**127**. Deﬁneerida muutuja arvTõeväärtuseks väärtuseks funktsioon, mis võtab argumendiks täisarvu n ja annab tulemuseks tõeväärtuse True, kui n = 1,ja tõeväärtuse False, kui n = 0. Kui n / ∈ {0, 1}, siis peab rakendamine lõppema eestikeelse veateatega.

import Data.Char

arvTõeväärtuseks

= \ n

-> case n of

0

-> False

1

-> True

\_

-> error "arvTõeväärtuseks: pole tõeväärtus"

**128.** Deﬁneerida muutuja esipaar väärtuseks funktsioon, mis võtab argumendiks listi ning, kui selles on vähemalt 2 elementi, annab väärtuseks paari kahest esimesest elemendist, vastasel korral lõpetab sobiva eestikeelse veateatega. Arvutuse maht ei tohi kasvada listi pikkuse kasvades.

esipaar

= \ ls

-> case ls of

(a:b:\_)

-> (a,b)

\_

-> error "esipaar: tegemist pole vähemalt kahe elemendilise listiga"

**129**. Deﬁneerida muutuja esiAbs väärtuseks funktsioon, mis võtab argumendiks arvude listi: mittetühja listi korral annab väärtuseks listi, mille saab argumentlistist esimese elemendi asendamisel tema absoluutväärtusega; muul juhul tühja listi.

esiAbs

= \ ls

-> case ls of

(a:\_)

-> (abs a):(drop 1 ls)

\_

-> []

**137.** Deﬁneerida uuel viisil muutuja erinevus väärtuseks ülesandes 116 kirjeldatud funktsioon.

erinevus a b

= abs (a-b)

a `erinevus` b

= abs (a-b)

**138.** Võimalikult lühikese ja võimalikult vähe muutujaid kasutava deklaratsiooniga kujul (34) deﬁneerida muutuja imelik väärtuseks ülesandes 119 kirjeldatud funktsioon.

imelik c@(a,b) = (c,(b,a))

**139**. Deﬁneerida operaator takeLõpust, mis töötab nagu take, aga elemente võetakse listi lõpust. Näiteks takeLõpust 2 [3, 2, 4] väärtustamine peab andma [2, 4].

takeLõpust n ls = drop (length ls - n) ls

**140.** Deﬁneerida muutuja karritErinevus väärtuseks ülesandes 116 kirjeldatud funktsiooni karritamise tulemus. kasutades deﬁneeritavat muutujat preﬁksselt.

takeLõpust n ls = drop (length ls - n) ls

**149.** Deﬁneerida muutuja esiNegSgn väärtuseks funktsioon, mis võtab argumendiks arvude listi: kui selle esimene element on negatiivne, siis annab väärtuseks listi, mille saab argumentlistist esimese elemendi asendamisel arvuga −1, kõigil ülejäänud juhtudel annab välja argumentlisti enda.

esiNegSgn lst@ ~(a:ls)

| not (null lst) && a < 0

= -1:ls

| otherwise

= lst

**152.** Deﬁneerida muutuja summanali väärtuseks funktsioon, mis võtab argumendiks arvupaari ja annab väärtuseks selle paari komponentide summa siinuse ja summa enda suhte. Sama summat ei tohi arvutada korduvalt. Kõik vajalikud oma abimuutujad deﬁneerida lokaalselt.

summanali (a,b)

= let

summa = a+b

in

sin summa / summa

**153**. Deﬁneerida muutuja pooleks väärtuseks funktsioon, mis võtab argumendiks täisarvu n ja annab välja paari kahest täisarvust, mille summa on n ja vahe on 0 või 1.

pooleks n

=let

a= div n 2

in

(a,n-a)

**158**. Lahendada ülesanne 28 komprehensioonsüntaksi abil.

import System.IO

list

= [2\*\*n|n <- [10,20,30]]

**159**. Kirjutada listikomprehensioonavaldis, mille väärtuseks on sõnede list, mille elemendid on kahekohalised kümnendarvud suuruse järjestuses.

stringList

= [ (x:y:[]) | x <- "123456789", y <- "0123456789"]

**167**. Modiﬁtseerida ülesande 160 lahendust lokaalse deklaratsiooni abil, et sama numbrite list poleks korduvalt kirjeldatud.

