CSCI 335
First programming assignment (100 points)
Due September 11

Please follow the blackboard instructions on writing and submitting programming assignments.
We will not debug your assignment. It should compile to receive any credit. It should run correctly to receive full credit.
Make sure to include a README.txt file with your submission where you state what you have completed.

Programming: Matrix manipulation

Create and test a class called **Matrix**. For instance a two by two matrix of integers could be
[10 20
 1 0].
A three by one matrix of doubles could be
[10.0
 2.0
 50.0].
A two by two matrix of strings could be
["aa" "hi"
 "hey" "there"].

A matrix can have any dimension (i.e. N rows and M columns where N, M are integers greater than or equal to one). An empty matrix has 0 rows and 0 columns.

The purpose of this assignment is to have you create a matrix class from scratch without using the STL. In the sample code provided in the book, you can see an initial implementation of the matrix class that uses a vector. You can start with this implementation but you can't use a vector class as a private data member. You should use pointers instead. The private data members should be:

```
size_t num_cols_; size_t num_rows_; Comparable **array_;
```

Comparable is the template type parameter.

Pay special attention to Weiss's **"big five",** the destructor, copy constructor, copy assignment operator, move constructor and move assignment operator.

This assignment will help you revisit constructors, destructors, overloading of operators, and templates.

PART 1 [50 points]

Implement the “big five”, along with a function to fill in the matrix (call it ReadMatrix()) as well as a function that overloads the output stream << operator for the matrix.

Demonstrate that you are able to read and write correctly by including the following code:

```
void TestPart1() {

  Matrix<int> a, b;  //Two empty matrices are created
  cout << a.NumRows() << " " << a.NumCols() << endl; // yields 0 0

  a.ReadMatrix(); // User provides number of rows, columns and matrix values.

  cout << a << endl;  // Output should be what user entered.
  b.ReadMatrix();
  cout << b << endl;

  a = b;  // Should call the copy assignment operator for a.
  cout << a << endl;

  Matrix<int> c = b;  // Should call the copy constructor for a.
  cout << c << endl;

  Matrix<int> d = std::move(c);  // Move constructor for d.
  cout << d << endl;

  a = std:move(d);  // Move assignment operator for a.
  cout << a << endl;
}  // Destructors will be called.
```

PART 2 [40 points]

Overload the + and [] operators for your Matrix class. Test with the following code.

```
void TestPart2() {

  Matrix<string> a, b;

  a.ReadMatrix();  // User provides input for matrix a.

  cout << a << endl;
  b.ReadMatrix();  // User provides input for matrix b.
  cout << b << endl;

  cout << a + b << endl; // Matrices should have same size.
                         // The default + operator for strings
                         // will be used.

  Matrix<string> d = a + b;
  cout << d <<endl;

  cout << d + "hi_there";
  PrintVec(a[1]);  // Should print the first row of a.
  PrintVec(b[2]);  // Should be print the second row of b.
                   // Note, that the [] operator should return
                   // a vector object.
                   // PrintVec() is a templated function that
                   // couts the elements of a vector.
}  // End of TestPart2.
```

PART 3 [10 points]

Implement the Swap() function for a Matrix using moves. If you have correctly implemented the move constructors and assignment operators, the Swap() will be straightforward. You have to swap all private variables. Test with the following code.

```
void TestPart3() {

  Matrix<double> a, b;

  a.ReadMatrix();  // User provides input for matrix a.

  cout << a << endl;
  b.ReadMatrix();  // User provides input for matrix b.
  cout << b << endl;

  a.Swap(b);

  cout << a << endl;
  cout << b << endl;
}  // End of TestPart3.
```