{EPITECH}
LEARN DIFFERENT*

ALICE'S JOURNEY

FOLLOWING THE WHITE RABBIT

ALICE'S JOURNEY

binary name: solver
language: C
compilation: via Makefile, including re, clean and fclean rules

✓ The totality of your source files, except all useless files (binary, temp files, objfiles,...), must be included in your delivery.

Alice just saw a talking white rabbit wearing a waistcoast, that ran past her shouting "Oh dear! Oh dear! I shall be too late!". Weird.
She needs to follow the rabbit through a maze-like place.

The goal of this project is to solves mazes (in a reasonable amount of time) and print the solution.

System functions and libC are allowed, but not external libraries.

Turn-in methods

Your solver must be able to function as follows:

```
Terminal
~/B-CPE-000> ./solver maze.txt
```

The width/height of the given maze.txt must be between 1 and 10000

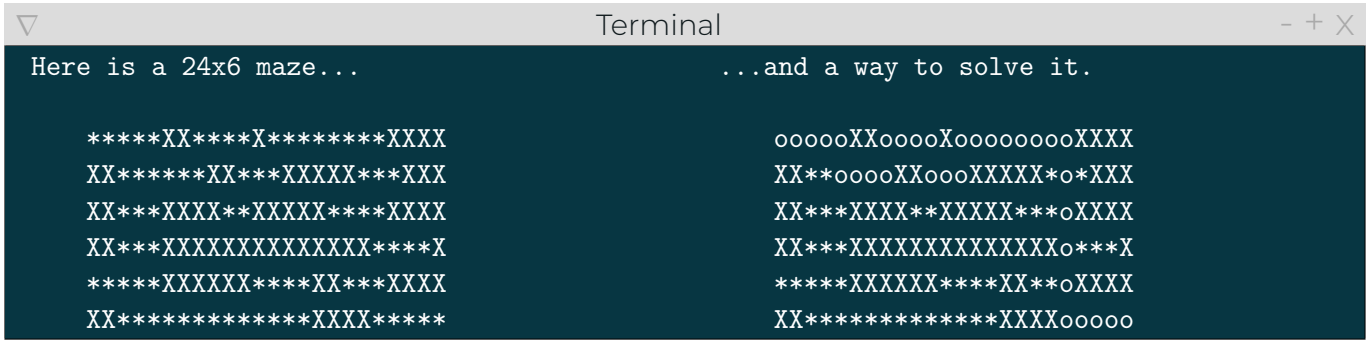
and print the solved maze on the standard output.
If there is no solution you must print "**no solution found**" on the same output.

What is a maze?

Here is the maze format description:

- ✓ Mazes are rectangular.
- ✓ They are coded in ASCII.
- ✓ The 'X's represent the walls and the'*'s represent the free spaces.
- ✓ It is possible to move to the four surrounding squares (up, down, right, left).
- ✓ "Start" is in the upper left-hand corner (0;0)
- ✓ "Finish" is in the bottom right-hand corner.
- ✓ A solution is a series of free, adjacent squares, from "Start" to "Finish" included.
- ✓ "Start" and "Finish" can be occupied. In this case, there is no solution.
- ✓ The last line of the maze doesn't terminate with a return line.
- ✓ Resolution: in order to write the solution in the maze, we use 'o' for the solution path.

Example



Escaping the maze...

Your resolution program should take the name of a file containing the maze and write the solution on the standard output.
You can choose the algorithm you want but be careful, you will be ask to solve perfect and imperfect mazes.

Performance

Solving mazes is good, but how efficient is the algorithm you use?
Will you be able to deal with very large mazes?
Will you be able to do it quickly enough?

You will have to choose between RAM and CPU... think about it.

Measuring the speed of your program raises some points:

- ✓ the system-dependency: your machine is not always strictly in the same state when the program is launched (resource consumption, sticky bit,...),
- ✓ the maze-dependency: you will get different results for different types of mazes
- ✓ randomness: if it turns out that your algorithm uses randomness, the results will vary constently, even for the same maze.

Further into optimization

There are a lot of possible optimizations, whether it be heuristic searches, data structure work, local optimizations, maze preprocessing,...

Be creative!

Heuristic

When we simply scan our state space, you might get the idea to scan it in a slightly more enlightened manner.
You probably alrealdy integrated a heuristic! Didn't you?

Be sure you are using the best possible heuristic.

Don't forget to measure the newly obtained performance.
Is it more efficient? For all of the mazes?


{EPITECH}
LEARN DIFFERENT*