CS_191 Functional Programming I

Coursework 2

Due date: Thursday, 5 May 2011.
Assessment: Only Questions 4, 5, 6, 7, 8 are assessed.
                 Solutions to Questions 1, 2, 3 are available on the course web page.
Submission: Please follow the instructions on the last page.

In this coursework we practice programming with lists and higher-order functions. The techniques are illustrated by programs for the graphical display of 2-dimensional regions.

We represent a region as a *bitmap*, that is, a function $f \colon \mathbf{R} \times \mathbf{R} \to \mathbf{B}$ where $\mathbf{R}$ is the set of real numbers and $\mathbf{B} = \{\mathbf{T}, \mathbf{F}\}$ is the set of Booleans: $f(x,y) = \mathbf{T}$ means the point (x,y) is in the region represented by f, $f(x,y) = \mathbf{F}$ means it is outside. Therefore, f represents the region $\{(x,y) \mid f(x,y) = \mathbf{T}\}$.


```
type Point  = (Float,Float)
type Bitmap = Point -> Bool
```


For example, the function $f(x,y) := x^2 + y^2 \le 1$, that is, $f(x,y) = \mathbf{T}$ if $x^2 + y^2 \le 1$ and $f(x,y) = \mathbf{F}$ otherwise, represents the region $\{(x,y) \mid x^2 + y^2 \le 1\}$ that is, the set of points that lie on a disc with radius 1 around the origin, $(0,0)$.


```
discBm :: Bitmap
discBm (x,y) = x^2 + y^2 <= 1
```


A *picture* is a list of lists of Booleans representing a region restricted to a rectangular segment of the plane.


```
type Picture = [[Bool]]
```


Example:


```
discPic :: Picture
discPic =
     [[False,False,False,False,False,True, False,False,False,False,False],
      [False,False,True, True, True, True, True, True, True, False,False],
      [False,True, True, True, True, True, True, True, True, True, False],
      [False,True, True, True, True, True, True, True, True, True, False],
      [True, True, True, True, True, True, True, True, True, True, True],
      [False,True, True, True, True, True, True, True, True, True, False],
      [False,True, True, True, True, True, True, True, True, True, False],
      [False,False,True, True, True, True, True, True, True, False,False],
      [False,False,False,False,False,True, False,False,False,False,False]]
```

The picture `discPic` has been obtained by sampling the bitmap `discBm` on a 11×9 grid in the square $\{(x,y) \mid -1 \leq x, y \leq 1\} \subseteq \mathbf{R} \times \mathbf{R}$. More precisely, the sampling points are of the form $(-1 + 0.2 * j, 1 - 0.25 * i)$ where $j \in \{0,1,2,3,4,5,6,7,8,9,\mathbf{10}\}$ and $i \in \{0,1,2,3,4,5,6,7,\mathbf{8}\}$.

In general, the following additional data are required to compute a picture from a bitmap:

- The *frame*: A rectangle, given by its bottom left and top right corner (in our example $(-1,-1)$ and $(1,1)$).

- The *resolution*: Two positive integers, n, m, determining the grid points (in our example 10 and 8). More precisely, $n + 1$ is the number of grid points in x-direction and $m + 1$ is the number of grid points in y-direction, so there are $(n + 1) * (m + 1)$ grid points in total.

We call these data (the frame together with the resolution) a *setting*:

```
type Setting = (Point,Point,Int,Int)
```

An example of a setting is

```
set0 :: Setting
set0 = ((-1,-1),(1,1),10,8)
```

Question 1. Define a function `takePicture ::  Setting -> Bitmap -> Picture` that computes a picture from a given setting and a bitmap.

If Question 1 has been solved correctly, the expressions `discPic` and `takePicture set0 discBm` should evaluate to the same result.

A simple way of displaying a picture is to transform it into a string and then display the string on the terminal using `putStrLn`. A picture

```
[bs1,bs2,...,bsm]
```

is transformed into the string

```
('\n':s1) ++ ('\n':s2) ++ ... ++ ('\n':sm)
```

where `'\n'` is the *newline character* and each string si is obtained from the Boolean list bs$i$ by replacing `True` and `False` by two characters representing the colours black and white, say `'@'` and `'.'`. For example, the picture

```
[[True,True,True],[True,True,False],[True,False,False]]
```

is transformed into the string `"\n@@@\n@@.\n@.."`.

