Learning to Program with F#
Exercises

Department of Computer Science
University of Copenhagen

Jon Sporring, Martin Elsman, Torben Mogensen, Christina Lioma

October 8, 2021

In this assignment, you are to work with a puzzle called Rotate. The puzzle consists of a square board with $n \times n, n \in \{2,3,4,5\}$ fields similarly to a chess-board. Each field has a unique id-number, which we will call the field's position, and on each field is one unique letter from the alphabet 'a', 'b', ... For example, when $n =$, then the board could look like,

$$\begin{array}{cccc}
\text{h} & \text{o} & \text{l} & \text{k} \\
\text{b} & \text{i} & \text{g} & \text{e} \\
\text{f} & \text{m} & \text{c} & \text{a} \\
\text{j} & \text{n} & \text{d} & \text{p}
\end{array}$$

and the position of the respective fields are

$$\begin{array}{cccc}
1 & 2 & 3 & 4 \\
5 & 6 & 7 & 8 \\
9 & 10 & 11 & 12 \\
13 & 14 & 15 & 16
\end{array}$$

The puzzle is solved by rotating the letters in small 2×2 subsquares clockwise until the board reaches the state

$$\begin{array}{cccc}
\text{a} & \text{b} & \text{c} & \text{d} \\
\text{e} & \text{f} & \text{g} & \text{h} \\
\text{i} & \text{j} & \text{k} & \text{l} \\
\text{m} & \text{n} & \text{o} & \text{p}
\end{array}$$

A rotation is specified by the position of its top-left corner, and all but the right-most column and the bottom-most row are valid inputs to the rotation operation. Let $p_1, p_2, p_3, p_4 \to q_1, q_2, q_3, q_4$ denote a rotation from p_* to q_*, where p_1 is the top-left corner, then for example, specifying a rotation of subsquare 1 results in $1,2,5,6 \to 2,6,5,1$, or equivalently,

$$\begin{array}{cccc}
\text{h} & \text{o} & \text{l} & \text{k} \\
\text{b} & \text{i} & \text{g} & \text{e} \\
\text{f} & \text{m} & \text{c} & \text{a} \\
\text{j} & \text{n} & \text{d} & \text{p}
\end{array}
\rightarrow
\begin{array}{cccc}
\text{b} & \text{h} & \text{l} & \text{k} \\
\text{i} & \text{o} & \text{g} & \text{e} \\
\text{f} & \text{m} & \text{c} & \text{a} \\
\text{j} & \text{n} & \text{d} & \text{p}
\end{array}$$

The overall task of this assignment is to build a program, that will generate rotate-puzzles and allow you to iteratively enter a sequence of positions untill the puzzle is solved. Detailed requirements are:

- If your program includes loops, then the loops must be programmed using recursion
- Your program must use lists and not arrays.
- Your program must not use mutable values (variables).
- Your solution must be parametrized by n, the size of the board.
- you must represent your board as a list of letters, e.g., for $n = 4$ the board for a solved puzzle must be `['a' .. 'p']`

- Your program must consist of the following files `game.fsx`, `rotate.fsi`, `rotate.fs`, `whiteboxtest.fsx` and `blackboxtest.fsx`. The files `rotate.fsi`, `rotate.fs` must be the interface and implementation of a library with your main types, functions, and values; the file `game.fsx` must be a maximally 10 line program, which defines the value *n* and starts the game; and `test.fsx` must contain your test for the library.

As part of this assignment, you are to write a maximally 10-page report following `rapport.tex` template.

Note that calls to `System.Random ()` returns a random number generator which has a function `Next : n:int -> int` which draws a random non-negative integer less than `n`. For example,

Listing 1: Generating random integers.

```
let rnd = System.Random ()
for i = 1 to 3 do
  printfn "%d" (rnd.Next 10)
```

prints 3 random non-negative integers less than 10.

0.1 Rotate

0.1.1 Opgave(r)

0.1.1: Write the interface file for the library `rotate` with user defined types `Board` which is a list of characters and `Position` which is an integer and with the following functions,

```
create : n:uint -> Board
board2Str : b:Board -> string
validPosition : b:Board -> p:Position -> bool
rotate : b:Board -> p:Position -> Board
scramble : b:Board -> m:uint -> Board
solved : b:Board -> bool
```

The function `create` must take an integer *n* and return a $n \times n$ board in its solved state.

The function `board2Str` must take a board and return a string, containing the board formatted such that it can be printed with the `printfn "%s"` command and formatting string.

The function `validPosition` must take a board and rotation position and return true or false depending on whether the position is a valid rotation position or not.

The function `rotate` must take a board and a position and return another board, which is identical to the original but where a local 2×2 rotation has been performed at the indicated position. If an invalid position is given, then the function must return an empty board.

The function `scramble` must take a board and an unsigned int `m` and return another board, where all the elements of the original board have been rotated by `m` random set of legal rotations using `rotate`.

The function `solved` must take a board and return true or false depending on whether the puzzle has been solved or not.

The interface must include documentation following the documentation standard.

0.1.2: Write a program `blackboxtest.fsx` which performs a blackbox test of the yet to be implemented `rotate` library.

0.1.3: Write the implementation file of the `rotate` library.

0.1.4: Write a program `whiteboxtest.fsx` which performs a whitebox test of the `rotate` library.

0.1.5: Write a short program, `game`, which defines the size of the board *n* and starts the game, and which has all the interaction with the user in a game-loop.

0.1.6: Extend your libray and your whiteboxtest with a function

```
solutions : Board -> int -> int list list
```

which takes an integer *m* and seeks possible solutions to a given board within maximally *m* rotations. The function should return 0 or more lists of possible rotation sequences, of length no longer than *m*, and which solves the puzzle.