More Haskell I/O

Prof. Susan Older

18 November 2019

Recap: IO Actions

IO types:

For each Haskell type `t`, there is a type `IO t` whose values are:

I/O actions (or programs) that yield a result of type `t`.

When an I/O action is executed, it does two things:

1. It (possibly) performs some input/output or other side effects.
2. It produces/yields a result.

Built into Haskell:

```
putChar :: Char -> IO ()          getChar :: IO Char
 putStr :: String -> IO ()        getLine :: IO String
putStrLn :: String -> IO ()   getContents :: IO String
   print :: Show a => a -> IO ()      return :: a -> IO a
```


Recap: Sequencing Actions

The `>>` ("chain") and `>>=` ("bind") operators:

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

Both can be expressed using `do` notation:

```
sample :: IO String
sample = do putStrLn "Please enter a character: "
            ch <- getChar
            putStrLn "Thank you!"
            return (replicate 10 ch)
```


The `main` Program

The `main` program is an I/O action:

```
main :: IO ()
main = do putStr "Enter a string: "
          resp <- getLine
          putStrLn ("Your string was: " ++ resp)
          putStrLn "\n\nGoodbye!"
```

Two options for executing `main` (in file `io.hs`) outside of Ghci:

1. You can use `runhaskell` or `runghc`:

```
runhaskell io
```

2. Create an executable file and then run it:

```
ghc --make io
./io
```

Our Own Word-Count Program

Let's write a file `ourWC.hs`:

```
wordCount :: String -> (Int, Int, Int)
wordCount inp = (l,w,c)
  where
    l = length (lines inp)
    w = length (words inp)
    c = length inp

main :: IO ()
main = do inp <- getContents
          print (wordCount inp)
```

To try it out:

```
runhaskell ourWC < sampleFile
```


Some Built-In Control Structures

Two ways to sequence a series of actions:

```
sequence :: [IO a] -> IO [a]
sequence_ :: [IO a] -> IO ()
```

Compare the following:

```
sequence  (map print [1..10])
sequence_ (map print [1..10])

sequence  (replicate 5 getChar)
sequence_ (replicate 5 getChar)
```


Putting it All Together

Let's write a program that does the following:

1. Prompts the user to enter an integer (say, `n`)
2. Reads `n` characters from standard input
3. Returns the number of uppercase characters read

The power of `sequence`:

```
readN :: IO Int
readN = do putStr "How many characters will you enter? "
           n <- getInteger
           chars <- sequence [ getChar | x <- [1..n] ]
           return (length (filter isUpper chars))
```