CS205 C/C++ Programming - Project 3

Name: 钟元吉(Zhong Yuanji)

SID: 12012613

Part 1 - Analysis

> This project is to implement only `float` elements matrix structure and some relavant functions only by using `C` language.

本次项目在要求只使用`C`语言，建立矩阵结构体及实现构造、复制、删除、矩阵间及矩阵与数的加法、减法、乘法、最大最小值、矩阵行列式、求逆等数学计算函数，其中需要考虑的点较多。在设计这样的矩阵操作库时，我们需要考虑以下问题：

1. 检查函数中传入的参数是否合理
2. 无法使用C++中的`#program once`，如何防止头文件重复包含导致的重定义
3. 如何记录返回结果的同时返回错误信息
4. 如何快速地进行矩阵求行列式与矩阵求逆
5. 如何从字符串中读取矩阵

以及在不同系统下运行产生的问题(见Part 4 问题1)。

1. 检查函数中传入的参数是否合理

考虑到使用者在使用函数是可能会将空指针作为参数传入函数，或者在取子矩阵和取余子式矩阵时输入了越界的行数或列数，也有可能在没有判断行列式不为0的时候将一个不可逆的矩阵求逆，因此，我们的程序需要对这些错误进行一一检验。

对输入的代码进行以下类型的错误检验：

错误类型	对应输出整数
运算中输入的第一个矩阵是空指针	-1
运算中输入的第二个矩阵是空指针	-2
运算中输出矩阵是空指针	-3
运算中输入的两个矩阵行数或列数错误，不满足运算要求	-4
取子矩阵或余子式时输入的行数或列数越界或有误	-5
求逆矩阵时，矩阵不可逆	-6

注意：在判断出现错误后，函数将不会对输出矩阵`ret`进行赋值。

前三种错误类型可以归纳为对于传入参数指针非空的要求，其中，传入未初始化的指针也是不允许的，例如，下面的做法是禁止的：

```
Matrix *mat; // 禁止传入未初始化的指针
float num = det(mat);
```

这里，我们可以通过4种方式初始化：

```
Matrix *mat1 = NULLMatrix;
Matrix *mat2 = createMatrix(1, 2, (__f *)malloc(2 * __SIZEF));
Matrix *mat3 = createMatrixFromStr("[1,2;3 4]");
Matrix *mat4 = (Matrix *)malloc(__SIZEM);
mat4->row = 1;
mat4->col = 2;
mat4->data = (__f *)malloc(2 * __SIZEF));
```

2. 如何防止头文件重复包含导致的重定义

在C++中，我们一般会在头文件开始时使用`#program once`来防止头文件重复包含。在C中，由于`#program once`是C++在C的基础上增加的内容，我们无法使用。因此我们在`.h`的头文件开始时，用将点变为下划线的文件名做为变量名，在**未定义过的条件下进行宏定义**，并在该条件下定义其他内容。如果之前已经导入该文件，或者关联到已经导入该头文件的`.o`文件时，其中一个将不再起作用。

进一步我们会在宏名称的前后加上双下划线，以防止出现重复。

3. 如何记录返回结果的同时返回错误信息

我们知道，在C和C++中，函数无法像在`matlab`中一样，同时返回多个值。因此我们可以考虑改变传入指针或引用对应的数据，以达到返回结果的目的。引用也是C++在C的基础上增加的内容，我们无法在C中使用，因此我们将函数**传入参数的最后一个作为返回结果的指针**。

但是，这也意味着存在一个问题，函数无法对这个矩阵结构体指针重新分配内存，也就是无法使这个指针指向新的矩阵结构体，于是，我们**要求传入的输出矩阵结构体指针不能是空指针**，要求输出矩阵结构体的浮点数数组指针不能指向非零的无效位置（因为非零时无法判断是否有效）。

4. 如何快速地进行矩阵求行列式与矩阵求逆

矩阵求行列式的方法有许多，但是每种方法有不同的时间复杂度。例如，我们可以用逆序数的方法，计算出$n!$项n个数相乘的展开，但是这样的的时间复杂度为$O(nn!)$。我们也可以使用某一行的余子式展开成n个`n-1`阶行列式来进行计算，这样计算的复杂度为$O(n!)$。而如果我们使用**高斯消元法**，将矩阵化简为上三角矩阵，然后取对角线相乘得到行列式，这样计算的时间复杂度为$O(n^3)$，这远远比前面的两种方法耗时短。我们只需在遇到$M_{i,i}=0$时寻找$M_{j,i}\neq 0, j>i$然后交换i,j两行，若$M_{j,i}=0, \forall j>i$，则这时行列式为0。

