

CPP_ASSIGNMENT3_REPORT@Jiko

Struct matrix

一个矩阵主要有三个基本信息需要存储，分别为行数、列数以及矩阵每个格子的对应值。另外根据矩阵的特点又分为方阵、对称矩阵、对角矩阵、单位矩阵等等。因此我构建的结构体分为两个部分，一个是基础信息部分，另一个是附加信息部分，基础信息即三个基本信息，实时更新；附加信息即矩阵的特点，可以选择在需要的时候再进行更新（更新详见：*Function for convenience - refreshMatrix()*）。

为使可存储的矩阵尽可能大，我选择了long来存储行数和列数。为方便读取，我选择了一维数组而不是二维数组。因为特征只有是或不是两个可能性，所以使用bool类型来存储。

```
1  struct matrix{
2      //basic info
3      long row;
4      long column;
5      float * data;
6      //addition info
7      bool square;      //方阵
8      bool diagnose;    //对角矩阵
9      bool symmetric;   //对称矩阵
10     bool identical;   //单位矩阵
11     //...可以继续添加
12 };
```

Function for users

createMatrix()

```
1  struct matrix * createMatrix(const long r,const long c,float * data);
```

思路构建

创建一个矩阵。将数组第n行拼到第n-1行后面，形成一个一维数组，从这一数组的0位开始依次放入data，直到 行数与列数乘积减一 位。

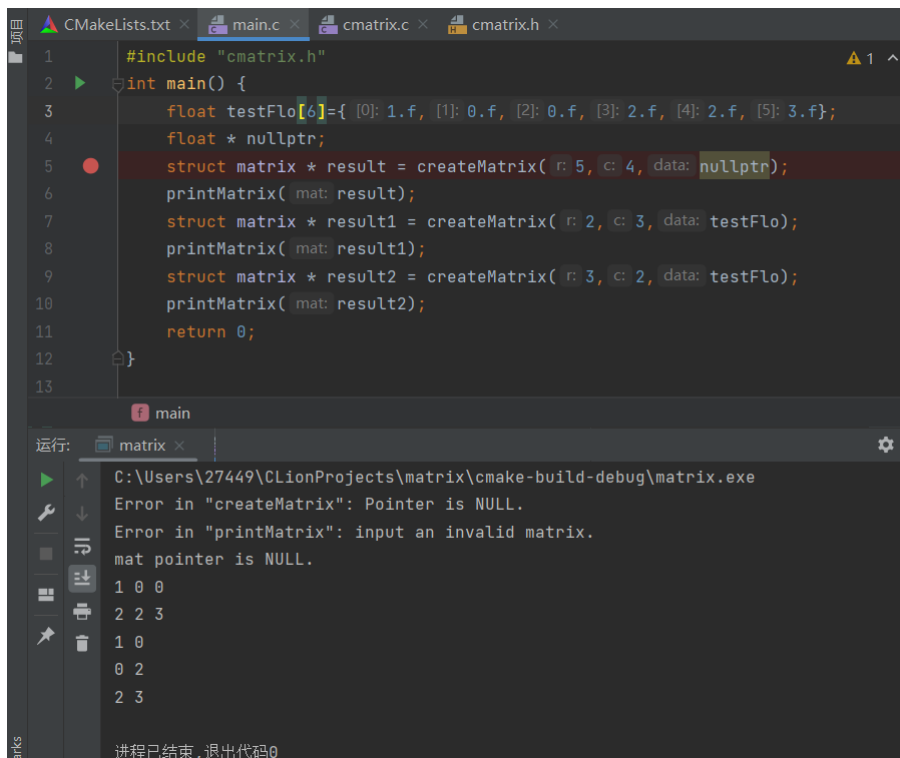
错误检查

如果传入的矩阵指针为空、矩阵r和c其中至少有一个小于等于0、data*指向为空则会输出对应错误提示，并返回一个NULL指针。

代码实现

```
1  struct matrix *createMatrix(const long r, const long c, float *data) { //一维数组导入
2
3      if (data == NULL) {
4          puts("Error in \"createMatrix\": Pointer is NULL.\n");
5          return NULL;
6      } else if (r * c == 0 || data == NULL) {
7          puts("Error in \"createMatrix\": Matrix is empty.\n");
8          return NULL;
9      }
10     struct matrix *newMat = (struct matrix *) malloc(1);
11     newMat->row = r;
12     newMat->column = c;
13     float *saveData;
14     saveData = (float *) malloc(r * c * sizeof(float));
15     for (int i = 0; i < r * c; ++i) {
16         saveData[i] = data[i];
17     }
18     newMat->data = saveData;
19     refreshType(newMat);
20     return newMat;
21 }
```

