# Programação Funcional

Exame 617

16 de Junho de 2017

DCC/FCUP

Nome: _____

Nº mecanográfico: _____

- **Duração: 2h + 30m tolerância.**
- **Este exame contém 7 questões e 4 páginas.**
- **Responda às questões no espaço marcado no enunciado.**
- **Pode usar funções auxiliares e/ou do prelúdio-padrão de Haskell.**
- **Nas questões 2 a 7, indique sempre o tipo da função definida.**

**1. (6 valores)** Responda a cada uma das seguintes questões, indicando **apenas** o resultado de cada expressão.

(a) `length ([1,2]:[3]:[5]:[[]])` = _____

(b) `head ([]:[3]:[[5,2]])` = _____

(c) `((2+).(5*)) 4` = _____

(d) `map (<8) [2,4..10]` = _____

(e) `foldl (\x y -> 2*x+y) 0 [1,1,0,1,0]` = _____

(f) `[(x,y) | x <- [1..4], y <- [1..x], x == y+1]` = _____

(g) Sem usar explicitamente a lista dada, defina `[(1,2),(3,4),(5,8),(7,16),(9,32),...]`:

_____

(h) Considere a seguinte definição em Haskell:

```
f [x] = [3*x]
f (x:xs) = (2*x) : f xs
```

A avaliação da expressão `f [1,2,1,5,4]` tem como resultado: _____

(i) Indique um tipo admissível para `[(2>),(==4)]`: _____

(j) Indique o tipo mais geral de `(1+).(*3)`: _____

(k) Considere as seguintes definições:

```
data Arv = ???

f :: Arv -> Int
f (Folha a) = a
f (No esq dir) = f dir
```

Complete a definição do tipo `Arv`, para que a função `f` esteja bem definida:

_____

(l) Indique o tipo mais geral de `f xs ys = sum (zipWith (*) xs ys)` :

_____

**2. (3 valores)** Nas duas alíneas seguintes pode utilizar funções do prelúdio-padrão e/ou listas em compreensão mas não deve usar directamente **recursão**.

(a) Defina a função `nafrente` que dado um elemento e uma lista de listas, coloca o elemento à cabeça de cada uma das listas. Por exemplo, `nafrente 'c' [“ola”,“ela”] = [“cola”,“cela”]`.

(b) Defina uma função `ocorreN`, que dado um elemento `x` e uma lista `l`, retorna `True` se existem exactamente `n` ocorrências de `x` em `l` e `False`, caso contrário.

**3. (3 valores)** Uma sublista de `xs` é qualquer lista composta por elementos de `xs`, mantendo a ordem entre os elementos na lista original.

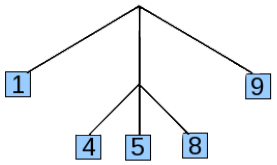
(a) Defina uma função `subs` que dada uma lista finita `xs` como argumento retorna a lista de todas as suas sublistas. Por exemplo,

`subs [1,5,3] = [[1,5,3],[1,5],[1,3],[1],[5,3],[5],[3],[]]`.

(b) Sem recorrer à função `subs`, defina uma função `subsAsc` que calcula apenas as sublistas ascendentes de `xs`. Por exemplo: `subsAsc [1,5,3] = [[1,5],[1,3],[1],[5],[3],[]]`.

**4. (1 valor)** Defina uma função `soma` que *leia do teclado* uma sequência de inteiros terminada em `0` e *imprima no écran* a soma total dos inteiros lidos.

**5. (3 valores)** Considere a seguinte árvore:




(a) Defina um tipo de dados `ArvT a`, para representar árvores binárias com valores nas folhas e tal que cada nó tem três subárvores. Defina um valor `arv :: ArvT Int` que represente a árvore da figura.

(b) Defina a função `nelementos :: ArvT a -> Int`, que calcula o número de elementos em cada árvore.

(c) Recorde a função de ordem-superior `map` definida para listas. Defina uma função `mapTree`, que se comporte como a função `map`, mas opere sobre árvores do tipo `ArvT`.

**6. (2 valores)** A função `scanl` funciona como uma junção entre as funções `map` e `foldl`: acumula um valor como a função `foldl`, mas retorna a lista de todos os valores intermédios. Por exemplo: `scanl (+) 0 [1,2,3] = [0,1,3,6]`. Sem recorrer às funções `map`, `foldl` e `scanl`, escreva uma definição em Haskell para a função `scanl`, que opere sobre listas infinitas. Por exemplo:

```
scanl (*) 1 [2,2..]  = [1,2,4,8,16,32,64,128,256,512,...]
```

**Nota:** Cada valor intermédio deve ser calculado uma única vez.

**7. (2 valores)** Responda **(apenas)** a uma das seguintes alíneas, usando indução matemática. *Nota: pode utilizar qualquer propriedade que tenha sido demostrada nas aulas, ou demonstrar qualquer resultado adicional que facilite a prova.*

(a) Considerando as funções/definições anteriormente mostre que, para qualquer árvore `t` do tipo `ArvT`, `nelementos t` retorna um inteiro da forma $2n - 1$, com $n \in \mathbb{N}$.

(b) Seja `flip` a função definida como `flip f x y = f y x`, mostre que `foldr f v (reverse xs) = foldl (flip f) v xs` para todo `f`, `e` e `xs`. Pode usar (sem demonstrar) o seguinte lema: `foldr f v (xs ++ [b]) = foldr f (f b v) xs` para todo `f`, `v`, `xs` e `b`.