

Cvičení 8

8.1 Úvod do IO

Příklad 8.1.1 Do interpretru zapište nejprve výraz `"**\n**\n"` a poté výraz `putStr "**\n**\n"`. Rozdíl chování interpretru vysvětlete.

Příklad 8.1.2 Uvažme seznam definovaný následovně:

```
pt :: [[Integer]]
pt = iterate (\r -> zipWith (+) ([0] ++ r) (r ++ [0])) [1]
```

- Vypište jeho prvních 15 prvků.
- Co seznam vyjadřuje?
- Vyhodnoťte `take 7 pt`.
- Vyhodnoťte `show (take 7 pt)`.
- Vyhodnoťte `map show (take 7 pt)`.
- Vyhodnoťte `(unlines . map show) (take 7 pt)`.
- Vyhodnoťte `(putStr . unlines . map show) (take 7 pt)`.

Příklad 8.1.3 Stáhněte si program `08_pt.hs`, spusťte ho a pochopte, jak funguje. Soubor naleznete v ISu a v příloze sbírky.

8.2 IO pomocí do-notation

Příklad 8.2.1 Naprogramujte funkci `compareInputs :: IO ()`, která od uživatele načte dva řetězce a pak vypíše na obrazovku delší z nich.

Příklad 8.2.2 Definujte akci `getInt :: IO Int`, která ze standardního vstupu načte celé číslo. Využijte knihovní funkci `read :: (Read a) => String -> a`.

Příklad 8.2.3 Upravte a doplňte následující zdrojový kód tak, aby program vyžadoval a načel postupně tři celá čísla a o nich určoval, zda mohou být délkami hran trojúhelníku.

```
main :: IO ()
main = do putStrLn "Enter one number:"
         x <- getInt
         putStrLn (show (1 + x))
```

Příklad 8.2.4 Napište program, který vyzve uživatele, aby zadal jméno souboru, poté ověří, že zadaný soubor existuje, a pokud ano, vypíše jeho obsah na obrazovku, pokud ne, informuje o tom uživatele. Úkol řešte s využitím `doesFileExist` z modulu `System.Directory`

Příklad 8.2.5 Vysvětlete význam a rizika rekurzivního použití volání funkce `main` v následujícím programu.

```
main :: IO ()
main = do putStr "Enter string: "
         s <- getLine
         if null s then putStrLn "pa pa"
           else do putStrLn (reverse s)
                 main
```

8.3 IO pomocí operátorů `>=` a `>>`

Příklad 8.3.1 Převeďte následující program v `do`-notaci na notaci s použitím `>=`.

```
main = do
  f <- getLine
  s <- getLine
  appendFile f (s ++ "\n")
```

Příklad 8.3.2 Napište program, který načte jeden řádek textu od uživatele, ze kterého pak odstraní všechny znaky, které nejsou znaky abecedy. Výsledek následně vypíše na výstup. V úkolu použijte funkci `isAlpha` z modulu `Data.Char`.

Příklad 8.3.3 Které z následujících výrazů jsou korektní? Co tyto výrazy dělají?

- `getLine (>=) putStrLn . tail`
- `(>>) getLine getLine`
- `getLine >= \s -> return >> putStrLn "ok"`
- `getLine >> getLine >= (\a b -> putStrLn (a ++ b))`
- `readFile "/etc/passwd" >= (\s -> writeFile "backup") >> putStrLn s`
- `getLine >= \f -> putStrLn "N/A"`
- `getLine >> _ -> return 1`
- `x <- getChar >> putStrLn x >> putStr "done"`

Příklad 8.3.4 Následující funkci přepište do tvaru, ve kterém nepoužijete konstrukci `do`, také určete typ funkce.

```
query question = do putStrLn question
                   answer <- getLine
                   return (answer == "ano")
```

Příklad 8.3.5 Funkci `query` z předchozího příkladu modifikujte tak, aby:

- a) Rozlišovala kladné i záporné odpovědi a při nekorektní nebo nerozpoznané odpovědi otázku opakovala.
- b) Akceptovala odpovědi s malými i velkými písmeny, interpunkcí, případně ve více jazycích.

Příklad 8.3.6 Upravte program `08_guess.hs` tak, aby parametry funkce `guess` četl z příkazové řádky. Soubor naleznete v ISu a v příloze sbírky.

Příklad 8.3.7 Vymyslete a naprogramujte několik triviálních programků manipulujících s textovými soubory:

- počítání řádků
- výpis konkrétního řádku podle zadaného indexu
- vypsání obsahu pozpátku
- seřazení řádků, ...

Definice alternativně přepište s a bez pomoci syntaktické konstrukce `do`.

Příklad 8.3.8 Definujte funkci `(>>)` pomocí funkce `(>>=)`.

Řešení

Řešení 8.1.1 V prvním případě dojde jenom k vypsání hodnoty výrazu `"**\n**\n"` jako řetězce. Jelikož jde o výraz vytvořený jenom z datových konstruktorů (seznam znaků), nelze jej dále vyhodnotit.

