Middle East Technical University Department of Computer Engineering

CENG 242

Programming Language Concepts

Spring '2016-2017

Homework 1

Due date: 26 March 2017, Sunday, 23:55

1 Objectives

This assignment aims to assist you to discover the functional programming world by the help of Haskell programming language.

2 Problem Definition

In this assignment, you will be simulating a bacterial environment where the environment is represented by a 2D grid and each cell in this grid is either a dead or an alive bacteria.

You can see an example of a representation of this environment in the Figure 1:

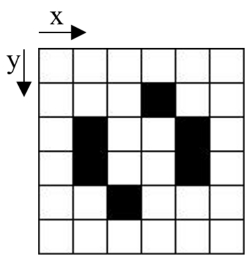

Figure 1: An example of 2D bacterial environment

In this representation;

- Black cells show alive bacterias while white cells show dead ones.
- The coordinate of the top left corner is (0, 0). Therefore we can say that bacterias in (3,1), (1,2), (4,2), (1,3), (4,3) and (2,4) are alive in this environment.
- A bacteria is called as *adjoint* of another bacteria if they are vertically, horizontally or diagonally adjacent.

There are four rules in this environment which you will be using to determine the status of each bacteria for the next time step:

1. If the number of alive adjoints of an alive bacteria is *two* or *three* then it **stays alive**.
2. If the number of alive adjoints of an alive bacteria exceeds *three* then it **dies**.
3. If the number of alive adjoints of an alive bacteria is under *two* then it **dies**.
4. A dead bacteria **becomes alive** again, if it has *exactly three* alive adjoints.

The environment becomes as Figure 2 in the next time step according to the rules above.

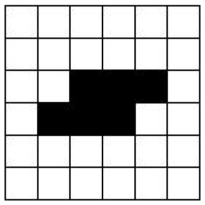
Figure 2: The new environment after one time step

If we take the simulation one step further, we get the environment as Figure 3:

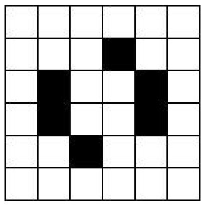
Figure 3: The new environment after two time steps

As you can observe the environment oscillates between two states for the given initial configuration.

3 Specifications

You are expected to write a Haskell function called "next" which takes a nested list of string representing the state of a 2D bacterial environment and returns a nested list of string representing the same environment after one time step.

Type declaration of this function is given below.

```
next :: [[String]] -> [[String]]
```

In the nested list of string two strings will appear to represent cells containing alive or dead bacterias. The string "B" will show the cell that contains alive bacteria while "W" will show the cell that contains dead bacteria.

Be aware that the width and the height of the environment may differ. In other words, the lengths of outer and inner list may be different.

Here is the representation of environment from Figure 1 in the nested list form:

```
[["W","W","W","W","W","W"],["W","W","W","B","W","W"],["W","B","W","W","B","W"],["W"
    ,"B","W","W","B","W"],["W","W","B","W","W","W"],["W","W","W","W","W","W"]]
```

The *next* function must be defined in the file **the1.hs** which you can find in homework files. After the implementation of this function you can load your module to ghci by typing to terminal:

```
$ ghci the1.hs
```

For moodle system running the file will load the file automatically to the ghci.

Then you can call the function *next* as:

```
*Main> next [["W","W","W","W","W","W"],["W","W","W","B","W","W"],["W","B","W","W","
    B","W"],["W","B","W","W","B","W"],["W","W","B","W","W","W"],["W","W","W","W","W"
    ,"W"]]
```

Then you should get the following results:

```
[["W","W","W","W","W","W"],["W","W","W","W","W","W"],["W","W","B","B","B","W"],["W"
    ,"B","B","B","W","W"],["W","W","W","W","W","W"],["W","W","W","W","W","W"]]
```

If you are working on your own terminal, you can use the function *writeSteps* which is provided in **the1.hs**, to write the simulation steps to a file for an environment.

```
*Main> writeSteps "test.out" [["W","W","W","W","W","W"],["W","W","W","B","W","W"],[
    "W","B","W","W","B","W"],["W","B","W","W","B","W"],["W","W","B","W","W","W"],["W
    ","W","W","W","W","W"]] 100
```

Then you should get the following output and the file *test.out* containing the steps of the simulation must have been created:

```
100 steps are succesfully written to 'test.out'
```

4 Grading & Simulation

Your code will be graded **only** according to the function *next* that you will implement. Your implementation must obey the **type declaration** stated in the *the1.hs* file.

You can create an animated gif of the simulation from the output of *writeSteps* function by using the file **simulator.py** which is provided in the homework files. Assuming that the output file created by *writeSteps* is called *test.out*, an example of its execution would be like:

```
$ ./simulator.py test.out env1
```

Then *env1.gif* must have been created in the same folder.

This is **neither** an obligation **nor** will be graded but it may provide you a better understanding the trace of simulation. You may even discover some "cool" patterns by yourself.

The following libraries must be installed in your system to run the **simulator.py** successfully:

- python-opencv
- imagemagick

They can be installed with the commands below:

```
$ sudo apt-get install python-opencv
$ sudo apt-get install imagemagick
```

5 Regulations

1. **Programming Language:** You must code your program in Haskell. Your submission will be tested with `ghci` interpreter on moodle system. You are expected make sure your code loads successfully in `ghci` interpreter on the moodle system.
2. **Late Submission:** You have been given 10 late days in total and at most 3 of them can be used for an assignment. After 3 days you get 0, no excuse will be accepted.
3. **Cheating: We have zero tolerance policy for cheating**. People involved in cheating will be punished according to the university regulations.
4. **Newsgroup:** You must follow the newsgroup (news.ceng.metu.edu.tr) for discussions and possible updates on a daily basis.
5. **Evaluation:** Your program will be evaluated automatically using "black-box" technique so make sure to obey the specifications.

6 Submission

Submission will be done via moodle system. You can either download the template file, make necessary additions and upload the file to the system or edit using the editor on the moodle and save your changes.