H - Workshop - HUB

HUB

Workshop Criterion

Test your program !

1.0

Workshop Criterion

binary name:	tests_run
language:	C
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 Critérion 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 5 exercises:

- Function return in [int]
- Function return in [char]
- Function return in [char * or int *]
- print on standard output
- print on 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: (gcc) for the compilation with the flags (– coverage -lcriterion -lgcov) and the launch of the binary test. Do not forget to add the libraries in your my.h.

Terminal
```
~/HUB> cat Makefile
tests_run: $(SRCS) $(SRCSTEST) clean
gcc -coverage $(SRCS) -lcriterion -lgcov $(SRCSTEST) -o (NAMETEST).
./(NAMETEST)
gcovr
```

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.c                    7         7       100%
../src/fill_map.c                         13        13       100%
../src/get_map.c                          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!

EXERICE 1 (TEST RETURN INT)

Go to the exercise 1 folder of the github in the C part

The goal of this exercise is to check if the return of the "error_handling.c" function is equal to 1 or 0 or 84. No worries everything is already ready you just have to modify the file "tests_return_one.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!

Exerice 2 (Test return char)

Go to the exercise 2 folder of the github in the C part

The purpose of this exercise is to look at the return value of the "error_handling" function, if it is equal to the char e, p or o. Do not worry ! As in the previous exercise, you just have to modify the "tests_return_two.c" file to look at the return value of the function. Attention here is send in parameter a number / number. 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*

Be careful what you use to declare the value to compare !

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

Exerice 3 (Test return char * or int *)

Go to the exercise 3 folder of the github in the C part

Here always the same instruction. Check the return value of the "error_handling.c" function, the return value is a char * in this exercise, you will have either "error, passed or other". Don't worry you just have to modify the "tests_return_three.c" file to check the return value of the function. You will always use the criterion library only!"

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

> *Those functions won't allow you to compare the output of your progam with a given reference string :*
>
> *Just like cr_assert_eq(), but will check two strings, character by character.*
> *cr_assert_str_eq(Actual, Reference)*
> *cr_assert_str_neq(Actual, Reference)*
>
> *Will pass if the string is empty (or is not empty is you used not_empty).*
> *cr_assert_empty(Value)*
> *cr_assert_not_empty(Value)*
>
> *epitech-2022-technical-documentation*

Be careful what you use to declare the value to compare !

Whether a char * or an int *, these are the same functions and method to use!

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

Exerice 4 (Test print stdout)

Go to the exercise 4 folder of the github in the C 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). Nothing changes, you just have to modify the "tests_print_stdout.c" file to check the print of the "error_handling.c" function.

Warning to watch the standard 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() function :*
>
> *Compares the content of stdout with Value. This assertion behaves similarly to cr_assert_str_eq()*
> *cr_assert_stdout_eq_str(Value)*
> *cr_assert_stdout_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 5 (TEST PRINT SDRERR)

Go to the exercise 5 folder of the github in the C part

In this last exercise (YES!), You must check the print of the attention function here, it is a print on stderr (error output). Nothing changes, you just have to complete the file "tests_print_stderr.c" to check the print of the "error_handling.c" function

Warning to watch 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() function :*
>
> *Compares the content of stdout with Value. This assertion behaves similarly to cr_assert_str_eq()*
> *cr_assert_stderr_eq_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 6 (It's your turn)

Go to the exercise 6 folder of the github in the C part

It's up to you! Here is a program that makes an "Identity Card" according to the parameters entered in the function! This time, if the information is stored in a structure! 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 !

Terminal

```
~/HUB> ./exercice_6 40 Jerome Renaud | cat -e
Id card of EPITECH$
********************$
First name is : Jerome$
Last name is : Renaud$
Age is : 40$
```

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!