

CMP 220 – Mini-Project 1

By:

Hassan Al Ali – B00068258

Muhammed Kamran - B00064929

Amna Al Ali - G00067024

Section 17

Due: April 9, 2017

**Question 1:**

Matrix.h

#include <iostream>

using namespace std;

#ifndef Matrix\_h

#define Matrix\_h

class Matrix { //class declaration

//declaraing friend functions

friend ostream & operator<<(ostream & out, const Matrix &a);

friend istream & operator >> (istream & in, const Matrix &a);

public:

Matrix(int r = 2, int c = 3); //default constructor

Matrix(const Matrix&); //copy constructor

~Matrix(); //destructor

//get functions

int get(int r, int c) const;

int getRow() const;

int getColumn() const;

void set(int r, int c, int value); //set function

Matrix operator-(); //Matrix negation

Matrix & operator++(); //Matrix pre-increment

Matrix & operator--(); //Matrix pre-decrement

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

//Matrix addition

Matrix operator+(const int &a);

Matrix operator+(const Matrix &a);

//Matrix subtraction

Matrix operator-(const int &a);

Matrix operator-(const Matrix &a);

//Matrix multiplication

Matrix operator\*(const int &a);

Matrix operator\*(const Matrix &a);

//Matrix "+=" operation

Matrix &operator+=(const int &a);

Matrix &operator+=(const Matrix &a);

//Matrix "-=" operation

Matrix &operator-=(const int &a);

Matrix &operator-=(const Matrix &a);

//Matrix "\*=" operation

Matrix &operator\*=(const int &a);

Matrix &operator\*=(const Matrix &a);

bool operator==(const Matrix &a); //Matrix equality

bool operator!=(const Matrix &a); //Matrix inequality

int & operator()(int r, int c); //double scripting operator (left hand side)

const int & operator()(int r, int c) const; //double scripting operator (right hand side)

private:

int row; //number of rows

int col; //number of cols

int \*\*arr;

};

#endif

Matrix.cpp

#include "Matrix.h"

#include <iostream>

#include <cmath>

using namespace std;

ostream & operator<<(ostream & out, const Matrix &a) //overloading the stream insertion << operator

{

for (int i = 0; i < a.row; i++)

{

out << "| ";

for (int j = 0; j < a.col; j++)

{

out << a(i, j) << " ";

}

out << "|" << endl;

}

out << endl;

return out;

}

istream &operator >> (istream &in, const Matrix &a) //overloading the stream extraction >> operator

{

for (int i = 0; i < a.row; i++)

for (int j = 0; j < a.col; j++)

{

cout << "Input value for (" << i << ", " << j << "): ";

in >> a.arr[i][j];

}

return in;

}

Matrix::Matrix(int r, int c) //array initialized with 0 values and default size of 2 rows and 3 cols

{

row = r;

col = c;

arr = new int \*[row];

for (int i = 0; i < row; i++)

{

arr[i] = new int[col];

}

for (int i = 0; i < row; i++)

{

for (int j = 0; j < col; j++)

{

(\*this)(i, j) = 0;

}

}

}

Matrix::Matrix(const Matrix& a) //copy constructor

{

row = a.row;

col = a.col;

arr = new int \*[row];

for (int i = 0; i < row; i++)

{

arr[i] = new int[col];

}

for (int i = 0; i < row; i++)

{

for (int j = 0; j < col; j++)

{

(\*this)(i, j) = a(i, j);

}

}

}

Matrix::~Matrix() //destructor

{

for (int i = 0; i < row; i++)

{

delete[] arr[i];

}

delete[] arr;

}

//get functions

int Matrix::get(int r, int c) const

{

if (r >= row || c >= col || c < 0 || r < 0)

{

cout << "Row/Col Size Error. Exiting!" << endl;

exit(1);

}

else {

return (\*this)(r, c);

}

}

int Matrix::getRow() const

{

return row;

}

int Matrix::getColumn() const

{

return col;

}

void Matrix::set(int r, int c, int value) //set function

{

if (r >= row || c >= col || c < 0 || r < 0)

{

cout << "Row/Col Size Error. Exiting!" << endl;

exit(1);

}

else {

(\*this)(r, c) = value;

}

}

Matrix Matrix::operator-() //Matrix negation

{

Matrix negate(\*this);

for (int i = 0; i < row; i++)

for (int j = 0; j < col; j++)

negate(i, j) \*= -1;

return negate;

