

Primitiivtüübid

- Täisarvud: Int, Integer

```
15      :: Int      -18      :: Integer    0014 :: Integer
0o17    :: Int      -0x12    :: Integer    0XC   :: Integer
```

- Ujukomaarvud: Float, Double

```
3.14    :: Float    -7.3e5 :: Float      2.5E-3 :: Double
```

- Aritmeetilisi funktsioone:

```
negate, abs      :: Num a => a -> a
(+), (-), (*)    :: Num a => a -> a -> a
div, mod         :: Integral a => a -> a -> a
(/)              :: Fractional a => a -> a -> a
```

Primitiivtüübid

- Tõeväärtused: Bool

False :: Bool True :: Bool

- Eeldefineeritud funktsioone tõeväärtustega:

(&&), (||) :: Bool -> Bool -> Bool

not :: Bool -> Bool

otherwise :: Bool

(==), (/=) :: Eq a => a -> a -> Bool

(<), (<=) :: Ord a => a -> a -> Bool

(>), (>=) :: Ord a => a -> a -> Bool

Primitiivtüübid

- Sümbolid: Char

'A' '\32' '\x20' '\o40' '\n' '\\ ' '\BEL'

- Eeldefineeritud funktsioone sümbolitega:

isSpace, isDigit, isAlpha :: Char -> Bool

isUpper, isLower :: Char -> Bool

toUpper, toLower :: Char -> Char

ord :: Char -> Int

chr :: Int -> Char

- Asuvad standardteegis Char

Eeldefineeritud tüübikonstruktorid

- Ennikud: $(t_1, \dots, t_n)$

$(1, \text{fact2}, 'A') \quad :: \quad (\text{Int}, \text{Int} \rightarrow \text{Int}, \text{Char})$

$((\text{True}, \text{"Hello"}), ()) \quad :: \quad ((\text{Bool}, \text{String}), ())$

- Eeldefineeritud funktsioone paaridega:

$\text{fst} \quad :: \quad (a,b) \rightarrow a$

$\text{snd} \quad :: \quad (a,b) \rightarrow b$

- fst , snd on polümorfised funktsioonid
- Ennikute dekonstrueerimine näidiste sobitamisega

$\text{thd4 } (x,y,z,w) = z$

Eeldefineeritud tüübikonstruktorid

- Funktsioonid $t1 \rightarrow t2$

`fact2 :: Int -> Int`

`\ x y -> x + 5 * y :: Int -> Int -> Int`

– Nool on parem-, aplikatsioon vasakassotsiatiivne!

- Kõik funktsioonid on üheargumendilised!
- Mitmeargumendilisi funktsioone saab modelleerida
 - “pakkides” argumendid ühte ennikusse
 - kasutades “curried” funktsioone

Mitmeargumendilised funktsioonid

- Näide — gravitatsioonijõud kahe keha vahel

$$g = 6.67e-11$$

```
gravity m1 r m2
  | r == 0.0   = 0.0
  | otherwise = (g*m1*m2)/(r*r)
```

- “Curried” funktsioone saab osaliselt rakendada

```
mass_of_earth = 5.96e24
```

```
radius_of_earth = 6.37e6
```

```
my_mass = 80.0
```

```
earths_force = gravity mass_of_earth
```

```
surface_force = earths_force radius_of_earth
```

```
my_force = surface_force my_mass
```

Operaatorid

- Erisümbolitest koosnevad identifikaatorid on infiks-operaatorid
`!#$%&*+./<=>?@\^|-~:`

- Näide:

$$x \wedge 0 = 1$$

$$x \wedge (n+1) = x * x^n$$

- Vasak-, parem- ja mitteasotsiatiivsed operaatorid:

$$\text{infixl } 6 + \quad \text{infixr } 8 \wedge \quad \text{infix } 4 ==$$

- Lubatud prioriteetidid 0 – 9 (aplikatsiooni prioriteet 10)
- Vaikimisi vasakassotsiatiivne, prioriteediga 9

Operaatorid

- Osaliselt rakendatud operaatorid — sektsioonid

(+1) (/2) (1/) (==0)

- Prefiks-kujul operaatorid

(+) 1 3

- Operaator peab olema prefiks-kujul

- tüübideklaratsioonis
- teise funktsiooni argumendiks olles

- Binaarseid funktsioone saab kasutada infiks-kujul

3 'max' 5

Eeldefineeritud tüübikonstruktorid

- Listid: `[t]`

`[1,2,3,4,5,6] :: [Int]`

`[[1,2],[3],[4,5,6]] :: [[Int]]`

- Stringid: `String`

`"Hello!\n"`

- NB! String on sümbolite list `[Char]`

`['H', 'e', 'l', 'l', 'o', '!', '\n']`

Listide konstrueerimine

- Listide konstrueerimise primitiivid:

- tühilist (`nil`)

