# Programming Paradigms Fall 2023 — Problem Sets

by Nikolai Kudasov and Khaled Ismaeel

October 26, 2023

## 1 Problem set №9

1. Implement the following functions over lists. You can use explicit recursion or higher-order functions that we covered so far. Make sure that your functions are **lazy** and work properly on all provided examples:

   (a) A function that checks whether a given list is a singleton (contains exactly one element):

   ```
   isSingleton :: [a] -> Bool
   ```

   ```
   >>> isSingleton [1]
   True
   >>> isSingleton [1..]
   False
   >>> isSingleton [[1..]]
   True
   ```

   (b) A function that inserts a number into a sorted (descending) list of numbers:

   ```
   insert :: Int -> [Int] -> [Int]
   ```

   ```
   >>> insert 7 [9,8,5,3]
   [9,8,7,5,3]
   >>> insert 7 [9,8,8]
   [9,8,8,7]
   >>> take 5 (insert 7 [9,8..])
   [9,8,7,7,6]
   ```

   (c) A function that puts a separator between every two consecutive elements:

   ```
   separateBy :: [a] -> [a] -> [a]
   ```

   ```
   >>> separateBy "," "hello"
   "h,e,l,l,o"

   >>> take 7 (separateBy [0,0] [1..])
   [1,0,0,2,0,0,3]
   ```

   (d) Split a list into a maximal prefix where all elements satisfy the predicate and suffix (all leftover elements):

   ```
   splitWhen :: (a -> Bool) -> [a] -> ([a], [a])
   ```

   ```
   >>> splitWhen (== ' ') "Hello, world!"
   ("Hello,"," world!")

   >>> take 10 (fst (splitWhen (>= 100) [1..]))
   [1,2,3,4,5,6,7,8,9,10]

   >>> take 10 (snd (splitWhen (>= 100) [1..]))
   [100,101,102,103,104,105,106,107,108,109]

   >>> take 10 (fst (splitWhen (< 0) [1..]))
   [1,2,3,4,5,6,7,8,9,10]
   ```

(e) A function that groups elements, removing separators (elements that satisfy a given predicate):

```
groupsSeparatedBy :: (a -> Bool) -> [a] -> [[a]]
```

```
>>> groupsSeparatedBy (== ' ') "Here are some words!"
["Here","are","some","words!"]

>>> take 3 (groupsSeparatedBy (\n -> n `mod` 4 == 0) [1..])
[[1,2,3],[5,6,7],[9,10,11]]
```

2. Define the following infinite lists:

(a) A sequence of Trifibonacci numbers: 0, 1, 1, 2, 4, 7, 13, 24, 44, 81, 149, .... The sequence starts with 0, 1, 1 and each of the next elements $x_n$ is defined as the sum of the three previous elements: $x_n = x_{n-1} + x_{n-2} + x_{n-3}$:

```
trifibonacci :: [Integer]
```

```
>>> take 10 trifibonacci
[0,1,1,2,4,7,13,24,44,81]
```

(b) A sequence of approximations of the square root of 3. Any given approximation $x$ can be improved into a better approximation $x'$ using the formula $x' = \frac{x}{2} + \frac{3}{2x}$.

```
approximationsOfRoot3 :: Double -> [Double]
```

```
>>> take 5 (approximationsOfRoot3 1)
[0.5,3.25,2.0865384615384617,1.7621632399858207,1.7323080932066346]
```