Functional Programming for Logicians
Homework 4

Péter Mekis
Department of Logic, ELTE Budapest

Deadline: 2019 March 11 17:59 pm

- Define *any five of the following functions* in Haskell. Defining more than five is appreciated, but not necessary. Some of the exercises are follow-ups to others; it may be a good idea to choose them together.
- Also, define *three functions that aren't in this list*, based on your ideas, preferably inspired by your main field of interest.
- Use recursion in every function you define. Get ideas from the functions we defined in this week's session, or the sample given below.
- If Haskell has a ready-made solution to an exercise, don't use it. **New: The preferred method of solution is indicated in the exercises.**
- If the description of an exercise is ambiguous, be creative.
- Declare the types of your functions. If you need non-integer numbers for your own functions, use the 'Double' type.
- If you get stuck with the exercises, contact me or your fellow students. Don't let yourself get frustrated by difficulties, developing a recursive mindset takes time. If you use code that was created by someone else, indicate it.
- Make sure you submit a code that compiles in ghci. Annotation is appreciated.
- The exercises range from the more elementary to the more advanced. Choose those that are at your level. Have fun! :)

Sample **Type** `[(Integer,Integer)] -> (Integer -> Integer))`

Method Recursion.

Description Creates a function from a list of pairs.

Examples

```
> fnFromPairs [(0,1),(1,2),(2,4),(3,8),(4,16),(5,32)] 4
16
```

Solution

```
fnFromPairs :: [(Integer,Integer)] -> (Integer -> Integer)
fnFromPairs [] z        = undefined
fnFromPairs ((x,y):l) z = if z==x then y else (fnFromPairs' l z)
```

Sample **Type** `Integer -> [Integer]`

Method List comprehension.

Description Returns the list of the first n primes.

Examples

```
> firstPrimes 7
[2,3,5,7,11,13,17]
```

Solution

```
firstPrimes :: Int -> [Integer]
firstPrimes n = take n [i | i <- [2..], properDivisors i == []] where
  properDivisors :: Integer -> [Integer]
  properDivisors i = [j | j <- [2..(i-1)], mod i j == 0]
```

---

1. **Type** `Integer -> Integer`

   Method Recursion.

   Description In the March 4 session we considered the following simple version of computing the nth Fibonacci number:

   ```
   fibonacci :: Integer -> [Integer]
   fibonacci 0 = 0
   fibonacci 1 = 1
   fibonacci n = fibonacci (n-1) + fibonacci (n-2)
   ```

   This function is very slow, since the recursion is bifurcating at every step, resulting in calculating the same things over and over again. At around $n = 35$ the function execution time started to grow dramatically. Define a version that is faster. Hint: use an auxiliary function with two more arguments, and use these to pass the last two numbers in the sequence at each recursive call.

   Example

   ```
   > fibonacciFast' 100
   354224848179261915075
   ```

2. **Type** `(Integer -> Integer -> Integer) -> (Integer -> Integer -> Integer)`

   Method Direct definition without recursion.

   Description Flips the arguments of a binary integer function.

   Examples

   ```
   > flip' (^) 2 3
   9
   > flip' (^) 3 2
   8
   ```

3. **Type** `(Integer -> Integer -> Integer) -> ((Integer,Integer) -> Integer)`

Method Direct definition without recursion.

Description This function takes a binary string function f and *uncurries* it; its input will be a unary function that takes a pair, and does the same job as f.

Examples

```
> uncurry' (+) (7,3)
10
uncurry' mod (7,3)
1
```

4. **Type** `(String -> Bool) -> [String] -> Bool`

   Method Recursion.

   Description Checks whether a predicate is true of all elements in a list of strings.

   Examples

   ```
   > all' (elem 'o') ["Bill Clinton","George Bush",
   "Barack Obama","Donald Trump"]
   False
   > all' (<"Viktor") ["Merkel","Macron","May"]
   True
   ```

5. **Type** `(Integer -> Bool) -> [Integer] -> Bool`

   Method Recursion.

   Description Checks whether a predicate is true of any elements in a list of integers.

   Examples

   ```
   > any' even [2,3,5,7,11,13,17]
   True
   > any' (<10) [11,12,13]
   False
   ```

6. **Type** `[String] -> [Integer] -> [(String,Integer)]`

   Method Recursion.

   Description Zips a string list and an integer list together, so that the resulting list consists of pairs the first member of which is from the string list, and the second from the integer list. The zipped list ends where either one of its inputs end.

   Examples

   ```
   > zip' ["Clinton","Bush","Obama","Trump"]  [1992,2000,2008,2016]
   [("Clinton",1992),("Bush",2000),("Obama",2008),("Trump",2016)]
   > ["un","dos","tres","ole, ole, ole"] [1,2,3]
   [("un",1),("dos",2),("tres",3)]
   ```

7. **Type** `(Integer -> Integer) -> [Integer] -> [Integer] -> [Integer]`

   Method Recursion.

