

Logičko i funkcijsko programiranje

VEŽBE 3

vezbe3.1.hs X

vezbe3.1.hs

```
1  ∨ data Temp = Cold | Hot
2      deriving (Eq, Ord, Enum, Show, Read)
3  ∨ data Season = Spring | Summer | Autumn | Winter
4      deriving (Eq, Ord, Enum, Show, Read)
5
6  weather :: Season -> Temp
7  weather Summer = Hot
8  weather _ = Cold
9
```

Enumerisani tipovi

vezbe3.2.hs X

vezbe3.2.hs

```
1  data People = Person Name Age
2  type Name = String
3  type Age = Int
4
5  showPerson :: People -> String
6  showPerson (Person st n) = st ++ " -- " ++ show n
7
```

Vektorski tipovi

showPerson (Person "Pera Peric" 22)

Vektorski tipovi

area (Circle 3)

vezbe3.3.hs X

vezbe3.3.hs

```
1  data Shape = Circle Float |
2  |           Rectangle Float Float
3
4  isRound :: Shape -> Bool
5  isRound (Circle _) = True
6  isRound (Rectangle _ _) = False
7
8  area :: Shape -> Float
9  area (Circle r) = pi * r * r
10 area (Rectangle h w) = h * w
11
```

» pattern.hs X

» pattern.hs

```
1 data Typename = Con1 t11 ... t1k1 |  
2   | Con2 t21 ... t2k2 |  
3   | ...  
4   | Conn tn1 ... tnkn
```

Opšti oblik definicije tipova

Rekurzivni tipovi

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

vezbe3.4.hs X

vezbe3.4.hs

```
1  data Expr = Lit Int |
2  |         |         | Add Expr Expr |
3  |         |         | Sub Expr Expr
4
5  eval :: Expr -> Int
6  eval (Lit n) = n
7  eval (Add e1 e2) = (eval e1) + (eval e2)
8  eval (Sub e1 e2) = (eval e1) -(eval e2)
9
```

Stablo celih brojeva

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

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

vezbe3.5.hs

vezbe3.5.hs

```
1 data NTree = NilT |
2   | Node Int NTree NTree
3 sumTree, depth :: NTree -> Int
4 sumTree NilT = 0
5 sumTree (Node n t1 t2) = n + sumTree t1 + sumTree t2
6
7 depth NilT = 0
8 depth (Node n t1 t2) = 1 + max (depth t1) (depth t2)
```

vezbe3.6.hs X

vezbe3.6.hs

```
1  data Tree a = Nil | Node a (Tree a) (Tree a)
2      deriving (Eq, Ord, Show, Read)
3
4  depth :: Tree a -> Int
5  depth Nil = 0
6  depth (Node n t1 t2) = 1 + max (depth t1) (depth t2)
7
```

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 X

vezbe3.7.hs

```
1  data Tree a = Nil | Node a (Tree a) (Tree a)
2  |         |         |         |         deriving (Eq, Ord, Show, Read)
3
4
5  mapTree :: (a -> b) -> Tree a -> Tree b
6  mapTree f Nil = Nil
7  mapTree f (Node x t1 t2)
8  |         = Node (f x) (mapTree f t1) (mapTree f t2)
9
```

vezbe3.8.hs X

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 X

vezbe3.9.hs

```
1  sumSquares :: Int -> Int -> Int
2
3  √ sumSquares n m = sqN + sqM
4      where
5          sqN = n*n
6          sqM = m*m
```

Lokalna definicija

vezbe3.10.hs

vezbe3.10.hs

```
1  f :: Int -> Int -> Int
2  f m n
3      | notNil xs = front xs
4      | otherwise = n
5      where
6          xs = [m .. n]
7
8  front :: [Int] -> Int
9  front (x:y:zs) = x+y
10 front [x] = x
11
12 notNil :: [a] -> Bool
13 notNil [] = False
14 notNil (_:_) = True
```

