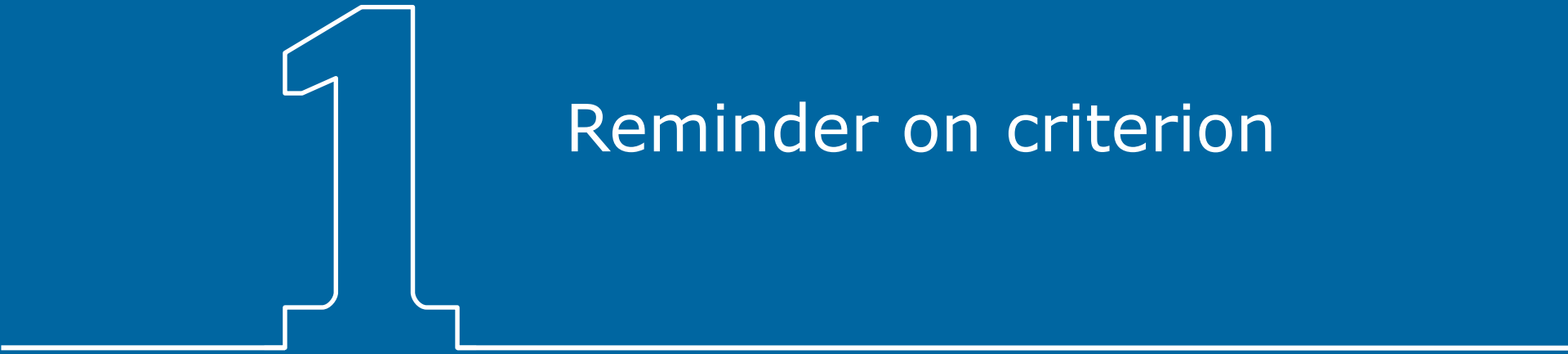
Google test framework

Sébastien Valat

20.02.2020



Reminder on criterion

Criterion basic usage : cr_assert()

```c
#include <criterion/criterion.h>

Test(simple, test_ok) {
    int a = 3;
    cr_assert(a == 3);
}

Test(simple, test_not_ok) {
    int a = 3;
    cr_assert(a == 4);
}
```

```
gcc -lcriterion main.c -o test
./test
```

```
[----] main.c:10: Assertion failed: The expression a == 4 is false.
[FAIL] simple::test_fail: (0,00s)
[====] Synthesis: Tested: 2 | Passing: 1 | Failing: 1 | Crashing: 0
```

Ways to link & use

- There is one libs:
  - libcriterion.so

```
Include_directories(${CRITERION_INCLUDE_DIRS})
add_executable(test main.c)
target_link_libraries(${CRITERION_LIBRARIES})

add_test(test test --xml=${REPORTS_DIR}/file.xml)
```

Criterion: cr_assert_eq()

```c
#include <criterion/criterion.h>

Test(simple, test_ok) {
   int a = 3;
   cr_assert_eq(a, 3);
}

Test(simple, test_not_ok) {
   int a = 3;
   cr_assert_eq(a, 4);
}
```

```
gcc -lcriterion main.c -o test
./test
```

```
[----] main.c:10: Assertion failed: The expression (a) == (4) is false.
[FAIL] simple::test_fail: (0,00s)
[====] Synthesis: Tested: 2 | Passing: 1 | Failing: 1 | Crashing: 0
```

Criterion usage: message

```c
#include <criterion/criterion.h>

Test(simple, test_ok) {
    int a = 3;
    cr_assert_eq(a, 3);
}

Test(simple, test_not_ok) {
    int a = 3;
    cr_assert_eq(a, 4,
                 "Message on error %d", a);
}
```

```
gcc -lcriterion main.c -o test
./test
```

```
[----] main.c:10: Assertion failed: Message on error 3
[FAIL] simple::test_fail: (0,00s)
[====] Synthesis: Tested: 2 | Passing: 1 | Failing: 1 | Crashing:
```

Criterion expect_eq()

```c
#include <criterion/criterion.h>

Test(simple, test_ok) {
   int a = 3;
   cr_assert(a == 3);
}

Test(simple, test_not_ok) {
   int a = 3;
   int b = 4;
   cr_expect_eq(a, 4);
   cr_expect_eq(b, 5);
}
```

```
gcc -lcriterion main.c -o test
./test
```

