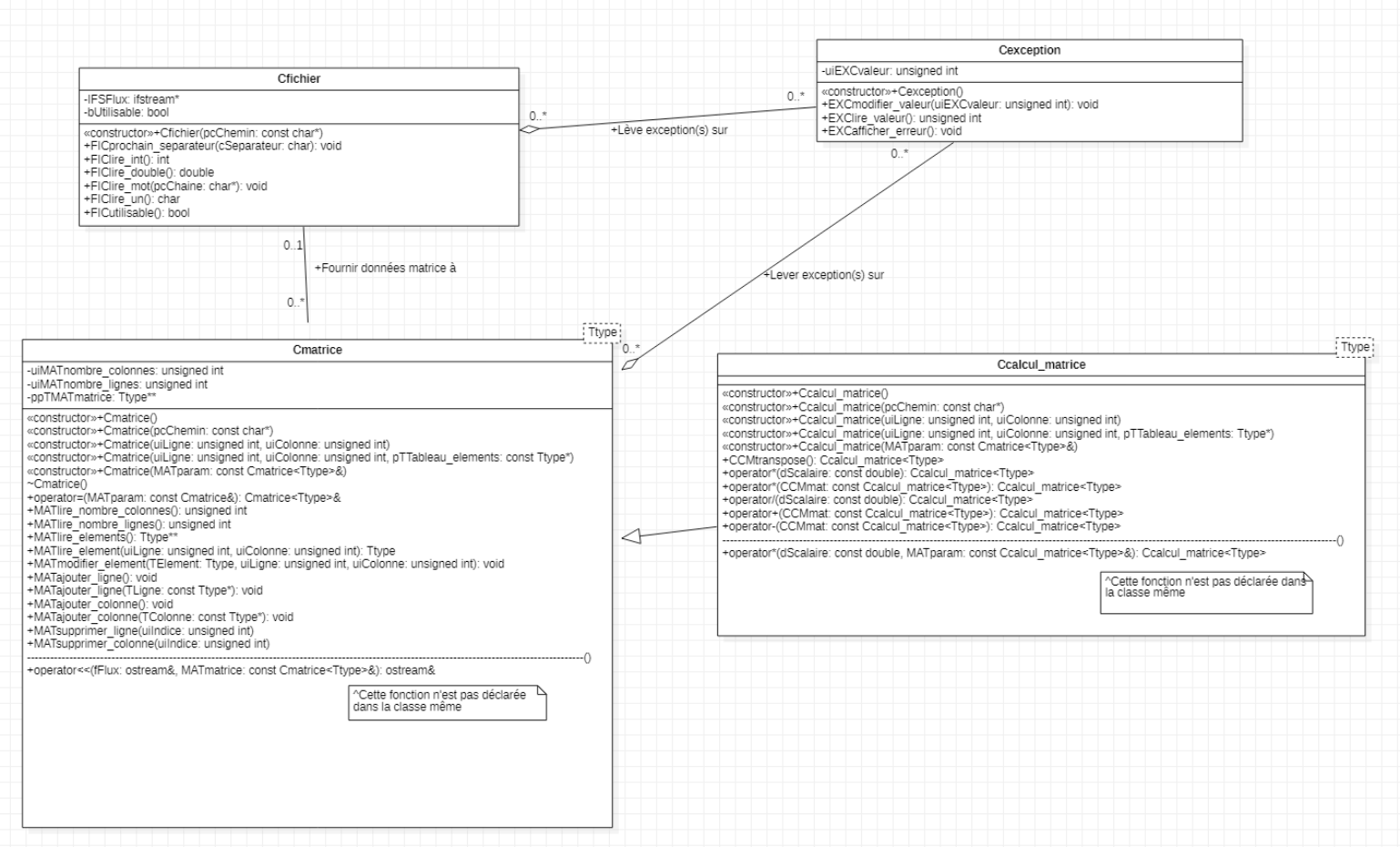
- Une matrice doit permettre des opérations élémentaires comme la multiplication ou la division par un scalaire ou encore l'affichage.

- Une matrice doit aussi permettre des opérations plus compliquées comme l'addition, la soustraction, la multiplication entre matrice ou encore le calcul de la transposée.

