Learning to Program with F#
Exercises
Department of Computer Science
University of Copenhagen

Jon Sporring, Martin Elsman, Torben Mogensen, Christina Lioma

October 21, 2022

0.1 Rotate

0.1.1 Teacher's guide

Emne Rekursion og lister

Sværhedsgrad Middel

0.1.2 Introduction

In this assignment, you will be working with a puzzle called Rotate. The puzzle consists of a square chess-like board with $n \times n, n \in \{2,3,4,5\}$ fields. Each field has a unique id-number, which we will call the field's position, and, for a particular configuration of the board, each field is associated with a unique letter from the alphabet 'a', 'b', ... For example, with $n = 4$, here is a possible board configuration:

h	o	l	k
b	i	g	e
f	m	c	a
j	n	d	p

Moreover, the positions of the fields are laid out as follows:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

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

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

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 $p_1, p_2, p_3, p_4 \rightarrow q_1, q_2, q_3, q_4$ denote a rotation from p_* to q_*, where p_1 is the top-left corner. Then, a rotation of subsquare 1 results in $1,2,5,6 \rightarrow 5,1,6,2$, or equivalently,

h	o	l	k		b	h	l	k
b	i	g	e	$\rightarrow$	i	o	g	e
f	m	c	a		f	m	c	a
j	n	d	p		j	n	d	p

The overall task of this assignment is to build a program that generates rotate-puzzles and that allows you iteratively to enter a sequence of positions until the puzzle is solved.

Here is a list of detailed requirements:

- If your program includes loops, the loops must be programmed using recursion.
- Your program must use lists and not arrays.
- Your program is not allowed to use mutable values (or variables).
- Your solution must be parameterized by n, the size of the board.
- You must represent your board as a list of letters. Thus, for $n = 4$, the board for a solved puzzle must be identical to the list `['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` and `rotate.fs` must constitute the interface and the implementation of a library with your main types, functions, and values; the file `game.fsx` must be a maximally 10-line program that defines the value n and starts the game; and `whiteboxtest.fsx` and `blackboxtest.fsx` must contain your tests for the library.

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

Notice that calls to `System.Random ()` returns a random number generator object. This object has a method `Next : n:int -> int`, which draws a random non-negative integer less than `n`. For example, the code

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

prints 3 random non-negative integers less than 10.

0.1.3 Exercise(s)

0.1.3.1: Write the interface file for the library `rotate`. The interface should specify two user-defined types, named `Board` and `Position`, which are defined to be a list of characters and an integer, respectively. The interface should also specify the following functions:

```
create : n:uint -> Board
board2Str : b:Board -> string
validRotation : 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 as argument and return an $n \times n$ board in its solved state.

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

The function `validRotation` must take a board and a rotation position as arguments 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 rotation position as arguments and return a board identical to the original but where a local 2×2 rotation has been performed at the indicated position. If an invalid position is given, the function must return the board that was passed as the first argument.

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

The function `solved` must take a board as argument and return true or false depending on whether the board is in the solved configuration.

The interface must include documentation following the documentation standard.

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

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

0.1.3.4: Write a program `whiteboxtest.fsx` that performs a whitebox test of the `rotate` library.

0.1.3.5: Write a short program `game` that defines the size of the board n, starts the game, and implements the interaction with the user in a game-loop using recursion.

0.1.3.6: A fellow programmer has made the function

```
solve : b:Board -> m:int -> int list
```

which takes as argument a non-negative integer m and returns either the empty list or a list of rotation positions (of length m or less) that will leave the board in the solved configuration (if one such list exists). The program compiles and runs without error, but for some combinations of board-size n and maximum rotations m, the program is very slow and the computer eventually runs out of memory. Why do you think this may be?