样例展示



```
1  #include "cmatrix.h"
2  int main() {
3      float testFlo[6]={ [0]: 1.f, [1]: 0.f, [2]: 0.f, [3]: 2.f, [4]: 2.f, [5]: 3.f};
4      float * nullptr;
5      struct matrix * result = createMatrix( r: 5, c: 4, data: nullptr);
6      printMatrix( mat: result);
7      struct matrix * result1 = createMatrix( r: 2, c: 3, data: testFlo);
8      printMatrix( mat: result1);
9      struct matrix * result2 = createMatrix( r: 3, c: 2, data: testFlo);
10     printMatrix( mat: result2);
11     return 0;
12 }

运行: matrix x
C:\Users\27449\CLionProjects\matrix\cmake-build-debug\matrix.exe
Error in "createMatrix": Pointer is NULL.
Error in "printMatrix": input an invalid matrix.
mat pointer is NULL.
1 0 0
2 2 3
1 0
0 2
2 3

进程已结束,退出代码0
```

case0构建时传入了空的指针，因此创建矩阵时打印了 `Error in "createMatrix": Pointer is NULL.`，case1、2为正确的传入方式，下面即为打印出来的矩阵1和矩阵2。

deleteMatrix()

```
1 void deleteMatrix(struct matrix ** mat);
```

思路构建

传入矩阵指针的地址，先释放其data的内存，再释放这个指针所占用的内存，最后再将这个指针指向NULL。

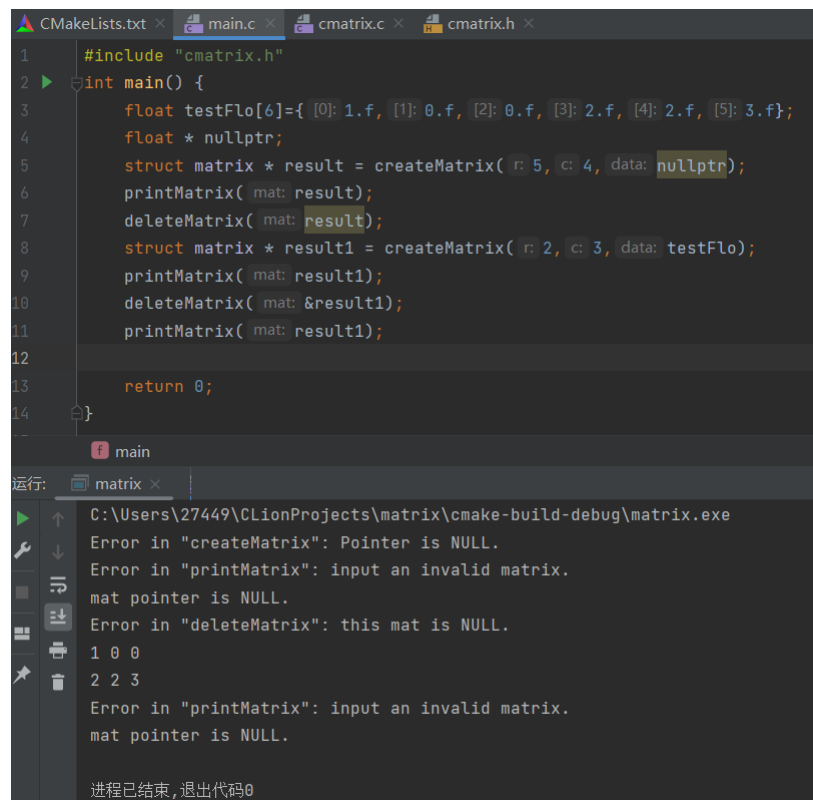
错误检查

如果传入的地址所指向的指针本身指向的就为NULL，那么除了输出错误提示什么也不会做。

代码实现

```
1 void deleteMatrix(struct matrix ** mat_loc){
2     if(mat_loc == NULL){
3         printf("Error in \"deleteMatrix\": this mat is NULL.\n");
4         return;
5     }
6     free((*mat_loc)->data);
7     free(*mat_loc);
8     *mat_loc = NULL;
9 }
```

样例展示



```
CMakeLists.txt x main.c x cmatrix.c x cmatrix.h x
1 #include "cmatrix.h"
2 int main() {
3     float testFlo[6]={ [0]: 1.f, [1]: 0.f, [2]: 0.f, [3]: 2.f, [4]: 2.f, [5]: 3.f};
4     float * nullptr;
5     struct matrix * result = createMatrix( R: 5, C: 4, data: nullptr);
6     printMatrix( mat: result);
7     deleteMatrix( mat: result);
8     struct matrix * result1 = createMatrix( R: 2, C: 3, data: testFlo);
9     printMatrix( mat: result1);
10    deleteMatrix( mat: &result1);
11    printMatrix( mat: result1);
12
13    return 0;
14 }
```

运行: matrix x

```
C:\Users\27449\CLionProjects\matrix\cmake-build-debug\matrix.exe
Error in "createMatrix": Pointer is NULL.
Error in "printMatrix": input an invalid matrix.
mat pointer is NULL.
Error in "deleteMatrix": this mat is NULL.
1 0 0
2 2 3
Error in "printMatrix": input an invalid matrix.
mat pointer is NULL.
进程已结束,退出代码0
```

case0是一个空的指针，所以delete操作会有错误提示；

case1是一个合法的矩阵指针，删除后再打印会提示矩阵指针已经指向了NULL。

copyMatrix()

```
1 struct matrix * copyMatrix(const struct matrix * const mat);
```

