SCALE FOR PROJECT GET_NEXT_LINE (/PROJECTS/42CURSUS-GET_NEXT_LINE)

You should evaluate 1 student in this team

★

Git repository

```
git@vogsphere.42heilbronn.de:vogsphere/intra-uuid-db598da3-6880-4be9
```

Introduction

Please adhere to the following rules:

- Always remain polite, courteous, respectful, and constructive during the evaluation process. The community's well-being relies on this.

- Work with the student or group to identify potential issues in their project. Take the time to discuss and debate any problems that may arise.

- Keep in mind that your peers may have interpreted the project's instructions and scope differently. Always keep an open mind and grade them as honestly as possible. The pedagogy is effective only if peer evaluation is conducted 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). Confirm that it contains the expected project and that it has been cloned into an empty directory.

- Ensure that no malicious aliases have been used to mislead you into evaluating content outside 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 issues such as an empty repository, non-functioning programs, Norm errors, or suspected cheating. In these cases, the evaluation process ends and the final grade is 0, or -42 in case of cheating. However, except in cases of cheating, students are strongly encouraged to review the submitted work together to identify mistakes that should not be repeated in the future.

- During the defense, ensure there are no segmentation faults or other unexpected, premature, or uncontrolled terminations of the program, else the final

grade is 0. Use the appropriate flag if necessary.
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 clearly explain the reasons to the evaluated student and ensure that both of you agree with the changes.

- 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 (https://cdn.intra.42.fr/pdf/pdf/172810/en.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 must be possible to compile the project with the usual flags, and with the -D BUFFER_SIZE as well.
The project must compile successfully both with and without this flag, in addition to the following 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, invalid file descriptor (e.g., 42) to get_next_line. 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 allow you to assess the robustness of the student's get_next_line implementation.
If there is an error, the evaluation stops here.

☑ Yes ☒ No

Bonus part

Evaluate the bonus part only if the mandatory part has been completed flawlessly and the error management properly handles unexpected or incorrect 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 before, but this time, launch multiple instances of get_next_line with different file descriptor.
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

Review the code to confirm that only one static variable is used.
Award the point accordingly.

☑ 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

Leave a comment on this evaluation (2048 chars max)

Finish evaluation