PROGRAMAÇÃO FUNCIONAL [BCC222]

Teste 2 (2017–2)
29 de novembro de 2017
Matrícula: _____
Nome: _____

Departamento de Computação
Universidade Federal de Ouro Preto
Prof. José Romildo Malaquias

1 Problema: validação de números de cartão de crédito

Alguma vez você já se perguntou como os sites validam o número do seu cartão de crédito quando você faz compras *online*? Eles não verifiquem um enorme banco de dados de números, e muito menos usam magia. Na verdade, a maioria dos provedores de crédito dependem de uma fórmula de verificação de soma (*checksum*) para distinguir entre números válidos de cartões e sequências aleatórias de dígitos (ou erros de digitação).

Nesta atividade você vai implementar o algoritmo de validação para cartões de crédito. A validação segue as etapas seguintes:

- Dobre o valor de cada segundo dígito começando pela direita. Ou seja, o último dígito não é alterado, o penúltimo dígito é dobrado, o antepenúltimo não é alterado, e assim por diante. Por exemplo, `[1,3,8,6]` torna-se `[2,3,16,6]`.
- Adicione os dígitos dos valores dobrados e não dobrados a partir do número original. Por exemplo, `[2,3,16,6]` torna-se `2 + 3 + 1 + 6 + 6 = 18`.
- Calcule o resto da divisão desta soma por 10. No exemplo acima, o resto é 8.
- O número é válido se e somente se o resultado for igual a 0.

Exercise 1

Precisamos primeiramente encontrar os dígitos de um número natural para processarmos o número do cartão de crédito.

Defina as funções

```
toDigits    :: Integer -> [Integer]
toDigitsRev :: Integer -> [Integer]
```

`toDigits` deve converter um número natural na lista dos dígitos que formam este número. Para números negativos, `toDigits` deve resultar na lista vazia.

`toDigitsRev` deve fazer o mesmo, mas com os dígitos em ordem invertida.

Exemplos:

```
toDigits 1234    ⇝ [1,2,3,4]
toDigitsRev 1234 ⇝ [4,3,2,1]
toDigits 0       ⇝ [0]
toDigits (-17)   ⇝ []
```

Para obter os dígitos decimais que formam um número natural você deverá considerar os casos:

- Se o número for menor que 10, a lista dos dígitos é uma lista unitária contendo o próprio número.
- Caso contrário, divida o número por 10. O *resto* desta divisão é o dígito menos significativo. Os demais dígitos são obtidos de maneira similar usando o quociente desta divisão. Por exemplo, se o número é 538, então o dígito menos significativo é 8 (o resto da divisão de 538 por 10), e os demais dígitos são obtidos a partir de 53 (o quociente da divisão de 538 por 10).

Lembre-se de considerar o caso do número negativo.

Dica: Primeiro defina `toDigitsRev`, e depois `toDigits` usando `toDigitsRev`.

Exercise 2

Uma vez obtidos os dígitos na ordem correta, precisamos dobrar um dígito não e outro sim, contando de trás para frente (ou seja, da direita para a esquerda, ou ainda, do dígito menos significativo para o dígito mais significativo).

Defina a função

```
doubleEveryOther :: [Integer] -> [Integer]
```

para este propósito. Lembre-se de que `doubleEveryOther` deve dobrar os números da lista de dois em dois, começando pelo penúltimo de trás para frente.

Exemplos:

```
doubleEveryOther [9,4,8,7,6,5] ⇝ [18,4,16,7,12,5]
doubleEveryOther [4,8,7,6,5]   ⇝ [4,16,7,12,5]
doubleEveryOther [1,2,3]       ⇝ [1,4,3]
```

Dica: Defina uma função auxiliar que dobra os elementos de uma lista de dois em dois, *do começo para o fim*. Isto é, o primeiro elemento não é dobrado, o segundo é, o terceiro não é, o quarto é, e assim por diante. Depois use esta função para definir `doubleEveryOther`. Neste caso será necessário inverter a lista antes e depois de chamar a função auxiliar.

Exercise 3

O resultado de `doubleEveryOther` pode conter números de um dígito ou de dois dígitos. Defina a função

```
sumDigits :: [Integer] -> Integer
```

para calcular a soma de todos os dígitos.

Exemplos:

```
sumDigits [16,7,12,5] ⇝ 1 + 6 + 7 + 1 + 2 + 5 ⇝ 22
```

Dicas:

- Observe que são os dígitos dos números que devem ser somados, e não os próprios números.
- Faça uma definição recursiva que soma os elementos da lista. Divida cada elemento da lista por dez e considere tanto o quociente quanto o resto ao efetuar a soma.

Exercise 4

Defina a função

```
validate :: Integer -> Bool
```

que indica se um número inteiro pode ser um número válido de cartão de crédito. Sua definição usará todas as funções definidas nos ítens anteriores.

Exemplos:

```
validate 4012888888881881 ⇝ True
validate 4012888888881882 ⇝ False
```

Exercise 5

Para concluir sua aplicação defina a variável `main :: IO ()` como uma açã de E/S que pemita interação com o usuário, solicitando o número do cartão de crédito e informando se o mesmo é válido ou não.