Project 3: A Libray for Matrix Operations in C

Name: 周思呈
SID: 12110644

Part 01 - Analysis

要求设计一个关于矩阵的库，使用对象为懂一些编程但可能会干坏事的程序员。要求使用方便、内存管理恰当。只能用c语言实现，所以不能用reference和overload。

Part 02 - Code

数据储存方式

定义一个叫做Matrix的结构体，里面包含三个元素，分别为行数、列数和矩阵里存的值。其中值的数组用指针实现，便于内存管理。

```
typedef struct _Matrix
{
    int row;
    int col;
    float *value;
} Matrix;
```

创建矩阵

输入四个参数，分别为行数、列数和是否随机生成。用const修饰输入的参数，防止输入参数被修改。
首先进行行列数检查，如果小于等于零则判定为非法，通过打印告诉使用者，并且返回一个空指针。如果合法，则申请一块内存给指针matrix，再申请一块内存给matrix中的指针value。
然后对合法矩阵进行赋值。如果随机生成，则调用为随机函数给value赋值。如果非随机生成，则指引用户输入value值，一次输入col个值，输入row次。对于输入，判断其是否为合法的float值，如果不合法则让用户重新输入。

```
Matrix *createMatrix(const int row, const int col, const bool random)
{
    //输入检查
    if (!legalSize(row, col))
    {
        printf("your row and col are too small!\n");
        return NULL;
    }
    else
    {
        Matrix *matrix = (Matrix *)malloc(sizeof(Matrix));
        matrix->col = col;
        matrix->row = row;
        char *input = (char *)malloc(sizeof(char) * 64); //用来接收输入的字符串
        // srand((unsigned)time(NULL));
        matrix->value = (float *)malloc(row * col * sizeof(float));
        for (int r = 0; r < row; r++)
        {
            for (int c = 0; c < col; c++)
            {
                if (random)
                {
                    matrix->value[r * col + c] = rand() % 100 + 5;
                }
                else
                {
                    if (c == 0)
                    {
                        printf("please input next %d values of your matrix\n", col);
                    }

                    scanf("%s", input);
                    if (legal(input))
                    {
                        matrix->value[r * col + c] = atof(input);
                    }
                    else
                    {
                        printf("you should input a legal float number\n");
                        c--;
                    }
                }
            }
        }
        free(input);
        return matrix;
    }
}

bool legal(const char *input)
{
    int *i = (int *)malloc(4);
    *i = 0;
    while (*(input + *i) != 0)
    {
        if ((*(input + *i) < '0' || *(input + *i) > '9') && (*(input + *i) != '.'))
        {
            free(i);
            return false;
        }
        else
        {
            (*i)++;
        }
    }
    free(i);
    return true;
}
bool legalSize(int row, int col)
{
    if (row <= 0 || col <= 0)
    {
        return false;
    }
    else
    {
        return true;
    }
}
```

删除矩阵

由两部分组成：第一部分定义一个宏，在删除矩阵的同时把对应的指针置NULL；第二部分实现一个方法，如果传入的指针不为空则进行以下操作：传入矩阵指针，把矩阵里面全部元素置0，free指针matrix和指针matrix->value的内存，再将value置NULL。返回是否成功删除。方法里面传入的matrix指针是一个地址的复制值，在方法内部置NULL并不能将方法外部的指针也置NULL，因此用宏来完成这个操作。

```
#define delete(p) (deleteMatrix(p), p=NULL)

bool deleteMatrix(Matrix *matrix)
{
    if (matrix != NULL)
    {
        int size = matrix->col * matrix->row;
        for (int i = 0; i < size; i++)
        {
            matrix->value[i] = 0;
        }
        free(matrix->value);
        matrix->value = NULL;
        matrix->col = 0;
        matrix->row = 0;
        free(matrix);
        // printf("please remember to set this pointer to NULL\n");
        return true;
    }
    else
    {
        return false;
    }
}
```

复制矩阵

定义一个复制矩阵的方法，传入一个矩阵的指针，如果非空则创建一个新的矩阵指针，讲原矩阵的行数、列数、值赋给新矩阵并将新矩阵返回。const修饰防止原矩阵被修改。

```
Matrix *copy(const Matrix *matrix)
{
    if (matrix == NULL)
    {
        printf("this matrix has been deleted!\n");
        return NULL;
    }
    else
    {
        Matrix *result = (Matrix *)malloc(sizeof(Matrix));
        int row = matrix->row;
        int col = matrix->col;
        result->col = col;
        result->row = row;
        int size = col * row;
        result->value = (float *)malloc(row * col * sizeof(float));
        for (int i = 0; i < size; i++)
        {
            result->value[i] = matrix->value[i];
        }
        return result;
    }
}
```

矩阵加法

定义矩阵加法的方法，输入两个矩阵指针，输出结果矩阵指针。const修饰输入变量防止修改。
首先进行输入合法性判断，如果输入的两个矩阵大小不相同，就提示用户无法进行矩阵加法并返回NULL。如果矩阵大小合法则创建结果矩阵。计算行列数乘积进行一维遍历，连续访问内存，提高数据读写效率。如果相加结果超出浮点数储存范围，则提示用户out of range，释放内存，返回NULL。如果一切正常，则返回结果矩阵指针。

```
Matrix *addMatrix(const Matrix *matrix1, const Matrix *matrix2)
{
    if (!(legalSize(matrix1->row, matrix1->col)
    && legalSize(matrix2->row, matrix2->col)))
    {
        printf("this matrix is empty!\n");
        return NULL;
    }
    if (!sameSize(matrix1, matrix2))
    {
        printf("two matrices have different size which can not be added!\n");
        return NULL;
    }
    else
    {
        Matrix *result = (Matrix *)malloc(sizeof(Matrix));
        int row = matrix1->row;
        int col = matrix1->col;
        result->col = col;
        result->row = row;
        int size = col * row;
        result->value = (float *)malloc(row * col * sizeof(float));
        for (int i = 0; i < size; i++)
        {
            if (matrix1->value[i] + matrix2->value[i] > FLT_MAX
            || matrix1->value[i] + matrix2->value[i] < -FLT_MAX)
            {
                printf("the result is out of range!\n");
                free(result->value);
                free(result);
                return NULL;
            }
            else
            {
                result->value[i] = matrix1->value[i] + matrix2->value[i];
            }
        }
        return result;
    }
}

bool sameSize(const Matrix *matrix1, const Matrix *matrix2)
{
    if ((matrix1->col == matrix2->col
    && matrix1->row == matrix2->row))
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

矩阵减法

思路大致与矩阵加法相同。

```
Matrix *subtractMatrix(const Matrix *matrix1, const Matrix *matrix2)
{
    if (!(legalSize(matrix1->row, matrix1->col)
    && legalSize(matrix2->row, matrix2->col)))
    {
        printf("this matrix is empty!\n");
        return NULL;
    }
    if (!sameSize(matrix1, matrix2))
    {
        printf("two matrices have different size which can not be subtracted!\n");
        return NULL;
    }
    else
    {
        Matrix *result = (Matrix *)malloc(sizeof(Matrix));
        int row = matrix1->row;
        int col = matrix1->col;
        result->col = col;
        result->row = row;
        int size = col * row;
        result->value = (float *)malloc(row * col * sizeof(float));
        for (int i = 0; i < size; i++)
        {
            if (matrix1->value[i] - matrix2->value[i] > FLT_MAX
            || matrix1->value[i] - matrix2->value[i] < -FLT_MAX)
            {
                printf("the result is out of range!\n");
                free(result->value);
                free(result);
                return NULL;
            }
            else
            {
                result->value[i] = matrix1->value[i] - matrix2->value[i];
            }
        }
        return result;
    }
}
```

标量运算

写完矩阵加法减法之后发现两个方法大同小异（基本上完全一样），标量四则运算也具有相同情况，于是增加输入参数opr表示操作符，将三种运算用一个方法实现。
输入矩阵指针、标量和运算符，const修饰防止修改。

```
#define ADD 1
#define MINUS 2
#define MULTIPLY 3

Matrix *operateScalar(const Matrix *matrix, const float scalar, const int opr)
{
    if (!(legalSize(matrix->row, matrix->col)))
    {
        printf("this matrix is empty!\n");
        return NULL;
    }

    Matrix *result = (Matrix *)malloc(sizeof(Matrix));
    int row = matrix->row;
    int col = matrix->col;
    result->col = col;
    result->row = row;
    int size = col * row;
    result->value = (float *)malloc(row * col * sizeof(float));
    for (int i = 0; i < size; i++)
    {
        if (((opr == ADD) && (matrix->value[i] + scalar > FLT_MAX
        || matrix->value[i] + scalar < -FLT_MAX))
        || ((opr == MINUS) && (matrix->value[i] - scalar > FLT_MAX
        || matrix->value[i] - scalar < -FLT_MAX))
        || ((opr == MULTIPLY) && (matrix->value[i] * scalar > FLT_MAX
        || matrix->value[i] * scalar < -FLT_MAX)))
        {
            printf("the result is out of range!\n");
            free(result->value);
            free(result);
            return NULL;
        }
        else
        {
            if (opr == ADD)
            {
                result->value[i] = matrix->value[i] + scalar;
            }
            else if (opr == MINUS)
            {
                result->value[i] = matrix->value[i] - scalar;
            }
            else if (opr == MULTIPLY)
            {
                result->value[i] = matrix->value[i] * scalar;
            }
            else
            {
                printf("wrong operator!\n");
                free(result->value);
                free(result);
                return NULL;
            }
        }
    }
    return result;
}
```

矩阵乘法

输入两个矩阵指针，判断左矩阵列数和右矩阵行数是否一致，若合法则进行矩阵乘法运算，返回结果矩阵指针。

