Universidade Federal de Uberlândia
Faculdade de Computação
Disciplina de Programação Funcional

Trabalho 2 - Tuplas, Listas, Classes e Generalização

Professora: MSc. Júlia Tannús de Souza
Valor: 35 pontos
Entregar até dia: 19/08/2022
Forma de entrega: Arquivo .hs único, via Teams (seção Tarefas). Entrega individual.

(Adaptação do livro "Haskell: The Craft of Functional Programming" -2ª Ed. pg. 108)

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.

Você está produzindo um software para um supermercado, o qual possui uma série de caixas com leitores de código de barras. Após passar todos os itens de uma compra, cada leitor gera uma lista de códigos de barra, por exemplo:

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

Sua tarefa é, a partir desta lista, **criar um arquivo de nota fiscal**, seguindo exatamente o modelo:

```
Chocolate.................5.25
Biscoito..................10.1
Laranja...................4.6
Sabao.....................2.1
Total:....................22.05
```

Para isto, você deverá fazer uso das definições de tipo a seguir:

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

type Produto = (Codigo, Nome, Preco)
```

Também, para representar os produtos já cadastrados no estoque, use uma lista de produtos como banco de dados, por exemplo:

```haskell
tabelaProdutos :: [Produto]
tabelaProdutos = [ (001, "Chocolate", 5.25)
                 , (002, "Biscoito", 10.10)
                 , (003, "Laranja", 4.60)
                 , (004, "Sabao", 2.10)
                 , (005, "Batata Chips", 6.90)
                 , (006, "Doritos", 8.90)]
```

Para gerar a nota fiscal, são **sugeridos** os seguintes passos:

1. Faça uma função `isCodigo` que receba um código de barras e um produto e retorne um booleano, indicando se o código de barras inserido corresponde ou não ao código do produto.

   ```
   *Main> isCodigo 005 (005, "Batata Chips", 6.90)
   True
   ```

2. Faça uma função `getPreco`, que recebe um produto e retorna apenas o preço do produto.

   ```
   *Main> getPreco (005, "Batata Chips", 6.90)
   6.9
   ```

3. Faça uma função `getNome`, que recebe um produto e retorna apenas o nome do produto.

   ```
   *Main> getNome (005, "Batata Chips", 6.90)
   "Batata Chips"
   ```

4. Faça uma função `buscaPrecoPorCodigo`, que recebe um código de barras e retorna o preço do produto correspondente na tabela de produtos. **Dica:** use funções de ordem superior.

   ```
   *Main> buscaPrecoPorCodigo 005
   6.9
   ```

5. Faça uma função `buscaNomePorCodigo`, que recebe um código de barras e retorna o nome do produto correspondente na tabela de produtos.

   ```
   *Main> buscaNomePorCodigo 005
   "Batata Chips"
   ```

6. Faça uma função `calculaPrecos`, que recebe uma lista de códigos de barras e retorna a soma dos preços dos produtos correspondentes na tabela de produtos. **Dica:** use funções de ordem superior.

   ```
   *Main> calculaPrecos [001,003,005]
   16.75
   ```

7. Faça uma função `formataStrProduto`, que recebe um código de barras e retorna uma string, que representa uma linha da nota fiscal, já formatada. O tamanho da linha deve ser sempre 30.

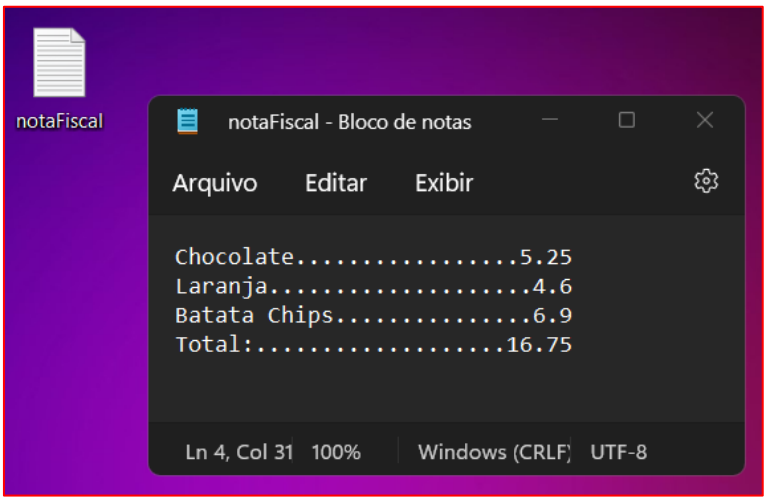
```
*Main> formataStrProduto 001
"Chocolate.................5.25\n"
```

Dica: use as funções (`++`), `show`, `length` e `replicate` do prelúdio. 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:

```
*Main> replicate 8 '.'
"........"
```

8. Faça uma função geraNotaFiscal, que recebe uma lista de códigos de barra e retorna uma ação de entrada e saída, a qual salva em arquivo texto os produtos correspondentes aos códigos de barra inseridos e o valor total da compra, segundo o modelo abaixo.

```
*Main> geraNotaFiscal [001,003,005]
```



Bom trabalho!