**Lab Description:**

\*\*In README file

\*\*Not on lab write up

\*\* Scalar bonus complete!!!!

**Graph:**

**Sample Output:**

Testing printMatrix Function

5 3 3

1 1 2

1 5 2

10 6 6

2 2 4

2 10 4

Testing Overloaded Constructor

Creating 5x4 matrix

2 2 5 2

4 3 5 2

2 4 3 3

5 5 5 2

2 2 3 3

Testing Scalar multiplication overloaded multiplication (BONUS)

Test3

3 2 1

4 1 2

5 1 4

Multiplying test3 by 4

12 8 4

16 4 8

20 4 16

Copy constructor

Test3

3 2 1

4 1 2

5 1 4

Test5 copied over from test3

3 2 1

4 1 2

5 1 4

Overloaded assignment

test6 assigned as test5

3 2 1

4 1 2

5 1 4

Preventing self assignment

test6

3 2 1

4 1 2

5 1 4

test2

2 2 5 2

4 3 5 2

2 4 3 3

5 5 5 2

2 2 3 3

Copy test2 over to test6

test6

2 2 5 2

4 3 5 2

2 4 3 3

5 5 5 2

2 2 3 3

Testing overloaded addition

test7

5 2 1

4 5 4

4 5 4

test8

2 1 2

5 3 1

4 1 4

Result

7 3 3

9 8 5

8 6 8

Testing overloaded subtraction

test7

5 2 1

4 5 4

4 5 4

test8

2 1 2

5 3 1

4 1 4

Result

3 1 -1

-1 2 3

0 4 0

Testing overloaded multiplication

test10

4 5

5 4

test11

1

1

Result

9

9

Testing Diagonal Matrix creation

0 2 5

4 0 3

4 3 0

Testing Triangle Matrix creation

Upper

5 5 5

0 2 5

0 0 3

Lower

1 0 0

5 3 0

3 5 1

Testing Identity Matrix creation

1 0 0

0 1 0

0 0 1

Starting tests of Matrix addition

Tests for Matrices of size 1

Finished at Sun Feb 23 19:33:31 2020

Elapsed time: 9.26e-07s

Tests for Matrices of size 3

Finished at Sun Feb 23 19:33:31 2020

Elapsed time: 1.471e-06s

Tests for Matrices of size 4

Finished at Sun Feb 23 19:33:31 2020

Elapsed time: 2.011e-06s

Tests for Matrices of size 8

Finished at Sun Feb 23 19:33:31 2020

Elapsed time: 4.757e-06s

Tests for Matrices of size 10

Finished at Sun Feb 23 19:33:31 2020

Elapsed time: 6.628e-06s

Tests for Matrices of size 20

Finished at Sun Feb 23 19:33:31 2020

Elapsed time: 2.3105e-05s

Tests for Matrices of size 50

Finished at Sun Feb 23 19:33:31 2020

Elapsed time: 0.000117124s

Tests for Matrices of size 70

Finished at Sun Feb 23 19:33:31 2020

Elapsed time: 0.00026146s

Tests for Matrices of size 150

Finished at Sun Feb 23 19:33:31 2020

Elapsed time: 0.000987404s

Tests for Matrices of size 300

Finished at Sun Feb 23 19:33:32 2020

Elapsed time: 0.00316745s

Tests for Matrices of size 700

Finished at Sun Feb 23 19:33:32 2020

Elapsed time: 0.0233778s

Tests for Matrices of size 1000

Finished at Sun Feb 23 19:33:32 2020

Elapsed time: 0.0362886s

Starting tests of Matrix subtraction

Tests for Matrices of size 1

Finished at Sun Feb 23 19:33:32 2020

Elapsed time: 1.999e-06s

Tests for Matrices of size 3

Finished at Sun Feb 23 19:33:32 2020

Elapsed time: 1.434e-06s

Tests for Matrices of size 4

Finished at Sun Feb 23 19:33:32 2020

Elapsed time: 1.613e-06s

Tests for Matrices of size 8

Finished at Sun Feb 23 19:33:32 2020

Elapsed time: 3.732e-06s

Tests for Matrices of size 10

Finished at Sun Feb 23 19:33:32 2020

Elapsed time: 5.633e-06s

Tests for Matrices of size 20

Finished at Sun Feb 23 19:33:32 2020