```
Matrix *multiplyMatrix(const Matrix *matrixLeft, const Matrix *matrixRight)
{
    if (matrixLeft->col != matrixRight->row)
    {
        printf("two matrices can not be multiplied due to illegal size!\n");
        return NULL;
    }
    else
    {
        Matrix *result = (Matrix *)malloc(sizeof(Matrix));
        int row = matrixLeft->row;
        int col = matrixRight->col;
        int mul = matrixLeft->col;
        result->col = col;
        result->row = row;
        result->value = (float *)malloc(row * col * sizeof(float));
        float t = 0;
        for (int i = 0; i < row; i++)
        {
            for (int j = 0; j < col; j++)
            {
                t = 0;
                for (int k = 0; k < mul; k++)
                {
                    if (t + matrixLeft->value[i * mul + k]
                    * matrixRight->value[k * col + j] > FLT_MAX
                    * || t + matrixLeft->value[i * mul + k]
                    * matrixRight->value[k * col + j] < -FLT_MAX)
                    {
                        printf("the result is out of range!\n");
                        free(result->value);
                        free(result);
                        return NULL;
                    }
                    else
                    {
                        t += matrixLeft->value[i * mul + k]
                        * matrixRight->value[k * col + j];
                    }
                }
                result->value[i * col + j] = t;
            }
        }
        return result;
    }
}
```

找出最大最小值

同样的，找最大值和最小值在函数实现上几乎完全相同，于是在输入是加入变量max，如果为true则说明要找最大值，false则最小值。

```
float extremeValue(const Matrix *matrix, bool max)
{
    if (matrix == NULL)
    {
        printf("this matrix has been deleted!\n");
        return 0;
    }
    int col = matrix->col;
    int row = matrix->row;
    if (row <= 0 || col <= 0)
    {
        printf("this matrix is empty!\n");
        return 0;
    }
    else
    {
        int size = col * row;
        float result = matrix->value[0];
        for (size_t i = 1; i < size; i++)
        {
            if (max)
            {
                if (matrix->value[i] > result)
                {
                    result = matrix->value[i];
                }
            }
            else
            {
                if (matrix->value[i] < result)
                {
                    result = matrix->value[i];
                }
            }
        }
        return result;
    }
}
```

矩阵转置

输入转置之前的矩阵指针，若参数合法则输出转置之后的结果矩阵指针。

```
Matrix *transpose(const Matrix *matrix)
{
    if (matrix == NULL)
    {
        printf("this matrix has been deleted!\n");
        return NULL;
    }
    int col = matrix->col;
    int row = matrix->row;
    if (row <= 0 || col <= 0)
    {
        printf("this matrix is empty!\n");
        return NULL;
    }
    else
    {
        Matrix *result = (Matrix *)malloc(sizeof(Matrix));
        result->col = row;
        result->row = col;
        result->value = (float *)malloc(row * col * sizeof(float));
        for (int i = 0; i < col; i++)
        {
            for (int j = 0; j < row; j++)
            {
                result->value[i * row + j] = matrix->value[j * col + i];
            }
        }
        return result;
    }
}
```

生成单位阵

输入边长，将对角线上元素赋值为1，其他元素为0，输出结果矩阵指针。

```
Matrix *identityMatrix(const int sideLength)
{
    //输入检查
    if (sideLength <= 0)
    {
        printf("your row and col are too small!\n");
        return NULL;
    }
    else
    {
        Matrix *matrix = (Matrix *)malloc(sizeof(Matrix));
        matrix->col = sideLength;
        matrix->row = sideLength;
        matrix->value = (float *)malloc(sideLength * sideLength * sizeof(float));
        for (int r = 0; r < sideLength; r++)
        {
            for (int c = 0; c < sideLength; c++)
            {
                if (r == c)
                {
                    matrix->value[r * sideLength + c] = 1;
                }
                else
                {
                    matrix->value[r * sideLength + c] = 0;
                }
            }
        }
        return matrix;
    }
}
```

修改指定位置

输入原矩阵、位置、新值，将指定位置的值修改为新值。返回是否修改成功。

```
bool set(Matrix *matrix, const int r, const int c, const float newValue)
{
    if (matrix == NULL)
    {
        printf("this matrix has been deleted!\n");
        return false;
    }
    if (matrix->col < c - 1 || matrix->row < r - 1)
    {
        printf("this place is illegal!\n");
        return false;
    }
    matrix->value[(r - 1) * matrix->col + (c - 1)] = newValue;
    return true;
}
```

生成全1矩阵和全0矩阵

输入矩阵大小，输出全0或者全1的矩阵指针。

```
Matrix *ones(const int row, const int col)
{
    if (row <= 0 || col <= 0)
    {
        printf("your row and col are too small!\n");
        return NULL;
    }
    Matrix *result = (Matrix *)malloc(sizeof(Matrix));
    result->col = col;
    result->row = row;
    int size = col * row;
    result->value = (float *)malloc(row * col * sizeof(float));
    for (int i = 0; i < size; i++)
    {
        result->value[i] = 1;
    }
    return result;
}

Matrix *zeros(const int row, const int col)
{
    if (row <= 0 || col <= 0)
    {
        printf("your row and col are too small!\n");
        return NULL;
    }
    Matrix *result = (Matrix *)malloc(sizeof(Matrix));
    result->col = col;
    result->row = row;
    int size = col * row;
    result->value = (float *)malloc(row * col * sizeof(float));
    for (int i = 0; i < size; i++)
    {
        result->value[i] = 0;
    }
    return result;
}
```

打印自定义精度矩阵

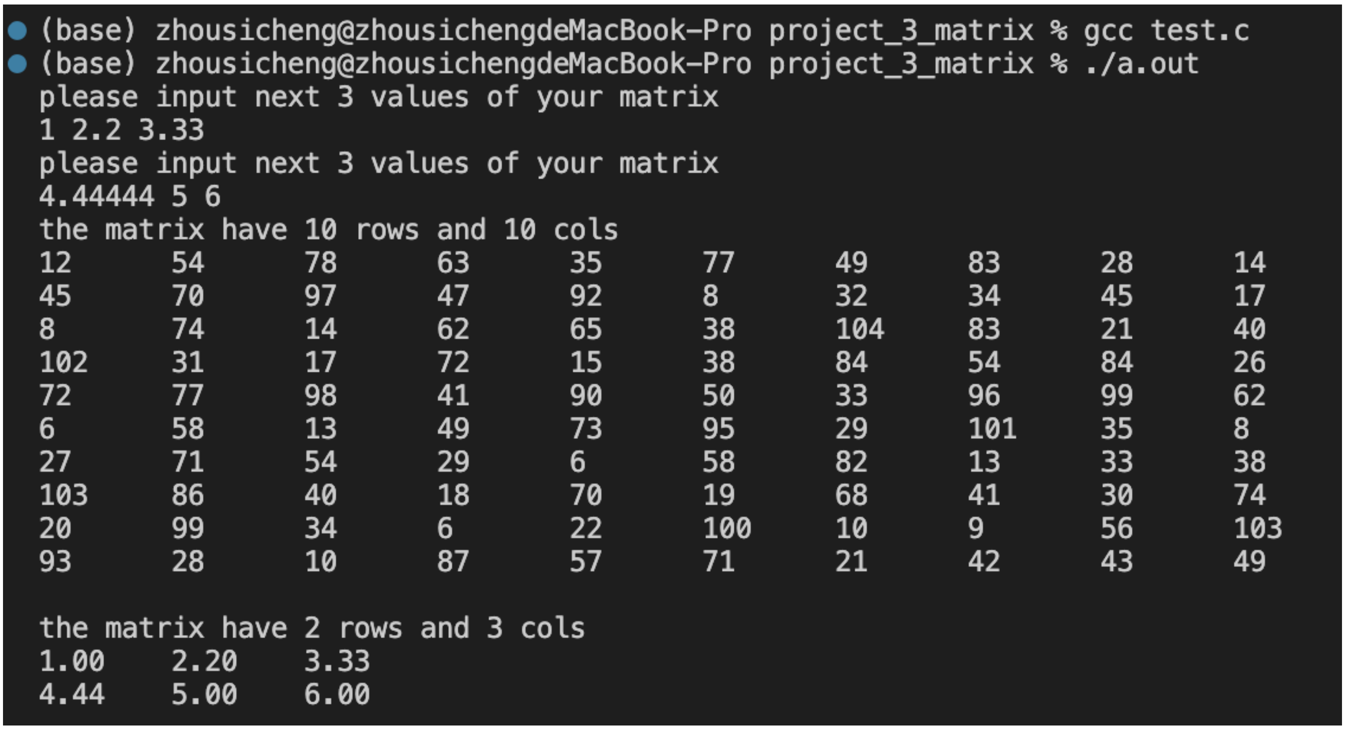
输入矩阵指针和自定义精度，打印矩阵相关信息。

