Chapter 1 - Getting Started

September 3, 2017

Libraries used in this file:

> **import** *Data.List.Split*

Hello World!

```
helloWorld :: IO ()
helloWorld = putStrLn "Hello World"
```

Haskell is a **purely functional** language. Pure functions are those which always return the same result, no matter what the evaluation strategy is - so given the same input, they will always return the same output They take immutable values as input and output new values. They also have no side effects and don't rely on external mutable state.

Pure functions are easier to test, as they behave consistenly in any context and we can easily state properties of iys behaviour that should always be true and test them on random inputs.

Haskell uses **lazy evaluation**, which means that computations are deferred until their results are actually needed. This is especially useful for processing streams.

Haskell is a **statically typed** language. This means that functions and variables have their types assigned when they are compiled. This makes code run mcuh faster than that of dynamic interpreted languages. It also uses **type inference**, so if the user omits types from the code, it can tell what types to use. This means that you don't need to spend time specifying types, but keep the performance benefit of a statically typed language.

Haskell has libraries for many applications such as database access, XML processing, web programming, GUIs and text processing and has been used in diverse areas such as verifying vehicle systems and imvestment banking.

1 The Glasgow Haskell Compiler (GHC)

GHC is the most widely used Haskell compiler. Real World Haskell is based on version 6.8.2. The current version is 8.0.2 and there have been some significant changes which we will discover as we get further into the book, particularly when we study monads.

ghci is the interative interpreter of ghc and runghc can be used to run Haskell programs as scripts without first compiling them. It can be used for basic arithmetic, boolean logic and defining varibles and functions.

2 Functions used in the exercises

All of the functions used in the exercise can be found in the standard Prelude. This is where all the functions that do not have to be imported into a Haskell program by default live.

2.1 interact

We define *interact'* to show the type signature:

$$interact' :: (String \to String) \to IO\ ()$$
$$interact' = \bot$$

The interact function takes a function of type $String \to String$ as its argument. The standard input is the argument and the output is a string displayed in the standard output.

2.2 lines

We define *lines'* to show the type signature (note this is not the actual Haskell definition of the function):

$$lines' :: String \to [String]$$
$$lines'\ s = splitOn\ \texttt{"\textbackslash n"}\ s$$

The lines function breaks a string into a list of strings at the new line character.

2.3 words

We define *words'* to show the type signature (note this is not the actual Haskell definition of the function):

$words' :: String \to [String]$
$words'\ s = splitOneOf$ `" \n"` s

The words function breaks a string up into a list of words, which were delimited by white space.

2.4 length

We define *length'* to show the type signature (note this is not the actual Haskell definition of the function):

$length' :: [a] \to Int$
$length'\ [\,] = 0$

$length'\ x : xs = 1 + length'\ xs$

The length function returns the length of a finite list as an Int. It is an instance of the more general Data.List.genericLength, the result type of which may be any kind of number.

2.5 show

The show function converts values to a readable string. Its type signature is:

$show' :: a \to String$
$show' = \bot$

3 Exercises

Here is the function from the book that gets the number of lines in a document:

$noOfLines :: IO\ ()$
$noOfLines = interact\ lineCount$
  where $lineCount\ input = show\ (length\ (lines'\ input))$

We can alter it in the following ways

1. This function gets the number of words in a document:

   $main :: IO\ ()$
   $main = interact\ wordCount$
     where $wordCount\ input = show\ (length\ (words'\ input))$

2. This function gets the number of characters in a document:

$$
\begin{aligned}
&noOfChars :: IO\ () \\
&noOfChars = interact\ charCount
\end{aligned}
$$

$$
\begin{aligned}
&charCount :: Foldable\ t \Rightarrow t\ a \rightarrow String \\
&charCount\ input = show\ (length\ input)
\end{aligned}
$$