= [ (a:b:[]) | let list="123456789abcdef", a <- list, b <-'0':list ]

**170**. Kirjutada programm, mis küsib kasutajalt eesti keeles rida ja kui see tuleb, siis kirjutab ekraanile selle rea sümbolid vastupidises järjekorras.

= do

putStrLn "Kirjuta rida: "

cs <- getLine

print (reverse cs)

**173**. Kirjutada programm, mis küsib kasutajalt eesti keeles sümbolit ja kui see tuleb, kirjutab selle sümboli kohe tühikuga eraldatult küsimuse järele, järgmisele reale aga kirjutab, mitmenda tähega ladina tähestikus tegu on (kui tegu pole tähega ladina tähestikus, siis kirjutab seda).

= do

hSetBuffering stdin NoBuffering

hSetEcho stdout False

putStr "kirjuta t2hem2rk: "

c <- getChar

print c

**182**. Arvu n faktoriaal n! avaldub kahaneva faktoriaali kaudu seosega n! = (n)n. Seetõttu saab deﬁnitsiooniga (65) antud muutujat kahFac kasutades kodeerida faktoriaalifunktsiooni eﬁnitsiooniga

fac n

= kahFac n n.

Anda muutujale fac samaväärne rekursiivne deﬁnitsioon ilma muutujat kahFac kasutamata.

import Data.Char

fac n

= case compare n 0 of

GT

-> n \* fac (n-1)

EQ

-> 1

\_

-> error "fac: neg. argument"

**183**. Kirjutada muutujale kahFac koodiga (65) samaväärne deﬁnitsioon, kus rekursioon põhineks võrduse (64) viimase teguri eraldamisel.

kahFac

:: (Num a, Integral b)

=> a -> b -> a

kahFac x k

= case compare k 0 of

GT

-> (x-(fromIntegral k-1)) \* kahFac x (k - 1)

EQ

-> 1

\_

-> error "kahFac: neg. teine argument"

**188**. Deﬁneerida muutuja suurtähega väärtuseks funktsioon, mis võtab argumendiks sõnede listi ja annab välja suurtähega algavate sõnede arvu selles.

suurtähega (bs@ ~(x:\_):xs)

= let

järgmised

= suurtähega xs

in

if not (null bs) && isUpper x then 1+ järgmised else järgmised

suurtähega \_

=0

**196**. Deﬁneerida kahel viisil muutuja kõikVõrdsed väärtuseks predikaat, mis võtab argumendiks listi ja kontrollib, kas listi elemendid on kõik võrdsed. Kui pole, siis peab väärtuseks tulema False, vastasel korral kui list on lõplik, siis peab väärtuseks tulema True. Esimene deﬁnitsioon peab realiseerima

idee kontrollida järjestikuste elementide võrdust, teine aga idee kontrollida esimese elemendi võrdumist ülejäänutega.

koikvordsed (x : bs@ (y:\_))

= case compare x y of

EQ -> koikvordsed bs

\_ -> False

koikvordsed \_

= True

test x

= if x==0 then True else False

**211**. Deﬁneerida muutuja grammar väärtuseks karritatud funktsioon, mis võtab argumendiks täisarvu n ja sõne s ja annab tulemuseks protseduuri, mis kirjutab ekraanile n korda sõnet s koos reavahetusega.

import System.IO

grammar n s

= let

printToScr n s

= case compare n 0 of

GT

-> do

putStrLn s

printToScr (n-1) s

EQ

-> return ()

in

printToScr n s

**215**. Deﬁneerida muutuja pööra väärtuseks protseduur, mis tsüklis küsib eesti keeles kasutajalt sisendit, loeb klaviatuurilt rea ja toob rea lõppedes ekraanile selle rea sümbolid vastupidises järjekorras, kuni sisestatakse tühi rida.

