- $\LaTeX$ of README.md may fail to display on github. For better experience, pls check report in pdf format.

CS205 C/ C++ Programming Project03

A Library for Matrix Operations in C

Name: 樊斯特(Fan Site)

SID: 12111624

Part 1 - Analysis

题目重述&主要思路

本题目要求**使用C语言**实现一个矩阵操作库，利用结构体存储矩阵，支持`float`矩阵的函数和基本运算，并对便捷性、鲁棒性、安全性、可扩展性有较高要求。

根据题目描述，题目要求的矩阵运算库需要支持的主要功能为：

1. 仅使用C语言，使用CMake管理项目
2. 结构体存储`float`类型矩阵
3. 矩阵操作：创建、删除、复制
4. 矩阵运算：矩阵加减乘，矩阵与标量四则运算，查询最值
5. 矩阵函数(安全性、易用性>效率)

本项目除完成上述全部基础要求外，支持以下内容：

6. **安全、用户友好**：使用指针操作时的安全性检查，严格明确的报错和警告判定，精确、安全的内存申请与释放。
7. **创建函数**：空矩阵、单位阵、对角阵、全赋值矩阵、随机矩阵、由数组创建矩阵、由字符串创建、由文件创建、由矩阵创建、提取子矩阵...较为丰富，可以覆盖相当程度的创建矩阵需求。
8. **矩阵操作**：安全删除矩阵并释放内存、值复制/引用复制、交换矩阵、矩阵变形、矩阵按行/列拼接、元素修改...安全易用。
9. **矩阵查询**：容量查询、自定义"最值"查询、单点查询、矩阵求秩、矩阵比较...安全易用易拓展。
10. **逐元素运算**：对单个矩阵进行一元运算、对两个矩阵进行二元运算、对单个矩阵与标量进行二元运算。极易拓展，支持自定义运算函数或使用库内置函数。
11. **矩阵运算**：矩阵加减乘、矩阵快速幂、求行列式...有两类实现：将结果存储于传入参数矩阵、将结果作为返回参数，前者节省内存，后者不影响传入矩阵。
12. **常用变换**：矩阵上三角化、高斯消元法矩阵求逆、转置矩阵、截取矩阵...安全易用且对无法计算的情况有强鲁棒性。
13. **文件操作**：从文件创建矩阵、输出结果到文件
14. **项目管理**：使用CMake和GitHub管理项目。

模型假设

项目要求关于应用场景仅给出了数据类型为`float`的要求，笔者考虑理工科计算所可能使用的数据范围作为本项目支持的数据规模，如下：

- 单个元素均为4字节`float`类型，有效位数默认为6位，数据范围约$-3.4*10^{-38}<val<3.4*10^{38}$
- 矩阵行数row与列数col之积，即矩阵容量满足$size=row*col\leq 10^8$
- 项目对安全性要求>计算效率，因此尽可能使用浅显稳定的算法，细节处有对效率的提升。

项目结构

```
CS205_Project03
│  CMakeLists.txt
│  README.md
│
├─doc
│      Report.pdf
│
├─inc
│      MatrixC.h
│
└─src
        Benchmark.c
        MatrixC.c
```

Part 2 - Code

宏与结构体

```c
#define TYPE float
#define float_equal(x, y) ((x-y)<1e-5&&(y-x)<1e-5)
#define MATRIX_MAX_CAPACITY 100000000
typedef struct MatrixC
{
    size_t row;
    size_t col;
    TYPE *data;
} Matrix;
```

本项目中，所有涉及矩阵元素下标的迭代变量均使用标准库常用的`size_t`类型，提高库在不同平台间的可移植性。

项目默认类型为`float`，笔者挺想做成类模板一样的可拓展形式，但由于C结构体不能像C++类一样使用模板，因而此处以宏进行丐版替代，有需求时可以调整`TYPE`，并对一些运算细节做微调就能适用不同数据类型。

经课程提醒，`float`类型的精度损失常导致`==`不能按预期判断相等，因此以宏的形式实现了浮点判等，误差在10^{-5}以内的数判断为相等。

矩阵容量上限为10^8个浮点元素，满足大部分情景的需求。

矩阵元素均存储于浮点指针`data`所指向的"一维浮点数组"中，data的容量由创建时赋为$row*col$后不作改变，更改方式仅有释放内存后重定向指针，从而避免了正常使用下段错误的发生。

在MatrixC.h头文件中，记录了本项目实现的矩阵相关的函数，经整理分为以下若干类：

创建矩阵

```
//Functions For Creating Matrices
Matrix *create_empty(size_t row, size_t col);//空矩阵

Matrix *create_full(size_t row, size_t col, TYPE value);//全部赋相同值

Matrix *create_from_array(TYPE *src, size_t row, size_t col);//由数组赋值

Matrix *create_from_string(char *src, size_t row, size_t col);//由字符串赋值

Matrix *create_from_file(char *f_path, size_t row, size_t col);//由文件赋值

Matrix *create_copy(Matrix *src);//创建浅拷贝

Matrix *create_identity(size_t order);//单位阵

Matrix *create_diagonal(TYPE *src, size_t order);//传入数组→对角阵

Matrix *create_random(size_t row, size_t col);//随机阵

Matrix *sub_matrix(Matrix *src, size_t row_begin, size_t col_begin, size_t
row_end, size_t col_end);//子矩阵截取
```

命名一目了然，简洁易懂，功能齐全，此处以创建空矩阵为例浅谈安全的实现：

```
Matrix *create_empty(size_t row, size_t col)
{
    if (row * col == 0)
    {
        print_error("Illegal matrix size", "Row and Col should be positive
integers.");
        return NULL;
    }
    else if (row * col > MATRIX_MAX_CAPACITY)
    {
        print_error("Illegal matrix size", "The maximum size of matrix is
row*col=1e8.");
        return NULL;
    }
    Matrix *new = malloc(sizeof(Matrix));
    new->row = row;
    new->col = col;
    new->data = malloc(sizeof(TYPE) * row * col);
    return new;
}
```

1. 传入参数为两个`size_t`类型变量(unsigned)，首先检查尺寸的合法性：$0 < row * col \leq 10^8$，不对就报错返空(返回的`NULL`指针在后续的使用中也会严格检查)
2. 使用`malloc`为新矩阵指针`new`分配一个`Matrix`类型的内存，其中包含结构体的两个`size_t`类型变量，即刻为其赋值。
3. 对分配的一个`float *`浮点数指针变量申请$row * col$大小的内存。
4. 返回矩阵指针，创建完毕。

其他矩阵的创建均有调用该函数，实现简单，并检查了返回`NULL`等情况，此处不做赘述。

矩阵级别操作

```
//Functions For Matrix Operations
bool delete_matrix(Matrix **pmat);//安全删除

bool reshape_matrix(Matrix *src, size_t row, size_t col);//不变尺寸的矩阵变形

bool copy_matrix(Matrix *dest, Matrix *src);//值复制

bool ref_matrix(Matrix *dest, Matrix *src);//引用复制

bool swap_matrix(Matrix *first, Matrix *second);//交换矩阵

Matrix *col_concat(Matrix *first, Matrix *second);//按列拼接

Matrix *row_concat(Matrix *first, Matrix *second);//按行拼接

bool set_value(Matrix *pmat, size_t row, size_t col, TYPE value);//单点修改
```

