INF-122 Compulsory assignment 2, fall 2017

Prelude

- Read the entire set – 3 pages – before you start working out your solution.
- On 122-page, clicking on "Oppgåver" and then "Obligatorisk oppgave 2" opens the page of the assignment. Choosing (on this page, again) "Obligatorisk oppgave 2" gives you the text of the assignment. The solution can be uploaded using the "Lever" button.
- Solution should be submitted as a plain-text file with the name **YourSurname.hs** (the suffix .hs is important) and contain interpretable code. (Files which are refused by the GHCi will not be considered and amount to immediate fail.) The file shall start with a comment containing your name followed by another comment, containing an answer to the problem specified at the very end of the assignment.
- It is allowed to discuss the approach and possible solutions, but everybody has to write an independent solution on his/her own. Suspicious repetitions of code pieces in distinct answers will qualify as an immediate fail. (What counts as a suspicious repetition is decided by the person evaluating the solutions.)
- The submission deadline is **Friday, November 17th 2017 at 12:00 (noon)**.
- ***There are no second chances, so take advantage of the workshops 13–17.11.***

Background: The Game of Life and cellular automata

You will program a special kind of cellular automata, the best know example of which is Conway's Game of Life. Referring indefinitely to "an automaton", we mean only the kind of automata addressed in this assignment.

Such an automaton runs (or, as one also says, a game is played) on an $N \times N$ grid, for some natural number $N > 0$, on which some "living" cells are initially placed. (Board is the grid, possibly with some living cells, representing a state of the automaton, i.e., a position in the game.) The first board below has living cells (2,3) and (3,4), while the second one (3,3) and (2,4).

```
  1 2 3 4 5          1 2 3 4 5
1 . . . . .        1 . . . . .
2 . . . . .        2 . . . . .
3 . X . .          3 . . X .
4 . . X . .        4 . X . . .
5 . . . . .        5 . . . . .
```
(1)

Each automaton (game) is defined by two rules determining which living cells survive to the next state (generation) and where new cells are born. This depends on the number of living neighbours in the current generation, where neighbours of each cell (x, y) are all cells at distance 1 from it, along the horizontal axis, vertical axis or diagonal. (That is, all cells (x', y') on the grid distinct from (x, y), where $|x' - x| \leq 1$ and $|y' - y| \leq 1$.) Below, `n`s mark all neighbours of `X` at (2,3).

```
  1 2 3 4 5
1 . . . . .
2 n n n . .
3 n X n . .
4 n n n . .
5 . . . . .
```

Border-vertices have fewer neighbours, e.g., the only neighbours of (1,1) are (1,2), (2,1) and (2,2). The rules are given by two triples (Char,Int,Int), specifying the survivors and new-borns ($x \leq y$):

`s` x y – a living cell survives if it has at least x and at most y living neighbours

`b` x y – a cell is born into an empty position with at least x and at most y living neighbours.

For instance, Conway's Game of Life is the automaton (`s 2 3`, `b 3 3`). With it, each board from drawing (1) becomes empty in the next generation. But the game (`s 2 2`, `b 2 2`), played on any of these two boards, would switch indefinitely between them.

The assignment

is to develop an interactive program allowing the user to execute the following commands. (An additional task is specified in point 4 of the following section **Additional requirements**.) Except for the carriage-return command, `CR`, the first character is a fixed identifier of the command; the following ones are parameters provided by the user – either *String* or Int. The latter are at least 0, and most of them are at least 1.

`c` n : create (and show) a new, empty board $n \times n$ (i.e., $n :: Int$ and we assume $1 \leq n \leq 99$)

`n` x y : place a new living cell at (x,y) ($x, y :: Int$ and $1 \leq x, y \leq n$ = the size of the grid)

`d` x y : make position (x,y) empty ($x, y :: Int$ and $1 \leq x, y \leq n$ = the size of the grid)

`s` x y : redefine the automaton so that a living cell with [x..y] living neighbours survives ($x, y :: Int$, $0 \leq x \leq y$)

`b` x y : redefine the automaton so that an empty cell with [x..y] living neighbours becomes alive ($x, y :: Int$, $0 \leq x \leq y$)

`?` : show the rules of the current game, i.e., the pairs: s x_s y_s and b x_b y_b. Choose appropriate place to show these, so that they do not overwrite the grid.

