Name: Taurean Roberts

Date: December 6, 2017

Assessment: Math for Games

1. **Requirements Documentation** 
   1. **Description of Problem**

**Name:** Math Library

**Problem Statement:** Making library to be used in future projects

**Problem Specifications:** Make a class of vectors and matrices ranging from 2d to 4d

* 1. **Input Information**

Coordinates on a simple plain by using the programs various axises.

* 1. **Output Information**

Gives gives the values set in the places a part of the grid

* 1. **User Interface**

The User interface allows the person using this to be specific on where the coordinates are set.

1. **System Architecture**

#pragma once

#include <iostream>

using namespace std;

class Matrix2

{

public:

//Prototype: float mMat[4]

//Description: a float array of size 4

//Precondition: takes a float up to 4 places

//Post Condition: returns 4 values

//Protection Level: Public

float mMat[4];

//Prototype: Matrix2();

//Description: default constructor

//Precondition: n/a

//Post Condition: n/a

//Protection Level: public

Matrix2();

//Prototype: Matrix2(float mat[]);

//Description: Constructor that has a float array arguement

//Precondition: takes an array

//Post Condition: gives a array

//Protection Level: Public

Matrix2(float mat[]);

//Prototype: Matrix2(float indexA, float indexB, float indexC, float indexD);

//Description: sets arguements for an array

//Precondition: takes floats into a array

//Post Condition: sets floats at set index

//Protection Level: Public

Matrix2(float indexA, float indexB, float indexC, float indexD);

//Prototype: Matrix2 operator +(Matrix2 &other);

//Description: adds matrix

//Precondition: adds two matrix arrays

//Post Condition: returns a matrix

//Protection Level: Public

Matrix2 operator +(Matrix2 &other);

//Prototype: Matrix2 operator -(Matrix2 &other);

//Description: subtracts array

//Precondition: takes two arrays

//Post Condition: returns a matrix array

//Protection Level: public

Matrix2 operator -(Matrix2 &other);

//Prototype: Matrix2 operator \*(Matrix2 &other);

//Description: scales the matrix

//Precondition: takes a matrix

//Post Condition: returns a matrix

//Protection Level: public

Matrix2 operator \*(Matrix2 &other);

//Prototype: bool operator == (Matrix2 &other);

//Description: compares a matrix

//Precondition: takes a matrix

//Post Condition: returns a matrix

//Protection Level: public

bool operator == (Matrix2 &other);

//Prototype: friend ostream& operator << (ostream &output, Matrix2 &other);

//Description: outputs a matrix

//Precondition: takes a matrix

//Post Condition: displays matrix

//Protection Level: public

friend ostream& operator << (ostream &output, Matrix2 &other);

//Prototype: friend istream& operator >> (istream &input, Matrix2 &other);

//Description: allows user input

//Precondition: gives input for matrix

//Post Condition: takes in information passed in

//Protection Level: public

friend istream& operator >> (istream &input, Matrix2 &other);

};

#pragma once

#include <iostream>

using namespace std;

class Matrix3

{

public:

//Prototype: float mMat[9];

//Description:

//Precondition:

//Post Condition:

//Protection Level: public

float mMat[9];

//Prototype:Matrix3();

//Description: default constructor

//Precondition: n/a

//Post Condition: n/a

//Protection Level: public

Matrix3();

//Prototype: Matrix3(float mat[]);

//Description: sets the array

//Precondition: takes in float array

//Post Condition: returns new array

//Protection Level: public

Matrix3(float mat[]);

//Prototype:Matrix3(float indexA, float indexB, float indexC, float indexD, float indexE, float indexF, float indexG, float indexH, float indexI);

//Description: sets the index for the matrix

//Precondition: takes in the array and give placement

//Post Condition: returns the arrays places

//Protection Level: public

Matrix3(float indexA, float indexB, float indexC, float indexD, float indexE, float indexF, float indexG, float indexH, float indexI);

//Prototype: Matrix3 operator +(Matrix3 &other);