上述函数分别为：删除、变形、浅拷贝、深拷贝、交换、按列拼接、按行拼接。

原本前五个是无返回函数，笔者觉得返回值不能就这么浪费了，因此传回布尔型，代表该操作是否成功进行(若发生异常会输出报错/警告信息，并返回false)。

以下是部分函数的展开讲解（其他的实现原理简单且类似，故不做赘述）：

删除矩阵

```
bool delete_matrix(Matrix **pmat)
{
    if (pmat == NULL)
    {
        print_error("NULL pointer exception",
                    "The address of the pointer to the matrix is null, delete
process interrupted.");
        return false;
    }
    if ((*pmat) == NULL)
    {
        print_error("NULL pointer exception", "The pointer to the matrix is
null, delete process interrupted.");
        return false;
    }
    free((*pmat)->data);
    free(*pmat);
    *pmat = NULL;
    return true;
}
```

矩阵的删除函数步骤如下：

1. 判断空指针并报错

2. 释放结构体内浮点指针所指内存
3. 释放结构体内存
4. 置空指针

本项目将指针指针(指针的地址)作为传入参数，这是由于C在处理空指针时有良好的鲁棒性(如`free(NULL)`将不进行操作)，在释放结构体内存后将指向结构体的**指针本身置空**，可以避免**对野指针进行操作**的高危行为。

然而将指针本身作为传入参数进行值传递后，在函数内将传入参数置空并不耽误原指针还是野指针，C也没有引用，故传入指针的指针对原指针进行置空。

值得一提，虽然C对于"释放空指针的内存"有所提防，但本函数依然对传入指针为空的情况做了报错，让用户清楚地知道可能进行了**释放空指针所指内存**这样的行为。

复制矩阵

```
bool copy_matrix(Matrix **dest, Matrix *src)
{
    if (src == NULL)
    {
        print_error("NULL pointer exception",
                    "The pointer to source matrix is null, copy process
interrupted.");
        return false;
    }
    if (*dest == NULL)
    {
        *dest = create_copy(src);
        if (*dest == NULL)
        {
            return false;
        }
        print_warning("Copy into null matrix",
                      "The pointer to destination matrix is null, the pointer
will point to a copy matrix of source matrix.");
    }
    else
    {
        if (((*dest)->row != src->row) || ((*dest)->col != src->col))
        {
            print_warning("Copy into matrix of different sizes",
                          "The sizes of two matrices are different, the data of
destination matrix will be covered by source matrix.");
        }
        delete_matrix(dest);
        *dest = create_copy(src);
        if (*dest == NULL)
        {
            return false;
        }
    }
    return true;
}

bool ref_matrix(Matrix **dest, Matrix *src)
```

```
{
    if (src == NULL)
    {
        print_error("NULL pointer exception",
                    "The pointer to source matrix is null, copy process
interrupted.");
        return false;
    }
    if(*dest!=NULL)
    {
        delete_matrix(dest);
    }
    *dest = src;
    return true;
}
```

两种复制的实现都很简洁安全：若目标矩阵为空则会为其申请空间并复制，若目标矩阵与源矩阵尺寸不同会警告用户复制将覆盖目标矩阵原有信息，且对可能出现的错误进行报错处理。

拼接矩阵

由于高斯消元求逆过程用到了按行拼接，此处对其进行简单讲解：

```
Matrix *row_concat(Matrix *first, Matrix *second)
{
    if (first == NULL || second == NULL)
    {
        print_error("NULL pointer exception", "The pointer to source matrix is
null, concat process interrupted.");
        return NULL;
    }
    if (first->row != second->row)
    {
        print_error("Illegal matrix shape",
                    "The number of row of source matrices should be the same,
concat process interrupted.");
        return NULL;
    }
    Matrix *new = create_empty(first->row, first->col + second->col);
    if (new == NULL)
    {
        return NULL;
    }
    for (size_t i = 0; i < first->row; i++)
    {
        for (size_t j = 0; j < first->col; j++)
        {
            new->data[i * new->col + j] = first->data[i * first->col + j];
        }
        for (size_t j = 0; j < second->col; j++)
        {
            new->data[i * new->col + j + first->col] = second->data[i * second-
>col + j];
        }
    }
```

```
    return new;
}
```

按行拼接的步骤如下：

1. 检查空指针，检查两个矩阵行数是否相等并报错
2. 新建合并大小的空矩阵指针`new`，检查大小是否在合法范围内
3. 逐行将两个矩阵的值传入`new`
4. 返回`new`

拼接原理即按行赋值，左边`first`，右边`second`。容易忽视的漏洞是拼接前两个大小合适的矩阵可能在拼接后超限，此时不应对空指针`new`进行赋值，而应报错。

查询操作

```
//Functions For Querying In Matrices
size_t size_of(Matrix *pmat);//查询容量

size_t rank(Matrix *pmat);//矩阵求秩

TYPE get_value(Matrix *src, size_t row, size_t col);//单点查询

TYPE max(Matrix *src);//矩阵最大值

TYPE min(Matrix *src);//矩阵最小值

TYPE extreme_value(Matrix *src, bool (*fun)(TYPE, TYPE));//矩阵自定义“最值”

bool equal(Matrix *first, Matrix *second);//矩阵判等
```

矩阵求秩主要依靠上三角化实现，矩阵的秩即上三角化后非零行的数量，详见下文上三角化函数。

自定义最值

求最值的思路是遍历`data`数组并逐个比较，方法朴素但安全，易于维护，下面简述可扩展的`extreme_value`函数：

```
TYPE extreme_value(Matrix *src, bool (*cmp)(TYPE, TYPE))
{
    if (src == NULL)
    {
        print_error("NULL pointer exception",
                    "The pointer to source matrix is null, return NaN.");
        return nanf("");
    }
    TYPE ans = src->data[0];
    for (size_t i = 1; i < size_of(src); i++)
    {
        if (cmp(src->data[i], ans))
        {
            ans = src->data[i];
        }
    }
```

```
    return ans;
}
```

矩阵自定义最值的步骤如下：

1. 判断空指针并报错，返回`NaN`
2. 根据`cmp`函数逐个比较矩阵元素
3. 返回答案

此处将函数指针作为第三参数传入，实现了“比较”过程的自定义可拓展，对于`TYPE`类型的数据，比较规则可以通过自定义`cmp`函数进行调整，灵感来自于`sort`的自定义比较函数。

自定义矩阵运算

```
//Functions For Customized Calculation
TYPE plus(TYPE first, TYPE second);

TYPE minus(TYPE first, TYPE second);

TYPE mul(TYPE first, TYPE second);

TYPE divide(TYPE first, TYPE second);

//Functions For Matrix Calculation
Matrix *unary_calc(Matrix *pmat, TYPE(*fun)(TYPE));

Matrix *binary_calc(Matrix *first, Matrix *second, TYPE (*fun)(TYPE, TYPE));

Matrix *scalar_calc(Matrix *pmat, TYPE scalar, TYPE(*fun)(TYPE, TYPE));
```

