UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO
PRIMEIRA LISTA DE PROGRAMAÇÃO-FUNCIONAL – PARTE 1

Profa. Leila Maciel de Almeida e Silva Data de entrega: 27/06/2019

Aula: Tipos Básicos e Definições

1. Dados quatro inteiros elabore:
    a. Uma função que determina se os quatro inteiros são iguais.
    b. Usando a função `tresIguais`, vista em sala de aula, refaça o item (a).
    c. Usando definições locais `let` e depois `where` refaça o item (a).
    d. Estabeleça os casos de teste para todas as funções pedidas.
    e. Execute as funções dos itens (a) a (c) no GHCi e teste-as para os casos de teste que estabeleceu. Anexe ao arquivo com os programas outro arquivo com as imagens das telas obtidas com a execução.
2. Dados três inteiros defina:
    a. Uma função para saber se os três são diferentes.
    b. Usando (a) e a função `tresIguais` elabore uma função para saber quantos são iguais.
    c. Usando (b) elabore uma função para saber, dados quatro inteiros, quantos são iguais.
    d. Estabeleça os casos de teste para todas as funções pedidas.
    e. Execute as funções dos itens (a) a (c) no GHCi e teste-as para os casos de teste que estabeleceu. Anexe ao arquivo com os programas outro arquivo com as imagens das telas obtidas com a execução.

Aula: Tuplas e Listas

As questões a seguir devem ser feitas usando compreensões e as funções básicas sobre listas. Todas elas estão relacionadas ao cenário a seguir descrito.

3. Considere um sistema para emitir contas de um restaurante. O restaurante tem um cardápio, que pode ser alterado ao longo do tempo. O cardápio inclui todos os itens que o restaurante comercializa. Cada item tem um código único, que é um número inteiro, um nome, que é uma String, e um preço, que está armazenado em centavos e é representado por um inteiro. Por exemplo, se um item custa R$ 3,50, então é representado por 350. Assim, o sistema tem os seguintes tipos:

```
type Codigo = Int
type Nome = String
type Preco = Int
type ItemRest = (Codigo, Nome, Preco)
```

O menu é armazenado no sistema como uma lista de itens da forma:

```
type Menu = [ItemRest]
```

Um exemplo de um menu poderia ser:

```
cardapio :: Menu
cardapio = [(150, "Pastel", 1000), (15, "Agua", 400),
            (2, "Cerveja", 800), (40, "Picanha", 8850),
            (52, "Pudim", 1275)]
```

O garçom registra o pedido da mesa baseado no código do produto. O cliente pode solicitar mais de um item de um mesmo produto. Um pedido seria dado por:

```
type Mesa = Int
type Quant= Int
type ItemCliente = (Codigo, Quant)
type PedidoCliente = [ItemCliente]
```

O sistema armazena todos os pedidos internamente numa lista de pedidos das várias mesas. Cada posição da lista está associada a uma mesa do restaurante.

```
type PedidosMesas = [PedidoCliente]
```

Assim, os pedidos da mesa 4 estão na quarta posição de uma lista do tipo `PedidosMesas`. Um exemplo de uma lista deste tipo seria:

```
pedidosRest :: PedidosMesas
pedidosRest  =  [[(150,1),(2,2)],  [],  [],  [(40,1),(2,2),(52,2)],
[]]
```

Neste exemplo, os pedidos da mesa 4 seriam `[(40,1),(2,2),(52,2)]` e a mesa 2 está vazia.

Enquanto o cliente está no restaurante, ele pode chamar o garçom várias vezes para fazer novas solicitações, inclusive de itens iguais aos já solicitados, que vão sendo armazenadas na lista de pedidos da sua mesa. Quando o garçom solicita a conta da mesa, ele informa o número da mesa e o sistema coleta todos os pedidos daquela mesa, gera a conta impressa, totalizando os gastos, e substitui os pedidos daquela mesa por um pedido vazio, representando que a mesa está vaga.

Para resolver o que se pede a seguir, defina os principais tipos do sistema, um cardápio inicial e uma lista de pedidos inicial como mostrado anteriormente.

Considerando esta situação problema pede-se:

3.1 Escreva as funções a seguir para manipular o cardápio do restaurante armazenado no sistema.

(a) Adiciona um item no menu. Se o código do item já existir no menu deve retornar uma mensagem de erro sinalizado que existe um item já cadastrado para aquele código.

```
adicionaItemMenu :: Menu -> ItemRest -> Menu
```

(b) Remove um item no menu, informando seu código. Se o código do item não existir no menu deve retornar uma mensagem de erro sinalizando que não existe um item no menu para aquele código.

```
removeItemMenu :: Menu -> Codigo -> Menu
```

(c) Coleta um item no menu, informando seu código. Para simplificar, considere que esta operação só tem o caso de sucesso, ou seja, o item consultado sempre vai existir no menu.

```
coletaItemMenu :: Menu -> Codigo -> ItemRest
```

3.2 Escreva funções que permitam realizar ações que normalmente ocorrem no restaurante.

(a) Adiciona um pedido de uma mesa na lista de pedidos do restaurante. Se a lista de pedidos da mesa não for vazia, a função precisa checar se o código do item agora solicitado já existe na lista de pedidos da mesa. Em caso afirmativo, a quantidade a ele associada vai ser incrementada com a nova solicitação. Caso contrário, o novo item vai apenas ser adicionado à lista já existente. Esta função assume, por simplicidade, que o código do item a ser incluído existe no menu do restaurante.

```
adicionaPedido :: Mesa -> ItemCliente -> PedidosMesas -> PedidosMesas
```

```
adicionaPedido 1 (15,2) [[(15,1),(40,1),(52,2)], [], [(15,1), (150,1)]]
```
retornará `[[(15,3),(40,1),(52,2)], [], [(15,1), (150,1)]]`

```
adicionaPedido 1 (150,2) [[(15,1),(40,1),(52,2)], [], [(15,1), (150,1)]]
```
retornará `[[(150,2),(15,1),(40,1),(52,2)], [], [(15,1), (150,1)]]`

