**Programação Funcional, 2018.1** **Prova 4**

(Turma T56 do Thanos) (points: 124; bonus: $0^\flat$; time: 80')

---

**Nome:**

---

*09/07/2018*

## Regras:

**I.** Não vires esta página antes do começo da prova.

**II.** Nenhuma consulta de qualquer forma.

**III.** Nenhum aparelho ligado (por exemplo: celular, tablet, notebook, *etc.*).[1]

**IV.** Nenhuma comunicação de qualquer forma e para qualquer motivo.

**V.** $\forall x\big(\text{Colar}(x) \to \neg\text{Passar}(x, \text{FUN})\big)$.

**VI.** Use caneta para tuas respostas.

**VII.** Responda dentro das caixas indicadas.

**VIII.** Escreva teu nome em *cada* folha de rascunho extra *antes de usá-la*.

**IX.** Entregue *todas* as folhas de rascunho extra, juntas com tua prova.

**X.** Nenhuma prova será aceita depois do fim do tempo!

**XI.** Os pontos bônus são considerados apenas para quem consiga passar sem.[2]

**XII. Responda em até 3 dos problemas.**[3]

*Boas provas!*

[1] Ou seja, *desligue antes* da prova.

[2] Por exemplo, 25 pontos bônus podem aumentar uma nota de 5,2 para 7,7 ou de 9,2 para 10,0, mas de 4,9 nem para 7,4 nem para 5,0. A 4,9 ficaria 4,9 mesmo.

[3] Provas com respostas em mais que três problemas não serão corrigidas (tirarão 0 pontos).

(36) **A**

Defina os 3 típos de dados:

```
Nat  ::  *      List  ::  * -> *    Either  :: * -> * -> *
```

e as 9 funções

```
(+)       :: Nat -> Nat -> Nat      sum       :: [Nat] -> Nat
concat    :: [[a]] -> [a]           repeat    :: a -> [a]
lefts     :: [Either a b] -> [a]    partition :: [Either a b] -> ([a], [b])
zipWith   :: (a -> b -> c) -> [a] -> [b] -> [c]
either    :: (a -> c) -> (b -> c) -> Either a b -> c
takeWhile :: (a -> Bool) -> [a] -> [a]
```

sem usar list comprehension.
DEFINIÇÕES.

(42) **N**

Primeiramente defina recursivamente as funções

```
all       :: (a -> Bool) -> [a] -> Bool
replicate :: Nat -> a -> [a]
```

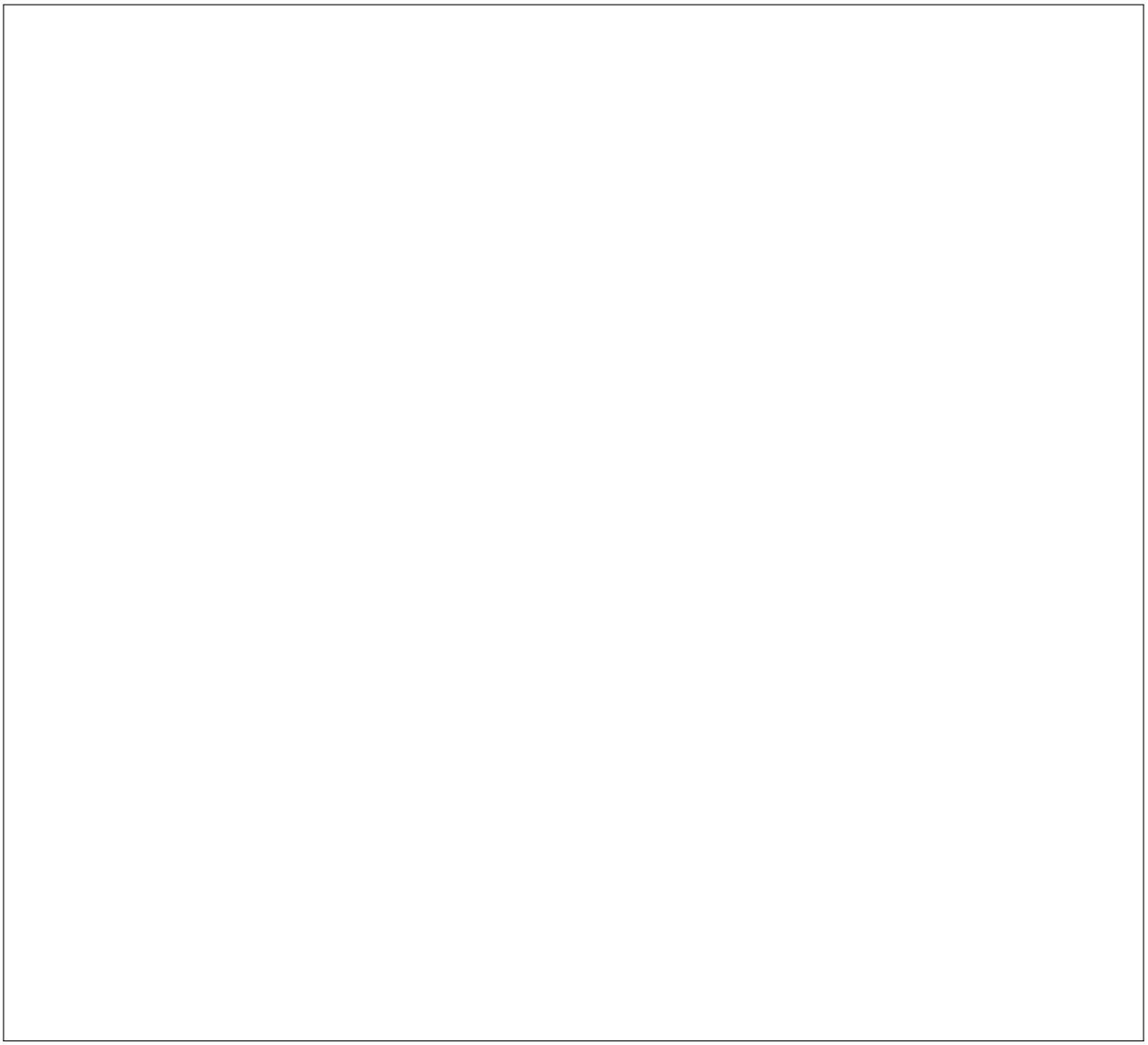
cada uma com duas equações. Depois prove que

```
all (== x) (replicate n x)
```

para todo finito `n :: Nat` e todo `x :: a`.

(4) Definições.

(38) Prova.



(46) # D

Primeiramente defina recursivamente as funções

```
take  :: Nat -> [a] -> [a]
drop  :: Nat -> [a] -> [a]
```

cada uma com até 3 equações. Depois prove que

```
take n xs ++ drop n xs  ==  xs
```

para todo finito `n :: Nat` e toda finita `xs :: [a]`.

(4) Definições.

(42) Prova.

(28) # R

Defina a função

```
foldr :: (a -> b -> b) -> b -> List a -> b
```

e depois use-a para definir a

```
length :: List a -> Nat
```

**como um fold**.

Definições.

(36) # L

Defina a função

```
foldl :: (b -> a -> b) -> b -> [a] -> b
```

e depois use-a para definir **como um fold** a

```
dec2int :: [Int] -> Int
```

que recebendo um numeral decimal, retorna o número que ele representa:

```
ghci> dec2int [1,9,5,0]
1950
```

Definições.

Só isso mesmo.

Rascunho