Word frequency count in Haskell

Haskell is a great language for doing a word frequency count for a variety of reasons.

One reason is that Haskell can easily mix source code with LaTeXformatting, as shown below.

In this program we use an associative array because the code to update the word count is simple in a pure language like Haskell.

This basic bookkeeping code declares this to be an executable module and imports some basic libraries.

```
module Main where
import Data.Char
import Data.List
import Data.Map hiding (foldr, map)
```

The *getWord* function takes a string and gives a list of strings that represent the words in the input, after lowercasing and discarding anything that is not a letter. It does this by taking one word from the string's beginning, then discarding non-letters, and then calling itself recursively on the remaining input until only the empty string is left.

```
getWords :: [Char] → [[Char]]
getWords "" = []
getWords abc = let (a, bc) = span isAlpha abc
                   (b, c) = span (¬ ∘ isAlpha) bc
                   in map toLower a : getWords c
```

This next function takes a word and a dictionary, and increments the value in the dictionary.

```
insertListIncrement w d = insertWith (+) w (1 :: Int) d
```

The function *insertWords* takes the list of words as input and returns the dictionary of word to count.

```
insertWords :: [[Char]] → Map [Char] Int
insertWords wordlist = foldr insertListIncrement empty wordlist
```

The *outs* function gets the list of keys from the dictionary, sorts those keys into lexicographical order and then outputs the word and its matching count.

```
outs :: Map [Char] Int → [[Char]]
outs d = map showItem (sort $ keys d)
  where showItem k = k ++ "  " ++ show (d ! k)
```

The *results* function sews together the function that builds the dictionary and the function that builds the output strings.

```
results :: [Char] → [[Char]]
results input = outs ∘ insertWords $ getWords input
```

The *main* function reads a file name inputs.txt and sends the contents of the file to the *results* function above.

```
main = do inputs ← readFile "inputs.txt"
  mapM_ putStrLn (results inputs)
```

As a demonstration, this program is given the non-code portions of this document as input, with the results included below.

a 11 above 1 after 1 an 2 and 11 anything 1 array 1 as 4 associative 1 basic 2 be 1 because 1 beginning 1 below 2 bookkeeping 1 builds 2 by 1 calling 1 can 1 code 4 contents 1 count 6 cs 1 declares 1 demonstration 1 dictionary 5 discarding 2 document 1 does 1 doing 1 easily 1 empty 1 erisson 1 executable 1 file 2 for 2 formatting 1 frequency 3 from 2 function 9 gets 1 getword 1 given 1 gives 1 great 1 haskell 5 imports 1 in 6 included 1 increments 1 input 4 inputs 1 insertwords 1 into 1 is 6 it 1 its 1 itself 1 keys 2 language 2 left 1 letter 1 letters 1 lexicographical 1 libraries 1 like 1 list 3 lowercasing 1 main 1 matching 1 mix 1 module 1 name 1 next 1 non 2 not 1 november 1 of 7 on 1 one 2 only 1 order 1 output 1 outputs 1 outs 1 portions 1 program 2 pure 1 reads 1 reason 1 reasons 1 recursively 1 remaining 1 represent 1 results 3 returns 1 s 1 sends 1 sews 1 shae 1 shown 1 simple 1 some 1 sorts 1 source 1 string 3 strings 2 takes 3 taking 1 that 5 the 28 then 3 this 7 those 1 to 4 together 1 txt 1 until 1 update 1 use 1 value 1 variety 1 w 1 we 1 with 2 word 8 words 2