`w` *name* : write the current rules and board to the file *name* :: *String* (point 3 in **Additional requirements** below describes the required file format)

`r` *name* : read the rules and board from the file *name* :: *String* (formatted as in `w` command)

`CR` : step one generation forward. Tell the user if a **stable configuration** is reached, i.e., a generation identical to its next (or previous) generation.

`l` x : enter a 'live' mode, i.e., the game is played without user's interaction, showing the changes on the board, for $x \geq 1$ generations or until a stable configuration is reached before the x-th generation, which is then announced to the user. Make the changes visible, i.e., not too fast (nor too slow).

`q` quits the program.

Optional (these are not required, but may be fun to implement):

- `l` x y: implementing this one, you do not have to implement '`l` x'. Here $1 \leq y :: Int$ and '`l n 1`' is the same as '`l n`'. But for $y > 1$, the game is played (without user) for x generations until a repeating pattern of boards of length $\leq y$ is encountered.

  E.g., for automaton (`s 2 2`, `b 2 2`) run on any of the two boards from Figure (1), '`l 5 1`' continues for 5 generations, switching between the two, while '`l 5 2`' terminates after 2 steps, discovering the repeating pattern of length 2.

- `p` : switch back to the previous board.

Additional requirements

1. At each step, starting from the execution of the initial `c` or `r` command, the board is shown on the screen using the representation as in Figure (1): empty cells are marked with dots '.' and living ones with 'X'. Columns and rows are numbered explicitly as shown, in the upper line/row and in the first column(s). Take care of the alignment, especially, of the column numbers, which become longer from the 10th column on. You can assume that grid never has more than 99 rows/columns.

2. As illustrated by the examples, the grid is numbered with (x, y)=(1,1) being the upper left corner, and with the horizontal coordinate x increasing to the right and the vertical one, y, downwards.

3. For reading from and writing to a file, the following format must be used: the file contains one line, with characters and numbers (numbers may be larger than 9, i.e., are not only single digits) separated by whitespace. The first two pairs of numbers are preceded by s and b. They specify the 'surviving interval' and the 'born interval', as do commands `s` and `b`. The following, seventh token (the fifth number) is the size of the grid (number of rows = number of columns). I.e., the first seven tokens being $s\ x_1\ y_1\ b\ x_2\ y_2\ n$ mean that one plays the game (`s` x_1 y_1, `b` x_2 y_2) on a board $n \times n$. Then come the pairs of numbers specifying living cells on the board: each cell is represented by a pair x y, where x is the horizontal and y the vertical coordinate. Numbers are separated by whitespaces. As an example, the shape 4.(a) below, from Conway's Game of Life on a board of size n, could be written in a file as: s 2 3 b 3 3 n 3 1 1 2 3 2 2 2 3 2. (The first triple, s 2 3, can be swapped with the other b 3 3. Likewise, the order in which the position pairs are written, after n, can vary.)

4. The following shape (a)

(a)
```
. . X . .
X . X . .
. X X . .
. . . . .
```

(b)
```
. . . . .
. . . X .
. X . X .
. . X X .
```

is the so called "slider" in Conway's Game of Life, i.e., after a few moves, it will slide "diagonally" as shown in (b), and then will continue such sliding.

You shall also specify a game, i.e., the parameters x_s, y_s, x_b, y_b defining an automaton
(`s` x_s y_s, `b` x_b y_b)
and a placement C of living cells on a board (of a sufficient size), such that running the automaton on C will cause the shape C move horizontally (possibly, with some intermediary configurations).

This part has to be formatted as a file-text, described in point 3 above, and included as a comment at the top of the file, right after your name.