Kõrvalefektid ja Haskell

Sissejuhatus

- Kõik senised programmid on olnud ilma kõrvalefektideta; so. puhtalt funktsionaalsed. Programmi täitmise ainsaks "efektiks" on tema väärtus.
- Osade ülesannete jaoks on kõrvalefektid vajalikud
  - "reaalse maailmaga" suhtlemine — sisend/väljund
  - efektiivsed imperatiivsed algoritmid
- Haskellis on "puhtad väärtused" eristatud "reaalsetest tegevustest" kahel viisil:
  - Kõrvalefektidega avaldised on spetsiaalset tüüpi
  - Erisüntaks efektide järjestamiseks ja täitmiseks

Kõrvalefektid ja Haskell

Käsud ja aktsioonid

- Haskellis on igal avaldisel mingi tüüp ja tema väärtsutamisel on tulemuseks sama tüüpi väärtus.
- Avaldist, mille väärtsutamisel lisaks "puhtale" väärtusele võib tekkida kõrvalefekt, nimetame käsuks (command).
- Käsk, mille "puhas väärtus" on tüüpi a, on ise abstraktset tüüpi $IO\ a$, mille väärtusi kutsume aktsioonideks.
- Aktsioone on võimalik täita süsteemi poolt, mille tulemusena toimub vastav kõrvalefekt.
- Terviklik Haskell-programm on aktsioon tüüpi $IO\ ()$

Eeldefineeritud IO operatsioonid

Standard-väljundisse kirjutamine

$$
\begin{aligned}
&\mathit{putChar} :: \mathit{Char} \to \mathit{IO}\ () \\
&\mathit{putStr} :: \mathit{String} \to \mathit{IO}\ () \\
&\mathit{print} :: \mathit{Show}\ a \Rightarrow a \to \mathit{IO}\ ()
\end{aligned}
$$

Standard-sisendist lugemine

$$
\begin{aligned}
&\mathit{getChar} :: \mathit{IO}\ \mathit{Char} \\
&\mathit{getLine} :: \mathit{IO}\ \mathit{String} \\
&\mathit{getContents} :: \mathit{IO}\ \mathit{String}
\end{aligned}
$$

Tekstifailide lugemine / kirjutamine

$$
\begin{aligned}
&\textbf{type}\ \mathit{FilePath} = \mathit{String} \\
&\mathit{readFile} :: \mathit{FilePath} \to \mathit{IO}\ \mathit{String} \\
&\mathit{writeFile} :: \mathit{FilePath} \to \mathit{String} \to \mathit{IO}\ () \\
&\mathit{appendFile} :: \mathit{FilePath} \to \mathit{String} \to \mathit{IO}\ ()
\end{aligned}
$$

IO teegid

module *Directory*

$$
\begin{aligned}
&\mathit{createDirectory} &&:: \mathit{FilePath} \to \mathit{IO}\ ()\\
&\mathit{removeDirectory} &&:: \mathit{FilePath} \to \mathit{IO}\ ()\\
&\mathit{removeFile} &&:: \mathit{FilePath} \to \mathit{IO}\ ()\\
&\mathit{renameDirectory} &&:: \mathit{FilePath} \to \mathit{FilePath} \to \mathit{IO}\ ()\\
&\mathit{renameFile} &&:: \mathit{FilePath} \to \mathit{FilePath} \to \mathit{IO}\ ()\\
&\mathit{getDirectoryContents} &&:: \mathit{FilePath} \to \mathit{IO}\ [\mathit{FilePath}]
\end{aligned}
$$

module *System*

$$
\begin{aligned}
&\mathit{getArgs} &&:: \mathit{IO}\ [\mathit{String}]\\
&\mathit{getProgName} &&:: \mathit{IO}\ \mathit{String}\\
&\mathit{getEnv} &&:: \mathit{String} \to \mathit{IO}\ \mathit{String}
\end{aligned}
$$