Question 2. Define a function `showPicture ::  Picture -> String` that computes a string from a picture as described above.

Question 3. Define a function `pp ::  Picture -> IO ()` ("print picture") that prints the string corresponding to a picture on the terminal.

It is convenient to also define functions printing bitmaps directly:

```
pb :: Setting -> Bitmap -> IO ()
pb set bm = pp (takePicture set bm)
```

If Questions 2 and 3 have been solved correctly, evaluation of the expressions `pp discPic` and `pb set0 discBm` should have the same effect, namely of printing the following on the terminal:

```
.....@.....
..@@@@@@@..
.@@@@@@@@@.
.@@@@@@@@@.
@@@@@@@@@@@
.@@@@@@@@@.
.@@@@@@@@@.
..@@@@@@@..
.....@.....
```

Question 4. Define the following region (= bitmap) producing functions:

(a) `disc :: Point -> Float -> Bitmap` computing for every point p and positive real number r a disc with center p and radius r.

(b) `tri :: Point -> Float -> Bitmap` computing for every point $p = (x, y)$ and positive real number d a rectangular triangle with the vertices $(x + d, y)$, $(x, y + d)$, $(x + d, y + d)$.

Using the setting `set1 = ((-1.5,-1.5),(1.5,1.5),80,40)` take pictures of the regions `disc (0,0.8) 0.5` and `tri (0,0) 0.7`.

Use a small font for displaying the results. Depending on the monitor you are working with you might wish to change the numbers 80 and 40 to a different resolution. [**10 marks**]

Question 5. Define the following operations on pictures:

(a) `appvPic`: appending two pictures vertically (second below first).

(b) `apphPic`: appending two pictures horizontally (side by side).

(c) `udPic`: flipping a picture upside down.

(d) `lrPic`: flipping left and right of a picture.

(e) `invPic`: inverting a picture (complement, that is, pointwise negation).

(f) `unionPic`, `interPic`, `exclPic`: Computing the union, intersection and symmetric difference of two pictures, that is, computing pointwise (`||`), pointwise (`&&`) and pointwise (`/=`).

In questions (a), (b), and (f) it may be assumed that the given pictures are of the same size (but this is not relevant for the implementation).

Test the operations with the examples from question 4 (using a different setting if necessary).
[**60 marks**]

Question 6. Define the following operations on regions (represented as bitmaps):

(a) `invBm`: inverting a region (complement, that is, pointwise negation).

(b) `unionBm`, `interBm`, `exclBm`: Computing the union, intersection and symmetric difference of two regions,

Test the operations with the examples from question 4 (using a different setting if necessary).
[10 marks]

Question 7. Define a function `unionsBm` that computes the union of a list of regions (represented as bitmaps). **[10 marks]**

Question 8. The *Sierpinski triangle* is a fractal composed from rectangular triangles. It is defined recursively as follows: Let a size d (a real number) and a point $p = (x, y)$ be given. Define:

$$
\begin{aligned}
d' &= d/2 \\
p_1 &= (x + d', y) \\
p_2 &= (x, y + d') \\
p_3 &= (x + d', y + d')
\end{aligned}
$$

The *Sierpinski triangle of size d at point p* consists of the rectangular triangle with vertices p_1, p_2, p_3 together with the three Sierpinski triangles of size d' at the points p, p_1, p_2 (draw a picture to see what's going on).

Slightly different (and more general) forms of the Sierpinski triangle are described at
http://en.wikipedia.org/wiki/Sierpinski_triangle

Define a function `sierpinski :: Float -> Float -> Point -> Region` such that the region `sierpinski s d p` is an approximation of the Sierpinski triangle of size `d` at point `p` where the recursion stops when `d < s`.

Take a picture of a Sierpinski triangle.

