



H - Workshop - HUB

HUB

Workshop Criterion

Test your program !





Workshop Criterion

binary name: tests_run
language: Cpp
compilation: Makefile
build tool: no need here



- Your repository must contain the totality of your source files, but no useless files (binary, temp files, obj files,...).
- All the bonus files (including a potential specific Makefile) should be in a directory named *bonus*.
- Error messages have to be written on the error output, and the program should then exit with the 84 error code (0 if there is no error).

Welcome to this Workshop on Criterion ! What is Criterion ? It is a library which allows you to create unit tests, unit tests allow them to test your program according to the expected results ! The Crit rion library is associated with the Coverage library which allows you to have the coverage percentage of your unit tests.

The purpose of this Workshop is to familiarize you with these two libraries! You will therefore work on several exercises which will allow you to learn more about the Criterion and Coverage unit tests.

The Workshop consists of 2 exercises:

- Function return in [int] [string]
- print on standard and error output



The exercises are examples of tests for you to modify and adapt them to your programs.



COMPILATION

As you will see in the exercises, the unit tests are compiled using the Makefile which contains the rules (see the Makefile workshop) `tests_run`. This rule is presented in this form: (g++) for the compilation with the flags (- coverage -lcrriterion -lgcov) and the launch of the binary test. Do not forget to add the libraries in your `my.h`.



Attention always remember to launch your unit tests in your “units_tests” rules and not to put your `main.c` in the compilation of your unit tests.

Once your unit tests launched here is the terminal output.

```
Terminal
~/HUB> ./units_tests
[===] Synthesis: Tested: 7 | Passing: 7 | Failing: 0 | Crashing: 0
gcovr
-----
GCC Code Coverage Report
Directory: .
-----
File                                Lines    Exec    Cover    Missing
-----
../src/error_handling.cpp            7         7    100%
../src/fill_map.cpp                  13        13    100%
../src/get_map.cpp                   20        20    100%
-----
TOTAL                                40        40    100%
-----
```



Now that you understand how the Criterion tests were launched with a Makefile, you could code your tests! The compilation that we have put in place now is compatible with Marvin's Automated Tests!



EXERCICE 1 (TEST THE RETURN OF THE FUNCTIONS)



Go to the ex00 folder of the github in the cpp part

The goal of this exercise is to check if the return of some functions is equal or not to 1 or 0... No worries everything is already ready you just have to modify the file "tests_basic.c" to check the return of the function whose parameters sent to it are character strings. To do this, you will use the function "cr...(Function_name (parameter, expected return));"



To successfully complete this exercise, I suggest some criterion functions.

please note that the following asserts only work for non-array comparison :

passes if and only if Actual is equal (or not equal, if you are using neq) to reference.

cr_assert_eq(Actual, reference)

cr_assert_neq(Actual, reference)

Will pass if Actual is less than (or less than or equal if you used leq) reference.

cr_assert_lt(Actual, reference)

cr_assert_leq(Actual, reference)

Will pass if Actual is greater than (or greater than or equal if you used geq) reference.

cr_assert_gt(Actual, reference)

cr_assert_geq(Actual, reference)

epitech-2022-technical-documentation

If you have a problem, don't hesitate to ask for help! Good luck to everyone!



EXERCICE 2 (TEST THE STDOUT OR STDERR OF THE FUNCTIONS)



Go to the ex01 folder of the github in the cpp part

In this exercise, we complicate things! We no longer want to check the return values, but here the print which is done by the function on the stdout (standard output) or stderr (error output). Nothing changes, you just have to modify the “tests_print.cpp” file to check the print of the “Workshop.cpp” function.



Warning to watch the standard or the error output I called the redirect criterion function!



To successfully complete this exercise, I suggest some criterion functions.

To use the following assertions, you must include “criterion/redirect.h” along with “criterion/criterion.h”. redirect.h allows Criterion to get the content of stdout and stderr and run asserts on it. You also need to create a function that calls the cr_redirect_stdout() or ct_redirect_stderr() functions :

Compares the content of stdout with Value. This assertion behaves similarly to cr_assert_str_eq()

cr_assert_stdout_eq_str(Value)

cr_assert_stderr_eq_str(Value)

cr_assert_stdout_neq_str(Value)

cr_assert_stderr_neq_str(Value)

epitech-2022-technical-documentation



Before testing your function with “cr _...” For redirects you must make a call to the function you wanted to test!

If you have a problem, don't hesitate to ask for help! Good luck to everyone!



EXERICE 3 (IT'S YOUR TURN)



Go to the ex02 folder of the github in the cpp part

It's up to you! You just have the program, it's up to you to find out how to do the unit tests (100%) of this project! Use all the functions seen just before.



Warning the EPITECH coding style must be follow in your test !

The unit tests for this project have been tested using the epitech docker and can be validated at 100% coverage. You must therefore check the function returns and the standard output. On this project no error handling was done. But it is possible to watch the signal send back by a function (SIGSEGV), for that here is a small link which can help you!

If you have a problem, don't hesitate to ask for help! Good luck to everyone!



CONCLUSION

Go boring moment the conclusion!

As you understood at Epitech unit testing is very important, and there are a ton of tests to run! For reasons of time, we have not shown the tests for the structures. I will nevertheless let you if necessary go check the link of the Epitech doc on the tests which is not bad.

You could see during this workshop, the compilation with flags, the compilation rules and what not to do! You have also seen how to create a unit test file. If you respect on all your projects this way of testing the TA will pass over it without problem every time!

Beyond the dimension of Epitech tests. The tests whether unit or functional are very important also in the business world, because it shows that your program has the behavior requested by your employer!



Thank you for participating in this Workshop!