INF-122 Compulsory assignment 2, fall 2020

- Read the entire assignment before you start working out your solutions.
- It is allowed to discuss the approach and possible solutions, but everybody has to write an independent solution on his/her own. Code from the book, lectures and groups can be reused, but duplication of other parts of code will result in points being deducted, for all persons sharing them. Copying code from other sources is not allowed.
- Questions about the assignment can be asked at the workshops 2-3.11 and 9-10.11.
- Submit your solutions (using "Lever" button under Oblig2) not later than **Friday, 13th November 2020, at 13:00**. Solutions submitted after that will not be considered.
- Solution must be submitted as a plain-text, interpretable file with the name **Oblig2.hs** (both the name Oblig2 and the suffix .hs are essential). The file must start with
  - a comment containing your name, followed by
  - a comment specifying the pattern from problem 2.4,
  - after which you write your program in the usual way, with the main action named `main`.

1 Introduction: The Game of Life and cellular automata

You will program a special kind of cellular automata, the best known example being Conway's Game of Life. Throughout this assignment, when we refer to "an automaton" we only mean the kind of automata which is 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. A *board* is then defined as a grid, possibly with some living cells, which represents a state of the automaton, i.e. a position in the game. The first board in Figure 1 has living cells $(2, 3)$ and $(3, 4)$, while the second one has living cells $(3, 3)$ and $(2, 4)$. These cells are denoted by the letter `O`.

```
   1 2 3 4 5          1 2 3 4 5
1  . . . . .       1  . . . . .
2  . . O . .       2  . . . O .
3  . . . O .       3  . . O . .
4  . . . . .       4  . . . . .
5  . . . . .       5  . . . . .
```

Figure 1: Example boards

Each automaton (or game) is defined by two rules determining which living cells survive to the next state 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), such that $|x' - x| \leq 1$ and $|y' - y| \leq 1$. The boards below illustrate all the neighbours `n` of the living cells $(2, 4)$ and $(5, 2)$, respectively.

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

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

Figure 2: Neighbours of $(2, 4)$ and $(5, 2)$

Cells at the border have fewer neighbours, e.g., corner vertices only have three neighbours. In particular, the board **does not wrap up**, e.g. on the examples in Figure 2, cells in row 1 are **not** neighbours of cells in row 5, and cells in column 1 are **not** neighbours of cells in column 5.

The rules of each automata are given by two triples (Char, Int, Int), which specifies the survivors and the newborns ($m_1 \leq n_1,\ m_2 \leq n_2$):

`s` m_1 n_1 – a living cell survives if it has at least m_1 and at most n_1 living neighbours.

`b` m_2 n_2 – a cell is born into an empty position with at least m_2 and at most n_2 living neighbours

For instance, Conway's Game of Life is the automaton (`s 2 3`, `b 3 3`). For this automaton, the boards in Figure 1 becomes empty in the next generation as each living cell only has one neighbour. However, the game (`s 2 2`, `b 2 2`) played on any of these two boards would switch indefinitely between them.

Figure 3 shows an example of another board progression (left-to-right) in Conway's Game of Life. The rightmost board is said to be *stable* as the position of living cells remains in the same pattern for all future generations.

```
   1 2 3 4 5
1  O O . . .
2  O O O . .
3  . O . . .
4  . . . O .
5  . . . . O
```

```
   1 2 3 4 5
1  O . O . .
2  . . O . .
3  O O . . .
4  . . . . .
5  . . . . .
```

```
   1 2 3 4 5
1  . O . . .
2  O . O . .
3  . O . . .
4  . . . . .
5  . . . . .
```

Figure 3: An example progression in Conway's Game of Life

2 The assignment

2.1 General overview of commands

You are to develop an interactive program, action named `main`, allowing the user to execute the following commands. Except for the command `quit`, the first character is a fixed identifier of the command; the following ones are parameters provided by the user – either *String*, *Int* or list of *Int* with even length.

