Matrix.h

```
/*************************************************************
 * The interface file for the Matrix class. The only private   *
 * members are the sizes of the matrix and the pointer that    *
 * points to the matrix in the heap.                           *
 * The constructor creates a matrix in the heap and the        *
 * destructor deletes the allocated memory in the heap.        *
 * The setup member function fills the matrices randomly       *
 * We have addition, subtraction, multiplication, and print    *
 * member functions.                                           *
 *************************************************************/

#include <iostream>
#ifndef MATRIX_H
#define MATRIX_H
#include <cmath>
#include <cstdlib>
#include <iomanip>
#include <cassert>
using namespace std;

// Matrix class definition
class Matrix
{
private:
    int rowSize;
    int colSize;
    int** ptr;
public:
    Matrix(int rowSize, int colSize);
    ~Matrix();
    void setup();
    void add(const Matrix& second, Matrix& result) const;
    void subtract(const Matrix& second, Matrix& result) const;
    void multiply(const Matrix& second, Matrix& result) const;
    void print() const;
};
#endif
```

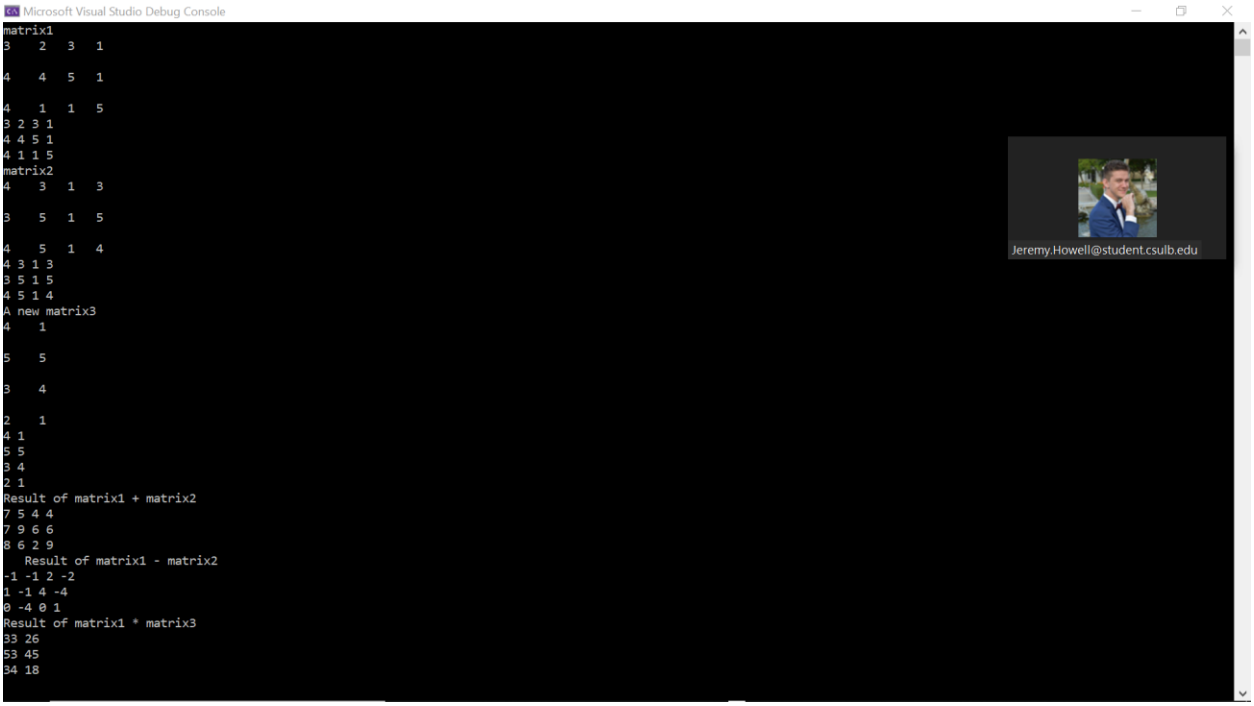
Matrix.cpp

```
#include "Matrix.h"

Matrix::Matrix(int rSize, int cSize)
{
       rowSize = rSize;
       colSize = cSize;
       ptr = new int* [rowSize];
       for (int i = 0; i < rowSize; i++) {
              ptr[i] = new int[colSize];
       }
}

Matrix::~Matrix() {}
```

```
void Matrix::setup()
{
	for (int i = 0; i < rowSize; i++) {
		for (int j = 0; j < colSize; j++) {
			cin >> ptr[i][j];
		}
	}
}

void Matrix::add(const Matrix& second, Matrix& result) const
{
	if ((this->colSize == second.colSize) && (this->rowSize == second.rowSize)) {
		for (int i = 0; i < rowSize; i++) {
			for (int j = 0; j < colSize; j++) {
				result.ptr[i][j] = second.ptr[i][j] + ptr[i][j];
			}
		}
	}
}

void Matrix::subtract(const Matrix& second, Matrix& result) const
{
	if ((this->colSize == second.colSize) && (this->rowSize == second.rowSize)) {
		for (int i = 0; i < rowSize; i++) {
			for (int j = 0; j < colSize; j++) {
				result.ptr[i][j] = ptr[i][j] - second.ptr[i][j];
			}
		}
	}
}

void Matrix::multiply(const Matrix& second, Matrix& result) const
{
	if (colSize == second.rowSize) {
		for (int i = 0; i < rowSize; i++) {
			for (int j = 0; j < second.colSize; j++) {
				int mul = 0;
				for (int z = 0; z < colSize; z++) {
					mul += ptr[i][z] * second.ptr[z][j];
				}
				result.ptr[i][j] = mul;
			}
		}
	}
}

void Matrix::print() const
{
	for (int i = 0; i < rowSize; i++) {
		for (int j = 0; j < colSize; j++) {
			cout << ptr[i][j] << ' ';
		}
		cout << endl;
	}
}
```

main.cpp

```cpp
/************************************************************
 * We create several matrix objects in the heap and we apply  *
 * some operations on them.                                   *
 ************************************************************/

#include "matrix.h"

int main()
{
	// Instantiation and setup of matrix1
	cout << "matrix1" << endl;
	Matrix matrix1(3, 4);
	matrix1.setup();
	matrix1.print();

	// Instantiation and setup of matrix2
	cout << "matrix2" << endl;
	Matrix matrix2(3, 4);
	matrix2.setup();
	matrix2.print();

	// Instantiation and setup of matrix3
	cout << "A new matrix3" << endl;
	Matrix matrix3(4, 2);
	matrix3.setup();
	matrix3.print();

	// Adding matrix2 to matrix1 and printing the result
	cout << "Result of matrix1 + matrix2" << endl;
	Matrix addResult(3, 4);
	matrix1.add(matrix2, addResult);
	addResult.print();

	// Subtracting matrix2 from matrix1 and printing the result
	cout << "   Result of matrix1 - matrix2" << endl;
	Matrix subResult(3, 4);
	matrix1.subtract(matrix2, subResult);
	subResult.print();

	// Multiplying matrix1 and matrix3 and printing the result
	cout << "Result of matrix1 * matrix3" << endl;
	Matrix mulResult(3, 2);
	matrix1.multiply(matrix3, mulResult);
	mulResult.print();
	return 0;
}
```



Demonstrated at 11:10am on 09/21/2021