Na druhou stranu, u výrazu `putStr "**\n**\n"` jde o výraz typu `IO ()`. To znamená, že výsledkem, na který se výraz vyhodnotí, bude nultice obalená monádou. Avšak, jako vedlejší efekt vyhodnocení akce `putStr` dojde k vypsání její argumentu na standardní výstup.

Poznámka: Srovnajte výsledek vyhodnocení následovných akcí v interpretu ghci:

```
putStrLn "test"
putStr ""
return ()
return True
return 100
return [1,2,3]
return (return 1 :: IO Int)
return id
return (id, not)
```

Na základě výsledku vyhodnocení těchto výrazů můžeme vidět, že interpret nevypíše hodnotu, kterou vracíme v monádě v případě, že jde o hodnotu typu, který není instancí třídy `Show`, protože ji vypsat nelze a také pokud jde o hodnotu `()`, protože ta reprezentuje v Haskellu hodnotu typu známého z jazyka C jako `void` a tudíž nenese žádnou užitečnou informaci. Navíc u hodnoty typu `IO ()` není výsledek vypisován patrně i proto, že by byl rušivý při výpisu hodnot pomocí funkcí `putStr`, `putStrLn`, `print`, ...

Řešení 8.2.1

```
compareInputs :: IO ()
compareInputs = do
    putStr "First string: "
    str1 <- getLine
    putStr "Second string: "
    str2 <- getLine
    let longer = if length str1 > length str2 then str1 else str2
    putStrLn $ "Longer string: " ++ longer
```

Řešení 8.2.2

```
getInt :: IO Int
getInt = getLine >>= \num -> return (read num :: Int)

getIntDo :: IO Int
getIntDo = do
    line <- getLine
    let num = read line :: Int
    return num
```

Řešení 8.2.3

```
trg :: IO ()
trg = do
  putStrLn "Enter three numbers separated by commas: "
  nums <- getLine
  let (a, b, c) = read "(" ++ nums ++ ")" :: (Int, Int, Int)
  print (a + b > c && b + c > a && c + a > b)
```

Řešení 8.2.4

```
import System.Directory

cat :: IO ()
cat = do
  putStr "Enter filename: "
  fileName <- getLine
  fileExists <- doesFileExist fileName
  if fileExists then do
    fileContents <- readFile fileName
    putStr fileContents
  else do
    putStrLn "Requested file was not found."
    return ()
```

Řešení 8.2.5 Obecně, rekurze v kontextu IO umožňuje opakované vykonávání akcí. V tomto případě vidíme, že od první akce, která je součástí main až po její další výskyt, se opakuje načtení řetězce a výpis jeho převrácené podoby. Tedy rekurze umožňuje kontrolovaně (zadáme-li prázdný řetězec – dojde k ukončení rekurze) opakovat akci. Pokud však neuhlídáme ukončení rekurze, může dojít k zacyklení vyhodnocování akcí.

Řešení 8.3.1

```
main = getLine >>= \f ->
  getLine >>= \s ->
  appendFile f (s ++ "\n")
```

Řešení 8.3.2

```
a)
import Data.Char
main :: IO ()
main = getLine >>= putStr . filter isAlpha

main' :: IO ()
main' = do x <- getLine
  putStr (filter isAlpha x)
```

Řešení 8.3.3

- a) Nekorektní, operátor je zapsaný v prefixovém tvaru, ale použitý infixově.
- b) Korektní.
- c) Nekorektní, funkci `return` chybí argument.
- d) Nekorektní, není možné „sesbírat“ hodnoty `getLine` takovým způsobem (použití `>>` za prvním `getLine` způsobí „zapomenutí“ jeho hodnoty).
- e) Nekorektní, platnost `s` končí závorkou, tedy při použití na konci řádku již není definována.
- f) Korektní, proměnnou `f` není nutné použít.
- g) Nekorektní, za operátorem `>>` musí následovat výraz typu `IO a`, ne funkce typu `a -> IO b`.
- h) Nekorektní, zápis `<-` je možno použít jenom v `do`-notaci (a intensionálních seznamech).

Řešení 8.3.4

```
query' :: String -> IO Bool
query' question =
    putStrLn question >> getLine >>= \answer -> return (answer == "ano")
```

Nebo lze upravit vzniklou λ -abstrakci na pointfree tvar:

```
query'' :: String -> IO Bool
query'' question =
    putStrLn question >> getLine >>= return . (=="ano")
```

Řešení 8.3.5

a)

```
query2 :: String -> IO Bool
query2 question = do
    putStrLn question
    answer <- getLine
    if answer == "ano"
        then return True
        else if answer == "ne"
            return False
            else query2 question
```

b)

```
import Data.Char

query3 :: String -> IO Bool
query3 question = do
    putStrLn question
    answer <- getLine
    let lcAnswer = map toLower answer
    if lcAnswer `elem` ["ano", "áno", "yes"] then
        return True
    else
        if lcAnswer `elem` ["ne", "nie", "no"] then
            return False
```

```
else  
    query3 question
```

Řešení 8.3.8

```
x >> f = x >>= \_ -> f
```