Lab 2: Library for Vector and Matrix Operation

Submission timestamps will be checked and enforced strictly by the CourseWeb; **late submissions will not be accepted**. Check the due date of this lab on the CourseWeb. Remember that, per the course syllabus, if you are not marked by your recitation instructor as having attended a recitation, your score will be cut in half.

For this lab, you are going to create your own library for vector and matrix operations. In this class, we are mainly use 4×1 matrices (column vectors) and 4×4 matrices. It is a good idea for you to create your own library to perform their operation. For this lab, you must create two files, header file and source file. You can name your file which ever you want (`.h`) for header file and (`.c`) for source file.

Vector and its Operations

A vector is a matrix (4×1) or column vector as shown below:

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

In OpenGL (C programming) environment, a 4×1 column vector is simply a four consecutive region in the memory that contains four floating-point value. For simplicity, we can create a new type in your header file called `vec4` as follows:

```
typedef struct
{
    GLfloat x;
    GLfloat y;
    GLfloat z;
    GLfloat w;
} vec4;
```

Note that if your system does not support `GLfloat` yet, just simply use `float`. `GLfloat` and `float` are identical. With the above declaration, a program can create the following vector:

$$\mathbf{v} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

using the following code:

```
#include "common.h"

int main(void)
{
    vec4 v = {1, 2, 3, 4};
    :
}
```

Note that you can also use a one-dimensional array of type `GLfloat` (or `float`) with four elements to represent a 4×1 column vector as shown below:

```
typedef float vec4[4];

int main(void)
{
    vec4 v = {1,2,3,4};
    :
}
```

In this section, create the following functions in your source file (and do not forget to put your function signatures into your header file). **Note** that you can change names and how your function passes values whichever way you prefer.

- **Prints a 4×1 column vector on the console screen**: This function will be very useful to help you debug your code. To reduce the amount of space to print, you can simply print a vector like a row vector and limit each floating-point number to just two digits after decimal point (e.g., `[1.23 -2.34 3.45 -4.56]`).
- **Scalar-Vector Multiplication**: Given a scalar value s (a floating-point number) and a vector $\mathbf{v}$ (4×1), this function should produce the result $s \times \mathbf{v}$ which is a 4×1 column vector.

$$\alpha \times \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} \alpha \times x \\ \alpha \times y \\ \alpha \times z \\ \alpha \times w \end{bmatrix}$$

- **Vector-Vector Addition**: Given two vectors $\mathbf{v}_1$ and $\mathbf{v}_2$, this function should produce the result $\mathbf{v}_1 + \mathbf{v}_2$ which is a column vector.

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ w_1 \end{bmatrix} + \begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ w_2 \end{bmatrix} = \begin{bmatrix} x_1 + x_2 \\ y_1 + y_2 \\ z_1 + z_2 \\ w_1 + w_2 \end{bmatrix}$$

- **Vector-Vector Subtraction** Given two vectors $\mathbf{v}_1$ and $\mathbf{v}_2$, this function should produce the result $\mathbf{v}_1 - \mathbf{v}_2$ which is a column vector.

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ w_1 \end{bmatrix} - \begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ w_2 \end{bmatrix} = \begin{bmatrix} x_1 - x_2 \\ y_1 - y_2 \\ z_1 - z_2 \\ w_1 - w_2 \end{bmatrix}$$

- **Magnitude of a Vector** The magnitude of a four-element vector v denoted by $|v|$ is as follows:

$$|v| = \sqrt{x^2 + y^2 + z^2 + w^2}$$

- **Normalize** A normalized vector is a vector with magnitude 1. A normalized of a vector v is a vector $\hat{v}$ codirectional with v and $|\hat{v}| = 1$. The vector $\hat{v}$ can be calculated by

$$\hat{v} = \frac{1}{|v|} \times \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

- **Dot product**: Given two vectors $\mathbf{v}_1$ and $\mathbf{v}_2$, this function should produce the result $\mathbf{v}_1 \cdot \mathbf{v}_2$. **Note** that the result is a scalar value (a floating-point number).

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ w_1 \end{bmatrix} \cdot \begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ w_2 \end{bmatrix} = (x_1 \times x_2) + (y_1 \times y_2) + (z_1 \times z_2) + (w_1 \times w_2)$$

- **Cross product**: Given two vectors $\mathbf{v}_1$ and $\mathbf{v}_2$, this function should produce the result $\mathbf{v}_1 \times \mathbf{v}_2$ which is a column vector. Note that the fourth element of the result vector should always be 0. The calculation should ignore the fourth elements of both $\mathbf{v}_1$ and $\mathbf{v}_2$. **Do not forget about the right-hand rule**.

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ w_1 \end{bmatrix} \times \begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ w_2 \end{bmatrix} = \begin{bmatrix} (y_1 \times z_2) - (z_1 \times y_2) \\ (z_1 \times x_2) - (x_1 \times z_2) \\ (x_1 \times y_2) - (y_1 \times x_2) \\ 0.0 \end{bmatrix}$$

Again, you can name your functions and their signatures whichever way you want.