```
void printMatrix(const Matrix *matrix, int precision)
{
    if (matrix == NULL)
    {
        printf("this matrix has been deleted!\n");
        return;
    }
    if (precision < 0 || precision > 10)
    {
        printf("the precision is out of range and is automatically set to 2\n");
        precision = 2;
    }

    int col = matrix->col;
    int row = matrix->row;
    printf("the matrix have %d rows and %d cols\n", row, col);
    if (!legalSize(row, col))
    {
        printf("this matrix is empty!\n");
    }
    else
    {
        for (int r = 0; r < row; r++)
        {
            for (int c = 0; c < col; c++)
            {
                printf("%.*lf\t", precision, matrix->value[r * col + c]);
            }
            printf("\n");
        }
    }
}
```

Part 03 - Result & Verifification

create a matrix (and print it out)

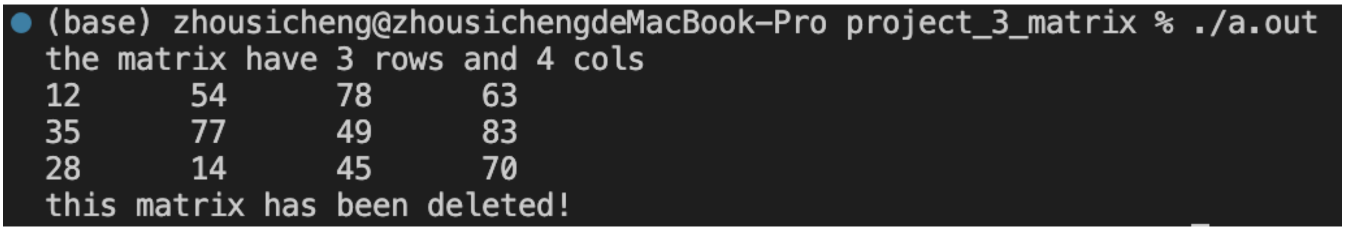
```
        Matrix *matrix1 = createMatrix(10, 10, 1); //随机生成
    Matrix *matrix2 = createMatrix(2, 3, 0); //需要输入
    printMatrix(matrix1, 0); //保留整数
    printf("\n");
    printMatrix(matrix2, 2); //保留两位小数
    printf("\n");
```



```
(base) zhousicheng@zhousichengdeMacBook-Pro project_3_matrix % gcc test.c
(base) zhousicheng@zhousichengdeMacBook-Pro project_3_matrix % ./a.out
please input next 3 values of your matrix
1 2.2 3.33
please input next 3 values of your matrix
4.44444 5 6
the matrix have 10 rows and 10 cols
12      54      78      63      35      77      49      83      28      14
45      70      97      47      92      8       32      34      45      17
8       74      14      62      65      38      104     83      21      40
102     31      17      72      15      38      84      54      84      26
72      77      98      41      90      50      33      96      99      62
6       58      13      49      73      95      29      101     35      8
27      71      54      29      6       58      82      13      33      38
103     86      40      18      70      19      68      41      30      74
20      99      34      6       22      100     10      9       56      103
93      28      10      87      57      71      21      42      43      49

the matrix have 2 rows and 3 cols
1.00    2.20    3.33
4.44    5.00    6.00
```

delete a matrix

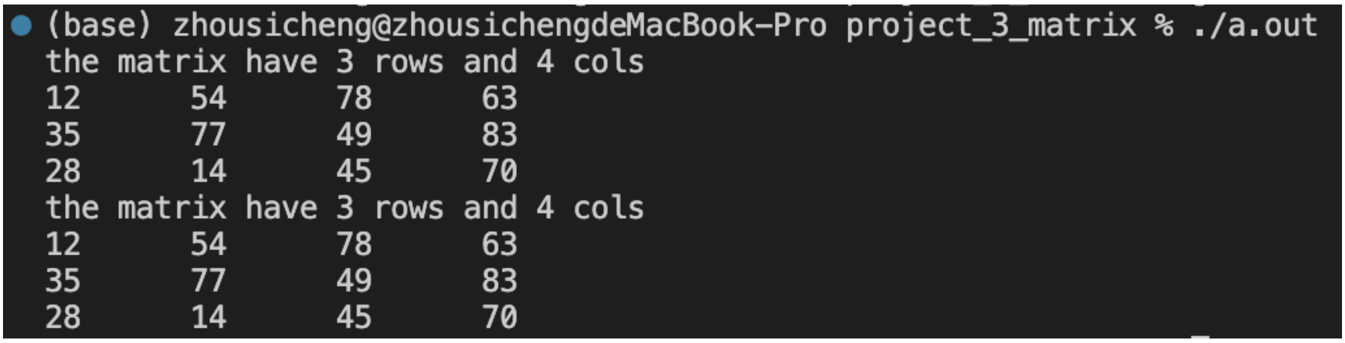
```
        Matrix *matrix = createMatrix(3, 4, 1);
    printMatrix(matrix, 0);
    delete (matrix);
    printMatrix(matrix, 0);
```



```
(base) zhousicheng@zhousichengdeMacBook-Pro project_3_matrix % ./a.out
the matrix have 3 rows and 4 cols
12      54      78      63
35      77      49      83
28      14      45      70
this matrix has been deleted!
```

copy matrix

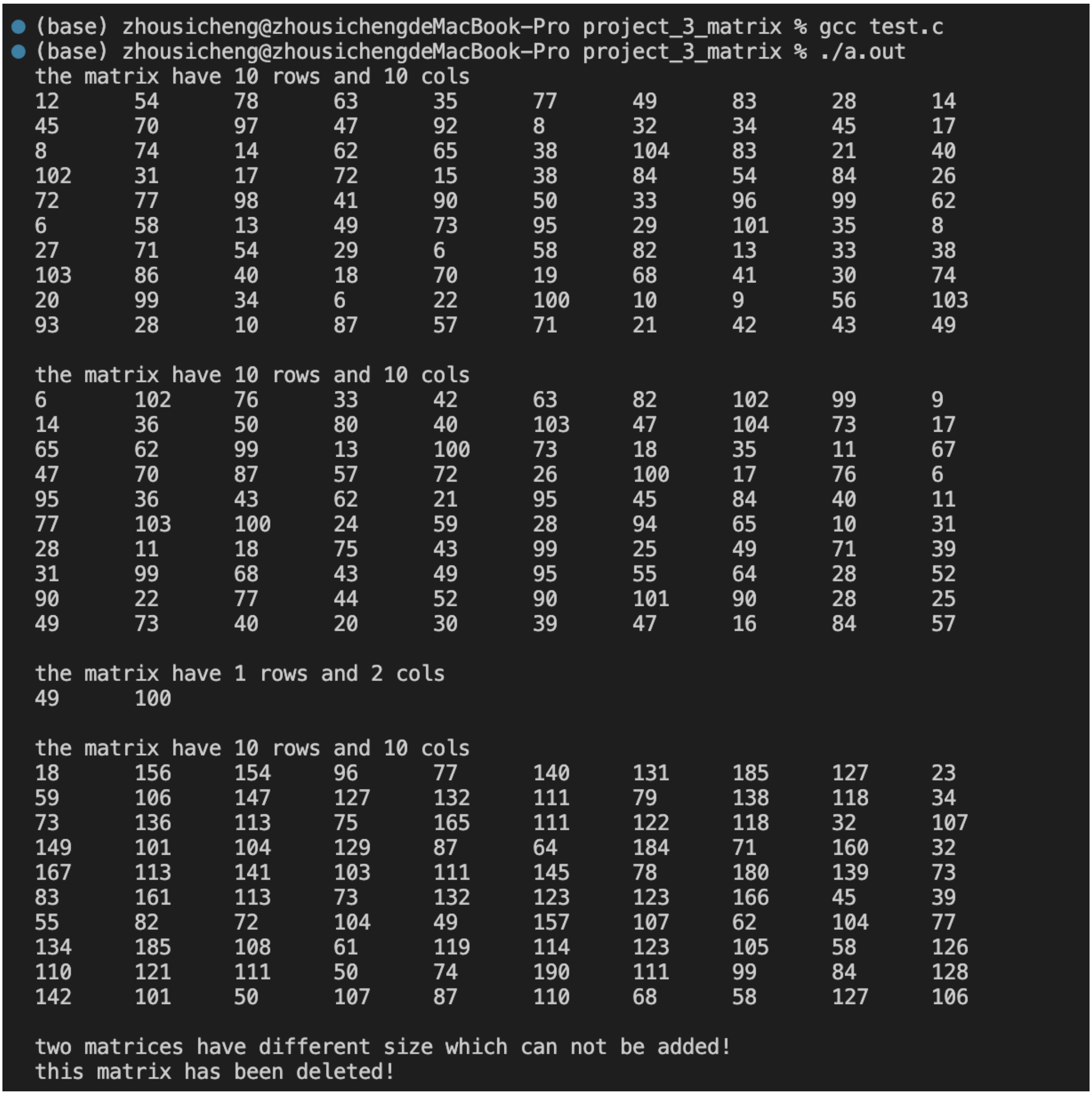
```
    Matrix *matrix = createMatrix(3, 4, 1);
    printMatrix(matrix, 0);
    Matrix *matrix2 = copy(matrix);
    printMatrix(matrix2, 0);
```



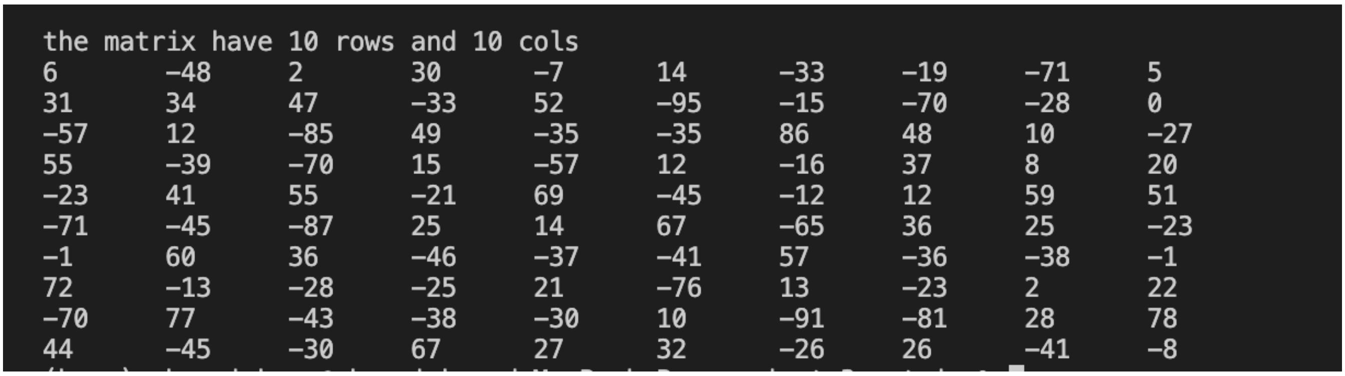
```
(base) zhousicheng@zhousichengdeMacBook-Pro project_3_matrix % ./a.out
the matrix have 3 rows and 4 cols
12      54      78      63
35      77      49      83
28      14      45      70
the matrix have 3 rows and 4 cols
12      54      78      63
35      77      49      83
28      14      45      70
```

add and subtract two matrices

