Disciplina: Paradigmas de Programação
Professor: Maicon Rafael Zatelli
Entrega: *Moodle*

Atividade X - Haskell

Atenção: Faça um ZIP com todos os arquivos de solução. Use o nome do arquivo de maneira a entender qual problema você está resolvendo. Por exemplo, problema1.hs, problema2.hs e assim por diante.

Resolva os seguintes problemas na linguagem Haskell:

1. Altere o exemplo visto em sala sobre a operação `Div` para suportar também as operações `Mul Expr Expr` (para multiplicação), `Add Expr Expr` (para soma) e `Sub Expr Expr` (para subtração). Assim, sua solução deve suportar expressões como (Mul (Div (Add (Val 28) (Val 2)) (Sub (Val 6) (Val 1))) (Val 3)). Faça as modificações necessárias nas funções de avaliação criadas e crie ao menos três exemplos de expressões para testar sua solução. As modificações devem ser feitas nas três formas de criar esta função vistas em sala de aula (sem usar monads, usando >>= e usando a notação `do`. Não preocupe-se com precedência de operadores. Qual forma foi mais fácil de dar manutenção? E mais difícil?

2. Além das operações incluídas no problema anterior, inclua também uma operação para raiz quadrada, na forma `Sqrt Expr`. Trate a situação de raiz quadrada de número negativo como erro (utilize o Monad `Maybe` para isso). Faça as modificações necessárias nas funções de avaliação criadas e crie ao menos três exemplos de expressões para testar sua solução. As modificações devem ser feitas nas três formas de criar esta função vistas em sala de aula (sem usar monads, usando >>= e usando a notação `do`. Não preocupe-se com precedência de operadores. Qual forma foi mais fácil de dar manutenção? E mais difícil?

3. Crie uma função com a assinatura `formaTriangulo :: (Float, Float) -> (Float, Float) -> (Float, Float) -> Maybe Float`, a qual recebe três pontos 2D como parâmetro e retorna a área do triângulo ou Nothing, caso os três pontos não formem um triângulo. Utilize o conceito de Monad para esta questão.

4. Considere a existência de um tabuleiro de Xadrez 8x8, onde temos um cavalo e desejamos efetuar movimentos. Para um determinado movimento, definimos qual casa gostaríamos de colocar o cavalo, porém o cavalo jamais poderá efetuar um movimento inválido, caso contrário o resultado da sequência de operações deve ser um erro, assim como ocorria nas questões anteriores. Um movimento inválido é o cavalo ir para uma casa inexistente do tabuleiro ou mesmo alcançar uma casa inválida para as regras do cavalo. Uma sequência de movimentos de um cavalo pode ser representada por meio de uma lista de duplas [(Int, Int)], onde cada duplas indica a posição desejada do cavalo após a realização do suposto movimento. Assim, crie uma função `calcPosicaoFinal :: [(Int, Int)] -> Maybe (Int, Int)` a qual deve retornar o último elemento da lista, caso todos os movimentos do cavalo forem válidos, ou deve retornar Nothing, caso algum movimento foi inválido. Assuma que o primeiro elemento da lista é a posição inicial do cavalo. Assuma também que as posições do tabuleiro começam em 0 e terminam em 7, sendo o canto esquerdo inferior a posição (0,0) e o canto direito superior a posição (7,7). Por exemplo, a sequência [(1,0), (2,2), (0,3), (2,4)] é uma sequência de movimentos válida, mas a sequência [(1,0), (3,2), (0,3), (2,4)] não. Crie funções auxiliares, caso necessário. Utilize o conceito de Monad para esta questão.

5. Pesquise sobre a classe Monad do Haskell. Que outras operações, além de *return* e >>= existem nela? Qual a diferença entre >> e >>=?

6. Crie uma função `escreva :: String -> Int -> IO ()`, a qual recebe como parâmetros uma string e um inteiro n e imprime na tela os n primeiros caracteres da string. Para imprimir cada caracter, utilize a função `putChar`. Não crie nenhuma função auxiliar e nem altere os parâmetros da função escreva. Crie uma versão usando >>= e outra usando a notação `do`.

7. Crie uma função `echo :: IO ()`, a qual lê um caracter e simplesmente o imprime na tela. Não crie nenhuma função auxiliar e nem altere os parâmetros da função escreva. Crie uma versão usando >>= e outra usando a notação `do`.

8. Pesquise sobre o Monad `Either` disponível no Haskell. Faça um pequeno exemplo e explique o seu funcionamento.

9. Listas, em Haskell, também são Monads. Faça um pequeno exemplo utilizando listas (e seus conceitos de Monads) e explique o seu funcionamento.