

Functional Programming for Logicians

Homework 3

Péter Mekis
Department of Logic, ELTE Budapest

Deadline: 2019 March 4 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.
- Don't use advanced tools like list comprehension, lambda abstraction, or importing modules. If Haskell's Prelude module has a built-in solution for an exercise, don't use it. **New: You can use built-in functions from Haskell's Prelude if they do not solve the exercise itself, but make your solution more easier to express. Feel free to google these.**
- 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** `Int -> [[Int]]`

Description Returns the first n rows of Pascal's triangle. (Cf. https://en.wikipedia.org/wiki/Pascal%27s_triangle)

Examples

```
> pascal 1
[[1]]
pascal 5
[[1],[1,1],[1,2,1],[1,3,3,1],[1,4,6,4,1]]
```

Solution

```
pascal :: Int -> [[Int]]
pascal 1 = [[1]]
pascal n = prev ++ [pascal_next (last prev)] where
  prev = pascal (n-1)
  pascal_next xs = head xs : pascal_nf xs where
  pascal_next :: [Int] -> [Int]
  pascal_nf :: [Int] -> [Int]
  pascal_nf xs
    | xs == []      = []
    | tail xs == [] = [head xs]
    | otherwise     = (head xs + head (tail xs)) : pascal_nf (tail xs)
```

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

Description Drops the first k characters of a string. (Special case of Haskell's built-in 'drop' function for strings.)

Examples

```
> drop' 3 "Haskell"
"kell"
> drop' 5 "Java"
""
```

2. **Type** `String -> Integer -> String`

Description Takes the first k characters of a string. (Special case of Haskell's 'take' built-in function for strings.)

Examples

```
> take 3 "Haskell"
"Has"
> take 5 "Java"
"Java"
```

3. **Type** `Integer -> Integer`

Description Integer division; special case of Haskell's built-in 'div' function for the Integer type.

Examples

```
> div' 7 3
2
> div' 0 2
0
```

4. **Type** `String -> Char`

Description Finds and returns the middle element of a string if there is one. Otherwise it returns an exclamation mark.

Example

```
> middlechar "abc"
'b'
> middlechar "abcd"
'!'
```

5. Type String -> Char -> Char

Description Finds the character next to the first occurrence of a character in a string. If there's none, it returns an exclamation mark.

Example

```
> nextto "Gottlob Frege" 'o'
't'
> nextto "abc" 'd'
'!'
```

6. Type String -> [String]

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

Example

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

7. Type String -> Integer

Description Evaluates a simple arithmetic expression with two nonnegative decimal numerals and basic operations +, -, and *.

Examples

```
> "3-7"
-4
> "7*5"
35
```

8. Type String -> (String, String)

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

Example

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

9. Type String -> String

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

Example

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

10. Type String -> [String]

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"]
```

11. **Type** Integer -> Integer]

Description Returns the minimal amount of coins needed to pay a certain amount in the Hungarian coin system. (Standard Hungarian coins are worth 5, 10, 20, 50, 100, and 200 Forints. Amounts are rounded to 5: 98 is rounded to 100, 97 to 95.)

Examples

```
> 198
1
> 572
5
```

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

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"]
```

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

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"
```

14. **Type** Integer -> [[Bool]]

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]]
```

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

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"]
```

16. **Type** String -> String -> Bool

Description Tells whether two strings use the same characters (number of occurrences may differ).

Examples

```
> same_chars "aabbccdd" "daccabacca"
True
> same_chars "aabbccdd" "daccamacca "
False
```

17. Type String -> [String]

Description Splits a string at the occurrences of a given character, if they are not embedded in parentheses.

Example

```
> split "a + (b + c) + d" '+'
["a","(b+c)","d"]
> split "w|((w|w)|(w))|w"
["w","((w|w)|(w))","w"]
```

18. Type String -> Integer

Description Evaluates a complex arithmetic expression with nonnegative decimal numerals and basic operations +, -, and *, fully parenthesized.

Examples

```
> "(((3-7)*4)*2)"
-32
> "((3-(7*4))*2)"
-50
```

19. Type [Int] -> [Int]

Description Sorts a list of integers using the bubble sort algorithm. For further details, cf. https://en.wikipedia.org/wiki/Bubble_sort

Example

```
> bubblesort [3,2,4,1]
[1,2,3,4]
```

20. Type [Int] -> [Int]

Description Sorts a list of integers using the quicksort algorithm. For further details, cf. <https://en.wikipedia.org/wiki/Quicksort>

Example

```
> quicksort [3,2,4,1]
[1,2,3,4]
```