```
        Matrix *matrix1 = createMatrix(10, 10, 1);
    Matrix *matrix2 = createMatrix(10, 10, 1);
    Matrix *matrix3 = createMatrix(1, 2, 1);
    printMatrix(matrix1, 0);
    printf("\n");
    printMatrix(matrix2, 0);
    printf("\n");
    printMatrix(matrix3, 0);
    printf("\n");
    Matrix *addresult1 = addMatrix(matrix1, matrix2);
    printMatrix(addresult1, 0);
    printf("\n");
    Matrix *addresult2 = addMatrix(matrix1, matrix3);
    printMatrix(addresult2, 0);
    printf("\n");
    Matrix *minusresult = subtractMatrix(matrix1, matrix2);
    printMatrix(minusresult, 0);
```



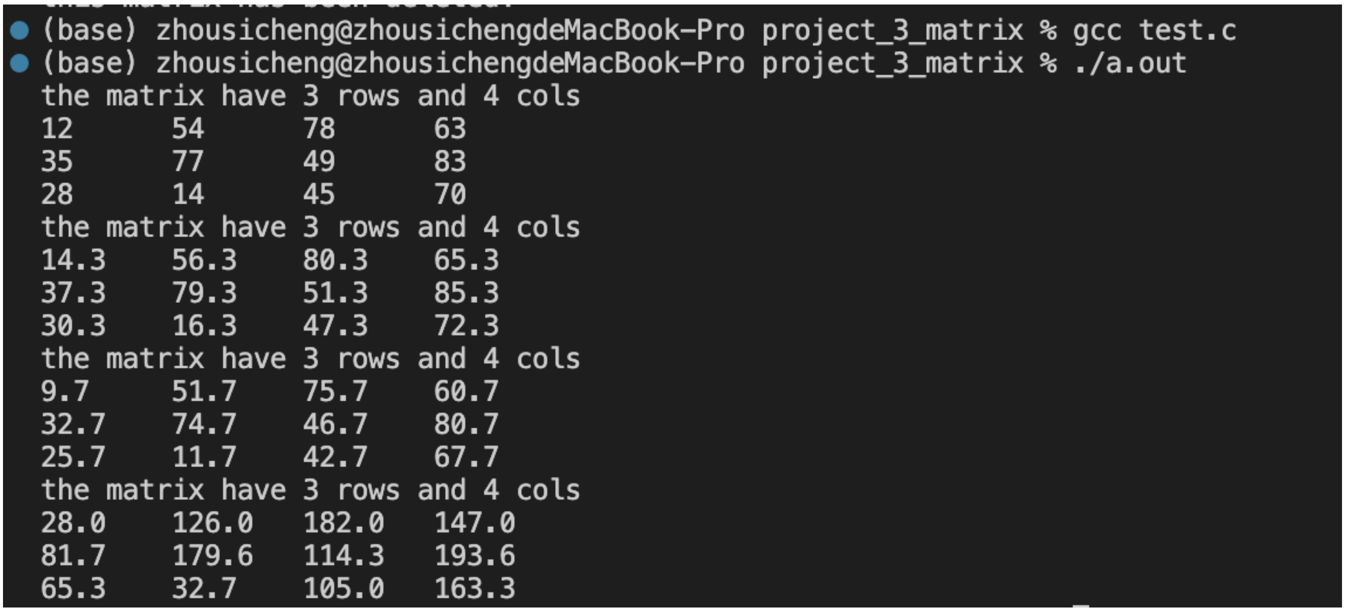
```
● (base) zhousicheng@zhousichengdeMacBook-Pro project_3_matrix % gcc test.c
● (base) zhousicheng@zhousichengdeMacBook-Pro project_3_matrix % ./a.out
  the matrix have 10 rows and 10 cols
  12      54      78      63      35      77      49      83      28      14
  45      70      97      47      92      8       32      34      45      17
  8       74      14      62      65      38      104     83      21      40
  102     31      17      72      15      38      84      54      84      26
  72      77      98      41      90      50      33      96      99      62
  6       58      13      49      73      95      29      101     35      8
  27      71      54      29      6       58      82      13      33      38
  103     86      40      18      70      19      68      41      30      74
  20      99      34      6       22      100     10      9       56      103
  93      28      10      87      57      71      21      42      43      49

  the matrix have 10 rows and 10 cols
  6       102     76      33      42      63      82      102     99      9
  14      36      50      80      40      103     47      104     73      17
  65      62      99      13      100     73      18      35      11      67
  47      70      87      57      72      26      100     17      76      6
  95      36      43      62      21      95      45      84      40      11
  77      103     100     24      59      28      94      65      10      31
  28      11      18      75      43      99      25      49      71      39
  31      99      68      43      49      95      55      64      28      52
  90      22      77      44      52      90      101     90      28      25
  49      73      40      20      30      39      47      16      84      57

  the matrix have 1 rows and 2 cols
  49      100

  the matrix have 10 rows and 10 cols
  18      156     154     96      77      140     131     185     127     23
  59      106     147     127     132     111     79      138     118     34
  73      136     113     75      165     111     122     118     32      107
  149     101     104     129     87      64      184     71      160     32
  167     113     141     103     111     145     78      180     139     73
  83      161     113     73      132     123     123     166     45      39
  55      82      72      104     49      157     107     62      104     77
  134     185     108     61      119     114     123     105     58      126
  110     121     111     50      74      190     111     99      84      128
  142     101     50      107     87      110     68      58      127     106

  two matrices have different size which can not be added!
  this matrix has been deleted!
```