`[] :: [a]`

- listi konstruktor (`cons`)

`infixr 5 :`

`(:) :: a -> [a] -> [a]`

- kõik listid on ehitatud `nil` ja `cons` abil

Hugs> `1 : 2 : 3 : 4 : []`

`[1,2,3,4]`

Eeldefineeritud funktsioonide listidel

- Funktsioonide defineerimine näidiste sobitamise

```
head :: [a] -> a
```

```
head (x:xs) = x
```

```
tail :: [a] -> [a]
```

```
tail (x:xs) = xs
```

```
null :: [a] -> Bool
```

```
null [] = True
```

```
null _ = False
```

Eeldefineeritud funktsioone listidel

- Rekursiivselt defineeritud funktsioone

```
length :: [a] -> Int
```

```
length [] = 0
```

```
length (_:xs) = 1 + length xs
```

```
infixr 5 ++
```

```
(++) :: [a] -> [a] -> [a]
```

```
[] ++ ys = ys
```

```
(x:xs) ++ ys = x : (xs ++ ys)
```

Eeldefineeritud funktsioone listidel

- Rekursiivselt defineeritud funktsioone

```
concat :: [[a]] -> [a]
```

```
concat [] = []
```

```
concat (xs:xss) = xs ++ concat xss
```

```
-- infixl 9 !!
```

```
(!!) :: [a] -> Int -> a
```

```
(x:_) !! 0 = x
```

```
(_:xs) !! n | n > 0 = xs !! (n-1)
```

Eeldefineeritud funktsioone listidel

- Rekursiivselt defineeritud funktsioone

`take, drop :: Int -> [a] -> [a]`

`take 0 xs = []`

`take n [] = []`

`take n (x:xs) = x : take (n-1) xs`

`drop 0 xs = xs`

`drop n [] = []`

`drop n (x:xs) = drop (n-1) xs`

`zip :: [a] -> [b] -> [(a,b)]`

`zip (x:xs) (y:ys) = (x,y) : zip xs ys`

`zip _ _ = []`

Eeldefineeritud funktsioone listidel

- Listide ümberpööramine

```
reverse :: [a] -> [a]
```

```
reverse [] = []
```

```
reverse (x:xs) = reverse xs ++ [x]
```

- Sabarekursiivne versioon

```
reverse xs = rev [] xs
```

```
    where rev ys [] = ys
```

```
          rev ys (x:xs) = rev (x:ys) xs
```

Listide konstrueerimine

- “Aritmeetilised” jadad:

```
Hugs> [1..5]
```

```
[1,2,3,4,5]
```

```
Hugs> [2,5..12]
```

```
[2,5,8,11]
```

```
Hugs> [3..1]
```

```
[]
```

```
Hugs> [1..]
```

```
[1,2,3,4,5,6,7, .....
```

```
Hugs> ['A'..'Z']
```

```
"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

Listide konstrueerimine

- Listide ZF-esitus (list comprehension) — notatsioon listi konstrueerimiseks olemasolevatest listidest

```
Hugs> [x^2 | x <- [1..5]]  
[1,4,9,16,25]
```

- Avaldist `x <- [1..5]` nimetatakse generaatoriks
- Ühes konstruktsioonis võib olla mitu generaatorit

```
Hugs> [(x,y) | x <- [1..2], y <- [1..3]]  
[(1,1),(1,2),(1,3),(2,1),(2,2),(2,3)]
```

- Generaatorite järjekorra muutmisel, muutub ka elementide järjekord tulemlistis

```
Hugs> [(x,y) | y <- [1..3], x <- [1..2]]  
[(1,1),(2,1),(1,2),(2,2),(1,3),(2,3)]
```

- Parempoolne generaator muutub vasakpoolsest kiiremini!

Listide konstrueerimine

- Hilisem (parempoolsem) generaator võib sõltuda varasema generaatori poolt sissetoodud muutjast

```
Hugs> [(x,y) | x <- [1..4], y <- [x+1..4]]  
[(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)]
```

- Näide: listide “lamendamine” uuesti

```
concat :: [[a]] -> [a]  
concat xss = [ x | xs <- xss, x <- xs]
```

- Genereeritud väärtuste kitsendamiseks võib kasutada valvureid

```
Hugs> [x * x | x <- [1..10], even x]  
[4,16,36,64,100]  
Hugs> [i | (i,c) <- zip [1..] "AbCD", isUpper c]  
[1,3,4]
```

Näide — algarvude leidmine

- Arvu teguriteks lahutamine

```
factors    :: Int -> [Int]
factors n = [x | x <- [1..n], n `mod` x == 0]
```

- Algarvulisuse kontroll

```
prime      :: Int -> Bool
prime n = null [i | i <- factors n, i /= 1, i /= n]
```

- Algarvud kuni etteantud piirini

```
primes     :: Int -> [Int]
primes n = [x | x <- [2..n], prime x]
```