- Une matrice devra aussi pouvoir être importée d'un fichier.

- La librairie en elle-même doit gérer les exceptions.

## 2. Spécifications



Grâce à ce cahier des charges, nous avons pu deduire la structure générale de la librairie. Celle ci se séparera en 4 classes : Cmatrice, Ccalcul_matrice, Cfichier et Cexception(fournis par V. T'kindt).

## B. Classes:

### 1. Cmatrice

La classe Cmatrice represente purement la structure de la matrice. Elle ne permet pas de calcul, elle gère uniquement la création de la matrice, l'ajoût et la suppresion d'éléments. La structure de la matrice est simple : elle comporte 3 variables « privates » : le nombre de colonnes, le nombre de lignes et un tableau de tableau d'éléments. Pour initialiser ces différentes variables, nous avons ajouté plusieurs constructeurs :

- Le constructeur par default initialise une matrice de dimensions (1, 1) vide. En effet, nous avons fait ce choix car pour nous la création d'une matrice de dimensions (0, 0) n'avait aucun sens. De plus, cela posait quelques difficultés techniques dans la suite du projet.
- Le constructeur de recopie, nécessaire dans le cas d'une classe avec une variable de type pointeur.
- Le constructeur avec fichier demandé par le client.
- Les constructeurs de confort : un constructeur avec dimensions et un avec dimensions et contenu pour remplir directement la matrice.

Une fois créée, la matrice n'est pas figée et la classe Cmatrice doit donc permettre les modifications. Pour cela nous avons rajouté plusieurs fonctions :

- fonctions de lecture des variables de la classe.
- fonctions de lecture et de modification des éléments de la matrice.
- fonctions d'ajout et de suppression de colonnes ou lignes de la matrice.

Enfin, en dehors de la déclaration de la classe, nous avons surchargé l'operateur « << » pour permettre l'affichage.

### 2. Ccalcul_matrice

Ccalcul_matrice hérite da la classe Cmatrice.

Cette nouvelle classe a pour objectif de rajouter tout l'aspect calcul non inplementé dans la classe Cmatrice.

Toutes les fonctions de la classe Cmatrice sont utilisables dans Ccalcul_matrice. A cela s'ajoute la surcharge de plusieurs opérateurs de calcul.

Nous avons fait le choix de séparer ces deux classes car nous trouvions important de faire la différence entre la structure de stockage qu'est la matrice et l'outil mathématique qu'elle peut devenir.

## 3. Cfichier

La classe Cfichier permet la lecture de fichiers. En effet, nous avions besoin d'y accéder dans le constructeur par fichier de Cmatrice. Ccependant, les fonctions de lecture n'ayant pas leur place dans Cmatrice, nous avons créé une nouvelle classe.

Etant complètement décrochée de Cmatrice, nous voulions cette classe la plus complète possible (permettant ainsi son utilisation dans d'autres projets).

La classe se compose donc de plusieurs fonctions de lecture pour les différents types et besoins mais aussi d'une fonction permettant de sauter à des points dans le fichier. De plus, la classe gère elle aussi les exceptions.

## 4. Cexception

Notre classe Cexception est en grande partie basée sur celle de V. T'kindt. Elle permet la gestion des erreurs dans les classes du projet.

Les seules modifications faites à la classe sont l'ajout des codes d'erreur et la fonction d'affichage de ces erreurs.

## C. Problématiques

### 1. Type

Pour gérer le fait que les matrices doivent pouvoir contenir n'importe quel type, nous utilisons les « template ».

Cette fonctionalité est très pratique car elle permet de généraliser le code à n'importe quel type.

Cependant, généraliser le code peut engendrer l'apparition de plusieurs erreurs. Nous n'avons pas toujours pu trouver une réelle correction (en ligne de code) à ces erreurs. Cependant nous nous avons fait en sorte de rendre notre code complètement utilisable. Nous allons vous présenter la manière dont nous avons géré la difficulté liée à la généralisation dans les autres problémnatiques.

