Programming in Haskell by Graham Hutton Exercises

1. Using a list comprehension, give an expression that calculates the sum 1^2 + 2^2 + ... 100^2 of the first one hundred integer squares.

2. In a similar way to the function length, show how the library function replicate :: Int -> a -> [a] that produces a list of identical elements can be defined using a list comprehension. For example:

```
> replicate 3 True
[True, True, True]
```

3. A triple (x, y, z) of positive integers is pythagorean if x^2 + y^2 = z^2. Using a list comprehension, define a function pyths :: Int -> [(Int, Int, Int)] that returns the list of all pythagorean 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)]
```

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

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

5. Show how the single comprehension [(x,y) | x <- [1,2,3], y <- [4,5,6]] with two generators can be re-expressed using two comprehensions with single generators. Hint: make use of the library function concat and nest one comprehension within the other.

I must admit I was totally confused over this one.

```
[[(x,y)| y <- [4,5,6]] | x <- [1,2,3]]
>[[(1,4),(1,5),(1,6)],[(2,4),(2,5),(2,6)],[(3,4),(3,5),(3,6)]]
```

6. Define the function find used in the function positions.

```
positions :: Eq a => a -> [a] -> [Int]
positions x xs = find x (zip xs [0..n])
    where n = (length xs) - 1
```

7. The scalar product of two lists of integers xs and ys of length n is given by the sum of the products of corresponding integers.

```
n=1
--
\ (xsi * ysi)
/
--
i=0
```

In a similar manner to the function chisqr, show how a list comprehension can be used to define a function scalarproduct :: [Int] -> [Int] -> Int that returns the scalar product of two lists. For example:

```
> scalarproduct [1,2,3] [4,5,6]
32
```

8. 1. Define the exponentiation operator &! for non-negative integers using the same pattern of recursion as the multiplication operator *, and show how 2 &! 3 is evaluated using your definition.

9. 1. Show how the list comprehension [f x | x <- xs, p x] can be re-expressed using the higher-order functions
map and filter. Try to understand and apply the example [(+7) x | x <- [1..10], odd x]

10. 4. Define a function dec2int :: [Int] -> Int that converts a decimal number into an integer. for example:

```
> dec2int [2,3,4,5]
2345
```

11. A higher-order function unfold that encapsulates a simple pattern of recursion for producing a list can be defined as follows:

```
unfold p h t x | p x = []
               |otherwise = h x : unfold p h t (t x)
```

Define new functions so that you can call unfold function passing a list x as parameter and new functions for passing a primitive type element x