//Description: adds matrix arrays

//Precondition: takes in one array and a place holder array

//Post Condition: returns set amount

//Protection Level: public

Matrix3 operator +(Matrix3 &other);

//Prototype: Matrix3 operator -(Matrix3 &other);

//Description: subtracts a

//Precondition: takes in a arrry

//Post Condition: returns a new array

//Protection Level: public

Matrix3 operator -(Matrix3 &other);

//Prototype: Matrix3 operator \*(Matrix3 &other);

//Description: scales the array

//Precondition: takes the original array

//Post Condition: scales it by the set size

//Protection Level: public

Matrix3 operator \*(Matrix3 &other);

//Prototype: bool operator == (Matrix3 &other);

//Description: compares the strings

//Precondition:

//Post Condition:

//Protection Level: public

bool operator == (Matrix3 &other);

//Prototype: friend ostream& operator << (ostream &output, Matrix3 &other);

//Description: shows the info input

//Precondition: takes the info

//Post Condition: displays the info

//Protection Level: public

friend ostream& operator << (ostream &output, Matrix3 &other);

//Prototype: friend istream& operator >> (istream &input, Matrix3 &other);

//Description: gives input to the user

//Precondition: takes info from user

//Post Condition: intakes info

//Protection Level: public

friend istream& operator >> (istream &input, Matrix3 &other);

};

1. **Source Code**

#include "Matrix\_2.h"

#include <iostream>

#include <math.h>

using namespace std;

Matrix2::Matrix2()

{

}

Matrix2::Matrix2(float mat[])

{

mMat[4] = mat[4];

}

Matrix2::Matrix2(float indexA, float indexB, float indexC, float indexD)

{

mMat[0] = indexA;

mMat[1] = indexB;

mMat[2] = indexC;

mMat[3] = indexD;

}

Matrix2 Matrix2::operator+(Matrix2 & other)

{

Matrix2 addNum;

addNum.mMat[0] = other.mMat[0] + mMat[0];

addNum.mMat[1] = other.mMat[1] + mMat[1];

addNum.mMat[2] = other.mMat[2] + mMat[2];

addNum.mMat[3] = other.mMat[3] + mMat[3];

return addNum;

}

Matrix2 Matrix2::operator-(Matrix2 & other)

{

Matrix2 subNum;

subNum.mMat[0] = other.mMat[0] - mMat[0];

subNum.mMat[1] = other.mMat[1] - mMat[1];

subNum.mMat[2] = other.mMat[2] - mMat[2];

subNum.mMat[3] = other.mMat[3] - mMat[3];

return subNum;

}

Matrix2 Matrix2::operator\*(Matrix2 & other)

{

Matrix2 multiNum;

multiNum.mMat[0] = ((mMat[0] \* other.mMat[0]) + (mMat[1] \* other.mMat[2]));

multiNum.mMat[1] = ((mMat[0] \* other.mMat[1]) + (mMat[1] \* other.mMat[3]));

multiNum.mMat[2] = ((mMat[2] \* other.mMat[0]) + (mMat[3] \* other.mMat[2]));

multiNum.mMat[3] = ((mMat[2] \* other.mMat[2]) + (mMat[3] \* other.mMat[3]));

return multiNum;

}

bool Matrix2::operator==(Matrix2 & other)

{

bool same = true;

for (int i = 0; i < 4; i += 1)

{

if (mMat[i] == other.mMat[i])

{

return true;

}

else if (mMat[i] != other.mMat[i])

{

same == false;

break;

}

}

return false;

}

ostream & operator<<(ostream & output, Matrix2 & other)

{

output << "< " << other.mMat[1] << " , " << other.mMat[2] << " >" << endl;

output << "< " << other.mMat[3] << " , " << other.mMat[4] << " >" << endl;

return output;

}

istream & operator >> (istream & input, Matrix2 & other)

{

input >> other.mMat[1];

input >> other.mMat[2];

input >> other.mMat[3];

input >> other.mMat[4];

return input;

}

#include <iostream>

#include "Matrix\_3.h"

using namespace std;

