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 maailamga" 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:

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

- 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/local/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)
```


Lihtne graafika

- Graafiline “Hello, World”

```
import SOEGraphics
main
  = runGraphics (
    do w <- openWindow
              "My First Graphics Program" (300,300)
       drawInWindow w
           (text (100,200) "Hello Graphics World")
       k <- getKey w
       closeWindow w
    )
```

Lihtne graafika

- Graafika käske:

```
type Title = String
type Size  = (Int,Int)
type Point = (Int,Int)

runGraphics  :: IO () -> IO ()
openWindow   :: Title -> Size -> IO Window
drawInWindow :: Window -> Graphic -> IO ()
text         :: Point -> String -> Graphic
getKey       :: Window -> IO Char
closeWindow  :: Window -> IO ()
```


Lihtne graafika

- Graafiliste objektide joonistamise käske:

```
ellipse      :: Point -> Point -> Graphic
line         :: Point -> Point -> Graphic
polyline     :: [Point] -> Graphic
polygon      :: [Point] -> Graphic
```

- Graafiliste objektide “värvimine”:

```
data Color = Black | Blue | Green | Cyan
           | Red | Magenta | Yellow | White

withColor :: Color -> Graphic -> Graphic
```

Lihtne graafika

- Näide:

```
pic1 = withColor Red (ellipse (150,150) (300,200))
pic2 = withColor Blue
         (polyline [(100,50),(200,50),
                    (200,250),(100,250),(100,50)])
main2 = runGraphics $
    do w <- openWindow
              "Some Graphics Figures" (300,300)
       drawInWindow w pic1
       drawInWindow w pic2
       spaceClose w
```