Epitech Documentation

Unit tests for C/C++

How to write them

2.6

{EPITECH.}

Prerequisite

Unit tests rely on Criterion library.
To install Criterion, you can use the script *install_criterion.sh* (which can be found next to this document) or follow the instructions from the Criterion's website:
`http://criterion.readthedocs.io/en/master/setup.html#installation`

Project tree

The only requirement for unit tests in your project tree is that the files containing the unit tests must be placed in a folder named `tests` at the root of the project.

The subtree of this `tests` directory is up to you.

Compiling and running using Criterion

To manually build your unit test manually you can do it using a command line such as:

```
Terminal
~/Epitech Documentation> gcc -o unit_tests project.c tests/test_project.c
--coverage -lcriterion
```

Criterion add it's own main, so none of the files present on the build line should contain a `main` function.

Then execute the produced binary to see your test results.

```
Terminal
~/Epitech Documentation> ./unit_tests
[====] Synthesis: Tested: 4 | Passing: 4 | Failing: 0 | Crashing: 0
```

+ Evaluation at Epitech

Unless specified otherwise in the project subject, your tests will be built and launched using your own Makefile.
The `Makefile` used to compile the unit tests is **the Makefile at the root of your project directory**.

This Makefile **must** contains a rule named **tests_run** which *must* do two things:
- Build your unit-tests binary,
- Execute your unit-tests binary.

If everything is good you should see the number of test that passed and failed when executing **make tests_run**.

Display code coverage of your tests

The execution of your unit-tests binary should create a bunch of `.gcda` and `.gcno` files. These can be used by tools to calculate the amount of code executed by your test.
This amount can be either in number of lines or number of branches (to check if every case of a condition has been properly tested).

To be able to compute a meaningful coverage you must always build your test with your code base (aka: all your C files).

The tool we use to calculate your coverage is called gcovr. We invite you to install it.
You can use it like so (the test files are excluded from the coverage calculation):

Terminal

```
~/Epitech Documentation> gcovr --exclude tests/
------------------------------------------------------------------------------
                           GCC Code Coverage Report
Directory: .
------------------------------------------------------------------------------
File                                       Lines    Exec  Cover   Missing
------------------------------------------------------------------------------
test_project.c                                 7       7   100%
------------------------------------------------------------------------------
TOTAL                                          7       7   100%
------------------------------------------------------------------------------

~/Epitech Documentation> gcovr --exclude tests/ --branches
------------------------------------------------------------------------------
                           GCC Code Coverage Report
Directory: .
------------------------------------------------------------------------------
File                                    Branches   Taken  Cover   Missing
------------------------------------------------------------------------------
test_project.c                                 6       6   100%
------------------------------------------------------------------------------
TOTAL                                          6       6   100%
------------------------------------------------------------------------------
```

Tests files

The tests files only contain the tests, following this format:

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

Test(suite_name, test_name)
{
    ...
}
```

with *suite_name + test_name* unique.

The list of asserts is here.
The most used are:

```
// Passes if Expression is true
cr_assert(Expression, FormatString, ...);
// Passes if Expression is false
cr_assert_not(Expression, FormatString, ...);
// Passes if Actual == Expected
cr_assert_eq(Actual, Expected, FormatString, ...);
// Passes if Actual != Expected
cr_assert_neq(Actual, Expected, FormatString, ...);
```

Examples

Criterion maintainer has written many example files.
Basic usage of Criterion can be found here and here.
Here is an example of a unit test file:

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

const char *str = "Hello world";
const int len = 11;

Test(utils, is_str_length_equal_to_len_v1)
{
    cr_assert(strlen(str) == len);
}

Test(utils, is_str_length_equal_to_len_v2)
{
    cr_assert_eq(strlen(str), len);
}

Test(utils, is_str_length_equal_to_len_v3)
{
    cr_assert_not(strlen(str) != len);
}
```

The 3 tests are doing the same thing with different syntaxes.
They check that the `"Hello world"` string has a length of 11 characters.

However the following test aborts:

```
Test(utils, is_str_length_different_to_len)
{
    cr_assert_neq(strlen(str), len);
}
```