考虑到我们经常要对矩阵进行逐元素的运算，例如矩阵统一求相反数是逐元素一元运算，矩阵加减法是逐元素二元运算...为了减轻用户为了不同的运算而自行实现多个函数的压力，本项目将**一元、二元、矩阵与标量的运算**整合为**拓展性极强的函数**，用户只需要自定义好运算函数，传入即可对矩阵进行**逐元素的自定义运算**。例如用户要对矩阵中的每个元素求正弦，那么只需要传入`Math.h`内置的`sinf(float x)`即可：

```
Matrix *sin_matrix = unary_calc(Matrix *pmat, sinf);
```

代码部分选择二元运算进行展示：

```
Matrix *binary_calc(Matrix *first, Matrix *second, TYPE (*fun)(TYPE, TYPE))
{
    if (first == NULL)
    {
        print_error("NULL pointer exception", "The pointer to the first matrix
is null, calculation interrupted.");
        return NULL;
    }
    if (second == NULL)
    {
        print_error("NULL pointer exception", "The pointer to the second matrix
is null, calculation interrupted.");
```

```
        return NULL;
    }
    if (first->row != second->row || first->col != second->col)
    {
        print_error("Illegal matrix shape",
                    "The shape of two matrices are different, calculation
process interrupted.");
        return NULL;
    }
    Matrix *new = create_copy(first);

    for (size_t i = 0; i < size_of(new); i++)
    {
        new->data[i] = fun(new->data[i], second->data[i]);
    }
    return new;
}
```

步骤也很简单，检查报错→申请空间→代入运算→存入结果，简洁高效易懂。

也因此，题目要求的四种标量运算只用向`scalar_calc`传入`plus`，`minus`，`mul`，`divide`四个自定义函数即可。

矩阵计算

```
//Functions For Matrix Calculation
bool add_by(Matrix *augend, Matrix *addend);

Matrix *matrix_add(Matrix *augend, Matrix *addend);

bool subtract_by(Matrix *minuend, Matrix *subtrahend);

Matrix *matrix_subtract(Matrix *minuend, Matrix *subtrahend);

bool multiply_by(Matrix **multiplicand, Matrix *multiplier);

bool multiply_to(Matrix *multiplicand, Matrix **multiplier);

Matrix *matrix_multiply(Matrix *multiplicand, Matrix *multiplier);

Matrix *matrix_pow(Matrix *base, int power);

bool add_scalar(Matrix *pmat, TYPE scalar);

bool subtract_scalar(Matrix *pmat, TYPE scalar);

bool multiply_scalar(Matrix *pmat, TYPE scalar);

bool divide_scalar(Matrix *pmat, TYPE scalar);
```

矩阵运算的实际应用场景中经常会有类似**自增**的需求(将结果保存在两个矩阵中的其中一个)，尤其体现在矩阵的**左乘和右乘**等方面。

本项目除了实现将结果作为新结构体返回的函数，也实现了将结果存入两个矩阵之一的函数，返回值为布尔型的函数会将结果存入传入的矩阵中。布尔型的返回值代表操作是否成功，若失败则返回`false`并报错，不对传入矩阵做任何操作。

以下是矩阵乘法的实现：

```
Matrix *matrix_multiply(Matrix *multiplicand, Matrix *multiplier)
{
    if (multiplicand == NULL || multiplier == NULL)
    {
        print_error("NULL pointer exception",
                    "The pointers to multiplicand and multiplier matrix are
null, multiplication process interrupted.");
        return NULL;
    }
    if (multiplicand->col != multiplier->row)
    {
        print_error("Illegal matrix shape",
                    "The col number of multiplicand matrix should equal to row
number of multiplier matrix.");
        return NULL;
    }
    Matrix *new = create_full(multiplicand->row, multiplier->col, 0);
    for (size_t i = 1; i <= multiplicand->row; i++)
    {
        for (size_t k = 1; k <= multiplicand->col; k++)
        {
            TYPE t = multiplicand->data[(i - 1) * multiplicand->col + k - 1];
            for (size_t j = 1; j <= multiplier->col; j++)
            {
                new->data[(i - 1) * new->col + j - 1] += t * multiplier->data[(k
- 1) * multiplier->col + j - 1];
            }
        }
    }
    return new;
}
```

除了~~已经敲到不能再熟的~~报错环节，由于项目对效率要求并不高，因此矩阵乘法依然使用的是$O(n^3)$的传统矩阵乘法，但使用了ikj的三层循环顺序，可以较好地提高内存访问的连续性，具体原理参考自知乎用户@赛森Lambda-CDM的文章。下图是摘自文章的各个循环顺序的内存访问跳跃数列表，作为参考：

- 顺序 ikj —— $2n^2+n$ （二维数组） —— n^2 （一维数组）
- 顺序 kij —— $3n^2$ （二维数组） —— $2n^2$ （一维数组）
- 顺序 jik —— n^3+2n^2 （二维数组） —— n^3+n^2+n （一维数组）
- 顺序 ijk —— n^3+n^2+n （二维数组） —— n^3+n^2-n （一维数组）
- 顺序 kji —— $2n^3+n$ （二维数组） —— $2n^3$ （一维数组）
- 顺序 jki —— $2n^3+n^2$ （二维数组） —— $2n^3+n^2$ （一维数组）

矩阵快速幂

喜闻乐见的快速幂环节，具体原理为将指数二进制表示后，通过倍乘`base`矩阵将乘法次数优化到`log(power)`次，对于$O(n^3)$的矩阵乘法而言优化力度较为客观，代码如下：

```
Matrix *matrix_pow(Matrix *base, int power)
{
    if (base == NULL)
    {
        print_error("NULL pointer exception", "The pointer to base matrix is
null, power process interrupted.");
        return NULL;
    }
    if (power == 1)
    {
        return create_copy(base);
    }
    if (base->row != base->col)
    {
        print_error("Illegal matrix shape", "The base matrix should be square
matrix.");
        return NULL;
    }
    if (power == 0)
    {
        return create_identity(base->row);
    }
    Matrix *new;
    if (power < 0)
    {
        new = create_copy(inverse(base));
        if (new == NULL)
        {
            print_error("NULL pointer exception",
                        "The source matrix has no inverse, negative power
calculation interrupted.");
            return NULL;
        }
        power = -power;
    }
    else
    {
        new = create_copy(base);
    }
    Matrix *Base = create_copy(base);
    power--;//因为new本来就是base一次方了所以-1
    while (power)
    {
        if (power & 1)
        {
            multiply_by(&new, Base);
        }
        multiply_by(&Base, Base);
        power >>= 1;
    }
```

```
    return new;
}
```

实际上，矩阵的幂运算在正整数之外并没有定义，本项目中为了便利，将矩阵的幂的定义进行拓展：

若矩阵为方阵，则其0次幂为单位阵；

若矩阵为方阵且可逆，则其负数次幂为其逆的对应正数次幂。

矩阵变换

```
//Functions For Matrix Transformations
TYPE determinant(Matrix *pmat);

Matrix *inverse(Matrix *pmat);

Matrix *transpose(Matrix *pmat);

Matrix *Uptriangular(Matrix *pmat);
```

