Le rapport de le projet Matrice

FENG Jiaming / MENG Qingling

1. **Le diagram de la classe de UML**



1. **La structure de projet de la matrice**

D’abord, mon projet contient 3 classes. Ils sont **CMatrice, CFichier, Cexception.** La classe **CMatrice** réalise les multiplation et opérations de la matrice, la classe **CFichier** réalise lire des données du fichier, la classe **CException** réalise la gestion des exceptions.

Parce que le titre demande que je peux créer des objets Matrice pour lequel le type des éléments est quelconque, donc je utilise **le patron** **de classe** pour le réaliser. La définition est . Après, je voudrais séparer la déclaration et définition de la classe CMatrice, donc je crée un CMatrice.hpp fichier pour le réaliser. A la fin, la déclaration de la classe CMatrice est dans le fichier CMatrice.h , et la définition de la classe CMatrice est dans le fichier CMatrice.hpp.

**3 La façon de projet**

Tout à bord, on appèlle la méthode de la classe CFichier pour lire des données du fichier, qui contient le nombre de ligne , le nombre de colonnes et des valeurs de la matrice. Ainsi le type du valeur de retour est CMatrice. Après on peux faire une série d’opérations à ce matrice, qui a besoin de appeler des méthodes de la classe CMatrice. Tous les exception possibles dans la classe de CFichier et CMatrice sont géré par la classe CException.

**4 La fonction principale :**

Pour tester tous les fonctions et méthodes, d’aboard, on crée un tableau de type CMatrice <double> qui s’appelle “**MatriceArray**” pour stocker les Matrices.

Et pour chaque nom de fichier passé en paramètre, on crée un objet de type CFichier et lire le fichier en utilisant le méthode **FICLire\_Fichier()**, ensuite on crée les matrice associées et affiche les matrices. S’il y a une erreur de lecture ou constructeur , on attraper une exception.

Après,on demande à l'utilisateur de saisir une valeur c et on fait l’algorithme suivant:

Afficher le résultat de la multiplication de chacune des matrices par la valeur c,

Afficher le résultat de la division de chacune des matrices par la valeur c,

Afficher le résultat de l’addition de toutes les matrices,

Afficher le résultat de l’opération :M1-M2+M3-M4+M5-M6...,

Afficher le résultat de du produit des matrices.

Afficher le résultat de la transposition de chacune des matrices.

En fin, on libérer la mémoire du tableau des matrices.

**5 La présentation des fonctions principaux**

* **Classe CMatrice**

Dans le fichier **CMatrice.h**, c’est la definition de la class Cmatrice. Dans la classe **CMatrice :**

Il y a 3 attributs: uiMATNBLignes, uiMATNBColonnes, pptMATMatrice

1. **uiMATNBLignes**：pour indique le nombre de ligne de la matrice
2. **uiMATNBColonnes**：pour indique le nombre de colonne de la matrice
3. **pptMATMatrice**：pour indique la matrice

Les opérations sur les matrices sont réalisées grâce aux surcharges d’opérateurs.

Il y a 22 méthodes, les informations dans la formulaire ci-dessous:

|  |  |
| --- | --- |
| **Les méthodes** | **La fonction** |
| 1. **CMatrice(void);** | Le constructeur par défault |
| 1. **CMatrice(unsigned int uiMATligne, unsigned int uiMATcolonne)** | Le constructeur avec MATligne lignes MATcolonne colonnes |
| 1. **CMatrice(CMatrice<Type> & MATB)** | Le constructeur recopie par défault |
| 1. **~CMatrice(void)** | Le destructeur |
| 1. **void DistribuerMemoire(unsigned int uiMATligne, unsigned int uiMATcolonne)** | Distribuer la mémoire pour la matrice de MATligne lignes MATcolonne colonnes |
| 1. **void MATModifierNBLignes(unsigned int uiValeur)** | Modifier le nombre de lignes |
| 1. **void MATModifierNBColonnes(unsigned int uiValeur)** | Modifier le nombre de colonnes |
| 1. **void MATModifierMatrice(unsigned int uiLigne, unsigned int uiColonne, Type tValeur)** | Modifier le valeur de uiLigne ligne uiColonne colonne |
| 1. **unsigned int MATLireNBLignes()** | Obtenir le nombre de lignes |
| 1. **unsigned int MATLireNBColonnes()** | Obtenirr le nombre de colonnes |
| 1. **Type MATLireMatrice()** | Obtenir la matrice |
| 1. **void MATAfficher();** | Afficher les 3 attributs de la matrice |
| 1. **void MATAfficherMatrice()** | Juste afficher la matrice |
| 1. **void MATAfficherTransposer()** | Afficher la matrice après transposer |
| 1. **CMatrice<Type> & operator = (const CMatrice<Type> & MATB)** | La surcharge d’operateur =, Réaliser M1=M2 |
| 1. **CMatrice<Type> operator +(const CMatrice<Type> & MATB)** | La surcharge d’operateur +,Réaliser M1+M2 |
| 1. **CMatrice<Type> operator -(const CMatrice<Type> & MATB)** | La surcharge d’operateur -,Réaliser M1-M2 |
| 1. **CMatrice<Type> operator \*(const CMatrice<Type> & MATB)** | La surcharge d’operateur \*,Réaliser M1\*M2 |
| 1. **CMatrice<Type> operator \*(Type tValeur)** | La surcharge d’operateur 8,Réaliser C x M |
| 1. **CMatrice<Type> \* MATMultiplier(Type tValeur)** | Réaliser M x c |
| 1. **CMatrice<Type> operator /(Type tValeur)** | La surcharge d’operateur /,Réaliser M / c |
| 1. **CMatrice<Type> \* MATTransposer()** | Réaliser M^T |

