**IN3043 Functional Programming - Coursework**

There is a single coursework in this module, counting for 30% of the overall module mark. This coursework is due at 5pm on Sunday 24th November. As with all modules, this deadline is hard, and extensions may only be requested via the standard Extenuating Circumstances procedure.

This coursework assesses the Skills objectives in the module description, specifically:

* to practice using functions from a supplied library,
* to use a combination of library functions and function definitions to achieve interesting effects with compact code.

This task does not require any material presented after week 7, and I do not expect to see it in your solutions.

**The task**

I have supplied an Animation module (discussed in the week 7 lecture), that defines functions that may be used to describe simple animations. You should not modify (or submit) this module. Study the documentation for this module, and the small examples provided there and in lecture 7. You are to create a Haskell module using this module and containing a definition of an animation of your own design, fitting within a rectangle 800 units wide and 600 high:

1. The submission must not be a static picture; i.e. it must vary over time.
2. The picture should feature at least one repetitive element, e.g.
   * a collection of similar shapes in different sizes, positions and/or orientations,
   * a polygon with many vertices,
   * a succession of similar shapes over time.
3. This repetition should be coded concisely using at least one of the following features of Haskell covered up to week 5: list comprehensions and higher-order functions from the Data.List library. (See exercises 2 and 4 of week 7 for examples.)

The picture may contain other elements, but only the repetitive element will count towards the marks. The aim is to use the features of Haskell to create an interesting effect with the minimum of repetitive code.

You may test your pictures by converting them to Scalable Vector Graphics file with lines like

writeFile "test.svg" (svg 800 600 picture)

which writes the generated string to a file test.svg that you can then view with an appropriate SVG viewer. For example, most modern web browsers can display SVG files. (writeFile can be used at the GHCi command line but not in your functions; at least not yet -- we'll be covering it in week 11.)

I will be expecting substantially different solutions from each student, and also solutions substantially different from the examples I have given.

**Submission format**

You are to create a file MyAnimation.hs, of the form

module MyAnimation where

import Animation

... other imports ...

picture :: Animation

picture = ...

... auxiliary definitions for picture (if any) ...

Please adhere to the above format (including names) precisely, so that I can run your code automatically. Submit only MyAnimation.hs, via Moodle. Do not submit your test module: your program must work with mine (which will be similar to the snippet above).

**Marking scheme**

Your image will be marked out of 100 according to the following scheme:

|  |  |
| --- | --- |
| 0-9 | The submission is not legal Haskell. |
| 10-19 | The submission makes very basic use of some functions from the library to produce a picture, but does not use the above-mentioned features to avoid repetition of code, or is a minor variation on an example I have given you. |
| 20-29 | The submission makes more varied use of functions from the library to produce a picture, but does not use language features to avoid repetition of code. For example, the output picture contains 6 shapes, and the source code contains 6 function calls to produce those shapes. Such solutions do not meet the second and third requirements above, and will not pass, no matter how complex the picture. |
| 30-39 | The submission uses functions of the library and attempts to use language features, but uses them incorrectly and ineffectively. |
| 40-49 | The submission uses functions of the library and attempts to use language features, but the attempt is only partially successful. |
| 50-59 | Library functions and the language features are used correctly to produce a repetitive element without corresponding repetition in the code. |
| 60-69 | Library functions and the required feature are used appropriately and effectively to produce a repetitive element. The code is readable, through use of appropriate names, layout and moderate commenting where required. |
| 70-79 | Use of library functions and the required feature to produce a repetitive element demonstrates a thorough understanding of the possibilities of both. As well as basic readability, the code has a clear structure. |
| 80-100 | As well as the above, the submission demonstrates a particularly imaginative use of the required feature. The criterion is not how much you draw, but the extent to which you obtain interesting effects from a minimum of code, and the elegance of that code. |

For example, all of the examples in the Week 7 gallery, if cleanly and economically coded, would be sufficient for a mark of 70 or more.

**Pair working**

If you wish, you may work on this coursework in pairs. The submitted source file should contain the names and IDs of both authors.

**Help and feedback**

Please ask general questions about the coursework on the Moodle discussion board. Queries by email will be answered on that board, so that everyone gets the same information.

I can view and discuss draft solutions in the weekly lab session. You can also come along to my office (A309D) during my Drop-in Office Hours, or make an appointment (by email to R.Paterson@city.ac.uk) to see me at other times.

Sample solutions will be published on Moodle after the deadline. Marks and comments on your code will be returned via Moodle.

**Advice**

* To do this coursework, you don't need to understand the SVG format or the implementation of the Animation module. You do however need to know what the functions in that module do, and how to use them.
* Instead of starting with a particular target picture in mind, start by exploring what you can do by combining the functions of this module with other library functions and general Haskell features. You might do this by taking the examples given in the documentation, figuring what they do and experimenting on them.
* I would recommend an abstract geometric animation rather than a pictorial one.
* You have over two weeks to do this coursework, but it doesn't require any material from after week 7. So after you've done the exercises up to week 7, you'll be able to start it, and indeed this would be a good idea.