Käskude kombineerimine

IO monaadioperatsioonid

$$
\begin{aligned}
&\mathit{return} :: a \to \mathit{IO}\ a \\
&(\gg\!=) \quad :: \mathit{IO}\ a \to (a \to \mathit{IO}\ b) \to \mathit{IO}\ b \\
&(\gg) \quad :: \mathit{IO}\ a \to \mathit{IO}\ b \to \mathit{IO}\ b
\end{aligned}
$$

Näide

$$
\begin{aligned}
&\mathit{getWord} :: \mathit{IO}\ \mathit{String} \\
&\mathit{getWord} = \mathit{getChar} \gg\!= \lambda c \to \\
&\qquad\qquad \textbf{if}\ \mathit{isSpace}\ c \\
&\qquad\qquad\quad \textbf{then}\ \mathit{return}\ "" \\
&\qquad\qquad\quad \textbf{else}\ \ \mathit{getWord} \gg\!= \lambda w \to \\
&\qquad\qquad\qquad\qquad \mathit{return}\ (c : w)
\end{aligned}
$$

Käskude kombineerimine

do-süntaks

$$
\begin{aligned}
\mathit{expr} &= \mathbf{do}\,\{\mathit{stmt}; \ldots; \mathit{stmt}\} \\
\mathit{stmt} &= \mathit{pat} \leftarrow \mathit{expr} \\
&\mid \mathit{expr} \\
&\mid \mathbf{let}\ \mathit{decls}
\end{aligned}
$$

Transleerimisreeglid

$$
\begin{aligned}
\mathbf{do}\,\{e\} &= e \\
\mathbf{do}\,\{e; \mathit{stmts}\} &= e \gg \mathbf{do}\,\{\mathit{stmts}\} \\
\mathbf{do}\,\{v \leftarrow e; \mathit{stmts}\} &= e \gg\!= \lambda v \rightarrow \mathbf{do}\,\{\mathit{stmts}\} \\
\mathbf{do}\,\{\mathbf{let}\ \mathit{decls}; \mathit{stmts}\} &= \mathbf{let}\ \mathit{decls}\ \mathbf{in}\ \mathbf{do}\,\{\mathit{stmts}\}
\end{aligned}
$$

Käskude kombineerimine

Näide

```
getWord :: IO String
getWord = do c ← getChar
             if isSpace c
                then return ""
                else  do w ← getWord
                          return (c : w)
```

Tekstifailide töötlemine

Kopeerida tekst standardsisendist standardväljundisse

```
#!/usr/bin/runhugs
module Main (main) where
main :: IO ()
main = do input ← getContents
          putStr input
```

Näide

```
$ echo 'Lühike tekst' | cat1
Lühike tekst
$
```

Tekstifailide töötlemine

Kopeerida tekstifailid standardväljundisse

```
import System (getArgs)
main :: IO ()
main = do args ← getArgs
          if null args
             then catFiles ["-"]
             else  catFiles args
```

Tekstifailide töötlemine

Kopeerida tekstifailid standardväljundisse (järg)

$$
\begin{aligned}
&catFiles :: [String] \to IO\ () \\
&catFiles\ [\,] &&= return\ () \\
&catFiles\ (\text{"-"} : xs) &&= \mathbf{do}\ input \leftarrow getContents \\
&&&\quad\ putStr\ input \\
&&&\quad\ catFiles\ xs \\
&catFiles\ (x : xs) &&= \mathbf{do}\ contents \leftarrow readFile\ x \\
&&&\quad\ putStr\ contents \\
&&&\quad\ catFiles\ xs
\end{aligned}
$$

IO vigade töötlus

Veatöötlemise käsud

$$
\begin{aligned}
&\mathit{ioError} &&:: \mathit{IOError} \to \mathit{IO}\ a \\
&\mathit{userError} &&:: \mathit{String} \to \mathit{IOError} \\
&\mathit{catch} &&:: \mathit{IO}\ a \to (\mathit{IOError} \to \mathit{IO}\ a) \to \mathit{IO}\ a
\end{aligned}
$$