Elapsed time: 1.6104e-05s

Tests for Matrices of size 50

Finished at Sun Feb 23 19:33:32 2020

Elapsed time: 8.6444e-05s

Tests for Matrices of size 70

Finished at Sun Feb 23 19:33:32 2020

Elapsed time: 0.000164853s

Tests for Matrices of size 150

Finished at Sun Feb 23 19:33:32 2020

Elapsed time: 0.00078694s

Tests for Matrices of size 300

Finished at Sun Feb 23 19:33:32 2020

Elapsed time: 0.00806028s

Tests for Matrices of size 700

Finished at Sun Feb 23 19:33:32 2020

Elapsed time: 0.0289026s

Tests for Matrices of size 1000

Finished at Sun Feb 23 19:33:32 2020

Elapsed time: 0.0504394s

Starting tests of Matrix multiplication (Random Matrices)

Tests for Matrices of size 1

Finished at Sun Feb 23 19:33:32 2020

Elapsed time: 1.006e-06s

Tests for Matrices of size 3

Finished at Sun Feb 23 19:33:32 2020

Elapsed time: 1.518e-06s

Tests for Matrices of size 4

Finished at Sun Feb 23 19:33:32 2020

Elapsed time: 1.795e-06s

Tests for Matrices of size 8

Finished at Sun Feb 23 19:33:32 2020

Elapsed time: 5.052e-06s

Tests for Matrices of size 10

Finished at Sun Feb 23 19:33:32 2020

Elapsed time: 8.171e-06s

Tests for Matrices of size 20

Finished at Sun Feb 23 19:33:32 2020

Elapsed time: 3.7648e-05s

Tests for Matrices of size 50

Finished at Sun Feb 23 19:33:32 2020

Elapsed time: 0.000499824s

Tests for Matrices of size 70

Finished at Sun Feb 23 19:33:32 2020

Elapsed time: 0.00122481s

Tests for Matrices of size 150

Finished at Sun Feb 23 19:33:32 2020

Elapsed time: 0.0261625s

Tests for Matrices of size 300

Finished at Sun Feb 23 19:33:32 2020

Elapsed time: 0.142612s

Tests for Matrices of size 700

Finished at Sun Feb 23 19:33:34 2020

Elapsed time: 1.77163s

Tests for Matrices of size 1000

Finished at Sun Feb 23 19:33:40 2020

Elapsed time: 5.89695s

Starting tests of Matrix multiplication (Diagonal Matrices)

Tests for Matrices of size 1

Finished at Sun Feb 23 19:33:40 2020

Elapsed time: 2.089e-06s

Tests for Matrices of size 3

Finished at Sun Feb 23 19:33:40 2020

Elapsed time: 1.545e-06s

Tests for Matrices of size 4

Finished at Sun Feb 23 19:33:40 2020

Elapsed time: 1.836e-06s

Tests for Matrices of size 8

Finished at Sun Feb 23 19:33:40 2020

Elapsed time: 5.373e-06s

Tests for Matrices of size 10

Finished at Sun Feb 23 19:33:40 2020

Elapsed time: 8.311e-06s

Tests for Matrices of size 20

Finished at Sun Feb 23 19:33:40 2020

Elapsed time: 7.0349e-05s

Tests for Matrices of size 50

Finished at Sun Feb 23 19:33:40 2020

Elapsed time: 0.000959466s

Tests for Matrices of size 70

Finished at Sun Feb 23 19:33:40 2020

Elapsed time: 0.00123924s

Tests for Matrices of size 150

Finished at Sun Feb 23 19:33:40 2020

Elapsed time: 0.0306972s

Tests for Matrices of size 300

Finished at Sun Feb 23 19:33:40 2020

Elapsed time: 0.173246s

Tests for Matrices of size 700

Finished at Sun Feb 23 19:33:43 2020

Elapsed time: 2.37791s

Tests for Matrices of size 1000

Finished at Sun Feb 23 19:33:54 2020

Elapsed time: 10.7376s

Starting tests of Matrix multiplication (Triangular Matrices upper)