}

Matrix & Matrix::operator++() //Matrix pre-increment

{

for (int i = 0; i < row; i++)

for (int j = 0; j < col; j++)

(\*this)(i, j)++;

return \*this;

}

Matrix & Matrix::operator--() //Matrix pre-decrement

{

for (int i = 0; i < row; i++)

for (int j = 0; j < col; j++)

(\*this)(i, j)--;

return \*this;

}

Matrix & Matrix::operator=(const Matrix &a) //Matrix assignment

{

if (row != a.row || col != a.col)

{

for (int i = 0; i < row; i++)

{

delete[] arr[i];

}

delete[] arr;

row = a.row;

col = a.col;

arr = new int \*[row];

for (int i = 0; i < row; i++)

{

arr[i] = new int[col];

}

}

for (int i = 0; i < row; i++)

{

for (int j = 0; j < col; j++)

{

(\*this)(i, j) = a(i, j);

}

}

return \*this;

}

//Matrix addition

Matrix Matrix::operator+(const int &a)

{

Matrix sum(\*this);

for (int i = 0; i < row; i++)

for (int j = 0; j < col; j++)

sum(i, j) += a;

return sum;

}

Matrix Matrix::operator+(const Matrix &a)

{

if (row != a.row || col != a.col)

{

cout << "Rows or columns not equal" << endl;

exit(1);

}

Matrix sum(row, col);

for (int i = 0; i < row; i++)

for (int j = 0; j < col; j++)

sum(i, j) = (\*this)(i, j) + a(i, j);

return sum;

}

//Matrix subtraction

Matrix Matrix::operator-(const int &a)

{

Matrix difference(\*this);

for (int i = 0; i < row; i++)

for (int j = 0; j < col; j++)

difference(i, j) -= a;

return difference;

}

Matrix Matrix::operator-(const Matrix &a)

{

if (row != a.row || col != a.col)

{

cout << "Rows or columns not equal" << endl;

exit(1);

}

Matrix difference(row, col);

for (int i = 0; i < row; i++)

for (int j = 0; j < col; j++)

difference(i, j) = (\*this)(i, j) - a(i, j);

return difference;

}

//Matrix multiplication

Matrix Matrix::operator\*(const int &a)

{

Matrix product(\*this);

for (int i = 0; i < row; i++)

for (int j = 0; j < col; j++)

product(i, j) \*= a;

return product;

}

Matrix Matrix::operator\*(const Matrix &a)

{

if (col != a.row)

{

cout << "Multiplication not possible.\n";

exit(1);

}

Matrix product(row, a.col);

for (int i = 0; i < product.row; i++)

{

for (int j = 0; j < product.col; j++)

{

int sum = 0;

for (int k = 0; k < col; k++)

sum += (\*this)(i, k) \* a(k, j);

product(i, j) = sum;

}

}

return product;

}

//Matrix "+=" operation

Matrix &Matrix::operator+=(const int &a)

{

\*this = \*this + a;

return \*this;

}

Matrix &Matrix::operator+=(const Matrix &a)

{

\*this = \*this + a;

return \*this;

}

//Matrix "-=" operation

Matrix &Matrix::operator-=(const int &a)

{

\*this = \*this - a;

return \*this;

}

Matrix &Matrix::operator-=(const Matrix &a)

{

\*this = \*this - a;

return \*this;

}

//Matrix "\*=" operation

Matrix &Matrix::operator\*=(const int &a)

{

\*this = \*this \* a;

return \*this;

}

Matrix &Matrix::operator\*=(const Matrix &a)

{

\*this = \*this \* a;

return \*this;

}

bool Matrix::operator==(const Matrix &a) //Matrix equality

{

if (row != a.row || col != a.col)

return 0;

for (int i = 0; i < a.row; i++)

for (int j = 0; j < a.col; j++)

if ((\*this)(i, j) != a(i, j))

return 0;

return 1;

}

bool Matrix::operator!=(const Matrix &a) //Matrix inequality

{

if (row != a.row || col != a.col)

return 1;

for (int i = 0; i < a.row; i++)

for (int j = 0; j < a.col; j++)

if ((\*this)(i, j) != a(i, j))

return 1;

return 0;

}

int & Matrix::operator()(int r, int c) //double scripting operator (left hand side)

{

if (r >= row || c >= col || c < 0 || r < 0)

