

LISTIST MAKSIMAALSE ELEMENDI LEIDMINE

```
listMax list@ (x:xs)
  = findMax list (-1)

listMax _
  = 0

findMax (x:xs) currentMax
  | null xs
  =
    if x > currentMax
    then
      x
    else
      currentMax
  | otherwise
  =
    if x > currentMax
    then
      findMax xs x
    else
      findMax xs currentMax
```

SUVALISE NUMBRI ÄRAARVAMINE (IO)

```
module Guess
  where

import IO
import Random

guess
  =
    do
      hSetBuffering stdin
LineBuffering
      num <- randomRIO (1::Int, 100)
      putStr "Guess a number between
1..100: "
      doGuessing num

doGuessing correctNum
  =
    do
      guess <- getLine
      let
        guessNum = read guess
      if guessNum < correctNum
      then
        do
          putStr "\nToo small, try
again: "
          doGuessing correctNum
      else if guessNum > correctNum
      then
        do
          putStr "\nToo big, try
again: "
          doGuessing correctNum
      else
        putStrLn ("You got it, the
number was indeed " ++
show(correctNum))
```

NIME KÜSIMINE JA VÄLJASTAMINE (IO)

```
name
  =
    do
      hSetBuffering stdin
LineBuffering
      putStr "Please enter your name:
"
      name <- getLine
      putStrLn ("> Hello " ++ name ++
"!")
```

RUUTVÖRRANDI NULLKOHTADE LEIDMINE

```
roots a b c
  =
    let
      det = sqrt (b*b - 4*a*c)
      twice_a = 2 * a
    in
      ((-b + det) / twice_a,
      (-b - det) / twice_a)
```

STRING BOOLEANITEKS, MAP, KOKKULUGEMINE

```
import Char
import IO

strToBool str
  = map charToBool str

charToBool char
  | Char.isLower char
  = True
  | otherwise
  = False

numLowerCase str
  = numBoolTrue(strToBool str)

numBoolTrue []
  = 0

numBoolTrue (x:xs)
  | x == True
  = 1 + numBoolTrue xs
  | otherwise
  = numBoolTrue xs
```

SIGNUM, IF-ELSE

```
signum x
  =
    if x < 0
    then -1
    else if x > 0
    then 1
    else
      0
```

NUMBRITE LUGEMINE, SUMMA, FAKTORIAAL

```
module Numbers
  where
import IO

main
  =
  do
    hSetBuffering stdin
LineBuffering
    putStrLn "Enter numbers (0 to
stop):"
    numbers <- readNumbers
    putStrLn ("You entered: " ++
show(numbers))
    putStrLn ("> sum: " ++ show
(findSum(numbers)))
    putStrLn ("> product: " ++
(show . findProduct) numbers)
    printFactorials numbers

readNumbers
  =
  do
    input <- getLine
    let
      number = read input
    if number == 0
      then
        return []
      else
        do
          rest <- readNumbers
          return (number : rest)

findSum numbers
  =
  foldr (+) 0 numbers

findProduct numbers
  =
  foldr (*) 1 numbers

printFactorials (x:xs)
  =
  let
    factorial = findFactorial x
  in
    do
      putStrLn("> " ++ show(x) ++
"! = " ++ show(factorial))
      printFactorials xs

printFactorials _
  =
  return ()

findFactorial 1
  = 1

findFactorial n
  = n * findFactorial(n-1)
```

ANDMETÜÜBI LOOMINE

```
data Quadruple a b
  = Quadruple a a b b

qFst (Quadruple x y z w)
  = [x,y]

qSnd (Quadruple x y z w)
  = [z,w]

test q
  =
  let
    fst = qFst(q)
    snd = qSnd(q)
  in
    putStrLn ("fst: " ++ show(fst)
++ "; snd: " ++ show(snd))
```

2 LISTIST TEINE ELEMENT KUI EKSISTEERIB JA ERINEV

```
teised list1@ (x:xs) list2@ (y:ys)
  =
  if length(list1) >= 2
  then
    if length(list2) >= 2
    then
      if head(xs) /= head(ys)
      then
        [head(xs),head(ys)]
      else
        [head(xs)]
    else
      [head(xs)]
  else
    if length(list2) >= 2
    then
      [head(ys)]
    else
      []

teised _ list2@ (y:ys)
  =
  if length(list2) >= 2
  then
    [head(ys)]
  else
    []

teised list1@ (x:xs) _
  =
  if length(list1) >= 2
  then
    [head(xs)]
  else
    []

teised _ _
  =
  []
```

TÄHTEDE LUGEMINE KUNI TÜHIKUNI, PIKKUS (IO)

```
import IO
import Char

kuniSpace
  :: IO ()

kuniSpace
  =
    do
      wl <- readWordLength
      putStrLn("Length: " ++
show(wl))

readWordLength
  =
    do
      if char /= ' '
      then
        do
          rest <- readWordLength
          return (1 + rest)
      else
        return 0
```

LISTI LÖPP PEALE KAHT SAMA JÄRJESTIKKU VÄÄRTUST

```
alatesSarnasusest (x:xs)
  | null xs
  = []
  | otherwise
  =
    if x == head(xs)
    then
      tail(xs)
    else
      alatesSarnasusest xs

alatesSarnasusest _
  = []
```

