Functional Programming for Logicians
Homework 2

Péter Mekis
Department of Logic, ELTE Budapest

Deadline: 2019 February 25 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. If the description 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** `Char -> String -> Bool`

Description Tells whether a given character occurs in a string.

Examples

```
> occurs 's' "Budapest"
True
> occurs 's' "Vienna"
False
```

Solution

```
-- version 1: pattern matching
occurs :: Char -> String -> Bool
occurs c ""    = False
occurs c (d:s) = if c == d then True else occurs c s
-- version 2: pattern matching
occurs' :: Char -> String -> Bool
occurs' c s
  | s == ""     = False
  | head s == c = True
  | otherwise   = occurs' c (tail s)
```

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

   Description Returns the last character of a string.

   Examples

   ```
   > my_last "Haskell"
   'l'
   > my_last "@"
   '@'
   ```

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

   Description Calculates the length of a string.

   Examples

   ```
   > my_length "Gottlob Frege"
   13
   > my_length ""
   0
   ```

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

   Description Counts the occurrences of a character in a string.

   Examples

   ```
   > count_occur 'o' "Scooby-Doo"
   4
   > count_occur '0' "3.14159265358979323846"
   0
   ```

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

   Description Repeats a character as many times as given.

   Examples

   ```
   > repeat 's' 3
   "sss"
   > repeat 'a' 0
   ""
   ```

5. **Type** `String -> Integer -> Char`

   Description Returns the nth character of a string; and '!' if i is too large. Start indexing from zero.

   Examples

   ```
   > nth_char 0 "Hello world"
   'H'
   > nth_char 7 "Hello world"
   'o'
   > nth_char 100 "Hello world"
   '!'
   ```

6. **Type** `String -> Bool`

   Description Checks whether a string is a valid binary numeral.

   Examples

   ```
   > is_bin "10"
   True
   > is_bin "20"
   False
   > is_bin "01"
   False
   ```

7. **Type** `String -> String`

   Description Reverses a string.

   Examples

   ```
   > reverse "Gottlob Frege"
   "egerF bolttoG"
   > reverse "ahha"
   "ahha"
   ```

8. **Type** `String -> Bool`

   Description Checks whether a string is a strong palindrome, ie. reads the same backward as forward.

   Examples

   ```
   > is_palindr "kayak"
   True
   > is_palindr "(#^_^#)"
   False
   ```

9. **Type** `Integer -> Integer`

   Description Returns the nth element of the Fibonacci sequence. This sequence begins with $F_0 = 0$ and $F_1 = 1$; and for larger indices, $F_n = F_{n-2} + F_{n-1}$: https://en.wikipedia.org/wiki/Fibonacci_number

   Examples

   ```
   > fibonacci 10
   55
   > fibonacci 30
   832040
   ```

10. **Type** `String -> Bool`

    Description Checks whether a string of '('s and ')'s is correctly parenthesised.

Examples

```
> well_parenth "(()(()()))"
True
> begins "(()(()()))"
False
```

11. **Type** `String -> String -> Bool`

    Description Checks whether the first string begins with the second.

    Examples

    ```
    > begins "Gottlob Frege" "Gott"
    True
    > begins "Heidegger" "Heil"
    False
    ```

12. **Type** `Integer -> Integer -> Bool`

    Description Checks whether an integer divides another integer.

    Examples

    ```
    > divides 3 2019
    True
    > divides 2 2019
    False
    ```

13. **Type** `Integer -> Integer`

    Description Sums the digits of an integer.

    Examples

    ```
    > digitsum 2019
    12
    > digitsum 1999
    28
    ```

14. **Type** `Integer -> Integer -> Int`

    Description Returns the greatest common divisor of two integers. (If you get stuck: https://en.wikipedia.org/wiki/Euclidean_algorithm.)

    Examples

    ```
    > gcd 102 74
    2
    > gcd 1024 768
    256
    ```

15. **Type** `String -> String -> Bool`

    Description Checks whether the first string is part of the second.

    Examples

    ```
    > begins "Wit" "Ludwig Wittgenstein"
    True
    > begins "True" "Donald Trump"
    False
    ```

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

    Description Returns the successor of a binary numeral.

    Examples

    ```
    > bsucc "0"
    "1"
    > bsucc "111"
    "1000"
    ```

17. **Type** `String -> String -> String`

    Description Returns the the sum of two binaries.

    Examples

    ```
    > bplus "1" "1"
    "10"
    > bplus "100" "101"
    "1001"
    ```

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

    Description Checks whether the input is a prime.

    Examples

    ```
    > is_prime 2017
    True
    > is_prime 2019
    False
    ```

19. **Type** `Integer -> Integer`

    Description Returns the next prime.

    Examples

    ```
    > next_prime 2
    3
    > next_prime 2019
    2027
    ```

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

    Description Returns the list of the first n primes.

    Examples

    ```
    > nprimes 2
    [2,3]
    > nprimes 12
    [2,3,5,7,11,13,17,19,23,29,31,37]
    ```