Vstup a výstup

Ing. Lumír Návrat
katedra informatiky, A-1018
☎ 59 732 3252

Motivace

- Načtení čísla

```
val :: Int
val = 42

function :: Int -> Int
function = val + n

inputInt :: Int

inputDiff = inputInt - inputInt
```

Motivace

- Komunikace s okolním světem
  - printAString :: RealWorld -> String -> RealWorld
  - readAString :: RealWorld -> (RealWorld, String)

```
main rW =
    let rW' = printAString rW "Please enter your    name: "
        (rW'', name) = readAString rW'
    in printAString rW' ("Hello, " ++ name ++ ", how are you?")
```

Problém!!!

Minimální komunikace ☺

- Zobrazení řetězce na obrazovku
- Načtení řetězce z klávesnice
- Zápis dat do souboru
- Načtení dat ze souboru

- => v Haskellu modul IO.hs

Akce ve funkcionálním programu

- Imperativní jazyky
  - program tvořen posloupností *akcí*
  - čtení a nastavování globálních proměnných
  - čtení a zápis souborů
- Haskell
  - oddělení akcí od čistě funkcionálního kódu
  - *monadické operátory*
  - akce je funkce, jejíž výsledek je typu **(IO a)**

Příklady akcí

- Čtení a výpis znaku
  - `getChar :: IO Char`
    `putChar :: Char -> IO ()`
- Převod hodnoty na akci
  - `return :: a -> IO a`
- Test odpovědi y/n – posloupnost akcí
  - ```
    ready :: IO Bool
    ready = do c <- getChar
               return (c == 'y')
    ```


do Notace

- Flexibilní mechanismus podporující
  - Sekvenční přístup k I/O programům
  - Zachycování hodnot vrácených z IO akcí a předání do akcí následujících v programu
- Základem je operace (>>=)

  (>>=) :: IO a -> (a-> IO b) -> IO b

Cyklus while

- Požadavek opakování IO operací

```
while :: IO Bool -> IO () -> IO ()

while test action
   = do res <- test
        if res then do action
                       while test action
               else return ()
```
```

Funkce main

- Představuje hlavní program
- Akce, která nic nevrací:
  - ```
    main :: IO ()
    main = do c <- getChar
              putChar c
    ```

1. Přečte znak a uloží do proměnné c
2. vypíše znak c
3. Vrátí výsledek poslední akce – IO ()

Čtení řádku textu

1. Přečteme první znak.
2. Je-li to konec řádku, vrátíme prázdný řetězec.
3. Jinak přečteme zbytek řádku, spojíme s prvním znakem a vrátíme

```
getLine :: IO String
getLine = do x <- getChar
             if x==‘\n’ then return “”
             else do xs <- getLine
                     return (x:xs)
```


Výpis řetězce

- Na všechny znaky řetězce zavoláme funkci putChar, např.
  - map putChar xs
- Výsledkem je seznam akcí
  - ```
    map :: (a -> b) -> [a] -> [b]
    putChar :: Char -> IO ()
    map putChar s :: [IO ()]
    ```
- Převod na jedinou akci
  - ```
    sequence :: [IO()] -> IO ()
    putStr :: String -> IO ()
    putStr s = sequence (map putChar s)
    ```


Zpracování výjimek

- Výjimky jsou instance abstraktní třídy `IOError`
- Ke každé výjimce XXX existuje funkce `isXXX :: IOError -> Bool`
  - isEOFError
  - isDoesNotExistError
    ```

Zpracování výjimek

- Generování výjimky – funkce fail
  `fail :: IOError -> IO a`
  - typ výsledku se přizpůsobí kontextu
- Zachycení výjimky - funkce catch:
  `catch :: IO a -> (IOError -> IO a) -> IO a`
  1. prováděná akce
  2. obsluha výjimky – zavolá se, pokud nastane výjimka; vrátí náhradní výsledek
  3. výsledkem je akce z bodu 1 nebo 2

Čtení znaku + výjimky

- S ignorováním všech výjimek

```
getChar' = getChar `catch`
             ( \ _ -> return '\n' )
```

  - nastane-li chyba (např. konec souboru), vrátí znak konce řádku
  - nerozliší konec souboru od ostatních chyb
  - použití `catch` jako infixového operátoru

Čtení znaku + výjimky

- S rozlišením konce souboru

```
getChar' = getChar `catch` eofHandler
           where eofHandler e =
                 if isEOFError e
                    then return '\n'
                    else fail e
```

  - Pokud se dostaneme na konec souboru, vracíme znak konce řádku
  - Ostatní výjimky se předávají dále

Otevření a uzavření souboru

```
type FilePath = String
data IOMode = ReadMode   | WriteMode
            | AppendMode | ReadWriteMode
data Handle

openFile :: FilePath -> IOMode -> IO Handle
hClose   :: Handle -> IO ()
```

Čtení ze souboru

```
stdin, stdout, stderr :: Handle

hGetChar :: Handle -> IO Char
getChar = hGetChar stdin
```

- funkce začínající na ‘h’ dostávají jako první parametr referenci na otevřený soubor, ostatní pracují se standardními soubory

```
hGetContents :: Handle -> String
```

- přečte celý obsah souboru
- líné vyhodnocení – vstup se požaduje až při požadavku na konkrétní hodnotu

Příklad

```
import IO
main = do hin  <- opf “From: “ ReadMode
          hout <- opf “To: “   WriteMode
          contents <- hGetContents hin
          hPutStr hout contents
          hClose hout
          putStr “Done.”
opf :: String -> IOMode -> IO Handle
opf prompt mode = do putStr prompt
                     name <- getLine
                     openFile name mode
```


Příklad

- Funkce opf se zeptá na jméno souboru a otevře ho v zadaném režimu
- Pokud se otevření nepodaří, vznikne výjimka

```
opf prompt mode =
  do putStr prompt
     name <- getLine
     catch (openFile name mode)
       (\_ -> do putStr (“Open error\n”)
                 opf prompt mode)
```


Úkol pro cvičení

Vytvořte program, který opíše soubor na standardní výstup s očíslovanými řádky.

1) Převod textu na seznam řádků
   `text2lines :: String -> [String]`
2) Očíslování řádků
   `numbering  :: [String] -> [String]`
3) Převod seznamu řádků na text
   `lines2text :: [String] -> String`

```
(lines2text . numbering . text2lines) contents
```