Matrix3::Matrix3()

{

}

Matrix3::Matrix3(float mat[])

{

mMat[9] = mat[9];

}

Matrix3::Matrix3(float indexA, float indexB, float indexC, float indexD, float indexE, float indexF, float indexG, float indexH, float indexI)

{

mMat[0] = indexA;

mMat[1] = indexB;

mMat[2] = indexC;

mMat[3] = indexD;

mMat[4] = indexE;

mMat[5] = indexF;

mMat[6] = indexG;

mMat[7] = indexH;

mMat[8] = indexI;

}

Matrix3 Matrix3::operator+(Matrix3 & other)

{

Matrix3 addNum;

addNum.mMat[0] = other.mMat[0] + mMat[0];

addNum.mMat[1] = other.mMat[1] + mMat[1];

addNum.mMat[2] = other.mMat[2] + mMat[2];

addNum.mMat[3] = other.mMat[3] + mMat[3];

addNum.mMat[4] = other.mMat[4] + mMat[4];

addNum.mMat[5] = other.mMat[5] + mMat[5];

addNum.mMat[6] = other.mMat[6] + mMat[6];

addNum.mMat[7] = other.mMat[7] + mMat[7];

addNum.mMat[8] = other.mMat[8] + mMat[8];

return addNum;

}

Matrix3 Matrix3::operator-(Matrix3 & other)

{

Matrix3 subNum;

subNum.mMat[0] = other.mMat[0] - mMat[0];

subNum.mMat[1] = other.mMat[1] - mMat[1];

subNum.mMat[2] = other.mMat[2] - mMat[2];

subNum.mMat[3] = other.mMat[3] - mMat[3];

subNum.mMat[4] = other.mMat[4] - mMat[4];

subNum.mMat[5] = other.mMat[5] - mMat[5];

subNum.mMat[6] = other.mMat[6] - mMat[6];

subNum.mMat[7] = other.mMat[7] - mMat[7];

subNum.mMat[8] = other.mMat[8] - mMat[8];

return subNum;

}

Matrix3 Matrix3::operator\*(Matrix3 & other)

{

Matrix3 multiNum;

multiNum.mMat[0] = ((mMat[0] \* other.mMat[0]) + (mMat[1] \* other.mMat[3]) + (mMat[2] \* other.mMat[6]));

multiNum.mMat[1] = ((mMat[0] \* other.mMat[1]) + (mMat[1] \* other.mMat[4]) + (mMat[2] \* other.mMat[7]));

multiNum.mMat[2] = ((mMat[0] \* other.mMat[2]) + (mMat[1] \* other.mMat[5]) + (mMat[2] \* other.mMat[8]));

multiNum.mMat[3] = ((mMat[3] \* other.mMat[0]) + (mMat[4] \* other.mMat[3]) + (mMat[5] \* other.mMat[6]));

multiNum.mMat[4] = ((mMat[3] \* other.mMat[1]) + (mMat[4] \* other.mMat[4]) + (mMat[5] \* other.mMat[7]));

multiNum.mMat[5] = ((mMat[3] \* other.mMat[2]) + (mMat[4] \* other.mMat[5]) + (mMat[5] \* other.mMat[8]));

multiNum.mMat[6] = ((mMat[6] \* other.mMat[0]) + (mMat[7] \* other.mMat[3]) + (mMat[8] \* other.mMat[6]));

multiNum.mMat[7] = ((mMat[6] \* other.mMat[1]) + (mMat[7] \* other.mMat[4]) + (mMat[8] \* other.mMat[7]));

multiNum.mMat[8] = ((mMat[6] \* other.mMat[2]) + (mMat[7] \* other.mMat[5]) + (mMat[8] \* other.mMat[8]));

return multiNum;

}

bool Matrix3::operator==(Matrix3 & other)

{

bool same = true;

for (int i = 0; i < 8; i += 1)

{

if (mMat[i] == other.mMat[i])

{

return true;

}

else if (mMat[i] != other.mMat[i])