```
  the matrix have 10 rows and 10 cols
  6       -48     2       30      -7      14      -33     -19     -71     5
  31      34      47      -33     52      -95     -15     -70     -28     0
  -57     12      -85     49      -35     -35     86      48      10      -27
  55      -39     -70     15      -57     12      -16     37      8       20
  -23     41      55      -21     69      -45     -12     12      59      51
  -71     -45     -87     25      14      67      -65     36      25      -23
  -1      60      36      -46     -37     -41     57      -36     -38     -1
  72      -13     -28     -25     21      -76     13      -23     2       22
  -70     77      -43     -38     -30     10      -91     -81     28      78
  44      -45     -30     67      27      32      -26     26      -41     -8
```

scalar calculation

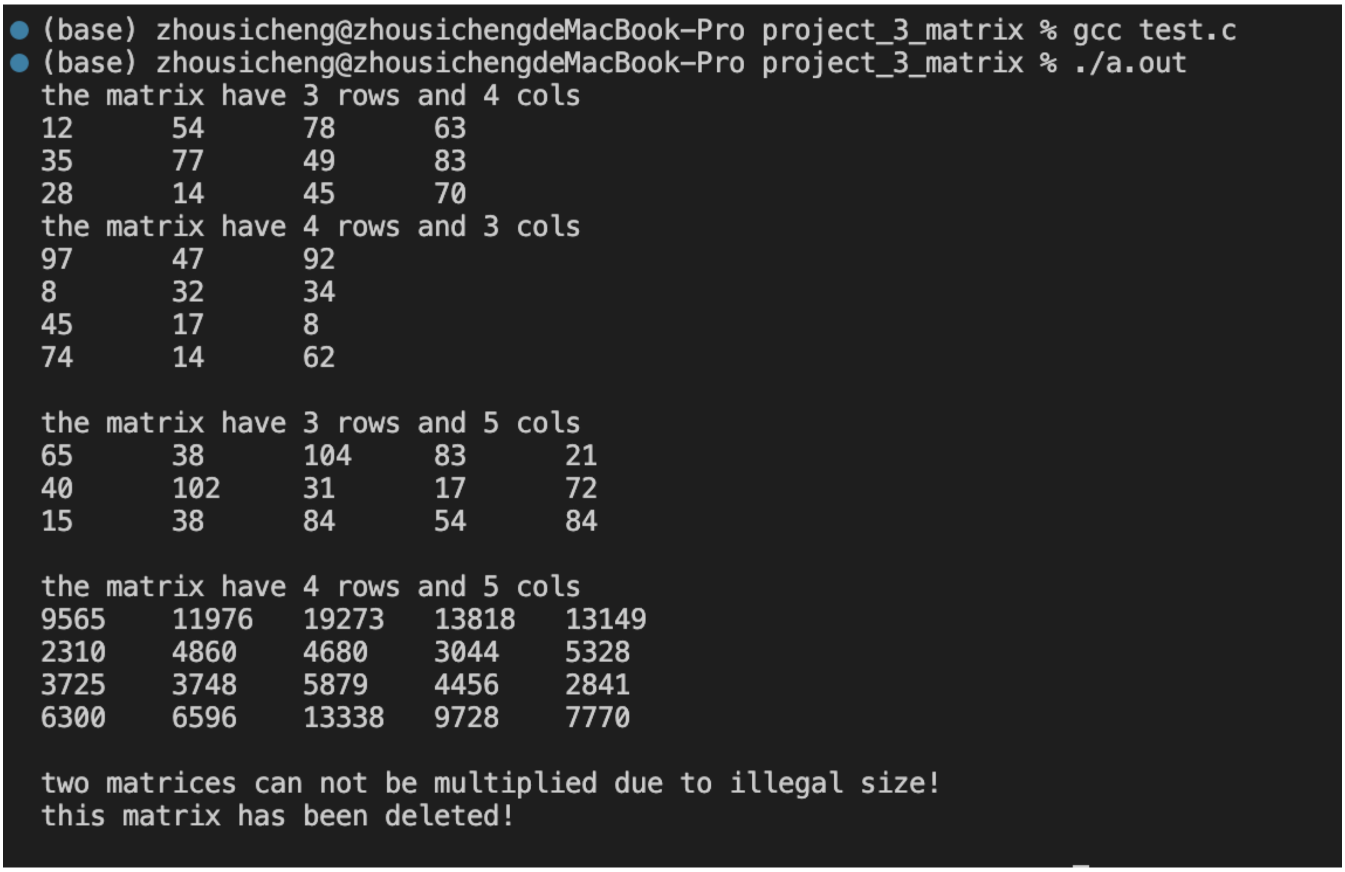
```
        Matrix *matrix = createMatrix(3, 4, 1);
    printMatrix(matrix, 0);
    float scalar = 2.333;
    Matrix *addresult = operateScalar(matrix, scalar, ADD);
    printMatrix(addresult,1);
    Matrix *minusresult = operateScalar(matrix, scalar, MINUS);
    printMatrix(minusresult,1);
    Matrix *mulresult = operateScalar(matrix, scalar, MULTIPLY);
    printMatrix(mulresult,1);
```



```
(base) zhousicheng@zhousichengdeMacBook-Pro project_3_matrix % gcc test.c
(base) zhousicheng@zhousichengdeMacBook-Pro project_3_matrix % ./a.out
the matrix have 3 rows and 4 cols
12      54      78      63
35      77      49      83
28      14      45      70
the matrix have 3 rows and 4 cols
14.3    56.3    80.3    65.3
37.3    79.3    51.3    85.3
30.3    16.3    47.3    72.3
the matrix have 3 rows and 4 cols
9.7     51.7    75.7    60.7
32.7    74.7    46.7    80.7
25.7    11.7    42.7    67.7
the matrix have 3 rows and 4 cols
28.0    126.0   182.0   147.0
81.7    179.6   114.3   193.6
65.3    32.7    105.0   163.3
```

matrix multiplication

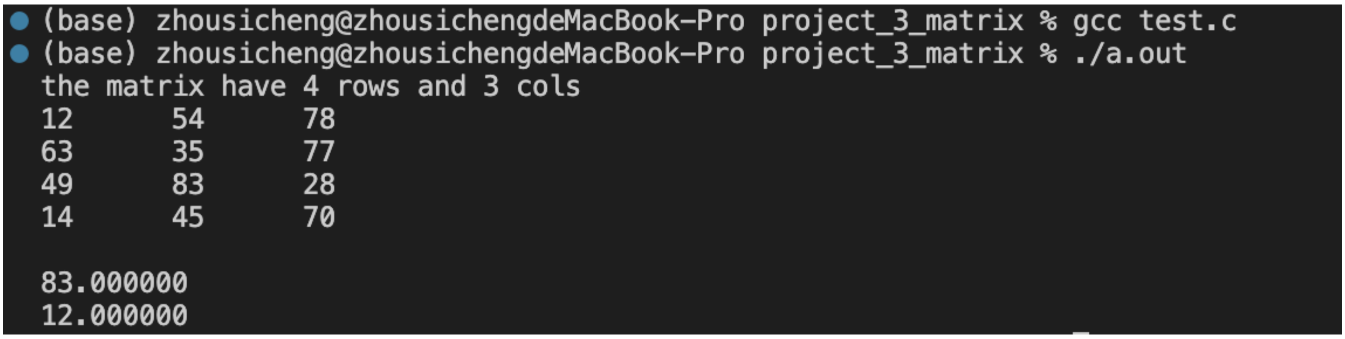
```
    Matrix *matrix1 = createMatrix(4, 3, 1);
    Matrix *matrix2 = createMatrix(3, 5, 1);
    printMatrix(matrix1,0);
    printf("\n");
    printMatrix(matrix2,0);
    printf("\n");
    printMatrix(multiplyMatrix(matrix1, matrix2),0);
    printf("\n");
    printMatrix(multiplyMatrix(matrix2, matrix1),0);
    printf("\n");
```



```
(base) zhousicheng@zhousichengdeMacBook-Pro project_3_matrix % gcc test.c
(base) zhousicheng@zhousichengdeMacBook-Pro project_3_matrix % ./a.out
the matrix have 3 rows and 4 cols
12      54      78      63
35      77      49      83
28      14      45      70
the matrix have 4 rows and 3 cols
97      47      92
8       32      34
45      17      8
74      14      62

the matrix have 3 rows and 5 cols
65      38      104     83      21
40      102     31      17      72
15      38      84      54      84

the matrix have 4 rows and 5 cols
9565    11976   19273   13818   13149
2310    4860    4680    3044    5328
3725    3748    5879    4456    2841
6300    6596    13338   9728    7770

two matrices can not be multiplied due to illegal size!
this matrix has been deleted!
```

find extreme value

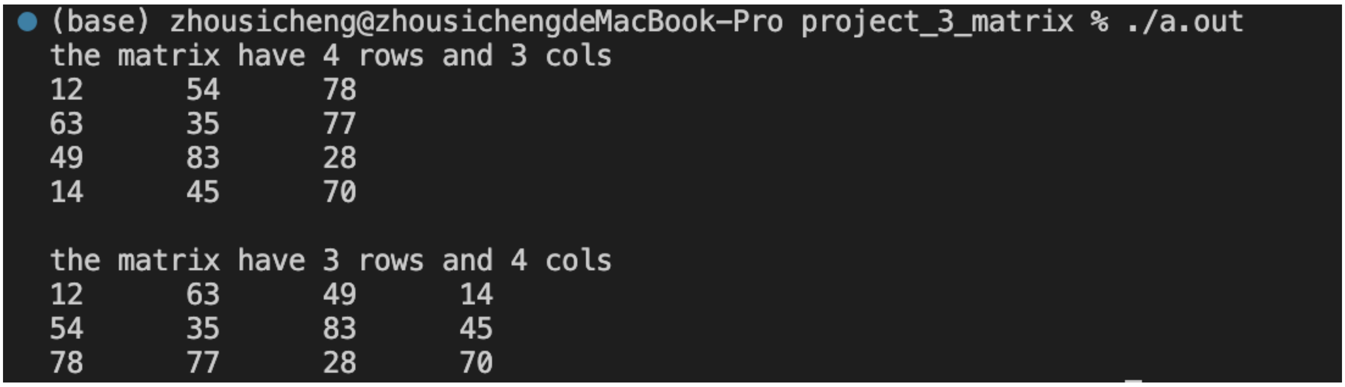
```
    printf("%f\n", extremeValue(matrix1, 1));
    printf("%f\n", extremeValue(matrix1, 0));
```