思路构建

将要拷贝的数组的所有数据都拷贝一份，然后重新create一个内存位置不同的矩阵。

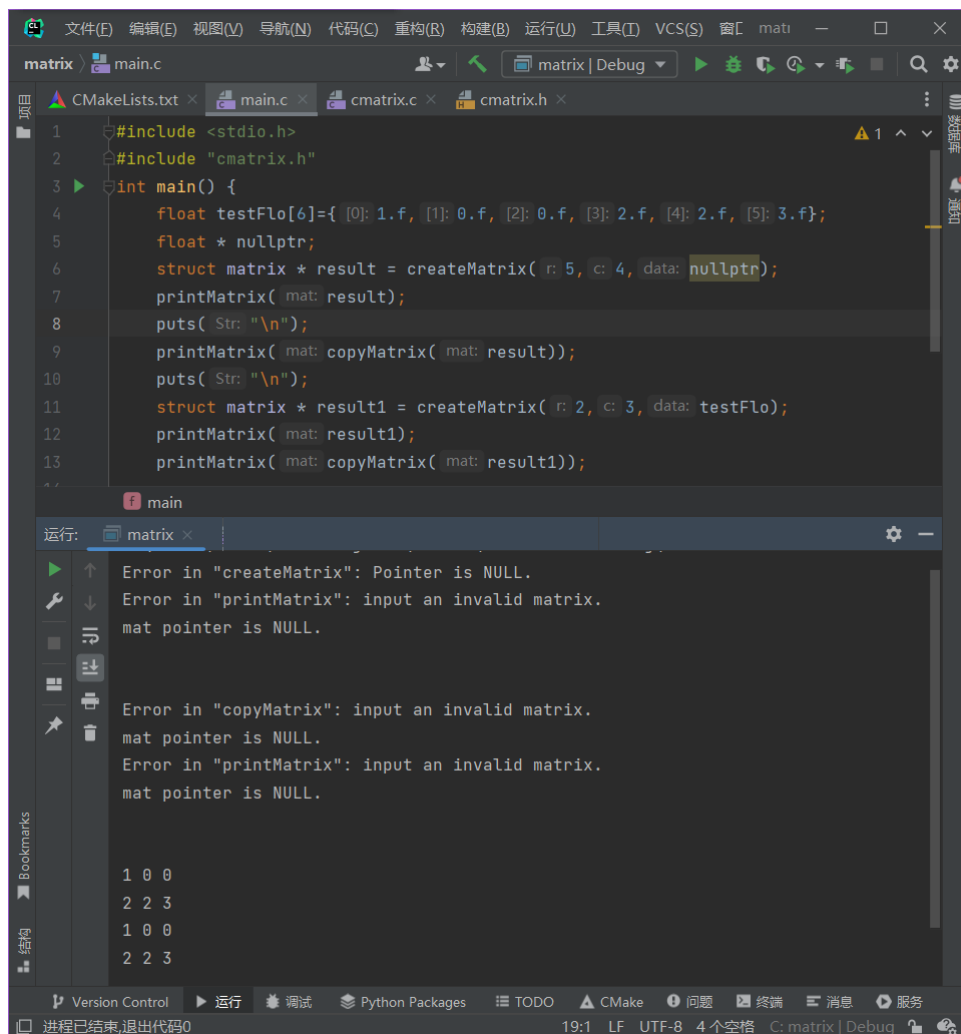
错误检查

检查这个将要被拷贝的矩阵是否合法，若不合法则错误提示，并返回一个空指针。

代码实现

```
1 struct matrix *copyMatrix(const struct matrix *const mat) {
2     if (mat == NULL || mat->column<=0 || mat->row<=0 || mat->data==NULL) {
3         printf("Error in \"copyMatrix\": input an invalid matrix.\n");
4         printFalse(mat);
5         return NULL;
6     }
7     float array[mat->row * mat->column];
8     for (int i = 0; i < mat->row * mat->column; ++i) {
9         array[i] = mat->data[i];
10    }
11    return createMatrix(mat->row, mat->column, array);
12 }
```

样例展示



```
1 #include <stdio.h>
2 #include "cmatrix.h"
3 int main() {
4     float testFlo[6]={ [0]: 1.f, [1]: 0.f, [2]: 0.f, [3]: 2.f, [4]: 2.f, [5]: 3.f};
5     float * nullptr;
6     struct matrix * result = createMatrix( r: 5, c: 4, data: nullptr);
7     printMatrix( mat: result);
8     puts( Str: "\n");
9     printMatrix( mat: copyMatrix( mat: result));
10    puts( Str: "\n");
11    struct matrix * result1 = createMatrix( r: 2, c: 3, data: testFlo);
12    printMatrix( mat: result1);
13    printMatrix( mat: copyMatrix( mat: result1));
14 }
```

