Exercise Sheet 3 : List Comprehension and Recursion

The aim of this tutorial is to introduce the iteration operations in Haskell. Primarily these consist of list comprehensions and recursively defined functions. By the end of this tutorial you should be able to use both constructs.

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

Using a list comprehension, give an expression that calculates the sum of the squares of odd numbers and cubes of even numbers for the first 100 integers.

Exercise Two

Suppose that a coordinate grid of size m x n is given by the list of all pairs (x,y) of integers for 0 <= x <= m and 0 <= y <= n. Using a

list comprehension, define a function `grid :: Int -> Int -> [(Int,Int)]` that returns a coordinate grid of a given size. For example

```
> grid 1 2
[(0,0), (0,1), (0,2), (1,0), (1,1), (1,2)]
```

Now use a list comprehension to create a function `square :: Int -> [(Int,Int)]` that produces a square grid but excluding the diagonal from (0,0) to (n,n). For example

```
> square 2
[(0,1), (0,2), (1,0), (1,2), (2,0), (2,1)]
```

Exercise Three

Define the standard library funcion `replicate :: Int -> a -> [a]` using a list comprehension.

Exercise Four

A triple (x,y,z) of positive integers is *Pythagorean* if it satisfies the equation x^2 + y^2 = z^2. Using a list comprehension with three generators, define a function `pyths :: Int -> [(Int,Int,Int)]` that returns the list of

all such triples whose components are at most a given limit. For example

```
> pyths 10
[(3,4,5), (4,3,5), (6,8,10), (8,6,10)]
```

Exercise Five

A positive integer is *perfect* if it equals the sum of all of its factors, excluding the number itself. Using a list comprehension and the function `factors`, define a function `perfect :: Int -> [Int]` that returns the list of all perfect numbers up to a given limit. For example

```
> perfects 500
[6,28,496]
```

Exercise Six

Redefine the function `positions :: Eq a => a -> [a] -> [Int]` given in the lecture notes that returns all indexes the given element appears at in the given list. However, rather than using a list comprehension use the function `find` where

```
```

```
find :: Eq a => a -> [ (a,b)] -> [b]
find k t = [ v | (k',v) <- t, k==k']
```

Exercise Seven

The *scalar product* of two lists of integers xs and ys of length n is given by the sum of the products of the corresponding integers: (x1*y1) + (x2*y2) + ... + (xn*yn). Define a function to calculate the scalar product of two lists using a list comprehension.

Exercise Eight

Define a recursive function `sumdown :: Int -> Int-> Int` that returns the sum of the non-negative integers from the first given value down to the second given value. For example sumdown 7 4 should return the result of 7 + 6 + 5 + 4

Exercise Nine

Define the recursive function `euclid :: Int -> Int -> Int` that implements *Euclid's algorithm* for calculating the greatest common divisor of two non-negative integers: if the two numbers are equal, this number is the result, otherwise, subtract the small from the larger and repeat.

Exercise Ten

Without using any list sorting functions, write a recursive function `merge :: Ord a => [a] -> [a] -> [a]` that merges two sorted lists to give a single sorted list. Now write a function called `mergeSort :: Ord a => [a] -> [a]` that sorts a list using the merge sort algorithm. You will find it helpful to define a local function `halve ::[a] -> ([a],[a])` that splits a list in to two halves whose lengths differ by at most one.