Asserting Using Catch

Dror Helper
AUTHOR TITLE

@dhelper blog.drorhelper.com

Overview

- Using REQUIRE
- Multiple Asserts in one test
- Checking for exceptions
- Adding more information to failures
- Converting types into strings

```
TEST_CASE("This is a test name"
{
    MyClass myClass;


    REQUIRE(myClass.MeaningOfLi
}
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
DeepThought.exe is a Catch v1.5.6 host application.
Run with -? for options

-------------------------------------------------------------------------------
This is a test name
-------------------------------------------------------------------------------
c:\projects\deepthought\computer.cpp(10)
...............................................................................

c:\projects\deepthought\computer.cpp(14): FAILED:
  REQUIRE( myClass.MeaningOfLife() == 42 )
with expansion:
  -1 == 42

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

Press any key to continue . . .
```

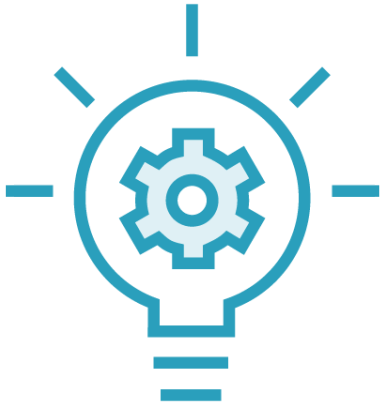
REQUIRE

Single macro for all/most assertions needs

Write the assertion in plain code

Excellent failure messages

Why Should You Care About Failure Messages?



Understand why the test failed

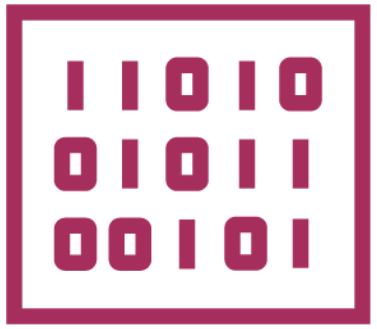
Reduce debugging time

It's the purpose of the test

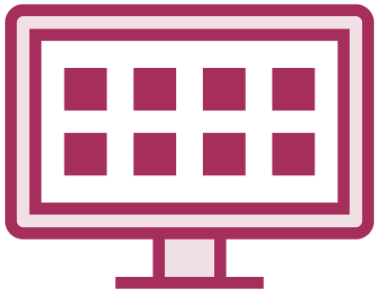
What's Wrong With This Test?

```
TEST_CASE("Encode uppercase letter --> return digit")
{
    StringToDigitsEncoder encoder;

     Digits expected({ 2 });

    REQUIRE(encoder.Encode("A") == expected);
    REQUIRE(encoder.Encode("B") == expected);
    REQUIRE(encoder.Encode("C") == expected);
}
```

The Problem with Multiple Assertions



Lose information



Testing more than one aspect

Create complicated tests

Multiple Assertions for a Single Result

```
TEST_CASE(“Tree has other word that begins with same letter”) {
    WordsTree tree;
    tree.AddWord("ab", { 2, 2 });
    tree.AddWord("ad", { 2, 3 });

    auto result = tree.GetWords(Digits{ 2, 3 });

    REQUIRE(result.size() == 1);
    REQUIRE(result[0] == "ad");
}
```

When to use Multiple Assertions?

Multiple checks for single “concept”

Checking related logic

Always be pragmatic

REQUIRE and CHECK

```
REQUIRE (2 + 2 == 5);  // Abort test --> Test fail
CHECK (2 + 2 == 5);    // Continue test --> Test fail
```

~~`REQUIRE(!MethodReturnsFalse());`~~

```
REQUIRE_FALSE(MethodReturnsFalse);
CHECK_FALSE(MethodReturnsFalse);
```

Handling Multiple Assertions in One Test

- Split to multiple tests
- Use CHECK
- Override operator ==
- Compare Collections
- Use Multiple asserts

Demo

Fixing existing tests

- REQUIRE vs. CHECK
- Splitting tests
- Overloading *operator==*
- Comparing collections

Asserting for Exceptions

```
REQUIRE_THROWS( expression )
CHECK_THROWS( expression )

REQUIRE_THROWS_AS( expression, type )
CHECK_THROWS_AS( expression type )

REQUIRE_NOTHROW( expression )
CHECK_NOTHROW( expression )
```

Demo

Testing for exceptions

Adding More Information to Test Run

INFO

WARN

FAIL

CAPTURE

Logging Macros

```
INFO(“Passed first step”);
INFO(“Customer name is: ” << customer.get_name());

CAPTURE(someValue); // someValue := 123
```

Simple Information from Complex Types

```
class SomeClass
{
public:
    int my_int_;
    double my_double_;
};
```

```
REQUIRE(result == expected)
```

```
-------------------------------------------------------------
Complex result
-------------------------------------------------------------
c:\projects\deepthought\someclasstests.cpp(5)
.............................................................

c:\projects\deepthought\someclasstests.cpp(15): FAILED:
  REQUIRE( result == expected )
with expansion:
  {?} == {?}
```

String Conversions

operator<<

Catch::toString

Catch::StringMaker specialisation

CATCH_TRANSLATE_EXCEPTION

Operator << Overloading for std::ostream

```
ostream& operator<< (ostream& os, MyType const& value )
{
    os << convert ( value );
    return os;
}
```

Catch::toString Overload

```
namespace Catch
{
    string toString(MyType const& value )
    {
        return convert ( value );
     }
}
```

Catch::StringMaker Specialisation

```
namespace Catch {
    template<> struct StringMaker<T> {
        static std::string convert( T const& value ) {
            return convert ( value );
        }
    };
}
```

Custom Exception Text

```
CATCH_TRANSLATE_EXCEPTION( MyType& ex )
{
    return ex.message();
}
```

Demo

Logging Macros

Fixing test output for complex results

Summary

REQUIRE and CHECK

Multiple asserts in one test

Why we care about failure messages

Logging test information

Customizing the way objects are shown