Universidade Federal do ABC
MCTA016-13 - Paradigmas de Programação
2019.Q2

Lista de Exercícios 3

Prof. Emílio Francesquini

8 de julho de 2019

1. Escreva a função McCarthy91 em Haskell

   ```
   mc91 :: Integral a => a -> a
   ```

   que é definida por:

   ```
   MC(n) =  n - 10, se n > 100;
         =  MC (MC (n + 11)), caso contrário.
   ```

2. Escreva uma função recursiva

   ```
   elem' :: Eq a => a -> [a] -> Bool
   ```

   que verifica se um valor está contido numa lista

3. Defina a função recursiva

   ```
   euclid :: Int -> Int -> Int
   ```

   que implementa o algoritmo de Euclides para calcular o máximo divisor comum de dois inteiros não-negativos: se dois números são iguais, o resultado é o próprio número, senão, o menor é subtraído do maior e o processo se repete. Exemplo:

   ```
   > euclid 6 27
   3
   ```

4. Defina a função recursiva

   ```
   concat' :: [[a]] -> [a]
   ```

que recebe uma lista de listas e devolve uma única lista contendo os elementos das listas internas.

5. Defina a função

   ```
   intersperse' :: a -> [a] -> [a]
   ```

   que recebe um char separador e uma lista de chars (ou string) e devolve uma string com os elementos intercalados entre o separador. Exemplo:

   ```
   > intersperse' '-' "abcde"
   "a-b-c-d-e"
   ```

6. Escreva a função

   ```
   wordNumber :: Char -> Int -> [Char]
   ```

   que recebe um char separador e um inteiro, e devolve uma string com os nomes dos números separados por um char sep. Você poderá utilizar as funções concat' e interspace' definidas anteriormente. Outras funções auxiliares podem ser definidas, como transformar um inteiro em uma lista de dígitos, e substituir cada dígito pelo seu correspondente por extenso. Exemplo:

   ```
   > wordNumber '-' 123
   "um-dois-três"
   ```

7. Defina uma função recursiva

   ```
   merge :: Ord a => [a] -> [a] -> [a]
   ```

   que recebe duas listas ordenadas e devolve uma única lista ordenada. Exemplo:

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

8. Utilizando a função merge implementada anteriormente, defina

   ```
   msort :: Ord a => [a] -> [a]
   ```

   que implementa o algoritmo merge sort. Para auxiliar, defina

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

   que divide uma lista em duas de mesmo tamanho ou com uma diferença máxima de um elemento.

9. Defina duas funções

```
quotRem' :: Integral a => a -> a -> (a, a)
divMod' :: Integral a => a -> a -> (a, a)
```

que implementam a divisão de inteiros como uma subtração recursiva, retornando uma tupla com o quociente da divisão e o resto. Trate o caso de divisão por zero com error.

Existem duas maneiras de se fazer a divisão com números negativos: arredondando para zero (como a função quotRem faz) ou para o infinito negativo (como feito pela função divMod). Para resolver o exercício, comece por quotRem' e use pattern matching listando todos os possíveis casos de dividendos (n), divisores (d), quociente(q) e resto (r) quanto aos sinais:

```
n = 0 <=> q = 0, r = 0
d = 0 => error
n > 0, d > 0 <=> q > 0, r > 0
n < 0, d < 0 <=> q > 0, r < 0
n < 0, d > 0 <=> q < 0, r < 0
n > 0, d < 0 <=> q < 0, r > 0
```

Para definir divMod', utilize quotRem', observando que para n e d com sinais iguais, divMod' devolve os mesmos resultados de quotRem'. Quando os sinais de n e d são opostos, deve-se subtrair 1 do quociente e somar d ao resto.

Por fim, compare seus resultados com as funções quotRem e divMod originais.

10. Defina a função

```
skips :: [a] -> [[a]]
```

cuja entrada é uma lista e a saída é uma lista de listas. A primeira lista interna é igual à lista de entrada. A segunda lista contém cada segundo elemento da lista de entrada e a enésima lista contém cada enésimo elemento da lista de entrada. Exemplos:

```
> skips "ABCD"
["ABCD", "BD", "C", "D"]
> skips "hello!"
["hello!", "el!", "l!", "l", "o", "!"]
> skips [1]
[[1]]
```

```
> skips [True,False]
[[True,False], [False]]
> skips []
[]
```