

UFU/FACOM

Disciplina: PF

Período: 2018/1

Ref: Atividade Prática de Programação

1 - Título: Produção de Um Recibo Fiscal em um Supermercado

2 - Objetivo: desenvolver parte de um sistema de software para uso num caixa de supermercado. Também é objetivo deste trabalho que você assimile o processo de definição de funções com listas. As funções relacionadas a listas deverão ser implementadas neste trabalho sem o uso de funções de alta ordem. A definição de funções usando funções de ordem superior é tema do próximo trabalho de programação.

3 - Descrição: um scanner no caixa produz uma lista de códigos de barra como, por exemplo,

[1234, 3216, 4719, 1112, 1113, 3814]

que deve ser convertido para uma nota impressa como:

Supermercado QLegal

Oleo Dobom, 1 lt..... 1.95

Arroz Dobom..... 2.13

Queijo Mineirim..... 4.49

Rapadura QuebraDente 0.80

Cafe Dobom 2.85

Sorvete Qgelo, 1lt..... 6.95

Total..... 19.17

4 - Desenvolvimento:

Inicialmente você deve decidir como modelar os objetos envolvidos no sistema. Os códigos de barras e preços (em centavos) podem ser modelados por inteiros e os nomes das mercadorias podem ser representados como strings. Pode-se, portanto declarar alguns tipos sinônimos úteis:

```
type Nome = String
```

```
type Preco = Int
```

```
type CodBar = Int
```

A conversão do código de barras para o recibo será baseada numa base de dados relacionando códigos de barra, nomes e preços. Este relacionamento pode ser modelado por uma lista de tuplas:

```
type BaseDeDados = [ (CodBar, Nome, Preco) ]
```

Para simplificar uma base de dados de produtos apresenta-se a especificação que segue.

ListaDeProdutos :: BaseDeDados

```
ListaDeProdutos = [ (1234, "Oleo DoBom, 1l" , 195),  
(4756, "Chocolate Cazzeiro, 250g" , 180),  
(3216, "Arroz DoBom, 5Kg", 213),  
(5823, "Balas Pedregulho, 1Kg" , 379),  
(4719, "Queijo Mineirim, 1Kg" , 449),  
(6832, "Iogurte Maravilha, 1Kg" , 499),  
(1112, "Rapadura QuebraDente, 1Kg", 80),  
(1111, "Sal Donorte, 1Kg", 221),  
(1113, "Cafe DoBom, 1Kg", 285),  
(1115, "Biscoito Bibi, 1Kg", 80),  
(3814, "Sorvete QGelo, 1l", 695)]
```

Uma decomposição simples do problema é:

- 1 - primeiro converter a lista de códigos de barras em uma lista de pares (Nome, Preço);
- 2 - converter esta lista intermediária de códigos de barra num string para visualização. Tipos sinônimos apropriados para as estruturas de dados são:

```
type ListaDeCodigos = [CodBar]  
type Recibo = [(Nome, Preço)]
```

As funções correspondentes às duas fases desta solução podem ser declaradas com tipos:

```
fazRecibo :: ListaDeCodigos -> Recibo  
formataRecibo :: Recibo -> String
```

A solução completa consiste em compor estas duas funções:

```
geraRecibo :: ListaDeCodigos -> String  
geraRecibo lc = formataRecibo ( fazRecibo lc)
```

Admita que comprimento de uma linha no recibo impresso tenha 30 caracteres.

```
tamLinha :: Int  
tamLinha = 30
```

Definindo o comprimento da linha desta forma significa que para mudar o comprimento das linhas no recibo basta alterar somente uma definição. Se usássemos 30 em cada função de formatação então cada uma deveria ser modificada se viéssemos a alterar o tamanho.

O programa final pode ser desenvolvido de uma forma uniforme e controlada por seguir os passos descritos a seguir. Certifique-se de testar cada nova definição garantindo que ela proceda corretamente para todo dado de entrada. Faça isto antes de seguir para um novo passo.

1. Dado um determinado número de centavos, digamos 1023, as partes inteiras e os centavos dados são respectivamente dados por $1023/100$ e $1023 \bmod 100$. Usando este fato defina uma função do tipo

```
formataCentavos :: Preço -> String
```

tal que:

uso: `formataCentavos 1023` gera "10.23"

2. Usando a função `formataCentavos`, defina uma função para formatar uma linha do recibo:

```
formataLinha :: (Nome,Preco) -> String
```

tal que:

uso: `formataLinha ("Arroz, 5Kg", 699)` gera "Arroz, 5Kg.....6.99\n"

Lembre-se que:

- `\n` corresponde ao caractere de nova linha
- o operador `++` pode ser usado para concatenar dois strings
- a função `length` retorna o comprimento de um string.

Algo útil na definição de `formataLinha` seria uma função que devolve um string formado pela reprodução de um caractere um certo número de vezes. Implemente esta função que deve ter como nome `replicate`. Uso: `replicate 4 'a' => "aaaa"`

Usando a função `formataLinha` defina:

```
formataLinhas :: [(Nome,Preco)] -> String
```

que aplica `formataLinha` para cada par `(Nome, Preco)` e concatena o resultado. O que resta é incluir o total e o cabeçalho.

Defina uma função:

```
geraTotal :: Recibo -> Preco
```

que dada uma lista de pares `(Nome,Preco)` devolve o total dos preços. Por exemplo,

```
geraTotal [("...",449),("...",213)] =) 662
```

Defina a função

```
formataTotal :: Preco -> String
```

tal que

```
formataTotal 662 => ""\nTotal.....$6.62".
```

Qual das funções que você já projetou será útil para construir `formataTotal`?

Usando as funções `formataLinhas`, `geraTotal` e `formataTotal`, defina a função

```
formataRecibo :: TipoDoRecibo -> String
```

tal que aplicada a

```
[("Oleo Dobom, 1 lt",195),("Arroz Dobom",213),("Queijo Mineirim",449),  
("Rapidura QuebraDente",80),("Cafe Dobom",285),("Sorvete Qgelo, 1lt",695)]
```

gera o recibo mostrado no início deste documento.

[illegible]