{

same == false;

break;

}

}

return false;

}

ostream & operator<<(ostream & output, Matrix3 & other)

{

output << "< " << other.mMat[0] << " , " << other.mMat[1] << "," << other.mMat[2] << " >" << endl;

output << "< " << other.mMat[3] << " , " << other.mMat[4] << "," << other.mMat[5] << " >" << endl;

output << "< " << other.mMat[6] << " , " << other.mMat[7] << "," << other.mMat[8] << " >" << endl;

return output;

}

istream & operator >> (istream & input, Matrix3 & other)

{

input >> other.mMat[0];

input >> other.mMat[1];

input >> other.mMat[2];

input >> other.mMat[3];

input >> other.mMat[4];

input >> other.mMat[5];

input >> other.mMat[6];

input >> other.mMat[7];

input >> other.mMat[8];

return input;

}

#include <iostream>

#include "Matrix\_4.h"

#include <math.h>

Matrix4::Matrix4()

{

}

Matrix4::Matrix4(float mat[])

{

mMat[12] = mat[12];

}

Matrix4::Matrix4(float indexA, float indexB, float indexC, float indexD, float indexE, float indexF, float indexG, float indexH, float indexI, float indexJ, float indexK, float indexL, float indexM, float indexN, float indexO, float indexP)

{

mMat[0] = indexA;

mMat[1] = indexB;

mMat[2] = indexC;

mMat[3] = indexD;

mMat[4] = indexE;

mMat[5] = indexF;

mMat[6] = indexG;

mMat[7] = indexH;

mMat[8] = indexI;

mMat[9] = indexJ;

mMat[10] = indexK;

mMat[11] = indexL;

mMat[12] = indexM;

mMat[13] = indexN;

mMat[14] = indexO;

mMat[15] = indexP;

}

Matrix4 Matrix4::operator+(Matrix4 & other)

{

Matrix4 addNum;

addNum.mMat[0] = other.mMat[0] + mMat[0];

addNum.mMat[1] = other.mMat[1] + mMat[1];

addNum.mMat[2] = other.mMat[2] + mMat[2];

addNum.mMat[3] = other.mMat[3] + mMat[3];

addNum.mMat[4] = other.mMat[4] + mMat[4];

addNum.mMat[5] = other.mMat[5] + mMat[5];

addNum.mMat[6] = other.mMat[6] + mMat[6];

addNum.mMat[7] = other.mMat[7] + mMat[7];

addNum.mMat[8] = other.mMat[8] + mMat[8];

addNum.mMat[9] = other.mMat[9] + mMat[9];

addNum.mMat[10] = other.mMat[10] + mMat[10];

addNum.mMat[11] = other.mMat[11] + mMat[11];

addNum.mMat[12] = other.mMat[12] + mMat[12];

addNum.mMat[13] = other.mMat[13] + mMat[13];

addNum.mMat[14] = other.mMat[14] + mMat[14];

addNum.mMat[15] = other.mMat[15] + mMat[15];

return addNum;

}

Matrix4 Matrix4::operator-(Matrix4 & other)

{

Matrix4 subNum;

subNum.mMat[0] = other.mMat[0] - mMat[0];

subNum.mMat[1] = other.mMat[1] - mMat[1];

subNum.mMat[2] = other.mMat[2] - mMat[2];

subNum.mMat[3] = other.mMat[3] - mMat[3];

subNum.mMat[4] = other.mMat[4] - mMat[4];

subNum.mMat[5] = other.mMat[5] - mMat[5];

subNum.mMat[6] = other.mMat[6] - mMat[6];

subNum.mMat[7] = other.mMat[7] - mMat[7];

subNum.mMat[8] = other.mMat[8] - mMat[8];

subNum.mMat[9] = other.mMat[9] - mMat[9];

subNum.mMat[10] = other.mMat[10] - mMat[10];

subNum.mMat[11] = other.mMat[11] - mMat[11];

subNum.mMat[12] = other.mMat[12] + mMat[12];

subNum.mMat[13] = other.mMat[13] + mMat[13];

subNum.mMat[14] = other.mMat[14] + mMat[14];

subNum.mMat[15] = other.mMat[15] + mMat[15];

return subNum;