运行: matrix

```
Error in "createMatrix": Pointer is NULL.
Error in "printMatrix": input an invalid matrix.
mat pointer is NULL.

Error in "copyMatrix": input an invalid matrix.
mat pointer is NULL.
Error in "printMatrix": input an invalid matrix.
mat pointer is NULL.

1 0 0
2 2 3
1 0 0
2 2 3
```

case0为空指针，复制时返回出错提示；case1为合法的矩阵指针，打印发现两个矩阵相同。

```
2 #include "cmatrix.h"
3
4 int main() {
5     float testFlo[6]={ [0]: 1.f, [1]: 0.f, [2]: 0.f, [3]: 2.f, [4]: 2.f, [5]: 3.f};
6     struct matrix * result1 = createMatrix( r: 2, c: 3, data: testFlo);
7     printMatrix( mat: result1);
8     struct matrix * copiedMatrix = copyMatrix( mat: result1);
9     deleteMatrix( mat: &result1);
10    printMatrix( mat: copiedMatrix);
11
12    return 0;
13 }
```

运行: matrix x

C:\Users\27449\CLionProjects\matrix\cmake-build-debug\matrix.exe

```
1 0 0
2 2 3
1 0 0
2 2 3
```

进程已结束,退出代码0

删掉被拷贝的矩阵后，拷贝的矩阵依然不变，说明二者没有共用内存。

addMatrix()

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

思路构建

输入两个矩阵指针，两个指针的row和column相同，创建一个新数组，数组的长度为row * column，依次将对应的data存入这一数组，最后再使用这一数组create一个行数为row，列数为column的新矩阵，返回这个矩阵的指针。

错误检查

检查矩阵指针是否指向NULL，是否存在column或row<=0，是否data指向NULL，如果有则输出相应错误提示，最后返回指向NULL的矩阵指针。

代码实现

```
1 struct matrix *addMatrix(const struct matrix *const mat1, const struct matrix
    *const mat2) {
2     if (mat1 == NULL || mat1->column<=0 || mat1->row<=0 || mat1->data==NULL) {
3         printf("Error in \"addMatrix\": input an invalid matrix(left).\n");
4         printFalse(mat1);
5         return NULL;
6     } else if (mat2 == NULL || mat2->column<=0 || mat2->row<=0 || mat2->data==NULL) {
7         printf("Error in \"copyMatrix\": input an invalid matrix(right).\n");
8         printFalse(mat2);
9         return NULL;
10    } else if (mat1->column != mat2->column || mat1->row != mat2->row) {
11        printf("Error in \"addMatrix\": matrix1 %d columns, matrix1 %d rows\n",
            mat1->column, mat1->row);
12        printf("Error in \"addMatrix\": matrix2 %d columns, matrix2 %d rows\n",
            mat2->column, mat2->row);
```

```

13         return NULL;
14     }
15     int c = mat1->column;
16     int r = mat1->row;
17     float array[c * r];
18     for (int i = 0; i < c * r; ++i) {
19         array[i] = mat1->data[i] + mat2->data[i];
20     }
21     return createMatrix(r, c, array);
22 }

```

样例展示

```

#include <stdio.h>
#include "cmatrix.h"
int main() {
    float case1_data[6]={ [0]: 1.f, [1]: 0.f, [2]: 0.f, [3]: 2.f, [4]: 2.f, [5]: 3.f};
    float case2_data[6]={ [0]: 1.f, [1]: 0.f, [2]: 5.f, [3]: 7.f, [4]: 7.f, [5]: 0.f};
    struct matrix * case1_1 = createMatrix( r: 2, c: 3, data: case1_data);
    struct matrix * case1_2 = createMatrix( r: 3, c: 2, data: case1_data);
    struct matrix * case2 = createMatrix( r: 2, c: 3, data: case2_data);
    printMatrix( mat: addMatrix( mat1: case1_1, mat2: case1_2));
    printf( format: "\n");
    printMatrix( mat: addMatrix( mat1: case1_1, mat2: case2));

    return 0;
}

```

main

```

C:\Users\27449\CLionProjects\matrix\cmake-build-debug\matrix.exe
Error in "addMatrix": matrix1 3 columns, matrix1 2 rows
matrix2 2 columns, matrix2 3 rows
Error in "printMatrix": input an invalid matrix.
mat pointer is NULL.

2 0 5
9 9 3

进程已结束,退出代码0

```

第一个case是两个尺寸不同的矩阵相加，因此输出空指针。第二个case合法，输出了正确的矩阵。

subMatrix()

```

1 struct matrix * subMatrix(const struct matrix * const mat1, const struct matrix * const mat2);

```

思路构建

输入两个矩阵指针，两个指针的row和column相同，创建一个新数组，数组的长度为row * column，依次将对应的data相减存入这一数组，最后再使用这一数组create一个行数为row，列数为column的新矩阵，返回这个矩阵的指针。

