Assignment #2

CS 225, Fall 2018

Due Wednesday, October 3

Overview

In this assignment, you will implement a `C++` style API for matrices. You will use some of the techniques discussed in class: dynamically allocated 2D arrays, subscripting, and exception handling. In addition, you will improve on the API by implementing some `C++11` specific features: move semantics and initializer lists.

API interface

I will give you the file `Matrix.h` which contains the following (public and private) declarations.

```
class Matrix {
  public:
    Matrix(unsigned r, unsigned c);
    Matrix(const Matrix &M);
    ~Matrix(void);
    Matrix& operator=(const Matrix &M);
    const float* operator[](unsigned r) const;
    float* operator[](unsigned r);
    unsigned getRows(void) const { return rows; }
    unsigned getCols(void) const { return cols; }
    Matrix(Matrix &&M) noexcept;
    Matrix& operator=(Matrix &&M) noexcept;
    Matrix(std::initializer_list<std::initializer_list<float>> list);
  private:
    unsigned rows,
             cols;
    float *data;
};

Matrix operator+(const Matrix &M, const Matrix &N);
Matrix operator-(const Matrix &M, const Matrix &N);
Matrix operator*(const Matrix &M, const Matrix &N);
Matrix operator*(float s, const Matrix &M);
```

Here are the details of the class member functions.

`Matrix(nr,nc)` — (constructor) creates a $nr \times nc$ matrix. That is, a matrix with nr rows and nc columns. The data for this matrix should be allocated dynamically.

`Matrix(M)` — (copy constructor, including move version) creates a matrix that is a (deep) copy of the matrix M.

`~Matrix()` — (destructor) destroys the matrix. As the data for the matrix was allocated dynamically, you will need to explicitly deallocate the data.

`operator=(M)` — (assignment operator, including move version) replaces the matrix with a (deep) copy of the matrix M.

`operator[](r)` — (subscripting operator, both constant and non–constant versions) returns the address of the r–th row of matrix data.

`getRows()` — returns the number of rows of the matrix. [Implemented]

`getCols()` — returns the number of columns of the matrix. [Implemented]

`Matrix(list)` — (constructor) creates a $nr \times nc$ matrix from an initializer list that has nr rows and nc columns. A `runtime_error` exception should be thrown if the initializer list does not have rows of equal size.

The helper functions of the API are described below.

`operator+(M,N)` — returns the sum $M + N$ of the matrices M and N. If the dimensions of M and N are not same, a `runtime_error` exception should be thrown.

`operator-(M,N)` — returns the difference $M - N$ of the matrices M and N. If the dimensions of M and N are not same, a `runtime_error` exception should be thrown.

`operator*(M,N)` — returns the product MN of the matrices M and N. Recall that if M has dimensions $mr \times mc$ and N has dimensions $nr \times nc$, then MN is only defined if $mc = nr$. In this case, MN has dimensions $mr \times nc$, and the element at the r–th row and c–th column of MN is given by the formula

$$(MN)_{rc} = \sum_{k=0}^{mc-1} M_{rk} N_{kc}$$

If the matrices M and N cannot be multiplied, a `runtime_error` exception should be thrown.

`operator*(r,M)` — returns the scaling rM of the matrix M by the factor r.

C++11 and compilers

The last three functions in the public section use C++11 specific language features. When using the GNU g++ compiler, you will need to use the -std=c++11 switch. E.g.,

```
g++ -std=c++11 MatrixTest.cpp Matrix.cpp
```

For the Visual Studio compiler, nothing special need be done.

What to turn in

Your assignment submission should consist of a *single* source file named Matrix.cpp. Only the header files Matrix.h, stdexcept, and initialzer_list may be included.