本节包含矩阵专属的一些常用函数：求行列式、求逆、转置、上三角化，求行列式和求逆均依赖上三角化进行。

矩阵转置

```
Matrix *transpose(Matrix *pmat)
{
    if (pmat == NULL)
    {
        print_error("NULL pointer exception", "The pointer to source matrix is
null, transpose process interrupted.");
        return NULL;
    }
    Matrix *new = create_empty(pmat->col, pmat->row);
    for (size_t i = 0; i < pmat->row; i++)
    {
        for (size_t j = 0; j < pmat->col; j++)
        {
            *(new->data + j * new->col + i) = *(pmat->data + i * pmat->col + j);
        }
    }
    return new;
}
```

行列交换即可。

上三角化

```
Matrix *Uptriangular(Matrix *pmat)
{
    if (pmat == NULL)
    {
        print_error("NULL pointer exception",
```

```
                    "The pointer to source matrix is null, uptriangular process
interrupted.");
        return NULL;
    }
    Matrix *new = create_copy(pmat);
    size_t lim = new->row < new->col ? new->row : new->col;
    for (size_t i = 0; i < lim; i++)
    {
        if (float_equal(new->data[i * new->col + i], 0.0f))
        {
            for (size_t j = i + 1; j < new->row; j++)
            {
                if (!float_equal(new->data[j * new->col + i], 0.0f))
                {
                    for (size_t k = 0; k < new->col; k++)
                    {
                        TYPE t = new->data[i * new->col + k];
                        new->data[i * new->col + k] = new->data[j * new->col +
k];
                        new->data[j * new->col + k] = t;
                    }
                    break;
                }
            }
        }
        if (float_equal(new->data[i * new->col + i], 0.0f))
        {
            continue;
        }
        for (size_t j = i + 1; j < new->row; j++)
        {
            TYPE t = new->data[j * new->col + i] / new->data[i * new->col + i];
            for (size_t k = 0; k < new->col; k++)
            {
                new->data[j * new->col + k] -= new->data[i * new->col + k] * t;
            }
        }
    }
    Matrix *res = create_full(new->row, new->col, 0.0f);
    for (size_t i = 0, it = 0; i < new->row; i++)
    {
        bool emp = true;
        for (size_t j = 0; j < new->col; j++)
        {
            if (!(float_equal(new->data[i * new->col + j], 0.0f)))
            {
                res->data[it * res->col + j] = new->data[i * new->col + j];
                emp = false;
            }
        }
        it += emp ? 0 : 1;
    }
    delete_matrix(&new);
    return res;
}
```

上三角化是线性代数第一课就会讲的内容，也是线性代数大部分变换的基础，实现后求逆、求行列式等操作则迎刃而解。

上三角化的主要步骤为：

1. 检查报错，记录行数与列数中较小的一个作为循环上界
2. 按列进行，对于矩阵A的第i列：
    - 若$A[i][i]$非零，则将其下方的所有行减去其倍数，直到$A[i][i]$下方所有元素消为0
    - 若$A[i][i]=0$，则向下寻找第一个$j>i$使得$A[j][i]\neq 0$，交换第i行和第j行
    - 若到达最后一行仍没有找到，则说明该列主元缺失，$rank--$
3. 减除完毕后，部分全零行会存在矩阵中，此时将全零行下沉到矩阵最下方，得到完整的上三角矩阵U。
4. 释放临时矩阵，返回结果

矩阵求秩

矩阵的秩经过基础行变换不会变化，因此我们只需统计上三角化后的矩阵的非零行数量即可。

矩阵求行列式

对矩阵进行除了换行以外的初等行变换不会影响矩阵的行列式，换行操作会使行列式变为相反数，因此在上三角化的过程中，我们只需记录换行的次数，并对上三角阵U的对角元素进行累乘运算即可得到矩阵的行列式。

代码较上三角化仅添加了几行(换行时记录符号，最后累乘)，此处不费篇幅展示。

矩阵求逆

```
Matrix *inverse(Matrix *pmat)
{
    if (pmat == NULL)
    {
        print_error("NULL pointer exception",
                    "The pointer to source matrix is null, inverse calculation
interrupted.");
        return NULL;
    }
    if (pmat->row != pmat->col)
    {
        print_error("Illegal matrix shape",
                    "The matrix should be square to have inverse, inverse
calculation interrupted.");
    }
    Matrix *new = row_concat(pmat, create_identity(pmat->row));
    size_t row = pmat->row;
    size_t col = row * 2;
    for (size_t i = 0; i < row; i++)
    {
        if (float_equal(new->data[i * col + i], 0.0f))
        {
            for (size_t j = i + 1; j < row; j++)
            {
                if (!float_equal(new->data[j * col + i], 0.0f))
                {
```

```
                    for (size_t k = 0; k < col; k++)
                    {
                        TYPE t = new->data[i * col + k];
                        new->data[i * col + k] = new->data[j * col + k];
                        new->data[j * col + k] = t;
                    }
                    break;
                }
            }
        }
        if (float_equal(new->data[i * col + i], 0.0f))
        {
            print_error("Inverse doesn't exist",
                        "The rank of source matrix is not full, inverse
calculation interrupted.");
            return NULL;
        }
        for (size_t j = i + 1; j < row; j++)
        {
            TYPE t = new->data[j * col + i] / new->data[i * col + i];
            for (size_t k = 0; k < col; k++)
            {
                new->data[j * col + k] -= new->data[i * col + k] * t;
            }
        }
    }
    for (size_t i = 0; i < row; i++)
    {
        TYPE u = new->data[i * col + i];
        for (size_t j = 0; j < i; j++)
        {
            TYPE v = new->data[j * col + i] / u;
            for (size_t k = i; k < col; k++)
            {
                new->data[j * col + k] -= new->data[i * col + k] * v;
            }
        }
    }
    for (size_t i = 0; i < row; i++)
    {
        TYPE t = new->data[i * col + i];
        for (size_t j = row; j < col; j++)
        {
            new->data[i * new->col + j] /= t;
        }
    }
    return sub_matrix(new, 1, pmat->col + 1, new->row, new->col);
}
```

完成了上三角化，我们可以用$Gauss\ Jordan\ Elimination$进行矩阵求逆：

1. 检查报错，记录行数与列数中较小的一个作为循环上界
2. 在原矩阵的右侧拼合一个同阶单位矩阵，与原矩阵进行同样的行变换
3. 按列进行，对于矩阵A的第i列：

- 若$A[i][i]$非零，则将其下方的所有行减去其倍数，直到$A[i][i]$下方所有元素消为0
- 若$A[i][i]=0$，则向下寻找第一个$j>i$使得$A[j][i]\neq 0$，交换第i行和第j行
- 若到达最后一行仍没有找到，则说明该列主元缺失，矩阵不满秩，不存在逆，**返回空指针**

