Início	quinta, 27 de outubro de 2022 às 17:14
Estado	Prova submetida
Data de submissão:	quinta, 27 de outubro de 2022 às 17:33
Tempo gasto	19 minutos 5 segundos
Nota	**4,75** de um máximo de 6,00 (**79**%)

Pergunta 1

Correta Pontuou 0,500 de 0,500

What is the type of the following function?

```
orderedPair (a, b)
  | a <= b = (a, b)
  | otherwise = (b, a)
```

a. `(Num a, Num b) => (a, b) -> (b, a)`

b. `(Ord a) => (a, a) -> (a, a)` ✔

c. `(Num a, Num b) => (a, b) -> (a, b)`

d. `(Ord a, Ord b) => (a, b) -> (b, a)`

e. `(Num a, Ord a, Num b, Ord b) => (a, b) -> (b, a)`

Pergunta 2

Correta Pontuou 0,500 de 0,500

Tem 3 elementos maiores que 0

What is the result of the following expression?

```
(length . (filter (> 0))) [1, 2, -3, 4, -5]
```

a. `0`

b. The evaluation of the expression produces an error.

c. `3` ✔

d. `1`

e. `2`

(++) []: Isso representa a função de concatenação de listas, onde uma lista vazia ([]) é adicionada ao final de outra lista. A assinatura de tipo seria [a] -> [a], indicando que ela aceita uma lista de qualquer tipo a e retorna uma lista do mesmo tipo a.

map (+1): Isso representa a aplicação da função (+1) a cada elemento de uma lista. A assinatura de tipo seria (Num a) => [a] -> [a], indicando que ela aceita uma lista de números (Num a) e retorna uma lista do mesmo tipo.

Pergunta 3

Correta Pontuou 0,500 de 0,500

What is the type of the following expression?

```
[(++) [], map (+1)]
```

a. `[[a] -> [a]]`

b. `(Num a) => [a]`

c. `(Num a, Num b) => [[a] -> [b]]`

d. `[[a] -> [b]]`

e. `(Num a) => [[a] -> [a]]` ✔

Pergunta 4

Correta Pontuou 0,500 de 0,500

(x,y,x) = (b,a,b)

What is the type of the following function?

```
fun (x, y, _) = (y, x, y)
fun (_, y, x) = (y, x, y)
```

a. `(a, a, a) -> (a, a, a)`

b. `(a, b, c) -> (a, b, c)`

c. `(a, b, a) -> (b, a, b)` ✔

d. `(a, b, c) -> (d, e, f)`

e. `(a, a, a) -> (b, b, b)`

Pergunta 5

Correta Pontuou 0,500 de 0,500

Os valores de a têm que ser <= que 'd'

What is the result of the following expression?

```
[(a, b) | a <- "abc", b <- [1, 2], a <= 'd']
```

a. `[('a',1), ('b',1), ('c',1), ('a',2), ('b',2), ('c',2)]`

b. `[('a',1), ('a',2), ('b',1), ('b',2), ('c',1), ('c',2)]` ✔

c. The evaluation of the expression produces an error.

d. `[('a',1), ('b',1), ('c',1)]`

e. `[]`

Pergunta 6

Correta Pontuou 0,500 de 0,500

200/1=200/2=100/4=25.0

What is the result of the following expression?

```
foldl (/) 200 [1, 2, 4]
```

a. `50.0`

b. The evaluation of the expression produces an error.

c. `25.0` ✔

d. `100.0`

e. `1.0e-2`

Pergunta 7

Correta Pontuou 0,500 de 0,500

Which of the following Prelude functions does NOT necessarily return a list?

a. `(++)` concatenação de duas listas.

b. `(:)` adiciona um elemento ao início de uma lista.

c. `zip` combina duas listas em uma lista de pares

d. `init` retorna todos os elementos de uma lista

e. `(!!)` retorna o elemento em uma posição específica de uma lista ✔

Pergunta 8

Correta Pontuou 0,500 de 0,500

Consider the three following statements about the "type" and "data" keywords.

A - "type" does not allow the use of type variables, unlike "data".

B - "type" does not allow recursive type definitions.

C - It is possible to define an instance of Eq using "data".

Which statements are correct?

Ambos "type" e "data" podem usar variáveis de tipo. A diferença é que "type" cria sinônimos de tipos, enquanto "data" define novos tipos de dados.

Quando você usa "type" para criar sinônimos de tipos, não é permitido ter definições recursivas.

a. Only B and C. ✔

b. Only A and B.

c. Only A and C.

d. Only B.

e. A, B and C.

Você pode definir instâncias da classe de tipo Eq tanto para "type" quanto para "data". Ambos podem ser usados para criar tipos que têm uma instância de Eq para permitir comparações de igualdade.

Pergunta 9

Incorreta Pontuou -0,125 de 0,500

Among the types Maybe, State and IO, which of them are monads?

a. Maybe, State and IO.

b. Only Maybe and IO. ✖

c. Only State and IO.

d. Only IO.

e. None of these types is a monad.

The correct answer is:
d. Only IO.

Explanation:

Maybe: Maybe is a monad. It represents computations that may or may not return a value. It has instances for the Monad type class.

State: State is a monad. It represents computations that carry a state along as they proceed. It also has instances for the Monad type class.

IO: IO is a monad. It represents computations that interact with the external world, such as reading from or writing to files, performing I/O operations, etc. It is a fundamental monad in Haskell.

Pergunta 10

Correta Pontuou 0,500 de 0,500

What is the correct type of the following function?

```
howdy name = putStrLn ("howdy " ++ name ++ "!")
```

a. `String -> IO ()` ✔

b. `String -> String`

c. `IO ()`

d. `IO (String)`

e. `String -> IO (String)`

Pergunta 11

Incorreta Pontuou -0,125 de 0,500

Haskell has lazy evaluation, which allows for ...

a. the automatic inference of the functions' type. ✖

b. the optimization of the memory consumption of a program, in exchange for an increased execution time.

c. the definition of polymorphic functions.

d. certain computations with infinite data structures to be finite.

e. the definition of higher-order functions.

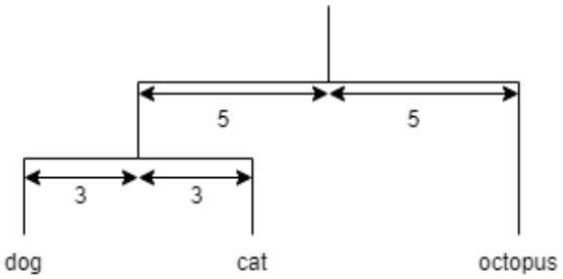
Haskell's lazy evaluation allows for the definition and manipulation of infinite data structures. In Haskell, computations are only performed as needed, and values are not evaluated until they are required. This enables the creation and manipulation of potentially infinite data structures without the need to compute all elements at once.

Pergunta 12

Correta Pontuou 0,500 de 0,500

Consider a dendrogram as a binary tree where each path leads to a string. Each non-leaf node of the dendrogram specifies the horizontal distance from the father node to each of the two child nodes. A father node is always at an equal horizontal distance from both its children.

Example of a dendrogram:



What is the most correct definition of the Dendrogram type?

a. `data Dendrogram = Leaf String | Node Int Int Dendrogram`

b. `(Integral a) => data Dendrogram = Leaf (String, a) | Node Dendrogram Dendrogram`

c. `data Dendrogram = Leaf (String, Int) | Node Dendrogram Dendrogram`

d. `(Integral a) => data Dendrogram = Leaf String | Node Dendrogram a a Dendrogram`

e. `data Dendrogram = Leaf String | Node Dendrogram Int Dendrogram` ✔