```
● (base) zhousicheng@zhousichengdeMacBook-Pro project_3_matrix % gcc test.c
● (base) zhousicheng@zhousichengdeMacBook-Pro project_3_matrix % ./a.out
  the matrix have 4 rows and 3 cols
  12      54      78
  63      35      77
  49      83      28
  14      45      70

  83.000000
  12.000000
```

transpose

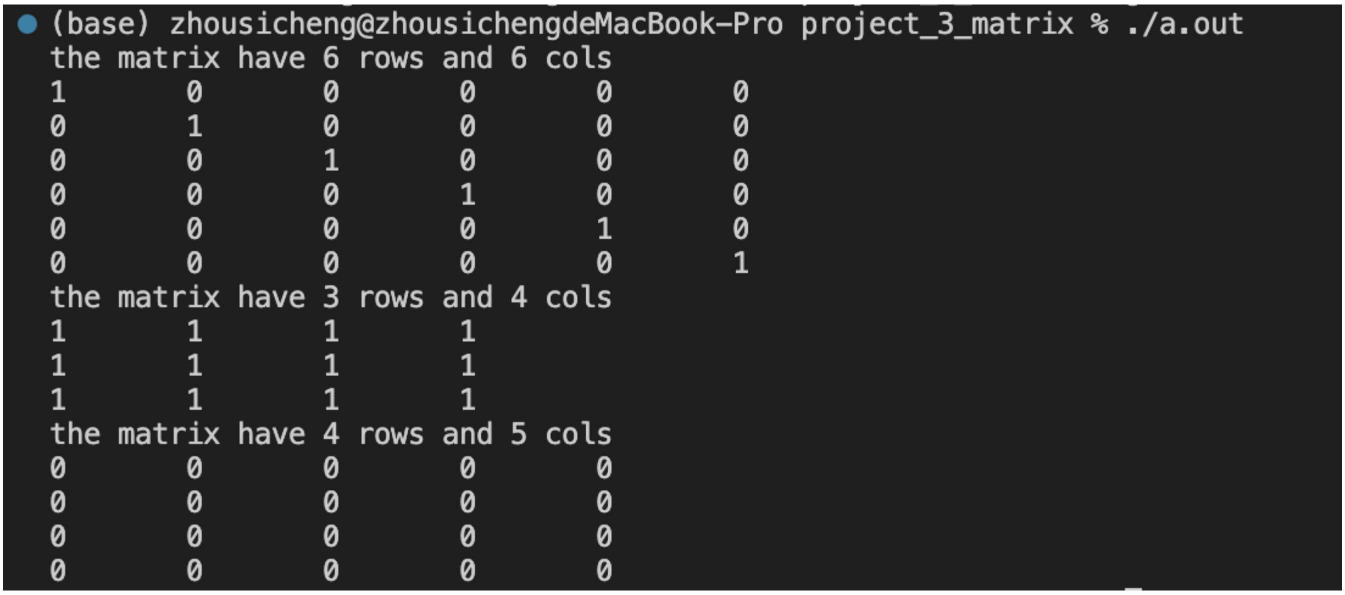
```
printMatrix(transpose(matrix1), 0);
```



```
● (base) zhousicheng@zhousichengdeMacBook-Pro project_3_matrix % ./a.out
  the matrix have 4 rows and 3 cols
  12      54      78
  63      35      77
  49      83      28
  14      45      70

  the matrix have 3 rows and 4 cols
  12      63      49      14
  54      35      83      45
  78      77      28      70
```

generate special matrices

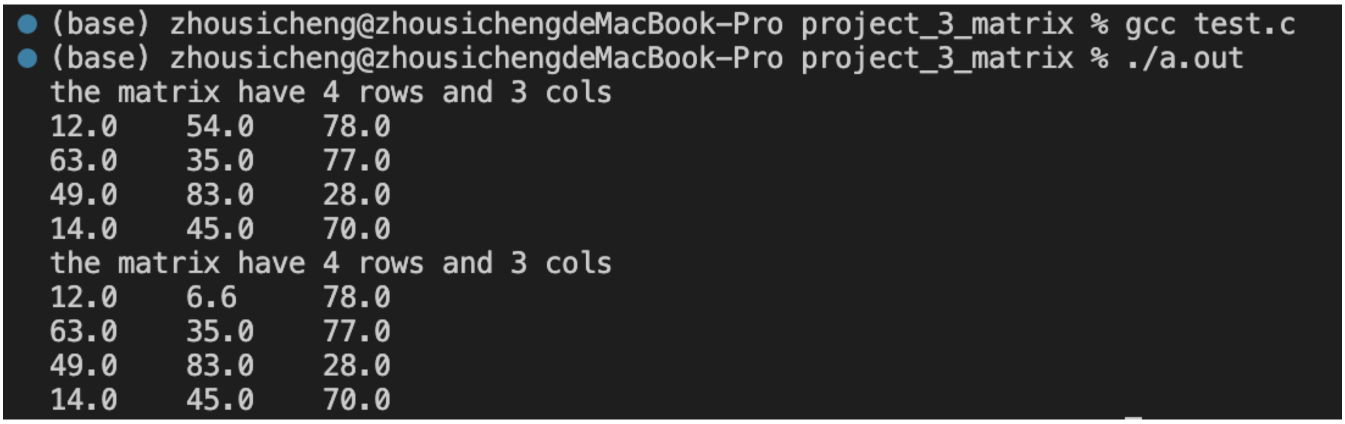
```
    printMatrix(identityMatrix(6), 0);
    printMatrix(ones(3, 4), 0);
    printMatrix(zeros(4, 5), 0);
```



```
● (base) zhousicheng@zhousichengdeMacBook-Pro project_3_matrix % ./a.out
  the matrix have 6 rows and 6 cols
  1       0       0       0       0       0
  0       1       0       0       0       0
  0       0       1       0       0       0
  0       0       0       1       0       0
  0       0       0       0       1       0
  0       0       0       0       0       1
  the matrix have 3 rows and 4 cols
  1       1       1       1
  1       1       1       1
  1       1       1       1
  the matrix have 4 rows and 5 cols
  0       0       0       0       0
  0       0       0       0       0
  0       0       0       0       0
  0       0       0       0       0
```

set value

```
    Matrix *matrix2 = createMatrix(4, 3, 1);
    printMatrix(matrix2, 1);
    set(matrix2, 1, 2, 6.6);
    printMatrix(matrix2, 1);
```



```
(base) zhousicheng@zhousichengdeMacBook-Pro project_3_matrix % gcc test.c
(base) zhousicheng@zhousichengdeMacBook-Pro project_3_matrix % ./a.out
 the matrix have 4 rows and 3 cols
 12.0    54.0    78.0
 63.0    35.0    77.0
 49.0    83.0    28.0
 14.0    45.0    70.0
 the matrix have 4 rows and 3 cols
 12.0    6.6     78.0
 63.0    35.0    77.0
 49.0    83.0    28.0
 14.0    45.0    70.0
```

Part 04 - Difficulties & Solutions

把释放内存的矩阵指针置NULL

写deleteMatrix方法的时候，在单步调试中发现，如果在函数内写matrix=NULL，函数外部的该指针并不会变NULL，这是因为传入函数的指针是地址的复制值。采用了宏之后解决了这个问题。以及，写宏的时候要注意不要在奇怪的地方加空格。

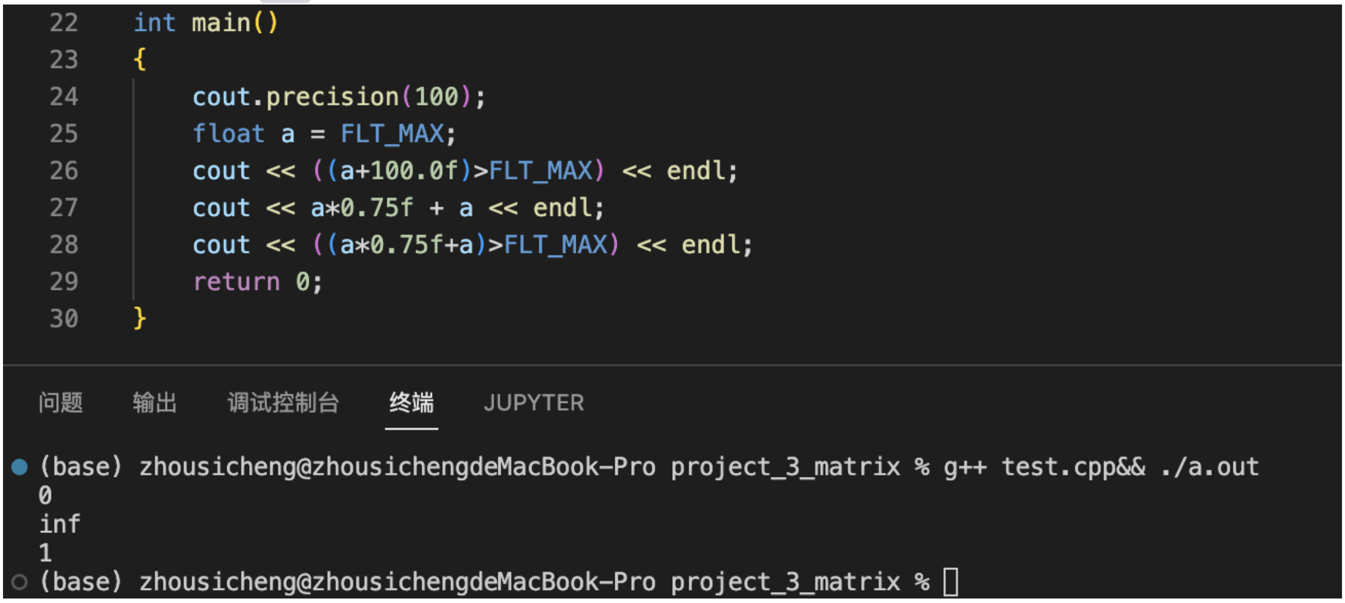
超出范围

原本在四则运算中进行了一个超出范围的判断，如下。

