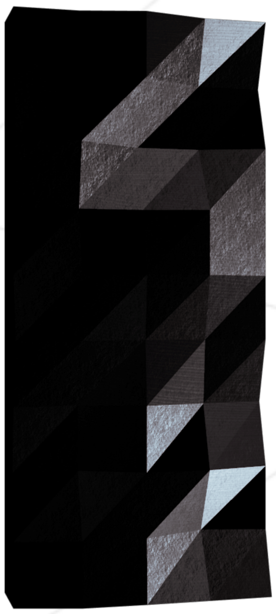

Push_swap

Because Swap_push isn't as natural

Staff pedago pedago@42.fr

Summary:
This project will make you sort data on a stack, with a limited set of instructions, using the lowest possible number of actions. To succeed you'll have to manipulate various types of algorithms and choose the one (of many) most appropriate solution for an optimized data sorting.

Contents

(https://profile.intra.42.fr/searches) **eluceon**

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

Remember that the quality of the defenses, hence the quality of the of the school on the labor market depends on you. The remote defences during the Covid crisis allows more flexibility so you can progress into your curriculum, but also brings more risks of cheat, injustice, laziness, that will harm everyone's skills development. We do count on your maturity and wisdom during these remote defenses for the benefits of the entire community.

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

You should evaluate 1 student in this team

★

Git repository

```
git@vogsphere.msk.21-school.ru:vogsphere/intra-uuid-9cbc351
```

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 (https://cdn.intra.42.fr/pdf/pdf/23502/en.subject.pdf)

checker_Mac (/uploads/document/document/3515/checker_Mac)

checker_linux (/uploads/document/document/3516/checker_linux)

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.

✓ Yes ✕ No

Ratings

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

✔ Ok ★ Outstanding project

Empty work Incomplete work No author file Invalid compilation Norme

Cheat Crash Leaks ⊘ Forbidden function

Conclusion

Leave a comment on this evaluation

Finish evaluation

Chapter I

Foreword

- C

```c
#include <stdio.h>

int main(void)
{
    printf("hello, world\n");
    return 0;
}
```

- ASM

```asm
cseg segment
assume cs:cseg, ds:cseg
org 100h
main proc
jmp debut
mess db 'Hello world!$'
debut:
mov dx, offset mess
mov ah, 9
int 21h
ret
main endp
cseg ends
end main
```

- LOLCODE

```
HAI
CAN HAS STDIO?
VISIBLE "HELLO WORLD!"
KTHXBYE
```

- PHP

```php
<?php
 echo "Hello world!";
?>
```

- BrainFuck

```
++++++++++[>+++++++>++++++++++>+++>+<<<<-]
>++.>+.+++++++..+++.>++.
<<+++++++++++++++.>.+++.------.--------.>+.>.
```

- C#

```
using System;

 public class HelloWorld {
    public static void Main () {
       Console.WriteLine("Hello world!");
    }
 }
```

- HTML5

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Hello world !</title>
    </head>
    <body>
        <p>Hello World !</p>
    </body>
</html>
```

- YASL

```
"Hello world!"
print
```

- OCaml

```
let main () =
    print_endline "Hello world !"

let _ = main ()
```

Chapter II

Introduction

The `Push_swap` project is a very simple and highly effective algorithm project: data will need to be sorted. You have at your disposal a set of int values, 2 stacks and a set of instructions to manipulate both stacks.

Your goal ? Write a program in `C` called `push_swap` which calculates and displays on the standard output the smallest program using `Push_swap` instruction language that sorts the integer arguments received.

Easy?

We'll see about that...

Chapter III

Goals

To write a sorting algorithm is always a very important step in a coder's life, because it is often the first encounter with the concept of complexity.

Sorting algorithms, and their complexities are part of the classic questions discussed during job interviews. It's probably a good time to look at these concepts because you'll have to face them at one point.

The learning objectives of this project are rigor, use of `C` and use of basic algorithms. Especially looking at the complexity of these basic algorithms.

Sorting values is simple. To sort them the fastest way possible is less simple, especially because from one integers configuration to another, the most efficient sorting algorithm can differ.

Chapter IV

General Instructions

- This project will only be corrected by actual human beings. You are therefore free to organize and name your files as you wish, although you need to respect some requirements listed below.
- The executable file must be named `push_swap`.
- You must submit a `Makefile`. That `Makefile` needs to compile the project and must contain the usual rules. It can only recompile the program if necessary.
- If you are clever, you will use your library for this project, submit also your folder `libft` including its own `Makefile` at the root of your repository. Your `Makefile` will have to compile the library, and then compile your project.
- Global variables are forbidden.
- Your project must be written in `C` in accordance with the Norm.
- You have to handle errors in a sensitive manner. In no way can your program quit in an unexpected manner (Segmentation fault, bus error, double free, etc).
- Neither program can have any `memory leaks`.
- Within your mandatory part you are allowed to use the following functions:
  - `write`
  - `read`
  - `malloc`
  - `free`
  - `exit`
- You can ask questions on the forum & Slack...

Chapter V

Mandatory part

V.1 Game rules

- The game is composed of 2 stacks named `a` and `b`.
- To start with:
  - `a` contains a random number of either positive or negative numbers without any duplicates.
  - `b` is empty
- The goal is to sort in ascending order numbers into stack `a`.
- To do this you have the following operations at your disposal:

sa : `swap a` - swap the first 2 elements at the top of stack `a`. Do nothing if there is only one or no elements).

sb : `swap b` - swap the first 2 elements at the top of stack `b`. Do nothing if there is only one or no elements).

ss : `sa` and `sb` at the same time.

pa : `push a` - take the first element at the top of `b` and put it at the top of `a`. Do nothing if `b` is empty.

pb : `push b` - take the first element at the top of `a` and put it at the top of `b`. Do nothing if `a` is empty.

ra : `rotate a` - shift up all elements of stack `a` by 1. The first element becomes the last one.

rb : `rotate b` - shift up all elements of stack `b` by 1. The first element becomes the last one.

rr : `ra` and `rb` at the same time.

rra : `reverse rotate a` - shift down all elements of stack `a` by 1. The last element becomes the first one.

rrb : `reverse rotate b` - shift down all elements of stack `b` by 1. The last element becomes the first one.

rrr : `rra` and `rrb` at the same time.

V.2 Example

To illustrate the effect of some of these instructions, let's sort a random list of integers. In this example, we'll consider that both stack are growing from the right.

