Project 3 - A Library for Matrix Operations in C

Name: 匡亮(KuangLiang)

SID: 12111012

提交文件说明：

1. matrix.h: 定义、解释了所有实现的功能
2. matrix.c: 实现了 matrix.h 中定义的所有功能
3. test.c: 功能测试代码，实际应用时可删除
4. makefile
5. report.pdf

项目位于：https://github.com/sustechkl/libmatrix （10月31日零点后设置为 public ）

接口解释：

```
inline struct matrix *createMatrix(int r, int c);
```

生成一个 r 行 c 列的矩阵。

```
inline void clearMatrix(struct matrix *const mat);
```

将矩阵 mat 中的所有元素设为 0 。

```
inline struct matrix *createZeroMatrix(int r, int c);
```

生成一个 r 行 c 列的矩阵，并将所有元素初始化为 0 。

```
inline struct matrix *createIdenticalMatrix(int r);
```

生成一个 r 行 r 列的单位矩阵。

```
inline int deleteMatrix(struct matrix *const mat);
```

删除一个矩阵。

```
inline void deleteAllMatrices();
```

删除所有矩阵。

```
inline float getMatrixElement(const struct matrix *mat, int r, int c);
```

获取矩阵 mat 中第 r 行第 c 列的元素。

```
inline int getMatrixRowNumber(const struct matrix *mat);
```

获取矩阵 mat 的行的数量。

```
inline int getMatrixColumnNumber(const struct matrix *mat);
```

获取矩阵 mat 的列的数量。

```
inline int setMatrixElement(struct matrix *const mat, int r, int c, float val);
```

将矩阵 mat 中第 r 行第 c 列的元素设置为 val 。

```
inline int copyMatrix(struct matrix *const targetedMat, const struct matrix *originalMat);
```

将矩阵 originalMat 中的所有数据拷贝到 targetedMat 中。

targetedMat 的值不能被修改，否则用户将无法接收到 copy 的结果；originalMat 指向的内容应该是只读；因此 targetedMat 为 `struct matrix *const` 类型，originalMat 为 `const struct matrix *` 类型。

注意，targetedMat 必须是一个合法的矩阵地址（即由 createMatrix 函数得到的地址），否则拷贝将不会进行。

```
inline struct matrix *createMatrixCopy(const struct matrix *mat);
```

新建一个和 mat 完全相同的矩阵。

有时上一个函数使用起来不够方便，于是补充了一种使用方法。

```
inline int addScalar(struct matrix *const mat, float val);
```

将 mat 中所有元素加上 val 。

```
inline int subtractScalar(struct matrix *const mat, float val);
```

将 mat 中所有元素减去 val 。

```
inline int multiplyScalar(struct matrix *const mat, float val);
```

将 mat 中所有元素乘以 val 。

```
inline int addMatrix(struct matrix *const mat1, const struct matrix *mat2);
```

将 mat1 变为 mat1 + mat2 。

```
inline int subtractMatrix(struct matrix *const mat1, const struct matrix *mat2);
```

将 mat1 变为 mat1 - mat2 。

```
inline int multiplyMatrix(struct matrix *const mat1, const struct matrix *mat2);
```

将 mat1 变为 mat1 * mat2 。

```
inline float findMinimal(const struct matrix *mat);
```

找到 mat 中的最小值。

```
inline float findMaximal(const struct matrix *mat);
```

找到 mat 中的最大值。

用户不应当调用的接口：

用户只应当调用上文中的接口，以保证访问的安全性。下文中的接口则是为了确保访问的安全性。

为了确保用户不访问到错误的内存，以上所有接口在信任用户传入的指针之前都会进行合法性检查，当前所有合法的矩阵指针都将被一个链表维护（这个链表也是实现 deleteAllMatrices 的基础）。

理想情况下，用户总是用以上接口进行交互，总是不会遇到错误；否则，如果用户使用了错误的或自己生成的矩阵指针，接口将发现并拒绝为该指针服务，以避免发生错误，同时返回一个错误码。

```
inline int __checkMatrix(const struct matrix *mat);
```

检查 mat 是不是一个合法的指针。

```
inline void __addMatrixNode(struct matrix *mat);
```

将 mat 设为一个合法的指针。

```
inline void __removeMatrixNode(const struct matrix *mat);
```

将 mat 设为一个不再合法的指针。

测试

可以在本地编译运行 test.c， 里面实现了所有接口的测试。结果如下：

matrix A is:
1.00 2.00 3.00
4.00 5.00 6.00
7.00 8.00 9.00
matrix B is:
1.00 1.00 1.00 1.00
2.00 3.50 4.50 5.50
-1.00 -2.00 -3.00 -4.00
matrix C is:
1.00 0.00 0.00
0.00 1.00 0.00
0.00 0.00 1.00
matrix D is:
0.00 0.00 0.00
0.00 0.00 0.00
0.00 0.00 0.00
matrix A = A + 3 is:
4.00 5.00 6.00
7.00 8.00 9.00
10.00 11.00 12.00
matrix A = A - 3 is:
1.00 2.00 3.00
4.00 5.00 6.00
7.00 8.00 9.00
matrix A = A * 3 is:
3.00 6.00 9.00
12.00 15.00 18.00
21.00 24.00 27.00
matrix A = A + C is:
4.00 6.00 9.00
12.00 16.00 18.00
21.00 24.00 28.00
matrix A = A - C is:

3.00 6.00 9.00
12.00 15.00 18.00
21.00 24.00 27.00
matrix A = A * B is:
6.00 6.00 3.00 0.00
24.00 28.50 25.50 22.50
42.00 51.00 48.00 45.00
matrix H = A is:
6.00 6.00 3.00 0.00
24.00 28.50 25.50 22.50
42.00 51.00 48.00 45.00
matrix G = H is:
6.00 6.00 3.00 0.00
24.00 28.50 25.50 22.50
42.00 51.00 48.00 45.00
min in H is 0.00
max in H is 51.00
row number of H is 3
column number of H is 4
matrix H = 0 is:
0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00
Is matA exists? 1
delete matrix A
Is matA exists? 0
Is matB exists? 1
delete all matrices
Is matB exists? 0

困难与解决方案

一个难点在于，指针运用不熟练，担心在使用过程中错误地操作了一些内存。后来，通过 const 关键字限制了很多指针的访问，减少了很多可能发生的错误。

另一个难点则是如何处理用户的错误的指针使用。本来是通过判断传入的指针是不是 NULL ，但这样其实只能处理很少部分的错误。最后考虑维护了所有合法的指针组成的链表，虽然降低了效率，但如果用户只通过接口进行交互，基本不会遇到错误。如果将链表修改为平衡树，矩阵的数量很庞大时效率可以大幅提升。