CS205 C/C++ Programming - Projetct03

姓名：秦世华

SID：11812309

git hub link: https://github.com/Kishore4399/C-C-Project.git

The *struct* for matrices

```
typedef struct Matrix {
    INT row;
    INT column;
    DATA_TYPE *data;
} Matrix;
```

define.h

在define.h文件中定义了自定义了数据类型，精度，内存分配，内存释放，以及exception相关的处理，代码内容如下

```
#ifndef  __DEFINE_H__
#define  __DEFINE_H__
typedef float DATA_TYPE;
typedef int INT;
typedef void VOID;
#define PRECISION "%.5lf\t"
#define MALLOC(n, type) \
         ((type *)malloc( (n) * sizeof(type)))
#define FREE(p)             \
         if(p!=NULL)         \
         {                   \
               free(p);       \
               p = NULL;     \
         }
#define ERROR_INPUT_POINTER printf("ERROR: input data pointer error
\n")
#define ERROR_INPUT_INPUTPARA printf("ERROR:input paramter error
\n")
#define ERROR_MEM_ALLOCATE printf("ERROR:failed to allocate memeory
\n")
#define ERROR_SIZE_MATCH printf("ERROR:matrix size does not match
\n")
#endif
```

createMatrix()

```
Matrix* createMatrix(INT row, INT column, INT elenum, DATA_TYPE*
data)
{
    //Generate Matrix Struct
    //Please remember to free the memory due to dynamic allocate
    if (row <= 0 || column <= 0)
    {
        ERROR_INPUT_INPUTPARA;
        return NULL;
    }
    if (data == NULL)
```

```
    {
        ERROR_INPUT_POINTER;
        return NULL;
    }
    INT mat_size = row * column;

    if (mat_size == elenum)
    {
        Matrix* mat = MALLOC(1, Matrix);
        mat->row = row;
        mat->column = column;
        mat->data = MALLOC(mat_size, DATA_TYPE);
        if (mat == NULL || mat->data == NULL)
        {
            free(mat);
            mat = NULL;
            free(mat->data);
            mat->data = NULL;
            ERROR_MEM_ALLOCATE;
            return NULL;
        }
        INT i;
        for (i = 0; i < mat_size; i++)
        {
            mat->data[i] = data[i];
        }
        return mat;
    }
    else
    {
        printf("ERROR: the number of data does not match the matrix
size \n");

        return NULL;
    }

}
```

分析和反思：

在创建矩阵时考虑到用户输入不规范的问题，包含两类异常处理，ERROR_INPUT_INPUTPARA; ERROR_INPUT_POINTER; 判断行列是否大于零，以及数据指针是否为空。

为了确保矩阵能够成功被创建，使用if (mat == NULL || mat->data == NULL)，来判断矩阵是否创建成功

deleteMatrix()

```
VOID deleteMatrix(Matrix* mat)
{
    if (mat == NULL)
    {
        ERROR_INPUT_POINTER;
        return;
    }
    FREE(mat);
}
```

分析和反思：

闯入待删除矩阵的指针，需要判断传入指针是否为空，若为空指针，抛出ERROR_INPUT_POINTER。

copyMatrix()

部分说明代码：

```
Matrix* copyMatrix(const Matrix* mat_src)
{
    //Copy Matrix
    if (mat_src == NULL)
    {
       ERROR_INPUT_POINTER;
       return NULL;
    }
    INT elenum = mat_src->row * mat_src->column;
    Matrix* mat_copy = createMatrix(mat_src->row, mat_src->column,
elenum, mat_src->data);

    return mat_copy;
}
```

分析和反思：

复制矩阵前同样需要检查传入的指针是否为空，同时为了确保闯入的原始矩阵不被修改，在函数参数部分加上const。同时在copyMatrix()中直接调用creatMatrix()。

addMatrix()

部分说明代码

```
VOID array_sum(const DATA_TYPE* a, const DATA_TYPE* b, DATA_TYPE*
sum, INT length)
{
    if (a == NULL || b == NULL || sum == NULL)
    {
        ERROR_INPUT_POINTER;
        return;
    }
    INT  i;
    for (i = 0; i < length; i++)
    {
        sum[i] = a[i] + b[i];
    }
    return;
}
Matrix* addMatrix(const Matrix* A, const Matrix* B)
{
```

```
    if (A == NULL || B == NULL)
    {
        ERROR_INPUT_POINTER;
        return NULL;
    }
    if (A->column != B->column || A->row != B->row)
    {
        ERROR_SIZE_MATCH;
        return NULL;
    }
    INT elenum = A->row * A->column;
    DATA_TYPE sum[elenum];
    array_sum(A->data, B->data, sum, elenum);
    Matrix* mat_sum =  createMatrix(A->row, A->column, elenum,
sum);
    return mat_sum;
}
```

