UNIVERSIDAD DE MÁLAGA | uma.es

Dpt. Languages and Computer Sciences

Advanced Programming II
Midterm exam on Functional Programming. March 28, 2025

Instructions:

- In this exam you have to develop all the requested functions in a single Scala worksheet and you have to submit to the Virtual Campus only such a worksheet to be marked by the teacher.
- A solution will be evaluated only in case it compiles, otherwise it will be marked with 0 pts.
- This exam is composed of five exercises.

1. (1.5 pts.) Using **`foldRight`**, define a function:
   ```
   def collapse[A](l: List[A]): List[(A, Int)] =
   ```
   that summarizes adjacent equal values into a pair (value, numRepetitions). For example:
   ```
   collapse(List(2, 2, 2, 5, 3, 1, 1, 6, 6, 6))
   ```
   gives as a result:
   ```
   List((2,3), (5,1), (3,1), (1,2), (6,3))
   ```
2. (1.5 pts.) Using **`foldLeft`** or **`foldRight`**, define a function:
   ```
   def movingSum(lst: List[Int]): List[Int]
   ```
   that, given a list of integers, return a list where each element at index **`i`** is the sum of all previous elements up to **`i`**. For example:
   ```
   movingSum(List(1, 2, 3, 4))
   ```
   gives as a result:
   ```
   List(1, 3, 6, 10)
   ```
   Note: Efficiency will be evaluated.

3. (2.0 pts.) Write a tail-recursive function with the next declaration:
   ```
   def countItems(lst: List[Any]): Int
   ```
   that counts all elements in a nested list. For example, if the list contains integers or other lists then the next function call:
   ```
   countItems(List(List(1, 2), 3, List(4, List(5, 6)), 7))
   ```
   gives as a result:
   ```
   7
   ```

Continue in the next page...

4. (2.5 pts.) Write a tail-recursive function with the next declaration:

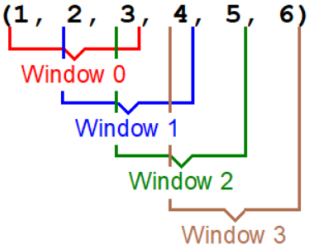
```
def slidingTransform[T, U](l: List[T], size: Int, f: List[T] => U): List[U]
```

that takes three parameters:

- a list of elements.
- a window size (number of elements to process at each step)
- a function f that processes each window

and it should apply f to each window and return a transformed list. In the next example each window is composed of the adjacent items because the **size** parameter is 3:

```
slidingTransform(List(1, 2, 3, 4, 5, 6), 3, _.sum)
```



gives as a result:

```
List(6, 9, 12, 15)
```

5. (2.5 pts.) Using concatenated higher-order functions, write a function with the next declaration:

```
def mostFrequentCharPerWord(words: List[String]): List[(String, Char)]
```

that takes a list of words and returns a list of pairs where each word is associated with its most frequently occurring character. If there is a tie, it returns any of the tied characters. For example:

```
mostFrequentCharPerWord(List("banana", "apple", "cherry"))
```

gives as a result:

```
List((banana,a), (apple,p), (cherry,r))
```

Hint: You may use higher-order functions like **groupBy**, **values**, **toList**, **sortBy**, **map**, etc. Remember that a String can be managed like a List of chars.