Exercise Sheet Two: Types, Classes and Functions

The aim of this tutorial is for you to gain an understanding of types and type inference in Haskell. Included in this is the use of polymorphic functions and overloaded functions by way of type classes. By the end of the tutorial you should also be able to define functions using both guarded equations and pattern matching.

Attempt the exercsise in your own time before Tuesday's lab and use that session for obtaining feedback and for asking questions related to the exercises.

Exercise One

What are the types (if any) of the following expressions?

```
['a','b','c']

('a','b','c')

['a',3,True]
```

```
('a',3,True)

[ (False, '0'), (True,'1')]

( [True,False] , ['0','1'] )

[tail, init, reverse]

[]

2 : 3 : [] : 4 : 5 : []

[] : []
```

Check your answers using GHCi and the command ":type expr"

Exercise Two

Write down expressions that have the following types:

```
bools :: [Bool]

nums :: [[Int]]

add :: Int -> Int -> Int -> Int

copy :: a -> (a,a)
```

```
apply :: (a -> b) -> a -> b

explode :: String -> [Char]
```

Exercise Three

What are the types of the following functions?

```
second xs = head (tail xs)

swap (x,y) = (y,x)

pair x y = (x,y)

double x = x*2

palindrome xs = reverse xs == xs

twice f x = f ( f x )
```

Take care to include necessary class constraints where appropriate.

Exercise Four

The class Eq is instantiated by all of the basic types as well as Lists and Tuples built from these. However, function types are not an instance of the Eq class. Suggest reasons why this is the case.

Exercise Five

Use library functions that work with List types to define a function `halve :: [a] -> ([a], [a])` that splits an even length list in to two halves. For example

```
> halve [1,2,3,4,5,6]
([1,2,3],[4,5,6])
```

Can you define this function directly using pattern matching on Lists?

Exercise Six

Define a function `third :: [a] -> a` that returns the third element in a list that contains at least this many elements. Do this using the following three techniques:

- `head` and `tail`
- list indexing `!!`
- pattern matching

Think about what to do in the cases where there are fewer than three elements in the list.

Exercise Seven

Consider a function `safetail :: [a] -> [a]` that behaves in the same way as `tail` except that rather than produing an error it returns the empty list in cases where no tail exists. Using `tail` and `null :: [a] -> Bool` that decides whether a list is empty or not, define `safetail` using

- if-then-else
- guarded equations
- pattern matching

For your solutions, which approach produced the most concise code?

Exercise Eight

Define the logical disjunction operator || using pattern matching. Is there more than one way of doing this?

Exercise Nine

Write a function `enc :: Int -> String -> String` that encrypts a String by adding some given integer to each character's Unicode value. Now, using `enc` as a locally declared function write a function that accepts a String and an Int and returns a pair of an encrypted string and a function with which to decrypt the string.

Exercise Ten

The Luhn algorithm is used to check bank card numbers for simple errors such as mistyping a digit. It proceeds as follows:

- Consider each digit as a separate number
- Moving left, double every other number from the second last
- Subtract 9 from each number that is now greater than 9
- Add all of the resulting numbers together
- If the total is divisible by 10 then card number is valid

Define a function `luhnDouble :: Int -> Int` that doubles a digit and subtracts 9 if the result is greater than 9. Using `luhnDouble` and the function `mod` define a function `luhn :: Int -> Int -> Int -> Int > Bool` that decides whether a four digit card number is is valid. For example

```
> luhn 1 7 8 4
True
> luhn 4 7 8 3
False
```