CS-231 Haskell Language Project 2

Winter 2022

Overview

You will write a program in Haskell to check the spelling of words in a document. You will write this code to run efficiently following the guidelines outlined in this requirements document. Your program will have 3 command line arguments in this order: the name of the document to check (your input document); the name of the dictionary (also an input document), and the name of an output file in which your program will write the results.

Specifics

You will write all of your code in one module in one file. You will have three command line arguments in this order: input file, dictionary file, output file. You will be writing the code to do input and output in this program.

Your program will take all words from the input file, store the line number and position in the line for each word, and sort these data items alphabetically by word. A word is defined as a sequence of alphabetic characters. Note that this means every character which is not alphabetic is not part of a word, so each nonalphabetic character in each line should be changed to a space. Note that your dictionary will have only lower case letters, so the words from your document may have all letters in lower case if that will aid in your coding.

Input

Your input file can be any text file. You can even check your program source file for spelling, and it will have some number of characters which are not letters and are not white space, and you will have upper case and lower case letters with multiple words on a line.

Processing

The reason for using a sorted dictionary is so that your program can sort the document words, then do a comparison of these words with the dictionary with a single scan of the list of document words and a single scan of the dictionary file. It is very easy to write code using the elem function of Haskell to check each document word for inclusion in the dictionary. This coding is very inefficient. If you write your code this way, you will have a 20% point penalty for your solution. Instead, you will have code which will process the two lists, document words and dictionary words, with a single scan of each list. This will be similar to the merge algorithm from merge sort (except, of course, you are not collection any items from the dictionary as values for your final list).

Output

The output file should be easy to read in determining which words are spelled correctly and which ones are not. Have each word from the input document (with the information of line number and position) on its own line in the output file with labels for correct and incorrect words. Also have the correct words starting in the same column, with the incorrect words starting in another column.

Further note that the words will be printed in alphabetic order, not in the order of all incorrect words followed by all correct words (or vice versa).

Notes

1. Your use of the Haskell do keyword will be limited. The only parts of the code where do is allowed is in functions which perform input and/or output.

2. Make sure your write efficient code for comparing the document words with the dictionary.

3. You may want to read each line separately to keep track of the line numbers for the words, or you may want to read the entire file and use the lines functions to break the input into a list of strings, each string in the list corresponding to a line in the file.

4. Remember that the Data.Char module has some functions on characters which might be useful.

5. Note that since we are not using just white space to separate words, the Haskell function words will not work for taking words from the input.

6. Use the dictionary file I provide to use with this function.

Grading

This is a 40 point program. You must turn in a program which compiles in order to receive any credit for this assignment. If your program does not do efficient comparison of the words in the document with the dictionary words there will be a 20% point deduction from the score. On a program which is correct, 6 points are reserved for style, comments, appearance of code and output. In particular, use good variable names; comment each function stating its purpose; make sure you have a leading comment with your name; do not let lines of code or comment be so long as to leave the screen (causing wrap-around on most editors).

Deliverables

Your source code file must be submitted via Blackboard no later than 11:59 p.m. Friday March 11.