

CS2040C Data Structures & Algorithms

Assignment #0

Objective

This assignment is meant to introduce you to the basics of working with C++, including compiling code with CMake and unit testing. We will use the same project structure throughout the semester. If you're able to build and work with this project you should be able to work with all future assignments. You will be provided a zip file that contains:

- RGB.h, a header file containing the definition of the RGB class
- RGB.cpp with empty function definitions
- RGB_test.cpp with some example unit tests
- CMakeLists.txt for building and running the project

Your task is to complete the RGB class so that it passes the unit tests. You may modify any of the files, but will only submit the contents of RGB.cpp.

RGB

The RGB class models a colour as independent red, green, and blue channel values from 0 to 255. The skeleton you are given includes a constructor, accessors, and print functions. Note that the class is *immutable*, in that once created the values cannot be changed. You are to implement methods to mix and scale colours.

Mix

Calling the mix method of an RGB instance and passing it another instance produces a new RGB instance with each channel set to the sum of the two inputs. For example `RGB(10, 20, 30).mix(200, 100, 0)` produces `RGB(210, 120, 30)`. Note that outputs must be bound to `[0, 255]`.

Scale

Calling the scale method of an RGB instance scales each of the channel values by the floating point value provided, rounded to the nearest integer and bound to `[0, 255]`.

Building

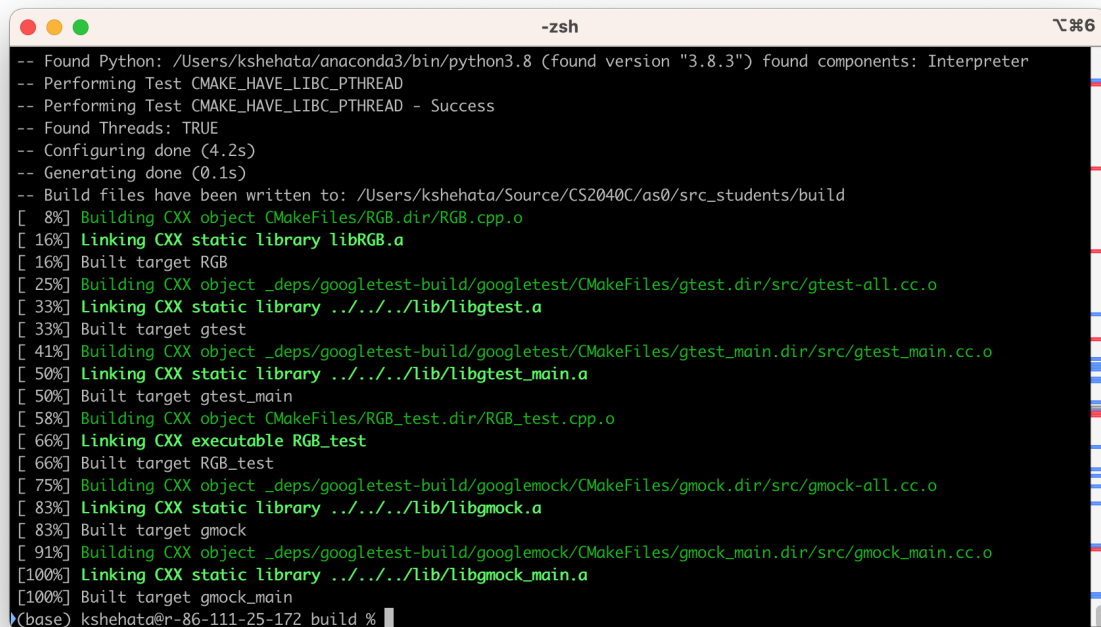
Building the project requires [CMake](#). You can install CMake on its own, or via [Visual Studio Code's C++ Extension Pack](#). We suggest the latter, particularly on lab computers, but the advantage to CMake is that it supports many environments, including Linux, Mac (both XCode and command line), Windows (both Visual Studio and command line), and many others. On Windows you can build via CMake from Visual Studio directly. To use the Visual Studio Code method:

1. Install Visual Studio Code (if not already installed)
2. Go to Extensions and make sure the "C/C++ Extension Pack" is installed.
3. Open the folder that contains the source code
4. Press ctrl-shift-p and type "CMake run tests".
5. It may ask you for which compiler to use. On Windows choose the most recent version of Visual Studio available.
6. It should build the project and run your unit tests directly.

