

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FEELT - FACULDADE DE ENGENHARIA ELÉTRICA
ENGENHARIA DE COMPUTAÇÃO

LUCAS ALBINO MARTINS

Matrícula: 12011ECP022

**TRABALHO 01: 1ª. Lista de Exercícios: Expressões e Funções
em Linguagem Haskell.**

Disciplina: Programação Funcional

Uberlândia
2020

1a. Lista de Exercícios: Expressões e Funções em Linguagem Haskell.

- 1) Teste as seguintes expressões no sistema GHCi, descreva a operação realizada e informe o resultado obtido:**

-- Divisão inteira numerador `operador` denominador, seis dividido por três.

>6 `div` 3

2

-- Divisão inteira primeiro valor e o numerador o segundo o denominador, seis dividido por três.

>div 6 3

2

-- Divisão modular(mostra o resto da divisão), dez dividido por sete, o resto da divisão é três.

>10 `mod` 7

3

-- Exponencial para números inteiros, cinco elevado ao cubo.

> 5^3

125

-- Exponencial para ponto flutuante(Float), cinco elevado ao cubo.

> 5**3

125.0

-- Exponencial para números inteiros, cinco elevado a menos três, como não é permitido para esse operador o expoente negativo devido a resposta da operação ser um valor em ponto flutuante e não um inteiro.

> 5^(-3)

*** Exception: Negative exponente

-- Exponencial para ponto flutuante com expoente negativo, cinco elevado a menos três.

> 5**(-3)

8.0000000000000002e-3

-- Exponencial para inteiros, dois elevado a três elevado por quatro.

```
> 2^3^4  
2417851639229258349412352
```

-- Exponencial para ponto flutuante, dois elevado a três elevado por quatro.

```
> 2**3**4  
2.4178516392292583e24
```

-- Multiplicação e função exponencial, quatro vezes dois elevado por três.

```
> 4*2^3  
32
```

-- Operador de raiz sqrt, raiz da soma de cinco ao quadrado com nove ao quadrado.

```
> sqrt ((5**2) + (9**2))  
10.295630140987
```

-- Operador de raiz, raiz de vinte cinco mais setenta e três, mas como não foi colocado () na operação sqrt(valor) ele apenas executou a operação de soma para ponto flutuante.

```
> sqrt 25 + 73  
78.0
```

-- Função trigonométrica pré-definida da linguagem haskell que imprime o valor do seno, seno de pi dividido por seis.

```
> sin(pi/6)  
0.49999999999999994
```

-- Função trigonométrica pré-definida da linguagem haskell que imprime o valor do cosseno.

```
> cos 0.5  
0.8775825618903728
```

-- Operador de soma, e efetuado a soma de dois mais três e depois e somado com o um.

```
> (+) 1 ((+) 2 3)  
6
```

-- **Operador de comparação**, a multiplicação entre $36*14$ e comparada com a subtração e divisão de $450-23/2$, caso for igual ele responde True e se for diferente ele responde False.

```
> 36*14 == 450-23/2  
False
```

-- **Função pré-definida da linguagem haskell** que imprime o número de elementos de qualquer tipo de lista.

```
> length ['a'..'z']  
26
```

-- **Operador de incremento**, palavra código e e somada com a palavra -fonte.

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

-- **Condicional**, se for verdade que 12 e maior que 5 ele imprime o valor 100 caso seja falso imprime o valor 200.

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

-- **Função pré definida** que calcula a soma dos elementos de uma lista de números, soma de um até 115.

```
> sum[1..115]  
6670
```

-- **Função aritmética** para calcular o logaritmo do valor, logaritmo de 2.718.

```
> log 2.718  
0.999896315728952
```

-- **Função aritmética** para calcular o logaritmo do valor, logaritmo de 10.

```
> log 10  
2.302585092994046
```

-- **Função aritmética** para calcular o logaritmo neperiano, logaritmo neperiano de 2.

```
> exp 2  
7.38905609893065
```

-- Função para impressão de valores inteiros, qualquer operação aritmética o operador imprime apenas valores inteiros sem aproximação, floor na operação EXP 2 teria um valor em float mas o operador imprime apenas o valor inteiro 7.

```
> floor (exp 2)
7
```

-- Função aritmética para calcular o logaritmo do valor, logaritmo do logaritmo neperiano de dois.

```
> log (exp 2)
2.0
```