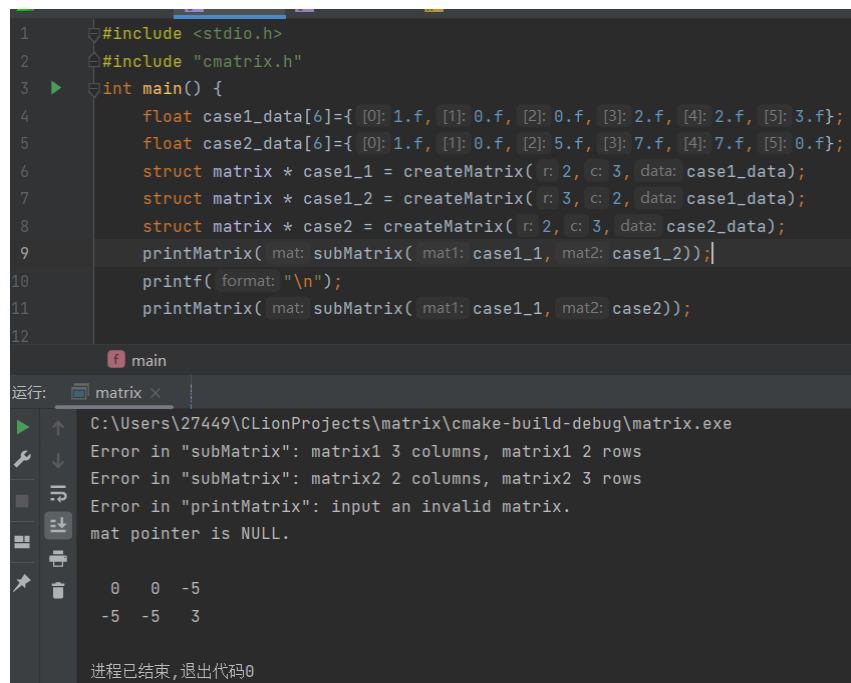
错误检查

检查矩阵指针是否指向NULL，是否存在column或row<=0，是否data指向NULL，如果有则输出相应错误提示，最后返回指向NULL的矩阵指针。

代码实现

```
1 struct matrix *subMatrix(const struct matrix *const mat1, const struct matrix
   *const mat2) {
2     if (mat1 == NULL || mat1->column <= 0 || mat1->row <= 0 || mat1->data == NULL) {
3         printf("Error in \"subMatrix\": input an invalid matrix(left).\n");
4         printfFalse(mat1);
5         return NULL;
6     } else if (mat2 == NULL || mat2->column <= 0 || mat2->row <= 0 || mat2->data == NULL) {
7         printf("Error in \"subMatrix\": input an invalid matrix(right).\n");
8         printfFalse(mat2);
9         return NULL;
10    } else if (mat1->column != mat2->column || mat1->row != mat2->row) {
11        printf("Error in \"subMatrix\": matrix1 %d columns, matrix1 %d rows\n",
mat1->column, mat1->row);
12        printf("Error in \"subMatrix\": matrix2 %d columns, matrix2 %d rows\n",
mat2->column, mat2->row);
13        return NULL;
14    }
15    int c = mat1->column;
16    int r = mat1->row;
17    float array[c * r];
18    for (int i = 0; i < c * r; ++i) {
19        array[i] = mat1->data[i] - mat2->data[i];
20    }
21    return createMatrix(r, c, array);
22 }
```

样例展示



The screenshot shows a C++ IDE with a code editor and a console window. The code defines a `subMatrix` function and a `main` function. The `main` function creates two matrices, `case1_1` and `case1_2`, and calls `subMatrix` to subtract `case1_2` from `case1_1`. The console output shows the execution of the program, including error messages for invalid matrix dimensions and a successful subtraction result.

```
1 #include <stdio.h>
2 #include "cmatrix.h"
3 int main() {
4     float case1_data[6] = { [0]: 1.f, [1]: 0.f, [2]: 0.f, [3]: 2.f, [4]: 2.f, [5]: 3.f };
5     float case2_data[6] = { [0]: 1.f, [1]: 0.f, [2]: 5.f, [3]: 7.f, [4]: 7.f, [5]: 0.f };
6     struct matrix * case1_1 = createMatrix( r: 2, c: 3, data: case1_data );
7     struct matrix * case1_2 = createMatrix( r: 3, c: 2, data: case1_data );
8     struct matrix * case2 = createMatrix( r: 2, c: 3, data: case2_data );
9     printMatrix( mat: subMatrix( mat1: case1_1, mat2: case1_2 ) );
10    printf( format: "\n" );
11    printMatrix( mat: subMatrix( mat1: case1_1, mat2: case2 ) );
12 }
```

运行: matrix x

C:\Users\27449\CLionProjects\matrix\cmake-build-debug\matrix.exe

Error in "subMatrix": matrix1 3 columns, matrix1 2 rows

Error in "subMatrix": matrix2 2 columns, matrix2 3 rows

Error in "printMatrix": input an invalid matrix.

mat pointer is NULL.

