Curso: Ciência da Computação
Disciplina: Programação Funcional

ATIVIDADE PRÁTICA

Cupom fiscal do supermercado[1]

Nas tarefas que se seguem temos por objetivo desenvolver uma aplicação em Haskell para automatizar o caixa de um supermercado usando técnicas de manipulação de listas empregando funções de ordem superior. Um leitor de código de barras é usado no caixa de um supermercado para produzir uma lista de códigos de barras a partir dos produtos que se encontram em um carrinho de compras contendo os produtos comprados. Usando os códigos de barra cria-se uma nota descritiva da compra. Considere por exemplo a seguinte lista de códigos de barra:

```
[1234,4719,3814,1112,1113,1234]
```

Esta lista deve ser convertida para uma conta como mostra a figura a seguir:

```
Haskell Stores

Dry Sherry, 1lt.....5.40
Fish Fingers.....1.21
Orange Jelly.....0.56
Hula Hoops (Giant).....1.36
Unknown Item.....0.00
Dry Sherry, 1lt.....5.40

Total.....13.90
```

Primeiro devemos decidir como modelar os objetos envolvidos. Códigos de barra e preços (em centavos) podem ser representados por números inteiros, e nomes de mercadorias podem ser representados por strings. Então usaremos os seguintes tipos:

```
type Nome   = String
type Preco  = Int
type Codigo = Int
```

A conversão dos códigos de barras será baseada em um banco de dados que relaciona códigos de barras, nomes de mercadorias, e preços. Usaremos uma lista para representar o banco de dados de mercadorias:

```
type Mercadorias = [ (Codigo, Nome, Preco) ]
```

O banco de dados para o exemplo dado é:

---

[1] Exercício elaborado a partir da fonte:
MALAQUIAS, J. S. Programação funcional em haskell. BCC222: Programação Funcional. Universidade Federal de Ouro Preto. Departamento de Computação. 2017.

```
tabelaMercadorias :: Mercadorias
tabelaMercadorias = [ (4719, "Fish Fingers",       121 )
                    , (5643, "Nappies",            1010)
                    , (3814, "Orange Jelly",       56  )
                    , (1111, "Hula Hoops",         21  )
                    , (1112, "Hula Hoops (Giant)", 133 )
                    , (1234, "Dry Sherry, 1lt",    540 )
                    ]
```

O objetivo do programa é primeiramente converter uma lista de códigos de barra em uma lista de pares (Nome, Preco) por meio de uma consulta à tabela de mercadorias. Em seguida esta lista de pares deve ser convertida em uma string para exibição na tela. Para representar um carrinho de compras e uma conta (cupom fiscal) que corresponde a uma compra, usaremos as seguintes definições de tipo:

```
type Carrinho = [Codigo]
type Conta    = [(Nome,Preco)]
```

Defina uma função `formataCentavos :: Preco -> String` que recebe o preço em centavos e resulta em uma string representando o preço em reais.

Por exemplo:

```
formataCentavos 1023  ⇝ "10.23"
formataCentavos 56015 ⇝ "560.15"
formataCentavos 780   ⇝ "7.80"
formataCentavos 309   ⇝ "3.09"
formataCentavos 15    ⇝ "0.15"
formataCentavos 5     ⇝ "0.05"
```

Use as funções `div`, `mod` e `show`. Observe que ao dividir o preço em centavos por 100, o quociente corresponde à parte inteira do preço em reais, e o resto corresponde à parte fracionária do preço em reais. Preste atenção no caso do resto menor do que 10: deve-se inserir um 0 à esquerda explicitamente.

Defina uma função `formataLinha :: (Nome,Preco) -> String` que recebe um par formado pelo nome e preço de uma mercadoria e resulta em uma string representando uma linha da conta do supermercado.

Por exemplo:

```
formataLinha ("Dry Sherry, 1lt",540)⇝ "Dry Sherry, 1lt.....5.40\n"
formataLinha ("Nappies, 1lt",1010)  ⇝ "Nappies.....10.10\n"
```

O tamanho de uma linha em uma conta deve ser 30. Use a variável abaixo para representar este valor.

```
tamanhoLinha :: Int
tamanhoLinha = 30
```

Use as funções `(++)`, `show`, `length` e `replicate` do prelúdio, e a função `formataCentavos`.

