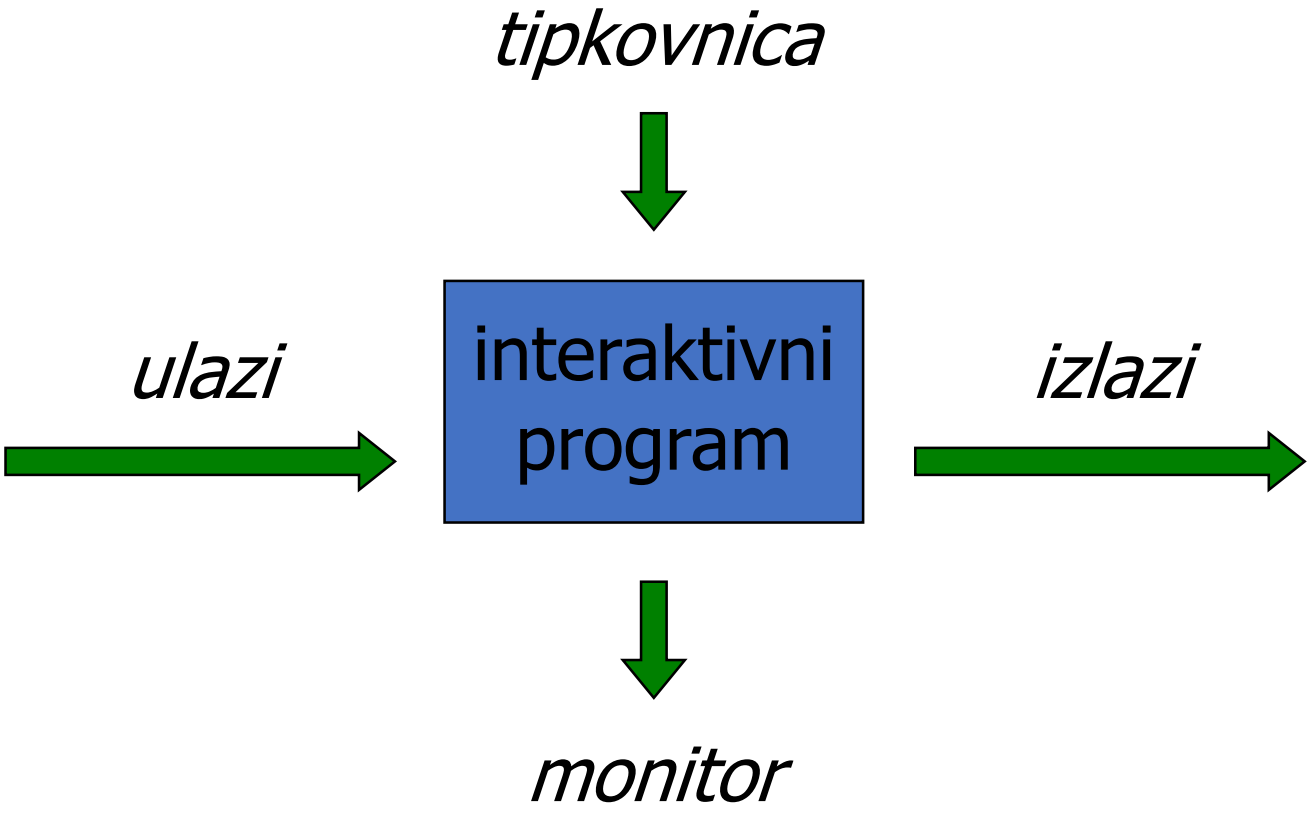
FUNKCIJSKO PROGRAMIRANJE

Interaktivno programiranje

Uvod

Do sada smo se uglavnom bavili tzv. **batch** programima koji uzimaju ulaz i vraćaju izlaz na kraju.

Sada bismo željeli pisati interaktivne programe koji tijekom svog izvršavanja čitaju sa standardnog ulaza (tipkovnica) i ispisuju na standardni izlaz (monitor).



Problem

Haskell programi su matematičke funkcije:

- Haskell programi nemaju „side” efekata.

Čitanje s tipkovnice i ispisivanje na monitor su „side efekti”:

- Interaktivni programi imaju „side” efekte.

Rješenje

Interaktivni programi mogu se pisati u Haskellu koristeći tipove koji razlikuju čiste izraze od onih koji sadrže akcije koje podrazumijevaju side efekte.

```
IO a
```

Tipovi akcija koje vraćaju vrijednost tipa a.

Primjerice:

```
IO Char
```

Tip akcija koje vraćaju znak.

```
IO ()
```

Tip akcije sa čistim side efektom koja nema tip koji vraća.

Primjedba:

- () je tip torke koja nema komponenti.

Osnovne akcije

Standardna biblioteka sadrži niz akcija uključujući i sljedeće tri:

- akcija getChar čita znak s tipkovnice, ispisuje ga na monitor i vraća taj znak kao rezultirajući znak:

```
getChar :: IO Char
```

- Akcija putChar c ispisuje znak na monitor, ali ne vraća rezultat:

```
putChar :: Char → IO ()
```

- Akcija return v vraća vrijednost v bez izvođenja bilo kakve interakcije:

```
return :: a → IO a
```

Sekvencioniranje

Niz akcija može se kombinirati u jednu složenu akciju koristeći ključnu riječ **do**.

Primjerice:

```
act :: IO (Char,Char)
act = do x ← getChar
         getChar
         y ← getChar
         return (x,y)
```

Izvedene osnovne akcije

- Čitanje znakovnog niza s tipkovnice:

```
getLine :: IO String
getLine = do x ← getChar
             if x == '\n' then
                return []
             else
                do xs ← getLine
                   return (x:xs)
```

- Ispisivanje znakovnog niza na monitor:

```
putStr :: String → IO ()
putStr []     = return ()
putStr (x:xs) = do putChar x
                   putStr xs
```

- Ispisivanje znakovnog niza i prijelaz u novi redak:

```
putStrLn :: String → IO ()
putStrLn xs = do putStr xs
                 putChar '\n'
```

Primjer

Definiranje akcije koja učitava znakovni niz s tipkovnice i ispisuje njegovu duljinu:

```
strlen :: IO ()
strlen = do putStr "Enter a string: "
            xs ← getLine
            putStr "The string has "
            putStr (show (length xs))
            putStrLn " characters"
```

Primjerice:

```
> strlen

Enter a string: Haskell
The string has 7 characters
```

Primjedba:

- Evaluiranje akcije izvršava njezine side efekte, dok se rezultirajuća vrijednost odbacuje.

Vješala

Razmotrimo sljedeću verziju popularne igre Vješala:

- Prvi igrač tajno upisuje riječ (frazu, ime, itd…)
- Drugi igrač pokušava pogoditi riječ putem niza pogađanja.
- Za svako pogađanje, program evidentira slova u tajnoj frazi koja su pogođena.

- Igra završava točnim pogađanjem tajne riječi/fraze.

Koristit ćemo top down pristup u implementiranju igre vješala u Haskellu počevši od sljedeće funckije:

```
hangman :: IO ()
hangman = do putStrLn "Think of a word: "
             word ← sgetLine
             putStrLn "Try to guess it:"
             play word
```

Akcija sgetLine čita liniju teksta s tipkovnice ispisujući umjesto svakog znaka crticu:

```
sgetLine :: IO String
sgetLine = do x ← getCh
              if x == '\n' then
                 do putChar x
                    return []
              else
                 do putChar '-'
                    xs ← sgetLine
                    return (x:xs)
```

Akcija getCh čita jedan znak s tipkovnice bez ispisivanja na monitor:

```
import System.IO

getCh :: IO Char
getCh = do hSetEcho stdin False
           x ← getChar
           hSetEcho stdin True
           return x
```

Funkcija play zahtijeva i procesuira pogađanja od početka do kraja igre.

```
play :: String → IO ()
play word =
   do putStr "? "
      guess ← getLine
      if guess == word then
         putStrLn "You got it!"
      else
         do putStrLn (match word guess)
            play word
```

Funkcija match vraća string u kojem se ispisuju oni znakovi iz prvog stringa koji se pojavljuju u drugom stringu:

```
match :: String → String → String
match xs ys =
   [if elem x ys then x else '-' | x ← xs]
```

Primjerice:

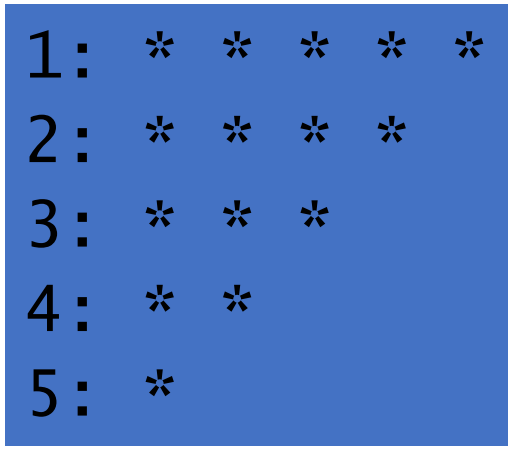
```
> match "haskell" "pascal"

"-as--ll"
```

Vježba

Implementirajte igricu nim u Haskellu koja ima sljedeća pravila:

- Ploča se sastoji od 5 redaka zvjezdica:



- Dva igrača redom povlače poteze u kojima uklanjaju jednu ili više zvjezdica skraja jednog od redaka.

- Pobjednik je onaj igrač koji ukloni posljednju zvijezdu/zvijezde s ploče.

Uputa:

Reprezentirajte ploču listom od 5 prirodnih brojeva koji predstavljaju broj preostalih zvjezdica u svakom od redaka. Primjerice, početna ploča je lista [5,4,3,2,1].