Grading with Catch2

Programming Projects

Introduction

The programming projects in this class will be tested using the Catch2 (https://github.com/catchorg/Catch2) test framework. This framework only requires that a program include the header to run the test file. I will typically include the Catch2 header with the files you use to create your project. From the GitHub of the project (https://github.com/catchorg/Catch2):

> Catch2 stands for C++ Automated Test Cases in a Header and is a multi-paradigm test framework for C++. which also supports Objective-C (and maybe C).

> Catch2 is header only. All you need to do is drop the file somewhere reachable from your project - either in some central location, you can set your header search path to find, or directly into your project tree itself! This is a particularly good option for other Open-Source projects that want to use Catch for their test suite.

Example

So, let us walk through a simple project to show how Catch2 works. You can download this folder off canvas if you want to look at the source code. Notice that the Catch2 Header file is contained within the project directly. Including the header is the simplest way to use Catch2.

The project folder contains four files:

PP0Test.cpp	Written by **instructor**, used to grade the assignment and distributed on canvas
catch.hpp	Given by instructor, the Catch2 Header file
example_code.cpp	Written by the **student**
example_code.hpp	Written by the **student**

In this example, we have some simple functions to calculate the factorial. Here is the example_code.hpp:

```
/************************************************************/
// Header file for factorial
/************************************************************/

unsigned int factorial(unsigned int n);
```

And here is the example_code.cpp:

```cpp
#include "example_code.hpp"

// A tail recursive function to calculate factorial
unsigned factorialAccum(unsigned int n, unsigned int accum) {
    if (n == 0) {
        return accum;
    }

    return factorialAccum(n - 1, n * accum);
}

// A wrapper over factTR
unsigned int factorial(unsigned int n) {
    return factorialAccum(n, 1);
}
```

Above is an example of code that could be written by a student. The included test file (PP0Test.cpp in this case) is what contains the main entry point for the program.

```cpp
#define CATCH_CONFIG_MAIN
#include "catch.hpp"

#include "example_code.hpp"

TEST_CASE( "Factorials are computed") {
  REQUIRE( factorial(0) == 1 );
  REQUIRE( factorial(1) == 1 );
  REQUIRE( factorial(2) == 2 );
  REQUIRE( factorial(3) == 6 );
  REQUIRE( factorial(10) == 3628800 );
}
```

Notice that the test file includes the header file for the student code. That include is why it is important for students to name their files exactly what the instructions say to name them. Since we have our Catch2 header in the directory of our files, we can compile this code like so:

```
g++ -std=c++17 -Wall example_code.cpp PP0Test.cpp
```

This will produce an output file (since we didn't name it using the "o" flag it will just be a.out or a.exe) that will run the tests for the program. If the tests pass (all the require statements are true), you will see something like this:

```
===============================================================================
All tests passed (5 assertions in 1 test case)
```

If we do not pass all the tests (the assertion result in false), the following message will be output:

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
a.out is a Catch v2.2.3 host application.
Run with -? for options

-------------------------------------------------------------------------------
Factorials are computed
-------------------------------------------------------------------------------
PP0Test.cpp:14
...............................................................................

PP0Test.cpp:15: FAILED:
  REQUIRE( factorial(0) == 1 )
with expansion:
  7 == 1

===============================================================================
=========
test cases: 1 | 1 failed
assertions: 1 | 1 failed
```

When the test for factorial(0) runs, we are getting a 7 out, but the test code is expecting a 1. The failed tests allow you to see what is not working in your project for the test. You will need to fix your code and recompile your source code and the test source code to create a new test.