Abstraktsed andmestruktuurid

- Mitmed vaadatud andmestruktuurid on teostatud Idrises.
  - `Bool`, `Nat`, `()`, `Maybe a`, `List a`, `Either a b` jne.
  - Konstruktorid otse kasutatavad.
- Osad on teostatud madalamal tasemel, näiteks:
  - `Int`, `Integer`, `Char`, `Double`, `String`
  - Konstruktorid peidetud — tüübid abstraktsed!
- Vahetevahel on mõistlik luua ise abstraktseid andmestruktuure. Näiteks hulgad `Set a`
  - Ei soovi eksportida moodulist konstruktorite nimesid
  - ... kuid eksporditakse muid funktsioone:

```
empty     : Set a
null      : Set a → Bool
singleton : a → Set a
insert    : Ord a ⇒ a → Set a → Set a
delete    : Ord a ⇒ a → Set a → Set a
foldr     : (a → b → b) → b → Set a → b
...
```

Liidesed

Idrisis tuleb tihti kirjutada funktsioone, mis erinevad ainult natuke tüübi poolest:

```
equalChar   :   Char →   Char → Bool
equalInt    :    Int →    Int → Bool
equalString : String → String → Bool
```

Sellist koodi *ei saa* kirjutada ühe parameetrilise polümorfse funktsiooniga, kuna funktsioonide implementatsioon on erinev:

```
equal : a → a → Bool
equal = ?eq_v  -- pole def. mis töötaks iga tüübi korral
```

Selleks saame teha liidese (Haskellis tüübiklassi)

```
interface Equal a where
    (==) : a → a → Bool

Equal Int where
    a == b = ?eq_Int    -- Int'ide võrdsus
Equal Bool where
    a == b = ?eq_Bool   -- Bool'ide võrdsus
```

Tüübiklassid standardteegis

Idrise standardteegis on defineeritud tüübiklass `Eq`:

```
interface Eq a where
   (==), (≠) : a → a → Bool

    -- Minimaalne definitioon peab sisaldama:
    -- (==) või (≠)
   x ≠ y = not (x == y) -- vaikedefinitsioon
   x == y = not (x ≠ y) -- vaikedefinitsioon
```

Võrldusoperaatori tüüp on:

```
(==) : Eq a ⇒ a → a → Bool
```

s.t me saame operaatorit kasutada, kui tema argumentitüübil on defineeritud `Eq` instants!

Kõikidel polümorfsetel funktsioonidel, mis kasutavad võrdust peab tüübi *kontekstis* olema `Eq`:

```
lookup : Eq a ⇒ a → List (a, b) → Maybe b
```

Näide

```
data ValgusFoor = Punane | Kollane | Roheline

Eq ValgusFoor where
   Punane   == Punane   = True
   Kollane  == Kollane  = True
   Roheline == Roheline = True
   _        == _        = False

test : Bool
test = Kollane == Roheline   -- False
```

Range tüübiklass

Enumeratsioone kirjeldab järgnev tüübiklass:

```
interface Range a where
   rangeFromTo : a → a → List a              -- [x..y]
   rangeFromThenTo : a → a → a → List a      -- [x,y..z]

   rangeFrom : a → Stream a                  -- [x..]
   rangeFromThen : a → a → Stream a          -- [x,y..]
```

Sisend-väljund Idrises

- Idrise puhtad funktsioonid ei võimalda teha mittepuhtaid arvutusi.
  - Puhas — funktsiooni tulemus sõltub ainult argumentide väärtusest.
  - Ei saa teha näiteks juhuarvude funktsiooni `random : () → Int`
- Lahendus: `IO` liides
  - '`IO a`' tüüpi väärtus — „masin mis arvutab `a` tüüpi väärtuse“
  - `pure : a → IO a` — masin tagastab esimese argumendi väärtuse
  - `(>>=) : IO a → (a → IO b) → IO b` — masin käivitab esimese argumendi ja rakendab tulemuse teisele
  - ... lisaks baasfunktsioonid nagu `putStrLn : String → IO ()` ja `getLine : IO String`.
- Nii saab kombineerida olemasolevaid `IO` „masinaid“. Näiteks:

```
main : IO ()
main = randomRIO (1, 10) >>= classify >>= putStrLn
   where classify : Int → IO String
         classify x = if x `mod` 2 == 1 then
                          pure "paaritu"
                      else
                          pure "paaris"
```

do-süntaks I

Eelneval slaidil olnud koodi on keeruline lugeda ja kirjutada:

```
main : IO ()
main = randomRIO (1, 10) >>= classify >>= putStrLn
   where classify : Int → IO String
         classify x = if x `mod` 2 == 1 then
                          pure "paaritu"
                      else
                          pure "paaris"
```

Sama saab saavutada järgnevalt

```
main : IO ()
main = do
   r ← randomRIO (1, 10)
   c ← classify r
   putStrLn c
      where classify : Int → IO String
            classify x = ...
```

või

```
main : IO ()
main = do
   r ← randomRIO (1, 10)
   if r `mod` 2 == 1
    then putStrLn "paaritu"
    else putStrLn "paaris"
```

do-süntaks II

Näide

```
proc : IO ()
proc = do
   s ← getLine
   let n : Int
       n = read s
       n2 : Int
       n2 = 2*n
   putStrLn ("Kaks korda " ++ s ++ " on " ++ show n2)
```

Do-süntaks algab `do`-võtmesõnaga, millele järgnevad *järjest töödeldavad* laused.

- Laused mustriga $x \leftarrow p$, kus p : `IO` a siis $x : a$,
- `let` laused ning
- avaldised e, mille tüüp on `IO a`.

Mitme do kasutamine

- do seob kokku IO-avaldised, kuid ei saa vaadata konstruktsioonide sisse
- S.t. ühe avaldise jaoks pole do-d vaja
  - `main = putStrLn "Hello World!"`
- Hargenmise puhul võib olla vaja kasutada mitut do-d:

```
main : IO ()
main = do
   putStrLn "Kirjuta midagi!"
   xs ← getLine
   if (xs==="")
    then putStr "Sõnakuulmatu!"
    else do
       putStrLn "Tänan!"
       putStrLn ("Kirjutasid: " ++ xs)
```

do tähendus

- `a >>= f` on sama mis

```
do x ← a
   f x
```

- `a >> b` on sama mis

```
do a
   b
```

```
do a                    do do a                    do a
   b    on sama mis          b    on sama mis        do b
   c                       c                          c
```

- Fixity:

```
infixl 1 >>
infixl 1 >>=
```

Näide

```
main : IO ()
main = do
 putStrLn "Kirjuta␣midagi!"
 xs ← getLine
 if (xs==="")
   then putStr "Sõnakuulmatu!"
   else do
     putStrLn "Tänan!"
     putStrLn ("Kirjutasid:␣" ++ xs)
```

on sama mis

```
main : IO ()
main =
 putStrLn "Kirjuta␣midagi!" >>
 getLine  >>= (λ xs ⇒
 if (xs==="")
   then putStr "Sõnakuulmatu!"
   else
     putStrLn "Tänan!" >>
     putStrLn ("Kirjutasid:␣" ++ xs)
 )
```

Sisend-väljund

+ IO on hea näide, kuidas mittepuhtaid arvutusi saab modelleerida puhaste funktsioonide abil.
+ Selline modelleerimine on teoreetiliselt huvitav, kuna võimaldab katsetada erinevaid võimalusi ja kitsendusi. Näiteks erindid ja jätkud.

- Tekitab palju segadust, kui mõisted pole (veel) selged.
- Praktiline kasu pole ilmne: lihtsam kasutada mittepuhast programmeerimiskeelt.

Seetõttu: Selles kursuses harjutame IO-d aga ei lähe seda rada kaugemale.