Logičko i funkcijsko programiranje

VEŽBE 3

vezbe3.1.hs

```haskell
data Temp = Cold | Hot
    deriving (Eq, Ord, Enum, Show, Read)
data Season = Spring | Summer | Autumn | Winter
    deriving (Eq, Ord, Enum, Show, Read)

weather :: Season -> Temp
weather Summer = Hot
weather _ = Cold

```

Enumerisani tipovi

vezbe3.2.hs ×

vezbe3.2.hs

```haskell
data People = Person Name Age
type Name = String
type Age = Int

showPerson:: People -> String
showPerson (Person st n) = st ++ " -- " ++ show n

```

Vektorski tipovi

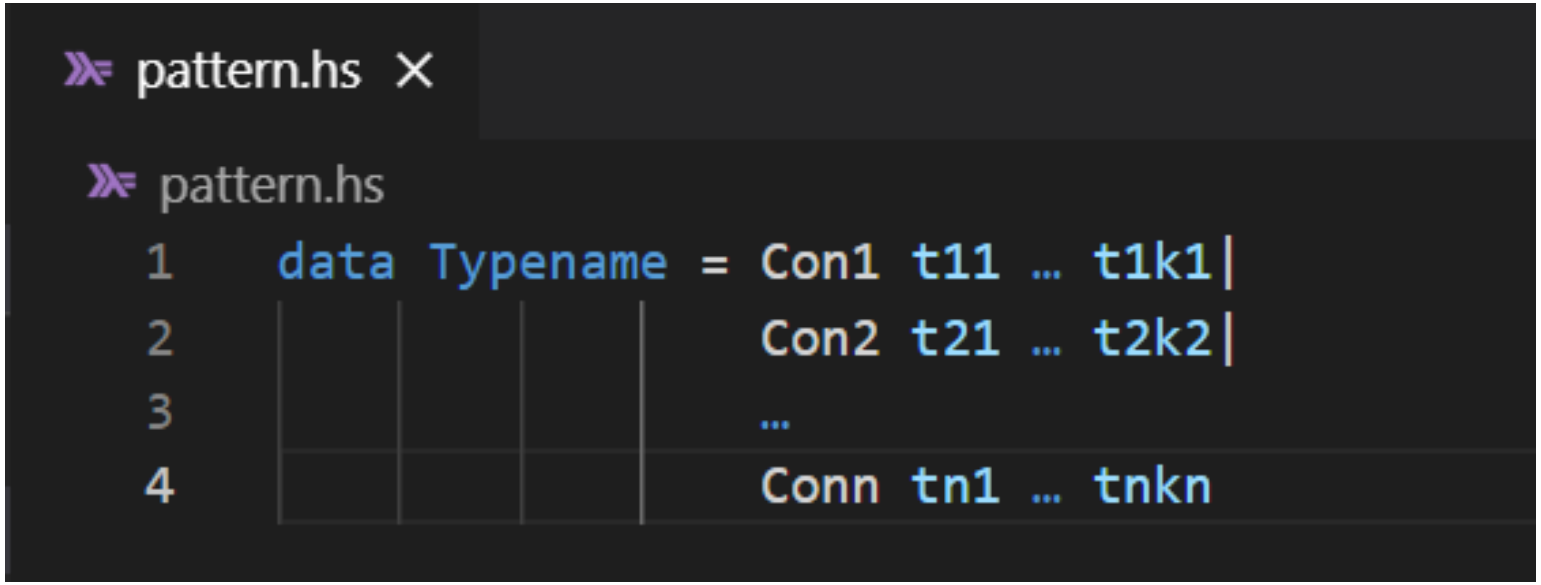
showPerson (Person "Pera Peric" 22)