Veatöötlusega cat

```
catFiles (x : xs)
  = do cont <- catch (readFile x)
                     (\_ -> return (msg ++ x ++ "\n"))
       putStr cont
       catFiles xs
    where msg = "ERROR reading file: "
```

Tekstifailide töötlemine

Mitterekursiivne cat

```
catFiles :: [String] -> IO ()
catFiles xs = sequence_ [catFile x | x <- xs]

catFile  :: String -> IO ()
catFile "-" = do input <- getContents
                 putStr input
catFile x   = do cont <- catch (readFile x)
                               (\_ -> return (msg ++ x ++ "\n"))
                 putStr cont
  where msg = "ERROR reading file: "
```

Tekstifailide töötlemine

Unixi wc

```
wcFiles :: [String] → IO ()
wcFiles xs = sequence_ (map wcFile xs)
wcFile  :: String → IO ()
wcFile x    = do cont ← getFileContents
                 putStr (lwcCount x cont)
  where getFileContents
                 | x ≡ "-"    = getContents
                 | otherwise = readFile x
```

Tekstifailide töötlemine

Unixi wc (järg)

```
lwcCount :: String → String → String
lwcCount fname cont
    = format lc ++ format wc ++ format cc
                 ++ " " ++ fname ++ " \n"
  where ls = lines cont
        lc = length ls
        wc = sum (map (length ∘ words) ls)
        cc = length cont
        format x = rjustify 8 (show x)
```

”Hangman”

Sõna äraarvamine — ”hangman”

```
Main> main
Enter a word: ---
k ---
h h--
m h-m-
o h-m-
n h-n-m-n
a han-man
g hangman
Enter a word:
```

"Hangman" (versioon 1)

Puhverdused ja kajad

```
import System.IO

main = do
  hSetEcho stdin False
  hSetBuffering stdin  NoBuffering
  hSetBuffering stdout NoBuffering
  hangman
```

Mängu põhifunktsioon

```
hangman :: IO ()
hangman = do
  putStr "Enter a word: "
  word ← getWordEchoDashes
  game word []
```

"Hangman" (versioon 1)

Sõna lugemine

```
getWordEchoDashes :: IO String
getWordEchoDashes = do
  c ← getChar
  if c ≡ '\n'
    then do putChar '\n'
            return ""
    else  do putChar '-'
             w ← getWordEchoDashes
             return (c : w)
```

"Hangman" (versioon 1)

Sõna äraarvamine

```
game :: String → String → IO ()
game word guess = do
  c ← getChar
  let reveal = map (dash (c : guess)) word
  putStrLn ([c] ++ "  " ++ reveal)
  if elem '-' reveal
    then     game word (c : guess)
    else     hangman

dash :: String → Char → Char
dash guess w | elem w guess = w
             | otherwise    = '-'
```

”Hangman” (versioon 2)

ANSI-ekraani kontrollimine

```
cls :: String
cls = "\ESC[2J"

highlight :: String → String
highlight s = "\ESC[7m" ++ s ++ "\ESC[0m"

goto :: Int → Int → String
goto x y = "\ESC[" ++ show y ++ ";" ++ show x ++ "H"

home :: String
home = goto 1 1
```

"Hangman" (versioon 2)

ANSI-ekraani kontrollimine

$$
\begin{aligned}
&at :: (Int, Int) \to String \to String \\
&at\ (x, y)\ s = goto\ x\ y +\!\!+ s \\
&clearScreen :: IO\ () \\
&clearScreen = putStr\ cls \\
&writeAt :: (Int, Int) \to String \to IO\ () \\
&writeAt\ pos\ s = putStr\ (at\ pos\ s)
\end{aligned}
$$

"Hangman" (versioon 2)

Mängu põhifunktsioon

```
hangman :: IO ()
hangman = do
  clearScreen
  writeAt (1,1) "Enter a word: "
  word ← getWordEchoDashes
  game word []
```

Uue mängu alustamine

```
newgame = do
  putStr "Start a new game? (y/n)"
  c ← getChar
  if toUpper (c) ≡ 'Y'
    then hangman
    else return ()
```