German University in Cairo
Media Engineering and Technology
Prof. Dr. Slim Abdennadher

Concepts of Programming Languages, Spring term 2016
Project Description
"Furnishing a Room"

Deadline: 24.04.2016 - 11:59 pm

The project will put your knowledge in Haskell to the test. Before proceeding, make sure that you read each section carefully. Enjoy.

Instructions

Please read and follow the instructions carefully:

a) The project should be implemented using Haskell (Hugs98) based on the syntax discussed in class.

b) This is a team project. You can work in groups of maximum 3 members. The groups for this project are the same as project 1. In case of any changes, please report it by sending an email to your TA arwa.sayed-ismail@guc.edu.eg or nada.hamed@guc.edu.eg. You must send complete information about your new team including full names and unique GUC application numbers. The deadline for sending group change requests is Wednesday 13.04.2016 (11:59 pm).

c) You have to write a formal report on your project covering the following points:

- A brief formal description of the datatypes that you have created.
- A brief formal description of the *main* functions used in your implementation.
- A listing of the *helper* functions that you called from your main functions. You are allowed to use built-in Haskell functions, like `length` or other functions in `Hugs.Prelude` without explanation: https://www.haskell.org/onlinereport/prelude-index.html

d) Every team has to submit a soft copy of the following files (in a zipped directory) on MET Website's submission link:

- your Haskell project **source file** named after your team code, e.g. `T03.hs`
- the report file (.doc or .pdf) also named after your team code , e.g. `T03.pdf`
- the zipped file should also be named after your team code, e.g. `T03.zip`
- it is **your** responsibility to ensure that you have submitted the correct files and saved the correct content before uploading.

e) Every team has to submit a **proper** hard copy of the following:

- the report
- a printout of the source file

f) You are always welcome to discuss the project with the TAs during the available office hours. You must work with your team members only. Do not exchange information with other teams or individuals.

g) Cheating and plagiarism will be strictly punished with a grade of 0 in the project.

h) Please respect the submission deadline marked at the beginning of the document (**Sunday 24.04.2016 - 11:59 pm**). Any delay will result in a **rejection** of the submission.

Project Description

You are required to implement a generic furnishing algorithm. It should be able to learn some statistics from a given set of furnished rooms containing different types of furniture, and generate a room that is furnished based on the learnt statistics. For this project, we will use a grid representation of an `NxN` room.

For simplicitly, we have chosen a small set of furniture that should be handled for this project: `"couch"`, `"lamp"`, `"tv"`, and `"table"`. Thus, a room can contain, and can be filled with, any of the previously mentioned furniture. We add one more object `"e"` to the set of objects which denotes that a spot (or a cell in the grid) is `empty` in the room. The file FurnitureResources.hs will contain the set of furniture objects. Your code should use the list `training` from that file in order to learn. Make sure you have that file in the same folder as your solution `.hs` file. Then, you can write in your file:

```
import FurnitureResources
```

In order to implement the furnishing generator, there are two phases; **training** and **generation**.

Training phase

For the training phase you are required to:

- Implement the function `statsList` which will dynamically generate a list of statistics according to the content of the training list in the `FurnitureResources.hs` file. The type of `statsList` will be:

  ```
  statsList :: [([Char],[[([Char],[Char],Int)]])]
  ```

  The size of the output list will be 5 items since we have 5 different furniture objects to gather statistics about. Every entry in that list is a pair that contains the following **respectively**:

  a) An object `X`

  b) A list of lists representing statistical information about the object `X`. The list consists of exactly two inner lists respectively:

  - The first list contains information about objects that have appeared to the **right** of object `X` along with their frequency of appearance to the right of `X`. The list must be **sorted** from the highest to the lowest frequency value.
  - The first list contains information about objects that have appeared **below** the object `X` along with their frequency of appearance below of `X`.The list must be **sorted** from the highest to the lowest frequency value.

  Note that the statistical information only cares about objects that are directly **side-by-side** with `X`

- Implement the function `generate`:

  ```
  generate :: [[[Char]]] -> [([Char],[([Char],[Char],Int)])]
  ->[([Char],[([Char],[Char],Int)])]
  ```

  The input to this function is a room and the current list of furniture statistics. The function returns an updated list of statistics based on examining the furniture in the input room. Your implementation should be generic enough to handle any `NxN` room. The output must follow and preserve the format of the statistics list.

