# OS2 Projects 2019/2020

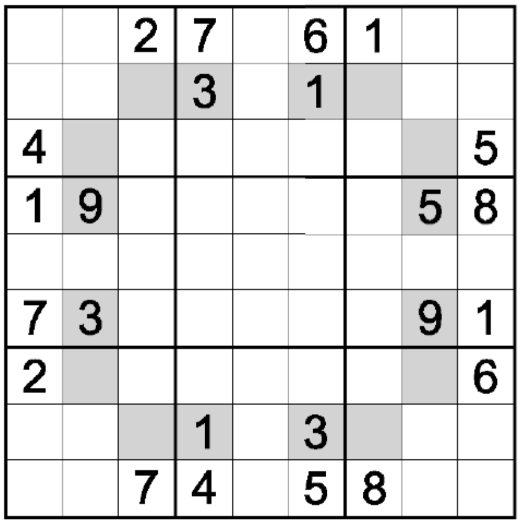
*General Notes:*

- *you must implement the project in both **sequential** and **parallel** code*
- *you must use C# programming language*
- *You must provide a GUI for your project*

## 1. Sudoku Solver

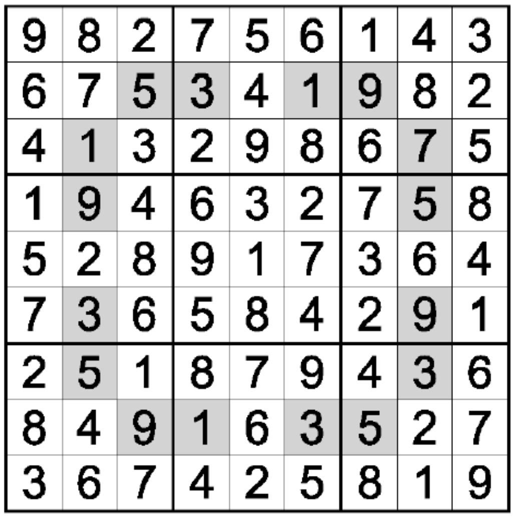
Place a digit from 1 to 9 into each of the empty squares so that each digit appears exactly once in each of the rows, columns and the nine outlined 3x3 regions.

Sudoku



| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 2 | 7 | | 6 | 1 | | |
| | | | 3 | | 1 | | | |
| 4 | | | | | | | | 5 |
| 1 | 9 | | | | | | 5 | 8 |
| | | | | | | | | |
| 7 | 3 | | | | | | 9 | 1 |
| 2 | | | | | | | | 6 |
| | | | 1 | | 3 | | | |
| | | 7 | 4 | | 5 | 8 | | |

Solution



| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 9 | 8 | 2 | 7 | 5 | 6 | 1 | 4 | 3 |
| 6 | 7 | 5 | 3 | 4 | 1 | 9 | 8 | 2 |
| 4 | 1 | 3 | 2 | 9 | 8 | 6 | 7 | 5 |
| 1 | 9 | 4 | 6 | 3 | 2 | 7 | 5 | 8 |
| 5 | 2 | 8 | 9 | 1 | 7 | 3 | 6 | 4 |
| 7 | 3 | 6 | 5 | 8 | 4 | 2 | 9 | 1 |
| 2 | 5 | 1 | 8 | 7 | 9 | 4 | 3 | 6 |
| 8 | 4 | 9 | 1 | 6 | 3 | 5 | 2 | 7 |
| 3 | 6 | 7 | 4 | 2 | 5 | 8 | 1 | 9 |

To give you 9 * 9 grid, in this grid has been filled in part of the yard numbers.
Now is this game will have any solution?

- ⇨ If there is no solution, output "NO Solution Found"
- ⇨ If there is a solution, output the solution.

You must provide many sample inputs (many text files) during the discussion time, to test your project on many samples. Each text file represents one sudoku problem.

The input of the project should be in a text file, for example a sample input that represents the prev. figure could be:

```
**27*61**
***3*1***
4*******5
19*****58
*********
73*****91
2*******6
***1*3***
**74*58**
```

## 2. ***Maze***

You've landed on an alien planet, in the middle of some strange maze, and you want to find the quickest way out.
The maze can be thought of as a m×n grid where some of the squares in the grid are blocked. You know the entire maze (including the size of the grid, which squares are blocked, and your current location). You can drive north, south, west, or east from each square in your hovercar, but not diagonally. Unfortunately, your hovercar was damaged during the rough landing onto the planet, and it can't turn left or make U-turns: the hovercar can only go straight or turn right. For instance, if you leave the current square by going north, you'll only be able to leave the next square by going north or east; west is not possible, because that would require a left turn, and south is not possible either, because would require a U-turn. Your hovercar was landed facing north, so on your first move you will need to drive north.