4. 减除完毕后，有逆的矩阵不存在全零行，得到上三角矩阵U
5. 按列反向进行，对于矩阵A的第i列，将$A[i][i]$上方的所有行减去其倍数直到消为0
6. 上一步完成后，拼合矩阵的左侧为对角阵，对于每一行除以其主元$A[i][i]$即可
7. 利用`sub_matrix`函数截取起初拼合在右侧的单位矩阵，此时经过行变换已经变成了矩阵的逆
8. 返回结果

报错与警告

```
//Functions For Debugging, Error & Warning
void print_matrix(Matrix *pmat, int precision);

void print_error(char *err_type, char *err_info);

void print_warning(char *w_type, char *w_info);
```

本项目中，参数列表带有矩阵的函数均有报错/警告语句，为了综合报错和警告而非排布`printf`语句，本项目使用两个函数进行规范化报错和警告，分为类型和具体信息两部分，内容详细具体。

输出矩阵

```
void print_matrix(Matrix *pmat, int precision)
{
    if (pmat == NULL)
    {
        print_error("NULL pointer exception", "The pointer to source matrix is
null, print process interrupted.");
        return;
    }
    if (precision < 0)
    {
        print_error("Illegal precision", "Precision should be non-negative,
print process interrupted.");
        return;
    }
    if (precision > 6)
    {
        print_warning("Precision too large", "Float numbers are accurate to at
most the 6th decimal place.");
        precision = 6;
    }
    for (size_t i = 0; i < pmat->row; i++)
    {
        for (size_t j = 0; j < pmat->col; j++)
        {
            if (float_equal(pmat->data[i * pmat->col + j], 0.0f))
            {
                pmat->data[i * pmat->col + j] = 0.0f;
            }
```

```
            printf("%.*f\t", precision, pmat->data[i * pmat->col + j]);
        }
        printf("\n");
    }
}
```

考虑到用户可能并不是每次都想输出小数点后6位数字，继承前两次project优良传统的笔者给输出函数加了精度参数，如果只是整数级别的运算就没必要显示小数点后了嘛。

不过就算把`precision`设置为114514，囿于`float`型的精度也只能精确到6位以内，因此程序在接收到高于6的精度要求后会抛个警告并坚持设置精度为6。

输出的部分也是朴素安全，在调试时笔者注意到一个细节：有时候会输出`-0.0`这样的数据，看上去很怪，这是因为没有显示完全一个很接近0的负数，所以采用了`float_equal`进行处理，顺便把原矩阵的值也修改为常规的`0.0f`。

输出矩阵到文件

```
bool print_matrix_to_file(char *filename, Matrix *pmat, int precision)
{
    FILE *file = fopen(filename, "w");
    if (file == NULL)
    {
        print_error("NULL pointer exception",
                    "The output file is not found, print to file process
interrupted.");
        return false;
    }
    if (pmat == NULL)
    {
        print_error("NULL pointer exception",
                    "The pointer to source matrix is null, print to file process
interrupted.");
        return false;
    }
    if (precision < 0)
    {
        print_error("Illegal precision", "Precision should be non-negative,
print to file process interrupted.");
        return;
    }
    if (precision > 6)
    {
        print_warning("Precision too large", "Float numbers are accurate to at
most the 6th decimal place.");
        precision = 6;
    }
    for (size_t i = 0; i < pmat->row; i++)
    {
        for (size_t j = 0; j < pmat->col; j++)
        {
            if (float_equal(pmat->data[i * pmat->col + j], 0.0f))
            {
                pmat->data[i * pmat->col + j] = 0.0f;
            }
```

```
            fprintf(file, "%.*f\t", precision, pmat->data[i * pmat->col + j]);
        }
        fprintf(file, "\n");
    }
    fclose(file);
    return true;
}
```

对于规模较大的矩阵，用户会有将结果输出到指定文件的需求，本项目也进行了实现。

Part 3 - Result & Verification

Testcase #1 创建矩阵

```
//Benchmark.c (main function)
int main()
{
    Matrix *mat = create_full(2, 2, 2);
    print_matrix(mat, 2);
    putchar('\n');
    delete_matrix(&mat);

    TYPE arr[] = {2, 0, 2, 2, 1, 0, 3, 2};
    mat = create_from_array(arr, 4, 2);
    print_matrix(mat, 2);
    putchar('\n');
    delete_matrix(&mat);

    mat = create_from_string("1,1,4;5,1,4", 2, 3);
    print_matrix(mat, 2);
    putchar('\n');
    delete_matrix(&mat);

    Matrix *mat2 = create_copy(mat);
    delete_matrix(&mat);
    print_matrix(mat2, 2);
    putchar('\n');
    delete_matrix(&mat2);

    mat = create_identity(3);
    print_matrix(mat, 2);
    putchar('\n');
    delete_matrix(&mat);

    mat= create_random(4,4);
    print_matrix(mat, 2);
    putchar('\n');
    delete_matrix(&mat);
}
```

Result:

```
gutao@FIRST-MICROSOFT:/mnt/e/Cpp/Project03$ ./matlib
2.00    2.00
2.00    2.00

2.00    0.00
2.00    2.00
1.00    0.00
3.00    2.00

1.00    1.00    4.00
5.00    1.00    4.00

1.00    1.00    4.00
5.00    1.00    4.00

1.00    0.00    0.00
0.00    1.00    0.00
0.00    0.00    1.00

0.43    0.86    0.97    0.32
0.50    0.00    0.13    0.15
0.39    0.69    0.85    0.32
0.37    0.75    0.03    0.31
```

Testcase #2 矩阵级别操作

删除

```c
//Benchmark.c (main function)
int main()
{
    Matrix *mat = create_full(3, 3, 1.2);
    print_matrix(mat, 1);
    delete_matrix(&mat);
    print_matrix(mat, 1);
}
```

Result:

```
gutao@FIRST-MICROSOFT:/mnt/e/Cpp/Project03$ ./matlib
1.2     1.2     1.2
1.2     1.2     1.2
1.2     1.2     1.2
Error: NULL pointer exception
Error log: The pointer to source matrix is null, print process interrupted.
Current operation interrupted, please check and try again.
```

复制

```c
//Benchmark.c (main function)
int main()
{
    TYPE arr[] = {0.1, -0.2, 0.3, -0.4};
    Matrix *mat = create_from_array(arr, 2, 2);
    Matrix *cpy=NULL;
    Matrix *ref=NULL;
```

```
    copy_matrix(&cpy,mat);
    ref_matrix(&ref,mat);
    set_value(mat,2,2,2);
    printf("copy:\n");
    print_matrix(cpy,1);
    printf("reference:\n");
    print_matrix(ref,1);
}
```

Result:

```
gutao@FIRST-MICROSOFT:/mnt/e/Cpp/Project03$ ./matlib
Warning: Copy into null matrix
Details: The pointer to destination matrix is null, the pointer will point to a copy matrix of source matrix.
copy:
0.1     -0.2
0.3     -0.4
reference:
0.1     -0.2
0.3     2.0
```

