```
/**
  ****************************************************************************************
  * @file    : main.cpp
  * @brief   : Main Program
  *          : Lab 3: Matrix
  *          : CS-3210/021
  * @date    : MAR 30 2021
  * @author : Julian Singkham
  ****************************************************************************************
  * @attention
  *
  * This program handles the creation of a matrix and the +,=, and * operators.
  *
  *
  ****************************************************************************************
**/
#include<iostream>
#include "matrix.h"

using namespace std;
//======================================Methods=========================================
 /**
  * @brief The program entry point. Assume tests are successful unles otherwise stated
  * by "Failed" or error is thrown.
  *
  * @param NOT USED
  *
  * @retval NOT USED
  */
int main(){
    // Test malforned matrix creation
    cout << "Test malforned matrix creation" << endl;
    try{
        matrix m1(0,-2);
        cout << "Failed" << endl;
    }
    catch(matrix_Exception& e){
        cout << e.what() << endl;
    }
    cout << endl;
    cout << endl;

    // Test proper matrix creation
    cout << "Test proper matrix creation" << endl;
    matrix m_out(2,2);
    try{
        cout << m_out << endl;
        if(m_out[0][0] != 0 && m_out[0][1] != 0 && m_out[1][0] != 0
           && m_out[0][0] != 0)
            cout << "Failed" << endl;
    }
    catch(matrix_Exception& e){
        cout << e.what() << endl;
    }
    cout << endl;
    cout << endl;

    // Test malforned matrix multiplication
    cout << "Test malforned matrix multiplication" << endl;
    matrix m1(2,2);
    m1[0][0] = 0;
    m1[0][1] = 1;
    m1[1][0] = 2;
    m1[1][1] = 3;
    cout << m1 << endl;

    matrix m2(3,2);
    m2[0][0] = 4;
    m2[0][1] = 5;
    m2[1][0] = 6;
    m2[1][1] = 7;
```

```cpp
    m2[2][0] = 8;
    m2[2][1] = 9;
    cout << m2 << endl;

    m_out = matrix::identity(3);

    try{
        m_out = m1*m2;
        cout << "Failed" << endl;
    }
    catch(matrix_Exception& e){
        cout << e.what() << endl;
    }
    cout << endl;
    cout << endl;

    //Test proper matrix multiplication
    cout << "Test proper matrix multiplication" << endl;
    matrix m3(2,3);
    m3[0][0] = 0;
    m3[0][1] = 1;
    m3[0][2] = 2;
    m3[1][0] = 3;
    m3[1][1] = 4;
    m3[1][2] = 5;
    cout << m3 << endl;

    cout << m2 << endl;

    try{
        m_out = m3*m2;
        cout << m_out << endl;
        if(m_out[0][0] != 6 && m_out[0][1] != 7 && m_out[1][0] != 36
           && m_out[0][0] != 43)
            cout << "Failed" << endl;
    }
    catch(matrix_Exception& e){
        cout << e.what() << endl;
    }
    cout << endl;
    cout << endl;

    //Test improper + operator
    cout << "Test improper + operator" << endl;
    cout << m1 << endl;
    cout << m2 << endl;
    cout << m3 << endl;
    try{
        m_out = m1 + m3;
        cout << "Failed" << endl;
    }
    catch(matrix_Exception& e){
        cout << e.what() << endl;
    }
    cout << endl;
    cout << endl;

    //Test proper + operator
    cout << "Test proper + operator" << endl;
    cout << m1 << endl;
    m2 = matrix::identity(2);
    cout << m2 << endl;

    try{
        m_out = m1 + m2;
        cout << m_out << endl;

        if(m_out[0][0] != 1 && m_out[0][1] != 1 && m_out[1][0] != 2
           && m_out[0][0] != 4)
            cout << "Failed" << endl;
    }
    catch(matrix_Exception& e){
        cout << e.what() << endl;
    }
    cout << endl;
    cout << endl;

    //Test scale multiplication
    cout << "Test scale multiplication of 5" << endl;
    cout << m1 << endl;

    m_out = m1 * 5;
    cout << m_out << endl;
    if(m_out[0][0] != 0 && m_out[0][1] != 5 && m_out[1][0] != 10
       && m_out[0][0] != 15)
        cout << "Failed" << endl;
    cout << endl;
    cout << endl;
}
```

```
/**
  ***********************************************************************************
  * @file    : matrixm.cpp
  * @brief   : matrix multiplier
  *          : Lab 2: Arrays and Matrices
  *          : CS-3210/021
  * @date    : MAR 23 2021
  * @author  : Julian Singkham
  ***********************************************************************************
  * @attention
  *
  * This API handles the creation, deletion, and =,+,* operators of a matrix as well
  * as retrieving matrix values using []. In essence the double matrix used in this
  * API is an array of arrays where the **double matrix points to rows *double arrays
  * that then point to col elements.