Tests for Matrices of size 1

Finished at Sun Feb 23 19:33:54 2020

Elapsed time: 1.42e-06s

Tests for Matrices of size 3

Finished at Sun Feb 23 19:33:54 2020

Elapsed time: 1.426e-06s

Tests for Matrices of size 4

Finished at Sun Feb 23 19:33:54 2020

Elapsed time: 1.923e-06s

Tests for Matrices of size 8

Finished at Sun Feb 23 19:33:54 2020

Elapsed time: 5.413e-06s

Tests for Matrices of size 10

Finished at Sun Feb 23 19:33:54 2020

Elapsed time: 8.292e-06s

Tests for Matrices of size 20

Finished at Sun Feb 23 19:33:54 2020

Elapsed time: 3.8743e-05s

Tests for Matrices of size 50

Finished at Sun Feb 23 19:33:54 2020

Elapsed time: 0.000616289s

Tests for Matrices of size 70

Finished at Sun Feb 23 19:33:54 2020

Elapsed time: 0.00113857s

Tests for Matrices of size 150

Finished at Sun Feb 23 19:33:54 2020

Elapsed time: 0.0286443s

Tests for Matrices of size 300

Finished at Sun Feb 23 19:33:54 2020

Elapsed time: 0.148388s

Tests for Matrices of size 700

Finished at Sun Feb 23 19:33:58 2020

Elapsed time: 3.29854s

Tests for Matrices of size 1000

Finished at Sun Feb 23 19:34:04 2020

Elapsed time: 6.35468s

Starting tests of Matrix multiplication (Triangular Matrices lower)

Tests for Matrices of size 1

Finished at Sun Feb 23 19:34:04 2020

Elapsed time: 9.73e-07s

Tests for Matrices of size 3

Finished at Sun Feb 23 19:34:04 2020

Elapsed time: 2.11e-06s

Tests for Matrices of size 4

Finished at Sun Feb 23 19:34:04 2020

Elapsed time: 1.848e-06s

Tests for Matrices of size 8

Finished at Sun Feb 23 19:34:04 2020

Elapsed time: 5.504e-06s

Tests for Matrices of size 10

Finished at Sun Feb 23 19:34:04 2020

Elapsed time: 8.291e-06s

Tests for Matrices of size 20

Finished at Sun Feb 23 19:34:04 2020

Elapsed time: 7.5702e-05s

Tests for Matrices of size 50

Finished at Sun Feb 23 19:34:04 2020

Elapsed time: 0.00047289s

Tests for Matrices of size 70

Finished at Sun Feb 23 19:34:04 2020

Elapsed time: 0.00146046s

Tests for Matrices of size 150

Finished at Sun Feb 23 19:34:04 2020

Elapsed time: 0.0371058s

Tests for Matrices of size 300

Finished at Sun Feb 23 19:34:04 2020

Elapsed time: 0.175796s

Tests for Matrices of size 700

Finished at Sun Feb 23 19:34:07 2020

Elapsed time: 2.32686s

Tests for Matrices of size 1000

Finished at Sun Feb 23 19:34:14 2020

Elapsed time: 6.85191s

Starting tests of Matrix multiplication (Identity Matrices)

Tests for Matrices of size 1

Finished at Sun Feb 23 19:34:14 2020

Elapsed time: 1.131e-06s

Tests for Matrices of size 3

Finished at Sun Feb 23 19:34:14 2020

Elapsed time: 1.442e-06s

Tests for Matrices of size 4

Finished at Sun Feb 23 19:34:14 2020

Elapsed time: 1.908e-06s

Tests for Matrices of size 8

Finished at Sun Feb 23 19:34:14 2020

Elapsed time: 5.444e-06s

Tests for Matrices of size 10

Finished at Sun Feb 23 19:34:14 2020

Elapsed time: 8.573e-06s

Tests for Matrices of size 20

Finished at Sun Feb 23 19:34:14 2020

Elapsed time: 5.9247e-05s

Tests for Matrices of size 50

Finished at Sun Feb 23 19:34:14 2020

Elapsed time: 0.00045346s

Tests for Matrices of size 70

Finished at Sun Feb 23 19:34:14 2020

Elapsed time: 0.00117639s

Tests for Matrices of size 150

Finished at Sun Feb 23 19:34:14 2020

Elapsed time: 0.0293789s

Tests for Matrices of size 300

Finished at Sun Feb 23 19:34:14 2020

Elapsed time: 0.169648s

Tests for Matrices of size 700

Finished at Sun Feb 23 19:34:16 2020

Elapsed time: 2.02166s

Tests for Matrices of size 1000

Finished at Sun Feb 23 19:34:22 2020

Elapsed time: 6.06323s

Starting tests of Matrix multiplication (Vectors)