拼接

```
//Benchmark.c (main function)
int main()
{
    Matrix *mat = create_from_string("1,1,2;3,5,8;13,21,34;55,89,144",4, 3);
    Matrix *i3=create_identity(3);
    Matrix *i4=create_identity(4);
    printf("col_concat:\n");
    print_matrix(col_concat(mat,i3),0);
    printf("row_concat:\n");
    print_matrix(row_concat(i4,mat),0);
}
```

Result:

```
gutao@FIRST-MICROSOFT:/mnt/e/Cpp/Project03$ ./matlib
col_concat:
1       1       2
3       5       8
13      21      34
55      89      144
1       0       0
0       1       0
0       0       1
row_concat:
1       0       0       0       1       1       2
0       1       0       0       3       5       8
0       0       1       0       13      21      34
0       0       0       1       55      89      144
```

Testcase #3 矩阵查询

自定义最值(从文件中读取)

```
//matfile
1,3,5,7,9;
2,4,6,8,10;
9,8,7,6,5;
4,3,2,1,0;
1.1,2.2,3.3,4.4,5.5;
```

```
//Benchmark.c (customized compare function & main function)
bool mycmp(TYPE x,TYPE y)
{
    return fabs(x-2.1)<fabs(y-2.1);
}
int main()
{
    Matrix *mat=create_from_file("matfile",5,5);
    print_matrix(mat,1);
    printf("The max value is %.1f\nThe min value is %.1f\n",max(mat),min(mat));
    printf("The value closest to 2.1 is %.1f\n",extreme_value(mat,mycmp));
}
```

Result:

```
gutao@FIRST-MICROSOFT:/mnt/e/Cpp/Project03$ ./matlib
1.0     3.0     5.0     7.0     9.0
2.0     4.0     6.0     8.0     10.0
9.0     8.0     7.0     6.0     5.0
4.0     3.0     2.0     1.0     0.0
1.1     2.2     3.3     4.4     5.5
The max value is 10.0
The min value is 0.0
The value closest to 2.1 is 2.0
```

Testcase #4 自定义矩阵运算

```
//Benchmark.c (main function)
int main()
{
    Matrix *mat=create_from_file("matfile",5,5);
    printf("origin mat:\n");
    print_matrix(mat,1);
    printf("unary operation mat:\n");
    Matrix *una=unary_calc(mat,cosf);
    print_matrix(una,2);
    printf("binary operation mat:\n");
    Matrix *bin=binary_calc(mat,una,divide);
    print_matrix(bin,2);
    printf("scalar operation mat:\n");
    Matrix *sca=scalar_calc(mat,6,minus);
    print_matrix(sca,2);
```

```
}
```

Result:

```
gutao@FIRST-MICROSOFT:/mnt/e/Cpp/Project03$ ./matlib
origin mat:
1.0     3.0     5.0     7.0     9.0
2.0     4.0     6.0     8.0     10.0
9.0     8.0     7.0     6.0     5.0
4.0     3.0     2.0     1.0     0.0
1.1     2.2     3.3     4.4     5.5
unary operation mat:
0.54    -0.99   0.28    0.75    -0.91
-0.42   -0.65   0.96    -0.15   -0.84
-0.91   -0.15   0.75    0.96    0.28
-0.65   -0.99   -0.42   0.54    1.00
0.45    -0.59   -0.99   -0.31   0.71
binary operation mat:
1.85    -3.03   17.63   9.29    -9.88
-4.81   -6.12   6.25    -54.98  -11.92
-9.88   -54.98  9.29    6.25    17.63
-6.12   -3.03   -4.81   1.85    0.00
2.43    -3.74   -3.34   -14.32  7.76
scalar operation mat:
-5.00   -3.00   -1.00   1.00    3.00
-4.00   -2.00   0.00    2.00    4.00
3.00    2.00    1.00    0.00    -1.00
-2.00   -3.00   -4.00   -5.00   -6.00
-4.90   -3.80   -2.70   -1.60   -0.50
```

Testcase #5 矩阵计算

简单运算

```c
//Benchmark.c (main function)
int main()
{
    srand(time(NULL));
    Matrix *A=create_random(3,3);
    printf("A:\n");
    print_matrix(A,2);
    Matrix *B=create_random(3,3);
    printf("B:\n");
    print_matrix(B,2);
    multiply_by(&A,B);
    printf("A=A*B=\n");
    print_matrix(A,2);
    printf("A+B=\n");
    print_matrix(matrix_add(A,B),2);
    printf("A-B=\n");
    print_matrix(matrix_subtract(A,B),2);
}
```

Result:

```
gutao@FIRST-MICROSOFT:/mnt/e/Cpp/Project03$ make && ./matlib
Consolidate compiler generated dependencies of target matlib
[100%] Built target matlib
A:
0.66    0.77    0.36
0.70    0.79    0.92
0.74    0.47    0.38
B:
0.27    0.45    0.58
0.53    0.06    0.27
0.55    0.75    0.00
A=A*B=
0.78    0.61    0.60
1.10    1.04    0.62
0.66    0.65    0.56
A+B=
1.05    1.07    1.18
1.63    1.10    0.89
1.21    1.40    0.56
A-B=
0.51    0.16    0.02
0.58    0.99    0.35
0.11    -0.10   0.56
```

矩阵加速斐波那契数列

```c
//Benchmark.c (main function)
int main()
{
    Matrix *A=create_from_string("1,1;1,0",2,2);
    Matrix *v=create_from_string("1;1",2,1);
    for(size_t i=0;i<20;i++)
    {
        Matrix *t=matrix_multiply(matrix_pow(A,i),v);
        printf("fib[%d] = %.0f\n",i,get_value(t,2,1));
    }
}
```

Result:

```
gutao@FIRST-MICROSOFT:/mnt/e/Cpp/Project03$ make && ./matlib
Consolidate compiler generated dependencies of target matlib
[100%] Built target matlib
fib[0] = 1
fib[1] = 1
fib[2] = 2
fib[3] = 3
fib[4] = 5
fib[5] = 8
fib[6] = 13
fib[7] = 21
fib[8] = 34
fib[9] = 55
fib[10] = 89
fib[11] = 144
fib[12] = 233
fib[13] = 377
fib[14] = 610
fib[15] = 987
fib[16] = 1597
fib[17] = 2584
fib[18] = 4181
fib[19] = 6765
```

Testcase #6 矩阵变换

上三角化，行列式，逆

```c
//Benchmark.c (main function)
int main()
{
    Matrix *A=create_random(6,6);
    printf("A:\n");
    print_matrix(A,2);
    Matrix *U=Uptriangular(A);
    printf("U:\n");
    print_matrix(U,2);
    Matrix *I=inverse(A);
    printf("A^-1:\n");
    print_matrix(I,2);
    printf("A*(A^-1)=\n");
    print_matrix(matrix_multiply(A,I),2);
    printf("|A| = %f\n",determinant(A));
}
```