}

Matrix4 Matrix4::operator\*(Matrix4 & other)

{

Matrix4 multiNum;

multiNum.mMat[0] = ((mMat[0] \* other.mMat[0]) + (mMat[1] \* other.mMat[4]) + (mMat[2] \* other.mMat[8]) + (mMat[3] \* other.mMat[12]));

multiNum.mMat[1] = ((mMat[0] \* other.mMat[1]) + (mMat[1] \* other.mMat[5]) + (mMat[2] \* other.mMat[9]) + (mMat[3] \* other.mMat[13]));

multiNum.mMat[2] = ((mMat[0] \* other.mMat[2]) + (mMat[1] \* other.mMat[6]) + (mMat[2] \* other.mMat[10]) + (mMat[3] \* other.mMat[14]));

multiNum.mMat[3] = ((mMat[0] \* other.mMat[3]) + (mMat[1] \* other.mMat[7]) + (mMat[2] \* other.mMat[11]) + (mMat[3] \* other.mMat[15]));

multiNum.mMat[4] = ((mMat[4] \* other.mMat[0]) + (mMat[5] \* other.mMat[4]) + (mMat[6] \* other.mMat[8]) + (mMat[7] \* other.mMat[12]));

multiNum.mMat[5] = ((mMat[4] \* other.mMat[1]) + (mMat[5] \* other.mMat[5]) + (mMat[6] \* other.mMat[9]) + (mMat[7] \* other.mMat[13]));

multiNum.mMat[6] = ((mMat[4] \* other.mMat[2]) + (mMat[5] \* other.mMat[6]) + (mMat[6] \* other.mMat[10]) + (mMat[7] \* other.mMat[14]));

multiNum.mMat[7] = ((mMat[4] \* other.mMat[3]) + (mMat[5] \* other.mMat[7]) + (mMat[6] \* other.mMat[11]) + (mMat[7] \* other.mMat[15]));

multiNum.mMat[8] = ((mMat[8] \* other.mMat[0]) + (mMat[7] \* other.mMat[4]) + (mMat[10] \* other.mMat[8]) + (mMat[11] \* other.mMat[12]));

multiNum.mMat[9] = ((mMat[8] \* other.mMat[1]) + (mMat[7] \* other.mMat[5]) + (mMat[10] \* other.mMat[9]) + (mMat[11] \* other.mMat[13]));

multiNum.mMat[10] = ((mMat[8] \* other.mMat[2]) + (mMat[7] \* other.mMat[6]) + (mMat[10] \* other.mMat[10]) + (mMat[11] \* other.mMat[14]));

multiNum.mMat[11] = ((mMat[8] \* other.mMat[3]) + (mMat[7] \* other.mMat[7]) + (mMat[10] \* other.mMat[11]) + (mMat[11] \* other.mMat[15]));

multiNum.mMat[12] = ((mMat[12] \* other.mMat[0]) + (mMat[13] \* other.mMat[4]) + (mMat[14] \* other.mMat[8]) + (mMat[15] \* other.mMat[12]));

multiNum.mMat[13] = ((mMat[12] \* other.mMat[1]) + (mMat[13] \* other.mMat[5]) + (mMat[14] \* other.mMat[9]) + (mMat[15] \* other.mMat[13]));

multiNum.mMat[14] = ((mMat[12] \* other.mMat[2]) + (mMat[13] \* other.mMat[6]) + (mMat[14] \* other.mMat[10]) + (mMat[15] \* other.mMat[14]));

multiNum.mMat[15] = ((mMat[12] \* other.mMat[3]) + (mMat[13] \* other.mMat[7]) + (mMat[14] \* other.mMat[11]) + (mMat[15] \* other.mMat[15]));

return multiNum;

}

bool Matrix4::operator==(Matrix4 & other)

