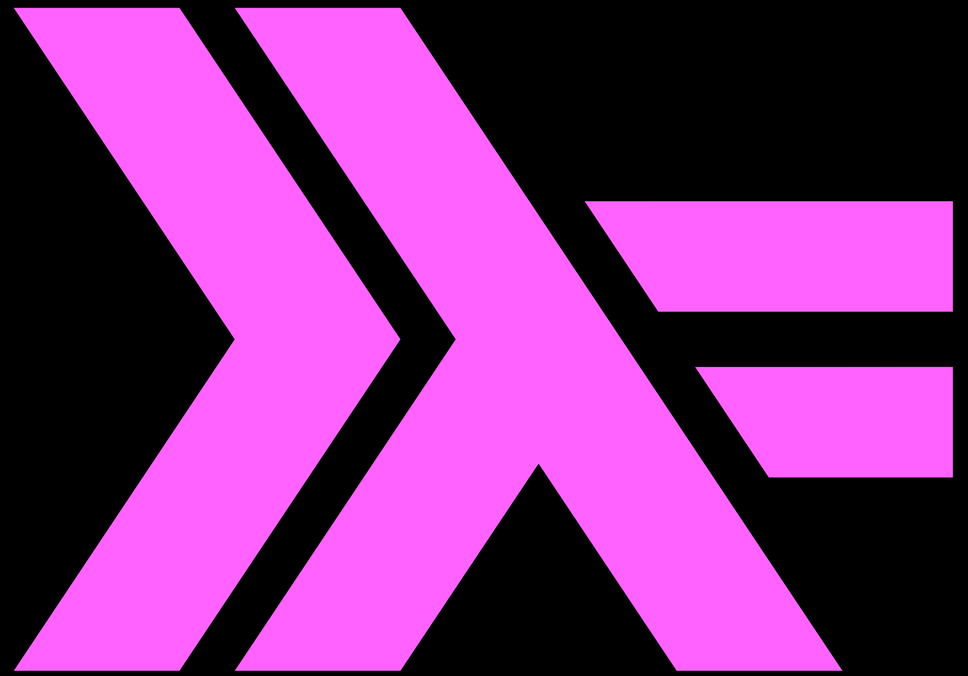
{Exame Especial}

BCC222

Thalles Felipe Rodrigues de Almeida Santos



Questão 1. [Parcial 1] Qual é o resultado da compilação e avaliação da expressão a seguir? Escolha a alternativa correta.

```
let x = if "bola" > "gato" then "sim" else "nao"
in case reverse x of
     [] → 0
     ['m', _, _] → 2
     'o':_ → 3
     _ → length x
```

a) erro de sintaxe
b) erro de tipo
c) 0
d) 2
e) 3
f) 5

```
let x = if "bola" > "gato" then "sim" else "nao"
```

```
"bola" > "gato"
```

```
"b" > "g"
```

False

```
let x = if "bola" > "gato" then "sim" else "nao"
```

```
let x = "nao"
```

```
in case reverse x of
    [] → 0
    ['m', _, _] → 2
    'o':_ → 3
    _ → length x
```

reverse x

reverse “nao”

“oan”

```
in case “oan” of
    [] → 0
    ['m', _, _] → 2
    'o':_ → 3
    _ → length x
```


'o':_ → 3

e) 3

Questão 2. [Parcial 1] Defina a função *contem* que recebe duas listas e retorna verdadeiro se e somente se a primeira lista contém todos os elementos da segunda lista.

Instruções:
Escreva a assinatura de tipo mais geral;
Use recursividade explícita;
Use a função *elem*.

Exemplos de saída esperadas:

```
contem [5, 3, 1, 8, 6, 10, 2] [1, 2, 5]              → True
contem [5, 3, 1, 8, 6, 10, 2] [1, 3, 5, 7, 9]        → False
contem [5, 3, 1, 8, 6, 10, 2] [100, 200, 300, 400] → False
contem [5, 3, 1, 8, 6, 10, 2] []                     → True
contem "bom dia brasil" "ao"                          → True
contem "bom dia brasil" "aeiou"                       → False
```

caso base

```
contem _ [] = True
```

recursividade explícita e usando a função *elem*.

```
contem xs (y:ys) = (elem y xs) && (contem xs ys)
```

```
contem :: Eq a ⇒ [a] → [a] → Bool
contem _ [] = True
contem xs (y:ys) = (elem y xs) && (contem xs ys)
```

contem [5, 3, 1, 8, 6, 10, 2] [] = True

caso base

contem [5, 3, 1, 8, 6, 10, 2] [1, 2, 5]

```
contem [5, 3, 1, 8, 6, 10, 2] [1, 2, 5]
```

