Go to homepage

SCALE FOR PROJECT GET_NEXT_LINE

You should evaluate 1 student in this team

Introduction

Please comply with 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 student or group whose work is evaluated the possible dysfunctions in their project. Take the time to discuss and debate the problems that may have been identified.

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

Guidelines

- Only grade the work that was turned in the Git repository of the evaluated student or group.

- Double-check that the Git repository belongs to the student(s). Ensure that the project is the one expected. 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 that is not the content of the official repository.

- To avoid any surprises and if applicable, review together any scripts used to facilitate the grading (scripts for testing or automation).

- If you have not completed the assignment you are going to evaluate, you have to read the entire subject prior to starting the evaluation process.

- Use the available flags to report an empty repository, a non-functioning program, a Norm error, cheating, and so forth.
In these cases, the evaluation process ends and the final grade is 0, or -42 in case of cheating. However, except for cheating, student are strongly encouraged to review together the work that was turned in, in order to identify any mistakes that shouldn't be repeated 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

subject.pdf

Mandatory Part

Norminette

Run the Norminette. If there is an error, the evaluation stops here.
You can keep going and discuss the implementation of the code, but the assignment will not be graded.

Yes | No

Compilation

It should be possible to compile the project with the usual flags and -D BUFFER_SIZE.
It must be possible to compile this project with and without this flag in addition to the flags: -Wall -Wextra -Werror. If there is an error, the evaluation stops here.
You can keep going and discuss the implementation of the code, but the assignment will not be graded.

Yes | No

Error management

Carry out AT LEAST the following tests to try to stress the error management.

- Pass an arbitrary file descriptor to the get_next_line function on which it is not possible to read, for example 42. The function must return NULL.
- Check the error returns for read and malloc. If there is an error, the evaluation stops here. You can keep going and discuss the implementation of the code, but the assignment will not be graded.

Yes | No

Testing

As the evaluator, you are expected to provide a main which will always check:

- The return value of the get_next_line is NULL in case of error.
- Otherwise, the function returns the line read, always with a `\n` at the end except if the end of

file was reached and does not end with a `\n` character.

Test all the possible combinations of the following rules:

- Large BUFFER_SIZE (>1024)
- Small BUFFER_SIZE (< 8, and 1)
- BUFFER_SIZE exactly the length of the line to read
- 1 byte variant (+/-) between the line and the BUFFER_SIZE
- Read on stdin
- Read from a file
- (Multiple/Single) Long line (2k+ characters)
- (Multiple/Single) Short line (< 4 characters, even 1)
- (Multiple/Single) Empty line

These tests should enable you to verify the strength of the student's get_next_line.
If there is an error, the evaluation stops here.

Yes | No

Bonus part

Evaluate the bonus part if, and only if, the mandatory part has been entirely and perfectly done, and the error management handles unexpected or bad usage. In case all the mandatory points were not passed during the defense, bonus points must be totally ignored.

Multiple fd reading

Perform the same tests as you did before, this time launch multiple instances of get_next_line with a different file descriptor on each. Make sure that each get_next_line is returning the correct line. Combine with a non-existing fd to check for errors.

Yes | No

Single static variable

Check the code and verify if there is indeed only one static variable.
Give the points if that's the case.

Yes | No

Ratings

Don't forget to check the flag corresponding to the defense

Ok | Outstanding project

Empty work | Incomplete work | Norme | Cheat | Crash

Concerning situation | Leaks | Forbidden function

Can't support / explain code

Conclusion

Give this repository a star.