{

bool same = true;

for (int i = 0; i < 16; i += 1)

{

if (mMat[i] == other.mMat[i])

{

return true;

}

else if (mMat[i] != other.mMat[i])

{

same == false;

break;

}

}

return false;

}

ostream & operator<<(ostream & output, Matrix4 & other)

{

output << "< " << other.mMat[0] << " , " << other.mMat[1] << " , " << other.mMat[2] << " , " << other.mMat[3] << " >" << endl;

output << "< " << other.mMat[4] << " , " << other.mMat[5] << " , " << other.mMat[6] << " , " << other.mMat[7] << " >" << endl;

output << "< " << other.mMat[8] << " , " << other.mMat[9] << " , " << other.mMat[10] << " , " << other.mMat[11] << " >" << endl;

output << "< " << other.mMat[12] << " , " << other.mMat[13] << " , " << other.mMat[14] << " , " << other.mMat[15] << " >" << endl;

return output;

}

istream & operator >> (istream & input, Matrix4 & other)

{

input >> other.mMat[0];

input >> other.mMat[1];

input >> other.mMat[2];

input >> other.mMat[3];

input >> other.mMat[4];

input >> other.mMat[5];

input >> other.mMat[6];

input >> other.mMat[7];

input >> other.mMat[8];

input >> other.mMat[9];

input >> other.mMat[10];

input >> other.mMat[11];

input >> other.mMat[12];

input >> other.mMat[13];

input >> other.mMat[14];

input >> other.mMat[15];

return input;

}

#include "Vector\_2.h"

#include <iostream>

#include <string>

#include <math.h>

using namespace std;

Vector2::Vector2()

{

}

Vector2::Vector2(float x, float y)

{

mX = x;

mY = y;

}

Vector2 Vector2::operator+(Vector2 & other)

{

Vector2 Sum;

Sum.mX = other.mX + mX;

Sum.mY = other.mY + mY;

return Sum;

}

Vector2 Vector2::operator-(Vector2 & other)

{

Vector2 Sub;

Sub.mX = other.mX - mX;

Sub.mY = other.mY - mY;

return Sub;

}

Vector2 Vector2::operator\*(float & other)

{

Vector2 scale;

scale.mX = mX \* other;

scale.mY = mY \* other;

return scale;

}

bool Vector2::operator==(Vector2 &other)

{

if ((mX != other.mX) || (mY != other.mY))

{

return false;

}

else

return true;

}

float Vector2::Dot(Vector2 & other)

{

float mDot;

mDot = ((mX \* other.mX) + (mY \* other.mY));

return mDot;

}

float Vector2::Magnitude()

{

float mag;

mag = sqrt((mX \* mX) + (mY \*mY));

return mag;

}

Vector2 Vector2::Normalize()

{

Vector2 mNorm;

mNorm.mX = mX / Magnitude();

mNorm.mY = mY / Magnitude();

return mNorm;

}

ostream & operator<<(ostream & output, Vector2 &other)

{

output << "< " << other.mX << " , " << other.mY << " >" << endl;

return output;

}

istream & operator >> (istream &input, Vector2 &other)

{

input >> other.mX;

input >> other.mY;

return input;

}

#include "Vector\_3.h"

#include "math.h"

Vector3::Vector3()

{

}

Vector3::Vector3(float x, float y, float z)

{

mX = x;

mY = y;

mZ = z;

}

Vector3 Vector3::operator+(Vector3 & other)

{

Vector3 sum;

sum.mX = other.mX + other.mX + mX;

sum.mY = other.mY + other.mY + mY;

sum.mZ = other.mZ + other.mZ + mZ;

return sum;

}

Vector3 Vector3::operator-(Vector3 & other)

{

Vector3 sub;

sub.mX = other.mX - other.mX - mX;

sub.mY = other.mY - other.mY - mY;

sub.mZ = other.mZ - other.mZ - mZ;

return sub;

}

Vector3 Vector3::operator\*(float & other)

