Graduação em Ciência da Computação — DECOM/ICEB/UFOP
Programação funcional — 1º semestre de 2023
Professor: Rodrigo Geraldo Ribeiro
Primeira Avaliação — 14 de junho de 2023.

Nome: _____

Simulando um sistema de arquivos

O objetivo dessa prova é a implementação de um conjunto de funções para simular algumas características presentes em sistemas de arquivos modernos. Inicialmente, vamos definir tipos de dados para representar nomes (tipo Name) e o conteúdo (tipo Data) de arquivos.

```
type Name = String
type Data = String
```

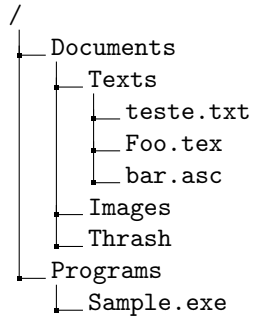
Os componentes básicos de um sistema de arquivos são arquivos e diretórios. Representaremos esses itens utilizando o seguinte tipo de dados algébrico:

```
data Item
  = File Name Data
  | Dir Name [Item]
  deriving Eq
```

Em que o construtor File representa um arquivo e é composto por um nome e os dados armazenados neste arquivo. Por sua vez, o construtor Dir representa um diretório que é formado por um nome e os items armazenados neste diretório. Finalmente, um sistema de arquivos é apenas uma lista de itens.

```
type FileSystem = [Item]
```

Como um exemplo de valor do tipo FileSystem, considere a seguinte estrutura de diretórios:



Essa mesma estrutura de arquivo pode ser representada pelo seguinte valor de FileSystem:

```
root :: FileSystem
root = [
  Dir "Documents"
    [
      Dir "Texts"
        [
          File "teste.txt"
            "12345"
        , File "Foo.tex" "45678"
        , File "bar.asc" ""
        ]
    , Dir "Images" []
    , Dir "Thrash" []
    ]
  , Dir "Programs"
    [File "Sample.exe" "1010111101"]
  ]
```

Com base no apresentado, faça o que se pede.

1. (**Valor:** 2,0 pts.) Desenvolva a função

   ```
   name :: Item → Name
   ```

   que retorna o nome de um determinado item do sistema de arquivos. Os seguintes casos de teste ilustram o comportamento esperado desta função.

   ```
   name (File "teste.txt" "12345") ≡ "teste.txt"
   name (Dir "Images" []) ≡ "Images"
   ```

2. (**Valor:** 2,5 pts.) Desenvolva a função

   ```
   directories :: FileSystem → [Item]
   ```

   que retorna a lista de diretórios presentes em um sistema de arquivos fornecido como argumento. O seguinte caso de teste ilustra o comportamento esperado dessa função quando aplicada ao valor de exemplo root:

   ```
   map name (directories root) = ["Documents", "Texts", "Images", "Trash", "Programs"]
   ```

3. (**Valor:** 2,5 pts.) Desenvolva a função

   ```
   files :: FileSystem → [Item]
   ```

   que retorna a lista de arquivos presentes em um sistema de arquivos fornecido como argumento. O seguinte caso de teste ilustra o comportamento esperado dessa função quando aplicada ao valor de exemplo root:

   ```
   map name (files root) = ["teste.txt", "Foo.tex", "bar.asc", "Sample.exe"]
   ```

4. (**Valor:** 3,0 pts.) O objetivo desta questão é o desenvolvimento de uma função para calcular o consumo em bits do sistema de arquivos. Para isso, consideraremos dois tipos de arquivos:

   (a) Arquivos binários: Arquivos cujo conteúdo é formado apenas por uma string de bits. Exemplo: File `"ls"` `"1100010101"`. O tamanho deste arquivo é de 10 bits pois, é o tamanho da string de dados deste arquivo.

   (b) Arquivos convencionais: Arquivos cujo conteúdo é formado por caracteres quaisquer. Exemplo: File `"foo.txt"` `"abc"`. Para determinar o tamanho em bits deste arquivo, basta considerar que cada caractere é representado por 8 bits. Como seu conteúdo é formado por 3 caracteres, seu tamanho deve ser 24 bits.

   Com base no apresentado, desenvolva a função:

   ```
   totalSize :: FileSystem → Int
   ```

   que calcula o tamanho em bits de um sistema de arquivos fornecido como argumento. O seguinte caso de teste ilustra o resultado esperado desta função quando aplicada ao valor de exemplo root:

   ```
   totalSize root ≡ 90
   ```

5. (**Questão extra:** Valor 4,0 pts.) Um caminho é representado por uma string que utiliza um separador especial.

   ```
   type Path = String
   sep :: String
   sep = "/"
   ```

   Desenvolva a função

   ```
   paths :: FileSystem → [Path]
   ```

   que produz uma lista contendo o caminho completo de todos os arquivos presentes no sistema de arquivos fornecido como argumento de entrada. Como exemplo do comportamento desta função, considere o seguinte caso de teste:

   ```
   paths root ≡ ["/Documents/Texts/teste.txt"
     ,"/Documents/Texts/Foo.tex"
     ,"/Documents/Texts/bar.asc"
     ,"/Programs/Sample.exe"]
   ```