

Mid-semester Exam — Practice Questions

Question 1

An obvious way to represent a point in the xy plane is as a pair of Floats, being the cartesian coordinates of the point:

```
type Point = (Float,Float)
```

The distance between two points (x_1, y_1) and (x_2, y_2) is given by the following formula:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Define a Haskell function `distance :: Point -> Point -> Float` to calculate the distance between two points using this formula.

Question 2

In the cartesian coordinate system, a triangle can be represented by its three vertices:

```
type Triangle = (Point,Point,Point)
```

A triangle is *equilateral* if all its sides are the same length. Define a Haskell function:

```
isEquilateral :: Triangle -> Bool
```

which checks whether a triangle is equilateral.

Question 3

Describe the behaviour of the following function:

```
f :: [Point] -> [Float]
f pts = map (distance origin) pts
  where origin = (0,0)
```

Question 4

For security reasons, computer systems managers often apply conditions on the selection of ‘words’ that can be used for passwords. Such a set of conditions may be that the password must have:

- not less than 7 characters;
- at least one upper-case letter;
- at least two lower-case letters;
- at least one non-alphabetic character

Design and implement a Haskell function that will check passwords for compliance with these criteria. Your main function should have type:

```
checkPass :: String -> Bool
```

Question 5

The following function definitions are missing their type signatures. What are their types?

```
add ns          = summer ns 0
summer []      r = r
summer (n:ns) r = summer ns (r+n)

every xs       = each xs True
each [] r      = r
each (x:xs) r  = each xs (x && r)
```

Question 6

Are each of the following valid Haskell expressions? If so, what are their values? If not, why are they invalid?

```
length [[1,2,3,4],[5],[]]
tail [[1,2,3,[4]],5,[]]
map isUpper "Cracking cheese, Gromit!!"
[1] 'elem' [[1,2,3,4],[5],[]]
[[1,2,3,4],[5],[]] ++ [7,8,9]
filter isUpper "Sheik Yerbouti"
```

Question 7

The following functions have been disguised by giving them nonsensical names. Describe the behaviour of each function and suggest more meaningful names.

```
happy :: Eq a => a -> [a] -> Bool
happy x []      = False
happy x (y:ys) = (x == y) || happy x ys

sad :: [Int] -> [Int] -> Bool
sad [] []       = True
sad xs []       = False
sad [] ys       = False
sad (x:xs) (y:ys) = (x == y) && sad xs ys

watermelon :: [Int] -> Bool
watermelon []   = True
watermelon [x] = True
watermelon (x:xs) = (x <= y) && watermelon xs
  where (y:ys) = xs
```

```

frank :: [a] -> [a]
frank [] = []
frank (x:xs) = frank xs ++ [x]

cheese :: [a] -> [a]
cheese [] = error "No cheese."
cheese [x] = []
cheese (x:xs) = x : cheese xs

play :: [Float] -> Float
play [] = 0
play xs = your xs / guitar xs
  where your [] = 0
        your (x:xs) = x + your xs
        guitar [] = 0
        guitar (x:xs) = 1 + guitar xs

```

Question 8

Give a Haskell definition of a function

```
stutter :: Eq a => a -> [a] -> [a]
```

such that the expression `stutter x ys` returns a list identical to `ys`, except that each occurrence of `x` has been replaced by *two* occurrences of `x`. For example:

```

stutter 'f' "Offer Fred some food." ==> "Offffer Fred some ffood."
stutter 3 [1,2,3,4,3,2,1] ==> [1,2,3,3,4,3,3,2,1]

```

Briefly explain the type signature of `stutter`.

Question 9

Draw the structure of a binary search tree with keys inserted in the following order:

```
[42, 23, 22, 30, 65, 101, 16, 10, 17]
```

Question 10

What is the depth of the tree in the previous question?

Question 11

The binary trees discussed in lectures were represented with the following data type:

```
data Tree a = Null | Node a (Tree a) (Tree a)
```

Write a function:

```
treeDepth :: Tree a -> Int
```

to calculate the depth of an arbitrary binary tree.