(https://
profile.intra.42.fr)

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

You should evaluate 1 student in this team

★

Git repository

`git@vogsphere.42paris.fr:vogsphere/intra-uuid-add8cc94-812f-4012-84a` 📋

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 possible dysfunctions in the evaluated student's or group's 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. Pedagogy is effective only if peer evaluation is taken seriously.

Guidelines

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

- Ensure that the Git repository belongs to the student(s) and that
the project is the expected one. Additionally, ensure that 'git clone'
is executed in an empty folder.

- Check carefully that no malicious aliases were 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, students are
strongly encouraged to review together the work that was submitted, in order
to identify any mistakes that shouldn't be repeated in the future.

- 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. If memory leaks are detected, select
the appropriate flag.

Attachments

☐ subject.pdf (https://cdn.intra.42.fr/pdf/pdf/168831/en.subject.pdf)

☐ checker_Mac (https://cdn.intra.42.fr/document/document/35874/checker_Mac)

☐ checker_linux (https://cdn.intra.42.fr/document/document/35875/checker_linux)

Mandatory part

Reminder: During the defense, no segmentation faults, unexpected crashes, premature terminations, or uncontrolled exits are allowed. Otherwise, the final grade will be 0. Use the appropriate flag. This rule applies throughout the entire defense.

Memory leaks

Throughout the defense, monitor the memory usage of push_swap
(e.g., using the top command) to detect anomalies and ensure that
all 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" followed by a '\n' on the standard error.

- Run push_swap with a duplicate numeric parameter. The program
  must display "Error" followed by a '\n' on the standard error.

- Run push_swap with only numeric parameters including one greater
  than MAXINT. The program must display "Error" followed by a '\n'
  on the standard 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 will evaluate push_swap's behavior when given
an already sorted list. Execute the following 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 2 3".
  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).

- Run the following command "$>./push_swap <Between 0 and 9 randomly
  chosen sorted values>".
  The program should display nothing (0 instruction).

⊘ Yes   ✕ No

Push_swap - Simple version

If the following tests fail, no points will be awarded for this
section. Move to the next one. Use the checker binary provided in 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.

- Run "$>ARG="<Between 0 and 3 randomly chosen values>"; ./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 between 0 AND 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 provided in 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 instruction list from push_swap does not exceed 12.
  Kudos if the instruction list contains only 8 steps.

- 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. Use the checker
binary provided in 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 multiple times with different
    permutations before validating 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. Use the checker
binary provided in 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 the number of instructions is within the following limits:
    - 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.If the mandatory part does not receive a perfect score during this
defense, the bonus section will be completely 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" followed by a '\n' on the standard error.

- Run checker with a duplicate numeric parameter. The program
  must display "Error" followed by a '\n' on the standard error.

- Run checker with only numeric parameters including one greater
  than MAXINT. The program must display "Error" followed by a
  '\n' on the standard 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" followed by a '\n' on the standard 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"
  followed by a '\n' on the standard 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
instruction 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]". The 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.The 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 list 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 correctly orders 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.

� Yes   ✕ No

Ratings

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

✔ Ok   ★ Outstanding project

Empty work   🏴 Incomplete work   🕷 Invalid compilation   🎵 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