分析与反思：

两个矩阵相加时需要保证两个矩阵的行和列的长度相等，因此本函数包含if (A->column != B->column || A->row != B->row)的判断。在对数据做加法时，需要保证原始数据不会被改变，因此函数的返回值是一个新矩阵的指针

subtractMatrix()

部分说明代码：

```
VOID array_sub(const DATA_TYPE* a, const DATA_TYPE* b, DATA_TYPE*
sub, INT length)
{
    if (a == NULL || b == NULL || sub == NULL)
    {
        ERROR_INPUT_POINTER;
        return;
    }
    INT  i;
    for (i = 0; i < length; i++)
    {
        sub[i] = a[i] - b[i];
    }
```

```
    return;
}
Matrix* subtractMatrix(const Matrix* A, const Matrix* B)
{
    if (A == NULL || B == NULL)
    {
        ERROR_INPUT_POINTER;
        return NULL;
    }
    if (A->column != B->column || A->row != B->row)
    {
        ERROR_SIZE_MATCH;
        return NULL;
    }
    INT elenum = A->row * A->column;
    DATA_TYPE sub[elenum];
    array_sub(A->data, B->data, sub, elenum);
    Matrix* mat_sub = createMatrix(A->row, A->column, elenum, sub);

    return mat_sub;
}
```

addScalar() --- add a scalar to a matrix.

部分代码说明：

```
Matrix* addscalar(const Matrix* A, const DATA_TYPE b)
{
    if (A == NULL)
    {
        ERROR_INPUT_POINTER;
        return NULL;
    }
    Matrix* C = copyMatrix(A);
    INT  i;
    for (i = 0; i < (A->column * A->row); i++)
    {
        C->data[i] = C->data[i] + b;
    }
    return C;
```

```
}
```

分析与反思

为了避免修改原矩阵的内容，在做scalar之前 我们先用到上面创建的copyMatrix(),得到输出矩阵

subScalar() --- subtract a scalar from a matrix.

部分代码说明：

```
Matrix* subScalar(const Matrix* A, const DATA_TYPE b)
{
    if (A == NULL)
    {
        ERROR_INPUT_POINTER;
        return NULL;
    }
    Matrix* C = addScalar(A, -1 * b);
    return C;

}
```

分析与反思

subscalar()中内嵌addscalar()函数，即传入addscalar()中的scalar 乘以-1

multiplyScalar() --- multiply a matrix with a scalar.

部分代码说明：

```
Matrix* multiplyScalar(const Matrix* A, const DATA_TYPE b)
{
    if (A == NULL)
    {
        ERROR_INPUT_POINTER;
        return NULL;
    }
    Matrix* C = copyMatrix(A);
    INT  i;
```

```
    for (i = 0; i < (A->column * A->row); i++)
    {
        C->data[i] = C->data[i] * b;
    }
    return C;
}
```

mulMatrix() --- multiply two matrices.

部分代码说明：

```
Matrix* mulMatrix(const Matrix* A, const Matrix* B)
{
    if (A == NULL || B == NULL)
    {
        ERROR_INPUT_POINTER;
        return NULL;
    }
    if (A->column != B->row)
    {
        ERROR_SIZE_MATCH;
        return NULL;
    }
    INT i, j, k, size;
    size = A->row * B->column;
    DATA_TYPE c_data[size];
    INT C_index = 0;
    DATA_TYPE C_element = 0;
    for(i=0; i<A->row;i++){
        for(j=0;j<B->column;j++){
            for(k=0; k<A->column; k++){
                C_element += A->data[(i*A->column) + k] * B-
>data[(k*B->column) + j];
            };
            c_data[C_index] = C_element;
            C_index += 1;
            C_element = 0;
        };
    };
    Matrix* C = createMatrix( A->row, B->column, size, c_data);
```

```
    return C;
}
```

分析与反思

两个矩阵相乘时需要保证前一个矩阵的列数等于后一个矩阵的行数

maxelem() --- find the maximal values of a matrix.