### 2. Opérations de base

Les calculs de base ont été mis dans la classe Ccalcul_matrice. Leurs codes se résument à la surcharge des opérateurs correspondants. Nous n'avons pas effectué de test sur les types des matrices dans les calculs. Dans les préconditons, nous avons marqué que pour pouvoir multiplier une matrice d'un type par un scalaire, il fallait que ce type supporte la multiplication.

Ainsi, nous ne bloquons pas les calculs aux types de base. Si un utilisateur a créé sa propre classe qui surcharge l'opérateur de multiplication, il pourra multiplier sa matrice. Il en est de même pour tous les calculs de la classe Ccalcul_matrice.

Pour l'affichage, nous surchargeons l'opérateur « << » entre un flux et une Cmatrice. De plus, nous avons fait le choix d'un affichage en une seule ligne. Celui ci peut sembler moins clair au premier abord, cependant il est beaucoup plus pratique qu'un affichage multiligne lorsqu'on affiche une matrice de matrice (par exemple). Pour que cette affichage reste efficace, les objets stockés dans les matrices doivent eux aussi s'afficher en une ligne (pas necessaire) et surcharge l'opérateur « << » entre le flux et l'objet.

### 3. Calculs complexes

Les calculs complexes fonctionnent sur le même principe que les calculs simples : ils n'ont aucune restriction sur le type mais demandent à ce que, dans la classe du type, les opérateurs correspondants soient surchargés.

### 4. Import via fichier

Pour permettre la création d'une matrice via un fichier, nous avons créé un constructeur dans Cmatrice prenant un chemin en paramètre. Grâce à la classe Cfichier, nous recupérons toutes les informations du fichier necessaires à la creation de la matrice. Si le type est different de « double », la matrice n'est pas créée. De plus, le client nous a fourni un format de fichier, si celui ci n'est pas respecté, la matrice ne sera pas créée non plus.

### 5. Gestion des exceptions

Pour gérer les exceptions, nous avons utilisé la classe de V. T'kindt : Cexception. A cette classe nous avons ajouté differents codes d'erreur qui sont « throw » dans beaucoup de fonctions du projet.

Pour repérer ces erreurs, l'utilisateur n'as qu'à entourer la section critique de son code d'un « try{...}catch(Cexception e){..} ». Il peut ensuite décider d'afficher l'erreur ou bien de la gérer comme il le souhaite.

L'utilisation de code d'erreur entré à la main directement dans le code de Cexception n'est pas très modulaire. Cependant nous n'avions pas le temps nécessaire pour modifier cela. Nous aurions pu, par exemple, créer un constructeur de Cexception qui prend en paramètre des codes d'erreur et leurs affichages. Ainsi dans la classe Cmatrice, nous aurions créer un seul objet Cexception commun à tout Cmatrice avec toutes les erreurs relatives et simplement modifier la valeur de l'objet avans de « throw ».

## D. Tests

Pour les tests, nous avons travaillé fonction par fonction. En effet, pour chaqu'une d'elle, nous avons tracé tous les chemins possibles que pouvais prendre l'exécution. De ces chemins nous avons deduis des paramètre pour les traverser, puis, dans le main, nous avons appeler autant de fois que necessaire chaque fonction pour passer dans tout le code.

Grâce a cette technique, nous pensons ainsi avoir testé une très grande partie des boucles, conditions et erreurs des fonctions de notre projet.

De plus, le client nous avait fourni un main. Nous avons donc aussi testé toutes les possibilités relatives a sa demande.

Enfin, nous avons utilisé visual leak detector pour repérer et supprimer une grande partie des fuites memoires de notre projet.

## E. Conclusion

Nous avions sous-estimé la force de ce projet. En effet, au début, nous pensions que le projet se terminerait assez vite, mais il etait plus complexe que l'on ne le pensait.
Il s'avère que cette librairie touche à beaucoup de points du developpement C++ : les templates, les exceptions, les fichiers… Et ce qui nous semblait être un projet banal au début, s'est retrouvé être un très bon moyen d'en apprendre plus et de nous donner quelques habitudes qui nous seront très utiles plus tard.

Grâce à ce projet, nous avons acquis de nouvelles connaissances autant en développement qu'en gestion de projet.