

My Report

Me

Thursday 9th February, 2017

Abstract

We give a toy example of a report in *literate programming* style. The main advantage of this is that source code and documentation can be written and presented next to each other. We use the listings package to typeset Haskell source code nicely.

Contents

1	How to use this?	2
2	The most basic library	2
3	Wrapping it up in an executable	3
4	Simple Tests	3
5	Conclusion	4
	Bibliography	4

1 How to use this?

To generate the PDF, open the report.tex in your favorite LaTeX editor and hit compile, or manually do this:

```
pdflatex report
bibtex report
pdflatex report
pdflatex report
```

You should have stack installed (see <http://haskellstack.org/>) and open a terminal in the same folder.

To compile everything: `stack build`.

To open ghci and play with your code: `stack ghci`

To run the executable generated from Section 3: `stack exec myprogram`

To run the tests from Section 4: `stack clean; stack test -coverage`

2 The most basic library

This section describes a module which we will import later on.

```
module Basics where

import Control.Monad
import System.Random

thenumbers :: [Integer]
thenumbers = [1..]

somenumbers :: [Integer]
somenumbers = take 10 thenumbers

randomnumbers :: IO [Integer]
randomnumbers = replicateM 10 $ randomRIO (0,10)
```

We can interrupt the code anywhere we want.

```
funnyfunction :: Integer -> Integer
funnyfunction 0 = 42
```

Even in between cases, like here. It's always good to cite something [Knu11].

```
funnyfunction n | even n    = funnyfunction (n-1)
                | otherwise = n*100
```

Something to reverse lists.

```
myreverse :: [a] -> [a]
myreverse [] = []
myreverse (x:xs) = myreverse xs ++ [x]
```

That's it, for now.

3 Wrapping it up in an executable

We will now use the library from Section 2 in a program.

```
module Main where

import Basics

main :: IO ()
main = do
    putStrLn "Hello!"
    print somenumbers
    print (map funnyfunction somenumbers)
    myrandomnumbers <- randomnumbers
    print myrandomnumbers
    print (map funnyfunction myrandomnumbers)
    putStrLn "GoodBye"
```

The output of the program is something like this:

```
Hello!
[1,2,3,4,5,6,7,8,9,10]
[100,100,300,300,500,500,700,700,900,900]
[1,3,0,1,1,2,8,0,6,4]
[100,300,42,100,100,100,700,42,500,300]
GoodBye
```

Note that the above `showCode` block is only shown, but it gets ignored by the Haskell compiler.

4 Simple Tests

We now use QuickCheck

```
module Main where

import Basics

import Text.Printf (printf)
import Test.QuickCheck
import Test.QuickCheck.Test (isSuccess)
import System.Exit (exitSuccess, exitFailure)

tests :: [(String, IO Result)]
tests =
    [ ("funnytest1", quickCheckResult (\n -> funnyfunction n 'elem' [42, n*100, (n
        -1)*100]) )
```

```

    , ("reversetest", quickCheckResult (\str -> myreverse (myreverse str) == (str
      ::String))) )
  ]

main :: IO ()
main = do
  results <- mapM \(s,a) -> printf "\n%-20s: " s >> a) tests
  if all isSuccess results
  then exitSuccess
  else exitFailure

```

To run this, use `stack clean; stack test -coverage`. In particular this will generate a nice report using `hpc`. Look for “The coverage report for ... is available athtml” and open this file in your browser. See https://wiki.haskell.org/Haskell_program_coverage for more examples and information how to read the report.

5 Conclusion

Finally, we can see that [LW13] is a nice paper.

References

- [Knu11] Donald E. Knuth. *The Art of Computer Programming. Combinatorial Algorithms, Part 1*, volume 4A. Addison-Wesley Professional, 2011.
- [LW13] Fenrong Liu and Yanjing Wang. Reasoning about agent types and the hardest logic puzzle ever. *Minds and Machines*, 23(1):123–161, 2013.