University of Connecticut
ECE1101
Spring 2021

Homework – Matrix Multiplication
Due April 28, 2021 End of Day

Matrix multiplication is a common operation in electrical engineering whether it's solving linear circuits or decoding communications schemes. Your task for this homework is to write a program that reads data from two files (having matrix data), performing the matrix multiplication, and then writing out the result.

If you haven't seen matrices before, let's do a quick overview. A matrix is a two-dimensional array of numbers, as shown below:

$$A = \begin{bmatrix} 1 & 0 & 2 \\ -1 & 4 & 3 \\ 5 & 2 & 1 \end{bmatrix}$$

Matrix multiplication takes two matrices and multiplies them, but instead of multiplying element by element, the multiplication is done row by column. The row/column multiplication is done using dot products. Dot product multiplication takes two arrays or vectors of numbers - $[a_0 \quad a_1 \quad a_2]$ and $[b_0 \quad b_1 \quad b_2]$, and multiplies each pair of numbers as follows: $a_0b_0 + a_1b_1 + a_2b_2$. In matrix multiplication, if we have two matrices, *A* and *B*, we multiply each row of A with each column of B in a dot product multiplication. For example, take the two matrices below:

$$A = \begin{bmatrix} 1 & 0 & 2 \\ -1 & 4 & 3 \\ 5 & 2 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 2 & -2 & 5 \\ 2 & 3 & 1 \\ 0 & 3 & -1 \end{bmatrix}$$

The multiplication of matrices *C=A•B* will be as follows – the element at row 0 and column 0 of matrix *C* will be the dot product of row 0 of *A* and column 0 of *B,* the element at row 0 and column 1 of matrix *C* will be the dot product of row 0 of *A* and column 1 of *B*, and more generally the element at row *i* and column *j* of matrix *C* will be the dot product of row *i* of *A* and column *j* of *B*.

$$C = \begin{bmatrix} 1\cdot 2 + 0\cdot 2 + 2\cdot 0 & 1\cdot -2 + 0\cdot 3 + 2\cdot 3 & 1\cdot 5 + 0\cdot 1 + 2\cdot -1 \\ -1\cdot 2 + 4\cdot 2 + 3\cdot 0 & -1\cdot -2 + 4\cdot 3 + 3\cdot 3 & -1\cdot 5 + 4\cdot 1 + 3\cdot -1 \\ 5\cdot 2 + 2\cdot 2 + 1\cdot 0 & 5\cdot -2 + 2\cdot 3 + 1\cdot 3 & 5\cdot 5 + 2\cdot 1 + 1\cdot -1 \end{bmatrix}$$

$$C = \begin{bmatrix} 2 & 4 & 3 \\ 6 & 23 & -4 \\ 14 & -1 & 26 \end{bmatrix}$$

You should run the program as follows:

```
./hw5 3 hw5-a-matrix.txt hw5-b-matrix.txt hw-c-matrix.txt
```

where, in this case, `3` is the size of the matrix, and `hw5-a-matrix.txt` and `hw5-b-matrix.txt` are the names of files that contain the input *A* and *B* matrices. Two sample *A* and *B* files are provided on HuskyCT. You will write the output matrix *C* to the file named `hw5-c-matrix.txt`. The matrix files are written with each line containing one row of the matrix with each number separated by a space.

Writing the code

This code must be written in C++. You are given a `main` function that looks as follows:

```
#include <stdlib.h>
#include "Matrix.h"

int main(int argc, char **argv)
{
    int m = atoi(argv[1]);

    Matrix A(m, argv[2]);
    Matrix B(m, argv[3]);
    Matrix C = A * B;

    C.write(argv[4]);
}
```

`m` is the size of the matrix, and `argv[2]`, `argv[3]`, and `argv[4]` are the file names for the two input files and the output file respectively. `Matrix A` and `Matrix B` are initialized using a constructor that takes in the size and the filename. `Matrix C` is created using an overloaded `*` operator. The `Matrix` class is defined in `Matrix.h` as follows:

```
class Matrix {
public:
    Matrix(int);
    Matrix(int, char *);
    Matrix(const Matrix &);
    ~Matrix();
    Matrix operator * (Matrix&) const;
    void write(char *);
private:
    int size;
    double *data;
};
```

You will need to complete the code for the `Matrix(int, char *)` constructor, the `Matrix(const Matrix &)` copy constructor, `Matrix *` operator, and the `write` method. The `data` field is an array that stores the two-dimensional data for the matrix. Since we don't know the size of the array until we run the program and the user tells us, we can’t predefine the matrix as a 2D array. C/C++ does not support variable-sized two-dimensional arrays, so the array has been created in the constructors using `new` as follows:

```
    double *data = new double [size*size];
```

Unfortunately, we can’t use two-dimensional array syntax like `data[i][j]` because the array is variable sized. So, instead we map the 2D matrix into a 1D array row by row as shown below:

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \rightarrow [a_{00} \quad a_{01} \quad a_{02} \quad a_{10} \quad a_{11} \quad a_{12} \quad a_{20} \quad a_{21} \quad a_{22}]$$

Thus, if we want the element at row i and column j, we do the following: `data[i*m+j]`

The methods that you need to write are described below.

`Matrix(int m, char *filename)` constructor

Use the `fscanf` function to read the file. Here is an example of using `fscanf` to read a number from a file:

```
        double value;
        FILE *file = fopen(filename, “r”);
        fscanf(file, "%lf ", &value);
```

This will read one number as a double from the file into the variable `value`.

You will need to use this call within two nested loops that iterate over the matrix array going through the rows in the outer loop and the columns in the inner loop to read the data from the file into the `data` array.

`Matrix(Matrix &original)` copy constructor

Iterate through the original `data` and copy the values into the new object `data`

`operator *(Matrix &b)`

This method will do the multiply of the current `Matrix` object with the `b Matrix`. You will need a set of nested loops – this time 3 nested loops. The first two nested loops *i* and *j* will iterate over the rows and columns of the output matrix `out`. The third and innermost loop will iterate over the *i*-th row of the current `data` and the *j*-th column of `b data` to do the dot product. This innermost loop will do an accumulation of the multiplications to calculate the dot product and the accumulated sum will go into the `out.data` element at row *i* and column *j*.

`write(char *filename)`

This method should write to the output file in the same format as the input matrix files. You can use the `fprintf` function to print the data to the file

```
fprintf(file, "%f ", value);
```

You will need to iterate over the rows and columns of `data` to print each element of the matrix to the file. Make sure you put a newline at the end of each row.

The code will be compiled as follows:

```
g++ -o hw5 hw5-matrix-main.cpp hw5-matrix.cpp
```

Submission

Submit your `.cpp` file on HuskyCT. Use comments to explain what your code does.