poora

= do

putStrLn "Anna m6ni lause:"

s <- getLine

if s == "" then return () else do putStrLn (reverse s)

poora

**217**. Kirjutada programm koodiga (86) antud muutuja loendaRead testimiseks.loendaRead

:: (Integral a)

=> IO a

loendaRead

= do

rida <- getLine

if null rida

then return 0

else do

ridadeArv <- loendaRead

return (1+ridadeArv)

**224**. Anda muutujale tuia väärtuseks list elementidega 1, 2, 3, 2, 3, 4, 3, 4, 5, . . ..

muu

:: (Ord a)

=> [a] -> [a] -> [a]

tuia

= 1:2:3:muu [x+1 | x <- tuia]

**239**. Deﬁneerida muutuja peaEsinemisteArv väärtuseks funktsioon, mis argumentlisti järgi leiab arvu, mitu korda esineb listi pea selles listis.

import System.IO

peaEsinemisteArv (x:xs)

= let

esinemised p n (z : zs)

= esinemised p (if p == z then n+1 else n) zs

esinemised p n \_

=n

in

esinemised x 1 xs

peaEsinemisteArv \_

=0

**240**. Anda ülesandes 204 kirjutatud muutujale paarisNegArv samaväärne deﬁnitsioon loenduriga.

paarisNegArv xs

= let

negArvud even count (head:list)

= negArvud (if even then False else True) ( if even && head < 0 then count+1 else count) list

negArvud even count \_

= count

in

negArvud True 0 xs

**245**. Anda muutujale kuula01 väärtuseks protseduur, mis loeb standardsisendist sümboleid, kuni sealt tulevad järjest sümbolid 0 ja 1, ja lõpetab siis töö.

kuula01

= let

kuula d

= do

c <- getChar

if c == '1' && d == '0' then putStrLn "" else kuula c

in

do

hSetBuffering stdin NoBuffering

c <- getChar

kuula c

**246**. Deﬁneerida muutuja facs väärtuseks list, mille elementideks on järjest kõigi naturaalarvude faktoriaalid. Listi algusosa väljaarvutamine kulutagu aega võrdeliselt arvutatud osa pikkusega.

facs

= let

fac n 9

= []

fac n i

= n:fac (n\*(i+1)) (i+1)

in

fac 1 1

**300**. Kirjutada võimalikult lühike avaldis, mille väärtuseks on funktsioon, mille väärtus leitakse, võttes argumendist siinuse ja saadud tulemusest koosinuse.Testida interaktiivses keskkonnas.

import Data.Char

yl300 x

= (cos . sin) x

**303**. Kasutades erisüntaksi asemel muutujat iterate, kirjutada avaldis, mille väärtuseks on mingi aritmeetiline jada.

yl303 x

=iterate (+3) x

**309**. Deﬁneerida muutuja ilmaVäikestetaAlgus väärtuseks funktsioon, mis

argumenttekstil annab välja tema pikima väiketähti mitte sisaldava lgusjupi.

ilmaV2ikestetaAlgus (xs)

= takeWhile isLower xs

**312**. Deﬁneerida muutuja elimTaanded väärtuseks funktsioon, mis võtab argumendiks sõnede listi ja annab tulemuseks nende sõnede algusest tühisümbolite jorude kaotamisel järele jäävate sõnede listi.

elimTaanded (xs:xss)

= dropWhile isSpace xs :elimTaanded(xss)

elimTaanded (\_)

= []

**318**. Kirjutada avaldis, mille väärtus on funktsioon, mis võtab argumendiks sõne ja annab tulemuseks tema iga tähe suureks teisendamisel saadava sõne.

yl318 xs

= map (toUpper x) xs

**324**. Kasutades operaatorit zipWith, deﬁneerida muutuja indeksid väärtu-

seks funktsioon, mis võtab argumendiks suvalise listi ja annab tulemuseks

listi, mis on niisama pikk kui argumentlist ja mille elemendid on järjestikused naturaalarvud alates 0-st.

indeksid (xs)

= zipWith (+) (tail xs)