```
if (matrix1->value[i] - matrix2->value[i] > FLT_MAX
        || matrix1->value[i] - matrix2->value[i] < FLT_MIN)
{
    printf("the result is out of range!\n");
    free(result->value);
    free(result);
    return NULL;
}
else
{
result->value[i] = matrix1->value[i] - matrix2->value[i];
}
```

这样的做法是误以为FLT_MIN表示的是最小的浮点数，实际上只是最小的正浮点数，所以一旦结果为负就会打印“超出范围”，这显然是不合理的。所以把`FLT_MIN`改成了`-FLT_MAX`。实际上单精度浮点数能精确计算的数据范围非常有限，如下图所示，`FLT_MAX+100`是不会大于`FLT_MAX`的，而`1.75*a`之所以能大于`FLT_MAX`是因为已经变成了inf。如果要精准判断是否超出数据范围的话，使用float显然是不合适的，那不如再回project2写一个高精度科学计算器。所以此处做出

一些牺牲，只在离谱的`inf`时打印超出范围。



```
int main()
{
    cout.precision(100);
    float a = FLT_MAX;
    cout << ((a+100.0f)>FLT_MAX) << endl;
    cout << a*0.75f + a << endl;
    cout << ((a*0.75f+a)>FLT_MAX) << endl;
    return 0;
}
```

```
(base) zhousicheng@zhousichengdeMacBook-Pro project_3_matrix % g++ test.cpp&& ./a.out
0
inf
1
(base) zhousicheng@zhousichengdeMacBook-Pro project_3_matrix %
```

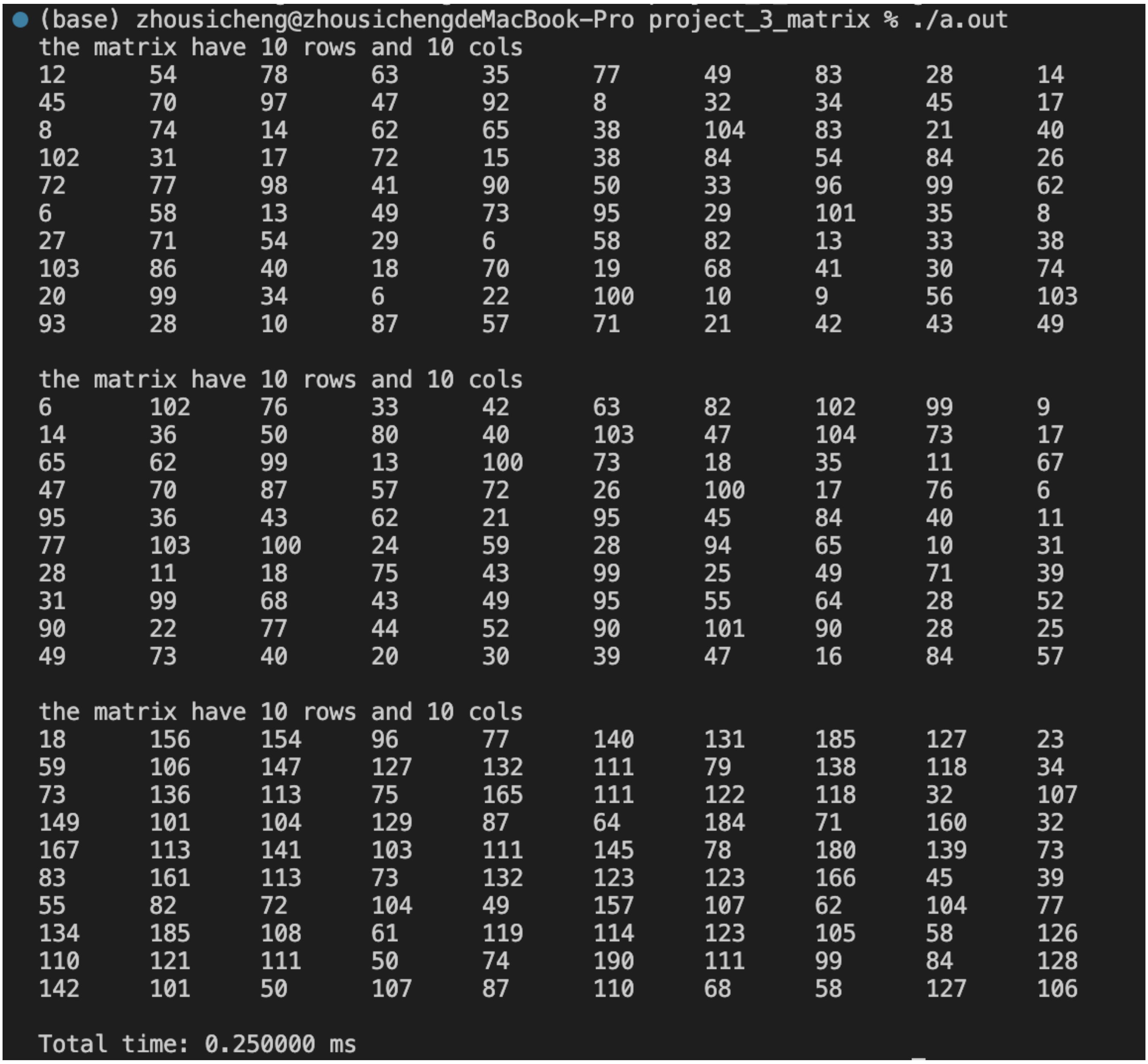
封装方法还是复制粘贴?

以矩阵加法为例，每次运算之前需要判断输入指针是否为空、矩阵行列数是否大于零、大小是否一致。同样，在矩阵减法和标量运算的时候也需要进行一通类似的判断，这就带来了大量重复的代码。按照一般的习惯，这种重复的判断内容应该封装成方法，而不是多次对相同内容复制粘贴，但是考虑到这些方法中的逻辑内容十分简单，而调用方法可能带来复制、压栈等等之后效率的降低。
总的来说，进行这些合法判断的方法有三种，一是手动复制粘贴，二是封装`legal`方法，三是定义宏。考虑到宏的本质也就是复制粘贴，并且宏比较容易出错，此处只比较前两种的效率。
以矩阵加法（10*10）为例。
测试代码如下。

```c
#include "matrix.c"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <float.h>
#include <time.h>
clock_t start, end;

int main()
{
    start = clock();