Lokalne definicije

» pattern2.hs X

» 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 X

vezbe3.11.hs

```
1  type Vector = [Float]
2  scaleProduct :: Vector -> Vector -> Float
3  scaleProduct xs ys =
4  |   sum [ x*y | (x,y) <- zip xs ys]
5
```

Vektori i matrice

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

vezbe3.11.hs X

vezbe3.11.hs

```
5
6  type Matrix = [Vector]
7  matrixProduct :: Matrix -> Matrix -> Matrix
8  matrixProduct m p
9  |   = [[scaleProduct r c | c <- columns p] | r <- m]
10
11  columns :: Matrix -> Matrix
12  columns y = [[z!!j | z <- y] | j <- [0 .. a]]
13  |   |   |   where
14  |   |   |   a = length (head y)-1
15
```

Beskonačne liste

ones

addFirstTwo ones

```
» vezbe3.12.hs X
```

```
» vezbe3.12.hs
```

```
1  ones :: [Int]
```

```
2  ones = 1 : ones
```

```
3
```

```
4  addFirstTwo :: [Int] -> Int
```

```
5  addFirstTwo (x:y:zs) = x+y
```

```
6
```

Beskonačné liste

from 5

fromStep 5 5

vezbe3.12.hs X

vezbe3.12.hs

```
6
7  from :: Int -> [Int]
8  from n = n : from (n+1)
9
10 fromStep :: Int -> Int -> [Int]
11 fromStep n m = n : fromStep (n+m) m
```

vezbe3.12.hs X

vezbe3.12.hs

```
12
13 pytagTriples = [(x,y,z) | z <- [2 ..],
14 | y <- [2 .. z-1], x <- [2 .. y-1],
15 | x*x + y*y == z*z]
16 primes :: [Int]
17
18 primes = sieve [2 ..]
19 sieve(x:xs)
20 |   = x: sieve [y | y <- xs, y `mod` x > 0]
21
22
```

Beskonačne liste

vezbe3.13.hs X

vezbe3.13.hs

```
1  putStr:: String -> IO () --ugrađena funkcija
2  helloWorld:: IO ()
3  helloWorld = putStr "Hello, World"
4
5  putStrLn:: String -> IO ()--ugrađena funkcija
6  putStrLn= putStr.(++ "\n")
7
```

Ispis stringova

helloWorld

putStrLn "HelloWorld"

Komanda do

OMOGUĆAVA IZVRŠAVANJE VIŠE
KOMANDI

vezbe3.14.hs X

vezbe3.14.hs

1

2 `putStrLn :: String -> IO ()`

3 `putStrLn str = do putStrLn str`

4 `|` `|` `|` `|` `|` `putStrLn "\n"`

5

vezbe3.15.hs

vezbe3.15.hs

```
1  putNtimes :: Int -> String -> IO ()
2  putNtimes n str
3      = if n <= 1
4          then putStrLn str
5          else do putStrLn str
6                putNtimes (n-1) str
```

Komanda do

putNtimes 5 "Hello, World"



`getLine :: IO String`



`getChar:: IO Char`

Čitanje ulaza

» vezbe3.16.hs X

» vezbe3.16.hs

```
1  getNput :: IO ()  
2  getNput = do line <- getLine  
3  putStrLn line
```

Komanda do

vezbe3.16.hs X

vezbe3.16.hs

```
5  getInt :: IO Int
6  getInt = do putStr "Unesibroj"
7             line <- getLine
8             return (read line :: Int)
9
```

Učitavanje celog
broja |

Suma brojeva

vezbe3.16.hs X

vezbe3.16.hs

```
9
10  sumInts :: IO Int
11  sumInts
12      = do n <- getInt
13           if n == 0
14             then return 0
15             else (do m <- sumInts
16                   return (n+m))
17  sumabrojeva :: IO ()
18  sumabrojeva = do s <- sumInts
19                print s
20
```

Domaći

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

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

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

Lista Fibonačijevih brojeva `fib :: [Int]`