{

cout << "Out of range Error" << endl;

exit(1);

}

return arr[r][c];

}

const int & Matrix::operator()(int r, int c) const //double scripting operator (right hand side)

{

if (r >= row || c >= col || c < 0 || r < 0)

{

cout << "Out of range Error" << endl;

exit(1);

}

return arr[r][c];

}

Driver.cpp

/\*

Project 1 - Hassan - Muhammed - Amna

\*/

#include "Matrix.h"

#include <iostream>

using namespace std;

int main() {

cout << "Please input rows and columns for Matrix A: " << endl;

int a, b;

cin >> a >> b;

cout << "Now creating Matrix A...\n";

Matrix A(a, b);

cin >> A;

cout << "A=\n" << A << "Testing negate function...\n-A=\n" << -A;

cout << "Testing copy constructor on B with A...\n";

Matrix B(A);

cout << "B=\n" << B;

cout << "Testing set function by setting the value of B(" << a - 1 << ", " << b - 1 << ") to 120...\n";

B.set(a - 1, b - 1, 120);

cout << "B=\n" << B;

cout << "Testing boolean Matrix operators == and !=...\n";

cout << "A == B : " << (A == B) << endl

<< "A != B : " << (A != B) << endl;

cout << "\nTesting pre-increment and pre-decrement operators on A...\n";

cout << "A after ++A=\n" << ++A;

cout << "A after --A=\n" << --A;

cout << "Testing assignment operator by creating Matrix C and assigning it with values of B...\n";

Matrix C = B;

cout << "C = \n" << C;

cout << "Testing +, -, and \* operators with A and B...\n";

cout << "A + B=\n" << A + B;

cout << "A - B=\n" << A - B;

cout << "A \* B =\n" << A \* B;

const int int\_const = 3;

cout << "Testing +, -, and \* operators with A and integer constant " << int\_const << "...\n";

cout << "A + " << int\_const << " =\n" << A + int\_const;

cout << "A - " << int\_const << " =\n" << A - int\_const;

cout << "A \* " << int\_const << " =\n" << A \* int\_const;

cout << "Testing +=, -=, and \*= operators with A and B...\n";

A += B;

cout << "A after += B =\n" << A;

A -= B;

cout << "A after -= B =\n" << A;

A \*= B;

cout << "A after \*= B =\n" << A;

cout << "Testing +=, -=, and \*= operators with A and integer constant " << int\_const << "...\n";

A += int\_const;

cout << "A after += " << int\_const << " =\n" << A;

A -= int\_const;

cout << "A after -= " << int\_const << " =\n" << A;

A \*= int\_const;

cout << "A after \*= " << int\_const << " =\n" << A;

cout << "Testing cascading operation A += C \*= B - C + A \* " << int\_const << "...\n";

A += C \*= B - C + A \* int\_const;

cout << "A =\n" << A;

cout << "B =\n" << B;

cout << "C =\n" << C;

cout << "Testing default constructor for D...\n";

Matrix D;

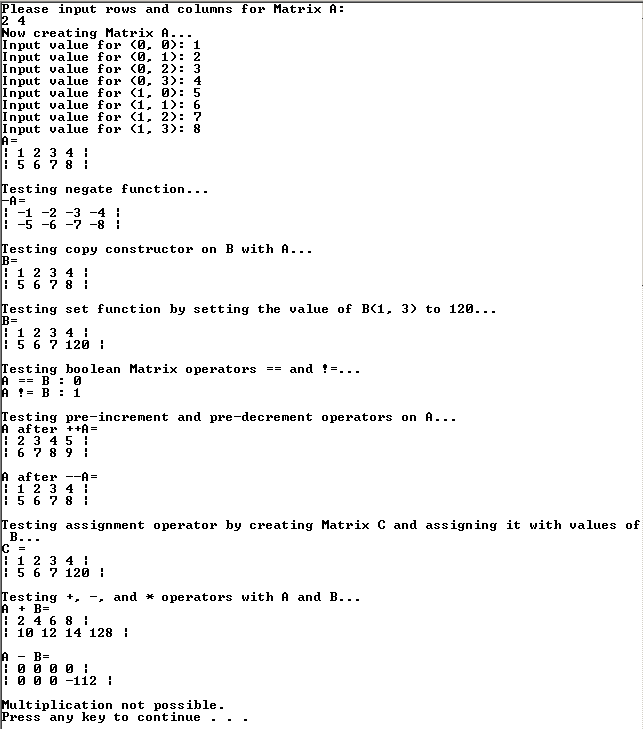
cout << "D =\n" << D;

