Curso: Ciência da Computação
Disciplina: Programação Funcional

EXERCÍCIOS – LISTA 05

1) Crie o tipo Pergunta com os values constructors *Sim* ou *Nao*. Faça as funções seguintes, determinando seus tipos explicitamente.

   a) pergNum : recebe via parâmetro uma Pergunta . Retorna 0 para *Nao* e 1 para *Sim*. listPergs : recebe via parâmetro uma lista de Perguntas , e retorna 0 s e 1 s correspondentes aos constructores contidos na lista.
   b) and' : recebe duas Perguntas como parâmetro e retorna a tabela verdade do and lógico, usando Sim como verdadeiro e Nao como falso.
   c) or' : idem ao anterior, porém deve ser usado o ou lógico.
   d) not' : idem aos anteriores, porém usando o not lógico.

2) Faça o tipo Temperatura que pode ter valores Celsius, Farenheit ou Kelvin. Implemente as funções:

   a) converterCelsius : recebe um valor double e uma temperatura, e faz a conversão para Celsius.
   b) converterKelvin : recebe um valor double e uma temperatura, e faz a conversão para Kelvin.
   c) converterFarenheit : recebe um valor double e uma temperatura, e faz a conversão para Farenheit.

3) Implemente uma função que simule o vencedor de uma partida de pedra, papel e tesoura usando tipos criados. Casos de empate devem ser considerados em seu tipo.

4) Faça uma função que retorne uma string, com todas as vogais maiúsculas e minúsculas eliminadas de uma string passada por parâmetro usando list compreenshion.

5) Faça um novo tipo chamado Mes, que possui como valores todos os meses do ano. Implemente:

   a) A função checaFim , que retorna o número de dias que cada mês possui (considere fevereiro tendo 28 dias).
   b) A função prox, que recebe um mês atual e retorna o próximo mês.
   c) A função estacao, que retorna a estação do ano de acordo com o mês e com o hemisfério.

6) Faça uma função que receba uma String e retorne True se esta for um palíndromo; caso contrário, False.

7) Construa um tipo algébrico Valor para representar valores dos tipos ValorInt, ValorFloat, ValorString, ValorChar, onde cada construtor deve ter um valor que depende do seu tipo (Ex: ValorFloat deve armazenar um valor do tipo Float).
Faça também uma função plus :: Valor -> Valor -> Valor que realiza uma soma genérica definida da seguinte forma:

   a) A aplicação de ValorString com qualquer outro Valor deve retornar um ValorString concatenando-se os dois valores;
   b) A aplicação de ValorChar com outro ValorChar deve retornar um ValorString correspondendo à concatenação dos dois caracteres;
   c) A aplicação de um ValorChar com um ValorFloat ou ValorInt deve retornar o respectivo tipo numérico adicionado do valor do char na tabela ASCII;
   d) A aplicação de um ValorFloat com um ValorInt deve retornar um ValorFloat da soma dos valores;
   e) A aplicação de valores de mesmo tipo deve retornar um novo valor deste tipo.

   Exemplos:

```
Prelude> plus (ValorChar 'P') (ValorInt 24)
ValorInt 104
Prelude> plus (ValorInt 10) (ValorFloat 12.3)
ValorFloat 22.3

Prelude> plus (ValorString "PLP") (ValorFloat 3.7)
ValorString "PLP3.7"
```

   Obs.: Lembre-se de incluir "deriving Show" (sem as aspas) no final da definição do tipo algébrico. Dessa forma o GHCi consegue mostrar os valores resultantes para você.

8) Um grafo é uma estrutura de dados baseada em uma abstração matemática bastante popular. Grafos são usados em diferentes nichos de aplicação como implementação de mapas inteligentes, gerenciamento de tráfego em rede e interpretação de linguagens funcionais. Um grafo é um par consistindo em um conjunto de vértices e um conjunto de arestas. Cada aresta conecta um par de vértices. Diz-se que que dois vértices conectados por uma aresta são adjacentes. Em Haskell, é possível definir um grafo como um par:

```
type Vertice = Int
type Aresta = (Int, Int)
type Grafo = ([Vertice], [Aresta])
```

Neste exercício consideraremos grafos direcionados, ou seja, se um par (v1, v2) pertence à lista de arestas de um grafo, isso significa que há uma aresta de v1 para v2 mas não o contrário, ou seja, dizemos que v2 é um vizinho de v1 mas a recíproca não é verdadeira. Implemente uma função que recebe como entrada um grafo e produz como resultado uma lista de tuplas de dois elementos (V, VS), onde V é um nó do grafo e VS é a lista de todos os nós vizinhos de V no grafo.

Exemplo:
Prelude> vizinhos ([1,2,3,4,5], [(1,2), (3,4), (1,4), (2, 1), (2,3), (2, 4), (5, 3), (4, 2)])
[(1, [2, 4]), (2, [1, 3, 4]), (3, [4]), (4, [2]), (5, [3])]