-- Função aritmética não definida, os valores são adicionados posteriormente, seno de x ao quadrado + cosseno de x ao quadrado, sendo x igual a 2. Observação que a função utilizando where deve ser criada anteriormente.

```
Ex: n = (sin x)^2 + (cos x)^2
      where
      x = 2
```

```
> (sin x)^2 + (cos x)^2 where x = 2
1.0
```

-- Função aritmética não definida, os valores são adicionados posteriormente, pi que multiplica r que multiplica r, sendo r igual a 3. Observação que a função utilizando where deve ser criada anteriormente.

```
Ex: n = pi * r * r
      where r = 3
```

```
> pi * r * r where r = 3
28.274333882308138
```

-- Função definida para operações aritméticas, para representar seu uso foi efetuada a operação de a mais b, e depois a e b recebem os valores de dois e três e depois é efetuado o cálculo de adição. Observação que a função utilizando where deve ser criada anteriormente.

```
Ex: n = add 2 3
      where add a b = a + b
```

```
> add 2 3 where add a b = a + b
5
```

-- Função definida para operações, foi então criada uma função para resolver uma operação de somatório, quatro e cinco recebem o somatório de quatro mais cinco.

Ex: add 4 5 = (+) 4 5

> add 4 5

9

2) Analise a função seguinte escrita em Haskell e explique sua finalidade.

fun m n p = (m==n) && (n==p)

Sua finalidade tem como a função fun receber três valores (m, n e p) e retornar verdadeiro (True) se os três valores forem iguais e falso (False) se forem diferentes.

Alguns exemplos de aplicação são:

Saída:

```
*Main> :l fun.hs
[1 of 1] Compiling Main      ( fun.hs, interpreted )
Ok, one module loaded.
*Main> fun 5 89 3
False
*Main> fun 156 156 156
True
```

3) Sejam as duas funções abaixo que verificam se um dado número é par. Teste cada função e explique a estratégia utilizada na implementação de cada uma.

par x = (mod x 2) == 0

**par1 x = if (x == 0) then True
else not (par1 (x-1))**

Em ambas as funções recebem um número e retornam verdadeiro (True) caso o mesmo for par.

par x = (mod x 2) == 0

Explicando de maneira sucinta a função par utiliza como estratégia obter o resto da divisão do número por dois, e caso seja 0 (zero), o número é par.

```
*Main> par 3
False
```

**par1 x = if (x == 0) then True
else not (par1 (x-1))**

Já a função par1 utiliza uma definição recursiva, em que o caso base é a admissão que 0 (zero) é par, então a partir daí, um número é par se o seu anterior for ímpar (não-par) e assim sucessivamente até que 0 seja atingido e como zero é par, o resultado será verdadeiro para os pares e falso para ímpares.

```
*Main>par1 3
not (par1 2)
not (not (par1 1))
not (not (not (par1 0)))
not (not (not True))
not (not False)
not True
False
```

4) Considere a seguinte função escrita em Haskell:

```
test n = if (n `mod` 2 == 0) then n
        else test(2 * n + 1)
```

Para quais valores de entrada (n) a função não se encerra? Por que? Use exemplos simples para explicar sua resposta.

A função test não se encerra quando aplicada à um número ímpar, pois a condição $(n \text{ `mod` } 2 == 0)$ será falsa, e a chamada recursiva sempre se aplica à outro número ímpar $(2 * n + 1)$.

Exemplo de testes:

```
*Main> test 6
6
*Main> test 7
{Interrupted!}
```

5) Escreva funções para calcular:

(a) Uma equação do primeiro grau ($ax + b$)

`primeirograu::Float->Float->Float`

`primeirograu x y = - y / x`

(b) Uma equação do segundo grau ($ax^2 + bx + c$)

`delta::Float->Float->Float->Float`

`delta a b c = -b^2 * 4 * a * c`

`segundograu::Float->Float->Float->(Float, Float)`

segundograu a b c = if (delta a b c) < 0.0

then undefined

else ((- b - sqrt(delta a b c)) / 2.0 * a, (- b + sqrt(delta a b c)) / 2.0 * a)

6) Construa uma função que calcule o valor do mínimo múltiplo comum de três números inteiros.

main> mmc 2 3 4

12

Saída:

Prelude> :l mmc.hs

[1 of 1] Compiling Main (mmc.hs, interpreted)

Ok, one module loaded.

*Main> mmc 3 4 5

60

*Main> mmc 10 6 2

30

Código da função.

-- Programa com função de calcular o mínimo múltiplo comum de três números inteiros.

-- mmc valor1 valor2 valor3

mdc::Int->Int->Int

mdc a b | a < b = mdc b a

 | b == 0 = a

 | otherwise = mdc b (mod a b)

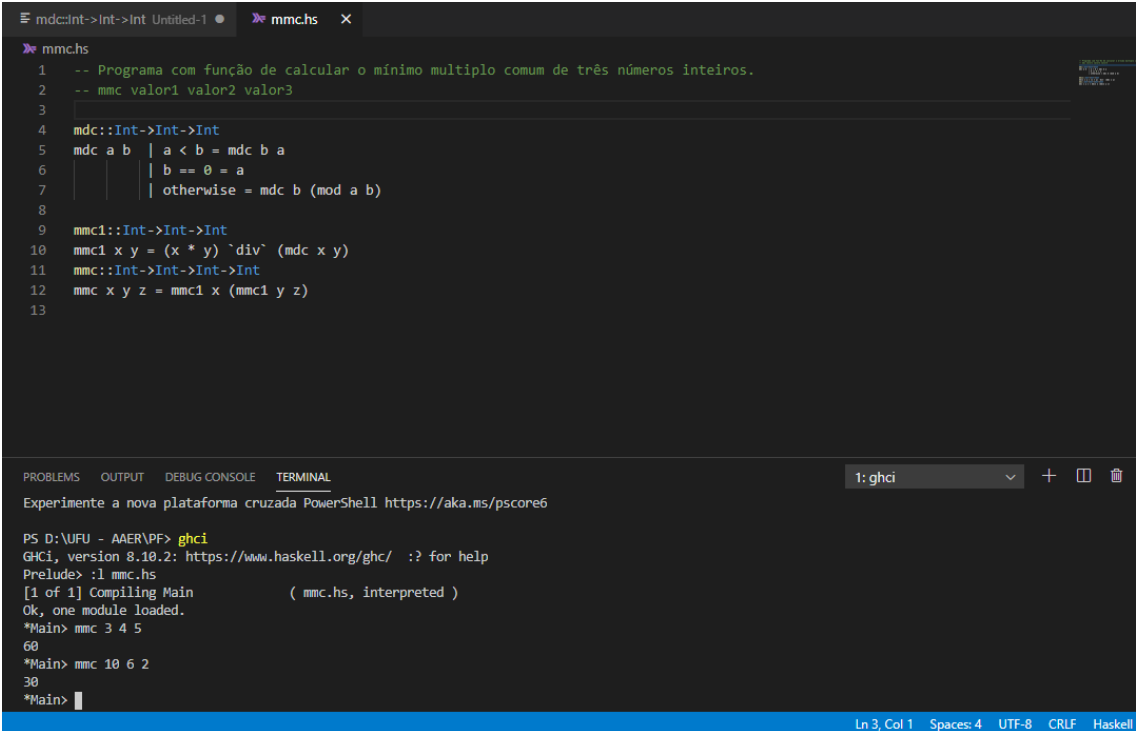
mmc1::Int->Int->Int

mmc1 x y = (x * y) `div` (mdc x y)

mmc::Int->Int->Int->Int

mmc x y z = mmc1 x (mmc1 y z)

Imagem da função executando mmc.hs.



```
mmc.hs
1  -- Programa com função de calcular o mínimo múltiplo comum de três números inteiros.
2  -- mmc valor1 valor2 valor3
3
4  mdc::Int->Int->Int
5  mdc a b | a < b = mdc b a
6          | b == 0 = a
7          | otherwise = mdc b (mod a b)
8
9  mmc1::Int->Int->Int
10 mmc1 x y = (x * y) `div` (mdc x y)
11 mmc::Int->Int->Int->Int
12 mmc x y z = mmc1 x (mmc1 y z)
13
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Experimente a nova plataforma cruzada PowerShell https://aka.ms/pscore6

PS D:\UFU - AAER\PF> ghci
GHCi, version 8.10.2: https://www.haskell.org/ghc/  :? for help
Prelude> :l mmc.hs
[1 of 1] Compiling Main             ( mmc.hs, interpreted )
Ok, one module loaded.
*Main> mmc 3 4 5
60
*Main> mmc 10 6 2
30
*Main>
```

7) Construa uma função que calcule o valor do máximo divisor comum entre três números inteiros.

main> mdc 2 3 4

1

Saída:

*Main> :l mdc.hs

[1 of 1] Compiling Main (mdc.hs, interpreted)

Ok, one module loaded.

*Main> mdc 3 4 5

1

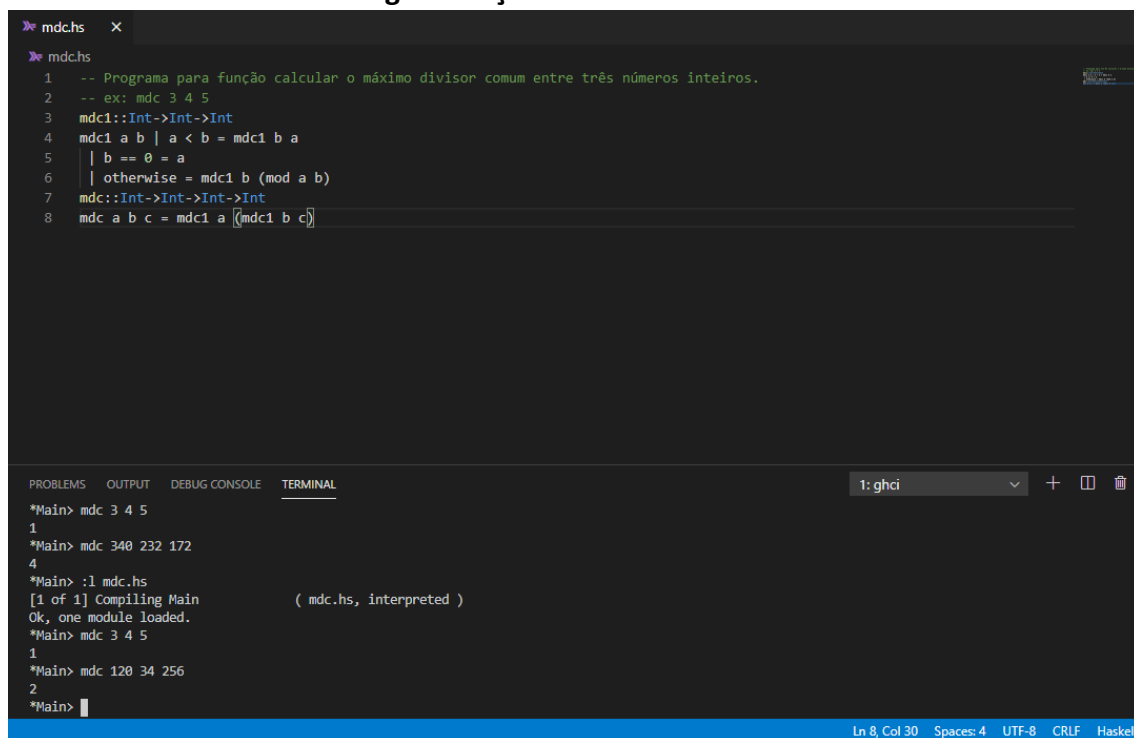
*Main> mdc 120 34 256

2

Código da função.

```
-- Programa para função calcular o máximo divisor comum entre três números inteiros.  
-- ex: mdc 3 4 5  
  
mdc1::Int->Int->Int  
mdc1 a b | a < b = mdc1 b a  
| b == 0 = a  
| otherwise = mdc1 b (mod a b)  
  
mdc::Int->Int->Int->Int  
mdc a b c = mdc1 a (mdc1 b c)
```

Imagem função executando mdc.hs.



The screenshot shows a Haskell IDE with a file named `mdc.hs` open. The code in the editor is as follows:

```
1 -- Programa para função calcular o máximo divisor comum entre três números inteiros.  
2 -- ex: mdc 3 4 5  
3 mdc1::Int->Int->Int  
4 mdc1 a b | a < b = mdc1 b a  
5 | b == 0 = a  
6 | otherwise = mdc1 b (mod a b)  
7 mdc::Int->Int->Int->Int  
8 mdc a b c = mdc1 a (mdc1 b c)
```

The terminal at the bottom shows the execution of the program:

```
*Main> mdc 3 4 5  
1  
*Main> mdc 340 232 172  
4  
*Main> :l mdc.hs  
[1 of 1] Compiling Main ( mdc.hs, interpreted )  
Ok, one module loaded.  
*Main> mdc 3 4 5  
1  
*Main> mdc 120 34 256  
2  
*Main> 
```

The status bar at the bottom indicates the current position is `Ln 8, Col 30`, with `Spaces: 4`, `UTF-8` encoding, `CRLF` line endings, and the `Haskell` language.

8) A sequência de Fibonacci é dada pela seguinte série:

0 1 1 2 3 5 8 13 ...

Construa uma função para retornar o n-ésimo termo da sequência.

```
main> fibonacci 6
```

8

Saída:

```
*Main> :l fibonacci
```

```
[1 of 1] Compiling Main      ( fibonacci.hs, interpreted )
```

```
Ok, one module loaded.
```

```
*Main> fib 8
```

```
21
```

```
*Main> fib 6
```

```
8
```

```
*Main> fib 11
```

```
89
```

Código da função.

```
-----  
-- Programa para calcular a função fibonacci  
-- A sequência de Fibonacci é dada pela seguinte série:  
-- 0 1 1 2 3 5 8 13 ...  
-----
```

```
fib :: Int -> Int
```

```
fib x
```

```
  | x == 0 = 0
```

```
  | x == 1 = 1
```

```
  | x > 0 = fibonacci (x-1) 0 1
```

```
  where
```

```
    fibonacci :: Int -> Int -> Int -> Int
```

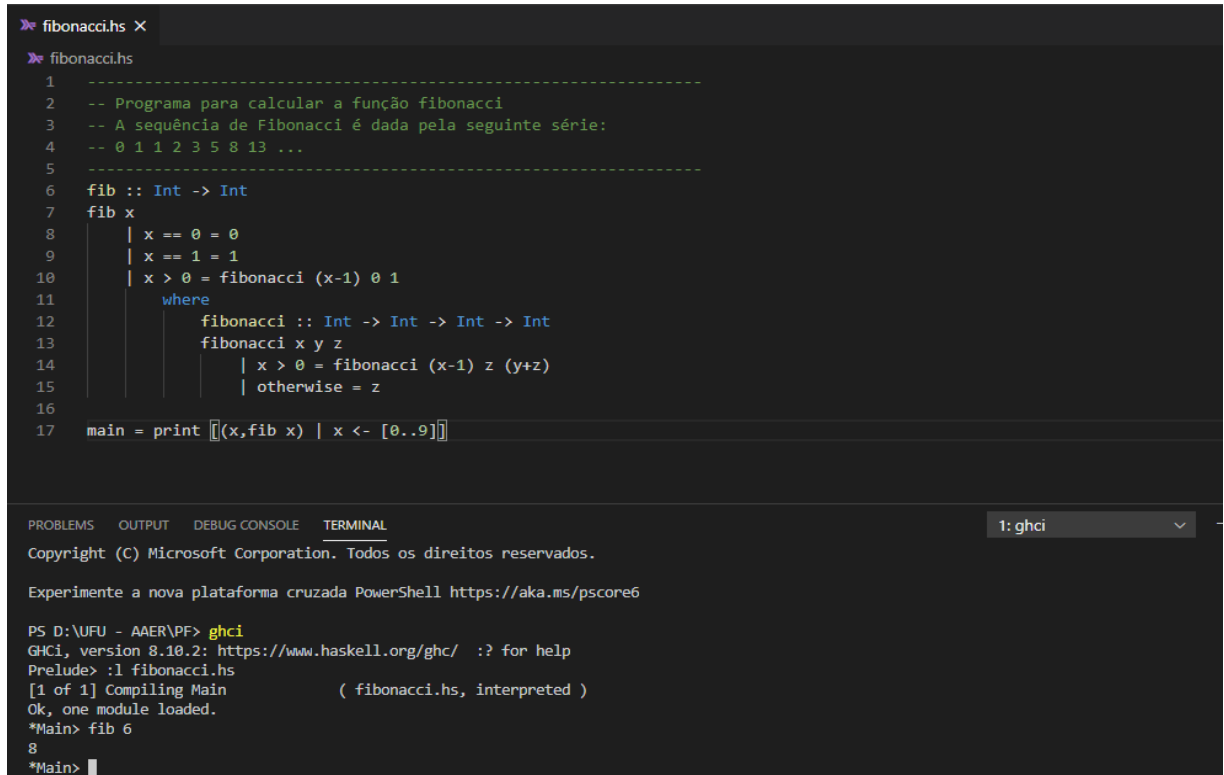
```
    fibonacci x y z
```

```
      | x > 0 = fibonacci (x-1) z (y+z)
```

```
      | otherwise = z
```

```
main = print [(x,fib x) | x <- [0..9]]
```

Imagem da função executando Fibonacci.hs.



```
fibonacci.hs
fibonacci.hs
1  -----
2  -- Programa para calcular a função fibonacci
3  -- A sequência de Fibonacci é dada pela seguinte série:
4  -- 0 1 1 2 3 5 8 13 ...
5  -----
6  fib :: Int -> Int
7  fib x
8  | x == 0 = 0
9  | x == 1 = 1
10 | x > 0 = fibonacci (x-1) 0 1
11     where
12         fibonacci :: Int -> Int -> Int -> Int
13         fibonacci x y z
14         | x > 0 = fibonacci (x-1) z (y+z)
15         | otherwise = z
16
17 main = print [(x,fib x) | x <- [0..9]]

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: ghci
Copyright (C) Microsoft Corporation. Todos os direitos reservados.

Experimente a nova plataforma cruzada PowerShell https://aka.ms/pscore6

PS D:\UFU - AAER\PF> ghci
GHCi, version 8.10.2: https://www.haskell.org/ghc/  :? for help
Prelude> :l fibonacci.hs
[1 of 1] Compiling Main             ( fibonacci.hs, interpreted )
Ok, one module loaded.
*Main> fib 6
8
*Main>
```

9) Faça uma função que, dado um ano, verifica se o mesmo é bissexto.

Saída na tela:

Prelude> :l bissexto.hs

[1 of 1] Compiling Main (bissexto.hs, interpreted)

Ok, one module loaded.

*Main> bissexto 2018

False

*Main> bissexto 2019

False

*Main> bissexto 2020

True

*Main>

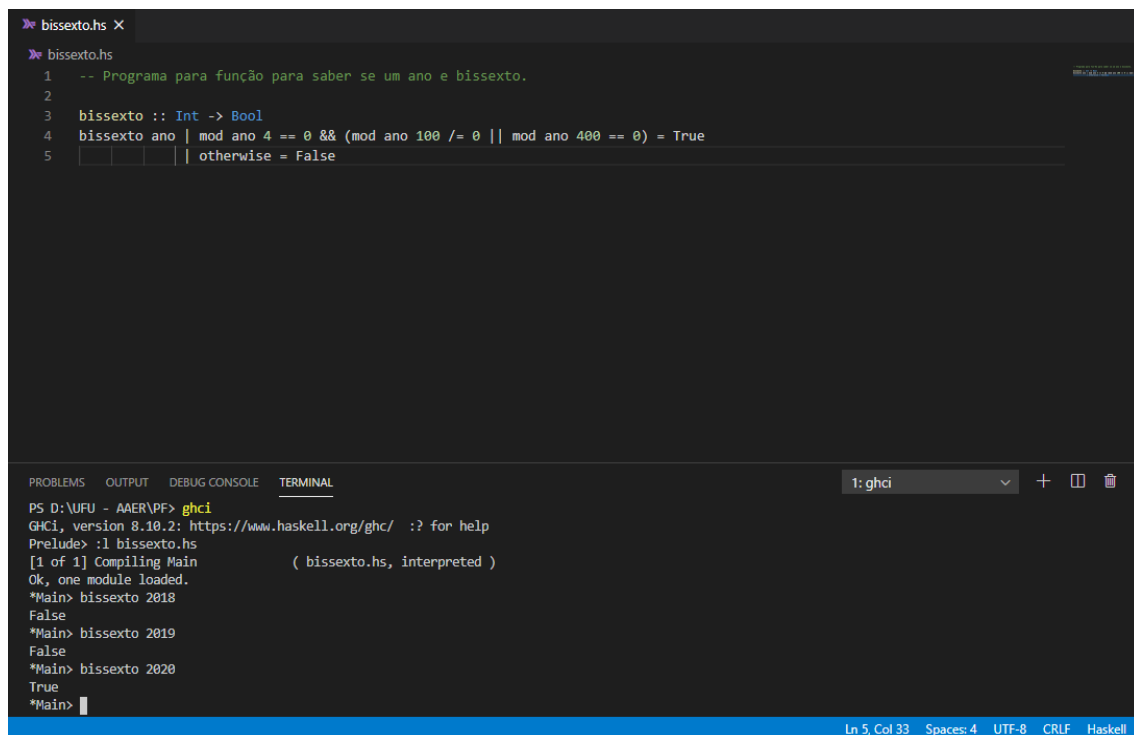
Código da função.

-- Programa para função para saber se um ano e bissexto.

```
bissexto :: Int -> Bool
```

```
bissexto ano | mod ano 4 == 0 && (mod ano 100 /= 0 || mod ano 400 == 0) = True  
            | otherwise = False
```

Imagem da execução da função bissexto.hs



```
bissexto.hs X  
bissexto.hs  
1  -- Programa para função para saber se um ano e bissexto.  
2  
3  bissexto :: Int -> Bool  
4  bissexto ano | mod ano 4 == 0 && (mod ano 100 /= 0 || mod ano 400 == 0) = True  
5  | otherwise = False  
  
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  
PS D:\UFU - AAER\PF> ghci  
GHCi, version 8.10.2: https://www.haskell.org/ghc/ :? for help  
Prelude> :l bissexto.hs  
[1 of 1] Compiling Main  
Ok, one module loaded.  
*Main> bissexto 2018  
False  
*Main> bissexto 2019  
False  
*Main> bissexto 2020  
True  
*Main>   
Ln 5, Col 33  Spaces: 4  UTF-8  CRLF  Haskell
```

10) Defina uma função que recebe três números inteiros representando, respectivamente, um dia, um mês e um ano e verifica se os números formam uma data válida.

Saída:

```
PS D:\UFU - AAER\PF> ghci
GHCi, version 8.10.2: https://www.haskell.org/ghc/ :? for help
Prelude> :l datavalida.hs
[1 of 1] Compiling Main          ( datavalida.hs, interpreted )
Ok, one module loaded.
*Main> valida (28,02,2020)
True
*Main> valida (29,02,2020)
True
*Main> valida (29,02,2021)
False
*Main> valida (29,02,2018)
False
*Main> valida (29,02,2016)
True
*Main>
```

Código da função.

```
-----
-- Programa com a função de validade de uma data.
-- valida (dia,mês,ano)
-----
```

```
type Data = (Int,Int,Int)
```

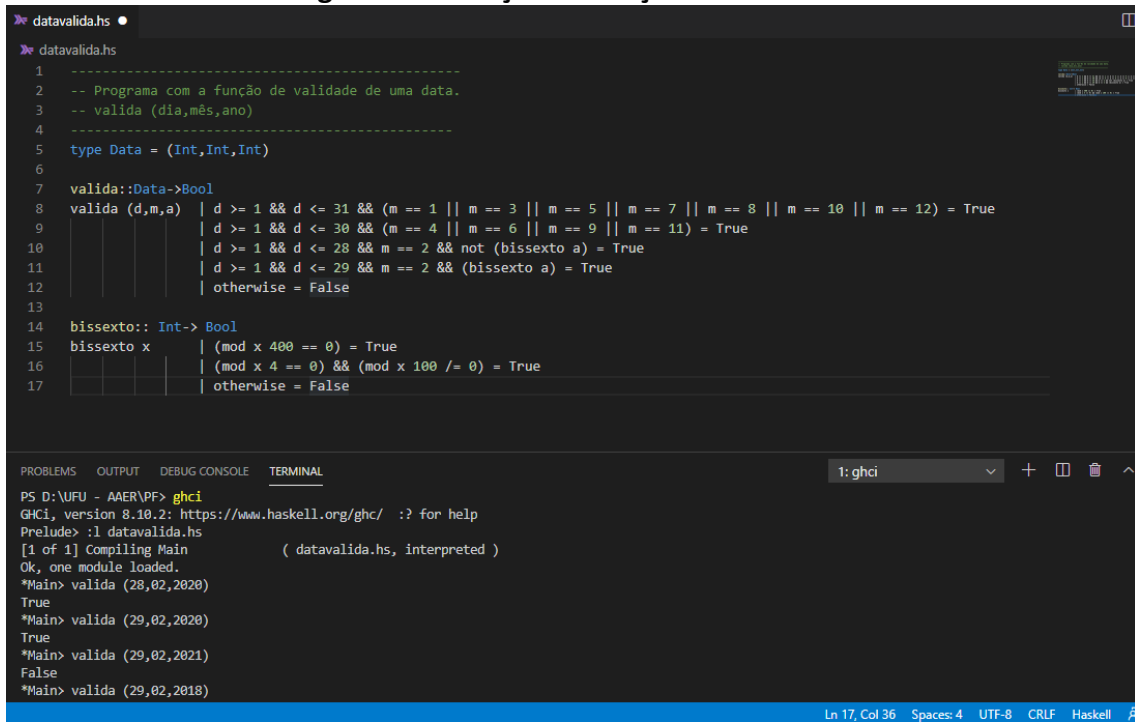
```
valida::Data->Bool
```

```
valida (d,m,a) | d >= 1 && d <= 31 && (m == 1 || m == 3 || m == 5 || m == 7 || m == 8
|| m == 10 || m == 12) = True
               | d >= 1 && d <= 30 && (m == 4 || m == 6 || m == 9 || m == 11) = True
               | d >= 1 && d <= 28 && m == 2 && not (bissexto a) = True
               | d >= 1 && d <= 29 && m == 2 && (bissexto a) = True
               | otherwise = False
```

```
bissexto:: Int-> Bool
```

```
bissexto x    | (mod x 400 == 0) = True
               | (mod x 4 == 0) && (mod x 100 /= 0) = True
               | otherwise = False
```