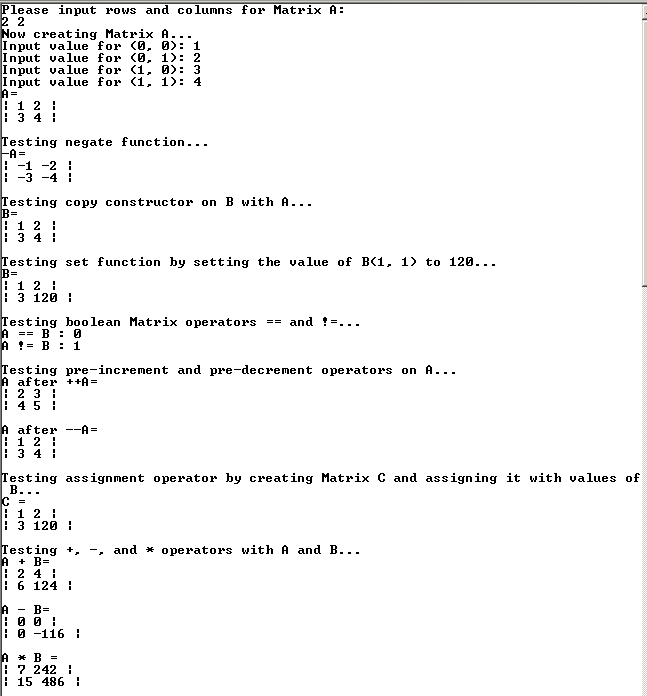
return 0;

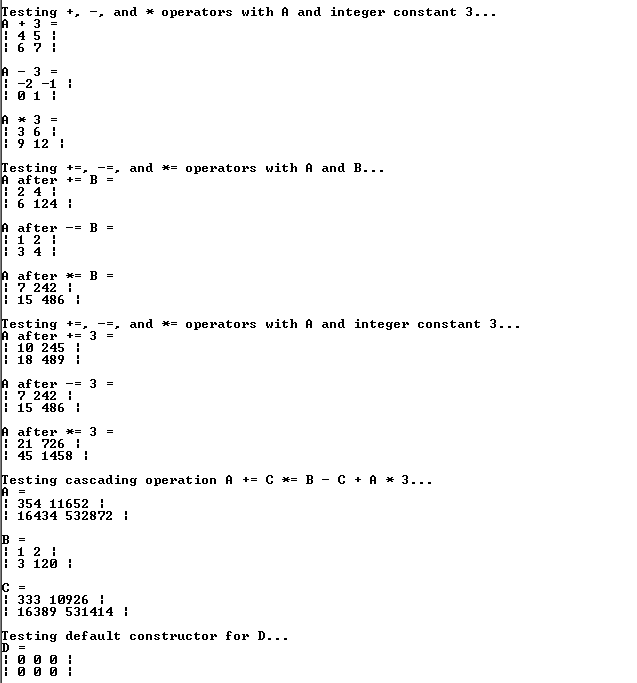
}

**Screenshots**

Example 1:



Example 2:



**Question 2:**

Matrix.h

#include <iostream>

using namespace std;

#ifndef Matrix\_h

#define Matrix\_h

class Matrix { //class declaration

//declaraing friend functions

friend ostream & operator<<(ostream & out, const Matrix &a);

friend istream & operator >> (istream & in, const Matrix &a);

public:

Matrix(int r = 2, int c = 3); //default constructor

Matrix(const Matrix&); //copy constructor

~Matrix(); //destructor

//get functions

int get(int r, int c) const;

int getRow() const;

int getColumn() const;

void set(int r, int c, int value); //set function

Matrix operator-(); //Matrix negation

Matrix & operator++(); //Matrix pre-increment

Matrix & operator--(); //Matrix pre-decrement

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

//Matrix addition

Matrix operator+(const int &a);

Matrix operator+(const Matrix &a);

//Matrix subtraction

Matrix operator-(const int &a);

Matrix operator-(const Matrix &a);

//Matrix multiplication

Matrix operator\*(const int &a);

Matrix operator\*(const Matrix &a);