  *
  ***********************************************************************************
**/
#include "matrix.h"
#include <string>
#include <cmath>

//=======================================Public======================================
/**
 * @brief Makes the insertion operator a friend so it can access matrix.
 * Basically allows the matrix to be printed to std.
 *
 * @param os: Stream to write to.
 * @param rhs: Reference to the matrix that is being printed.
 *
 * @retval Stream containing the matrix printout.
 */
std::ostream& operator<<(std::ostream& os, const matrix& rhs){
    for(int i = 0; i < rhs.rows; i++){
        os << "|";
        for(int j = 0; j < rhs.cols; j++){
            double temp = rhs.the_matrix[i][j];
            os << temp << "|";
        }
        os << std::endl;
    }
    return os;
}

/**
 * @brief Matrix constructor, it creates a matrix of given dimensions
 * with clear (zeroed) cells. Throws error if dimensions are not possible (<1).
 *
 * @param rows: How many rows in the matrix.
 * @param cols: How many columns in the matrix.
 *
 * @retval NONE
 */
matrix::matrix(unsigned int rows, unsigned int cols) : rows(rows), cols(cols){
    if(rows < 1 || cols < 1)
        throw matrix_Exception("p-constructor has bad arguments.");

    the_matrix = new double*[rows];//Allocates memory to the # of rows

    //Allocate memory for each row of the array to the # of columns
    //Basically creates a 1-D array of 1-D arrays
    for(int i = 0; i < rows; i++)
        the_matrix[i] = new double[cols];

    clear(); //Fill matrix with zeroes
}

/**
 * @brief Copy constructor that makes a new matrix from a given one.
 *
 * @param from: matrix to copy into the new matrix.
 *
 * @retval A copy of the given matrix.
 */
matrix::matrix(const matrix& from) : rows(from.rows), cols(from.cols){
    the_matrix = new double*[rows];//Allocates memory to the # of rows

    //Allocate memory for each row of the array to the # of columns
    //Basically creates a 1-D array of 1-D arrays
    for(int i = 0; i < rows; i++)
        the_matrix[i] = new double[cols];

    //Copy values from "from" into new matrix
    for(int i = 0; i < rows; i++)
        for(int j = 0; j < cols; j++)
            the_matrix[i][j] = from[i][j];
}

/**
 * @brief Frees allocated memory form matrix
 *
 * @param: NONE
 *
 * @retval NONE
 */
matrix::~matrix(){
    for(int i = 0; i <  rows; i++)
        delete[] the_matrix[i]; //Delete each row of the matrix
    delete[] the_matrix; //Delete the matrix itself
}

/**
 * @brief Named constructor, it creates a square identity matrix
 * of given size.
 *
 * Identity matrix is a matrix that is all zeros expect when
 * row#=col#, then it is 1.
 * // 1 0 0 [0][0] = 1
 * // 0 1 0 [1][1] = 1
 * // 0 0 1 [2][2] = 1
 *
 * @param size: Square dimensions of the matrix.
 *
 * @retval The square identity matrix.
 */
matrix matrix::identity(unsigned int size){
    if(size == 0)
        throw matrix_Exception("Can not create an identity matrix of size 0.");

    matrix return_matrix(size, size);
    for(int i = 0; i < size; i++)
        return_matrix[i][i] = 1.0;
    return return_matrix;
}

/**
 * @brief Assigns the matrix to the value stored in the given matrix.
 *
 * @param rhs: The given matrix to copy from.
 *
 * @retval A copy of the given matrix.
 */
matrix& matrix::operator=(const matrix& rhs){
    //Verify matrices match in size
    if(rows != rhs.rows || cols != rhs.cols){
        for(int i = 0; i <  rows; i++)
            delete[] the_matrix[i]; //Delete each row of the matrix
        delete[] the_matrix; //Delete the matrix itself

        rows = rhs.rows;
        cols = rhs.cols;
        the_matrix = new double*[rows];//Allocates memory to the # of rows

        //Allocate memory for each row of the array to the # of columns
        //Basically creates a 1-D array of 1-D arrays
        for(int i = 0; i < rows; i++)
            the_matrix[i] = new double[cols];
    }
    //Copy values from rhs into current matrix
    for(int i = 0; i < rows; i++)
        for(int j = 0; j < cols; j++)
            the_matrix[i][j] = rhs[i][j];

    return *this;
}

/**
 * @brief Matrix addiiton. The lhs and rhs must be the same size.