Matrix and its Operations

In OpenGL (C programming) environment, a 4×4 matrix is simply a four consecutive region in the memory that contains 16 floating-point value organized according to **Column Major**. With the declaration of a four-element column vector, a 4×4 matrix can be created as follows:

```
typedef struct
{
    vec4 x;
    vec4 y;
    vec4 z;
    vec4 w;
} mat4;
```

Note that the above `mat4` data type is a column major. Each column is a vector (`vec4`). With the above data type, we can declare the following 4×4 matrix:

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

using the following code snippet:

```
    mat4 m = {{1,5,9,13},{2,6,10,14},{3,7,11,15},{4,8,12,16}};
```

You can also use a 16-element one-dimensional array of floating-points as shown below:

```
typedef float mat4[16];
```

However, do not forget that it must be a column major. Suppose a 4×4 matrix is declared (from one-dimensional array of floating-points) as shown below:

```
typedef float mat4[16];

int main(void)
{
    mat4 m = ...;
    :
}
```

the actual matrix represented by the above declaration is shown below:

$$\begin{bmatrix} \texttt{m[0]} & \texttt{m[4]} & \texttt{m[8]} & \texttt{m[12]} \\ \texttt{m[1]} & \texttt{m[5]} & \texttt{m[9]} & \texttt{m[13]} \\ \texttt{m[2]} & \texttt{m[6]} & \texttt{m[10]} & \texttt{m[14]} \\ \texttt{m[3]} & \texttt{m[7]} & \texttt{m[11]} & \texttt{m[15]} \end{bmatrix}$$

In this section, create the following functions for matrix operations:

- **Print a matrix on the console screen** for debugging purpose
- **Scalar-Matrix multiplication**: Given a scalar value s and a matrix $\mathbf{m}$, this function should produce the result $s \times \mathbf{m}$ which is a matrix.

$$\alpha \times \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} = \begin{bmatrix} \alpha \times a_{11} & \alpha \times a_{12} & \alpha \times a_{13} & \alpha \times a_{14} \\ \alpha \times a_{21} & \alpha \times a_{22} & \alpha \times a_{23} & \alpha \times a_{24} \\ \alpha \times a_{31} & \alpha \times a_{32} & \alpha \times a_{33} & \alpha \times a_{34} \\ \alpha \times a_{41} & \alpha \times a_{42} & \alpha \times a_{43} & \alpha \times a_{44} \end{bmatrix}$$

- **Matrix-Matrix Addition**: Given two matrices $\mathbf{m}_1$ and $\mathbf{m}_2$, this function should produce the result $\mathbf{m}_1 + \mathbf{m}_2$ which is a matrix.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix} = \begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & a_{13}+b_{13} & a_{14}+b_{14} \\ a_{21}+b_{21} & a_{22}+b_{22} & a_{23}+b_{23} & a_{24}+b_{24} \\ a_{31}+b_{31} & a_{32}+b_{32} & a_{33}+b_{33} & a_{34}+b_{34} \\ a_{41}+b_{41} & a_{42}+b_{42} & a_{43}+b_{43} & a_{44}+b_{44} \end{bmatrix}$$

- **Matrix-Matrix Subtraction**: Given two matrices $\mathbf{m}_1$ and $\mathbf{m}_2$, this function should produce the result $\mathbf{m}_1 - \mathbf{m}_2$ which is a matrix.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} - \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix} = \begin{bmatrix} a_{11}-b_{11} & a_{12}-b_{12} & a_{13}-b_{13} & a_{14}-b_{14} \\ a_{21}-b_{21} & a_{22}-b_{22} & a_{23}-b_{23} & a_{24}-b_{24} \\ a_{31}-b_{31} & a_{32}-b_{32} & a_{33}-b_{33} & a_{34}-b_{34} \\ a_{41}-b_{41} & a_{42}-b_{42} & a_{43}-b_{43} & a_{44}-b_{44} \end{bmatrix}$$

- **Matrix-Matrix multiplication**: Given two matrices $\mathbf{m}_1$ and $\mathbf{m}_2$, this function should produce the result $\mathbf{m}_1 \times \mathbf{m}_2$ which is a matrix.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix}$$

