Functional Programming I (cs4620) Assignment 2

Recursion (Due: October 22. Marks: 5)

1 Introduction

For this assignment you will learn how to implement recursive functions in `Haskell`.

- The solution of this assignment requires a mutual-recursive solution technique, with recursive calls among two or several functions.
- To tackle the problem you should apply the five-steps identified by Hutton [2016, Section 6.6]:
  1. Define the type of the functions;
  2. Enumerate the cases;
  3. Define the simple cases;
  4. Define the other cases;
  5. Generalise and simplify.

Please remember that your programs should be properly commented. Also please note that all function definitions in your programs should include a proper type signature. Not only is adding them a proper form of documentation but it is also a good exercise.

2 Assignments Details

For this assignment you will implement a function called `paint_interior_bricks`. The function takes two arguments: `colours` and `walls`, where `colours` is a list of type `a` elements and `walls` is a list of list of type `a` elements. A member of the list `colours` is called a *colour*. A member of the list `walls` is called a *wall*. A wall consists of *brick* elements.

The purpose of the function is to replace, "*paint*," each *internal* brick of the walls in `walls` with some colour in `colours`. The colours in `colours` are used *cyclicly*, which means that the *i*th time you paint a brick in `walls` you should paint it with the *i*th (cyclic) colour in `colours`. The following are some examples.

```
*Main> paint_interior_bricks [0] [[3],[3,3],[3,3,3],[3,3,3],[3,3,3,3,3],[3,3,3,3,3,3]]
[[3],[3,3],[3,0,3],[3,0,3],[3,0,0,0,3],[3,0,0,0,0,3]]
*Main> paint_interior_bricks [0,1] [[3],[3,3],[3,3,3],[3,3,3],[3,3,3,3,3],[3,3,3,3,3,3]]
[[3],[3,3],[3,0,3],[3,1,3],[3,0,1,0,3],[3,1,0,1,0,3]]
*Main> paint_interior_bricks [0,1,2] [[3],[3,3],[3,3,3],[3,3,3],[3,3,3,3,3],[3,3,3,3,3,3]]
[[3],[3,3],[3,0,3],[3,1,3],[3,2,0,1,3],[3,2,0,1,2,3]]
*Main>
```

You should implement the function `paint_interior_bricks` using recursion, pattern matching, and the constructors `:` and `[]`. You are not allowed to use any other built-in or library functions.

For this assignment, all functions should be defined at the top level. All these functions should have a proper signature. For each function, please and add a short comment that explains the purpose of the function.

This assignment is not difficult. When I implemented it, it took me 13 lines, 4 of which were signatures. (I did not count comment lines.)

Hint: in order to use the colours cyclicly, some functions may require an additional argument that lets you "refill" your colours when you run out of colours.

Please remember that every `Haskell` program starts by calling the `main` function. Please insert the following code for the `main`.

```
main :: IO ()
main = putStrLn (show (paint_interior_bricks colours walls))
    where colours = [0,1,2]
          walls = [[3],[3,3],[3,3,3],[3,3,3],[3,3,3,3,3],[3,3,3,3,3,3]]
```

3 Submission Details

- Your program should start with a comment like the following:

```
{-
 - Name: Fill in your name.
 - Number: Fill in your student ID.
 - Assignment: 02.
 -}
```

- Use the CS4620 `moodle` site to upload your program as a single *.tgz* archive called *Lab-2.tgz* before 23.55p.m., October 22, 2017. To create the *.tgz* archive, do the following:
  - ⋆ Create a directory `Lab-2` in your working directory.
  - ⋆ Copy `Main.hs` (or `Main.lhs`) into the directory. Do not copy any other files into the directory.
  - ⋆ Run the command '`tar cvfz Lab-2.tgz Lab-2`' from your working directory. The option '`v`' makes `tar` very chatty: it should tell you exactly what is going into the `.tgz` archive. Make sure you check the `tar` command's output before submitting your archive; alternatively, use `tar -t` or `tar --list`.
  - ⋆ Notice that file names in `Unix` are case sensitive and should not contain spaces.
- Notice that the format is `.tgz`: do *not* submit `zip` files, do *not* submit `tar` files, do *not* submit `bzip` files, and do *not* submit `rar` files. If you do, it may not be possible to unzip your assignment.
- Marks are deducted for poor choice of identifier names and/or poor layout.
- Please should make sure your assignment submission has a `Main` class with a `main` in it. The `main` should be the main thread of execution of the program.
- No marks shall be awarded for scripts that do not compile.

References

Hutton, Graham [2016]. *Programming in `Haskell`*. Second. Cambridge University Press. ISBN: 978-1-316-62221-1.