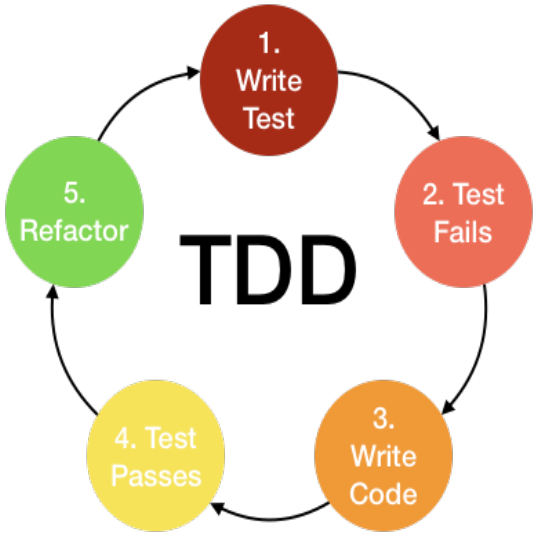
Linköpings Universitet
Institutionen för datavetenskap (IDA)
UPP-gruppen 2020-09-09

TDD and Catch - A short guide

This is a short guide on Test-Drive Development, TDD, and Catch.

What is TDD?

Test-driven development is a software development process which focuses on short development cycles that force the developer to specify requirements as tests before the actual implementation of said requirement. The cycle is started by writing tests for a requirement, seeing them fail (due to the requirement not being implemented), implementing the requirement, seeing the tests pass and finally improving the code if needed (refactoring). By writing the tests before the actual implementation we clearly define how the requirement should behave and what edge cases it might have.



The end result is that we get better tests that achieve a higher test coverage, greater quality of code and the tests can be used as documentation.

What is Catch?

Catch is a C++ framework which gives us tools to create powerful tests that are both easy to write and understand the output of.
The framework itself is very large and contains a lot of useful tools, but this guide will focus on some of the more basic ones that are sufficient to get you started with testing. If you wish to delve in deeper catch has a great wiki you can read: official catch documentation.

How do we write a test?

test_program.cc:

```
//This tells catch to create a main() for us
//and should only be done in one .cc-file.
#define CATCH_CONFIG_MAIN

//Here we include all of the tools that are part of catch.
//As the library is very large it will take some time to compile our file.
#include "catch.hpp"

//We can also include all the neccessary .h-files we need for our
//classes, functions etc.

//Note that you should not define the program we are testing in the same file
//as the tests, in this case it is only done to keep the guide short.
int sum(int a, int b)
{
    return a + b;
}

//A TEST_CASE can have several tests that are relevant for what we are testing,
//for example tests that test a specific function.
TEST_CASE( "Sum of ints") {

   //Here we define variables which will be used in the tests.
   int x{3};

   //CHECK(condition) is a simple test that checks if the condition is true.
   //The program will continue no matter the result of this test but if the
   //result is false then the test counts as a failed test.
   CHECK(sum(2, x) == 5);

   //Unlike the previous test CHECK_FALSE(condition) expects the condition to be
   //false for the test to succeed. The program will continue if this test fails.
   CHECK_FALSE(sum(2, x) == 6);

   //REQUIRE(condition) is a more strict variant of CHECK where if the test
   //result is false then the program execution will be halted
   //with a message that the test failed.
   REQUIRE(sum(2, x) == 5);

   //REQUIRE_FALSE(condition) is a variant of the previous test that expects the
   //condition to be false in order for the test to count as a success.
   //Like REQUIRE this test will halt program execution if the test fails.
   REQUIRE_FALSE(sum(2, x) == 5);
}
```

The test program is compiled just like any other C++ program which contains a 'int main()' and a executable file will be created. When the executable is ran it will either output that all tests succeeded or what tests failed and on which line in the test file these tests can be found.

Faster compilation of catch

As catch.hpp is a very large file it will take some time (usually more than 10s) to compile our tests, this might not seem as a big deal but will be very noticable once your are actually working with your code and making small changes. To avoid this we can split up our previous test file in to several files that are then compiled only when needed:

1. Create a file named test_main.cc (This file can be found among the given files)
2. Inside the given file you will find the following code:

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

3. Compile test_main.cc to a pre-compiled object file (.o-file) with the compiler flag -c:
   `g++ -std=c++17 -c test\_main.cc`
   This will compile test_main.cc with the included catch.hpp which will not need to be compiled again as we will not be making any changes to either file.
4. Remove the line #define CATCH_CONFIG_MAIN from test_program.cc (we are still using the example from the previous page).
5. Link the .o-file when compiling test_program.cc:
   `g++ -std=c++17 test_main.o test_program.cc <any-other-.cc-files>`