Listas:

xs = [5, 3, 1, 8, 6, 10, 2]

ys = [1, 2, 5]

contem [5, 3, 1, 8, 6, 10, 2] [1, 2, 5]

Cabeça:

y = 1

Avaliamos:

(*elem* 1 xs) && (*contem* xs ys)

(*elem* 1 xs) = True

True && (*contem* xs ys)

contem [5, 3, 1, 8, 6, 10, 2] [2, 5]

Cabeça:

y = 2

Avaliamos:

(*elem* 2 xs) && (*contem* xs ys)

(*elem* 2 xs) = True

True && (*contem* xs ys)

contem [5, 3, 1, 8, 6, 10, 2] [5]

Cabeça:

y = 5

Avaliamos:

(*elem* 5 xs) && (*contem* xs ys)

(*elem* 5 xs) = True

True && (*contem* xs ys)

Caso base:

```
contem _ [] = True

contem [5, 3, 1, 8, 6, 10, 2] [] = True
```

```
True && (contem xs [])
True && True
True
```

```
True && (contem xs [5])
True && True
True
```

```
True && (contem xs [2, 5])
True && True
True
```

```
True && (contem xs [2, 5])
True && True
True
```

Caso algum elemento não seja elemento de *xs*, logo retornará `False` e todas as operações lógicas de True && False ou False && False retornarão `False`.

Questão 3. [Parcial 1, Total] A nota final de um estudante é calculada a partir de três notas atribuídas respectivamente a um trabalho de laboratório, a uma apresentação de seminário, e a um exame final. A média ponderada das três notas mencionadas obedece aos pesos a seguir:

Nota	Peso
Laboratório	2,5
Seminário	3,0
Exame Final	4,5

Questão 3. [Parcial 1, Total] Escreva um programa em *Haskell* que recebe as três notas informadas pela usuário e determina e exibe o conceito obtido pelo aluno usando a tabela:

Média ponderada	Conceito
[8.0 - 10.0]	A
[7.0 - 8.0[	B
[6.0 - 7.0[	C
[4.0 - 6.0[	D
[0.0 - 4.0[	E

Exemplo de execução do programa:

```
Digite a nota do trabalho de laboratório: 5.1
Digite a nota da apresentação de seminário: 6.2
Digite a nota do exame final: 7.5
Conceito obtido: C
```

```
import System.IO (BufferMode (NoBuffering), hSetBuffering, stdout)
```

```haskell
calculaConceito :: Double → Double → Double → Char
calculaConceito lab sem final
  | media ≥ 8.0 = 'A'
  | media ≥ 7.0 = 'B'
  | media ≥ 6.0 = 'C'
  | media ≥ 4.0 = 'D'
  | otherwise = 'E'
  where
    media = (lab * 2.5 + sem * 3.0 + final * 4.5) / 10.0
```

```haskell
main :: IO ()
main = do
  hSetBuffering stdout NoBuffering

  putStr "Digite a nota do trabalho de laboratório: "
  notaLab ← readLn :: IO Double

  putStr "Digite a nota da apresentação de seminário: "
  notaSem ← readLn :: IO Double

  putStr "Digite a nota do exame final: "
  notaFinal ← readLn :: IO Double

  let conceito = calculaConceito notaLab notaSem notaFinal

  putStrLn ("\nConceito obtido: " ++ [conceito])
```

Questão 4. [Parcial 2, Total] Campeonato de futebol. Em uma aplicação pretende-se analisar informações sobre os resultados dos jogos de um campeonato de futebol usando as seguintes estruturas de dados na linguagem *Haskell*:

```haskell
type Time = String
type Gols = Int
type Jogo = ((Time, Gols), (Time, Gols))
type Campeonato = [Jogo]
type Pontos = Int
type Tabela = [(Time, Pontos)]

campMineiro :: Campeonato
campMineiro =
  [ (("cruzeiro", 0), ("atletico", 0)),
    (("uberlandia", 5), ("america", 1)),
    (("atletico", 1), ("america", 2)),
    (("uberlandia", 2), ("cruzeiro", 1)),
    (("uberlandia", 3), ("urt", 1)),
    (("atletico", 4), ("uberlandia", 4)),
    (("urt", 0), ("atletico", 1)),
    (("urt", 0), ("america", 2)),
    (("caldense", 3), ("atletico", 3))
  ]
```

Questão 4. [Parcial 2, Total] Defina uma função *empates* do tipo *Campeonato -> Time -> Int* que determina o número de empates de um determinado time em um dado campeonato.