//Matrix "+=" operation

Matrix &operator+=(const int &a);

Matrix &operator+=(const Matrix &a);

//Matrix "-=" operation

Matrix &operator-=(const int &a);

Matrix &operator-=(const Matrix &a);

//Matrix "\*=" operation

Matrix &operator\*=(const int &a);

Matrix &operator\*=(const Matrix &a);

bool operator==(const Matrix &a); //Matrix equality

bool operator!=(const Matrix &a); //Matrix inequality

int & operator()(int r, int c); //double scripting operator (left hand side)

const int & operator()(int r, int c) const; //double scripting operator (right hand side)

private:

int row; //number of rows

int col; //number of cols

int \*\*arr;

};

double SumMatrixElements(const Matrix &a);

double stdev(const Matrix &a);

#endif

Matrix.cpp

#include "Matrix.h"

#include <iostream>

#include <cmath>

using namespace std;

ostream & operator<<(ostream & out, const Matrix &a) //overloading the stream insertion << operator

{

for (int i = 0; i < a.row; i++)

{

out << "| ";

for (int j = 0; j < a.col; j++)

{

out << a(i, j) << " ";

}

out << "|" << endl;

}

out << endl;

return out;

}

istream &operator >> (istream &in, const Matrix &a) //overloading the stream extraction >> operator

{

for (int i = 0; i < a.row; i++)

for (int j = 0; j < a.col; j++)

{

cout << "Input value for (" << i << ", " << j << "): ";

in >> a.arr[i][j];

}

return in;

}

Matrix::Matrix(int r, int c) //array initialized with 0 values and default size of 2 rows and 3 cols

{

row = r;

col = c;

arr = new int \*[row];

for (int i = 0; i < row; i++)

{

arr[i] = new int[col];

}

for (int i = 0; i < row; i++)

{

for (int j = 0; j < col; j++)

{

(\*this)(i, j) = 0;

}

}

}

Matrix::Matrix(const Matrix& a) //copy constructor

{

row = a.row;

col = a.col;

arr = new int \*[row];

for (int i = 0; i < row; i++)

{

arr[i] = new int[col];

}

for (int i = 0; i < row; i++)

{

for (int j = 0; j < col; j++)

{

(\*this)(i, j) = a(i, j);

}

}

}

Matrix::~Matrix() //destructor

{

for (int i = 0; i < row; i++)

{

delete[] arr[i];

}

delete[] arr;

}

//get functions

int Matrix::get(int r, int c) const

{

if (r >= row || c >= col || c < 0 || r < 0)

{

cout << "Row/Col Size Error. Exiting!" << endl;

exit(1);

}

else {

return (\*this)(r, c);

}

}

int Matrix::getRow() const

{

return row;

}

int Matrix::getColumn() const

{

return col;

}

void Matrix::set(int r, int c, int value) //set function

{

if (r >= row || c >= col || c < 0 || r < 0)

{

cout << "Row/Col Size Error. Exiting!" << endl;

exit(1);

}

else {

(\*this)(r, c) = value;

}

}

Matrix Matrix::operator-() //Matrix negation

{

Matrix negate(\*this);

for (int i = 0; i < row; i++)

for (int j = 0; j < col; j++)

negate(i, j) \*= -1;

return negate;

}

Matrix & Matrix::operator++() //Matrix pre-increment

{

for (int i = 0; i < row; i++)

for (int j = 0; j < col; j++)

(\*this)(i, j)++;

return \*this;

}

Matrix & Matrix::operator--() //Matrix pre-decrement

{

for (int i = 0; i < row; i++)

for (int j = 0; j < col; j++)

(\*this)(i, j)--;

return \*this;

}

Matrix & Matrix::operator=(const Matrix &a) //Matrix assignment

{

if (row != a.row || col != a.col)

{

for (int i = 0; i < row; i++)

{

delete[] arr[i];

}

delete[] arr;

row = a.row;

col = a.col;

arr = new int \*[row];

for (int i = 0; i < row; i++)

{

arr[i] = new int[col];

}

}

for (int i = 0; i < row; i++)

{

for (int j = 0; j < col; j++)

{

(\*this)(i, j) = a(i, j);

}

}

return \*this;

}

//Matrix addition

Matrix Matrix::operator+(const int &a)

{

Matrix sum(\*this);

for (int i = 0; i < row; i++)

for (int j = 0; j < col; j++)

sum(i, j) += a;

return sum;

