Paradigmas da Programação I (LESI)
Programação Funcional (LMCC)

Ficha 5

Ano lectivo 2004/05

Entrada/Saida

Quando realizamos uma aplicação, necessitamos de "executar" operações de *Entrada/Saida* de dados. Estas operações escapam à identificação realizada no paradigma funcional de "execução de um programa" como o "cálculo do valor de uma expressão" — pretende-se antes especificar *acções* que devem ser realizadas numa dada sequência. Como exemplos de operações *Entrada/Saida* podemos citar:

- Ler um valor do teclado;
- Escrever uma mensagem no ecrán;
- Ler/escrever um ficheiro com dados;

Em *Haskell*, a integração destas operações é realizada por intermédio do *Monad IO*. Podemos entender o *monad IO* como uma marca que assinala que um valor de um dado tipo foi obtido fazendo uso de operações de entrada/saida. Assim, um valor do tipo `IO Int` pode ser entendido como "um programa que realiza operações entrada/saida e retorna um valor do tipo `Int`" (ver `leInt` no exemplo apresentado abaixo, onde é retornado um valor inteiro lido do teclado). Ora, esta distinção entre valores "puros" do tipo `Int`, e os valores obtidos por intermédio de operações entrada/saida (tipo `IO Int`) coloca um problema evidente: se tivermos uma função que opere sobre inteiros (por exemplo, a função `fact` apresentada abaixo), ela não pode ser directamente aplicada a `leInt::IO Int`. Como podemos então calcular o factorial de um valor introduzido no teclado? A resposta a esta questão encontra-se nas operações que caracterizam um *Monad* (estudadas na teórica) — na prática, é preferível utilizar a notação `do` disponibilizada pela linguagem *Haskell*.

Chamemos *computação* às expressões cujo calculo envolve operações entrada/saida (i.e. expressões do tipo `IO t`, qualquer que seja o tipo `t`). A notação `do` permite definir uma computação como a sequenciação de um conjunto de computações (o valor retornado é o da última). Mas nesta sequência vamos poder aceder aos valores que vão sendo calculados. Mais precisamente:

- a seta `<-` permite-nos aceder ao valor retornado pela computação correspondente. No exemplo apresentado abaixo, na linha `l<-getLine`, temos que `getLine::IO String` (é uma função pré-definida que lê uma linha do teclado). Assim, `l` será do tipo `String` e correponderá ao texto introduzido pelo utilizador;
- a operação `return` permite-nos embeber um valor numa computação. Ainda no exemplo apresentado, `((read l)::Int)` permite "ler" o inteiro da string `l` (a operação inversa do `show`). Assim, `return ((read l)::Int)` corresponde à computação que retorna esse valor.

```
leInt :: IO Int
leInt = do putStr "Escreva um número: "
           l <- getLine
           return ((read l)::Int)

fact :: Int -> Int
fact 0 = 1
fact n = n * fact (n-1)

prog1 :: IO ()
prog1 = do x <- leInt
           putStrLn ("o factorial de "++(show x)++" é "++(show (fact x)))
```

Finalizamos esta apresentação com a referência a algumas das funções mais uteis que envolvem o monad IO:

`putStr :: String -> IO ()` — escreve a `String` dada no ecrá;

`putStrLn :: String -> IO ()` — identica à anterior, mas muda de linha (equivalente a concatenar a string `"\n"` após a string dada);

`getLine :: IO String` — lê uma linha do teclado;

`getChar :: IO Char` — lê um único carácter;

`readFile :: FilePath -> IO String` — lê o conteúdo de um ficheiro para uma string (obs.: `FilePath` é um sinónimo de tipo de `String`);

`writeFile :: FilePath -> String -> IO ()` — escreve o conteúdo de uma string num ficheiro.

