Program Design & Data Structures (Course 1DL201)
Uppsala University Autumn 2016/Spring 2017
Makeup Homework Assignment: Pinball.

Prepared by Dave Clarke

Submission Deadline **18:00:00, Sunday 19 March, 2017**

This assignment is a makeup home assignment and may be attempted by any student who has failed some prior home assignment. Successful completion of this assignment can replace a previously obtained U with a 3.

Only one make up assignment is offered. There will be no lab and no supplementation.

Pinball

The goal of this assignment is to write functions to calculate the trajectory of a pinball through a small grid that contains "bumpers" that divert the path of the ball by 90 degrees. An example is given in the Figure 1. The orange dots indicate the path of the pinball, from where it enters until where it leaves. The purple and green slashes indicate the positions of the bumpers.

Your program will be provided with a description of a grid, such as the one in the figure, and the position from which the pinball enters the grid (IN). Your program will compute the positions in the grid the pinball passes through (its trajectory) *and* the position where the pinball leaves the grid (OUT).

The following data types are provided for your program.

Firstly, the grid is represented as a list of lists of cells, where each cell is either empty or a bumper leaning left or right. The representation convention is that elements of type `grid` correspond to a list of the rows of the grid. The first element of the first list corresponds to the top left position of the grid. The representation invariant is that each row has the same number of elements.

```
-- Left =  \, Right = /, Empty is empty cell
data Cell = Empty | Left | Right
type Grid = [[Cell]]
```

For example, the grid in Figure 1 is represented as the following value.

```
grid5x5 :: Grid
grid5x5 =
    [[Empty, Left,  Empty, Empty, Right],
     [Left,  Right, Empty, Empty, Left ],
     [Empty, Empty, Right, Empty, Right],
     [Right, Empty, Empty, Empty, Empty],
     [Empty, Empty, Empty, Left,  Empty]]
```

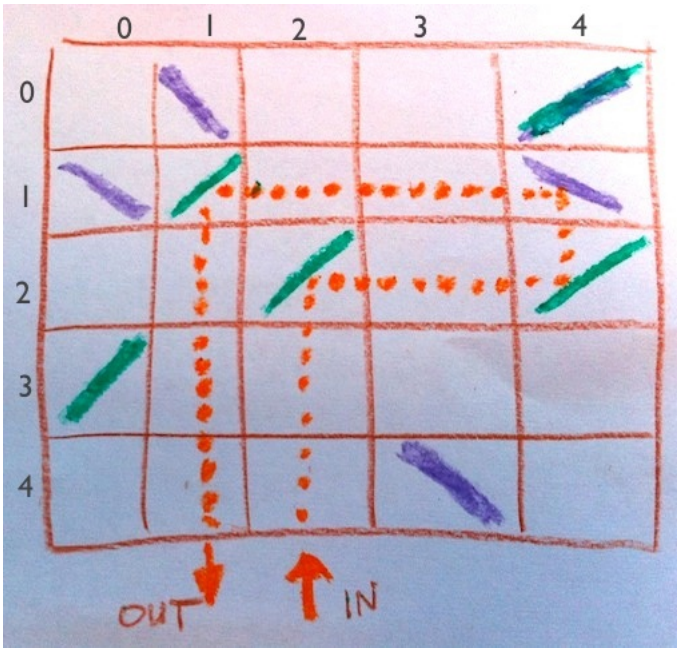

Figure 1: Trajectory of the pinball

The positions on the border of the grid are represented using a direction and an integer, which together represent the position along an edge. Directions are given by the following data type:

```
data Direction = North | South | East | West
```

`North` is the top of grid, `South` is the bottom, `East` is the right edge, and `West` is the left edge. Positions on the border are given by the following data type.

```
type Border = (Direction, Int)
```

The numbering of the border elements starts from the top left corner, starting from 0, as indicated in the figure. For instance, the IN position is given by value `(South, 2)` and the OUT position is given by `(South, 1)`.

Finally, paths are given by the following data type:

```
type Path = [(Int, Int)]
```

Each pair `(x,y)` in the list records the x- and y-coordinates of a grid square, according to the numbering scheme in the figure. For example, the following value corresponds to the trajectory of the pinball through the example grid, for the given IN position:

```
traj :: Path
traj = [(2, 4), (2, 3), (2, 2), (3, 2), (4, 2), (4, 1),
        (3, 1), (2, 1), (1, 1), (1, 2), (1, 3), (1, 4)]
```

Work to be done

1. Implement the following functions in the file `pinball.hs`, which is available on the student portal, making sure that they pass the training test case provided in that file.

   - Function `validGrid :: Grid -> Bool` takes an element of the `grid` type and determines whether it satisfies the representation invariant, namely, that all rows have the same length.
   - Function `validEntryPoint :: Grid -> Border -> Bool` takes a grid and one of the potential border positions and determines whether the border position is a valid entry point for the grid. For example, position `(North,12)` is not a valid entry point, because it falls outside the boundary.
   - Function `trajectory :: Grid -> Border -> Path`, given a valid grid and entry point (IN) where the pinball is rolled into the grid, calculates the grid cells visited by the pinball up until it exits the grid (OUT). The head of the list contains the first cell crossed, and the last element contains the final cell crossed.
   - Function `play :: Grid -> Border -> Border`, given a valid grid and valid entry point (IN) where the pinball is rolled into the grid, calculates the position where the ball will exit the grid (OUT).

2. Ensure that your code follows the coding convention.

3. Write an additional 3 test cases for each of the four functions, also in the file `pinball.hs`. Insert these tests into the list found at the end of the function `extraTests` at the bottom of `pinball.hs`:

   ```
   extraTests = runTestTT $ TestList $ [ YOUR TESTS HERE ]
   ```

Grading

Your solution will be given a U or a 3 based on the following criteria:

1. Your solution was submitted by the deadline, your program loads under the version of Haskell found on the student machines and it passes all of the training test cases provided.

   Hint: Test your submitted code in a freshly started `ghci` interpreter in order to ensure your code does not work because of some old data or functions lying around in the interpreter.

2. Your program will be run on an unspecified number of orthogonal grading test cases, which cover both valid and invalid problem and solution combinations. We reserve the right to run these tests automatically, so be careful to match exactly the imposed file names, function names, function types and argument orders.

3. Your program will be graded for style and comments. The following criteria will be used:

   - suitable breakdown of solution into auxiliary/helper functions
   - comments describing the purpose of each function, following the coding convention, including function specifications, and representation conventions and invariants

- code readability.

4. The reasonableness and coverage of your test cases.

Modalities

The assignment will be conducted in groups of 1 or 2. You can join/form a group via the student portal. Only one solution per group needs to be submitted. Be aware that there may not be enough students to form groups of 2, and you might need to work alone.

We assume that by submitting a solution you are certifying that it is solely the work of your group, except where explicitly attributed otherwise. We reserve the right to use plagiarism detection tools and point out that they are extremely powerful.