}

Matrix Matrix::operator+(const Matrix &a)

{

if (row != a.row || col != a.col)

{

cout << "Rows or columns not equal" << endl;

exit(1);

}

Matrix sum(row, col);

for (int i = 0; i < row; i++)

for (int j = 0; j < col; j++)

sum(i, j) = (\*this)(i, j) + a(i, j);

return sum;

}

//Matrix subtraction

Matrix Matrix::operator-(const int &a)

{

Matrix difference(\*this);

for (int i = 0; i < row; i++)

for (int j = 0; j < col; j++)

difference(i, j) -= a;

return difference;

}

Matrix Matrix::operator-(const Matrix &a)

{

if (row != a.row || col != a.col)

{

cout << "Rows or columns not equal" << endl;

exit(1);

}

Matrix difference(row, col);

for (int i = 0; i < row; i++)

for (int j = 0; j < col; j++)

difference(i, j) = (\*this)(i, j) - a(i, j);

return difference;

}

//Matrix multiplication

Matrix Matrix::operator\*(const int &a)

{

Matrix product(\*this);

for (int i = 0; i < row; i++)

for (int j = 0; j < col; j++)

product(i, j) \*= a;

return product;

}

Matrix Matrix::operator\*(const Matrix &a)

{

if (col != a.row)

{

cout << "Multiplication not possible.\n";

exit(1);

}

Matrix product(row, a.col);

for (int i = 0; i < product.row; i++)

{

for (int j = 0; j < product.col; j++)

{

int sum = 0;

for (int k = 0; k < col; k++)

sum += (\*this)(i, k) \* a(k, j);

product(i, j) = sum;

}

}

return product;

}

//Matrix "+=" operation

Matrix &Matrix::operator+=(const int &a)

{

\*this = \*this + a;

return \*this;

}

Matrix &Matrix::operator+=(const Matrix &a)

{

\*this = \*this + a;

return \*this;

}

//Matrix "-=" operation

Matrix &Matrix::operator-=(const int &a)

{

\*this = \*this - a;

return \*this;

}

Matrix &Matrix::operator-=(const Matrix &a)

{

\*this = \*this - a;

return \*this;

}

//Matrix "\*=" operation

Matrix &Matrix::operator\*=(const int &a)

{

\*this = \*this \* a;

return \*this;

}

Matrix &Matrix::operator\*=(const Matrix &a)

{

\*this = \*this \* a;

return \*this;

}

bool Matrix::operator==(const Matrix &a) //Matrix equality

{

if (row != a.row || col != a.col)

return 0;

for (int i = 0; i < a.row; i++)

for (int j = 0; j < a.col; j++)

if ((\*this)(i, j) != a(i, j))

return 0;

return 1;

}

bool Matrix::operator!=(const Matrix &a) //Matrix inequality

{

if (row != a.row || col != a.col)

return 1;

for (int i = 0; i < a.row; i++)

for (int j = 0; j < a.col; j++)

if ((\*this)(i, j) != a(i, j))

return 1;

return 0;

}

int & Matrix::operator()(int r, int c) //double scripting operator (left hand side)

{

if (r >= row || c >= col || c < 0 || r < 0)

{

cout << "Out of range Error" << endl;

exit(1);

}

return arr[r][c];

}

const int & Matrix::operator()(int r, int c) const //double scripting operator (right hand side)

{

if (r >= row || c >= col || c < 0 || r < 0)

{

cout << "Out of range Error" << endl;

exit(1);

}

return arr[r][c];

}

double SumMatrixElements(const Matrix &a) //returns the sum of the elements within a Matrix

{

double sum = 0;

int r = a.getRow();

int c = a.getColumn();

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

for (int j = 0; j < c; j++)

sum += a(i, j);

return sum;

}

double stdev(const Matrix &a) //returns the standard deviation of the Matrix

{

int r = a.getRow();

int c = a.getColumn();

double x\_average = SumMatrixElements(a) / (r \* c);

double sum = 0;

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

for (int j = 0; j < c; j++)

sum += pow(a(i, j) - x\_average, 2);

return sqrt(sum / (r\*c - 1));

}

Driver.cpp

/\*

Project 1 - Hassan - Muhammed - Amna

\*/

#include "Matrix.h"