0 0 -5

-5 -5 3

进程已结束,退出代码0

第一个case是两个尺寸不同的矩阵相减，因此输出空指针。第二个case合法，输出了正确的矩阵。

mulMatrix()

```
1 struct matrix * mulMatrix(const struct matrix * const mat1, const struct matrix *  
    const mat2);
```

思路构建

mat1的每一行分别乘mat2的每一列，依次放入数组array中。

最终结果的矩阵行数与mat1相同，列数与mat2相同。最后create一个新的矩阵并返回其指针。

错误检查

检查两个矩阵指针是否指向NULL，是否存在column或row<=0，是否data指向NULL，如果有则输出相应错误提示，且返回一个指向NULL的指针。

另外还需检查mat1的列数是否与mat2的行数相等，否则输出相应错误提示，且返回一个指向NULL的指针。

代码实现

```
1 struct matrix * mulMatrix(const struct matrix * const mat1, const struct matrix  
    * const mat2) {  
2     if (mat1 == NULL || mat1->column<=0 || mat1->row<=0 || mat1->data == NULL) {  
3         printf("Error in \"mulMatrix\": Input a invalid matrix(left)");  
4         printFalse(mat1);  
5         return NULL;  
6     } else if (mat2 == NULL || mat2->column<=0 || mat2->row<=0 || mat2->data == NULL)  
7     {  
8         printf("Error in \"mulMatrix\": Input a invalid matrix(right)");  
9         printFalse(mat2);  
10        return NULL;  
11    } else if (mat1->column != mat2->row) {  
12        printf("Error in \"mulMatrix\": mat1's column(%d) not equal mat2's  
13        row(%d).\n", mat1->column, mat2->row);  
14        return NULL;  
15    }  
16    long r = mat1->row;  
17    long c = mat2->column;  
18    float array[r * c];  
19    long indexResult = 0;  
20    for (int i = 0; i < mat1->row; ++i) {  
21        for (int j = 0; j < mat2->column; ++j) {  
22            array[indexResult] = 0;  
23            for (int k = 0; k < mat1->column; ++k) {  
24                array[indexResult] += mat1->data[i * mat1->column + k] * mat2-  
25                >data[j + mat2->column * k];  
26            }  
27            indexResult++;  
28        }  
29    }  
30    struct matrix * newMatrix = createMatrix(r, c, array);  
31    return newMatrix;  
32 }
```


样例展示

```
#include <stdio.h>
#include "cmatrix.h"

int main() {
    float case1_data[6]={ [0]: 1.f, [1]: 0.f, [2]: 0.f, [3]: 2.f, [4]: 2.f, [5]: 3.f};
    float case2_data[6]={ [0]: 1.f, [1]: 0.f, [2]: 5.f, [3]: 7.f, [4]: 7.f, [5]: 0.f};
    struct matrix * case1 = createMatrix( r: 3, c: 2, data: NULL);
    struct matrix * case2 = createMatrix( r: 2, c: 3, data: case2_data);
    printMatrix( mat: mulMatrix( mat1: case1, mat2: case2));
    printMatrix( mat: mulMatrix( mat1: case2, mat2: transportMatrix( mat: case2)));

    return 0;
}
```

main

matrix x

C:\Users\27449\CLionProjects\matrix\cmake-build-debug\matrix.exe
Error in "createMatrix": Pointer is NULL.
Error in "mulMatrix": Input a invalid matrix(left)mat pointer is NULL.
Error in "printMatrix": input an invalid matrix.
mat pointer is NULL.
26 7
7 98
进程已结束,退出代码0

addScalar() and subScalar() and mulScalar()

```
1 void addScalar(struct matrix * const mat, float scalar);
```

```
1 void subScalar(struct matrix * const mat, float scalar);
```

```
1 void mulScalar(struct matrix * const mat, float scalar);
```