Result:

```
gutao@FIRST-MICROSOFT:/mnt/e/Cpp/Project03$ make && ./matlib
[100%] Built target matlib
A:
0.79    0.85    0.97    0.67    0.30    0.82
0.57    0.79    0.91    0.40    0.15    0.22
0.58    0.54    0.71    0.78    0.06    0.38
0.06    0.36    0.57    0.76    0.66    0.53
0.64    0.60    0.42    0.04    0.24    0.44
0.15    0.03    0.29    0.13    0.70    0.59
U:
0.79    0.85    0.97    0.67    0.30    0.82
0.00    0.17    0.21    -0.09   -0.07   -0.38
0.00    0.00    0.10    0.24    -0.20   -0.41
0.00    0.00    0.00    0.56    1.02    1.68
0.00    0.00    0.00    0.00    -0.71   -1.76
0.00    0.00    0.00    0.00    0.00    -2.61
A^-1:
-2.75   -0.21   3.39    -1.28   1.81    1.52
0.61    -0.44   -2.19   2.30    2.08    -2.84
1.26    2.29    -0.73   -1.67   -3.17   1.69
-1.10   -0.94   1.85    0.96    0.58    -0.59
-2.85   0.54    1.02    0.94    1.80    0.95
3.69    -1.47   -2.01   -0.30   -1.29   -0.38
A*(A^-1)=
1.00    0.00    0.00    0.00    0.00    0.00
0.00    1.00    0.00    0.00    0.00    0.00
0.00    0.00    1.00    0.00    0.00    0.00
0.00    0.00    0.00    1.00    0.00    0.00
0.00    0.00    0.00    0.00    1.00    0.00
0.00    0.00    0.00    0.00    0.00    1.00
|A| = 0.013963
```

不满秩矩阵

```
//matfile
1,0,1,0,1,0
0,2,0,2,0,2
3,0,3,0,3,0
0,4,0,4,0,4
5,0,5,0,5,0
0,6,0,6,0,6
```

```
//Benchmark.c (main function)
int main()
{
    Matrix *not_full_rank_mat=create_from_file("matfile",6,6);
    printf("Not full rank matrix:\n");
    print_matrix(not_full_rank_mat,2);
    printf("inverse:\n");
    print_matrix(inverse(not_full_rank_mat),2);
    printf("determinant: %.0f\n",determinant(not_full_rank_mat));
}
```

Result:

```
gutao@FIRST-MICROSOFT:/mnt/e/Cpp/Project03$ make && ./matlib
Consolidate compiler generated dependencies of target matlib
[100%] Built target matlib
Not full rank matrix:
1.00    0.00    1.00    0.00    1.00    0.00
0.00    2.00    0.00    2.00    0.00    2.00
3.00    0.00    3.00    0.00    3.00    0.00
0.00    4.00    0.00    4.00    0.00    4.00
5.00    0.00    5.00    0.00    5.00    0.00
0.00    6.00    0.00    6.00    0.00    6.00
inverse:
Error: Inverse doesn't exist
Error log: The rank of source matrix is not full, inverse calculation interrupted.
Current operation interrupted, please check and try again.

Error: NULL pointer exception
Error log: The pointer to source matrix is null, print process interrupted.
Current operation interrupted, please check and try again.

determinant: 0
```

转置与秩

```c
//Benchmark.c (main function)
int main()
{
    Matrix *A=create_from_file("matfile",6,6);
    printf("A:\n");
    print_matrix(A,0);
    Matrix *T=transpose(A);
    printf("T:\n");
    print_matrix(T,0);
    printf("The rank of A is: %d\nThe rank of T is: %d\n",rank(A),rank(T));
}
```

Result:

```
gutao@FIRST-MICROSOFT:/mnt/e/Cpp/Project03$ make && ./matlib
[100%] Built target matlib
A:
1       0       1       0       1       0
0       2       0       2       0       2
3       0       3       0       3       0
0       4       0       4       0       4
5       0       5       0       5       0
0       6       0       6       0       6
T:
1       0       3       0       5       0
0       2       0       4       0       6
1       0       3       0       5       0
0       2       0       4       0       6
1       0       3       0       5       0
0       2       0       4       0       6
The rank of A is: 2
The rank of T is: 2
```

Testcase #7 鲁棒性测试

创建矩阵

```
//Benchmark.c (main function)
int main()
{
    //申请不合法空间
    Matrix *too_large=create_empty(10000,10001);
    Matrix *too_small=create_empty(0,100);
    //空数组作为数据源
    TYPE *arr=NULL;
    Matrix *null_src=create_from_array(arr,5,5);
    //试图拷贝null数组
    Matrix *copy_null=create_copy(NULL);
    //错误格式的字符串作为数据源
    Matrix *wrong_str=create_from_string("1,2,3;4,5;",3,2);
    //不存在的文件作为数据源
    Matrix *file_404=create_from_file("file.404",3,3);
    //截取超出原矩阵的子矩阵
    Matrix *ori=create_full(2,2,2);
    Matrix *sub=sub_matrix(ori,1,1,3,3);
}
```

Result:

```
gutao@FIRST-MICROSOFT:/mnt/e/Cpp/Project03$ make && ./matlib
Consolidate compiler generated dependencies of target matlib
[100%] Built target matlib
Error: Illegal matrix size
Error log: The maximum size of matrix is row*col=1e8.
Current operation interrupted, please check and try again.

Error: Illegal matrix size
Error log: Row and Col should be positive integers.
Current operation interrupted, please check and try again.

Error: NULL pointer exception
Error log: The source array is null, creating process interrupted.
Current operation interrupted, please check and try again.

Error: NULL pointer exception
Error log: The source matrix is null, copy process interrupted.
Current operation interrupted, please check and try again.

Error: Illegal matrix shape
Error log: The number of col mismatch the given col, matrix creation process interrupted.
Current operation interrupted, please check and try again.

Error: NULL pointer exception
Error log: Source file not found, matrix creation process interrupted.
Current operation interrupted, please check and try again.

Error: Illegal row index
Error log: The ending index of row exceeds the max row index of source matrix.
Current operation interrupted, please check and try again.
```

(*每个函数都内置了空指针报错，后续不作重复展示)

矩阵级别操作

```
// Benchmark.c (main function)
int main()
{
    //二次删除
    Matrix *A = create_full(2, 2, 2);
    Matrix *B = create_full(3, 3, 3);
    Matrix *C = create_full(5, 5, 5);
    delete_matrix(&A);
    delete_matrix(&A);
    // null作为源的拷贝
    copy_matrix(&B, A);
    printf("B:\n");
    print_matrix(B, 0);
    //覆盖null的拷贝
    copy_matrix(&A, B);
    printf("A:\n");
    print_matrix(A, 0);
    //大小不匹配时的警告
    copy_matrix(&A, C);
    printf("A:\n");
    print_matrix(A,0);
    //按列拼合两个列数不同的矩阵
    Matrix *cat=col_concat(A,B);
}
```

Result:

```
gutao@FIRST-MICROSOFT:/mnt/e/Cpp/Project03$ make && ./matlib
Consolidate compiler generated dependencies of target matlib
[100%] Built target matlib
Error: NULL pointer exception
Error log: The pointer to the matrix is null, delete process interrupted.
Current operation interrupted, please check and try again.

Error: NULL pointer exception
Error log: The pointer to source matrix is null, copy process interrupted.
Current operation interrupted, please check and try again.

B:
3       3       3
3       3       3
3       3       3
Warning: Copy into null matrix
Details: The pointer to destination matrix is null, the pointer will point to a copy matrix of source matrix.
A:
3       3       3
3       3       3
3       3       3
Warning: Copy into matrix of different sizes
Details: The sizes of two matrices are different, the data of destination matrix will be covered by source matrix.
A:
5       5       5       5       5
5       5       5       5       5
5       5       5       5       5
5       5       5       5       5
5       5       5       5       5
Error: Illegal matrix shape
Error log: The number of col of source matrices should be the same, concat process interrupted.
Current operation interrupted, please check and try again.
```