 *
 * @param rhs: The right hand side matrix.
 *
 * @retval The resulting matrix after addition.
 */
matrix matrix::operator+(const matrix& rhs) const{
    //Verify matrices match in size
    if(rows != rhs.rows || cols != rhs.cols)
        throw matrix_Exception("Size mismatch – The column/row of the left matrix does"
                              " not match the column/row of the right matrix:");

    matrix return_matrix(rows, cols);

    for(int i = 0; i < rows; i++)
        for(int j = 0; j < cols; j++)
            return_matrix[i][j] = the_matrix[i][j] + rhs[i][j];

    return return_matrix;
}

/**
 * @brief Matrix multiplication.
 * The lhs column size and rhs row size must match.
 *
 * @param rhs: The right hand side matrix.
 *
 * @retval The resulting matrix after multiplication.
 * Dimension: lhs.rows x rhs.cols.
 */
matrix matrix::operator*(const matrix& rhs) const{
    //Verify matrices match in size
    if(cols != rhs.rows)
        throw matrix_Exception("Size mismatch – The column of the left matrix does"
                              " not match the row of the right matrix:");

    matrix return_matrix(rows, rhs.cols);

    for (int i = 0; i < rows; ++i)
        for (int j = 0; j < rhs.cols; ++j)
            for (int k = 0; k < rhs.cols; ++k)
                return_matrix[i][j] += the_matrix[i][k] * rhs[k][j];
    return return_matrix;
}

/**
 * @brief Matrix scaler multiplication.
 * This only supports matrix * 5, not 5 * matrix.
 *
 * @param scale: Value to scale the matrix.
 *
 * @retval The scaled matrix.
 */
matrix matrix::operator*(const double scale) const{
    matrix return_matrix(rows, cols);

```

```cpp
    for (int i = 0; i < rows; ++i)
        for (int j = 0; j < cols; ++j)
            return_matrix[i][j] = the_matrix[i][j] * scale;
    return return_matrix;
}

/**
 * @brief This allows access of the matrix elements by using [].
 *
 * @param row: The desired row of the matrix.
 *
 * @retval A pointer to the desired element of the matrix.
 */
double* matrix::operator[](unsigned int row){
    //Verify row is within bounds
    if (row >= rows || row < 0)
        throw matrix_Exception("Size mismatch – The requested row is out of bounds.");

    double *ret = the_matrix[row];
    return ret;
}

/**
 * @brief This allows access of the matrix elements by using [].
 * Const version
 *
 * @param row: The desired row of the matrix.
 *
 * @retval A pointer to the desired element of the matrix.
 */
double* matrix::operator[](unsigned int row) const{
    //Verify row is within bounds
    if (row >= rows || row < 0)
        throw matrix_Exception("Size mismatch – The requested row is out of bounds.");

    double *ret = the_matrix[row];
    return ret;
}

/**
 * @brief Zeroes the elements of the matrix.
 *
 * @param: None
 *
 * @retval None
 */
void matrix::clear(){
    for(int i = 0; i < rows; i++)
        for(int j = 0; j < cols; j++)
            the_matrix[i][j] = 0.0;
    return;
}

//=======================================Global======================================
/**
 * @brief Matrix scaler multiplication.
 * This only supports matrix * 5, not 5 * matrix.
 *
 * @param scale: Value to scale the matrix.
 * @param rhs: The matrix to apply the scaling.
 *
 * @retval The scaled matrix.
 */
matrix operator*(const double scale, const matrix& rhs){
    matrix return_matrix(rhs);
    return_matrix = rhs * scale;
    return return_matrix;
}

```

# Makefile

```
CC = g++
CFLAGS = -c -MMD
LFLAGS = -Wall -Wextra -g
LDFLAGS ?= -lglut -lGLU  -lGL
SOURCES = $(wildcard *.cpp)
OBJECTS = $(SOURCES:.cpp=.o)
EXECUTABLE = ex

all: $(EXECUTABLE) clean
$(EXECUTABLE): $(OBJECTS)
    $(CC) $(LFLAGS) -o $@ $(OBJECTS) $(LDFLAGS)

%.o:%.cpp
    $(CC) $(CFLAGS) $<

-include *.d

clean:
    rm -f *.d
    rm -f *.o
```

**Table of Contents**