思路构建

传入矩阵指针以及要加(减/乘)的常数，直接在原矩阵的data上作修改。

错误检查

检查矩阵指针是否指向NULL，是否存在column或row<=0，是否data指向NULL，如果有则输出相应错误提示，除此之外不做任何操作。

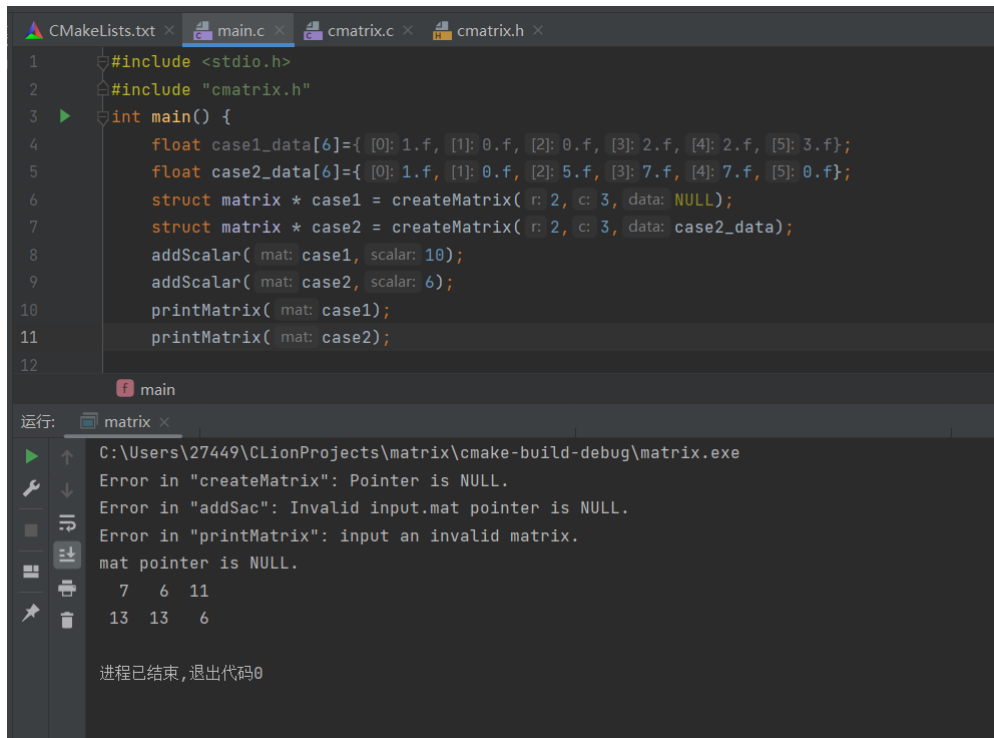
代码实现

以加法为例

```
1 void addScalar(struct matrix *const mat, float scalar) {
2     if (mat == NULL || mat->column<=0 || mat->row<=0 || mat->data==NULL) {
3         printf("Error in \"addSac\": Invalid input.");
4         printFalse(mat);
5         return;
6     }
7     for (int i = 0; i < mat->column * mat->row; ++i) {
8         mat->data[i] += scalar;
9     }
10 }
```

样例展示

以加法为例



```
1 #include <stdio.h>
2 #include "cmatrix.h"
3 int main() {
4     float case1_data[6]={ [0]: 1.f, [1]: 0.f, [2]: 0.f, [3]: 2.f, [4]: 2.f, [5]: 3.f};
5     float case2_data[6]={ [0]: 1.f, [1]: 0.f, [2]: 5.f, [3]: 7.f, [4]: 7.f, [5]: 0.f};
6     struct matrix * case1 = createMatrix( r: 2, c: 3, data: NULL);
7     struct matrix * case2 = createMatrix( r: 2, c: 3, data: case2_data);
8     addScalar( mat: case1, scalar: 10);
9     addScalar( mat: case2, scalar: 6);
10    printMatrix( mat: case1);
11    printMatrix( mat: case2);
12 }

运行: matrix x
C:\Users\27449\CLionProjects\matrix\cmake-build-debug\matrix.exe
Error in "createMatrix": Pointer is NULL.
Error in "addSac": Invalid input.mat pointer is NULL.
Error in "printMatrix": input an invalid matrix.
mat pointer is NULL.
7 6 11
13 13 6

进程已结束,退出代码0
```

findMin() and findMax()

```
1 float findMin(const struct matrix * const mat);
```

```
1 float findMax(const struct matrix * const mat);
```

思路构建

遍历矩阵的data，返回最小(大)的值。

错误检查

检查矩阵指针是否指向NULL，是否存在column或row<=0，是否data指向NULL，如果有则输出相应错误提示，且返回0.0f。

代码实现

以查找最小值为例

```
1 float findMin(const struct matrix *const mat) {
2     if (mat == NULL || mat->column<=0 || mat->row<=0 || mat->data==NULL) {
3         printf("Error in \"findMin\": Invalid input.");
4         printFalse(mat);
5         return 0.0f;
6     }
7     float min = mat->data[0];
8     for (int i = 0; i < mat->row * mat->column; ++i) {
9         if (mat->data[i] < min) {
10             min = mat->data[i];
11         }
12     }
13     return min;
```

样例展示

```
#include <stdio.h>
#include "cmatrix.h"
int main() {
    float case1_data[6]={ [0]: 1.f, [1]: 0.f, [2]: 0.f, [3]: 2.f, [4]: 2.f, [5]: 3.f};
    float case2_data[6]={ [0]: 1.f, [1]: 0.f, [2]: 5.f, [3]: 7.f, [4]: 7.f, [5]: 0.f};
    struct matrix * case1 = createMatrix( r: 2, c: 3, data: NULL);
    struct matrix * case2 = createMatrix( r: 2, c: 3, data: case2_data);
    printf( format: "case1 min : %f, case2 max : %f ", findMin( mat: case1), findMax( mat: case2));

    return 0;
}
main
matrix x i
C:\Users\27449\CLionProjects\matrix\cmake-build-debug\matrix.exe
Error in "createMatrix": Pointer is NULL.
Error in "findMin": Invalid input.mat pointer is NULL.
case1 min : 0.000000, case2 max : 7.000000
进程已结束,退出代码0
```

case1是一个非法的矩阵地址，因此有对应的错误提示，case2输出了矩阵中的最大值。

transportMatrix()

```
1 struct matrix * transportMatrix(const struct matrix * const mat);
```