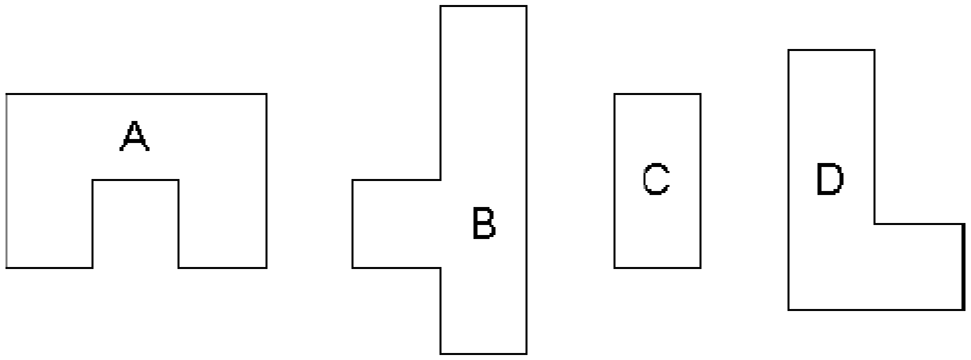
Find an efficient algorithm to find the shortest route to escape from the maze and implement it.

Input maze should be in text file filled with some characters, for example "." Could represent empty area, "*" could represent block, "c" could represents the car, "e" could represent the exit.

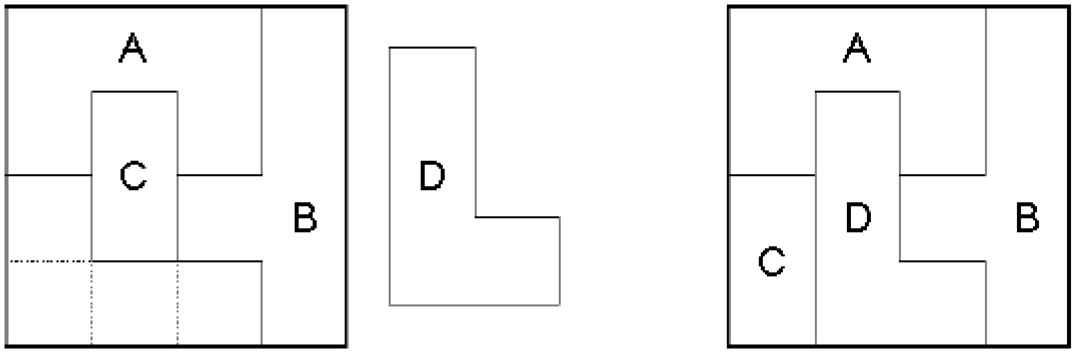
You must provide many sample inputs (many text files) during the discussion time, to test your project on many samples. Each text file represents one Maze.

## 3. ***Make a square***

Make a square with size 4X4 by using 4 or 5 pieces. The pieces can be rotated or flipped and all pieces should be used to form a square. Example sets of pieces.



There may be more than one possible solution for a set of pieces, and not every arrangement will work even with a set for which a solution can be found. Examples using the above set of pieces…
Rotate piece D 90 degree then flip horizontal {R 90 + F H}



**Input:**
The first line contains number of pieces. Each piece is then specified by listing a single line with two integers, the number of rows and columns in the piece, followed by one or more lines which specify the shape of the piece. The shape specification consists of 0 or 1 characters, with the 1 character indicating the solid shape of the puzzle (the 0 characters are merely placeholders). For example, piece A above would be specified as follows:

```
2 3
111
101
```

**Output:**
Your program should report all solution, in the format shown by the examples below. A 4-row by 4-column square should be created, with each piece occupying its location in the solution. The solid portions of piece #1 should be replaced with `1' characters, of piece #2 with `2' characters.

Sample output that represents the figure above could be:

```
1112
1412
3422
3442
```

For cases which have no possible solution simply report "No solution possible".

You must provide many sample inputs (many text files) during the discussion time, to test your project on many samples. Each text file represents one problem to be solved.