Kõrvalefektid ja Haskell

- 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

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 ()`

Primitiivsed käsud

- "Tühikäsk":

```
return :: a -> IO a
```

- Standard-väljundisse kirjutamine:

```
putChar         :: Char -> IO ()
putStr          :: String -> IO ()
print           :: Show a => a -> IO ()
```

- Näide:

```
Prelude> putStr "Hello, World!"
Hello, World!
Prelude>
```

Eeldefineeritud IO operatsioonid

- Standard-sisendist lugemine:

```
getChar         :: IO Char
getLine         :: IO String
getContents     :: IO String
```

- Tekstifailide lugemine / kirjutamine:

```
type FilePath  = String

readFile        :: FilePath -> IO String
writeFile       :: FilePath -> String -> IO ()
appendFile      :: FilePath -> String -> IO ()
```

IO teegid

- module Directory

```
createDirectory  :: FilePath -> IO ()
removeDirectory  :: FilePath -> IO ()
removeFile       :: FilePath -> IO ()
renameDirectory  :: FilePath -> FilePath -> IO ()
renameFile       :: FilePath -> FilePath -> IO ()
getDirectoryContents :: FilePath -> IO [FilePath]
```

- module System

```
getArgs          :: IO [String]
getProgName      :: IO String
getEnv           :: String -> IO String
```

Käskude kombineerimine

- do-süntaks:

$$\begin{array}{lcl} \mathit{expr} & = & \texttt{do}\ \{\mathit{stmt};\ \ldots;\ \mathit{stmt}\} \\ \mathit{stmt} & = & \mathit{pat}\ \texttt{<-}\ \mathit{expr} \\ & | & \mathit{expr} \\ & | & \texttt{let}\ \mathit{decls} \end{array}$$

- 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)

```
catFiles          :: [String] -> IO ()
catFiles []        = return ()
catFiles ("-":xs) = do input <- getContents
                       putStr input
                       catFiles xs
catFiles (x:xs)   = do contents <- readFile x
                       putStr contents
                       catFiles xs
```

IO vigade töötlus

- Veatöötlemise käsud:

```
ioError   ::  IOError -> IO a
userError ::  String -> IOError
catch     ::  IO a -> (IOError -> IO a) -> IO a
```

- 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)
```