Imagem da execução da função datavalida.hs.



```
1 -----
2 -- Programa com a função de validade de uma data.
3 -- valida (dia,mês,ano)
4 -----
5 type Data = (Int,Int,Int)
6
7 valida::Data->Bool
8 valida (d,m,a) | d >= 1 && d <= 31 && (m == 1 || m == 3 || m == 5 || m == 7 || m == 8 || m == 10 || m == 12) = True
9                | d >= 1 && d <= 30 && (m == 4 || m == 6 || m == 9 || m == 11) = True
10               | d >= 1 && d <= 28 && m == 2 && not (bissexto a) = True
11               | d >= 1 && d <= 29 && m == 2 && (bissexto a) = True
12               | otherwise = False
13
14 bissexto:: Int-> Bool
15 bissexto x   | (mod x 400 == 0) = True
16             | (mod x 4 == 0) && (mod x 100 /= 0) = True
17             | otherwise = False
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

1: ghci

```
PS D:\UFU - AAER\PF> ghci
GHCi, version 8.10.2: https://www.haskell.org/ghc/  :? for help
Prelude> :l datavalida.hs
[1 of 1] Compiling Main                ( datavalida.hs, interpreted )
Ok, one module loaded.
*Main> valida (28,02,2020)
True
*Main> valida (29,02,2020)
True
*Main> valida (29,02,2021)
False
*Main> valida (29,02,2018)
```

Ln 17, Col 36 Spaces: 4 UTF-8 CRLF Haskell

Visual Studio Code interface showing a Haskell file named `datavalida.hs` and its execution in the terminal.

datavalida.hs

```
1 -----
2 -- Programa com a função de validade de uma data.
3 -- valida (dia,mês,ano)
4 -----
5 type Data = (Int,Int,Int)
6
7 valida::Data->Bool
8 valida (d,m,a) | d >= 1 && d <= 31 && (m == 1 || m == 3 || m == 5 || m == 7 || m == 8 || m == 10 || m == 12) = True
9                  | d >= 1 && d <= 30 && (m == 4 || m == 6 || m == 9 || m == 11) = True
10                 | d >= 1 && d <= 28 && m == 2 && not (bissexto a) = True
11                 | d >= 1 && d <= 29 && m == 2 && (bissexto a) = True
12                 | otherwise = False
13
14 bissexto:: Int-> Bool
15 bissexto x    | (mod x 400 == 0) = True
16               | (mod x 4 == 0) && (mod x 100 /= 0) = True
17               | otherwise = False
```

Terminal Output:

```
1: ghci
Experiencie a nova plataforma cruzada PowerShell https://aka.ms/powershell
PS D:\UFU - AAER\PF> ghci
GHCi, version 8.10.2: https://www.haskell.org/ghc/  :? for help
Prelude> :l datavalida.hs
[1 of 1] Compiling Main                ( datavalida.hs, interpreted )
Ok, one module loaded.
*Main> valida (29,02,1987)
False
*Main> valida (29,02,1988)
True
*Main>
```

Ln 17, Col 36 Spaces: 4 UTF-8 CRLF Haskell