5. 如何从字符串中读取矩阵

为了更方便地构造矩阵，我们提供了从字符串到矩阵的构造方法，我们规定输入矩阵需要按照`matlab`中矩阵的输入格式，使用`[]`将矩阵包括，用英文逗号或前面无逗号、分号的空格作为列分割符，（前面有逗号、分号的空格将被忽略），用英文分号作为行分隔符（矩阵末尾`]`前，多余的逗号、分号、空格将被忽略）。对于不满足格式的字符串输入将会返回空矩阵`NULLMatrix`，例如：

```
Matrix *mat5 = createMatrixFromStr("[1, 2;3 4; ]");
printf(to_string(mat5));
```

则会输出：

```
Matrix 2x2:
[
   1.000000e+00   2.000000e+00
   3.000000e+00   4.000000e+00
]
```

实现过程：

1. 复制一份字符串。
2. 去除多余的空格，如果空格前为空格、逗号、分号将被去除，否则换为逗号。
3. 去除字符串末尾的右括号、逗号和分号。
4. 从第二个字符开始，记录逗号和分号个数，对应计算出行数和列数。
5. 初始化矩阵结构体，并逐行读取写入，并释放复制的字符串内存。
6. 如果出现错误，释放内存，返回空矩阵。

Part 2 - Code

由于代码较长，这里仅放置**矩阵求行列式**及**矩阵求逆**的部分代码，其他函数简介请参考源文件`MatrixFunc.c`或函数头文件`Matrix.h`，源文件中的注释比较详细。

注：将`float`类型定义为`__f`，如果需要使用`double`等其他类型时，便于转换。

```
// Compute the determinant of the matrix
__f det(const Matrix *mat)
{
    // Check if exist and rows equals columns
    if (!mat || mat->row == 0 || mat->col != mat->row)
        return NULLF;
    // Copy the data
```

```
    int row = mat->row, col = mat->col;
    __f *data1 = mat->data, ret = 1;
    __f *data = (__f *)malloc(row * col * __SIZEF);
    for (int i = 0; i < row * col; ++i)
        data[i] = data1[i];
    // Eliminate for each row using Gauss Method
    for (int i = 0; i < row; ++i)
    {
        // Check M_{i,i} != 0, if not change the rows
        for (int j = i; j < row; ++j)
            if (data[j * col + i] != 0)
            {
                if (i != j)
                { // Exchange two rows, determinant
                    for (int k = i; k < col; ++k)
                    {
                        __f tmpF = data[i * col + k];
                        data[i * col + k] = data[j * col + k];
                        data[j * col + k] = tmpF;
                    }
                    if ((j - i) % 2)
                        ret *= -1;
                }
                break;
            }
        // If no rows can make M_{i,i} != 0, the determinant is 0
        if (data[i * col + i] == 0)
            return 0;
        // Eliminate by using Gauss Method
        ret *= data[i * col + i];
        for (int j = i + 1; j < row; ++j)
        {
            __f num = data[j * col + i] / data[i * col + i];
            for (int k = i; k < col; ++k)
                data[j * col + k] -= num * data[i * col + k];
        }
    }
    // Delete the copy and return
    free(data);
    return ret;
}

// Compute the inverse of the matrix
int inv(const Matrix *mat, Matrix *ret)
{
    // Check if exist and rows equals columns
    __CheckMatRet;
    if (mat->row == 0 || mat->row != mat->col)
        return -4;
    __f matDet = det(mat);
    if (matDet == 0)
        return -6;
    // If the size of matrix is 1x1
    if (mat->row == 1)
    {
```

```
        if (!ret->data || ret->row != 1 || ret->col != 1)
        {
            if (ret->data)
                free(ret->data);
            ret->row = 1, ret->col = 1;
            ret->data = (__f *)malloc(__SIZEF);
        }
        ret->data[0] = 1 / mat->data[0];
        return 1;
    }
    // Else compute by using algebraic complement
    __RetMat(row, col,
             Matrix *tmp = NULLMatrix;
             cofactorMatrix(mat, j, i, tmp);
             data_[i * col + j] = ((i + j) % 2 ? -det(tmp) : det(tmp)) / matDet;
             deleteMatrix(tmp);
             ;);
    return 1;
}
```

Part 3 - Result & Verification

Test case #1: 基本要求的实现

注：程序中统一采用科学计数法进行输出

Test case #2: 对错误输入的判断

Test case #3: 对错误输入的判断

Test case #4: 程序特色功能

Part 4 - Difficulties & Solutions

问题1：在不同系统下运行产生的问题

问题2：为了更方便地使用其中的大部分功能，设计交互式的矩阵计算器

问题3：如何使矩阵的乘法效率更高

Part 5 - Summary

以上是本次Report的所有内容，感谢您的阅读！