Introduction

Please respect the following rules:

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

- Identify with the person (or the group) graded 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 correcting and the corrected students have reviewed the possible scripts used
to facilitate the grading.

- If the correcting 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.

Attachments

subject.pdf checker_Mac

Mandatory part

Reminder : Remember that for the duration of the defence, no segfault, nor other unexpected, premature, uncontrolled or unexpected termination of the program, else the final grade is 0. Use the appropriate flag. This rule is active thoughout the whole defence.

Memory leaks

Throughout the defence, pay attention to the amount of memory used by push_swap (using the command top for example) in order
to detect any anomalies and ensure that allocated memory is properly freed. If there is one memory leak (or more), the final grade is 0.

Yes
No

Error management

In this section, we'll evaluate the push_swap's error management.
If at least one fails, no points will be awarded for this section. Move to the next one.

- Run push_swap with non numeric parameters. The program must
display "Error".

- Run push_swap with a duplicate numeric parameter. The program
must display "Error".

- Run push_swap with only numeric parameters including one greater
than MAXINT. The program must display "Error".

- Run push_swap without any parameters. The program must not display anything and give the prompt back.

Yes

No

Push_swap - Identity test

In this section, we'll evaluate push_swap's behavior when given a list, which has already been sorted. Execute the following 3 tests. If at least one fails, no points will be awarded for this section. Move to the next one.

- Run the following command "$>./push_swap 42". The program
should display nothing (0 instruction).

- Run the following command "$>./push_swap 0 1 2 3". The program should display nothing (0 instruction).

- Run the following command "$>./push_swap 0 1 2 3 4 5 6 7 8
9". The program should display nothing (0 instruction).

Yes
No

Push_swap - Simple version

If the following test fails, no points will be awarded for this section. Move to the next one. Use the checker binary given on the
attachments.

- Run "$>ARG="2 1 0"; ./push_swap $ARG | ./checker_OS $ARG".
Check that the checker program displays "OK" and that the size of the list of instructions from push_swap is 2 OR 3. Otherwise the test fails.

Yes
No

Another simple version

Execute the following 2 tests. If at least one fails, no points will be awarded for this section. Move to the next one. Use the checker
binary given on the attachments.

- Run "$>ARG="1 5 2 4 3"; ./push_swap $ARG | ./checker_OS $ARG".
Check that the checker program displays "OK" and that the size of the list of instructions from push_swap isn't more than 12. Kudos if the size of the list of instructions is 8.

- Run "$>ARG="<5 random values>"; ./push_swap $ARG | ./checker_OS
$ARG" and replace the placeholder by 5 random valid values. Check that the checker program displays "OK" and that the size of the list of instructions from push_swap isn't more than 12. Otherwise this test fails. You'll have to specifically check that the program wasn't developed to only answer correctly on the test included in this scale. You should repeat this test couple of times with several permutations before you validate it.

Yes

No

Push_swap - Middle version

If the following test fails, no points will be awarded for this section. Move to the next one. Move to the next one. Use the checker
binary given on the attachments.

- Run "$>ARG="<100 random values>"; ./push_swap $ARG | ./checker_OS $ARG" and replace the placeholder by 100 random
valid values. Check that the checker program displays "OK" and that the size of the list of instructions.
Give points in accordance:
- less than 700: 5
- less than 900: 4
- less than 1100: 3
- less than 1300: 2
- less than 1500: 1
You'll have to specifically check that the program wasn't developed to
only answer correctly on the test included in this scale.
You should repeat this test couple of times with several permutations before you validate it.

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

Push_swap - Advanced version

If the following test fails, no points will be awarded for this section. Move to the next one. Move to the next one. Use the checker
binary given on the attachments.

- Run "$>ARG="<500 random values>"; ./push_swap $ARG | ./checker_OS $ARG" and replace the placeholder by 500 random
valid values (One is not called John/Jane Script for nothing). Check that the checker program displays "OK" and that the size of the list of instructions
- less than 5500: 5
- less than 7000: 4
- less than 8500: 3
- less than 10000: 2
- less than 11500: 1

You'll have to specifically check that the program wasn't developed to
only answer correctly on the test included in this scale.
You should repeat this test couple of times with several permutations before you validate it.

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

Bonus

Reminder : Remember that for the duration of the defence, no segfault, nor other unexpected, premature, uncontrolled or unexpected termination of the program, else the final grade is 0. Use the appropriate flag. This rule is active throughout the whole defence. We will look at your bonuses if and only if your mandatory part is EXCELLENT. This means that you must complete the mandatory part, beginning to end, and your error management needs to be flawless, even in cases of twisted or bad usage. So if the mandatory part didn't score all the point during this defence bonuses will be totally IGNORED.

Checker program - Error management

In this section, we'll evaluate the checker's error management. If at least one fails, no points will be awarded for this section. Move to the next one.

- Run checker with non numeric parameters. The program must display "Error".

- Run checker with a duplicate numeric parameter. The program must display "Error".

- Run checker with only numeric parameters including one greater
than MAXINT. The program must display "Error".

- Run checker without any parameters. The program must not display anything and give the prompt back.

- Run checker with valid parameters, and write an action that doesn't exist during the instruction phase. The program must display "Error".

- Run checker with valid parameters, and write an action with one or several spaces before and/or after the action during the instruction phase. The program must display "Error".

Yes
No

Checker program - False tests

In this section, we'll evaluate the checker's ability to manage a list of instructions that doesn't sort the list. Execute the following 2 tests. If at least one fails, no points will be awarded for this section. Move to the next one.

Don't forget to press CTRL+D to stop reading during the intruction phase.

- Run checker with the following command "$>./checker 0 9 1 8 2
7 3 6 4 5" then write the following valid action list "[sa, pb, rrr]". Checker should display "KO".

- Run checker with a valid list as parameter of your choice then write a valid instruction list that doesn't order the integers. Checker should display "KO". You'll have to specifically check that the program wasn't developed to only answer correctly on the test included in this scale. You should repeat this test couple of times with several permutations before you validate it.

Yes

No

Checker program - Right tests

In this section, we'll evaluate the checker's ability to manage a liss of instructions that sort the list. Execute the following 2 tests. If at least one fails, no points will be awarded for this section. Move to the next one.

Don't forget to press CTRL+D to stop reading during the instruction phase.

- Run checker with the following command "$>./checker 0 1 2" then press CTRL+D without writing any instruction. The program should display "OK".

- Run checker with the following command "$>./checker 0 9 1 8 2"
then write the following valid action list "[pb, ra, pb, ra, sa, ra, pa, pa]". The program should display "OK".

- Run checker with a valid list as parameter of your choice then write a valid instruction list that order the integers.
Checker must display "OK". You'll have to specifically check that the program wasn't developed to only answer correctly on the test included in this scale. You should repeat this test couple of times with several permutations before you validate it.