SCALE FOR PROJECT C PISCINE RUSH 01

Introduction

Please respect 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, a norm error, 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.

- Check that there are only the requested files available in the git repository. If not, the evaluation stop here.

Attachments

generator.py

subject.pdf

Preliminaries

Simple preliminaries

- Nothing submitted (or wrong file or directory) : 0, evaluation is over.
- Norm error: evaluation stops.
- As soon as you come across an exercise that isn't fully functional, evaluation stops. The following exercises won't be evaluated.
- Make sure that there are no garbage files in the repository, such as unused, hidden, or tmp files.

Yes | No

Eliminatory tests

First, run some elementary error handling tests. If you encounter any problem, discuss it with the students you're grading, but don't give them any points.

Carry out the following series of tests :

- Poorly shaped arguments: $> ./rush-01 "4 3 2 1 1 2 2 2 4 3 2 1 1 2 2 2 3" $> ./rush-01 "4 3 2 1 1 2 2 2 4 3 2 1 1 2 2" $> ./rush-01 "4 3 2 1 1 2 2 2 6 3 2 1 1 2 2 2" $> ./rush-01 "4321122243211222" $> ./rush-01 "Bonjours" Each of these (along with the others you're gonna try) should output an error
- A wrong grid: $> ./rush-01 "4 3 2 1 1 2 2 2 4 3 2 1 1 2 2 4"

=> This test is the perfect opportunity to check and discuss the control algorithm.

Yes | No

Features

Features testing

Create your own maps or use the map generator given in reference to generate maps. Try them and check for errors. Keep in mind that some maps have multiple solutions so check by yourself if the offered solution is correct.

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

Explanation

Ask the shyest team member to explain how the code works.

If the student can't explain it, no matter how other team members reply, leave No in the scale and carry on with the evaluation.

Proceed to asking a series of questions (try grilling the student who seems the most confused):

- Ask them about the algorithm they've used, without them being able to look at their code.
- Look for lines with bits transfers (<< and >>) and ask what they are.
- Look for complex lines and ask what it does.
- Ask them to explain recursivity.
- Which data structures ? And why ?
- Ask them what the return value is when the grid has been filled? What happens in the case of an impossible grid?
- Check malloc returns and break their program if possible => if it breaks, select Crash.
- Any other question that comes to mind, be creative ! :)

If any student cannot answer a question at any given time, it's considered cheating, select Cheat.

If explanations are satisfactory, tick Yes.

If even one student cannot answer a question, tick No.

Yes | No

Bonus

Bonuses

This part can be filled ONLY for those who've collected all points for features and explanation. Can you work with a bigger grid? Up to which? (5? 9?), give points accordingly, with a 9x9 grid being the highest.

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

Incomplete group

⊘ Forbidden function

Conclusion

Leave a comment on this evaluation

Preview!!!