

# Extra Haskell Exercises

## Informatics 1 — Functional Programming

October 12, 2010

1. Write the functions `squares` and `squaresRec` to square all the elements of a list. Use a list comprehension and then recursion.

```
squares [1..5] = [1,4,9,16,25]
```

2. Write the functions `odds` and `oddsRec` to remove even elements from a list. Use a list comprehension and then recursion.

```
odds [1..10] = [1,3,5,7,9]
```

3. Write the functions `greet` and `greetRec` to prepend the string "Hello " to each element of a list of strings. Use a list comprehension and then recursion.

```
greet ["Eliza", "Vixen", "Alice"] =  
  ["Hello_Eliza", "Hello_Vixen", "Hello_Alice"]
```

4. Write the functions `invert` and `invertRec` to negate all the booleans in a list. Use a list comprehension and then recursion.

```
invert [3 > 5, True, even 20] = [True, False, False]
```

5. Write the functions `multiples` and `multiplesRec` to obtain given multiples from a list. Use a list comprehension and then recursion.

```
multiples 3 [1..20] = [3,6,9,12,15,18]  
multiples 5 [1..20] = [5,10,15,20]
```

6. Write the functions `sqrts` and `sqrtsRec` to obtain square roots of non-negative elements of a list. Use a list comprehension and then recursion.

```
sqrts [9,2,-1,4] = [3.0,1.4142135,2.0]
```

7. Write the function `upperConcat` using a list comprehension to append two strings and convert the result to uppercase.

`upperConcat "hello" "_there!" = "HELLO_THERE!"`

8. Write the functions `exps` and `expsRec` which take a base and a list of exponents and raises the base to each of the non-negative exponents. Use a list comprehension and then recursion.

`exps 2 [1..5] = [2,4,8,16,32]`  
`exps 5 [1..5] = [5,25,125,625,3125]`

9. Write the function `half` which returns the first half of a string with even length. If the string has odd length, it should throw an error.

`half "hellogoodbye" = "hellog"`

10. Using your function `half`, write the functions `halfStrings` and `halfStringsRec` which half all the even-lengthed strings. Use a list comprehension and then recursion.

`halfStrings ["hellogoodbye", "hello", "goodbye!"] =`  
`["hellog", "good"]`

11. Write the functions `subtract` and `subtractRec` which take a lists of pairs. For each element `(x,y)`, they should compute `x - y` whenever `y` is greater than `x`. Use a list comprehension and then recursion.

`subtract [(2,8), (9,1), (1,7), (0,6), (4,3)] = [8,1]`

12. Write the functions `products` and `productsRec` which multiply the elements of two lists pairwise. Use a list comprehension and then recursion.

`products [2,9,1,0,4] [8,1,7,6,3] = [16,9,7,0,12]`

13. Write new versions of the above functions, `productsStrict` and `productsRecStrict`, which throw an error when the input lists have different lengths. Use a list comprehension and then recursion.

14. Write the functions `filterCond` and `filterCondRec` which take a list of pairs, and returns the second element of each pair whenever the first element is true. Use a list comprehension and then recursion.

`filterCond [(even 2,2), (even 3,3) (even 4,4)] = [2,4]`  
`filterCond [(even x,x) | x <- [1..10]] = [2,4,6,8,10]`

15. Write the functions `surround` and `surroundRec` which surround a string with another string. Use a list comprehension and then recursion.

`surround "!!!" "hello" = "!!!hello!!!"`

16. Write the functions **intersperse** and **intersperseRec** which combine a list of strings, placing a given string in between them. Use a list comprehension and then recursion.

```
intersperse "and_"
["It was red", "yellow", "green", "brown..."]
= "It was red and yellow and green and brown..."
```

17. Use your function **exps** to define the list of powers of 2.

```
powers2 = [1,2,4,8,16,...]
```

18. Use your function **products** and **powers2** to define a function which converts a list of binary digits into the equivalent decimal:

```
binaryConv [1,1,0,0,1] = 25
```