#include <iostream>

using namespace std;

int main() {

cout << "Please input rows and columns for Matrix A: " << endl;

int a, b;

cin >> a >> b;

cout << "Now creating Matrix A...\n";

Matrix A(a, b);

cin >> A;

cout << "A=\n" << A << "Testing negate function...\n-A=\n" << -A;

cout << "Testing copy constructor on B with A...\n";

Matrix B(A);

cout << "B=\n" << B;

cout << "Testing set function by setting the value of B(" << a - 1 << ", " << b - 1 << ") to 120...\n";

B.set(a - 1, b - 1, 120);

cout << "B=\n" << B;

cout << "Testing boolean Matrix operators == and !=...\n";

cout << "A == B : " << (A == B) << endl

<< "A != B : " << (A != B) << endl;

cout << "\nTesting pre-increment and pre-decrement operators on A...\n";

cout << "A after ++A=\n" << ++A;

cout << "A after --A=\n" << --A;

cout << "Testing assignment operator by creating Matrix C and assigning it with values of B...\n";

Matrix C = B;

cout << "C = \n" << C;

cout << "Testing +, -, and \* operators with A and B...\n";

cout << "A + B=\n" << A + B;

cout << "A - B=\n" << A - B;

cout << "A \* B =\n" << A \* B;

const int int\_const = 3;

cout << "Testing +, -, and \* operators with A and integer constant " << int\_const << "...\n";

cout << "A + " << int\_const << " =\n" << A + int\_const;

cout << "A - " << int\_const << " =\n" << A - int\_const;

cout << "A \* " << int\_const << " =\n" << A \* int\_const;

cout << "Testing +=, -=, and \*= operators with A and B...\n";

A += B;

cout << "A after += B =\n" << A;

A -= B;

cout << "A after -= B =\n" << A;

A \*= B;

cout << "A after \*= B =\n" << A;

cout << "Testing +=, -=, and \*= operators with A and integer constant " << int\_const << "...\n";

A += int\_const;

cout << "A after += " << int\_const << " =\n" << A;

A -= int\_const;

cout << "A after -= " << int\_const << " =\n" << A;

A \*= int\_const;

cout << "A after \*= " << int\_const << " =\n" << A;

cout << "Testing cascading operation A += C \*= B - C + A \* " << int\_const << "...\n";

A += C \*= B - C + A \* int\_const;

cout << "A =\n" << A;

cout << "B =\n" << B;

cout << "C =\n" << C;

cout << "Testing default constructor for D...\n";

Matrix D;

cout << "D =\n" << D;

cout << "sum of elements in A is " << SumMatrixElements(A) << endl;

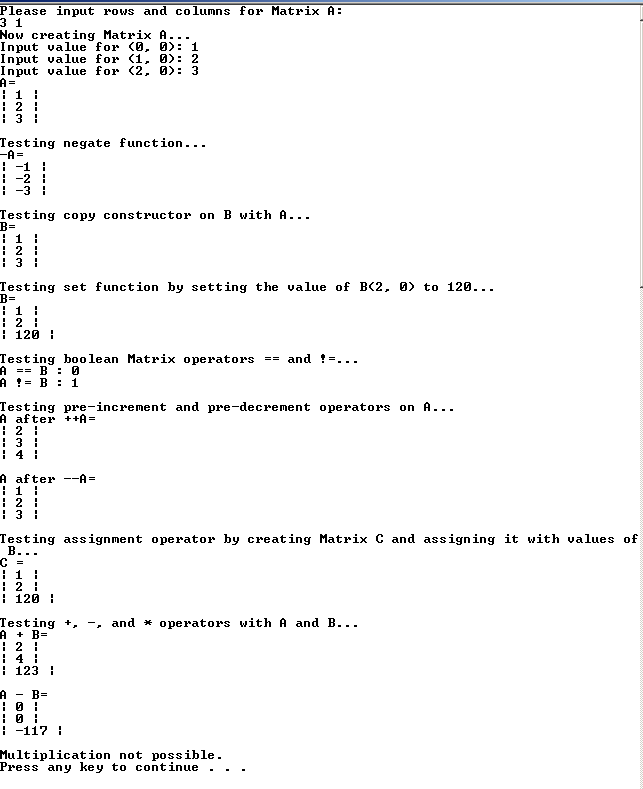
cout << "standard deviation of elements in A is " << stdev(A) << endl;

return 0;

}

**Screenshots**:

Example 1:



Example 2:

