Faculdade de Computação
Programação Funcional (BCC) - 1° Período
1a. Avaliação - 20 pontos (Questões e Gabarito)

1) Para as expressões abaixo, escritas em Haskell, descreva a operação realizada e informe o resultado obtido (3 pontos):

```
> "codigo" ++ "-fonte"
```

O operador ++ realiza a concatenação de duas listas, neste caso, listas de caracteres.

```
Hugs> "codigo" ++ "-fonte"
"codigo-fonte"
```

```
> if 12>5 then 100 else 200
```

A instrução **if** define uma seleção bidirecional, o que permite escolher entre dois caminhos de execução, tendo em consideração o resultado de uma expressão. Neste caso a expressão (12>5) é verdadeira, então o resultado é 100.

```
> pi * r * r where r = 3
```

A expressão representa o cálculo do raio de uma circunferência quando o valor do raio é 3. O resultado é 28.27.

```
> add 2 3 where add a b = a + b
```

A expressão representa a aplicação da função **add** aos valores 2 e 3 sendo que **add** é definida localmente para dois argumentos, retornando a soma de ambos. O resultado é 5.

```
> simple (simple 2 3 4) 5 6 where simple x y z = x * (y + z)
```

A função simple é chamada duas vezes para 3 argumentos, e é definida localmente por uma expressão aritmética após a cláusula **where**.

```
Hugs> simple (simple 2 3 4) 5 6 where simple x y z = x * (y + z)
154
```

```
> let square n = n^2 in square 5
```

A expressão **let** permite definir um bloco no qual podem ser feitas definições locais. Neste caso foi feita uma definição local para a função square, que será aplicada ao valor 5. O resultado é 25.

```
> 10 `mod` 5 == 0
```

Pretende-se verificar com esta expressão (utilizando a função **mod**) se o resto da divisão de 10 por 5 é zero. O resultado é verdadeiro (`True`).

```
> (10,20,30) < (1,2,3)
```

A expressão acima compara duas tuplas, verificando se os elementos da primeira são menores que os elementos da segunda. O resultado é falso (`False`).

```
> let x = 1 in (let y = 2 in x + y) * (let z = 3 in x * z)
```

As expressões let definem localmente valores para as variáveis x y e z, que serão utilizadas para a realização de um cálculo. O resultado é 9.

2) Sejam as funções abaixo `fun1` e `fun2`. Mostre a diferença entre as duas funções explicando a finalidade de cada uma. Analise e dê o resultado quando `x = 2.5555`. (3 pontos)

```
fun1 :: Float -> Int
fun1 x = floor (100 * x)
```

```
fun2 :: Float -> Int
fun2 x = 100 * floor x
```

Resposta:

```
> fun1 2.5555
255
```

```
> fun2 2.5555
200
```

As duas funções tem o objetivo de multiplicar um número por 100 e arredondar o resultado (a função floor faz o arredondamento para 'baixo'). Na primeira função o arredondamento é feito após a multiplicação, resultando em 255. Na segunda função o arredondamento é feito antes da multiplicação, e o resultado é 200.

3) Seja a função `steps` definida recursivamente. Explique a função e forneça o resultado, mostrando passo-a-passo, das chamadas com `n=2` e `n=-2`. (4 pontos)

```
next n = n + 1
steps n  |n == 1     = 1
         |otherwise = steps (next n) + 1

> steps 2
> steps (-2)
```

Resposta: A função steps calcula os 'passos' de um número negativo até o número 1. Ex: steps (-3) : (-3,-2,-1,0,1) = 5. A função não pára quando aplicada a um número maior que 1.

```
Main> steps 2
ERROR - C stack overflow

>>>>>>> Execução passo a passo: <<<<<<

    steps 2 =
       steps 3 + 1 =
            steps 4 + 1 =
                 steps 5 + 1 =
                       ...


Main> steps (-2)
4
>>>>>>> Execução passo a passo: <<<<<<

    steps (-2) =
       steps(-1) + 1 = 4
          ↑
    steps (-1) =
       steps 0 + 1 = 3
          ↑
    steps 0 =
       steps 1 + 1 = 2
          ↑
    steps 1 =
       1
```

4) Implemente em linguagem Haskell uma função `calculadora` que deve receber 3 argumentos: um operador (Char) e dois operandos (Float). Os operadores previstos são: (+) adição, (-) subtração, (/) divisão e (*) multiplicação. O objetivo da função é retornar o resultado da aplicação do operador aos operandos. A função deve prever a divisão por zero e retornar, neste caso, a mensagem “não eh possível realizar a divisão”. (5 pontos)

Resposta:

```
calculadora::Char->Float->Float->Float
calculadora op x y | (op == '+') = x + y
                   | (op == '-') = x - y
                   | (op == '*') = x * y
                   | (op == '/') =
                        if y /= 0  then x / y
                        else error "não eh possivel realizar a divisao"
                   | otherwise = error "operador invalido"

> calculadora '+' 3.41 5.8
9.21
```

5) Considere as seguintes definições em Haskell: (5 pontos)

```
type Segundo = Int
type Minuto = Int
type Hora = Int
type Relogio = (Hora, Minuto, Segundo)
```

(a) Faça uma função que retorna verdadeiro se um valor informado para segundos é valido.

```
segValido :: Segundo -> Bool
segValido s = s >= 0 && s <= 59

> segValido 55
True
```

(b) Faça uma função que retorna verdadeiro se um valor informado para minutos é valido.

```
minValido :: Minuto -> Bool
minValido m = m >= 0 && m <= 59

> minValido 23
True
```

(c) Faça uma função que retorna verdadeiro se um valor informado para hora é valido (considere o formato de 24 horas).

```
horValida :: Hora -> Bool
horValida h = h >= 0 && h <= 23

> horValida 11
True
```

(d) Faça uma função que retorna verdadeiro se um valor do tipo Relógio (h,m,s) é valido.

```
relValido :: Relogio -> Bool
relValido (h,m,s) = (horValida h) && (minValido m) && (segValido s)

> relValido (15,30,58)
True
```

(e) Escreva uma função para converter de um valor do tipo Relógio (h,m,s) para segundos.

```
relSeg :: Relogio -> Segundo
relSeg (h,m,s) = h*3600 + m*60 + s

> convRelSeg (15,30,45)
55845
```