Universidade Federal do Ceará	Campus de Quixadá
Bacherelado em Ciência da Computação	
Programação Funcional	
2° Avaliação Parcial	Data 08/06/2016
Nome:	

1. (1.0) Defina recursivamente a função `descompacta :: [(a, b)] -> ([a], [b])` que transforma uma lista de pares ordenado em um par ordenado onde o primeiro elemento é uma lista dos primeiros componentes dos pares ordenados e o segundo elemento é uma lista dos segundos componentes dos pares ordenados.

```
descompacta [(1,2),(3,4),(5,6),(4,5)] == ([1,3,5,4],[2,4,6,5])
descompacta [(1,2),(3,4),(5,6),(4,5),(5,6)] == ([1,3,5,4,5],[2,4,6,5,6])
```

2. (1.0) Implemente uma função

```
type Item = String
total :: [(Item,Double)] -> [Item] -> Double
```

que recebe uma lista de pares (item, valor) que associa cada item ao seu valor e uma lista de itens devolvendo o somatório dos valores dos itens passados como parâmetros.

```
preco =
 [("Leite",2.0),
 ("Manteiga", 2.5),
 ("Batata",4.0),
 ("Brocolis", 2.0),
 ("Cenoura",2.2)
 ]
```

Exemplos:

```
*Main> total preco ["Cenoura"]
2.2
*Main> total preco ["Cenoura","Leite"]
4.2
*Main> total preco ["Cenoura","Leite","Leite"]
6.2
```

3. (1.0) Uma lista é uma sublista de uma outra se os elementos da primeira ocorrem na segunda, de maneira contígua. Por exemplo, "abcd"é uma sublista de "XYabcd".

   (a) Escreva a função `subLista :: [a] -> [a] -> Bool` tal que subLista xs ys decide se xs é uma sublista de ys.

Dica: Defina uma função `isPrefix :: [a] -> [a] -> Bool` tal que (isPrefix xs ys) verifica se xs é um prefixo de ys.

4. (1.0) Considere o seguinte tipo de dados, modelando proposições lógicas em Haskell:

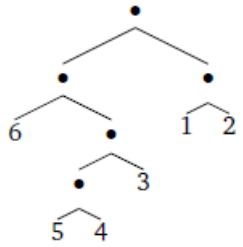
```
data Prop
 = And Prop Prop
 | Or Prop Prop
 | Not Prop
 | Val Bool

prop1 = (And (Val True) (Or (Val False) (Val True)))
prop2 = (Not (And (Val True) (Val False)))
```

Implemente uma função `eval :: Prop -> Bool` que avalia a proposição dada devolvendo um valor booleano.

Exemplo:

```
*Main> eval prop1
True
*Main> eval prop2
True
```

5. (2.0) A árvore binária é uma das estrutura de dados mais usadas. Nesta modelagem de árvore binária, o conteúdo da árvore é armazenado apenas nas folhas. Uma árvore binária é formada por nó Folha que contém um valor do tipo `a` ou nó Ramo formado por duas subárvores.



   (a) (0.5) Defina um tipo de dado (Arvore a) para representar uma árvore binária em que cada nó é um Folha ou um Ramo.

   (b) (0.5) Usando seu tipo de dado, crie a árvore binária mostrada na figura acima e associe a uma variável arv :: Arvore Int.

   (c) (1.0) Defina a função `foldTree :: (a->b) -> (b->b->b) -> Tree a -> b` que recebe duas função como argumento, a primeira deve ser chamada na folhas e a segunda deve ser chamada para os nós internos.

   Exemplo:

```
foldTree (\x -> x) (+) arv == 21
```

6. (1.0) Considere a seguinte definição para uma árvore binária:

```
data ArvBin a
  = Vazia
  | No a (ArvBin a) (ArvBin a)
  deriving (Eq,Show)
arv1 =
  No 4
    (No 3 Vazia Vazia)
    (No 5
        (No 6 Vazia Vazia)
        (No 4 Vazia Vazia)
    )
arv2 =
  No 7
    (No 4
        (No 5 Vazia Vazia)
        Vazia
    )
    (No 3
        (No 2 Vazia Vazia)
        Vazia
    )
```

   (a) Escreva a definição de `zipAB :: ArvBin a -> ArvBin b -> ArvBin (a,b)` que recebe duas árvores binárias e devolve uma árvore binária de pares ordenados com elementos das duas árvores binárias.

```
*Main> zipAB arv1 arv2
No (4,7) (No (3,4) Vazia Vazia) (No (5,3) (No (6,2) Vazia Vazia) Vazia)
```

7. (1.0) Considere o seguinte tipo de dado recursivo que representa estradas:

```haskell
data Estrada
 = Cidade String
 | Bifurcacao Estrada Estrada
```

`Cidade c` representa uma estrada que vai diretamente para a cidade c.

