

# SCALE FOR PROJECT FT\_ALITY (/PROJECTS/FT\_ALITY)

## Introduction

To ensure this evaluation goes smoothly, please respect the following set of rules :

- Please remain courteous, polite, respectful and constructive at all times during this exchange. The trust bond between the school's community and yourself depends on it.
- Should you notice any malfunctions within the submitted project, make sure you take the time to discuss those with the student (or group of students) being graded.
- Keep in mind that some subjects can be interpreted differently. If you come accross a situation where the student you're grading has interpreted the subject differently than you, try and judge fairly whether their interpretation is acceptable or not, and grade them accordingly. Our peer-evaluation system can only work if you both take it seriously.

## Guidelines

- You may only evaluate whatever is in the GiT submission directory of the student you are grading.
- Make sure to check wether the GiT submission directory belongs to the student (or group) you're grading, and that it's the right project.
- Make sure no mischievous aliases have been used to trick you into correcting something that is not actually in the official submitted directory.
- Any script created to make this evaluation session easier - whether

it was produced by you or the student being graded - must be checked rigorously in order to avoid bad surprises.


- If the student who is grading this project hasn't done the project him/herself yet, he/she must read the whole topic before starting the evaluation session.

- Use the flags available to you on this scale in order to report a submission directory that is empty, non-functional, that contains a norm errors or a case of cheating, etc...

In this case, the evaluation session ends and the final grade is 0 (or -42, in case of cheating). However, unless the student has cheated, we advise you to go through the project together in order for the two (or more) of you to identify the problems that may have led for this project to fail, and avoid repeating those mistakes for future projects.

---

## Attachments

 Subject ([https://cdn.intra.42.fr/pdf/pdf/3905/ft\\_ality.en.pdf](https://cdn.intra.42.fr/pdf/pdf/3905/ft_ality.en.pdf))

## Getting started

*This section is dedicated to setup the evaluation and to test the prerequisites. It does not reward points, but if something is wrong at this step or at any point of the evaluation, the grade is 0, and an appropriate flag might be checked if needed.*

---

### Compliance to the rules

- The graded student (or team) work is present in his or her repository.

- The graded student (or team) is able to explain his or her work at any time of the evaluation.

- The project is in OCaml. If it's not, grade a -42.

- The general rules and the possible day-specific rules are respected at any time of the evaluation.

- The compiler does not report any errors or warnings.

- The Makefile does not relink.

 Yes

 No

---

## Mandatory part

*This section covers the most elementary and fundamental features demanded by the subject. It has to be completed perfectly to unlock bonuses.*

---

### Automaton

There must be an automaton implemented somewhere in the code. It could be a class, a module, or it could be lost somewhere in one ginormous file containing the entire project. If you find the latter, please frown at the student and slap the back of his/her head.

The automaton has to at least loosely correspond to the formal definition given in the subject. In other words, you should be able to find some states, some recognition states, and some transitions.

Obviously, the student must have implemented the automaton him/herself. If he/she hasn't, (i.e. a magical automaton library was imported or the student cannot explain the code), that's a -42 and a slap.

☒ Yes

☐ No

---

### Basic program behaviour

Now it is time to run the program! Take a quick look at the grammar files, and try to make the program recognize some moves.

In this question we'll keep it simple. Run the program with any grammar you'd like. The program should display the keyboard mapping, and then wait for the user to press some keys; go ahead and check that the program detects you are pressing some keys, and print what the user is inputting. And from time to time, you should see move names popping up.

☒ Yes

☐ No

---

### Small grammars

Now make a small grammar file with only three or four moves, and run the program with it. You should be able to execute EVERY move from the file.

This question is also the opportunity to check if the keyboard mapping is computed from the grammar file. Because it has to be. If it is not, the question is failed.

☒ Yes

☐ No

---

### Homonymous rules

Make another small grammar with several rules, but all the rules must be executed through the same key combinations. Now run the program, and execute the key combinations. You must be able to see EVERY rule name in the grammar.

☒ Yes

☐ No

---

### Common prefixes

Make another small grammar with several rules, but all the rules must have the same prefix. For example, you could have something like:

Move A;a  
Move B;a,b  
Move C;a,b,a

The program should be able to display move names as they appear, and the user must be able to execute every move in your grammar.

☒ Yes

☐ No

---

## Error handling

*No one likes nitpicking and hunting for bugs for hours just to have the pleasure to grade the project a 0. But still, functional programming is also about rigor, so we have to at least test some simple erratic behaviours. In this section as well as in the rest of the defence, any uncaught exception, infinite loop, freeze or erratic behaviour whatsoever means the defence ends with a "Crash" status. Which means the grade is 0.*

---

### Command line arguments

Try running the program with no arguments, more than one argument, a file that does not exist and a file the program is not allowed to read (e.g. /etc/master.passwd). The program must exit gracefully.

✓ Yes

✗ No

### Input

Mash your keyboard. I mean, gently. But do try to press keys outside the keyboard mapping and try to execute keys that do not constitute a particular move in the grammar. The program must reset the input buffer when appropriate and keep running whatever happens.

✓ Yes

✗ No

## Bonus

*Bonus part! This part rewards the projects where special efforts have been made into making the end program awesome.*

### Bonus points

You can give bonus points as you like, but here are some bonus ideas you can give points for:

- Support for game pads
- Graphic interface
- Debug mode
- Clean and organised code

✓ 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

👤 Incomplete group

🚫 Forbidden function

## Conclusion

Leave a comment on this evaluation

Preview!!!

General term of use of the site  
(<https://signin.intra.42.fr/legal/terms/6>)

Privacy policy  
(<https://signin.intra.42.fr/legal/terms/5>)

Legal notices  
(<https://signin.intra.42.fr/legal/terms/3>)

Declaration on the use of cookies  
(<https://signin.intra.42.fr/legal/terms/2>)

Terms of use for video surveillance  
(<https://signin.intra.42.fr/legal/terms/1>)

Terms of use for video surveillance  
(<https://signin.intra.42.fr/legal/terms/1>)