Tests for Matrices of size 1

Finished at Sun Feb 23 19:34:22 2020

Elapsed time: 6.14e-07s

Tests for Matrices of size 3

Finished at Sun Feb 23 19:34:22 2020

Elapsed time: 9.5e-07s

Tests for Matrices of size 4

Finished at Sun Feb 23 19:34:22 2020

Elapsed time: 1.111e-06s

Tests for Matrices of size 8

Finished at Sun Feb 23 19:34:22 2020

Elapsed time: 2.069e-06s

Tests for Matrices of size 10

Finished at Sun Feb 23 19:34:22 2020

Elapsed time: 2.559e-06s

Tests for Matrices of size 20

Finished at Sun Feb 23 19:34:22 2020

Elapsed time: 5.035e-06s

Tests for Matrices of size 50

Finished at Sun Feb 23 19:34:22 2020

Elapsed time: 1.6025e-05s

Tests for Matrices of size 70

Finished at Sun Feb 23 19:34:22 2020

Elapsed time: 2.65e-05s

Tests for Matrices of size 150

Finished at Sun Feb 23 19:34:22 2020

Elapsed time: 9.5366e-05s

Tests for Matrices of size 300

Finished at Sun Feb 23 19:34:22 2020

Elapsed time: 0.000340851s

Tests for Matrices of size 700

Finished at Sun Feb 23 19:34:22 2020

Elapsed time: 0.00187317s

Tests for Matrices of size 1000

Finished at Sun Feb 23 19:34:23 2020

Elapsed time: 0.00344942s

**Main.cpp:**

#include <iostream> // cout, endl

#include <stdio.h> // printf

#include "matrix.h" // Matrix class

#include <time.h> // time functions

#include <chrono> // chrono

void time(Matrix&, Matrix&, int, bool, bool, bool); // Times the basic matrix arithmetic

int main() {

srand(time(NULL));

std::cout << "Testing printMatrix Function" << std::endl;

Matrix test;

test.printMatrix();

(test \* 2).printMatrix();

std::cout << "Testing Overloaded Constructor" << std::endl;

std::cout << "Creating 5x4 matrix" << std::endl;

Matrix test2(5,4);

test2.printMatrix();

std::cout << "Testing Scalar multiplication overloaded multiplication (BONUS)" << std::endl;

Matrix test3;

Matrix test4;

std::cout << "Test3\n";

test3.printMatrix();

std::cout << "Multiplying test3 by 4" << std::endl;

(test3 \* 4).printMatrix();

std::cout << "Copy constructor" << std::endl;

std::cout << "Test3" << std::endl;

test3.printMatrix();

std::cout << "Test5 copied over from test3" << std::endl;

Matrix test5(test3);

test5.printMatrix();

std::cout << "Overloaded assignment" << std::endl;

Matrix test6 = test5;

std::cout << "test6 assigned as test5" << std::endl;

test6.printMatrix();

std::cout << "Preventing self assignment" << std::endl;

test6 = test6;

std::cout << "test6" << std::endl;

test6.printMatrix();

std::cout << "test2" << std::endl;

test2.printMatrix();

std::cout << "Copy test2 over to test6" << std::endl;

test6 = test2;

std::cout << "test6" << std::endl;

test6.printMatrix();

std::cout << "Testing overloaded addition" << std::endl;

Matrix test7;

std::cout << "test7" << std::endl;

test7.printMatrix();

Matrix test8;

std::cout << "test8" << std::endl;

test8.printMatrix();

Matrix test9 = test7 + test8;

std::cout << "Result" << std::endl;

test9.printMatrix();

std::cout << "Testing overloaded subtraction" << std::endl;

std::cout << "test7" << std::endl;

test7.printMatrix();

std::cout << "test8" << std::endl;

test8.printMatrix();

test9 = test7 - test8;

std::cout << "Result" << std::endl;

test9.printMatrix();

std::cout << "Testing overloaded multiplication" << std::endl;

Matrix test10(2,2);

Matrix test11(2,1);

std::cout << "test10" << std::endl;

test10.printMatrix();

std::cout << "test11" << std::endl;

test11.printMatrix();

Matrix test12 = test10 \* test11;

std::cout << "Result" << std::endl;

test12.printMatrix();

std::cout << "Testing Diagonal Matrix creation" << std::endl;