    Matrix *matrix1 = createMatrix(10, 10, 1);
    Matrix *matrix2 = createMatrix(10, 10, 1);
    printMatrix(matrix1, 0);
    printf("\n");
    printMatrix(matrix2, 0);
    printf("\n");
    Matrix *addresult = addMatrix(matrix1, matrix2);
    printMatrix(addresult, 0);
    printf("\n");

    end = clock(); //程序结束用时
    double endtime = (double)(end - start) / CLOCKS_PER_SEC;
    printf("Total time: %f ms\n", endtime * 1000); // ms为单位

    return 0;
}
```

```c
bool sameSize(const Matrix *matrix1, const Matrix *matrix2)
{
    if ((matrix1->col == matrix2->col && matrix1->row == matrix2->row))
    {
        return true;
    }
    else
    {
        return false;
    }
}

bool legalSize(int row, int col)
{
    if (row <= 0 || col <= 0)
    {
        return false;
    }
    else
    {
        return true;
    }
}
Matrix *addMatrix(const Matrix *matrix1, const Matrix *matrix2)
{
    // if (!(legalSize(matrix1->row, matrix1->col)
    // && legalSize(matrix2->row, matrix2->col)))
    // {
    //     printf("this matrix is empty!\n");
    //     return NULL;
    // }
    // if (!sameSize(matrix1, matrix2))
    // {
    //     printf("two matrices have different size which can not be added!\n");
    //     return NULL;
    // }
    if (!(matrix1->col == matrix2->col && matrix1->row == matrix2->row))
    {
        printf("two matrices have different size which can not be added!\n");
        return NULL;
    }
    if (matrix1->col <= 0 || matrix1->row <= 0 || matrix2->col < 0 || matrix2->row <= 0)
    {
        printf("this matrix is empty!\n");
        return NULL;
    }
    else
    {
        Matrix *result = (Matrix *)malloc(sizeof(Matrix));
        int row = matrix1->row;
        int col = matrix1->col;
        result->col = col;
        result->row = row;
        int size = col * row;
        result->value = (float *)malloc(row * col * sizeof(float));
        for (int i = 0; i < size; i++)
        {
            if (matrix1->value[i] + matrix2->value[i] > FLT_MAX
            || matrix1->value[i] + matrix2->value[i] < -FLT_MAX)
            {
                printf("the result is out of range!\n");
                free(result->value);
                free(result);
                return NULL;
            }
            else
            {
                result->value[i] = matrix1->value[i] + matrix2->value[i];
            }
        }
        return result;
    }
}
```

输出结果类似下图。



```
● (base) zhousicheng@zhousichengdeMacBook-Pro project_3_matrix % ./a.out
  the matrix have 10 rows and 10 cols
  12      54      78      63      35      77      49      83      28      14
  45      70      97      47      92      8       32      34      45      17
  8       74      14      62      65      38      104     83      21      40
  102     31      17      72      15      38      84      54      84      26
  72      77      98      41      90      50      33      96      99      62
  6       58      13      49      73      95      29      101     35      8
  27      71      54      29      6       58      82      13      33      38
  103     86      40      18      70      19      68      41      30      74
  20      99      34      6       22      100     10      9       56      103
  93      28      10      87      57      71      21      42      43      49

  the matrix have 10 rows and 10 cols
  6       102     76      33      42      63      82      102     99      9
  14      36      50      80      40      103     47      104     73      17
  65      62      99      13      100     73      18      35      11      67
  47      70      87      57      72      26      100     17      76      6
  95      36      43      62      21      95      45      84      40      11
  77      103     100     24      59      28      94      65      10      31
  28      11      18      75      43      99      25      49      71      39
  31      99      68      43      49      95      55      64      28      52
  90      22      77      44      52      90      101     90      28      25
  49      73      40      20      30      39      47      16      84      57

  the matrix have 10 rows and 10 cols
  18      156     154     96      77      140     131     185     127     23
  59      106     147     127     132     111     79      138     118     34
  73      136     113     75      165     111     122     118     32      107
  149     101     104     129     87      64      184     71      160     32
  167     113     141     103     111     145     78      180     139     73
  83      161     113     73      132     123     123     166     45      39
  55      82      72      104     49      157     107     62      104     77
  134     185     108     61      119     114     123     105     58      126
  110     121     111     50      74      190     111     99      84      128
  142     101     50      107     87      110     68      58      127     106

  Total time: 0.250000 ms
```

统计结果如下。

方式	第一次	第二次	第三次	平均
函数	0.250000 ms	0.223000 ms	0.293000 ms	0.255333ms
复制粘贴	0.215000 ms	0.223000 ms	0.220000 ms	0.219333ms

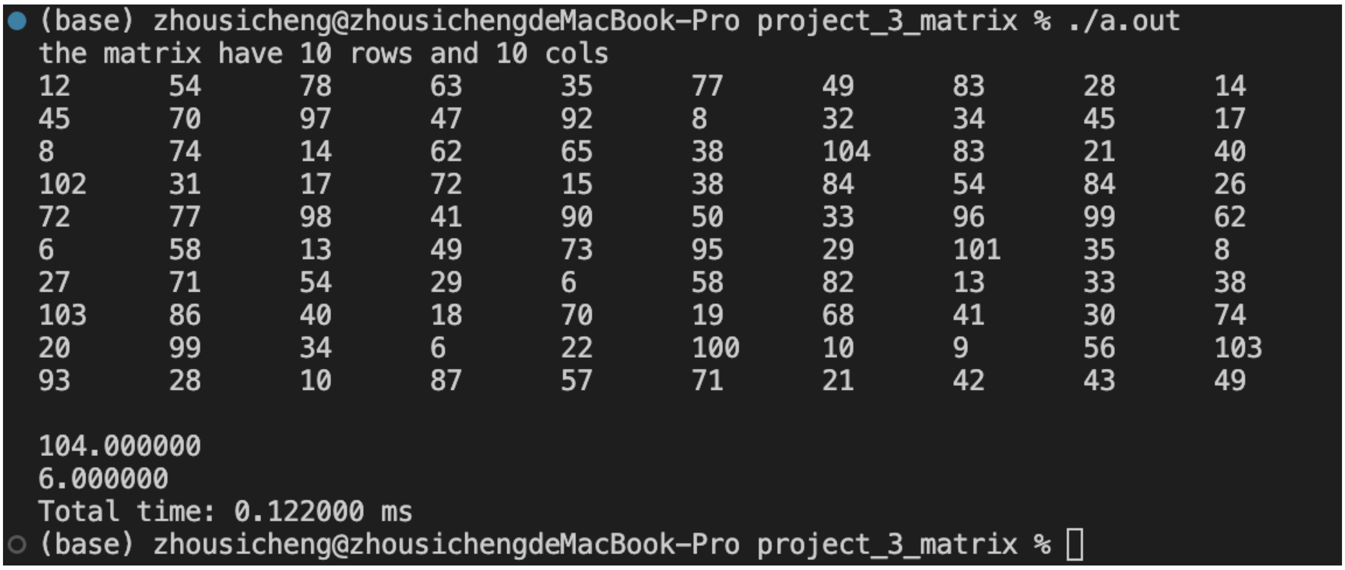
从统计结果看，复制粘贴大概比调用函数提高了10%的效率，这还是比较可观的。但实际上，运算的主要耗时并不在检验合法性上，所以在此处对效率的要求并没有那么高。由于封装函数具有易于维护的优点，故在本library中仍旧采用调用函数来检验输入的合法性。

两层循环还是一层循环?

创建矩阵和矩阵标量运算遍历矩阵元素时中都会遇到上述问题。于是尝试比较两种方法实现起来的效率。
测试代码如下。

```
float extremeValue(const Matrix *matrix, bool max)
{
    if (matrix == NULL)
    {
        printf("this matrix has been deleted!\n");
        return 0;
    }
    int col = matrix->col;
    int row = matrix->row;
    if (!legalSize(row, col))
    {
        printf("this matrix is empty!\n");
        return 0;
    }
    else
    {
        int size = col * row;
        float result = matrix->value[0];
        // for (size_t i = 1; i < size; i++)
        // {
        //     if (max)
        //     {
        //         if (matrix->value[i] > result)
        //         {
        //             result = matrix->value[i];
        //         }
        //     }
        //     else
        //     {
        //         if (matrix->value[i] < result)
        //         {
        //             result = matrix->value[i];
        //         }
        //     }
        // }

        for (int r = 0; r < row; r++)
        {
            for (int c = 0; c < col; c++)
            {
                if (max)
                {
                    if (matrix->value[r * col + c] > result)
                    {
                        result = matrix->value[r * col + c];
                    }
                }
                else
                {
                    if (matrix->value[r * col + c] < result)
                    {
                        result = matrix->value[r * col + c];
                    }
                }
            }
        }
        return result;
    }
}
```

输出结果大致如下。



```
● (base) zhousicheng@zhousichengdeMacBook-Pro project_3_matrix % ./a.out
  the matrix have 10 rows and 10 cols
  12      54      78      63      35      77      49      83      28      14
  45      70      97      47      92      8       32      34      45      17
  8       74      14      62      65      38      104     83      21      40
  102     31      17      72      15      38      84      54      84      26
  72      77      98      41      90      50      33      96      99      62
  6       58      13      49      73      95      29      101     35      8
  27      71      54      29      6       58      82      13      33      38
  103     86      40      18      70      19      68      41      30      74
  20      99      34      6       22      100     10      9       56      103
  93      28      10      87      57      71      21      42      43      49

  104.000000
  6.000000
  Total time: 0.122000 ms
○ (base) zhousicheng@zhousichengdeMacBook-Pro project_3_matrix % 
```

统计结果如下。

方式	第一次	第二次	第三次	平均
一层	0.133000 ms	0.081000 ms	0.139000 ms	0.117666ms
两层	0.122000 ms	0.062000 ms	0.162000 ms	0.115333ms

本来以为一层循环避免了元素访存时的计算能够有更高的效率，但事实证明二者效率近似，甚至在本次小范围测试中两层循环还略胜一筹。大概是节省的计算时间对于程序的总运行时间来说微不足道。