Instruções:
Use funções de ordem superior. Não use recursividade explícita.

Exemplo de execução do programa:

```
empates [] "cruzeiro"                → 0
empates campMineiro "santos"         → 0
empates campMineiro "atletico"       → 3
empates campMineiro "uberlândia"     → 1
empates campMineiro "urt"            → 0
```

```haskell
jogoEhEmpateDoTime :: Time → Jogo → Bool
jogoEhEmpateDoTime time ((timeA, golsA), (timeB, golsB)) =
    (timeA == time || timeB == time)  -- O time participou do jogo?
    && (golsA == golsB)               -- O jogo foi um empate?

empates :: Campeonato → Time → Int
empates campeonato time = length (filter (jogoEhEmpateDoTime time) campeonato)
```

Questão 5. [Parcial 2, Total] Implementação da classe *YesNo* e função *yesnoIf*.

Instruções:
Definir a classe, instâncias para *Integer*, *[a]*,
Bool, *Maybe a*, um tipo *Semaforo* e sua instância.
Avaliar expressões *yesno*. Definir e testar *yesnoIf*.

Definição da Classe *YesNo*

```
class YesNo a where
    yesno :: a → Bool
```

Instância para *Integer*

```
instance YesNo Integer where
    yesno 0 = False
    yesno _ = True
```

Instância para *[a]* (Listas)

```
instance YesNo [a] where
    yesno [] = False
    yesno _  = True
```

Instância para *Bool*

```
instance YesNo Bool where
  yesno True = True
  yesno False = False
```

Instância para *Maybe a*

```
instance YesNo (Maybe a) where
  yesno Nothing  = False
  yesno (Just _) = True
```

Avaliando expressões *yesno*:

```
(a) yesno $ length [] → False
(b) yesno "bom dia" → True
(c) yesno "" → False
(d) yesno (Just 12.4) → True
(e) yesno True → True
(f) yesno [] → False
(g) yesno [("ana", 10), ("pedro", 12), ("beatriz", 9)] → True
(h) yesno Vermelho → False
```

Definição da função *yesnoIf*

```
yesnoIf :: YesNo a ⇒ a → b → b → b

yesnoIf condicao valorSeYes valorSeNo =
    if yesno condicao then valorSeYes else valorSeNo
```

Avaliando expressões *yesnoIf*:

```
(a) yesnoIf [] "YEAH!" "NO!" → "NO!"
(b) yesnoIf [2,3,4] "YEAH!" "NO!" → "YEAH!"
(c) yesnoIf True "YEAH!" "NO!" → "YEAH!"
(d) yesnoIf (Just ("carla", 34, 174)) "YEAH!" "NO!" → "YEAH!"
(e) yesnoIf Nothing "YEAH!" "NO!" → "NO!"
(f) yesnoIf Verde "YEAH!" "NO!" → "YEAH!"
```