```
-----------------------------------------------------------------------------------------------------------
Init a and b:
2
1
3
6
5
8
_ _
a b
-----------------------------------------------------------------------------------------------------------
Exec sa:
1
2
3
6
5
8
_ _
a b
-----------------------------------------------------------------------------------------------------------
Exec pb pb pb:
6 3
5 2
8 1
_ _
a b
-----------------------------------------------------------------------------------------------------------
Exec ra rb (equiv. to rr):
5 2
8 1
6 3
_ _
a b
-----------------------------------------------------------------------------------------------------------
Exec rra rrb (equiv. to rrr):
6 3
5 2
8 1
_ _
a b
-----------------------------------------------------------------------------------------------------------
Exec sa:
5 3
6 2
8 1
_ _
a b
-----------------------------------------------------------------------------------------------------------
Exec pa pa pa:
1
2
3
5
6
8
_ _
a b
-----------------------------------------------------------------------------------------------------------
```

This example sort integers from `a` in 12 instructions. Can you do better ?

V.3 The "push_swap" program

- You have to write a program named `push_swap` which will receive as an argument the stack `a` formatted as a list of integers. The first argument should be at the top of the stack (be careful about the order).
- The program must display the smallest list of instructions possible to sort the stack `a`, the smallest number being at the top.
- Instructions must be separaed by a '\n' and nothing else.
- The goal is to sort the stack with the minimum possible number of operations. During defence we'll compare the number of instructions your program found with a maximum number of operations tolerated. If your program either displays a list too big or if the list isn't sorted properly, you'll get no points.
- In case of error, you must display `Error` followed by a '\n' on the standard error. Errors include for example: some arguments aren't integers, some arguments are bigger than an integer, and/or there are duplicates.

```
$>./push_swap 2 1 3 6 5 8
sa
pb
pb
pb
sa
pa
pa
pa
$>./push_swap 0 one 2 3
Error
$>
```

During the defence we'll provide a binnary to properly check your program. It will work as follows:

```
$>ARG="4 67 3 87 23"; ./push_swap $ARG | wc -l
      6
$>ARG="4 67 3 87 23"; ./push_swap $ARG | ./checker_OS $ARG
OK
$>
```

If the program `checker_OS` displays `KO`, it means that your `push_swap` came up with a list of instructions that doesn't sort the list. The `checker_OS` program is available in the resources of the project on the intranet. You can find in the bonus section of this document a description of how it works.

Chapter VI

Bonus part

We will look at your bonus part if and only if your mandatory part is EXCELLENT. This means that your 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 that's not the case, your bonuses will be totally IGNORED.

How interesting could it be to code yourself your own checker? Vvvvvvery interesting!!

VI.1 The "checker" program

- Write a program named `checker`, which will get as an argument the stack `a` formatted as a list of integers. The first argument should be at the top of the stack (be careful about the order). If no argument is given `checker` stops and displays nothing.
- `checker` will then wait and read instructions on the standard input, each instruction will be followed by '\n'. Once all the instructions have been read, `checker` will execute them on the stack received as an argument.
- If after executing those instructions, stack `a` is actually sorted and `b` is empty, then `checker` must display `"OK"` followed by a '\n' on the standard output. In every other case, `checker` must display `"KO"` followed by a '\n' on the standard output.
- In case of error, you must display `Error` followed by a '\n' on the **standard error**. Errors include for example: some arguments are not integers, some arguments are bigger than an integer, there are duplicates, an instruction don't exist and/or is incorrectly formatted.

Thanks to the checker program, you will be able to check if the list of instructions you'll generate with the program push_swap is actually sorting the stack properly.

```
$>./checker 3 2 1 0
rra
pb
sa
rra
pa
OK
$>./checker 3 2 1 0
sa
rra
pb
KO
$>./checker 3 2 one 0
Error
$>./checker "" 1
Error
$>
```

You DO NOT have to reproduce the exact same behavior as the binary we are giving to you. It is mandatory to manage the errors but it is up to you how you decide to parse the arguments.

Chapter VII

Submission and peer correction

Submit your work on your `GiT` repository as usual. Only the work on your repository will be graded.

Good luck to all!