Breakout / Lab 05: **(Lab 04, Part 2)**

Even More Operator Overloading

CSCI 1730 – Fall 2015

NOTE: This breakout lab assignment serves as an extension to the previous lab both with respect to time (you have another week to work on Lab 04) and with respect to content (this lab is going to have you do more work).

Problem / Exercise

In addition to finishing the requirements for Lab 04, you need to make sure your `Matrix` class supports the following features:

- **Overloaded Function Call Operator (`operator()(uint row, uint col)`):** After creating a (non-dynamically allocated) `Matrix` object, the user should be able to access the elements using the function call operator (as an alternative to using the `at` function):

  ```
  Matrix a(1, 1);
  a(0, 0) = 5;
  cout << a(0,0) << endl;
  ```

- **Overloaded Copy Assignment Operator (`operator=(const Matrix &)`):** You should have already overloaded the assignment operator to take in a special kind of initializer list. Now you need to provide an additional overload that supports copy assignment. This will make your `Matrix` class more consistent since copy assignment parallels copy construction. Here is an example:

  ```
  Matrix a(1, 1);
  a(0, 0) = 5;

  Matrix b(1, 1);
  b = a; // copy assignment
  ```

- **Overloaded Non-Member Arithmetic Operators for Scalers:** You should have already created overloads to support the basic arithmetic operations where the right-hand-side of an operation is a scaler value. Now you need to implement operator overloads so that scalers can be used on the left-hand-side of an operation. Here is an example showing the operators that you need to support:

  ```
  Matrix a = {{1, 2},
              {3, 4}};

  Matrix b = 4.0 + a; // [ 5, 6 ]
                      // [ 7, 8 ]

  Matrix c = 4.0 - a; // [ 3, 2 ]
                      // [ 1, 0 ]

  Matrix d = 2.0 * a; // [ 2, 4 ]
                      // [ 6, 8 ]

  Matrix e = 12.0 / a; // [ 12, 6 ]
                         [  4, 3 ]
  ```

- **Overloaded Unary Minus Operator (`operator-()`):** You need to support negating your `Matrix` objects:

  ```
  Matrix a = {{1, 2}};
  cout << -a << endl; // [ -1, -2 ]
  ```

Important Details

Make sure that adhere to the following guidelines:

- You **MAY** assume valid input for all operations.
- Make functions and overloads const-qualified whenever it makes sense to do so.
- Don't be afraid to ask the members of your group for help. Remember, we do ask you to include the names of all group members in your individual brainstorm. If you do ask for help, then make sure you understand the solution. There is a more than reasonable expectation that you should be able to reproduce any code you submit.

1 Group Brainstorm

You are NOT allowed to use the computers during this time.

Breakup into groups based on your seating and brainstorm about how to solve the problem or exercise. Make sure everyone understands the problem, and sketch out potential ways to move towards a solution. Perhaps something that was discussed during lecture might be useful?

2 Submit Individual Brainstorm

You may use a computer from this point forward.

Login to eLC and submit a version of your group's brainstorm, written in your own words. You may add additional information if you want. You need to write enough in order to convince the grader that you understand the problem or exercise and that you have a plan for moving forward towards a solution. Please include the last names of the other people in your group in your submission. The brainstorm submission should be available on eLC in your assignment dropbox. We prefer that you submit your individual brainstorms before the end of your breakout period, however, you generally have until 3PM on the day of your breakout (as indicated on eLC) to submit them.

NOTE: Submissions that do not include an individual brainstorm will not be graded.

NOTE: Once you have completed your individual brainstorm, please use the remainder of the lab period to actually begin work on the lab. Failure to do this may result in the taking of attendance.

3 C++ Code & Program

3.1 Setup

Make sure that all of your files are in a directory called `LastName-FirstName-lab04`, where `LastName` and `FirstName` are replaced with your actual last and first names, respectively.

3.2 Source Code Files

For this lab, you should organize your breakout lab into the following files:

- `Matrix.h`: This file should include the class prototype presented above as well as the prototypes for operator overloads that you implement. You **MAY NOT** modify the function prototypes that are included in the `Matrix` class prototype. However, you may add additional function prototypes and variables to the class prototype as needed. Make sure that this header file also includes a header guard (i.e., the `#ifndef` macro, etc.).
- `Matrix.cpp`: This file should contain the implementation of your class's functions as well as the implementation of any operator overloads that you implement.
- `lab04.cpp`: This file should contain a small/moderately sized driver that demonstrates the full range of functionality of your `Matrix` class.

Additionally, make sure that you adhere to the following:

- All functions must be documented using Javadoc-style comments. Use inline documentation, as needed, to explain ambiguous or tricky parts of your code.

3.3 `Makefile` File

You need to include a `Makefile`. Your `Makefile` needs to compile and link separately. Make sure that your `Matrix.cpp` file compiles to `Matrix.o`. This is very important because we will be testing your submission by linking against your `Matrix.o` file. The resulting executable should be called `lab04`.

Make sure that when you compile, you pass the following options to `g++` in addition to the `-c` option:

```
-Wall -std=c++11 -g -O0 -pedantic-errors
```

Here is a link to the `Makefile` for the "gradebook" example we coded up in class that you might find useful as a reference point: `https://gist.github.com/mepcotterell/45a10fa72b208a76d968`.

3.4 `README` File

Make sure to include a `README` file that includes the following information presented in a reasonably formatted way:

- Your Name and 810/811#
- Instructions on how to compile and run your program.
- A Reflection Section. In a paragraph or two, compare and contrast what you actually did to complete the problem or exercise versus what you wrote in your initial brainstorm. How will this experience impact future planning/brainstorms?

Here is a partially filled out, example `README` file: `https://gist.github.com/mepcotterell/3ce865e3a151a3b49ec3`.

NOTE: Try to make sure that each line in your `README` file does not exceed 80 characters. Do not assume line-wrapping. Please manually insert a line break if a line exceeds 80 characters.

3.5 Compiler Warnings

Since you should be compiling with both the `-Wall` and `pedantic-error` options, your code is expected to compile without `g++` issuing any warnings. For this lab, compiling without warnings will be one or more of the test cases.

3.6 Memory Leaks

Since this breakout lab makes use of dynamic memory allocation, you are expected to ensure that your `Matrix` implementation doesn't result in any memory leaks. We will test for memory leaks using the `valgrind` utility. For this lab, having no memory leaks will be one or more of the test cases.

4 Submission

Before the day of the next breakout session, you need to submit your code. You will still be submitting your project via nike. Make sure your work is on `nike.cs.uga.edu` in a directory called `LastName-FirstName-lab05`. From within the parent directory, execute the following command:

```
$ submit LastName-FirstName-lab05 cs1730a
```

It is also a good idea to email a copy to yourself. To do this, simply execute the following command, replacing the email address with your email address:

```
$ tar zcvf LastName-FirstName-lab05.tar.gz LastName-FirstName-lab05
$ mutt -s "lab01" -a LastName-FirstName-lab05.tar.gz -- your@email.com < /dev/null
```