部分代码说明：

```
DATA_TYPE maxelem(const Matrix* A)
{
    if (A == NULL)
    {
        ERROR_INPUT_POINTER;
        return NULL;
    }
    DATA_TYPE max = A->data[0];
    INT i;
    for ( i = 0; i < (A->column * A->row); i++)
    {
        if (max < A->data[i])
        {
            max = A->data[i];
        }
    }
    return max;
}
```

minelem() --- find the minimal values of a matrix.

部分代码说明：

```
DATA_TYPE minelem(const Matrix* A)
{
    if (A == NULL)
    {
        ERROR_INPUT_POINTER;
        return NULL;
```

```
    }
    DATA_TYPE min = A->data[0];
    INT i;
    for ( i = 0; i < (A->column * A->row); i++)
    {
        if (min > A->data[i])
        {
            min = A->data[i];
        }
    }
    return min;
}
```

transpMatrix() --- get transposed matrix.

部分代码说明：

```
Matrix* transpMatrix(const Matrix* A)
{
    if (A == NULL)
    {
        ERROR_INPUT_POINTER;
        return NULL;
    }
    INT i, j, size;
    size = A->row * A->column;
    DATA_TYPE B_data[size];
    INT B_index = 0;
    for(i=0; i<A->column; i++){
        for(j=0; j<A->row; j++){
            B_data[B_index] = A->data[j*A->column + i];
            B_index += 1;
        }
    }
    Matrix* B = createMatrix( A->column, A->row, size, B_data);
    return B;

}
```

分析与反思

用于输出矩阵的转置：矩阵是一种变换（映射），矩阵的转置就是这个变换其中关于旋转和镜射的部分有所改变，变为原来的反方向

identityMatrix() --- create identity matrix.

部分代码说明：

```
Matrix* transpMatrix(const Matrix* A)
{
    if (A == NULL)
    {
        ERROR_INPUT_POINTER;
        return NULL;
    }
    INT i, j, size;
    size = A->row * A->column;
    DATA_TYPE B_data[size];
    INT B_index = 0;
    for(i=0; i<A->column; i++){
        for(j=0; j<A->row; j++){
            B_data[B_index] = A->data[j*A->column + i];
            B_index += 1;
        }
    }
    Matrix* B = createMatrix( A->column, A->row, size, B_data);
    return B;

}
```

分析与反思

获取设定长度的单位矩阵。在矩阵的乘法中，单位矩阵起着特殊的作用，如同数的乘法中的1，这种矩阵被称为单位矩阵。它是个方阵，从左上角到右下角的对角线（称为主对角线）上的元素均为1。除此以外全都为0。根据单位矩阵的特点，任何矩阵与单位矩阵相乘都等于本身，而且单位矩阵因此独特性在高等数学中也有广泛应用。

全代码演示：

```
kishore@DESKTOP-9IRFFAD:/mnt/e/Ccourse/lab01/project03/src$ ./demo
test
****************************
create a Matrix
5.00000 2.00000 8.00000
4.00000 2.00000 5.00000
****************************
copy a Matrix
5.00000 2.00000 8.00000
4.00000 2.00000 5.00000
****************************
add two Matrices
10.00000        4.00000 16.00000
8.00000 4.00000 10.00000
****************************
subtrucy two Matrices
5.00000 2.00000 8.00000
4.00000 2.00000 5.00000
****************************
matrix add scalar(6.6)
11.60000        8.60000 14.60000
10.60000        8.60000 11.60000
****************************
matrix subtract scalar(6.6)
-1.60000        -4.60000        1.40000
-2.60000        -4.60000        -1.60000
****************************
matrix mutiple scalar(2.0)
10.00000        4.00000 16.00000
8.00000 4.00000 10.00000
****************************
find the minimal and maximal values of a matrix
max value = 8.000000, min value = 2.000000
****************************
multiply two matrices with unmatched size.
ERROR:matrix size does not match
ERROR: input data pointer error
****************************
transpose matrices.
5.00000 4.00000
```

```
2.00000 2.00000
8.00000 5.00000
*****************************
multiply two matrices after transposing a Matrix.
93.00000        64.00000
64.00000        45.00000
*****************************
create an identity Matrix.
1.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 1.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 1.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 1.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 1.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 1.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 1.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 1.00000
```