`c` n – create and show a new, empty board of size $n \times n$.
($n :: Int$ and $1 \leq n \leq 99$)

`n` x_1 y_1 **...** x_k y_k – place k living cells at $(x_1, y_1), ..., (x_k, y_k)$.
($x_i, y_i :: Int$ and $1 \leq x_i, y_i \leq n$, for $1 \leq i \leq k$, where n is the size of the grid)

`e` x_1 y_1 **...** x_k y_k − make positions $(x_1, y_1), ..., (x_k, y_k)$ empty.
($x_i, y_i :: Int$ and $1 \leq x_i, y_i \leq n$, for $1 \leq i \leq k$, where n is the size of the grid)

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

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

`?` − show the rules of the current game, i.e. the pairs `s` x_0 y_0 and `b` x_1 y_1. Pick an appropriate place to show these, so that they do not overwrite the grid.

`w` − show the user the positions of all living cells currently on the board. That is, pick an appropriate place, not overwriting the board, and show the positions as pairs of coordinates, each separated by comma inside parentheses, e.g. $(3, 2)$, $(5, 5)$, $(1, 4)$, ...

`r` *name* − read the rules and board from the file *name* :: *String*. See section 2.3 for more details about the format for the input.

`'Enter'` − i.e., pressing Enter/Return button with empty input, progresses one generation. It tells 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 interaction, showing the changes on the board, for x generations OR until a stable configuration is reached before the xth generation, which is then announced to the user. It is important that the changes on the board are visible, i.e. do not goo too fast nor too slow.
($x :: Int$ and $x \geq 1$)

`quit` − quits the program.

2.2 Visualising the board and prompts

- 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 shown in Figure 1. That is, empty cells are marked with dots "`.`" and living cells are marked with `O`'s. Columns and rows are numbered explicitly as shown, in the top row and leftmost column, respectively. Ensure proper alignment of rows and columns, especially past the 9th row/column. You can assume that the grid never has more than 99 rows/columns.

- As illustrated by the examples, the grid is numbered in such a way that $(x, y) = (1, 1)$ is the cell in the top left corner. The first coordinate x represents the row and the second coordinate y represents the column of any given cell (x, y). On the display, it is allowed to number columns (but not rows) `'mod 10'`, so that each number has only one digit.

- After initialising, the position of the board must remain fixed. Any changes – due to `e` or `n` command, as well as when running the game with cells dying and being born – must occur on the same board, and not by printing out a new board which "pushes away" the previous one.

- Similarly, all prompts and messages to the user must be printed without interfering with or changing the board. You are free to choose the format for the messages yourself based on the specification given in the previous section.
- Invalid inputs for the commands are to be expected. The interactive program must continue after executing commands with such input and provide an error message to the user.

2.3 Reading files and file format

As for reading from a file, the following format must be used: The file only contains a single line with characters and numbers separated by whitespace (also a single comma and a pair of parentheses).

1. The first token n is a positive natural number which specifies the size of the grid, i.e. the number of rows/columns.
2. Following are two pairs of numbers, one preceded by s and other by b, specifying the rules for the game, surrounded by parentheses and separated by comma.
3. After that, there follow pairs of numbers, separated by blanks, which specify the living cells on the board: the pair x y represents a living cell at row x and column y.

As an example, Conway's Game of Life on the left board in Figure 4 could be written in a file as: 5 (s 2 3, b 3 3) 3 3 1 3 2 1 3 2 2 3.

The order of the two rules s and b does not matter. Similarly, the order of the pairs denoting the living cells can vary. That is, your implementation must be able to read also the following line 5 (b 3 3, s 2 3) 2 1 2 3 3 3 3 2 1 3, yielding the same result (automaton and state) as in the previous example.

2.4 Specify a glider automaton

The left board in Figure 4 displays a so-called *glider* in Conway's Game of Life, which is a pattern that, after a few moves, will glide diagonally into the pattern on the right board, and then will continue such gliding.

```
   1 2 3 4 5          1 2 3 4 5
1  . . O . .       1  . . . . .
2  O . O . .       2  . . . O .
3  . O O . .       3  . O . O .
4  . . . . .       4  . . O O .
5  . . . . .       5  . . . . .
```

Figure 4: Glider in Conway's Game of Life

Your task here is to specify a vertical glider for the specific set of rules (`s` 0 0, `b` 2 2). That is, provide a pattern P of living cells on some board such that running this automaton on P will cause the shape to glide *vertically*, either upwards or downwards.

Format this automaton as a file-text using the guidelines from Section 2.3, placing it after the initial "10 (s 0 0, b 2 2)", and include it as a comment at the top of the file, just below your name and group. (The grid of size 10 should be more than enough but if your pattern requires larger grid, you may increase the leading 10.)