Building and running from the command line is relatively straightforward. From the folder containing the source code, run:

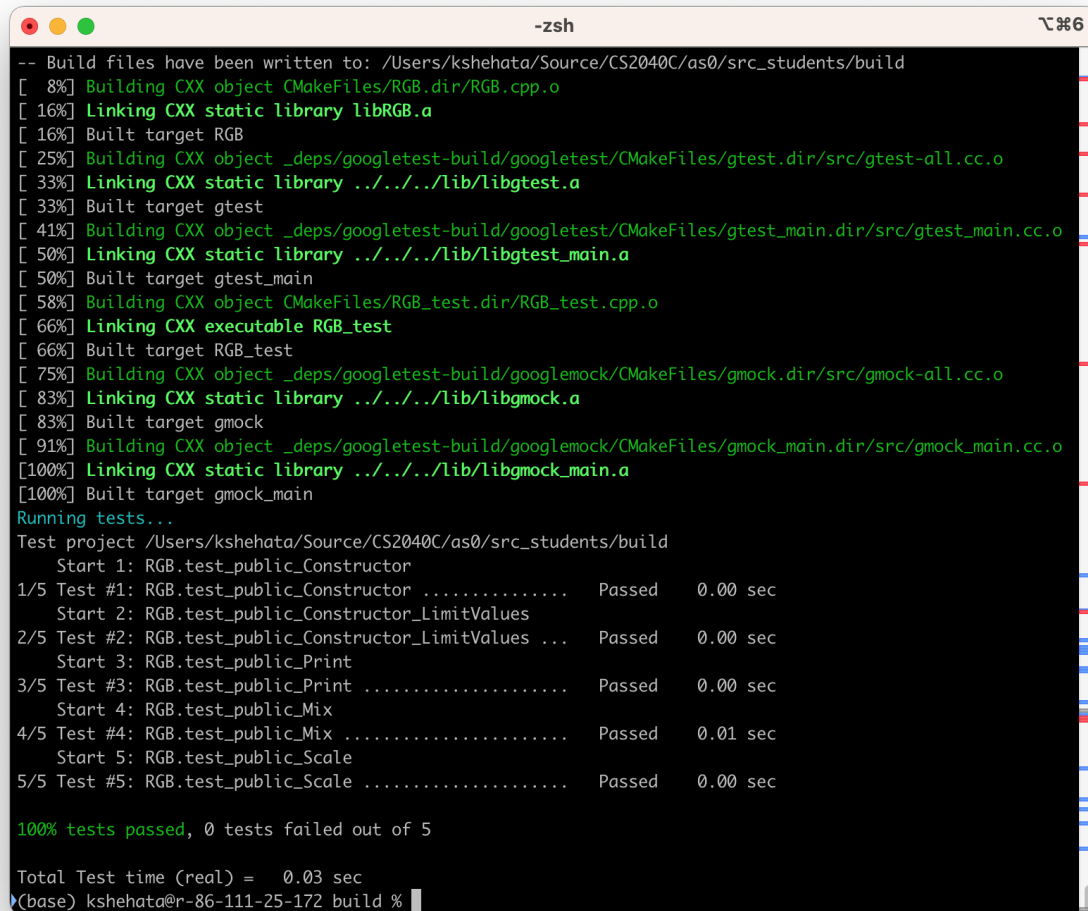
```
mkdir build && cd build
cmake ..
make
```

If you get output similar to what's below then you've built the project successfully:

A terminal window titled "-zsh" with a dark background and light-colored text. The output shows the CMake configuration and build process. It starts with finding Python 3.8, then configuring the project. The build process is shown with progress bars for each target: RGB, gtest, gtest_main, RGB_test, gmock, and gmock_main. The final output shows the project has been built successfully.

```
-- Found Python: /Users/kshehata/anaconda3/bin/python3.8 (found version "3.8.3") found components: Interpreter
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD - Success
-- Found Threads: TRUE
-- Configuring done (4.2s)
-- Generating done (0.1s)
-- Build files have been written to: /Users/kshehata/Source/CS2040C/as0/src_students/build
[ 8%] Building CXX object CMakeFiles/RGB.dir/RGB.cpp.o
[ 16%] Linking CXX static library libRGB.a
[ 16%] Built target RGB
[ 25%] Building CXX object _deps/googletest-build/googletest/CMakeFiles/gtest.dir/src/gtest-all.cc.o
[ 33%] Linking CXX static library ../lib/libgtest.a
[ 33%] Built target gtest
[ 41%] Building CXX object _deps/googletest-build/googletest/CMakeFiles/gtest_main.dir/src/gtest_main.cc.o
[ 50%] Linking CXX static library ../lib/libgtest_main.a
[ 50%] Built target gtest_main
[ 58%] Building CXX object CMakeFiles/RGB_test.dir/RGB_test.cpp.o
[ 66%] Linking CXX executable RGB_test
[ 66%] Built target RGB_test
[ 75%] Building CXX object _deps/googletest-build/googletest/CMakeFiles/gmock.dir/src/gmock-all.cc.o
[ 83%] Linking CXX static library ../lib/libgmock.a
[ 83%] Built target gmock
[ 91%] Building CXX object _deps/googletest-build/googletest/CMakeFiles/gmock_main.dir/src/gmock_main.cc.o
[100%] Linking CXX static library ../lib/libgmock_main.a
[100%] Built target gmock_main
(base) kshehata@r-86-111-25-172 build %
```

Eventually

A terminal window titled "-zsh" with a macOS-style title bar (red, yellow, green buttons). The terminal shows the output of a CMake build and test process. The build progress is shown in green text with percentage indicators in brackets. The test results are shown in white text with a table format. The terminal background is black.

```
-- Build files have been written to: /Users/kshehata/Source/CS2040C/as0/src_students/build
[ 8%] Building CXX object CMakeFiles/RGB.dir/RGB.cpp.o
[16%] Linking CXX static library libRGB.a
[16%] Built target RGB
[25%] Building CXX object _deps/googletest-build/googletest/CMakeFiles/gtest.dir/src/gtest-all.cc.o
[33%] Linking CXX static library ../../lib/libgtest.a
[33%] Built target gtest
[41%] Building CXX object _deps/googletest-build/googletest/CMakeFiles/gtest_main.dir/src/gtest_main.cc.o
[50%] Linking CXX static library ../../lib/libgtest_main.a
[50%] Built target gtest_main
[58%] Building CXX object CMakeFiles/RGB_test.dir/RGB_test.cpp.o
[66%] Linking CXX executable RGB_test
[66%] Built target RGB_test
[75%] Building CXX object _deps/googletest-build/googlemock/CMakeFiles/gmock.dir/src/gmock-all.cc.o
[83%] Linking CXX static library ../../lib/libgmock.a
[83%] Built target gmock
[91%] Building CXX object _deps/googletest-build/googlemock/CMakeFiles/gmock_main.dir/src/gmock_main.cc.o
[100%] Linking CXX static library ../../lib/libgmock_main.a
[100%] Built target gmock_main
Running tests...
Test project /Users/kshehata/Source/CS2040C/as0/src_students/build
  Start 1: RGB.test_public_Constructor
1/5 Test #1: RGB.test_public_Constructor ..... Passed    0.00 sec
  Start 2: RGB.test_public_Constructor_LimitValues
2/5 Test #2: RGB.test_public_Constructor_LimitValues ... Passed    0.00 sec
  Start 3: RGB.test_public_Print
3/5 Test #3: RGB.test_public_Print ..... Passed    0.00 sec
  Start 4: RGB.test_public_Mix
4/5 Test #4: RGB.test_public_Mix ..... Passed    0.01 sec
  Start 5: RGB.test_public_Scale
5/5 Test #5: RGB.test_public_Scale ..... Passed    0.00 sec

100% tests passed, 0 tests failed out of 5

Total Test time (real) =  0.03 sec
(base) kshehata@r-86-111-25-172 build %
```

Unit Tests

Rather than running a main function to execute some desired functionality, unit testing runs a series of tests designed to exercise different elements of the project. As the name implies, each test should exercise some “unit” of the software. This is particularly helpful in Data Structures and Algorithms, as we are usually not concerned with a specific application, but instead with general purpose abstractions.