矩阵运算(修改传入值)

```
// Benchmark.c (main function)
int main()
{
    //被加数为空
    Matrix *A=NULL;
    Matrix *B=create_full(3,4,5);
    printf("A=0+B=\n");
    add_by(&A,B);
    print_matrix(A,0);
    //被减数为空
    B=NULL;
    subtract_by(&B,A);
    printf("B=0-A=\n");
    print_matrix(B,0);
}
```

Result:

```
gutao@FIRST-MICROSOFT:/mnt/e/Cpp/Project03$ make && ./matlib
Consolidate compiler generated dependencies of target matlib
[100%] Built target matlib
A=0+B=
Warning: NULL pointer warning
Details: The pointer to augend matrix is null, a copy of addend matrix will be assigned to augend matrix instead.
5       5       5       5
5       5       5       5
5       5       5       5
Warning: NULL pointer warning
Details: The pointer to minuend matrix is null, a negative copy of subtrahend matrix will be assigned to minuend matrix instead.
B=0-A=
-5      -5      -5      -5
-5      -5      -5      -5
-5      -5      -5      -5
```

矩阵计算

```
// Benchmark.c (main function)
int main()
{
    Matrix *A = create_full(3, 2, 2);
    Matrix *B = create_full(3, 3, 3);
    //不同形状的矩阵相加
    Matrix *C = matrix_add(A, B);
    //不合法的矩阵相乘
    C = matrix_multiply(A, B);
    //非方阵的矩阵幂
    C = matrix_pow(A, 2);
    //逆不存在时的负数幂
    C=matrix_pow(B,-2);
}
```

Result:

```
gutao@FIRST-MICROSOFT:/mnt/e/Cpp/Project03$ make && ./matlib
Consolidate compiler generated dependencies of target matlib
[100%] Built target matlib
Error: Illegal matrix shape
Error log: The shape of augend and addend matrices are different, add process interrupted.
Current operation interrupted, please check and try again.

Error: Illegal matrix shape
Error log: The col number of multiplicand matrix should equal to row number of multiplier matrix.
Current operation interrupted, please check and try again.

Error: Illegal matrix shape
Error log: The base matrix should be square matrix.
Current operation interrupted, please check and try again.

Error: Inverse doesn't exist
Error log: The rank of source matrix is not full, inverse calculation interrupted.
Current operation interrupted, please check and try again.

Error: NULL pointer exception
Error log: The source matrix is null, copy process interrupted.
Current operation interrupted, please check and try again.

Error: NULL pointer exception
Error log: The source matrix has no inverse, negative power calculation interrupted.
Current operation interrupted, please check and try again.
```

矩阵变换

```c
// Benchmark.c (main function)
int main()
{
    Matrix *A = create_full(3, 2, 2);
    Matrix *B = create_full(3, 3, 3);
    //非方阵求逆
    Matrix *I=inverse(A);
    //非方阵求行列式
    printf("detA=%f\n",determinant(A));
}
```

Result:

```
gutao@FIRST-MICROSOFT:/mnt/e/Cpp/Project03$ make && ./matlib
Consolidate compiler generated dependencies of target matlib
[100%] Built target matlib
Error: Illegal matrix shape
Error log: The matrix should be square to have inverse, inverse calculation interrupted.
Current operation interrupted, please check and try again.

Error: Illegal matrix shape
Error log: Determinant exists only when the matrix is square, determinant calculation interrupted.
Current operation interrupted, please check and try again.

detA=nan
```

Part 4 - Difficulties & Solutions

1. 安全、报错与警告

需求：由于项目需要实现一个安全的库，不至于用户正常操作导致崩溃，因此对于可能的危险操作要进行检查、报错和处理。

解决方案：

标准库中，一个函数有时会有对应的`safe`版本，以效率作为代价提高安全性。本项目根据情景需求，将所有函数按`safe`标准编写，对于每个函数都内置内存管理、指针操作等细节，用户直接调用函数即可。

由于C并不能便捷地通过`throw catch`来捕获错误，本项目中在每个函数中，在进行函数操作前都使用判断语句检查安全性，若有错误则及时返回并报错/警告，报错/警告通过综合的函数实现。同时，丰富的函数接口覆盖了用户对于矩阵操作的需求，可以便利地使用，避免了用户直接对矩阵进行内存管理而可能导致的高危隐患。

2. 移植性与拓展性

需求：理工科在不同操作系统进行矩阵运算的情境下，经常需要对整个矩阵进行一元/二元的各种类型的运算，如果按照传统的方式，对于每一种运算都实现一个函数，则会导致库的冗余重复，调用时也很麻烦。

解决方案：

可移植方面，本项目依据标准库模式使用`size_t`类型记录下标，既避免了负下标可能导致的段错误，也保证了不同位数操作系统下的一致性。

数据类型方面，囿于C语言的限制，本项目以宏丐版代替模板，模拟了类模板的效果，更改`TYPE`后只需要简单修改细节即可适用于不同类型的数据。

矩阵运算方面，本项目实现支持自定义运算的矩阵运算函数，对于用户自定义的任意一元/二元运算，只需要将运算的函数指针作为参数传入库内的函数，则可以对矩阵逐元素进行自定义运算，扩展性良好~~，不能重载运算符的确让人挺难受的~~。

3. 复杂函数的实现

需求：求行列式，求秩，求逆这三个问题与上三角化均有绑定关系，但上三角化的实现较为复杂。

解决方案：回去翻了线性代数教材，手动模拟了几次$Gauss\ Jordan\ Elimination$，然后将模拟的过程在2h的debug后码出来了，前三个问题也迎刃而解，实现这个之后，矩阵的若干种其他分解要实现也很轻松了。

Part 5 - Summary

感谢您能读到这里，报告为了尽力展示项目全貌略显冗长，下次改正，感谢理解（磕头

和前两次project不同，这次笔者先没有直接开写，而是首先观摩了GitHub上Amoiensis的Matrix_hub项目，了解了矩阵运算常用的需求，对照题目构思了可能可以实现的功能以及相比他的项目我可以做出的改进。也因此，在项目的实现中并非想到一个函数就写一个，加入了很多扩展性的内容。

本次project对于安全性的要求较高，笔者也因此再次加强了对程序鲁棒性的要求，在没有`try catch`的帮助下进行`error handling`确实是个技术活，在编写过程中也有参考大家讨论中提出的异常情况来进行优化。

感觉这次project的主要难点在要自己给自己出难题(实际上这学期的project都有这个成分在的)，对于加深C的理解还是很有帮助的，不过对于初学者来说可能是个不小的挑战吧(笑)。