# SCALE FOR PROJECT NM (/PROJECTS/NM)

You should evaluate 1 student in this team

★

Git repository

```
git@vogsphere.42porto.com:vogsphere/intra-uuid-3216cbec-b22e-499
```

## Introduction

Please follow the following rules :

- Remain polite, courteous, respectful and constructive throughout the evaluation process. The well-being of the community depends on it.

- Identify with the person (or the group) evaluated the eventual dysfunctions of the work. Take the time to discuss and debate the problems you have identified.

- You must consider that there might be some difference in how your peers might have understood the project's instructions and the scope of its functionalities. Always keep an open mind and grade him/her as honestly as possible. The pedagogy is valid only and only if peer-evaluation is conducted seriously.

## Guidelines

- Only grade the work that is in the student or group's GiT repository.

- Double-check that the GiT repository belongs to the student or the group. Ensure that the work is for the relevant project and also check that "git clone" is used in an empty folder.

- Check carefully that no malicious aliases was used to fool you and make you evaluate something other than the content of the official repository.

- To avoid any surprises, carefully check that both the evaluating and the evaluated students have reviewed the possible scripts used to facilitate the grading.

- If the evaluating student has not completed that particular project yet, it is mandatory for this student to read the entire subject prior to starting the defence.

- Use the flags available on this scale to signal an empty repository, non-functioning program, cheating etc. In these cases, the grading is over and the final grade is 0 (or -42 in case of cheating). However, with the exception of cheating, you are encouraged to continue to discuss your work (even if you have not finished it) in order to identify any issues that may have caused this failure and avoid repeating the same mistake in the future.

- Remember that for the duration of the defence, no segfault, no other unexpected, premature, uncontrolled or unexpected termination of the program, else the final grade is 0. Use the appropriate flag.
You should never have to edit any file except the configuration file if it exists.
If you want to edit a file, take the time to explicit the reasons with the evaluated student and make sure both of you are okay with this.

- You must also verify the absence of memory leaks. Any memory allocated on the heap must be properly freed before the end of execution.
You are allowed to use any of the different tools available on the computer, such as leaks, valgrind, or e_fence. In case of memory leaks, tick the appropriate flag.

# Attachments

- header_and_prog copy (https://cdn.intra.42.fr/document/document/29527/header_and_prog_copy)
- header_offset_error (https://cdn.intra.42.fr/document/document/29528/header_offset_error)
- header copy (https://cdn.intra.42.fr/document/document/29529/header_copy)
- easy_test.c (https://cdn.intra.42.fr/document/document/29530/easy_test.c)
- not_so_easy_test.c (https://cdn.intra.42.fr/document/document/29531/not_so_easy_test.c)
- header_and_prog (https://cdn.intra.42.fr/document/document/29532/header_and_prog)
- wrong_arch (https://cdn.intra.42.fr/document/document/29533/wrong_arch)
- unterminated_string (https://cdn.intra.42.fr/document/document/29534/unterminated_string)
- header (https://cdn.intra.42.fr/document/document/29535/header)
- error_header (https://cdn.intra.42.fr/document/document/29536/error_header)
- subject.pdf (https://cdn.intra.42.fr/pdf/pdf/146923/en.subject.pdf)

# Preliminaries

*You will use the following code to test and you will compiled the second one in 32bit format <code> $> cc easy_test.c -o test_facile $> cc not_so_easy_test.c -o not_so_easy_test $> cc -m32 not_so_easy_test.c -o not_so_easy_test_32-bit $> </code> In case you dont know, to identify the type and architecture of a file:*

*man 1 file If you ve got issue to find a universal binary, a search in the PATH: <code> (IFS=$'\n'; for d in ${PATH//:/$IFS}; do find "$d" -type f -exec file '{}' \+ | grep -i -A3 universal ; done) </code> To create a universal binary: <code> clang/gcc: -m32 pour cross-compile a 32-bit lipo -create -output <universel> <binaire arch. 1> <binaire arch. 2> ... </code> If you cant find a dynamic library (.so, .dylib): <code> find /usr/lib -type f -iname '*\.dylib' 2>/dev/null </code>*

**Preliminary tests**

Please check the following first:

- Are the files in the right place in the git repo
- The Makefile work as intended and compile the binary
- No cheat (no forbidden functions, the student can explain his code...)

If something doesnt follow the subject, the mark for the evaluation will stop there.
But you can keep evaluating and you are strongly suggested to talk about the project a bit further

☑ Yes ☒ No

# tests

**nm error tests**

Test ft_nm with error files. A few can be found in the header of the evaluation,
but you can forge your own.
If the program quits in an unexpected manner, the evaluation stops here.

☑ Yes ☒ No

**Nm easy test**

Test ft_nm with the binary easy_test. The output should be
in accordance with the true nm.

☑ Yes ☒ No

**Test less easy**

Test ft_nm on the not_so_easy binaries. On the 32 and 64 bits binaries
the ft_nm output should be the same as nm
Make sure that the symbols list output is exactly the same compared to the system nm.
The rest can differ a bit.

☑ Yes ☒ No

**Other Test**

ft_nm output is always equal to the real nm output, with any test.

Make sure that the symbols list output is exactly the same compared to the system nm.
The rest can differ a bit.

☑ Yes ☒ No

**Multiples arguments**

ft_nm can take multiple arguments

Yes No

**Object files**

Test ft_nm with 32 and 64 bits object files (.o).
Order can differ with the real nm.

☑ Yes ☒ No

**Dynamic library**

Test ft_nm with a dynamic library (*.dylib *.so)
(Look in /usr/lib/). The output should look like the real nm output but
the symbol order can be arbitrary

☑ Yes ☒ No

**Universal binary**

Test ft_nm on a universal binary (example:/usr/bin/python). Dont forget to use the command "file <binary>".
The output should be the same as the real nm. But the order of the symbols can differ.

☑ Yes ☒ No

# Bonus

**Options**

Count one point per bonus from the following options

- a
- g
- u
- r
- p

**Rate it from 0 (failed) through 5 (excellent)**

# Ratings

**Don’t forget to check the flag corresponding to the defense**

✔ Ok ★ Outstanding project

Empty work No author file Invalid compilation Norme Cheat Crash

Forbidden function

# Conclusion

**Leave a comment on this evaluation**

**Finish evaluation**