You will find some basic unit tests provided to start you out. When you submit code to Coursemology, additional “private” tests will be run. We recommend writing your own tests to exercise any corner cases you can think of for your code and to ensure correctness and/or performance. These unit tests will not be submitted nor graded in this or future assignments, but may help you ensure your submitted code is correct.

When you first get the project it should build correctly, but the unit tests will fail since you haven't completed the project yet. Run "make && make test" from the build folder to run tests, or use Visual Studio code as above. The output will look something like this:

```
(base) kshehata@r-86-111-25-172 build % make test
Running tests...
Test project /Users/kshehata/Source/CS2040C/as0/src_students/build
  Start 1: RGB.test_public_Constructor
1/5 Test #1: RGB.test_public_Constructor ..... Passed    0.01 sec
  Start 2: RGB.test_public_Constructor_LimitValues
2/5 Test #2: RGB.test_public_Constructor_LimitValues ... Passed    0.01 sec
  Start 3: RGB.test_public_Print
3/5 Test #3: RGB.test_public_Print ..... Passed    0.01 sec
  Start 4: RGB.test_public_Mix
4/5 Test #4: RGB.test_public_Mix .....***Failed    0.00 sec
  Start 5: RGB.test_public_Scale
5/5 Test #5: RGB.test_public_Scale .....***Failed    0.00 sec

60% tests passed, 2 tests failed out of 5

Total Test time (real) =  0.04 sec

The following tests FAILED:
   4 - RGB.test_public_Mix (Failed)
   5 - RGB.test_public_Scale (Failed)
Errors while running CTest
Output from these tests are in: /Users/kshehata/Source/CS2040C/as0/src_students/build/Testing/Temporary/LastTest.log
Use "--rerun-failed --output-on-failure" to re-run the failed cases verbosely.
make: *** [test] Error 8
(base) kshehata@r-86-111-25-172 build %
```

After you complete the code to pass the unit tests the output should look like this:

```
[ 8%] Building CXX object CMakeFiles/RGB.dir/RGB.cpp.o
[16%] Linking CXX static library libRGB.a
[16%] Built target RGB
[33%] Built target gtest
[50%] Built target gtest_main
[58%] Linking CXX executable RGB_test
[66%] Built target RGB_test
[83%] Built target gmock
[100%] Built target gmock_main
Running tests...
Test project /Users/kshehata/Source/CS2040C/as0/src_students/build
  Start 1: RGB.test_public_Constructor
1/5 Test #1: RGB.test_public_Constructor ..... Passed    0.01 sec
  Start 2: RGB.test_public_Constructor_LimitValues
2/5 Test #2: RGB.test_public_Constructor_LimitValues ... Passed    0.01 sec
  Start 3: RGB.test_public_Print
3/5 Test #3: RGB.test_public_Print ..... Passed    0.01 sec
  Start 4: RGB.test_public_Mix
4/5 Test #4: RGB.test_public_Mix ..... Passed    0.01 sec
  Start 5: RGB.test_public_Scale
5/5 Test #5: RGB.test_public_Scale ..... Passed    0.01 sec

100% tests passed, 0 tests failed out of 5

Total Test time (real) =  0.04 sec
(base) kshehata@r-86-111-25-172 build %
```

Submission

Submit to Coursemology the contents of RGB.cpp **without** the `#include "RGB.h"` line. Coursemology will run a series of unit tests and provide you with feedback.

Extension Tasks

If you have completed the assignment, here are some additional tasks that will further help you develop your C++ skills and prepare you for future assignments:

1. Implement an `equals(RGB)` method that returns true if the two colours are equal. Can you use this to simplify your unit tests?
2. Implement a `greyscaleValue()` method that returns an integer that is the average of the channel values, rounded to the nearest integer.
3. Implement an overloaded `+` operator that mixes two colours.
4. Implement an overloaded `*` operator between a colour and a floating point value that produces a scaled colour.
5. Implement an overloaded `==` operator that compares two colours.