* **Classe CFichier**

Dans le fichier **CFichier.h**, c’est la definition de la class CFichier. Dans la classe **CFichier :**

Il y a un attribut : **char \* pcFICNom\_Fichier**, qui représente le nom de fichier. Cette méthode  lit le fihcier et les stocker dans un objet de CMatrice.La définition est dans le fichier CFichier.cpp,.La façon est ci-desous :

La lecture du fichier est réalisée à l’aide de la librairie fstream et par utilisation de getline() et de <ifstream>

ifstream File;

File.open(pcFICNom\_Fichier);

while (File.getline(buffer, 1024))

On lit les lignes du fichier en utilisant “getline” par une boucle et on distingue chaque ligne par variable fileligne.

Pour chaque ligne du fichier, on utilise le méthode suivant:

char \*\* FICLireValeur(char \* pcLine, char cDelimiter, unsigned int uiNBElement);

On sépare une ligne par “cDelimiter” et on stocke chaque élément dans un tableau de type char \*\* qui va retourner à le fonction FICLire\_Fichier().

Sur les trois premières lignes, on sépare une ligne par ‘=’.

Les lignes contient des éléments de la matrice est divisé par ‘ ’.

Et la fonction FICLire\_Fichier() se charge de lire et initialiser de nombre de lignes, de nombre de colonnes et des éléments de la matrice.

Le nom de type est comparé au type « double », pour les autres types, on lève une exception.

En fin, on retourne la matrice temporaire.

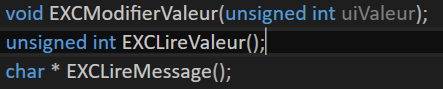
* **Classe CException**

Dans le fichier **CException.h**, c’est la definition de la class CException. Dans la classe **CException :**

Il y a 2 attributs : **unsigned int uiEXCValeur, char \* pcEXCMessage**

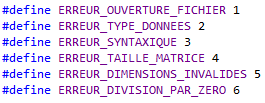
1. **unsigned int uiEXCValeur :** indiquer le numéro d’erreur.
2. **char \* pcEXCMessage**:indiquer le message d’erreur.

Il y a 3 méthodes,



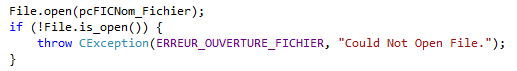
|  |  |
| --- | --- |
| **La méthode** | **La fonction** |
| **void EXCModifierValeur(unsigned int uiValeur)** | Modofier le numéro d’erreur |
| **unsigned int EXCLireValeur()** | Lire le numéro d’erreur |
| **char \* EXCLireMessage()** | Lire le message d’erreur |

On définit 6 codes d’erreur pour indiquer les types d’erreur.



Et dans le projet, on principalement utilise le constructeur suivant pour initialiser les exceptions.

CException(const unsigned int uivaleur, char \* pcMessage);



**5 Vérification de la mémoire**

Nous avons vérifié l’état de la mémoire de notre programme grâce à la bibliothèque crtdbg.h fournie par Visual Studio, afin d’éviter les fuites mémoire.

Pour utiliser cette bibliothèque, il faut mettre un fichier d'en - tête “DetectMemoryLeak.h”

#ifdef \_DEBUG

#define DEBUG\_CLIENTBLOCK new( \_CLIENT\_BLOCK, \_\_FILE\_\_, \_\_LINE\_\_)

#else

#define DEBUG\_CLIENTBLOCK

#endif

#define \_CRTDBG\_MAP\_ALLOC

#include <stdlib.h>

#include <crtdbg.h>

#ifdef \_DEBUG

#define new DEBUG\_CLIENTBLOCK

#endif

On appelle la fonction suivante, à la fin du main() :

**\_CrtDumpMemoryLeaks()**

Qui affichera dans la fenêtre de debug un résumé des fuites mémoire.