# ASSIGNMENT 5 – HOF & COMPREHENSIONS

*Advanced programming paradigms*

In this assignment, you will apply different higher-order functions on lists. In the second part of the assignment, you will use for comprehensions to produce an exhaustive search for postcards and to work with the $n$ queens problem.

## Question 1 – *Higher-order functions on lists*

Using only the `zip` and `map` higher-order functions, define the following functions that:

(a) Returns a list of the length of each string of a list
*Example*:
`lengthStrings(List("How","long","are","we?"))` returns `List(3,4,3,3)`

(b) Produces a list with $n$ identical elements of arbitrary type (don't use the `fill` method!)
*Examples*:
`dup("foo", 5)` returns `List("foo", "foo", "foo", "foo", "foo")`
`dup(List(1,2,3), 2)` returns `List(List(1,2,3), List(1,2,3))`

(c) Multiplies element-wise two lists of values and create a new list
*Example*:
`dot(List(1,2,3), List(2,4,3))` returns `List(2,8,9)`

## Question 2 – *Folding functions on lists*

Now using folding (right or left) define a function that:

(a) Determines if all logical values in a non-empty list are `true`.
*Examples*:
`areTrue(List(true, true, false))` returns `false`
`areTrue(List(true, true, true))` returns `true`

(b) Determine the total length of the strings in a list
*Example*:
`lString(List("Folding", "is", "fun"))` returns `12`

(c) Returns the longest string of a list as well as its size
*Example*:
`longest(List("What", "is", "the", "longest?"))` returns `(8, longest?)`

(d) Decide if a value is an element of a list of an arbitrary type
*Examples*:
`isPresent(List(1,2,3,4), 5)` returns `false`
`isPresent(List(1,2,3,4), 3)` returns `true`

(e) Flatten a nested list structure of any type.
*Example*:
`flattenList(List(List(1,1), 2, List(3, List(5, 8))))` returns `List(1,1,2,3,5,8)`

⚠*Turn page* →

## Question 3 – *Tuples*

This is the code given in class for returning the sums of the couple of integers under a certain value that are prime:

```
1  def primeSum(max: Int): List[(Int, Int)] =
2    for {
3      i <- (1 to max).toList
4      j <- (1 to max).toList
5      if (isPrime(i + j))
6    } yield (i, j)
7
8  def isPrime(i: Int): Boolean =
9    i match {
10     case i if i <= 1 => false
11     case 2 => false
12     case _ => !(2 to (i - 1)).exists(x => i % x == 0)
13   }
```

The problem of this code is that the `primeSum` function returns values twice. For instance, it returns `(1,2)` but also `(2,1)`. Because those values are only permuted, we would like to remove redundant pairs. Without changing the existing code, devise a function that removes the existing permutations from a similar list.

(a) Using pattern matching with tuples and recursion

(b) Using folding with tuples

## Question 4 – *Sending postcards*

In this exercise, you will use for comprehensions for generating postcards when you are travelling. You are given the following code that codes defines the city you visit, the person you want to send the cards to and the travellers that are sending postcards.

```
1  val cities = List("Paris", "London", "Berlin", "Lausanne")
2  val relatives = List("Grandma", "Grandpa", "Aunt Lottie", "Dad")
3  val travellers = List("Pierre-Andre", "Rachel")
```

(a) In each city you visit, each travellers send a postcard to all their relatives. Because the task is tedious, you are asked to write a program that generates the text for the postcards as follows:

```
Dear Grandma, Wish you were here in Paris! Love, Pierre-Andre
Dear Grandma, Wish you were here in London! Love, Pierre-Andre
...
Dear Grandpa, Wish you were here in Paris! Love, Pierre-Andre
Dear Grandpa, Wish you were here in London! Love, Pierre-Andre
...
Dear Grandma, Wish you were here in Paris! Love, Rachel
Dear Grandma, Wish you were here in London! Love, Rachel
...
Dear Dad, Wish you were here in Lausanne! Love, Rachel
```

(b) How do you modify your code in order to send postcards only to the family members whose name start with the letter `G`? You answer can (and should) be written in one line.

## Question 5 – *Printing the $n$ queens solutions*

A possible solution to the n-queen problem that your had to analyze is as follows:

```
1  def queens(n: Int): List[List[(Int, Int)]] = {
2    def placeQueens(k: Int): List[List[(Int, Int)]] =
3      if (k == 0)
4        List(List())
5      else
6        for {
7          queens <- placeQueens(k - 1)
8          column <- 1 to n
9          queen = (k, column)
10         if isSafe(queen, queens)
11       } yield queen :: queens
12
13   placeQueens(n)
14 }
15
16 def isSafe(queen: (Int, Int), queens: List[(Int, Int)]) =
17   queens forall (q => !inCheck(queen, q))
18
19 def inCheck(q1: (Int, Int), q2: (Int, Int)) =
20     q1._1 == q2._1 || // same row
21     q1._2 == q2._2 || // same column
22     (q1._1 - q2._1).abs == (q1._2 - q2._2).abs // on diagonal
```

Using for comprehensions, write a function `def printChessBoard(...)` that returns a string containing all the solutions for a given $n$. For instance,

```
1  println(printChessBoard(queens(4)))
```

should display the following on the console:

```
Solution 1:
|_|♛|_|_|
|_|_|_|♛|
|♛|_|_|_|
|_|_|♛|_|

Solution 2:
|_|_|♛|_|
|♛|_|_|_|
|_|_|_|♛|
|_|♛|_|_|
```

*Remark:* Note that the Unicode character for the black queen used in the above example is `\u265b`.