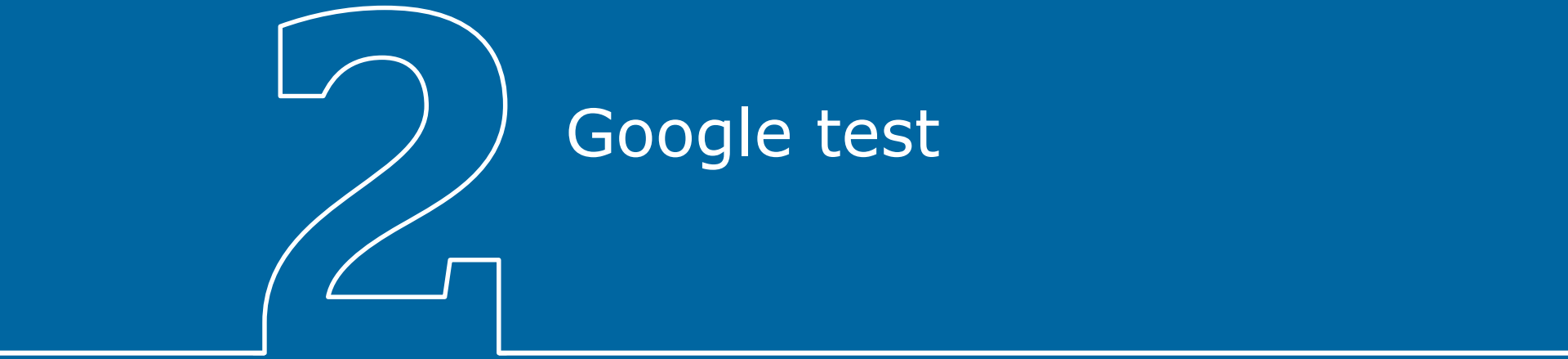
```
[----] main.c:11: Assertion failed: The expression (a) == (4) is false.
[----] main.c:12: Assertion failed: The expression (b) == (5) is false.
[FAIL] simple::test_fail: (0,00s)
[====] Synthesis: Tested: 2 | Passing: 1 | Failing: 1 | Crashing: 0
```

Debugging

- Criterion fork for every test

- Pros:
  - All test run even if one segfault
  - No test interaction via global variables

- Cons:
  - Cannot easily GDB
  - Has to run GDB as server then connect on it



Google test

Google test basic usage : ASSERT()

```
#include <gtest/gtest.h>

TEST(simple, test_ok) {
   int a = 3;
   ASSERT_TRUE(a == 3);
}

TEST(simple, test_not_ok) {
   int a = 3;
   ASSERT_TRUE(a == 4);
}
```

```
g++ -lgtest -lgtest_main main.c -o test
./test
```

```
Running main() from gtest_main.cc
[==========] Running 2 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 2 tests from simple
[ RUN      ] simple.test_ok
[       OK ] simple.test_ok (0 ms)
[ RUN      ] simple.test_not_ok
main.cpp:10: Failure
Value of: a == 4
  Actual: false
Expected: true
[  FAILED  ] simple.test_not_ok (0 ms)
[----------] 2 tests from simple (0 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test case ran. (0 ms total)
[  PASSED  ] 1 test.
[  FAILED  ] 1 test, listed below:
[  FAILED  ] simple.test_not_ok
```

Ways to link & use

- There is two libs:
  - libgtest.so
  - libgtest_main.so

- Both are pointed by **GTEST_BOTH_LIBRARIES**

```
Include_directories(${GTEST_INCLUDE_DIRS})
add_executable(test main.c)
target_link_libraries(${GTEST_BOTH_LIBRARIES})

add_test(test test --gtest_output=xml:${REPORTS_DIR}/run-modparams-criterion.xml)
```

Google test basic usage : ASSERT_EQ()

```
#include <gtest/gtest.h>

TEST(simple, test_ok) {
   int a = 3;
   ASSERT_EQ(3, a);
}

TEST(simple, test_not_ok) {
   int a = 3;
   ASSERT_EQ(4, a);
}
```

```
g++ -lgtest -lgtest_main main.c -o test
./test
```

```
Running main() from gtest_main.cc
[==========] Running 2 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 2 tests from simple
[ RUN      ] simple.test_ok
[       OK ] simple.test_ok (0 ms)
[ RUN      ] simple.test_not_ok
main.cpp:10: Failure
Value of: a
  Actual: 3
Expected: 4
[  FAILED  ] simple.test_not_ok (0 ms)
[----------] 2 tests from simple (0 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test case ran. (0 ms total)
[  PASSED  ] 1 test.
[  FAILED  ] 1 test, listed below:
[  FAILED  ] simple.test_not_ok
```

