5613B - Introduction to C++
Hilary Term - 2014-2015
Homework 1 - Due Feb. 9th, 2015

1. In analogy to the `Vector` container class, write a `Matrix` container class representing a double precision matrix with a number of rows and columns which is determined at run-time. This class should have the following functionality:

   (a) a constructor which takes two integers representing the size of the matrix.

   (b) `nrow()`, `ncol()` members which return the size of the matrix.

   (c) an overloaded `()` operator which takes two integers and returns the corresponding matrix element, so that element (i, j) of matrix a can be accessed as `a(i,j)`.

   (d) an overloaded `*=` operator taking a double precision argument to rescale each matrix element, so that all elements of the matrix a can be multiplied by a double d via the statement `a *= d;`. Enable the possibility of compound expressions by having this member function return a reference to the current state of the class using the `*this` pointer.

   Note that this class should safely cleanup any resources it allocates and members which do not change the internal state of the class should be accessible in modules to which `const` arguments are passed.

2. Write some additional functions (not members of the class) which use `Matrix`:

   (a) a function `mult(a,b,c)` (`a`, `b`, and `c` are matrices) which performs the matrix multiplication $a = b \times c$.

   (b) a function `add(a,b,c)` (`a`, `b`, and `c` are matrices) which performs element-wise matrix addition $a = b + c$.

   (c) an overloaded stream insertion operator with function declaration

   ```
   ostream& operator<< (ostream& strm, const Matrix& mat);
   ```

   which writes the matrix elements to an output stream and returns the stream afterward.

   (d) an overloaded stream extraction operator with function declaration

   ```
   istream& operator>> (istream& strm, Matrix& mat);
   ```

   which reads and assigns the matrix elements from an input stream and returns the stream afterward.

   (e) overloaded `!=` and `==` operators which compare two matrices. You may find the following C++ floating-point comparison function useful:

```
#include <limits>
#include <cmath>

bool not_equal(const double& a, const double& b) {

  return std::abs(a-b)>(std::abs(std::min(a,b))*
    std::numeric_limits<double>::epsilon()*20.0);
}
```

These functions can be declared in `matrix.h` and defined in `matrix.cc`, or declared in a separate header (e.g. `matrix_utils.h`) and defined in a separate `.cc` file (e.g. `matrix_utils.cc`).

3. Write a `main` program which tests all the functionality of the previous parts. In particular, a good test might be to write a matrix out to a file, read it in to another matrix, and test if the resulting matrices are equal.