[10 marks]

Question 9. The *logistic map* with parameter $r \in [0, 4]$

$$\varphi_r : [0, 1] \to [0, 1], \quad \varphi_r(x) = r * x * (1 - x)$$

$([a, b] = \{x \in \mathbf{R} \mid a \leq x \leq b\})$ is a simple example of a dynamical system with chaotic behaviour. It is used as a demographic model and as a testing function for implementations of real number operations (`http://en.wikipedia.org/wiki/Logistic_map`).

The chaotic behaviour of the logistic map is due to the fact that for many parameter values r close to 4 and many starting points $x \in [0, 1]$ the orbit of the logistic map,

$$\{\varphi_r^n(x) \mid n = 1, 2, 3, \ldots\} = \{\varphi_r(x), \varphi_r(\varphi_r(x)), \varphi_r(\varphi_r(\varphi_r(x))), \ldots\}$$

is a highly irregular set the elements of which (for large n) are difficult to compute because of the accumulation of rounding errors.

In the following we only consider the parameter $r = 4$ and set $\varphi := \varphi_4$.

Take pictures of the iterations, $\varphi^n \colon [0, 1] \to [0, 1]$, of the logistic map for $n = 1, \ldots, 10$, that is, take pictures of the regions

$$\{(x, y) \mid x, y \in [0, 1], y < \varphi^n(x)\} \qquad (n = 1, \ldots, 10)$$

Question 10. The *Mandelbrot set* (`http://en.wikipedia.org/wiki/Mandelbrot_set`) is a fractal which is defined as the set of all complex numbers $c \in \mathbf{C}$ with the property that the mapping

$$f_c : \mathbf{C} \to \mathbf{C}, \quad f_c(z) = z^2 + c$$

has a bounded orbit at 0, that is, the set

$$\begin{aligned}
\{|f_c^n(0)| \mid n = 1, 2, 3, \ldots\} &= \{|f_c(0)|, |f_c(f_c(0))|, |f_c(f_c(f_c(0)))|, \ldots\} \\
&= \{|c|, |c^2 + c|, |(c^2 + c)^2 + c|, \ldots\}
\end{aligned}$$

is bounded.

By identifying a complex number $c = x + \mathbf{i} * y \in \mathbf{C}$ with the point (x, y) we may view the Mandelbrot set as a region in the 2-dimensional plane.

Using the setting $((-2, -1.3), (1, 1.3), 160, 80)$ (maybe with a different resolution) take a picture of the following region that approximates the Mandelbrot set:

$$\{c \in \mathbf{C} \mid |f_c^n(0)| < 2 \text{ for all } n \in \{0, 1, 2, \ldots, 20\}\}$$

Reminder on how to calculate with complex numbers:

$$\begin{aligned}
(x + y * \mathbf{i}) + (x' + y' * \mathbf{i}) &= (x + x') + (y + y') * \mathbf{i} \\
(x + y * \mathbf{i}) * (x' + y' * \mathbf{i}) &= (x * x' - y * y') + (x * y' + y * x') * \mathbf{i} \\
|x + y * \mathbf{i}| &= \sqrt{x^2 + y^2}
\end{aligned}$$

Hence, $|x + y * \mathbf{i}| < 2$ if and only if $x^2 + y^2 < 4$.

Please see overleaf for instructions on how to do and submit your coursework.

Instructions for solving and submitting the coursework

1. It is recommended that coursework is done and submitted in pairs. Solutions by single students are allowed, but not by three or more students.

2. In the lab classes you will get help with the coursework. **Marks will be awarded only if the your programs have been demonstrated to the lab supervisor.**

3. The beginning of your coursework submission must contain the following information:

```
-- CS-191 Functional Programming I - Coursework 2
-- Author(s): name(s) and student number(s)
-- Submission date:
```

4. Print your coursework, attach a signed submission form (available at the students office) and put it in the wooden box on the second floor.

If you are working in pairs, please submit only **one** printout and attach only **one** submission slip with both names on it.

Please do **not** include printouts of test results, as these will be assessed in the labs.

5. Late submissions will be penalised by taking marks off.