Tarefa 1

Considere as seguintes declarações de tipo para uma versão simplificada do problema apresentado na Tarefa 2 da Ficha 4:

```
type Nome = String
type Numero = Integer
type NPratica = Float

data NTeorica = Faltou  | Nota Float
           deriving (Eq,Show)

type Aluno = (Numero, Nome, NTeorica, NPratica)


type PP1PF = [Aluno]
```

1. Defina um valor do tipo PP1PF contendo a informação dos seguintes alunos:

Numero	Nome	Nota Teórica	Nota Prática
1111	Jose	12	14
2222	Maria	Faltou	15
3333	Joao	18	19

2. Pretende-se que a informação referida na alínea anterior seja guardada no ficheiro `XXX.txt` como um valor (`tabela`) do tipo String com a seguinte forma:

   ```
   tabela =‘‘1111\tJose\tN12\t14\n2222\tMaria\tF\t15\n3333\tJoao\tN18\t19\n’’
   ```

Note a forma como se convencionou guardar a informação sobre as notas teóricas: se um aluno faltou regista-se "F" se um aluno teve uma nota `x` regista-se `‘‘Nx’’`.
Teste no interpretador a seguinte comando:

```
writeFile ‘‘XXX.txt’’ tabela
```

3. Verifique o conteúdo do ficheiro gerado, fazendo o comando `cat XXX.txt` numa shell do sistema operativo

4. Pretende-se agora ler a informação guardada no ficheiro `XXX.txt` para uma estrutura de dados do tipo `PP1PF`. Para tal definiram-se as seguintes funções em *Haskell*:

```
geraNT :: String -> NTeorica
geraNT "F" = Faltou
geraNT ('N':s)= Nota ((read s)::Float)

geratabela :: [[String]] -> PP1PF
geratabela [] = []
geratabela ([nu, no, nt, np]:t) =
   let nu'= (read nu)::Integer
       nt'= geraNT nt
       np'= (read np)::Float
    in (nu',no,nt',np'):(geratabela t)

loadTabela :: IO PP1PF
loadTabela = do s <- (readFile "XXX.txt")
                return (geratabela  (map words (lines s)))
```

   Teste este programa (sugestão: faça `t <- loadTabela` e confirme o valor de `t`).

5. Defina uma função que guarde em ficheiro a informação guardada numa estutura do tipo `PP1PF`.

6. Adicionar a informação de um aluno (lendo os dados a partir do teclado);

7. Defina a função `listagemNotas` que, dado um valor do tipo `PP1PF`, produza uma listagem com o seguinte aspecto e a apresente no ecrá:

```
Numero    Nome     Teorica    Pratica

1111      Jose     12         14
2222      Maria    Faltou     15
3333      Joao     18         19
```

Projecto Prático (continuação)

Pretendemos agora que enriqueça o projecto apresentado na ficha anterior, com operações de entrada/saída de dados.

Sugere-se o seguinte conjunto mínimo de operações:

- Leitura da informação sobre o campeonato a partir de ficheiros **(*)**.
- Consulta de informação sobre uma prova.
- Consulta de informação sobre um concorrente numa prova.
- Listagem de informação sobre um concorrente no total das provas.
- Listagem das classificações dos corredores numa prova.
- Listagem das classificações das equipa numa prova.
- Listagem das classificações dos corredores na totalidade das provas.
- Listagem das classificações das equipas na totalidade das provas.
- Listagem dos prémios obtidos por um corredor na totalidade das provas.
- Permitir que as listagens possam ser feitas para ficheiros.

Para além destas operações muitas outras poderão ser disponibilizadas (para valorizar o trabalho), nomeadamente, operações de análise estatistica sobre o campeonato e os corredores.

Apresente uma versão compilada do programa.

(*) Sugere-se que a informação seja guardada em tabelas (em ficheiros) com os seguintes items de informação:

Tabela de Corredores : Numero-concorrente, Nome , Equipa.
Regista informação sobre todos os corredores envolvidos na competição.

Tabela de Provas : Prova, Lista de pares (Lugar, Prémio).
Associa a cada prova informação sobre os prémios a atribuir nessa prova. Para simplificar, sugere-se que o prémio seja apenas função do lugar obtido na prova.

Tabela de Classificações : Prova, Numero-concorrente, Tempo ou Tempo e Km percorridos.
Para cada prova e para cada concorrente, regista-se o tempo obtido nessa prova ou, caso não termine a prova, o tempo em que desistiu e os Km percorridos enquanto esteve em prova. Note que no cálculo das classificações dos corredores que terminaram uma prova, o melhor resultado corresponde ao menor tempo obtido. No cálculo das classificações dos corredores que não terminaram a prova, o melhor resultado corresponde ao maior número de Km percorridos e, em caso de igualdade, ao melhor tempo em prova.

NOTA: Pode sempre simplificar o projecto, ou enriquece-lo com mais informação e novas funcionalidades. Por isso, pode adaptar as tabelas à sua solução. Na apresentação do projecto não se esqueça de trazer ficheiros de teste (de alguma dimensão).