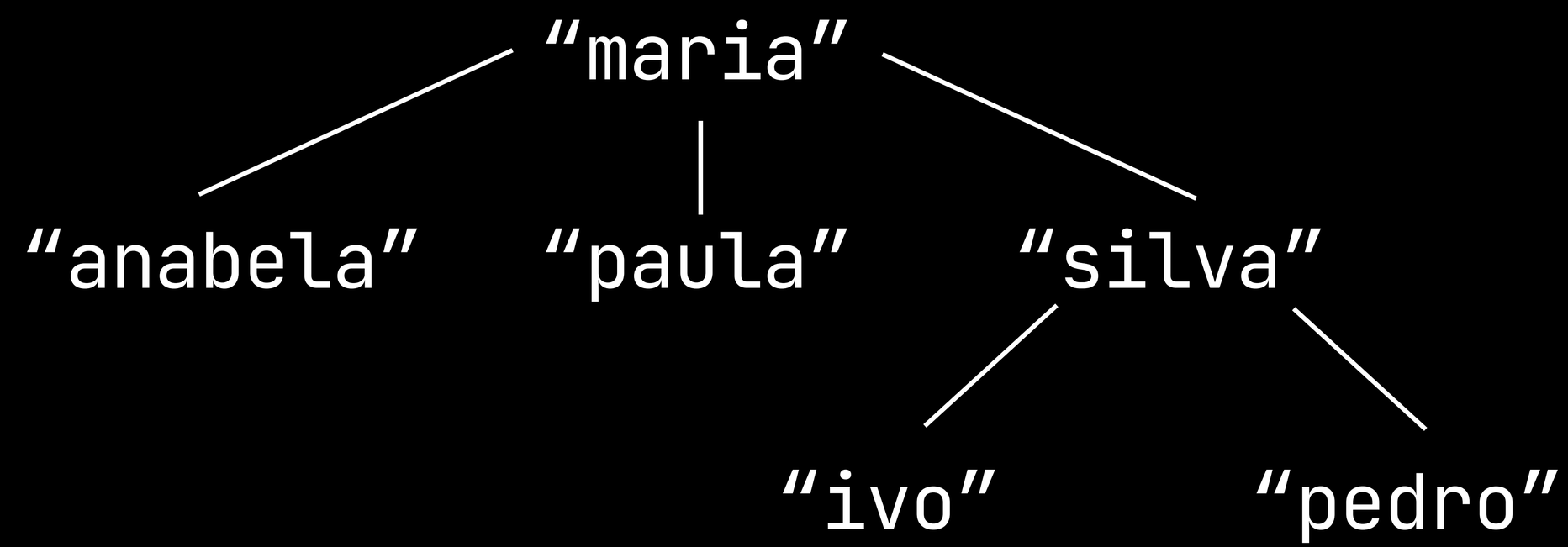
Questão 6. [Parcial 2, Total] Considere o tipo de dados a seguir, para representar árvores:

```
data Arv a
  = V -- Árvore vazia
  | N x [Arv a] -- Árvore não vazia: contém um elemento e uma lista de subárvores
  deriving (Eq, Show)
```

Exemplo de uma árvore:

```
a1 :: Arv String
a1 = No "maria" [ No "anabela" [],
                  No "paula" [],
                  No "silva" [ No "ivo" [],
                               No "pedro" []
                             ]
                ]
```

Ilustração da árvore *a1*:



Assim um valor do tipo *Arv a* pode ser vazio, ou pode encapsular vários valores. Logo a estrutura *Arv* pode ser um *functor*.

Questão 6. [Parcial 2, Total] Assim um valor do tipo *Arv a* pode ser vazio, ou pode encapsular vários valores. Logo a estrutura *Arv* pode ser um *functor*.

Defina a instância da classe *Functor* para a estrutura *Arv*. Lembre-se você terá que definir a função *fmap*.

Classe *Functor*:

```
class Functor f where
  fmap :: (a → b) → f a → f b
```

Implementação:

```
instance Functor Arv where
    fmap f V = V

    fmap f (N x subtrees) = N (f x) (map (fmap f) subtrees)
```

Verificação com o exemplo:

```
a1 :: Arv String
a1 = No "maria" [ No "anabela" [],
                  No "paula" [],
                  No "silva" [ No "ivo" [],
                               No "pedro" []
                             ]
                ]
```

Aplicando *fmap length a1*:

fmap length V seria *V*

fmap length (N "maria" [...]) torna-se:

- *N (length "maria") (map (fmap length) [...])*
- *N 5 (map (fmap length) [N "anabela" [], N "paula" [], N "silva" [...]])*

Aplicando *fmap length a1*:

Aplicando *map (fmap length)* à lista de subárvores:

- *fmap length (N "anabela" []) → N (length "anabela") (map (fmap length) []) → N 7 (map (fmap length) []) → N 7 []*
- *fmap length (N "paula" []) → N (length "paula") (map (fmap length) []) → N 5 []*
- *fmap length (N "silva" [...]) → N (length "silva") (map (fmap length) [N "ivo" [], N "pedro" []])*

Aplicando *fmap length a1*:

Aplicando *map (fmap length)* a *[N "ivo" [], N "pedro" []]*:

- *fmap length (N "ivo" [])* → *N (length "ivo") []* → *N 3 []*
- *fmap length (N "pedro" [])* → *N (length "pedro") []* → *N 5 []*

Resulta em *N 5 [N 3 [], N 5 []]*

Aplicando *fmap length a1*:

Juntando tudo: *N 5 [N 7 [], N 5 [], N 5 [N 3 [], N 5 []]]*

Obrigado!