`Birfucacao esq dir` representa uma estrada que virando para esquerda leva para uma estrada *esq* e virando para a direita leva a uma estrada *dir*.

Aqui um exemplo de uma estrada correspondendo ao mapa:

```haskell
estrada :: Estrada
estrada =
 Bifurcacao
  ( Cidade "Quixada" )
  ( Bifurcacao
   ( Bifurcacao
    ( Cidade "Quixeramobim" )
    ( Cidade "Senador Pompeu" )
   )
  ( Cidade "Madalena" )
 )
```

Agora, implemente a função `alcanca :: String -> Estrada -> Bool` que checa se uma dada cidade é alcançável pela estrada dada.

Exemplo:

```
*Main> alcanca "Quixada" estrada
True
*Main> alcanca "Quixeramobim" estrada
True
*Main> alcanca "Pedra Branca" estrada
False
```

Dicas: Uma vez que toda cidade em uma `Estrada` pode ser alcançada através da raiz da estrada, checar se uma cidade é alcançável é o mesmo que checar se ela está incluída na `Estrada`.

8. (2.0) Enoque está implementando um sistema de navegação de carros para dispositivos Android. Como representação interna, ele decidiu usar o tipo recursivo `Estrada`. Uma parte da funcionalidade do sistema é ser capaz de apresentar uma lista de direções dada uma posição atual e um chegada. A posição atual do carro é representada por um valor do tipo Estrada e o resultado da navegação do sistema é uma lista de direções, onde cada direção por ser esquerda ou direita:

```haskell
data Direcao = Esq | Dir deriving (Show)
```

Implemente a função `mapaRodoviario :: String -> Estrada -> Maybe [Direcao]` que gera uma lista de direções para alcançar uma cidade dada:

A chamada da função tem a seguinte a forma `mapaRodoviario cidade pos`, onde `cidade` é cidade de chegada e `pos` é a posição atual do carro.

Exemplo:

```
*Main> mapaRodoviario "Quixada" estrada
Just [Esq]
*Main> mapaRodoviario "Quixeramobim" estrada
Just [Dir,Esq,Esq]
*Main> mapaRodoviario "Senador Pompeu" estrada
Just [Dir,Esq,Dir]
```

Dica: use uma expressão case

```haskell
case mapaRodoviario chegada esq  of
 Just xs ->
 Nothing ->
```

9. (1.0) A Linguagem de Marcação de Hipertexto, melhor conhecida como HTML, é uma linguagem para descrever documentos. Documentos escritos em HTML tem uma estrutura que é determinada pelo uso de tags. Para colocar partes de um documento em tags, nós usamos tags para abrir e para fechar. Por exemplo:

   - Para colocar uma parte do texto em negrito usamos as tags `<B>...</B>` indicando que o texto deve está em negrito.
   - Para colocar uma parte do texto em itálico , usamos as tags `<EM>...</EM>` indicando que o texto deve está enfatizado.
   - Para colocar uma parte do texto em parágrafo, usamos as tags `<P>...</P>` indicando que o texto forma um parágrafo (uma linha em branco antes e depois)

   Um exemplo de código HTML:

   Bem-vindo a minha página!`<P><B>`Meus interesses são programação em `<EM>`Haskell`</EM>` e assistir `<EM>`Prison Break`</EM>`.`</B></P><P>`Obrigado por visitar! `<EM>` pedro@gmail.com`</EM></P>`

   Aqui está como ele parecia no navegador:

   Bem-vindo a minha página!

   Meus interesses são programação em *Haskell* e assistir *Prison Break*.

   Obrigado por visitar! *pedro@gmail.com*

   Nós podemos representar documentos HTML em Haskell da seguinte maneira. Primeiro, um documento é uma lista de partes de documentos.

```
type Doc = [DocPart]
```

   Existem dois tipos de partes de documentos:

   - Um texto
   - Um texto colocados em tags

```
data DocPart = Texto String
    | Tag String Doc
```

   Um exemplo de código de HTML:

```
pagina :: Doc
pagina =
 [ Texto "Bem-vindo a minha pagina!"
 , Tag "P" [ Tag "B" [ Texto "Meus interesses sao programacao em"
                     , Tag "EM" [ Texto "Haskell" ]
                     , Texto " e assistir "
                     , Tag "EM" [ Texto "Prison Break" ]
                     , Texto "."
                     ] ]
 , Tag "P" [ Texto "Obrigado por visitar!"
           , Tag "EM" [ Texto "pedro@gmail.com" ]
           ]
 ]
```

   (a) Defina uma função `showDoc :: Doc -> String` que produz uma String contendo o código HTML de um dado documento. Por exemplo,

```
*Main> showDoc pagina
"Bem-vindo a minha pagina!<P><B>Meus interesses sao programacao em <EM>Haskell</
EM> e assistir <EM>Prison Break</EM>.</B></P><P>Obrigado por visitar!<EM>pedro@g
mail.com</EM></P>"
```