CS11 Intro C++ Lab 7: Integer Matrix II

Hopefully you didn't find last week's lab too difficult to get working properly. This week you will continue to work on your `Matrix` class, adding matrix arithmetic operators (+, - and *, along with the corresponding compound-assignment operations), and also implementing *move operations* so that the compiler can use your class as efficiently as possible.

Move Operations

C++ is very careful about move operations: If your class implements its own custom copy comstructor, copy-assignment operator, or destructor, the compiler assumes that the *default* move operations will likely be incorrect for your class, and it will not generate any move operations. Is this a problem? Not really! Move operations are about improving performance, not maintaining correctness. That said, a move operation that functions incorrectly will definitely make your program crash or behave in confusing ways. Therefore, since it won't break the program to leave out the move operations, the compiler does the safe thing and leaves them out.

However, if you have a class where move operations are a good idea, and you already need to implement copy-construction and copy-assignment, then you also need to implement your own custom move-constructor and move-assignment operators that do the Right Thing for your class. This is an example of the Rule of Five, which is the C++11-and-later equivalent of the Rule of Three.

Provide implementations of the move operations on your `Matrix` class:

- `Matrix(Matrix &&m)` (the move-constructor) moves the contents of *m* into the new object being initialized. Don't forget to leave *m* in a state that will allow the `Matrix` destructor to run properly.

- `Matrix & Matrix::operator=(Matrix &&m)` (the move-assignment operator) moves the contents of *m* into the LHS of the assignment. Don't forget to release any memory currently held by the LHS, and don't forget to leave *m* in a state that will allow the `Matrix` destructor to run properly.

Other Operations

To give us a proper opportunity to exercise move operations, we should add the simple arithmetic and compound-assignment operators that are relevant to matrices! Provide implementations of these operators:

- Compound assignment operators (`+=`, `-=`, and `*=`), which should be implemented as member operator-overloads.

- Simple arithmetic operators (`+`, `-`, and `*`), which should be implemented as non-member operator-overloads.

- These functions should throw an `invalid_argument` exception when the dimensions of the matrices are not compatible. For example, if a 2x2 matrix is being added to a 5x5 matrix, an exception should be thrown. You can specify whatever error message makes sense to you; we do not test for a specific error message, but make sure the message corresponds to the general issue or you may receive a deduction.

- It is easiest to implement `+` in terms of `+=`, and `-` in terms of `-=`.

- Matrix multiplication is another beast entirely, since an *R*x_S_ matrix multiplied with an *S*x_T_ matrix will produce an *R*x_T_ matrix. Most students tend to find it easiest to implement `*=` in terms of `*`.

Coding Style

As with last week, comment all new functionality thoroughly in the Doxygen style, including a class-level description, and a comment for every data-member and member-function. (Comments may be brief if the function's purpose is obvious, but you must still comment all of these things.)

You do not need to create a `Doxyfile` or run Doxygen this time, as long as you follow the Doxygen format properly.

Make sure to always identify what arguments are illegal. Additionally, always document any exceptions that may be thrown, and the circumstances that would cause them to be thrown.

Testing

A test suite for your class is provided here; make sure to fix any test failures you encounter.

Build Automation

As before, you will need to submit a `Makefile` for your project that builds and tests your `Matrix` code.

- The `all` target should build the test binary, but not run it. **This week, make**

sure you are using the `test-matrix2` code, since it includes tests for the new functionality, as well as the tests for last week's functionality.

- The `test` target should run the test binary.
- The `clean` target should delete `.o` files and binaries. **As always, make sure to back up your code before testing your `clean` target for the first time!**
- Make sure all files are compiled with the `-Wall` and `-Werror` arguments (these will go in your `CXXFLAGS` variable) so that any malformed but otherwise legal code is identified by the compiler.

Submitting Your Work

Once you have completed the above tasks, submit your work through csman. Make sure to submit these files:

- `Makefile`
- `matrix.h`
- `matrix.cpp` (if present)

You do not need to submit the test code; we will test your program with a fresh copy of these files.

Assignment Feedback Survey

Please also complete and submit a feedback survey with your submission, telling us about your experience with this assignment. Doing so will help us to improve CS11 Intro C++ in the future.
