Ficha Prática 4

Programação Funcional CC

LCC 1^o ano (2007/2008)

Resumo

Nesta ficha pretende-se trabalhar com tipos de dados indutivos. Em particular utilizar-se-á o tipo `Maybe a`, e tipos definidos pelo utilizador com mais do que um construtor. Finalmente trabalhar-se-á com tipos *recursivos* definidos pelo utilizador.

Os casos de estudo correspondem à implementação de *tabelas*, ou *funções finitas*, em Haskell, e ainda à representação de expressões aritméticas.

Conteúdo

1 Implementação de uma Tabela por uma Lista

Considere que se representa informação relativa à avaliação dos alunos inscritos numa disciplina, que tem no seu sistema de avaliação uma componente teórica e uma componente prática, pelos seguintes tipos de dados. Observe-se que cada estudante pode ter ou não já obtido nota numa das componentes da disciplina (teórica ou prática), o que se representa recorrendo a tipos `Maybe`.

```
type Nome = String
type Numero = Int
type NT = Maybe Float
type NP = Maybe Float

data Regime = Ordinario | TrabEstud
   deriving (Show, Eq)

type Aluno = (Numero, Nome, Regime, NT, NP)

type Turma = [Aluno]
```

Tarefa 1

Defina as seguintes funções:

1. `pesquisaAluno :: Numero -> Turma -> Maybe Aluno` *que efectua a pesquisa de um(a) aluno(a) pelo seu número de estudante. Observe que este é um identificador apropriado, uma vez que todos os alunos têm números diferentes.*

2. `alteraNP :: Numero -> NP -> Turma -> Turma` *e* `alteraNT :: Numero -> NT -> Turma -> Turma`*, que permitem alterar as notas prática e teórica de um(a) estudante numa turma.*

3. `notaFinal :: Numero -> Turma -> Maybe Float` *que calcula a nota final de um aluno segundo a fórmula* $NF = 0.6NT + 0.4NP$*. Note que só é possível calcular esta nota caso o número fornecido exista na turma, e ambas as componente* NT *e* NP *atinjam a nota mínima de 9.5 valores.*

4. `trabs :: Turma -> Turma` *que filtra os alunos trabalhadores-estudantes apenas.*

5. `aprovados :: Turma -> [(Nome, Float)]` *que apresenta as notas de todos os alunos aprovados, i.e. com nota final superior ou igual a 10 valores.*

Tarefa 2

A pesquisa nesta tabela torna-se mais eficiente se se mantiver a informação ordenada. Sendo a chave de pesquisa o número de aluno, a informação dever-se-á manter ordenada segundo estre critério.

1. *Escreva uma função de inserção ordenada de novos alunos numa turma,* `insere :: Aluno -> Turma -> Turma`*.*

2. *Redefina a função de pesquisa da tarefa anterior, tendo em conta que a tabela é representada por uma lista ordenada.*

Tarefa 3

Pretende-se agora distinguir os alunos que frequentam a disciplina pela primeira vez dos restantes alunos. Estes últimos poderão ter uma nota prática "congelada", obtida no ano lectivo anterior, mas poderão optar por ser de novo avaliados na componente prática no ano actual, devendo ser guardadas ambas as notas. A nota prática final será a melhor das duas.

Utiliza-se para isto o seguinte tipo de dados definido pelo utilizador, com dois construtores:

```
data Aluno = Primeira (Numero, Nome, Regime, NT, NP)
           | Repetente (Numero, Nome, Regime, NT, NP, NP)
  deriving Show
```

Reescreva todas as funções das tarefas anteriores tendo em conta esta alteração no tipo `Aluno`*.*

2 Implementação de uma Tabela por uma Árvore Binária de Pesquisa

Relembre que estas árvores se caracterizam pela seguinte propriedade (invariante de ordem): o conteúdo de qualquer nó situado à esquerda de um nó X é necessariamente menor do que o conteúdo de X, que por sua vez é necessariamente menor do que o conteúdo de qualquer nó situado à sua direita. Admitindo-se a ocorrência de elementos iguais, uma destas restrições é relaxada para "menor ou igual".

Considere agora a seguinte definição de um tipo de dados polimórfico para árvores binárias:

```
data BTree a = Empty | Node a (BTree a) (BTree a)
  deriving Show
```

Para se utilizar uma árvore binária para implementar uma tabela de alunos, basta redefinir:

```
type Turma = BTree Aluno
```

Tarefa 4

Escreva as seguintes funções, cujo significado é o mesmo considerado na secção anterior desta ficha:

1. `insere :: Aluno -> Turma -> Turma`
2. `pesquisaAluno :: Numero -> Turma -> Maybe Aluno`
3. `alteraNP :: Numero -> NP -> Turma -> Turma` *e*
   `alteraNT :: Numero -> NT -> Turma -> Turma`
4. `notaFinal :: Numero -> Turma -> Maybe Float`
5. `trabs :: Turma -> [Aluno]` *(note que os alunos trabalhadores-estudantes são aqui apresentados numa lista)*
6. `aprovados :: Turma -> [(Nome, Float)]`

3 Árvores de Expressão

Considere os seguintes tipos de dados utilizados para a representação de expressões aritméticas por árvores binárias:

```
data OP = SOMA | SUB | PROD | DIV
  deriving (Show, Eq)
data Expr = Folha Int | Nodo OP Expr Expr
  deriving (Show, Eq)
```

Tarefa 5

1. *Escreva uma função* ***aplica*** *que aplica um operador binário a dois argumentos inteiros:*

   ```
   aplica :: OP -> Int -> Int -> Int
   ```

2. *Escreva uma função* ***avalia*** *que procede ao cálculo do valor de uma expressão:*

   ```
   avalia :: Expr -> Int
   ```

3. *Escreva finalmente uma função* ***imprime*** *que produz uma string com a representação usual de uma expressão representada por uma árvore:*

   ```
   imprime :: Expr -> String
   ```