  where

$$\begin{aligned}
c_{11} &= (a_{11} \times b_{11}) + (a_{12} \times b_{21}) + (a_{13} \times b_{31}) + (a_{14} \times b_{41}) \\
c_{21} &= (a_{21} \times b_{11}) + (a_{22} \times b_{21}) + (a_{23} \times b_{31}) + (a_{24} \times b_{41}) \\
c_{31} &= (a_{31} \times b_{11}) + (a_{32} \times b_{21}) + (a_{33} \times b_{31}) + (a_{34} \times b_{41}) \\
c_{41} &= (a_{41} \times b_{11}) + (a_{42} \times b_{21}) + (a_{43} \times b_{31}) + (a_{44} \times b_{41}) \\
c_{12} &= (a_{11} \times b_{12}) + (a_{12} \times b_{22}) + (a_{13} \times b_{32}) + (a_{14} \times b_{42}) \\
c_{22} &= (a_{21} \times b_{12}) + (a_{22} \times b_{22}) + (a_{23} \times b_{32}) + (a_{24} \times b_{42}) \\
c_{32} &= (a_{31} \times b_{12}) + (a_{32} \times b_{22}) + (a_{33} \times b_{32}) + (a_{34} \times b_{42}) \\
c_{42} &= (a_{41} \times b_{12}) + (a_{42} \times b_{22}) + (a_{43} \times b_{32}) + (a_{44} \times b_{42}) \\
c_{13} &= (a_{11} \times b_{13}) + (a_{12} \times b_{23}) + (a_{13} \times b_{33}) + (a_{14} \times b_{43}) \\
c_{23} &= (a_{21} \times b_{13}) + (a_{22} \times b_{23}) + (a_{23} \times b_{33}) + (a_{24} \times b_{43}) \\
c_{33} &= (a_{31} \times b_{13}) + (a_{32} \times b_{23}) + (a_{33} \times b_{33}) + (a_{34} \times b_{43}) \\
c_{43} &= (a_{41} \times b_{13}) + (a_{42} \times b_{23}) + (a_{43} \times b_{33}) + (a_{44} \times b_{43}) \\
c_{14} &= (a_{11} \times b_{14}) + (a_{12} \times b_{24}) + (a_{13} \times b_{34}) + (a_{14} \times b_{44}) \\
c_{24} &= (a_{21} \times b_{14}) + (a_{22} \times b_{24}) + (a_{23} \times b_{34}) + (a_{24} \times b_{44}) \\
c_{34} &= (a_{31} \times b_{14}) + (a_{32} \times b_{24}) + (a_{33} \times b_{34}) + (a_{34} \times b_{44}) \\
c_{44} &= (a_{41} \times b_{14}) + (a_{42} \times b_{24}) + (a_{43} \times b_{34}) + (a_{44} \times b_{44})
\end{aligned}$$

- **Inverse of a Matrix**: Given a matrix $\mathbf{m}$ this function should produce the result $\mathbf{m}^{-1}$ which is a matrix. It may be a good idea for you to test by simply calculate whether $\mathbf{m} \times \mathbf{m}^{-1} = \mathbf{1}$ and $\mathbf{m}^{-1} \times \mathbf{m} = \mathbf{1}$. See the lecture slides on how to calculate the inverse of a 4×4 matrix.

- **Transpose of a Matrix**: Given a matrix $\mathbf{m}$ this function should produce the result $\mathbf{m}^T$ which is a matrix.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}^T = \begin{bmatrix} a_{11} & a_{21} & a_{31} & a_{41} \\ a_{12} & a_{22} & a_{32} & a_{42} \\ a_{13} & a_{23} & a_{33} & a_{43} \\ a_{14} & a_{24} & a_{34} & a_{44} \end{bmatrix}$$

- **Matrix-Vector Multiplication**: Given a matrix $\mathbf{m}$ and a vector $\mathbf{v}$, this function should produce the result $\mathbf{m} \times \mathbf{v}$ which is a vector.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} (a_{11} \times x) + (a_{12} \times y) + (a_{13} \times z) + (a_{14} \times w) \\ (a_{21} \times x) + (a_{22} \times y) + (a_{23} \times z) + (a_{24} \times w) \\ (a_{31} \times x) + (a_{32} \times y) + (a_{33} \times z) + (a_{34} \times w) \\ (a_{41} \times x) + (a_{42} \times y) + (a_{43} \times z) + (a_{44} \times w) \end{bmatrix}$$

Again, you can name your functions and their signatures whichever way you want.

Test Your Library

Again, your library should be in two files, a header file and a source file. Now, you should test your library a little bit. For this section, create a new program named `test.c`. In this program, create the following vectors and matrices:

$$v_1 = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \qquad v_2 = \begin{bmatrix} 5 \\ 6 \\ 7 \\ 8 \end{bmatrix} \qquad m_1 = \begin{bmatrix} 1 & 2 & 3 & 4 \\ -5 & 6 & 7 & 8 \\ 9 & -10 & 11 & 12 \\ 13 & 14 & 15 & -16 \end{bmatrix} \qquad m_2 = \begin{bmatrix} 4 & 3 & 2 & 1 \\ 8 & 7 & 6 & 5 \\ 12 & 11 & 10 & 9 \\ 16 & 15 & 14 & 13 \end{bmatrix}$$

and a scalar value $s = 3.0$. The test program should print out the following result in this order:

- $s \times v_1$
- $v_1 + v_2$
- $v_1 - v_2$
- $|v|$
- $\hat{v}$
- $v_1 \cdot v_2$
- $v_1 \times v_2$
- $s \times m_1$
- $m_1 + m_2$
- $m_1 - m_2$
- $m_1 \times m_2$
- m_1^{-1}
- m_1^T
- $m_1 \times v_1$

Submission

Zip your header file, source file, and test file into a file named `lab02.zip`. Submit your `lab02.zip` file via CourseWeb before the due date stated on the CourseWeb.