Vektorski tipovi

area (Circle 3)

vezbe3.3.hs

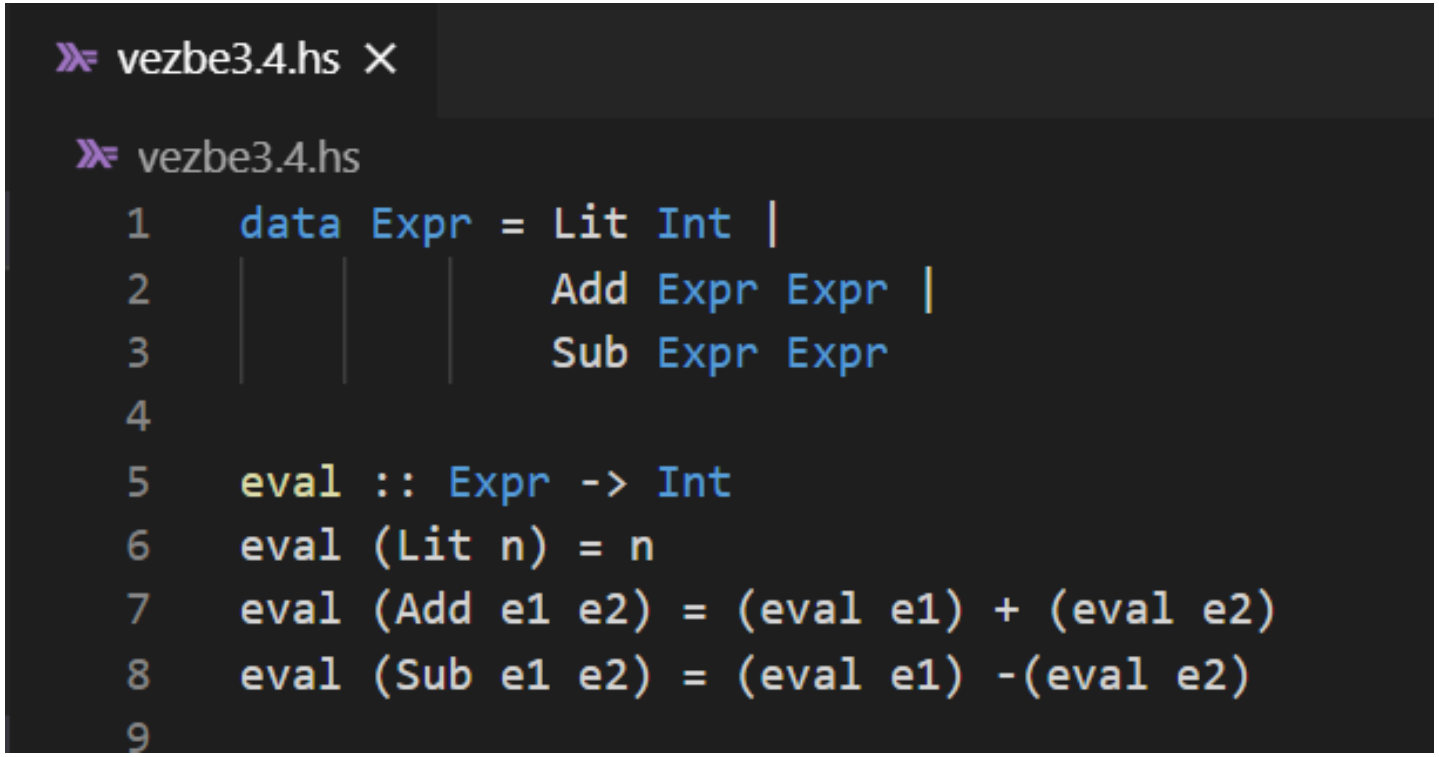
```haskell
data Shape = Circle Float |
    Rectangle Float Float

isRound :: Shape -> Bool
isRound (Circle _) = True
isRound (Rectangle _ _) = False

area :: Shape -> Float
area (Circle r) = pi * r * r
area (Rectangle h w) = h * w
```



Opšti oblik definicije tipova

Rekurzivni tipovi

eval (Add (Lit 3) (Lit 4))



Stablo celih brojeva

sumTree (Node 3 (Node 4 NilT NilT) NilT)

depth (Node 3 (Node 4 NilT NilT) NilT)

vezbe3.5.hs

```haskell
data NTree= NilT |
    Node Int NTree NTree
sumTree, depth :: NTree-> Int
sumTree NilT = 0
sumTree (Node n t1 t2) = n + sumTreet1 + sumTreet2

depth NilT  = 0
depth (Node n t1 t2) = 1 + max (depth t1) (depth t2)
```

vezbe3.6.hs

```haskell
data Tree a = Nil | Node a (Tree a) (Tree a)
              deriving (Eq, Ord, Show, Read)

depth :: Tree a -> Int
depth Nil = 0
depth (Node n t1 t2) = 1 + max (depth t1) (depth t2)
```

