159.272 Programming Paradigms

Tutorial 2 - Higher-order methods for Scala lists

Try to complete these exercises using, where possible, the higher-order methods for lists that we have seen in lectures. Use the file **`tutorial2.scala`** as a template to demonstrate your results. This file provides you with correct type annotations for the functions you are asked to define. Upload your completed **`tutorial2.scala`** file to Stream.

Recall the higher-order **`List`** methods introduced in lectures

- **`myList exists (pred)`**
- **`myList forall (pred)`**
- **`myList map (mapfn)`**
- **`myList flatMap (listfn)`**
- **`myList filter (pred)`**
- **`myList partition (pred)`**
- **`myList takeWhile (pred)`**
- **`myList dropWhile (pred)`**
- **`myList span (pred)`**
- **`myList.foldLeft (startVal) (binaryOp)`**
- **`myList.foldRight (startVal) (binaryOp)`**
- **`myList.reduceLeft (binaryOp)`**
- **`myList.reduceRight (binaryOp)`**

1. Define a predicate (Boolean-valued function) **`duplicated`** that takes a list and a list element as arguments and returns True if the list contains more than one copy of the element, False otherwise.

   ```
   def duplicated[T](xs:List[T], x:T): Boolean = {
     }
   ```

2. Using your **`duplicated`** predicate as an argument to a suitable higher-order list method, define a function **`noDups`** that takes a list as its argument and returns True if the list does not contain any duplicated elements, False otherwise.

   ```
   def noDups[T](xs:List[T]): Boolean = {
     }
   ```

3. Implement the following functions using applications of the **`fold`** methods we have seen in lectures.
   1. **`flength`** takes a list and returns the number of items in the list
   2. (harder) **`fcompress`** takes a list and returns a list with consecutive duplicate elements eliminated, as seen in tutorial 1 (Hint: use **`foldRight`**)

4. Implement the following function using an application of **`flatMap`**:
   `duplicateN` takes an integer **`n`** and a list **`ls`** and returns a list containing each element of **`ls`** duplicated **`n`** times

5. (harder) Implement the following function using any of the constructs we have seen so far:
   `flattenN` takes a nested list of type **`List[Any]`** and returns a flattened list

   For example:

   ```
   flattenN(List(1,2),List(List(1,2),List(3,4)), 1, List(2,3))
   List(1,2,1,2,3,4,1,2,3)
   ```

6. Implement the following function using any of the constructs we have seen so far:
   `isSorted` takes a list and an ordering function for elements of the list and returns True if the list is sorted according to the ordering function, False otherwise (Hint: **`zip`** the list with its tail to get a list of consecutive pairs of elements from the original list)

   ```
     def isSorted[A](xs:List[A], ord:(A, A) => Boolean): Boolean = {
   }
   ```