Matrix test13;

test13.fillMatrixDiagonal();

test13.printMatrix();

std::cout << "Testing Triangle Matrix creation" << std::endl;

std::cout << "Upper" << std::endl;

test13.fillMatrixTriangle(true);

test13.printMatrix();

std::cout << "Lower" << std::endl;

test13.fillMatrixTriangle(false);

test13.printMatrix();

std::cout << "Testing Identity Matrix creation" << std::endl;

test13.fillMatrixIdentity();

test13.printMatrix();

int testCases[] = {1, 3, 4, 8, 10, 20, 50, 70, 150, 300, 700, 1000};

int len = 12;

std::cout << "Starting tests of Matrix addition" << std::endl;

for (int i = 0; i < len; i++) {

Matrix t1(testCases[i], testCases[i]);

Matrix t2(testCases[i], testCases[i]);

time(t1, t2, testCases[i], true, false, false);

}

std::cout << "Starting tests of Matrix subtraction" << std::endl;

for (int i = 0; i < len; i++) {

Matrix t1(testCases[i], testCases[i]);

Matrix t2(testCases[i], testCases[i]);

time(t1, t2, testCases[i], false, false, false);

}

std::cout << "Starting tests of Matrix multiplication (Random Matrices)" << std::endl;

for (int i = 0; i < len; i++) {

Matrix t1(testCases[i], testCases[i]);

Matrix t2(testCases[i], testCases[i]);

time(t1, t2, testCases[i], false, true, false);

}

std::cout << "Starting tests of Matrix multiplication (Diagonal Matrices)" << std::endl;

for (int i = 0; i < len; i++) {

Matrix t1(testCases[i], testCases[i]);

Matrix t2(testCases[i], testCases[i]);

t1.fillMatrixDiagonal();

t2.fillMatrixDiagonal();

time(t1, t2, testCases[i], false, true, false);

}

std::cout << "Starting tests of Matrix multiplication (Triangular Matrices upper)" << std::endl;

for (int i = 0; i < len; i++) {

Matrix t1(testCases[i], testCases[i]);

Matrix t2(testCases[i], testCases[i]);

t1.fillMatrixTriangle(true);

t2.fillMatrixTriangle(true);

time(t1, t2, testCases[i], false, true, false);

}

std::cout << "Starting tests of Matrix multiplication (Triangular Matrices lower)" << std::endl;

for (int i = 0; i < len; i++) {

Matrix t1(testCases[i], testCases[i]);

Matrix t2(testCases[i], testCases[i]);

t1.fillMatrixTriangle(false);

t2.fillMatrixTriangle(false);

time(t1, t2, testCases[i], false, true, false);

}

std::cout << "Starting tests of Matrix multiplication (Identity Matrices)" << std::endl;

for (int i = 0; i < len; i++) {

Matrix t1(testCases[i], testCases[i]);

Matrix t2(testCases[i], testCases[i]);

t1.fillMatrixIdentity();

t2.fillMatrixIdentity();

time(t1, t2, testCases[i], false, true, false);

}

std::cout << "Starting tests of Matrix multiplication (Vectors)" << std::endl;

for (int i = 0; i < len; i++) {

Matrix t1(testCases[i], testCases[i]);

Matrix t2(testCases[i], 1);

time(t1, t2, testCases[i], false, true, true);

}

return 0;

}

void time(Matrix& lhs, Matrix& rhs, int size, bool isAdd, bool isMult, bool isVect) {

Matrix result(size, size);

if (isVect) {

Matrix other(size, 1);

result = other;

}

auto start = std::chrono::system\_clock::now();

if (isAdd) {

result = lhs + rhs;

} else if (isMult) {

result = lhs \* rhs;

} else {

result = lhs - rhs;

}

auto end = std::chrono::system\_clock::now();

std::chrono::duration<double> elapsed\_seconds = end - start;

std::time\_t end\_time = std::chrono::system\_clock::to\_time\_t(end);

std::cout << std::endl;

std::cout << "Tests for Matrices of size " << size << std::endl;

std::cout << "Finished at " << std::ctime(&end\_time) << "Elapsed time: " << elapsed\_seconds.count() << "s\n";

std::cout << std::endl;

