Trabalho Prático nº 2

Métodos de Programação I

2004/2005

1 Introdução

Um problema comum a vários sistemas informáticos é a manutenção da informação referente a uma determinada *chave*. Existem várias soluções para abordar esse problema e uma forma de abstrair o problema consiste em definir uma *classe* onde se explicitam os vários *métodos* associados à manipulação dessa informação.
Tomemos por exemplo a seguinte definição

```
class BD bd where
  vazia      :: bd ch inf
  acrescenta :: (Eq ch) => bd ch inf -> ch -> inf -> bd ch inf
  remove     :: (Eq ch) => bd ch inf -> ch -> bd ch inf
  chaves     :: bd ch inf -> [ch]
  definida   :: (Eq ch) => bd ch inf -> ch -> Bool
  daInfo     :: (Eq ch) => bd ch inf -> ch -> Maybe inf
```

Uma forma muito simples de implementar estas correspondências é usando listas de pares. Essa solução corresponde às seguintes definições:

```
data LP c i = LP [(c,i)]

instance BD LP where
  vazia = LP []
  acrescenta b c i = let (LP b') = remove b c in LP ((c,i):b')
  remove (LP l) c = LP [(a,b) | (a,b) <- l, a /= c]
  chaves (LP l) = map fst l
  definida b c = c `elem` (chaves b)
  daInfo (LP l) c = case [(a,b) | (a,b) <- l, a == c] of
                      (_,i):_ -> Just i
                      _ -> Nothing
```

Note que, todos os métodos da classe têm como argumento a base de dados e que, muitos deles, produzem como resultado a nova base de dados. Uma forma elegante de sequenciar várias acções sobre uma base de dados passa pelo uso da mónade de estado estudada em PP1. Relembre por isso as definições

```
data ST s l = ST { st :: s -> (s,l)}
instance Monad (ST s) where
  return x = ST (\s -> (s,x))
  p >>= f = ST (\s -> let (s',l) = st p s
                      in st (f l) s')
```

Note-se que, para uma instância `bd` da classe `BD`, o tipo `ST (bd ch inf) a` corresponde a um transformador de estados da base de dados que retorna um valor do tipo `a`. Podemos por isso generalizar os vários métodos da classe `BD` para este transformador de estados. Por exemplo

```
stAcrescenta :: (BD b, Eq ch) => ch -> inf -> ST (b ch inf) ()
stAcrescenta c i = ST (\b -> (acrescenta b c i, ()))

stChaves :: (BD b) => ST (b ch inf) [ch]
stChaves = ST (\b -> (b, chaves b))
```

Uma outra generalização possível na mónade de transformadores de estados é parametrizá-la com uma mónade genérica:

```
data STM m s l = STM { stm :: s -> m (s,l)}

instance (Monad m) => Monad (STM m s) where
   return x = STM (\s -> return (s,x))
   p >>= f = STM (\s -> do { (s',l) <- stm p s ;
                             stm (f l) s' })
```

Esta generalização é particularmente útil quando queremos que as várias operações sobre o estado sejam *intermeadas* por operações de I/O. Para isso podemos definir a seguinte abreviatura.

```
type STIO s l = STM IO s l
```

A generalização de operações de I/O ou de transformadores de estados para este tipo pode ser feita de uma forma muito linear usando as seguintes funções:

```
liftIO :: IO x -> STIO s x
liftIO i = STM (\s -> (do {a <- i ; return (s,a)}))

liftSTIO :: ST s v -> STIO s v
liftST p = STM (\s -> return (st p s))
```

Por exemplo,

- para a operação `putStr :: String -> IO ()` podemos definir a seguinte operação:

  ```
  stioPutStr = liftIO . putStr
  ```

- para a operação `stAcrescenta` acima, podemos definir

  ```
  stioAcrescenta :: (BD b, Eq ch) => ch -> inf -> STIO (b ch inf) ()
  stioAcrescenta c i = liftSTIO (stAcrescenta c i)
  ```

2 Trabalho a Realizar

1. Apresente outras (pelo menos uma) instâncias da classe `BD` (listas ordenadas, árvores binárias de procura, ...).
2. Apresente as definições das funções `stAcrescenta`, `stRemove`, `stDefinidas` e `stDaInfo` que generalizam os correspondentes métodos da classe `BD`; use a função `liftST` para obter as correspondentes operações da mónade `STIO`.
3. Defina, pelo menos para cada um dos métodos da classe `BD`, menus de entrada/saída de dados. Por exemplo, para o método `remove` poder-se-ia definir a função

   ```
   menuRemove :: (PutGet ch) => STIO s ch
   menuRemove = liftIO (do { putStr "Qual a chave a remover: " ;
                             c <- getValue ;
                             return c})
   ```

   Em que a classe `PutGet` poderia ser definida por

   ```
   class PutGet a where
     getValue :: IO a
     putValue :: a -> IO ()
   ```

4. Enuncie um contexto (gestão de um clube de vídeo, de um campeonato, dos trabalhos práticos de uma disciplina, ...) onde este tipo de dados possa ser usado e apresente a correspondente solução.

3 Código disponibilizado

O código LaTeX deste documento (`http://wiki.di.uminho.pt/twiki/pub/Education/MetodosProgramacaoI/tp2.tex`).