Opšta binarna stabla

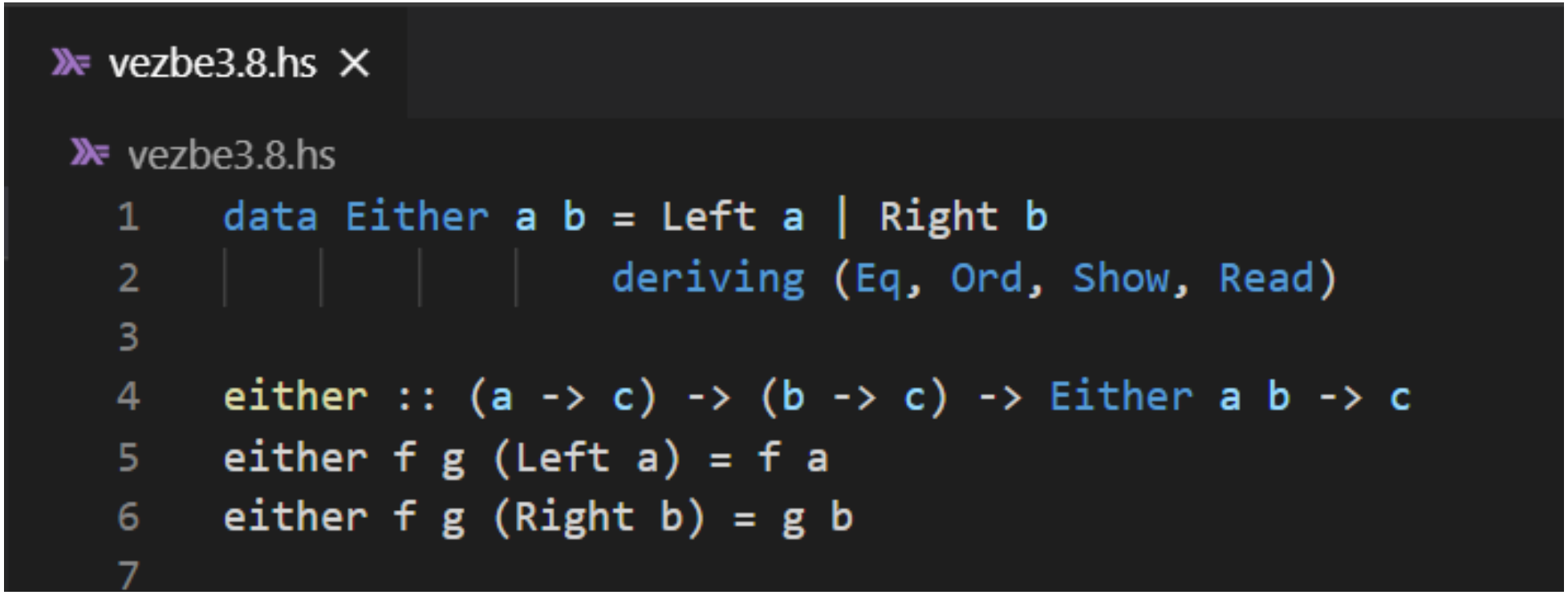
depth (Node 3 (Node 4 Nil Nil) Nil)

Opšta binarna stabla

mapTree (*2) (Node 4 (Node 3 Nil Nil) Nil)

vezbe3.7.hs

```haskell
data Tree a = Nil | Node a (Tree a) (Tree a)
              deriving (Eq, Ord, Show, Read)


mapTree:: (a -> b) -> Tree a -> Tree b
mapTree f Nil = Nil
mapTree f (Node x t1 t2)
    = Node (f x) (mapTree f t1) (mapTree f t2)

```


vezbe3.8.hs
vezbe3.8.hs
1 data Either a b = Left a | Right b
2 deriving (Eq, Ord, Show, Read)
3
4 either :: (a -> c) -> (b -> c) -> Either a b -> c
5 either f g (Left a) = f a
6 either f g (Right b) = g b
7

Unija

Lokalne definicije

sumSquares 3 5

vezbe3.9.hs

```haskell
sumSquares:: Int-> Int-> Int

sumSquares n m = sqN + sqM
                 where
                 sqN = n*n
                 sqM = m*m
```

Lokalna definicija

vezbe3.10.hs

```haskell
f :: Int-> Int-> Int
f m n
    | notNil xs = front xs
    | otherwise = n
    where
    xs= [m .. n]

front :: [Int] -> Int
front (x:y:zs) = x+y
front [x] = x

notNil :: [a] -> Bool
notNil [] = False
notNil (_:_) = True
```

Lokalne definicije

pattern2.hs

pattern2.hs

```
1    f p1 p2... pk
2        | g1= e1
3        ...
4        | otherwise = er
5        where
6        v1  a1... an = r1
7        v2= r2
8        ...
9
```

Lokalna definicija – let izraz

let x = 3+2 in x^2+2*x-4

let x = 3+2 ; y = 5-1 in x^2 + 2*x -y

Vektori i matrice

scaleProduct [1,2,3] [4,5,6]

scaleProduct [1,2,3] [4,5,6,7]

vezbe3.11.hs

```haskell
type Vector = [Float]
scaleProduct:: Vector -> Vector -> Float
scaleProduct xs ys=
    sum [ x*y | (x,y) <- zip xs ys]

```

Vektori i matrice

matrixProduct [[2,3,4],[5,6,-1]] [[1,0],[1,1],[0,-1]]

vezbe3.11.hs

vezbe3.11.hs

```haskell
type Matrix = [Vector]
matrixProduct:: Matrix -> Matrix -> Matrix
matrixProduct m p
    = [[scaleProduct r c | c <-columns p] | r <-m]

columns :: Matrix -> Matrix
columns y = [[z!!j | z <-y] | j <-[0 .. a]]
            where
            a = length (head y)-1
```

Beskonačne liste

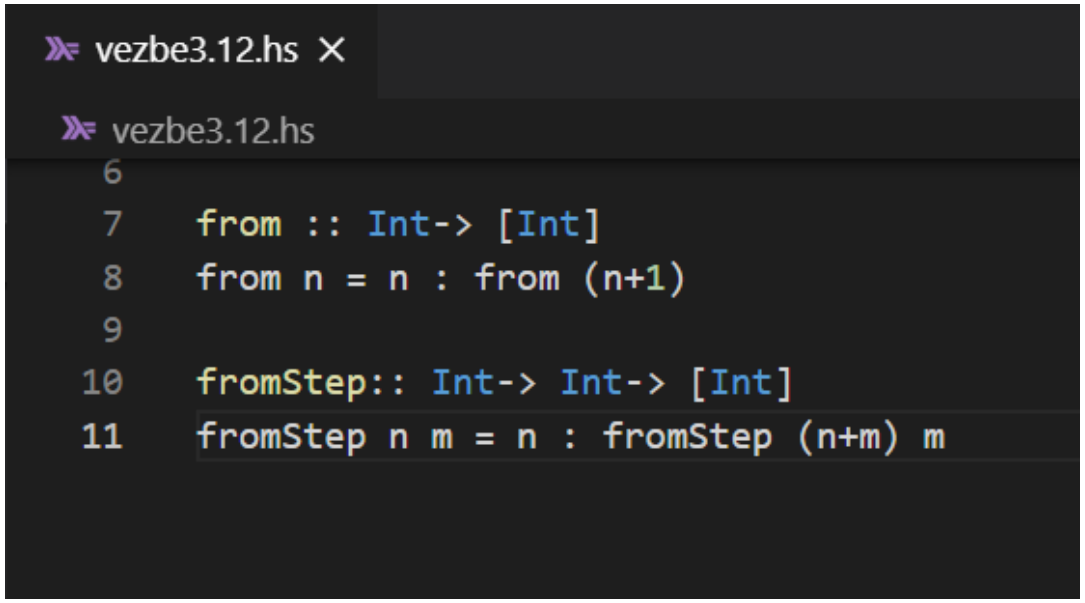
ones

addFirstTwo ones

vezbe3.12.hs

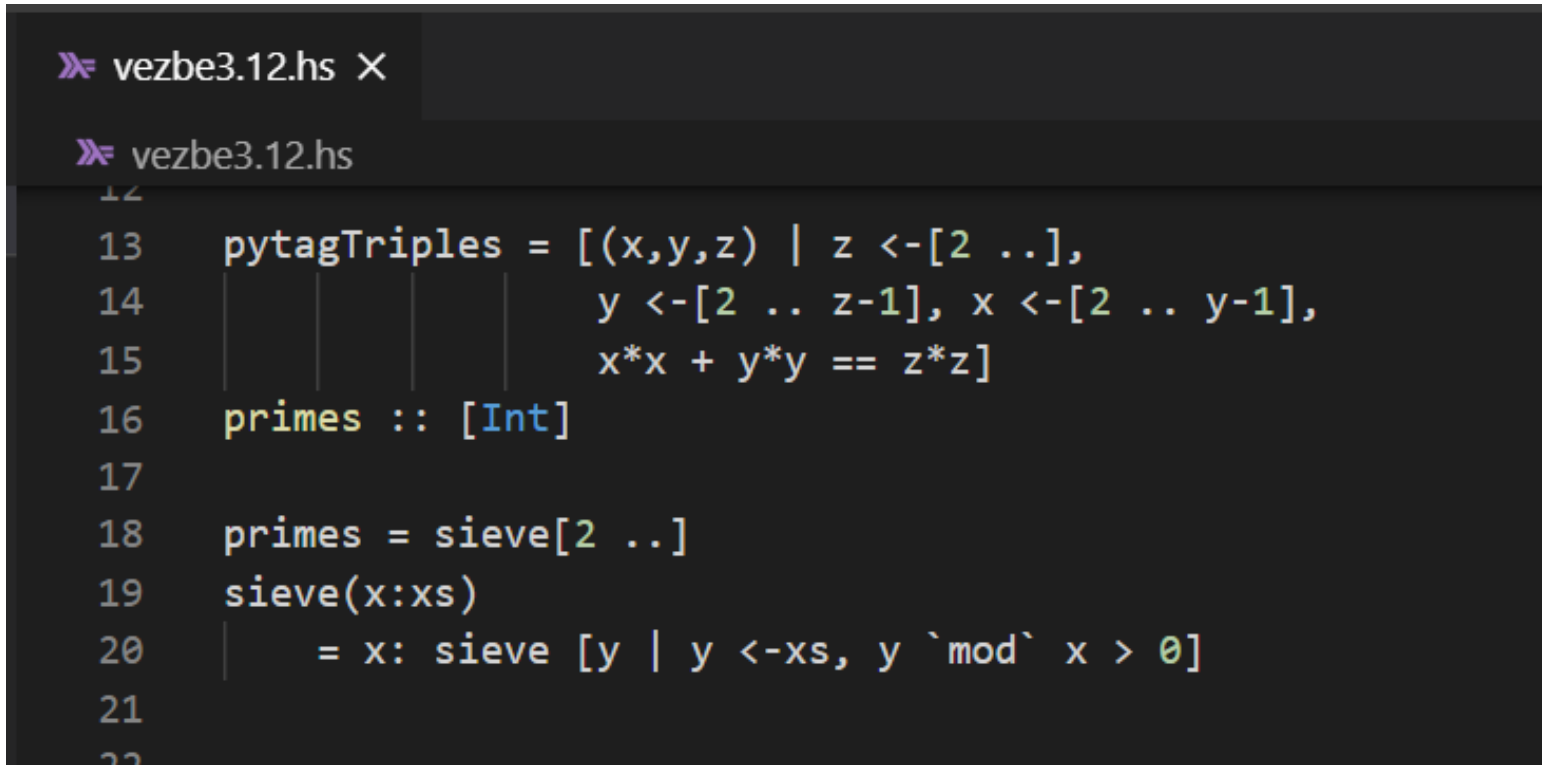
```haskell
ones :: [Int]
ones = 1 : ones

addFirstTwo:: [Int] -> Int
addFirstTwo (x:y:zs) = x+y
```

Beskonačne liste



from 5

fromStep 5 5



Beskonačne liste

vezbe3.13.hs

```haskell
putStr:: String -> IO () --ugrađena funkcija
helloWorld:: IO ()
helloWorld = putStr "Hello, World"

putStrLn:: String -> IO ()--ugrađena funkcija
putStrLn= putStr.(++ "\n")
```

Ispis stringova

helloWorld

putStrLn “HelloWOrld”

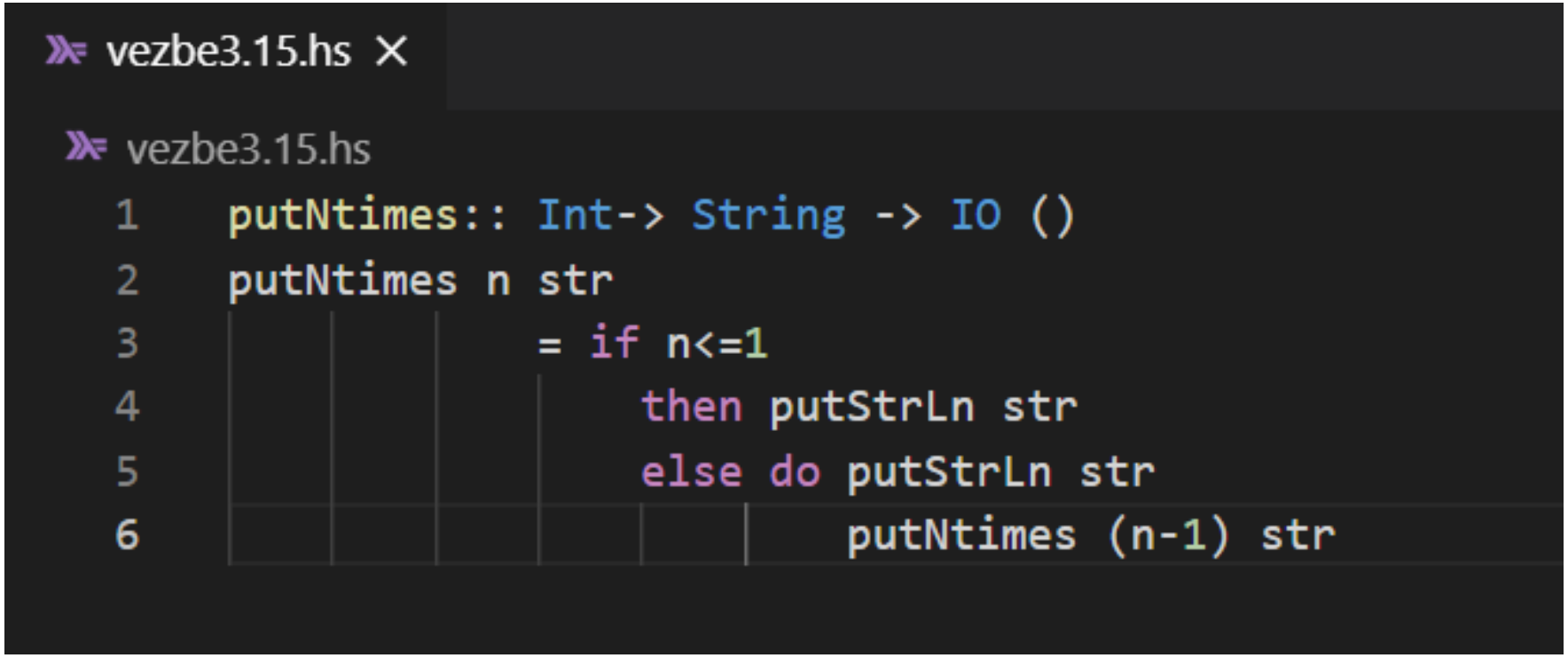
Komanda do

OMOGUĆAVA IZVRŠAVANJE VIŠE KOMANDI

vezbe3.14.hs

vezbe3.14.hs

```haskell
putStrLn :: String -> IO ()
putStrLn str = do putStr str
                  putStr "\n"
```



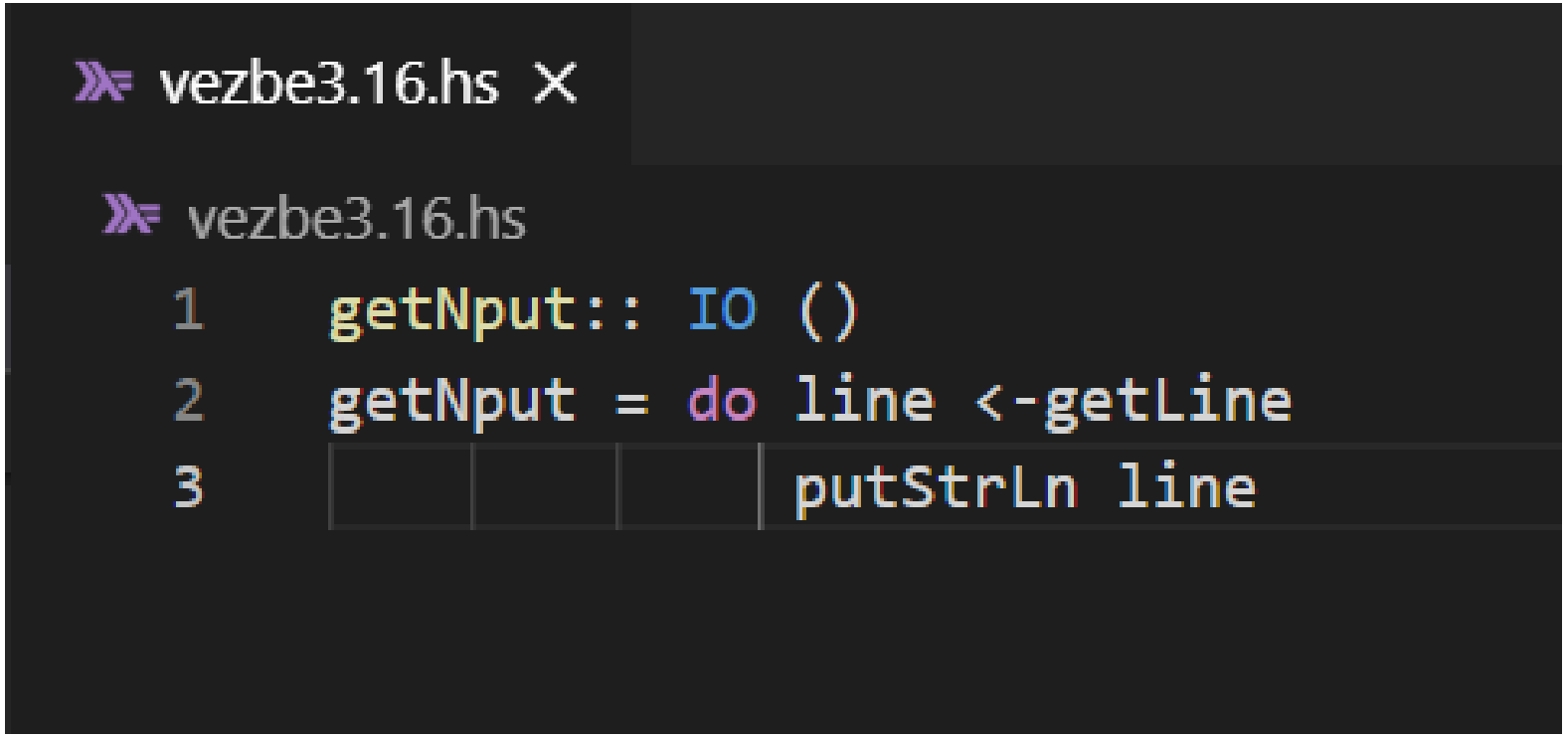
Komanda do

putNtimes 5 “Hello, World”

