

Introduction

Methodology

Catch

Gcov: code
coverage

Lab 8: Unit Testing

Comp Sci 1585
Data Structures Lab:
Tools for Computer Scientists



Introduction

Methodology

Catch

Gcov: code coverage

- 1 Introduction
- 2 Methodology
- 3 Catch
- 4 Gcov: code coverage

Introduction

Methodology

Catch

Gcov: code coverage

Unit testing lets you test your code piece-by-piece, instead of all at once at the end of development. It also automates the testing process so you know when you introduce bugs into your code. Some go so far as to write tests before writing code. This is called test-driven development (TDD). Some frameworks include

- Catch
- Boost Unit Test Framework (UTF)
- Google test
- Gcov: Code coverage tool

Introduction

Methodology

Catch

Gcov: code coverage

- 1 Introduction
- 2 Methodology**
- 3 Catch
- 4 Gcov: code coverage

Introduction

Methodology

Catch

Gcov: code coverage

- Each test should be testing one (and only one) small *unit* of your software
- A single test is typically broken up into:
 - Bootstrapping your test
 - Expected behavior
 - Actual behavior (running the code *unit* that is being tested)
 - Comparing expected to actual behavior
 - If actual behavior does not match expected behavior, print out an informative message describing the problem

Introduction

Methodology

Catch

Gcov: code coverage

```

// Function to be tested
int add(int x, int y)
{
    return x + y;
}
    
```

Introduction

Methodology

Catch

Gcov: code coverage

```

#include "add.hpp"
#include "catch.hpp"

TEST_CASE("Integer Arithmetic", "[int]")
{
    // Bootstrapping your test setup
    int value_to_add = 1;
    int initial_value = 4;

    /* ... */
}
  
```

Introduction

Methodology

Catch

Gcov: code
coverage

```
TEST_CASE("Integer Arithmetic", "[int]") {  
    /* Bootstrapping up here */  
  
    SECTION("Adding 1 to a number increases it by 1")  
    {  
        // Defining expected behavior  
        int expected_value = 5;  
  
    }  
}
```


Introduction

Methodology

Catch

Gcov: code
coverage

```
TEST_CASE("Integer Arithmetic", "[int]") {
    /* Bootstrapping up here */

    SECTION("Adding 1 to a number increases it by 1")
    {
        // Defining expected behavior
        int expected_value = 5;

        // Obtain actual behavior
        int actual_value = add(initial_value, // = 4
                               value_to_add); // = 1

        // Compare expected against actual
        CHECK(expected_value == actual_value);
    }
}
```

Test Driven Development (TDD) emphasizes writing tests first that guide the development of your code.

- Don't write your code first!
- Instead, start by writing tests
- Progression of development: Fail, Pass, Repeat
 - Scaffold your code:
 - Define interface (i.e. header files),
 - Return dummy values that do nothing and cause your test to fail
 - Implement a single test: bootstrap, expected behavior, actual behavior, comparison
 - Run your tests (it should fail)
 - Go back to code and fix it
 - Run your tests (it should pass)
 - Move on to next test

Introduction

Methodology

Catch

Gcov: code coverage

- 1 Introduction
- 2 Methodology
- 3 Catch**
- 4 Gcov: code coverage

<https://github.com/catchorg/Catch2>

- ① Make an simple `.cpp` (like `test_main.cpp`) with just:


```
#define CATCH_CONFIG_MAIN
#include "catch.hpp"
```
- ② Write some tests in their own `tests_whatever.cpp` and include your code files to test in this file.
- ③ Compile the `tests_whatever.cpp` like normal (or faster with a makefile)

Check out some examples

Introduction

Methodology

Catch

Gcov: code coverage

- ① Introduction
- ② Methodology
- ③ Catch
- ④ Gcov: code coverage

Introduction

Methodology

Catch

Gcov: code coverage

Are all lines of your code being executed by your tests?

Are there code blocks that are not being reached (and thus are not being *covered* by your test)?

Goal: obtain 100% coverage with your tests

Introduction

Methodology

Catch

Gcov: code coverage

- ① Compile your tests with the `--coverage` flag
- ② Run your test suite executable
- ③ Run `$ gcov -mr [.cpp files]` with all `.cpp` files in your project to compute code coverage

Check out the example