思路构建

将(n,m)处的值放到(m,n)的地方，把row和column交换，如此create一个新的矩阵，并返回其地址。

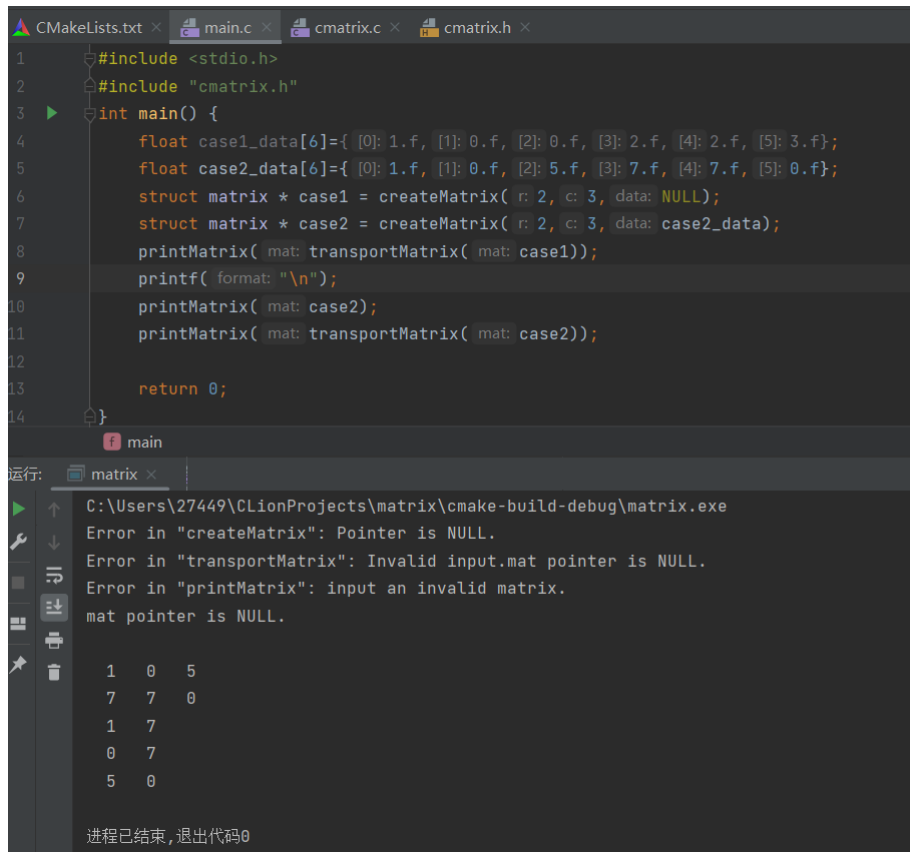
错误检查

检查矩阵指针是否指向NULL，是否存在column或row<=0，是否data指向NULL，如果有则输出相应错误提示，且返回一个指向NULL的指针。

代码实现

```
1 struct matrix *transportMatrix(const struct matrix *const mat) {
2     if (mat == NULL || mat->column<=0||mat->row<=0 || mat->data==NULL) {
3         printf("Error in \"transportMatrix\": Invalid input.");
4         printFalse(mat);
5         return NULL;
6     }
7     long r = mat->column;
8     long c = mat->row;
9     long index = 0;
10    float array[r * c];
11    for (int i = 0; i < mat->column; ++i) {
12        for (int j = 0; j < mat->row; ++j) {
13            array[index++] = mat->data[i + j * mat->column];
14        }
15    }
16    struct matrix *newMatrix = createMatrix(r, c, array);
17    return newMatrix;
18 }
```

样例展示



```
1 #include <stdio.h>
2 #include "cmatrix.h"
3 int main() {
4     float case1_data[6]={ [0]: 1.f, [1]: 0.f, [2]: 0.f, [3]: 2.f, [4]: 2.f, [5]: 3.f};
5     float case2_data[6]={ [0]: 1.f, [1]: 0.f, [2]: 5.f, [3]: 7.f, [4]: 7.f, [5]: 0.f};
6     struct matrix * case1 = createMatrix( r: 2, c: 3, data: NULL);
7     struct matrix * case2 = createMatrix( r: 2, c: 3, data: case2_data);
8     printMatrix( mat: transportMatrix( mat: case1));
9     printf( format: "\n");
10    printMatrix( mat: case2);
11    printMatrix( mat: transportMatrix( mat: case2));
12
13    return 0;
14 }
```

运行: matrix x

C:\Users\27449\CLionProjects\matrix\cmake-build-debug\matrix.exe
Error in "createMatrix": Pointer is NULL.
Error in "transportMatrix": Invalid input.mat pointer is NULL.
Error in "printMatrix": input an invalid matrix.
mat pointer is NULL.

1	0	5
7	7	0
1	7	
0	7	
5	0	

进程已结束,退出代码0

Function for convenience

refreshType()

```
1 void refreshType(struct matrix * const mat);
```

用于更新矩阵的附加条件的状态。每次create一个新矩阵的时候都会启用。

- 如果后续有函数需要使用到附加条件的时候只要在开头refresh这个矩阵就能正常使用了。
- 如果后续需要添加新的附加条件，只要在头文件中添加新的bool类型，并在refreshType中添加正则即可。

printMatrix()

```
1 void printMatrix(const struct matrix * mat);
```

用于打印矩阵，根据矩阵的数据有效位数可以修改%f的参数使得数据能够向右对齐。