Functional Programming / Funkcinis programavimas

Exercise set 3

Solutions to be sent until November 29th

Exercise 1. Extend the `Shape` datatype definition (see the slides of Lecture 6) by, for each shape, adding its position (`(x,y)` coordinates) as an extra argument or arguments.

Write a function

```
overlaps::  Shape -> Shape -> Bool,
```

that checks whether two given shapes are overlapping or not.

Exercise 2. Define your own versions for the standard functions

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

and

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

which test whether some or all list elements satisfy the given property. Please provide two versions, one relying on `filter` (very easy), and the other one relying on `map` and `foldr`.

Exercise 3. Redefine the standard function `unzip`

```
unzip::  [(a,b)] -> ([a],[b]),
```

which unzips a list of pairs into a pair of the corresponding lists, using the `foldr` function.

Exercise 4. Redefine the standard function `length` (returning the length of a list) using `map`, function composition `(.)`, and, if possible, lambda abstraction.

Also, write an alternative version of this function, which is based on folding (e.g., using `foldr`).

Exercise 5. Write a function

```
ff:: Integer -> [Integer] -> Integer,
```

which filters the given list (the second argument) by removing the negative numbers, then multiplying each number by 10, and finally adding list numbers together while their sum is not exceeding the given bound (the first argument). The function should be defined as functional composition `(.)` of the respective functions implementing the described actions, i.e.,

```
ff maxNum = ... . ... . ...
```

Exercise 6. Write a function

```
total:: (Integer -> Integer) -> Integer -> Integer
```

so that `total f` is the function which, for the given value `n`, returns

```
f 0 + f 1 + ... + f n
```

Your solution must rely either on applying `map` and functional composition or using folding (e.g., `foldr`).

Exercise 7. Define a function `iter n f` that composes the given function `f ::a -> a` with itself `n ::Integer` times, e.g., `iter 2 f = f . f`.

Give two versions of this function: one based on recursion, and the one based on the idea of first creating (by using replicate) the list of `n` copies of `f` and then folding this list. For the cases when $n \leq 0$, the Prelude function `id`, defined as `id x = x`, should be returned.

Exercise 8. Write a function

```
splits :: [a] -> [([a],[a])]
```

which returns all the ways that a list can be split into two consecutive ones, e.g.,

```
splits "Spy" == [("","Spy"),("S","py"),("Sp","y"),("Spy","")]
```