   Description This version of zipping takes a binary integer function f and two integer lists lst1 and lst2, and produces a list the nth element of which is the result of applying f to the nth elements of lst1 and lst2. The zipped list ends where either of its inputs end.

   Examples

   ```
   > zipWith' (*) [1,2,3,4] [5,6,7,8]
   [5,12,21,32]
   > zipWith' (^) [1,2,3,4] [5,6,7,8,9]
   [1,64,2187,65536]
   ```

8. **Type** `(Integer -> Integer -> Integer) -> Integer -> [Integer] -> Integer`

   Method Recursion.

   Description This function folds up an integer list from the left with the help of a binary function f, and an initial value n. First, f is applied to n and the head of the list; then f is applied to the result of the previous application and the head of the tail of the list; and so on.

   Examples

   ```
   > foldl' (+) 0 [1..10]
   55
   > foldl' (*) 1 [1..6]
   720
   ```

9. **Type** `(Integer -> Bool) -> [Integer] -> [Integer]`

   Method Recursion.

   Description This version of the `take` function takes elements from a list while a property f is true of them.

   Examples

   ```
   > takeWhile' odd [3,5,7,8,3]
   [3,5,7]
   > takeWhile' (/=7) [3,5,7,8,3]
   [3,5]
   ```

10. **Type** `(Integer -> Bool) -> [Integer] -> [Integer]`

    Method List comprehension.

    Description Returns the even numbers greater than zero and less than or equal to n.

    Examples

    ```
    > evensLeq 17
    [2,6,8,10,12,14,16]
    > evensLeq (-17)
    []
    ```

11. **Type** `(Integer -> Bool) -> [Integer] -> [Integer]`

    Method List comprehension.

    Description Returns the prime numbers less than or equal to n.

    Examples

    ```
    > primesLeq 17
    [2,3,5,7,11,13]
    > primesLeq (1)
    []
    ```

12. **Type** `String -> [String]`

    Method List comprehension. (From the previous set; but with new method.)

    Description Slices up a string into substrings that consist of a single character.

    Example

    ```
    > slice "Trump"
    ["T","r","u","m","p"]
    ```

13. **Type** `String -> (String, String)`

    Method List comprehension. (From the previous set; but with new method.)

Description Separates the vowels and the consonants of a word. Neglects any other character.

Example

```
> separate "Donald Trump"
("oau","DnldTrmp")
```

14. **Type** `String -> String`

    Method List comprehension. (From the previous set; but with new method.)

    Description Reduces a string so that it keeps only the first occurrences of every character.

    Example

    ```
    > reducestring "aaargh"
    "argh"
    > reducestring "Gottlob Frege"
    "Gotlb Freg"
    ```

15. **Type** `String -> [String]`

    Method List comprehension. (From the previous set; but with new method.)

    Description Creates a list with all substrings of a string. (The examples represent two different approaches.)

    Example

    ```
    > substrings "abc"
    ["", "a", "b", "c", "ab", "bc", "abc"]
    > substrings' "abc"
    ["", "a", "ab", "abc", "b", "bc", "c"]
    ```

16. **Type** `String -> [String]`

    Method List comprehension. (From the previous set; but with new method.)

    Description Splits a string at the occurrences of a given character.

    Example

    ```
    > split "my body is walking in space" ' '
    ["my","body","is","walking","in","space"]
    > split "ab.c.de.fgh"
    ["ab","c","de","fgh"]
    ```

17. **Type** `[(Integer, Char)] -> String`

    Method List comprehension. (From the previous set; but with new method.)

    Description Creates a string from a list of ordered pairs where the second member is a character and the first member is the number of its consecutive occurrences.

    Example

    ```
    > expand [(1,'a'),(2,'b'),(3,'c')]
    "abbccc"
    ```

18. **Type** `Integer -> [[Bool]]`

    Method List comprehension. (From the previous set; but with new method.)

    Description Creates the input rows (*truth possibilities*) of a truth table for n elementary propositions.

    Example

    ```
    > truth_poss 2
    [[True, True], [True, False], [False, True], [False, False]]
    ```

```
> truth_poss 3
[[True, True, True], [True, True, False],
[True, False, True], [True, False, False],
[False, True, True], [False, True, False],
[False, False, True], [False, False, False]]
```

19. **Type** `String -> Integer -> [String]`

    Method List comprehension. (From the previous set; but with new method.)

    Description Lists all the words of a given length over an alphabet in alphabetical order.

    Example

```
> wordlist "01" 2
["00","01","10","11"]
> wordlist "abc" 3
["aaa","aab","aac","aba","abb","abc","aca","acb","acc",
"baa","bab","bac","bba","bbb","bbc","bca","bcb","bcc",
"caa","cab","cac","cba","cbb","cbc","cca","ccb","ccc"]
```

20. **Type** `[String] -> String`

    Method List comprehension.

    Description Concatenates a list of strings, placing spaces between the parts.

    Example

```
> concat' ["I","want","to","break","free"]
"I want to break free"
```