A função `replicate :: Int -> a -> [a]` recebe um número inteiro n e um valor x e resulta em uma lista de comprimento n onde todos os elementos são x. Por exemplo:

```
replicate 5 13  ⇝ [13,13,13,13,13]
replicate 8 '.' ⇝ "........"
```

Defina a função `formataLinhas :: [(Nome,Preco)] -> String` que recebe uma lista de pares formados pelos nomes das mercadorias e seus respectivos preços em uma compra, e resulta na string correspondente ao corpo da conta do supermercado.

Por exemplo:

```
formataLinhas [ ("Dry Sherry, 1lt",     540)
              , ("Fish Fingers",        121)
              , ("Orange Jelly",        056)
              , ("Hula Hoops (Giant)", 136)
              , ("Unknown Item",        000)
              , ("Dry Sherry, 1lt",     540)
              ]
↝
"Dry Sherry, 1lt.....5.40\n\
\Fish Fingers.....1.21\n\
\Orange Jelly.....0.56\n\
\Hula Hoops (Giant).....1.36\n\
\Unknown Item.....0.00\n\
\Dry Sherry, 1lt.....5.40\n"
```

Use a função `formataLinha` para obter as linhas correspondentes a cada produto, e concatene estas linhas usando a função `(++)` do prelúdio. Não use recursividade explícita, mas use as funções `map` e `foldr` ou `foldl` do prelúdio. Alternativamente você poderá usar *list comprehension*.

Defina a função `formataTotal :: Preco -> String` que recebe o valor total da compra, e resulta em uma string representado a parte final da conta do supermercado.

Por exemplo:

```
formataTotal 1390 ↝ "\nTotal.....13.90"
```

Defina a função `formataConta :: Conta -> String` que recebe a lista dos itens comprados e resulta na string representando a conta do supermercado, já formatada.

Por exemplo:

```
formataConta [ ("Dry Sherry, 1lt",     540)
             , ("Fish Fingers",        121)
             , ("Orange Jelly",        056)
             , ("Hula Hoops (Giant)", 136)
             , ("Unknown Item",        000)
             , ("Dry Sherry, 1lt",     540)
             ]
```

resulta na string que é exibida pela função `putStr` como

```
Haskell Stores

Dry Sherry, 1lt.....5.40
Fish Fingers.....1.21
Orange Jelly.....0.56
Hula Hoops (Giant).....1.36
Unknown Item.....0.00
Dry Sherry, 1lt.....5.40

Total.....13.90
```

Defina a função `calculaTotal :: Conta -> Preco` que recebe uma conta (lista de pares formados pelo nome e preço das mercadorias de uma compra), e resulta no preço total da compra.

Por exemplo:

```
calculaTotal [("a",540),("b",121),("c",12)]  ⇝ 673
calculaTotal [("vinho",3540),("carne",7201)] ⇝ 10741
calculaTotal []                              ⇝ 0
```

Não use recursividade explícita, mas use as funções `map` e `sum` do prelúdio.

Defina uma função `procuraCodigo :: Mercadorias -> Codigo -> (Nome,Preco)` que recebe o banco de dados com os nomes e preços das mercadorias disponíveis no supermercado e o código de barras da mercadoria comprada, e resulta no par formado pelo nome e pelo preço da mercadoria, de acordo com o banco de dados. Se o código de barras não constar no banco de dados, o resultado deve ser o par `("Unknown Item",0)`.

Por exemplo:

```
procuraCodigo tabelaMercadorias 5643  ⇝ ("Nappies", 1010)
procuraCodigo tabelaMercadorias 9999  ⇝ ("Unknown Item", 0)
```

Defina a função `criaConta :: Mercadorias -> Carrinho -> Conta` que recebe o banco de dados com os nomes e preços das mercadorias disponíveis no supermercado, e a lista de códigos de barra correspondente a uma compra, e resulta na lista dos pares `(Nome,Preco)` para as mercadorias compradas.

Por exemplo:

```
criaConta tabelaMercadorias [3814, 5643]
⇝ [("Orange Jelly", 56), ("Nappies", 1010)]
```

Use a função `procuraCodigo` e a função `map` do prelúdio.

Defina a função `fazCompra :: Mercadorias -> Carrinho -> String` que recebe o banco de dados com os nomes e preços das mercadorias disponíveis no supermercado, e a lista de códigos de barra correspondente a uma compra, e resulta na string correspondente à nota da compra.

Use a função `criaConta` para criar a conta a partir dos argumentos, e a função `formataConta` para converter a conta para string. Use composição de funções.