}

**Matrix.h:**

#ifndef \_MATRIX\_H\_

#define \_MATRIX\_H\_

#include <iostream>

#include <stdio.h>

class Matrix {

private:

int\*\* arr;

int rowLength;

int columnLength;

void fillMatrix(); // Fills the matrix with random values

public:

Matrix(); // Default constructor

Matrix(int, int); // Overloaded constructor that takes a length and width

Matrix(const Matrix&); // Copy constructor

~Matrix(); // Destructor

Matrix& operator=(const Matrix&); // Overloaded assignment

void fillMatrixDiagonal(); // Creates a diagonal matrix

void fillMatrixTriangle(bool); // Creates a upper/lower triangular matrix

void fillMatrixIdentity(); // Creates an identity matrix

Matrix addMatrices(Matrix&); // Adds two matrices together

Matrix subtractMatrices(Matrix&); // Subtracts two matrices from each other

Matrix multMatrices(Matrix&); // Multiplies two matrices together

Matrix operator\*(int); // Used for multiplying a matrix by a scalar

Matrix operator\*(Matrix&); // Used for multiplying matrices together

Matrix operator+(Matrix&); // Used for more simple adding of two matrices

Matrix operator-(Matrix&); // Used for more simple subtracting of two matrices

void printMatrix(); // Prints out the corresponding matrix

};

#endif

**Matrix.cpp:**

#include "matrix.h"

/\*

\* Default Constructor:

\* Initializes the matrix and uses a fill function to fill the matrix completely

\*/

Matrix::Matrix() {

arr = new int\*[3];

for (int i = 0; i < 3; i++) {

arr[i] = new int[3];

}

rowLength = 3;

columnLength = 3;

fillMatrix();

}

/\*

\* Overloaded Constructor:

\* Initializes the matrix to specified row and column values and fills the matrix

\*/

Matrix::Matrix(int r, int c) {

if (r < 1 || c < 1) {

throw "Dimensions passed are not positive real numbers!";

}

arr = new int\*[r];

for (int i = 0; i < r; i++) {

arr[i] = new int[c];

}

rowLength = r;

columnLength = c;

fillMatrix();

}

/\*

\* fillMatrix Function:

\* Fills the matrix with values from 1 to 5 for testing purposes

\*/

void Matrix::fillMatrix() {

for (int i = 0; i < rowLength; i++) {

for (int j = 0; j < columnLength; j++) {

arr[i][j] = 1 + rand() % 5;

}

}

}

/\*

\* fillMatrixDiagonal Function:

\* Creates a diagonal matrix

\*/

void Matrix::fillMatrixDiagonal() {

for (int i = 0; i < rowLength; i++) {

for (int j = 0; j < columnLength; j++) {

if (i == j) {

arr[i][j] = 0;

} else {

arr[i][j] = 1 + rand() % 5;

}

}

}

}

/\*

\* fillMatrixTriangle Function:

\* Takes a bool to determine if the matrix will be an upper or lower triangle matrix

\* and then creates that type of matrix accordingly

\*/

void Matrix::fillMatrixTriangle(bool isUpper) {

if (isUpper) {

for (int i = 0; i < rowLength; i++) {

for (int j = 0; j < columnLength; j++) {

if (j >= i) {

arr[i][j] = 1 + rand() % 5;

} else {

arr[i][j] = 0;

}

}

}

} else {

for (int i = 0; i < rowLength; i++) {

for (int j = 0; j < columnLength; j++) {

if (j <= i) {

arr[i][j] = 1 + rand() % 5;

} else {

arr[i][j] = 0;

}

}

}

}

}

/\*

\* fillMatrixIdentity Function:

\* Creates an identity matrix

\*/

void Matrix::fillMatrixIdentity() {

for (int i = 0; i < rowLength; i++) {

for (int j = 0; j < columnLength; j++) {

if (i == j) {

arr[i][j] = 1;

} else {

arr[i][j] = 0;

}

}

}

}

/\*

\* Copy Constructor:

\* Performs deep copy on matrix

\*/

Matrix::Matrix(const Matrix& rhs) {

arr = new int\*[rhs.rowLength];

for (int i = 0; i < rhs.rowLength; i++) {

arr[i] = new int[rhs.columnLength];

}

rowLength = rhs.rowLength;

columnLength = rhs.columnLength;

for (int i = 0; i < rowLength; i++) {

for (int j = 0; j < columnLength; j++) {

arr[i][j] = rhs.arr[i][j];