RAKENDADA IGALE EL JRK ARV KORDA FUNKTSIOONI

```
mapRohkem funk list
  = leiaMapRohkem funk list 0

leiaMapRohkem funk list@ (x:xs)
indeks
  = [rakenda funk x indeks] ++
(leiaMapRohkem funk xs (indeks + 1))

leiaMapRohkem funk _ _
  =
    []

rakenda funk elem korda
  | korda > 0
  = rakenda funk (funk elem)
(korda-1)
  | otherwise
  = elem
```

KOLMENDPUU, VASAKU JA PAREMA HARU EL ARV

```
data Kolmend a
  = Tühi
  | Tipp a (Kolmend a) (Kolmend a)
  (Kolmend a)

kolmendUuri (Tipp _ vasak keskmine
parem)
  = 1 + kolmendUuri vasak +
kolmendUuri parem

kolmendUuri _
  = 0
```

SPLIT LIST KAHEKS PAARIS PAARITUD

```
-- split2 [1..10]
-- > ([1,3,5,7,9],[2,4,6,8,10])
split2 list
  = (split list 0, split list 1)

split (x:xs) index
  | index `mod` 2 == 0
  = [x] ++ split xs (index + 1)
  | otherwise
  = split xs (index + 1)

split _ _
  = []
```

DUPLITSEERIDA LISTI ELEMENTI EL VÄÄRTUS KORDI

```
-- duplicate [3,0,2,4]
-- > [3,3,3,2,2,4,4,4,4]
duplicate (x:xs)
  = rep x x ++ duplicate xs

duplicate _
  = []

rep item times
  | times > 0
  = [item] ++ rep item (times - 1)
  | otherwise
  = []
```

QUICKSORT

```
qsort1 [] = []
qsort1 (p:xs) = qsort1 lesser ++ [p]
++ qsort1 greater
  where
    lesser = [ y | y <- xs, y < p ]
    greater = [ y | y <- xs, y >= p ]
```

FIBONACCI $F(n) = F(n-1) + F(n-2)$

```
f = 1 : 1 : zipWith ( + ) f (tail f)
```

BINARY SEARCH TREE

```
data Tree a = Tip | Node a (Tree a)
              (Tree a)

-- Returns a leaf node, makes writing
-- leafs easier
leaf x = Node x Tip Tip

-- Example tree
t1 = Node 17 (Node 12 (Node 5 Tip
                      (leaf 8)) (leaf 15))
      (Node 115
        (Node 32 (leaf 30)
          (Node 46 Tip (leaf 57)))
        (leaf 163))

size Tip = 0
size (Node _ t1 tr) = 1 + size t1 +
                      size tr

treeToListOrd Tip = []
treeToListOrd (Node x xl xr) =
  treeToListOrd xl ++ x : treeToListOrd
  xr

farLft (Node x Tip _) = x
farLft (Node x xl _) = farLft xl

farRt (Node x _ Tip) = x
farRt (Node x _ xr) = farRt xr
```

ARV TÕEVÄÄRTUSEKS (CASE)

```
arvToevaartuseks
  = \ n
    -> case n of
      0 -> False
      1 -> True
      _ -> error "Viga!"
```

VÕTA LISTI LÕPUST

```
takeLopust n xs
  = reverse (take n (reverse xs))
```

PEA ESINEMISTE ARV LISTIS

```
peaEsinemisteArv as@ ~(x:xs)
  = let
      arv n (z : zs)
        = arv (if x == z then n +
1 else n) zs
      arv n _
        = n
    in
    arv 0 as
```

KUULAB SISENDIT KUNI MISKI KORDUB (IO)

```
kuulaKorduseni
  = let
      abi d xs
        = if not (elem d xs)
          then do
              c <- getChar
              abi c (xs ++ [d])
          else putStrLn ""
    in
    do
      c <- getChar
      abi c []
```

VÕTAB SÕNE ALGUSEST TÄHTI, MIS POLE VÄIKSED

```
ilmaVaikestetaAlgas xs
  = takeWhile (not . isLower) xs
```

EEMALDAB SÕNEST KÕIK TÜHIKUD

```
elimTaanded (x : xs)
  = (dropWhile (isSpace) x) :
    elimTaanded xs
```

EEMALDAB SÕNEST JÄRJEST KORDUVAD

```
elimJarjKorduvad (x : xs)
  = x : elimJarjKorduvad (dropWhile
    (==x) xs)
elimJarjKorduvad _
  = []
```

EELALDAB SÕNEST KÕIK KORDUVAD

```
elimKorduvad (x : xs)
  = if any (==x) xs
    then elimKorduvad xs
    else x : elimKorduvad xs
elimKorduvad _
  = []
```

LISTI ELEMENTIDE VAHEDE KORRUTIS

```
vahedeKorrutis xss@ (x : xs)
  = product (zipWith (-) xss xs) *
  vahedeKorrutis xs
vahedeKorrutis x
  = product x
```

FAILI LUGEMINE

```
import System.IO

lugemine
  = do
    putStr "Anna loetava faili nimi: "
    nimi <- getLine
    sisu <- readFile nimi
    putStrLn ("Faili " ++ nimi ++ "
sisu:")
    putStrLn ("--- FAILI ALGUS ---")
    putStrLn sisu
    putStrLn ("--- FAILI LÕPP ---")
```