**343**. Kirjutada programm kujul (159), mis kuulab standardsisendit ja kajab standardväljundisse kõik tähed, kuid mitte muid sümboleid. Programm lõpetagu töö, kui vajutatakse escape-klahvi.

import Data.Char

import System.IO

import System.IO.Error

main

= do

hSetBuffering stdin NoBuffering

hSetBuffering stdout NoBuffering

hSetEcho stdout False

interact ainultt2hed

hSetEcho stdout True

ainultt2hed

:: String -> String

ainultt2hed s

= filter isLetter (takeWhile (/='\ESC') s)

**349**. Kirjutada programm, mis küsib käivitudes kasutajalt faili nime ja kirjutab saadud nimega faili sisu ekraanile. Kui faili lugemine ebaõnnestub, siis teatab veast ja alustab sama tööd otsast peale.

pyynis1

:: IOError -> IO ()

pyynis1 e

= do

pyynis e

loeFail

pyynis

:: IOError -> IO ()

pyynis e

| isDoesNotExistError e

= putStrLn "Pole faili!"

| isPermissionError e

= putStrLn "pole faili lugemis]igust!"

| otherwise

= putStrLn "mingi jama."

loeFail

= do

hSetBuffering stdin NoBuffering

hSetBuffering stdout NoBuffering

loe `catch` pyynis1

hSetEcho stdout True

loe

= do

putStr "Sisesta faili nimi: "

s <- getLine

f <- readFile s

putStr f

**366**. Lahendada ülesanne 312 argumendivabas stiilis.

import Data.Char

import Data.List

elimTaanded

= map (dropWhile isSpace)

**378**. Kirjutada muutuja suurSumma deﬁnitsioon (105) samaväärselt ümber nii, et lokaalse operaatori deﬁnitsioonis oleks kasutatud argumendivaba stiili.

suurSumma n

| n >= 0

= let

summa i

| i <= n

= summa (i + 1) . ( (i ^ 4) +)

| otherwise

= id

in

summa 1 0

| otherwise

= error "suurSumma: neg. liidetavate arv"

**387**. Lahendada uuesti ülesanded 188, 189, 191, 192, 203, 314, kirjutades deﬁnitsioonid argumendivabas stiilis foldr abil.

--ylesanne 188

suurt2hega

= let

op (x:xs) j2rgmised

=if isUpper x then 1+ j2rgmised else j2rgmised

op (\_) j2rgmised

= j2rgmised

e = 0

in

foldr op e

**396**. Vaatame deﬁnitsiooni

concatLõpust

= reverse . concat.

Anda muutujale concatLõpust uus samaväärne võimalikult lühike deﬁnitsioon, mille korral arvutus vahestruktuure ei moodusta.

concatL2pust

= reverse . concat

concatL2pust2

= foldl (foldl (flip (:))) []

**418**. Viia tüüp Integer tsükliliste tüüpide klassi, deﬁneerides sobivalt kõik vajalikud meetodid.

class Tsükliline a where

nihe

:: Integer -> a -> a

vahe

:: a -> a -> Integer

posVahe

:: a -> a -> Integer

vahe

= posVahe

instance Tsükliline Integer where

nihe n i

= i+n

vahe i j

= i - j

posVahe i j

= abs (vahe i j)

**420**. Deklareerida tõeväärtusetüüp klassi Num esindajaks nii, et aritmeetilised operatsioonid vastaksid arvutustele modulo 2.

instance Num Bool where

(+)

= (/=)

negate

= id

(\*)

= (&&)

abs

= id

signum

= id

fromInteger

= (1 ==) . flip mod 2

**424.** Kirjutada oma moodulisse deklaratsioonid (226), (227), (228) ja testida interaktiivses keskkonnas meetodit kõik nii tõeväärtus- kui suurusvahekorratüübi jaoks.

class (Enum a, Bounded a) => Fin a where

kõik

:: [a]

kõik

= [minBound .. maxBound]

instance Fin Bool where

instance Fin Ordering where

instance Fin Char where

--välja kutsuda: <Main> kõik ::[Bool]

**429**. Kirjutada esindajadeklaratsioon, mis laiendab klassiNummeetodid elementhaaval listitüüpidele tingimusel, et elemenditüüp kuulub klassi Num. Anda muutujale fibs uue klassi meetodite kaudu uus deﬁnitsioon koodi (150) eeskujul.

--pooleli... vist viimane on kahtluse all

instance (Num a) => Num [a] where

(+)

= zipWith (+)

negate

= map negate

(\*)

= zipWith (\*)

abs

= map abs

signum

= map signum

fromInteger

= repeat . fromInteger

**289-294 lk õpikust**

data Kuu

= Jaanuar

| Veebruar

| Märts

| Aprill

| Mai

| Juuni

| Juuli

| August

| September

| Oktoober

| November

| Detsember

deriving (Enum,Show,Eq,Ord)

data Daatum

= Daatum Integer Kuu Int

deriving (Show, Eq, Ord)

päevi (y , m)

| elem jrk [1, 3, 5, 7, 8, 10, 12]

= 31

| elem jrk [4, 6, 9, 11]

= 30

| y `mod` 400 == 0

= 29

| y `mod` 100 == 0

= 28

| y `mod` 4 == 0

= 29

| otherwise

= 28

where

jrk

= fromEnum m + 1

**437**. Kirjutada deﬁnitsioon (235) oma moodulisse ja viia kuutüüp klassi Enum. Deﬁneerida muutuja tagurpidiAasta väärtuseks list, milles on parajasti kõik kalendrikuud aastas esinemisele vastupidises järjekorras.

tagurpidiAasta

= [Detsember,November ..]

**438**. Deﬁneerid muutuja eelmKuu väärtuseks funktsioon, mis leiab antud aasta antud kuule eelneva kuu. Tüüp peab olema sama mis koodiga (241) deﬁneeritud muutujal järgmKuu.

eelmKuu

:: (Integer , Kuu) -> (Integer , Kuu)

eelmKuu (y , m)

= case m of

Jaanuar

-> (pred y , Detsember)

\_

-> (y , pred m)

**442**. Deﬁneerida muutuja eelmPäev väärtuseks funktsioon, mis võtab argumendiks kuupäeva ja annab tulemuseks talle vahetult eelneva kuupäeva.

eelmPäev

:: Daatum -> Daatum

eelmPäev (Daatum y m d)

| 0 < d - 1

= Daatum y m (d - 1)

| otherwise

= let

(y' , m')

= eelmKuu (y , m)

in

Daatum y' m' (päevi (y', m'))

**443**. Deﬁneerida muutuja vaheAastates väärtuseks funktsioon, mis võtab argumendiks kuupäevad d1, d2 ja annab väärtuseks arvu, mitu täisaastat on kuupäeval d2 möödas kuupäevast d1.

vaheAastates (Daatum y1 m1 d1) (Daatum y2 m2 d2)

| m1 == m2

= if (d1>d2) then y2-y1-1 else y2-y1

| m1 > m2

= y2-y1-1

| otherwise

= y2-y1

**444**. Deﬁneerida muutuja päevaNr väärtuseks funktsioon, mis võtab argumendiks kuupäeva ja annab tulemuseks sellele kuupäevale eelnevate päevade arvu samas aastas.

päevaNr (Daatum y m d)

= let

päevadKuus y Jaanuar

= päevi (y,Jaanuar)

päevadKuus y m

=päevi (y , m) + päevadKuus y (snd (eelmKuu (y , m)))

in

if (m == Jaanuar) then d else d + päevadKuus y (snd (eelmKuu (y, m)))

**455**. Deﬁneerida muutuja võimalik väärtuseks predikaat, mis võtab argumendiks deklaratsiooniga (251) deﬁneeritud tüüpi andme ja kontrollib, kas tema esitatud piirang on täidetav.

data Eba d

= Teadmata

| Piirid d d

| Valik [d]

võimalik Teadmata

= True

võimalik (Piirid d d2)

= d<=d2

võimalik (Valik a)

= not (null a)