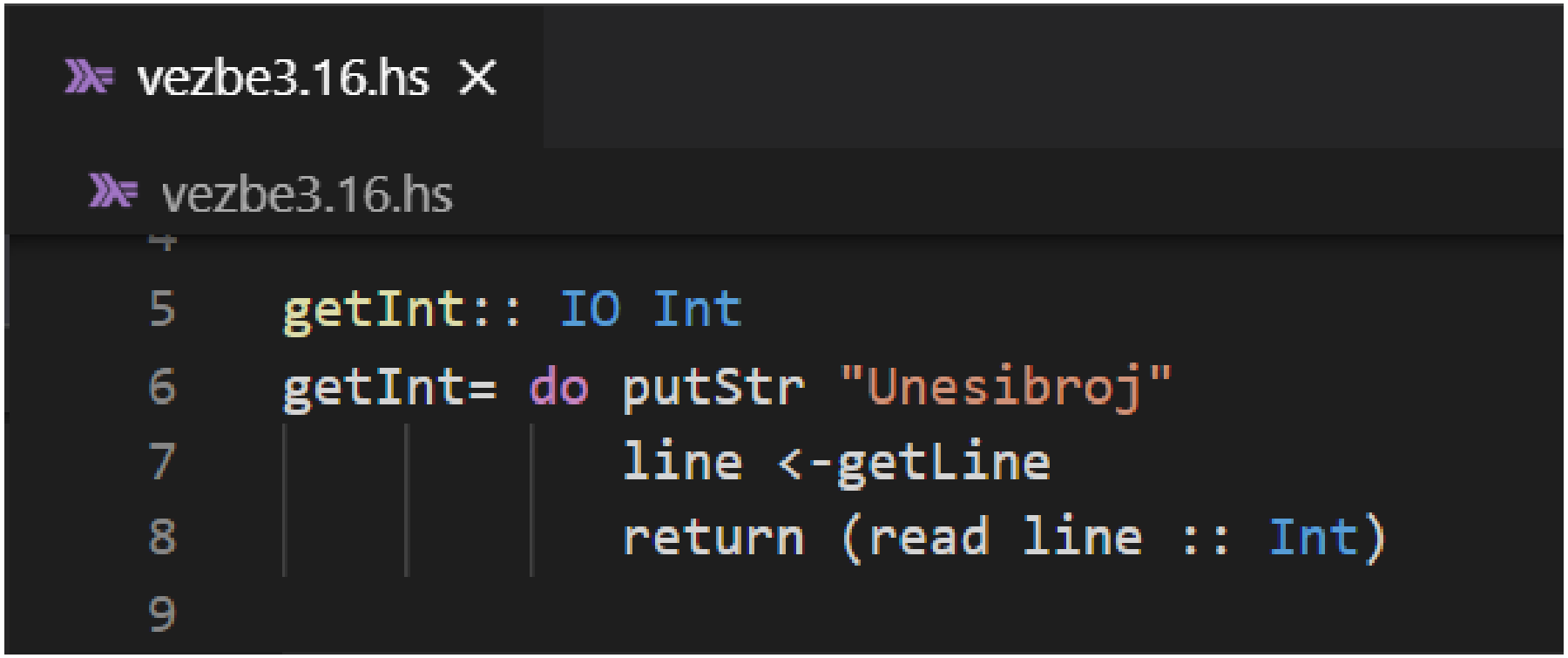
getLine :: IO String

getChar:: IO Char

Čitanje ulaza


vezbe3.16.hs
vezbe3.16.hs
1 getNput:: IO ()
2 getNput = do line <-getLine
3 putStrLn line

Komanda do



Učitavanje celog broja

Suma brojeva

vezbe3.16.hs

```haskell
sumInts:: IO Int
sumInts
    = do n <-getInt
         if n == 0
         then return 0
         else (do m <-sumInts
                  return (n+m))
sumabrojeva:: IO ()
sumabrojeva= do s <- sumInts
                print s
```

Domaći

Broj pojavljivanja čvora u stablu `brojPojavljivanja :: a -> Tree a -> Int`

Prevođenje binarnog stabla u listu brojeva `treeToList :: Tree a -> [a]`

Lista faktorijela prirodnih brojeva `fakt :: [Int]`

Lista Fibonačijevih brojeva `fib :: [Int]`