}

}

}

/\*

\* Destructor:

\* Performs memory deallocation

\*/

Matrix::~Matrix() {

for (int i = 0; i < rowLength; i++) {

delete [] arr[i];

}

delete [] arr;

}

/\*

\* Overloaded assignment operator:

\* Performs a deep copy of a matrix to another matrix

\*/

Matrix& Matrix::operator=(const Matrix& rhs) {

if (this == &rhs) {

return \*this;

}

for (int i = 0; i < rowLength; i++) {

delete [] arr[i];

}

delete [] arr;

arr = new int\*[rhs.rowLength];

for (int i = 0; i < rhs.rowLength; i++) {

arr[i] = new int[rhs.columnLength];

}

rowLength = rhs.rowLength;

columnLength = rhs.columnLength;

for (int i = 0; i < rowLength; i++) {

for (int j = 0; j < columnLength; j++) {

arr[i][j] = rhs.arr[i][j];

}

}

return \*this;

}

/\*

\* addMatrices Function:

\* Takes a matrix as an input and adds the two matrices together, returning a third matrix

\*/

Matrix Matrix::addMatrices(Matrix& rhs) {

if (rowLength != rhs.rowLength || columnLength != rhs.columnLength) {

throw "Matrix dimensions aren't equal!";

}

Matrix rtnMe(rowLength, columnLength);

for (int i = 0; i < rowLength; i++) {

for (int j = 0; j < columnLength; j++) {

rtnMe.arr[i][j] = arr[i][j] + rhs.arr[i][j];

}

}

return rtnMe;

}

/\*

\* Overloaded addition operator:

\* Allows for easy addition instead of using matrix functions

\*/

Matrix Matrix::operator+(Matrix& rhs) {

return addMatrices(rhs);

}

/\*

\* subtractMatrices Function:

\* Takes a matrix as an input and then subtracts it from the current class matrix

\*/

Matrix Matrix::subtractMatrices(Matrix& rhs) {

if (rowLength != rhs.rowLength || columnLength != rhs.columnLength) {

throw "Matrix dimensions aren't equal!";

}

Matrix rtnMe(rowLength, columnLength);

for (int i = 0; i < rowLength; i++) {

for (int j = 0; j < columnLength; j++) {

rtnMe.arr[i][j] = arr[i][j] - rhs.arr[i][j];

}

}

return rtnMe;

}

/\*

\* Overloaded subtraction operator:

\* Allows for easy subtraction instead of using matrix functions

\*/

Matrix Matrix::operator-(Matrix& rhs) {

return subtractMatrices(rhs);

}

/\*

\* multMatrices Function:

\* Takes a matrix as an input and then multipliese it with the current class matrix

\*/

Matrix Matrix::multMatrices(Matrix& rhs) {

if (columnLength != rhs.rowLength) {

throw "Matrix dimensions incorrect for algorithm! NxM & MxK not satisfies";

}

Matrix rtnMe(rowLength, rhs.columnLength);

for (int i = 0; i < rowLength; i++) {

for (int j = 0; j < rhs.columnLength; j++) {

int sum = 0;

for (int l = 0; l < columnLength; l++) {

sum += arr[i][l] \* rhs.arr[l][j];

}

rtnMe.arr[i][j] = sum;

}

}

return rtnMe;

}

/\*

\* Overloaded multiplication operator:

\* Takes a scalar (single int) and a matrix and multiplies them together

\*/

Matrix Matrix::operator\*(int scalar) {

Matrix rtnMe(rowLength, columnLength);

for (int i = 0; i < rowLength; i++) {

for (int j = 0; j < columnLength; j++) {

rtnMe.arr[i][j] = scalar \* arr[i][j];

}

}

return rtnMe;

}

/\*

\* Overloaded multiplication operator:

\* Takes a matrix and then multiplies the matrices together

\*/

Matrix Matrix::operator\*(Matrix& rhs) {

return multMatrices(rhs);

}

/\*

\* printMatrix Function:

\* Prints the current matrix of the class out to the screen

\*/

void Matrix::printMatrix() {

std::cout << std::endl;

for (int i = 0; i < rowLength; i++) {

for (int j = 0; j < columnLength; j++) {

std::cout << arr[i][j] << " ";

}

std::cout << std::endl;

}

std::cout << std::endl;

}