Tópicos Avançados de Programação Funcional

DCC/FCUP — Maio de 2017

- Este exame está cotado para 100% e contém **cinco (5) questoes** e **seis (6) páginas**
- Escreva as respostas numa folha de exame separada; pode alterar a ordem das questões desde que estas estejam claramente identificadas
- Pode consultar a folha de apêndice de documentação de algumas bibliotecas Haskell
- Duração total: 2h 30mins

Questão 1 (20%)

(a)

Considere a seguinte definição recursiva da função `filterM` da biblioteca `Control.Monad`:

```
filterM :: Monad m => (a -> m Bool) -> [a] -> m [a]
filterM f []     = return []
filterM f (x:xs) = do
    b <- f x
    ys <- filterM f xs
    return (if b then (x:ys) else ys)
```

Re-escreva a definição acima traduzindo a notação-`do` para aplicações das operações monádicas (`>>=`, `>>` e `return`).

(b)

Em versões mais recentes do GHC, a função `filterM` está definida usando apenas restrição de classe `Applicative` em vez de `Monad`:

```
filterM :: Applicative m => (a -> m Bool) -> [a] -> m [a]
```

Escreva uma definição recursiva de `filterM` usando apenas operadores aplicativos (`pure`, `fmap` e `<*>`) e sem usar notação-`do`.

Questão 2 (20%)

Considere a seguinte versão simplificada da classe de tipos `Show` de Haskell.

```
class Show a where
  show :: a -> String

instance Show Bool where
  show b = if b then "True" else "False"

instance (Show a, Show b) => Show (a,b) where
  show (x,y) = "(" ++ show x ++ "," ++ show y ++ ")"
```

(a)

Traduza a definição da classe `Show` e das instâncias para `Bool` e pares acima usando dicionários.

(b)

Complete a implementação da instância para listas:

```
instance Show a => Show [a] where
   {- completar -}
```

A sua definição **não** deve usar outras funções da classe `Show` (como `showList`) e deve produzir resultados idênticos à definição usual, e.g.

```
show []              = "[]"
show (False:[])      = "[False]"
show (False:True:[]) = "[False,True]"
```

Questão 3 (20%)

Considere a seguinte definição de um *monad* `Log` para registar mensagens de texto durante uma computação.

```
newtype Log a = Log { runLog :: (a, String) }
-- instâncias `Functor' e `Applicative' omitidas

instance Monad Log where
   return x        = Log (x, "")
   Log (x,s) >>= k = let (y, s')= runLog (k x)
                     in Log (y, s ++ s')

-- registar uma mensagem
write :: String -> Log ()
write msg = Log ((), msg)
```

Um exemplo de uso: calcular o *máximo divisor comum* pelo algoritmo de Euclides registando todos os passos intermédios.

```
mdc :: Int -> Int -> Log Int
mdc a b = do
    write ("mdc" ++ show (a,b) ++ " = ")
    if b == 0 then do write (show a)
                      return a
    else mdc b (a`mod`b)
{-
   runLog (mdc 54 30)
 = (6,"mdc(54,30) = mdc(30,24) = mdc(24,6) = mdc(6,0) = 6")
-}
```

Mostre que `Log` é de facto um *monad*, isto é, que a definições da instância acima verificam as três leis de *monads*.

Questão 4 (20%)

Pretendemos testar uma função `sort :: Ord a => [a] -> [a]` de ordenação de listas usando a biblioteca *QuickCheck*.

Considere as seguintes definições (incorretas) de propriedades:

```
-- o primeiro elemento da lista ordenada é o menor valor
prop_minimum xs = head (sort xs) == minimum xs

-- a lista ordenada está por ordem crescente
prop_ordered xs = ordered (sort xs)
   where ordered [] = True
         ordered (x:y:xs) = x<=y && ordered xs

-- a lista ordenada tem os mesmos elementos da lista dada
prop_permutation xs = permutation xs (sort xs)
   where permutation xs ys = null (ys \\ xs)
```

(a)

Cada uma das definições de propriedades acima têm pelo menos um erro ou omissão. Identifique e corriga esses erros.

(b)

Escreva uma definição de um gerador de listas `listOf :: Gen a -> Gen [a]` cujo argumento é um gerador de elementos. O comprimento das listas geradas deve ser aleatório (mas limitado pelo parâmetro de tamanho no mónada `Gen`).

Questão 5 (20%)

Considere uma DSL para formatação de caixas de texto com a seguinte API.

```
type Box                      -- tipo abstrato
-- observações
width, height :: Box -> Int   -- largura, altura
render :: Box -> [String]     -- texto formatado
-- construtores
empty :: Box                  -- caixa vazia
htext :: String -> Box        -- texto horizontal
vtext :: String -> Box        -- texto vertical
-- combinadores
(<>) :: Box -> Box -> Box     -- combinação horizontal
above :: Box -> Box -> Box    -- combinação vertical
```

Um valor de tipo `Box` consite numa grelha de carateres com uma largura e altura determinadas. A operação `<>` combina horizontalmente duas caixas; `above` combina verticalmente. Obtemos a *menor* caixa que contém as caixas dadas.

Note que as operações `<>` e `above` são associativas e têm `empty` como elemento neutro.

Por exemplo; sejam

```
b1 = htext "The" `above` htext "quick"
b2 = htext "brown" <> htext " " <> vtext "fox"
```

então temos:

```
width b1  = 5      width b2  = 7
height b1 = 2      height b2 = 3

b1  =  The                          b2  =  brown f
       quick                               o
                                           x


b1 <> b2  =  The  brown f          b1 `above` b2  =  The
             quick      o                          quick
                        x                          brown f
                                                         o
                                                         x
```

(a)

Usando a DSL acima, implemente as seguintes operações derivadas:

```
horiz :: [Box] -> Box    -- combinação horizontal
vert :: [Box] -> Box     -- combinação vertical
hline :: Int -> Box      -- linha horizontal
vline :: Int -> Box      -- linha vertical
frame :: Box -> Box      -- moldura
```

Note que deve usar *apenas* os construtores e combinadores da DSL e *não* implementar estas operações como primitivas.

Exemplos (onde `b1` é definido na alínea anterior)

```
horiz [vline 2, b1, vline 2]  =  |The  |
                                 |quick|

vert [hline 5, b1, hline 5]  =  -----
                                The
                                quick
                                -----

frame b1  =  -------
             |The  |
             |quick|
             -------
```

(b)

Implemente as operações primitivas da DSL usando *shallow embeding*.