{

Vector3 scale;

scale.mX = mX \* other;

scale.mY = mY \* other;

scale.mZ = mZ \* other;

return scale;

}

bool Vector3::operator==(Vector3 & other)

{

if ((mX != other.mX) || (mY != other.mY) || (mZ != other.mZ))

{

return false;

}

else

return true;

}

float Vector3::Dot(Vector3 & other)

{

float mDot;

mDot = ((mX \* other.mX) + (mY \* other.mY) + (mZ \* other.mZ));

return mDot;

}

Vector3 Vector3::Cross(Vector3 & other)

{

Vector3 mCross;

mCross.mX = ((mY \* other.mZ) - (mZ \* other.mY));

mCross.mY = ((mZ \* other.mX) - (mX \* other.mZ));

mCross.mZ = ((mX \* other.mY) - (mY \* other.mX));

return mCross;

}

float Vector3::Magnitude()

{

float mag;

mag = sqrt((mX \* mX) + (mY \* mY) + (mZ \* mZ));

return mag;

}

Vector3 Vector3::Normalize()

{

Vector3 mNorm;

mNorm.mX = mX / Magnitude();

mNorm.mY = mY / Magnitude();

mNorm.mZ = mZ / Magnitude();

return mNorm;

}

ostream & operator<<(ostream & output, Vector3 & other)

{

output << "< " << other.mX << " , " << other.mY << " , " << other.mZ << " > " << endl;

return output;

}

istream & operator >> (istream & input, Vector3 & other)

{

input >> other.mX;

input >> other.mY;

input >> other.mZ;

return input;

}

#include "Vector\_4.h"

#include "math.h"

Vector4::Vector4()

{

}

Vector4::Vector4(float x, float y, float z, float w)

{

mX = x;

mY = y;

mZ = z;

mW = w;

}

Vector4 Vector4::operator+(Vector4 & other)

{

Vector4 sum;

sum.mX = other.mX + other.mX + mX;

sum.mY = other.mY + other.mY + mY;

sum.mZ = other.mZ + other.mZ + mZ;

sum.mW = other.mW + other.mW + mW;

return sum;

}

Vector4 Vector4::operator-(Vector4 & other)

{

Vector4 sub;

sub.mX = other.mX - other.mX - mX;

sub.mY = other.mY - other.mY - mY;

sub.mZ = other.mZ - other.mZ - mZ;

sub.mW = other.mW - other.mW - mW;

return sub;

}

Vector4 Vector4::operator\*(float & other)

{

Vector4 scale;

scale.mX = mX \* other;

scale.mY = mY \* other;

scale.mZ = mZ \* other;

scale.mW = mW \* other;

return scale;

}

bool Vector4::operator==(Vector4 & other)

{

if ((mX != other.mX) || (mY != other.mY) || (mZ != other.mZ) || (mW != other.mW))

{

return false;

}

else

return true;

}

float Vector4::Dot(Vector4 & other)

{

float mDot;

mDot = ((mX \* other.mX) + (mY \* other.mY) + (mZ \* other.mZ) || (mW \* other.mW));

return mDot;

}

float Vector4::Magnitude()

{

float mag;

mag = sqrt((mX \* mX) + (mY \* mY) + (mZ \* mZ) + (mW \* mW));

return mag;

}

Vector4 Vector4::Normalize()

{

Vector4 mNorm;

mNorm.mX = mX / Magnitude();

mNorm.mY = mY / Magnitude();

mNorm.mZ = mZ / Magnitude();

mNorm.mW = mW / Magnitude();

return mNorm;

}

ostream & operator<<(ostream & output, Vector4 & other)

{

output << "< " << other.mX << " , " << other.mY << " , " << other.mZ << " , " << other.mW << " >" << endl;

return output;

}

istream & operator >> (istream & input, Vector4 & other)

{

input >> other.mX;

input >> other.mY;

input >> other.mZ;

input >> other.mW;

return input;

}

1. **Read Me**

(Be very clear as to how the assessor should go about getting your application, running it, and using it. You should assume the assessor knows nothing about your application.)