(b) Cancela um pedido na lista de pedidos do restaurante, para uma dada mesa. Por simplicidade, suponha que o pedido a ser cancelado existe na lista de pedidos da mesa. No ato do cancelamento, se a quantidade a ser cancelada for igual ou superior à quantidade já solicitada, o item deve ser removido da lista de pedidos da mesa. Se o cancelamento for parcial, o item permanecerá na lista, mas com a quantidade decrementada de acordo com a quantidade cancelada.

```
cancelaPedido :: Mesa -> ItemCliente -> PedidosMesas -> PedidosMesas
```

```
cancelaPedido 1 (40,1) [[(15,1),(40,1),(52,2)], [], [(15,1), (150,1)]]
```
retornará `[[(15,1),(52,2)], [], [(15,1), (150,1)]]`

```
cancelaPedido 1 (52,1) [[(15,1),(40,1),(52,2)], [], [(15,1), (150,1)]]
```
retornará `[[(15,1),(40,1),(52,1)], [], [(15,1), (150,1)]]`

(c) Gera lista completa do pedido, que será usada quando a conta for finalizada. Os nomes e os preços de cada item são coletados do menu, usando o código do

item para fazer a coleta. O preço gerado na lista de saída já é o preço do item totalizado, ou seja, o preço unitário multiplicado pela quantidade.

```
pedidoCompletoMesa :: Mesa -> PedidosMesas -> Menu ->
                                          [(Quant, Nome, Preco)]

pedidoCompletoMesa 1 [[(15,1),(40,1),(52,2)], [], [(15,1), (150,1)]]
             [(150, "Pastel", 1000), (15, "Agua", 400),
             (2, "Cerveja", 800), (40, "Picanha", 8850),
             (52, "Pudim", 1275)]
```

retornará `[(1,"Agua",400),(1,"Picanha",8800),(2,"Pudim",2550)]`

(d) Gera o total da conta a partir da lista de pedidos de uma mesa, após aplicar a função do item 3.2 (c).

```
totalMesa :: [(Quant, Nome, Preco)] -> Preco

totalMesa [(1,"Agua",400),(1,"Picanha",8800),(2,"Pudim",2550)]
```

retornará `11750`

3.3 Escreva funções para formatar a conta de uma mesa.

(a) Formata o preco unitário e preço total. Dado o valor do preço gere uma string com o valor do preço formatado. Você deve colocar duas casas decimais e justificar o preço à direita. O total previsto para o preço são 7 caracteres. Caso o valor não ocupe o 7 caracteres você deve preencher com '.'.

```
formataPreco :: Preco -> String

formataPreco 400 retorna "...4.00"
formataPreco 8800 retorna "..88.00"
```

(b) Formata uma linha da conta. A quantidade deve usar no máximo 2 caracteres, o nome do item 30 caracteres e o preço 7 caracteres. A quantidade deve vir justificada à direita deixando brancos precedendo o dígito se for o caso. O nome deve vir justificado à esquerda e se nem todos os caracteres forem usados deve ser preenchido com '.'. O preço deve ser formatado usando a função do item 3.3.(a). No final da linha deve ser adicionado '\n' para permitir pular de linha quando a conta for emitida.

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

formataPreco (1,"Agua",400)
retorna
" 1 Agua.............................4.00\n"
```

(c) Formata todas as linhas de uma conta usando a função do item (b). O efeito desta função é juntar as linhas correspondentes a todos os itens pedidos de uma mesa.

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

formataPreco [(1,”Agua”,400),(1,“Picanha”,8800),(2,”Pudim”,2550)]
```
retorna
```
“ 1 Agua.............................4.00\n
  1 Picanha.........................88.00\n
  2 Pudim...........................22.50\n”
```

(d) Formata o total da conta e usa a função 3.2 (d) para totalizar pedido. O nome Total deve vir abaixo dos nomes dos itens, deixando uma linha e branco entre os itens e ele.

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

formataTotal [(1,”Agua”,400),(1,“Picanha”,8800),(2,”Pudim”,2550)]
```
retorna
```
“\n    Total.........................114.50”
```

(fe) Gera conta. Usa várias funções anteriores e `putStr` para gerar a conta formatada.

```
geraConta:: Mesa -> PedidosMesas -> Menu -> IO ()

geraConta
  1 [[(15,1),(40,1),(52,2)], [], [(15,1), (150,1)]]
  [(150, “Pastel”, 1000), (15, “Agua”, 400), (2, “Cerveja”, 800),
   (40, “Picanha”, 8850), (52, “Pudim”, 1275)]
```

retorna
```
 1 Agua.............................4.00
 1 Picanha.........................88.00
 2 Pudim...........................22.50

   Total.........................114.50
```

(f) Libera mesa. Atualiza a lista de pedidos da mesa desocupada com a lista vazia.

```
liberaMesa:: Mesa -> PedidosMesas -> PedidosMesas
```

Para testar o seu programa, parta de um cardápio e uma lista de pedidos do restaurante e teste as diversas funções, inclusive exercitando o resultado obtido (usando `it`) durante a execução destas funções em invocações posteriores de funções. Por exemplo, você pode testar uma sequência como:

```
adicionaPedido 2 (150, 2) pedidosRest
adicionaPedido 2 (2,2) it
adicionaPedido 1 (150, 2) it
cancelaPedido 1 (150,1) it
geraConta 1 it cardápio
liberaMesa 1 pedidosRest
```