Purpose

The purpose of this assignment is to develop a collection of classes that allow us to perform linear algebra calculations.

Problem

Task 1

You will develop a class of vectors called Vector. You are provided with the header file Vector.h

```
#ifndef VECTORHEADERDEF
#define VECTORHEADERDEF

class Vector
{
private:
   double* mData; // data stored in vector
   int mSize; // size of vector
public:
   Vector(const Vector& otherVector);
   Vector(int size);
   Vector(int size, double v[]);
   ~Vector();
   int GetSize() const;
   double& operator[](int i); // zero-based indexing
   // read-only zero-based indexing
   double Read(int i) const;
   double& operator()(int i); // one-based indexing
   // assignment
   Vector& operator=(const Vector& otherVector);
   Vector operator+() const; // unary +
   Vector operator-() const; // unary -
   Vector operator+(const Vector& v1) const; // binary +
   Vector operator-(const Vector& v1) const; // binary -
   // scalar multiplication
   Vector operator*(double a) const;
};

#endif
```

Overloading the Square Bracket Operator

If v is a vector, then v[i] returns the entry of v with index i using zero-based indexing. This method first checks that the index falls within the correct range i.e. a nonnegative integer that is less than mSize and returns a reference to the value stored in this entry of the vector.

Read-Only Access to Vector Entries

To guarantee that some functions which read from a vector do not change it, we also supply a read-only const version. Similar to the square bracket operator but read-only version. Uses zero-based indexing.

Overloading the Round Bracket Operator

Access entries of a vector using one-based indexing. Notation similar to MATLAB. Same validation performed as the Square Bracket Operator overload.

Assignment Operator Overload

First checks that the vector on the left-hand side of the assignment statement is of the same size as the vector on the right-hand side. If so the entries of the vector on the right-hand side are copied into the vector on the left-hand side.

Unary Operator Overload

First declares a vector of the same size as the vector that the unary operator is applied to, entries of the new vector are then set to the appropriate value before the vector is returned.

Binary Operator Overload

First checks that the two vectors that are operated on are the same size. If they are, a new vector of the same size is created. The entries of this new vector are assigned based on the operation and the new vector is then returned.

Task 2

You will develop a class of matrices called Matrix. You are provided with the header file Matrix.h

```
#ifndef MATRIXHEADERDEF
#define MATRIXHEADERDEF
#include "Vector.h"
class Matrix
{
private:
   double** mData; // entries of matrix
   int mNumRows, mNumCols; // dimensions
public:
   Matrix(const Matrix& otherMatrix);
   Matrix(int numRows, int numCols);
   Matrix(int numRows, int numCols, double v[][3]);
   ~Matrix();
   int GetNumberOfRows() const;
   int GetNumberOfColumns() const;
   double Read(int i, int j) const;
   double& operator()(int i, int j); //1-based indexing
   //overloaded assignment operator
   Matrix& operator=(const Matrix& otherMatrix);
   Matrix operator+() const; // unary +
   Matrix operator-() const; // unary -
   Matrix operator+(const Matrix& m1) const; // binary +
   Matrix operator-(const Matrix& m1) const; // binary -
   // scalar multiplication
   Matrix operator*(double a) const;
   double CalculateDeterminant() const;
   // declare vector multiplication friendship
   friend Vector operator*(const Matrix& m,
                           const Vector& v);
   friend Vector operator*(const Vector& v,
                           const Matrix& m);
};
// prototype signatures for friend operators
Vector operator*(const Matrix& m, const Vector& v);
Vector operator*(const Vector& v, const Matrix& m);
#endif
```

Task 3

You will develop a class of called LinearSytem that may be used to solve linear systems. A linear system is determined by the size of the linear system, a square matrix, and vector (representing the right-hand side), with the matrix and vector being of compatible sizes.

A public method Solve should be written to solve tis linear system by Gaussian elimination with pivoting and row reduced Echelon technique.

You are provided the header file LinearSystem.h

```
#ifndef LINEARSYSTEMHEADERDEF
#define LINEARSYSTEMHEADERDEF
#include "Vector.h"
#include "Matrix.h"

class LinearSystem
{
private:
   int mSize; // size of linear system
   Matrix* mpA;  // matrix for linear system
   Vector* mpb;  // vector for linear system

   // Only allow constructor that specifies matrix and
   // vector to be used.  Copy constructor is private.
   LinearSystem(const LinearSystem& otherLinearSystem){};
public:
   LinearSystem(const Matrix& A, const Vector& b);

   // destructor frees memory allocated
   ~LinearSystem();

   // Method for solving system
   virtual Vector Solve();
};

#endif
```