Google test : extra messages

```
#include <gtest/gtest.h>

TEST(simple, test_ok) {
   int a = 3;
   ASSERT_EQ(3, a);
}

TEST(simple, test_not_ok) {
   int a = 3;
   ASSERT_EQ(4, a) << "Additional infos";
}
```

```
g++ -lgtest -lgtest_main main.c -o test
./test
```

```
Running main() from gtest_main.cc
[==========] Running 2 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 2 tests from simple
[ RUN      ] simple.test_ok
[       OK ] simple.test_ok (0 ms)
[ RUN      ] simple.test_not_ok
main.cpp:10: Failure
Value of: a
  Actual: 3
Expected: 4
Additional infos
[  FAILED  ] simple.test_not_ok (0 ms)
[----------] 2 tests from simple (0 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test case ran. (0 ms total)
[  PASSED  ] 1 test.
[  FAILED  ] 1 test, listed below:
[  FAILED  ] simple.test_not_ok
```

Google test : EXPECT

```
#include <gtest/gtest.h>

TEST(simple, test_ok) {
   int a = 3;
   ASSERT_EQ(3, a);
}

TEST(simple, test_not_ok) {
   int a = 3;
   int b = 5;
   EXPECT_EQ(4, a);
   EXPECT_EQ(6, b);
}
```

```
g++ -lgtest -lgtest_main main.c -o test
./test
```

```
Running main() from gtest_main.cc
[==========] Running 2 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 2 tests from simple
[ RUN      ] simple.test_ok
[       OK ] simple.test_ok (0 ms)
[ RUN      ] simple.test_not_ok
main.cpp:11: Failure
Value of: a
  Actual: 3
Expected: 4
main.cpp:12: Failure
Value of: b
  Actual: 5
Expected: 6
[  FAILED  ] simple.test_not_ok (0 ms)
[----------] 2 tests from simple (0 ms total)

[----------] Global test environment tear-down
[----------] 2 tests from 1 test case ran. (0 ms total)
```

Google death tests

```
#include <gtest/gtest.h>
using namespace testing;

void func(void) {
   fprintf(stderr, "Additional infos"):
   exit(3);
}

TEST(simple, test) {
   ASSERT_DEATH(func(), "Additional infos");
}

TEST(simple, test2) {
   ASSERT_EXIT(func(), ExistedWithCode(3),
               "Additional infos");
}
```

```
g++ -lgtest -lgtest_main main.c -o test
./test
```

Google fixtures

```
#include <gtest/gtest.h>
#include <list>
using namespace testing;

class SimpleTest : public Test {
    protected:
        void SetUp() override {
            lst.push_back(1);
            lst.push_back(2);
        }

        void TearDown() override {
        }

        std::list<int> lst;
};
```

```
TEST_F(SimpleTest, test) {
   ASSERT_EQ(2, lst.size());
}
```

```
g++ -lgtest -lgtest_main main.c -o test
./test
```

Debugging

- No forking
- Pro:
  - Ease test implementation fix
- Cons:
  - Global variable issue
  - Abort all test suite on segfault (failure still captured by ctest)

Mocking

Mocking

```
#include <gtest/gtest.h>
#include <gmock/gmock.h>
#include <turtle.hpp>
using namespace testing;

class MockTurtle : public Turtle {
    public:
        MOCK_METHOD1(forward, void(int distance));
};

TEST(Turtle, test) {
    MockTurtle turtle;
    EXPECT_CALL(turtle, forward(_))
                .Times(3);
    turtle.forward(4);
}
```

```
g++ -lgtest -lgmock -lgmock_main main.c -o test
./test
```

```
TEST_F(SimpleTest, test) {
   ASSERT_EQ(2, lst.size());
}
```

```
[----------] 1 test from suite
[ RUN      ] suite.test
main.cpp:20: Failure
Actual function call count doesn't match
EXPECT_CALL(turtle, forward(_))...
         Expected: to be called 3 times
           Actual: called once - unsatisfied and
active
[  FAILED  ] suite.test (0 ms)
[----------] 1 test from suite (0 ms total)
```

How to mock C function

```
class Mock {
    public:
        MOCK_METHOD1(forward, void(int distance));
};

Mock * gblMock;

void function_to_mock(int distance)
{
    gblMock->forward(distance);
}

TEST(Turtle, test) {
    Mock mock;
    gblMock = &mock;
    EXPECT_CALL(mock, forward(_))
                .Times(3);
    function_to_mock(4);
}
```

Ways to link

- There is two libs:
  - gmock.so
  - gmock_main.so
- Both are pointed by **GMOCK_BOTH_LIBRARIES**
- It also depend on google test

```
Include_directories(${GTEST_INCLUDE_DIRS} ${GMOCK_INCLUDE_DIRS})
add_executable(test main.c)
target_link_libraries(${GTEST_LIBRARIES} ${GMOCK_BOTH_LIBRARIES})

add_test(test test --gtest_output=xml:${REPORTS_DIR}/run-modparams-criterion.xml)
```

Thanks

20.02.2020