- Implement the function `findFurnitureUpdate`:

```
findFurnitureUpdate :: [Char] -> [Char] -> [Char] -> [([Char],[([Char],[Char],Int)])]
-> [([Char],[([Char],[Char],Int)])]
```

For the function `findFurnitureUpdate a b c currentStats`, the following applies for its four inputs:

- `c` is the position where the furniture `b` was found with respect to the examined furniture `a`. The position can either be `"right"` or `"below"`.
- `currentStats` is the list of statistics compiled so far for the explored rooms of the training set.

The function outputs the updated list of statistics based on the input parameters. Again, the output must follow and preserve the format of the statistics list.

Generation phase

As for the generation phase you are required to:

- Implement the function `furnishRoom`:

```
furnishRoom :: Int -> [Char] -> [[[Char]]]
```

The input to this function will be an integer representing the *dimension* of the room (to generate an `NxN` room), and a string representing the *starting furniture* object that will be placed at the **top left corner** of the room. The function `furnishRoom` must call `statsList` to use the information gathered about the furniture to generate the room.

In general when filling a cell in the room with a piece of furniture, a random furniture object is chosen from the set of possible values according to our statistics. Each cell of the room is filled with an object according to the furniture on its left (if applicable) and/or the furniture above it (if applicable).

- Implement the function `getFurnStat`:

```
getFurnStat :: [Char] -> [[([Char],[Char],Int)]]
```

The input to this function is a string representing the name of a furniture object. The function should search for the input furniture through the result of `statsList`, and output the corresponding list of statistics of that object. Accordingly, the output list is a list of, specifically two, lists.

- Implement the function `getPossibleNeighbour`

```
getPossibleNeighbour :: [[([Char],[Char],Int)]] -> [[([Char],[Char],Int)]] -> [Char]
```

The function outputs the name of the furniture object to be used to fill a certain cell. The input of the function are two lists representing the list of statistics of the furniture object on the left of the cell and the list of statistics of the object on top of it, respectively. Take into consideration that when a furniture object is being randomly chosen as a possible neighbour, the **weights** of the frequency of each option have to be taken into consideration.

Hint: You can use the function `randomZeroToX` in your solution which you will find in `FurnitureResources.hs`.

Example

Assume we have the following room setting and we would like to fill the cell in row 1, column 1:

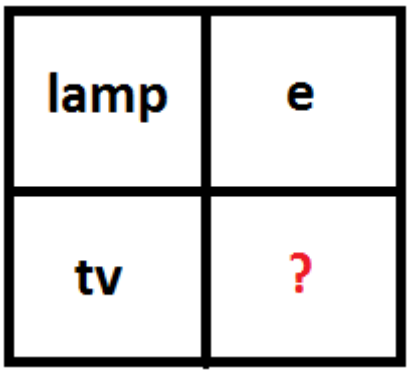

Figure 1: A 2X2 room with a spot to be filled.

We would need to pick a random object that either exists in the set of possible objects that could be found below `e` (row 0 column 1) or in the set of possible objects that could be found to the right of `tv` (row 1 column 0).

Thus, in order to fill cell (1,1), we take into consideration the following:

a) the statistics pair of `"tv"` which can be retrieved using the function `getFurnStat`. Assume that the result of the function is the following:
`("tv",[[("table","right",3),("e","right",2)],[("tv","below",5)]])`

   we are only interested in the list containing the furniture objects that could appear to the **right** of `"tv"`.

b) the statistics pair of `"e"` which can be retrieved using the function `getFurnStat`. Assume that the result of the function is the following:
`("e",[[("couch","right",4)],[("e","below",3),("tv","below",2)]])`

   we are only interested in the list containing the furniture objects that could appear **below** `"e"`.

Using the above information, if we combine the two lists of interest, we get the below list which we can choose a random object from to fill cell (1,1):

```
[("table","right",3),("e","right",2),("e","below",3),("tv","below",2)]
```

Taking into consideration that the **weights** of the frequency of each option, the furniture chosen should be `"e"` with a **total** probability of 50%, `"table"` with a **total** probability of 30%, or `"tv"` with a **total** probability of 20%.

Sample Test

The file `statsListCall.txt` shows the output of a sample `statsList` that was generated with a value of the training list equal to:

```
training = [room1,room2,room3]
```