Prof. Dr. Visvanathan Ramesh
Alperen Kantarcı
Gamze Akyol

Goethe-Universität Frankfurt am Main
Institut für Informatik

Programmierparadigmen und Compilerbau

Sommersemester 2023

Exercise Nr. 3

Deadline Thursday 08.06.2023, 10:00 in Moodle!

Question 0 (0 Points)

Read the following tips!

- Group work is **not allowed!** You can talk and discuss with your friends about the exercises, but you should solve the exercises **individually**. Plagiarism is not tolerated.
- You can use ChatGPT or other web sources but you cannot ask the whole question. You should be able to explain every step of your code.
- Upload all the files you want to submit into **a single .zip file on Moodle**. Please ensure that you **only submit .pdf and .hs files.**
- Please write your **matriculation number**, **name** and **group number** to each file that you are submitting.
- Your submission file name should be in the following order:

  Name_matriculationNumber_Group_GroupNumber_Exercise_ExerciseNumber.extension.

  For example **AlperenKantarci_1111111_Group_2_Exercise_2.zip**

Question 1 - Type Unification (5 + 5 = 10 Points)

Calculate the type of the following expressions:

a) map lines

b) scanr1 max

With the help of the type rules presented in the lecture. Specify the calculation path. The types are:

```
map :: (a -> b) -> [a] -> [b]
lines :: String -> [String]
scanr1 :: (a -> a -> a) -> [a] -> [a]
max :: Ord a => a -> a -> a
```

Note that when using the application rule, you must also calculate the type substitution using unification. To do this, also specify the calculation path.

Question 2 - List Comprehension & Lambda Expressions (10 Points)

You are given a main.hs file which takes a sentence and lists all its anagrams of that sentence. Along with the Haskell file you will be given a dictionary that contains all meaningful words. The program consists of multiple functions but some of them are not working correctly. Your job will be the fixing and completing the functions and report the mistakes that each function had. Main function is working correctly and you don't have to fix or change.

The general idea is to examine the list of characters. To find the anagrams of a word, we will find all the words from the dictionary which have the same character list. To finding anagrams of a sentence, we will extract subsets of characters to see if we can form meaningful words. For the remaining characters we will solve the problem recursively and then combine the meaningful words we have found.

For example, consider the sentence "You olive". The list of characters in this sentence is "eiloouvy". We start by selecting some subset of the characters, say "i". We are left with the characters "eloouvy". Checking the dictionary we see that "i" corresponds to the word "I", so we've found one meaningful word. We now solve the problem recursively for the rest of the characters "eloouvy". This will give us a list of solutions: [["love", "you"], ["you, "love"]]. We then combine "I" with that list to obtain the sentences "I love you" and "I you love", which are both valid anagrams.

The followings are the 7 functions that you should fix their code and report the errors.

1. wordCharCounts takes a word and returns its character counts. For example, if the word is "mississippi" the result should report that there are 1 of 'm', 4 of 'i', 4 of 's', and 2 of 'p'. **1 pts**

2. sentenceCharCounts takes a list and returns its character counts. **1 pts**

3. dictCharCounts takes a list of dictionary words and returns a mapping from words to their character counts. For example, if the dictionary contains the words "eat", "all", and "tea", this should report that the word "eat" has 1 of 'e', 1 of 'a', 1 of 't' the word "all" contains 1 of 'a', 2 of 'l' the word "tea" contains 1 of 't', 1 of 'e', 1 of 'a'. **1 pts**

4. dictWordsByCharCounts takes in the result of the previous function and returns a map of words which have the same character counts. For the example above the result should map the character counts 1 of 'a', 2 of 'l' to the word list ["all"], and the character counts 1 of 'a', 1 of 'e', 1 of 't' to the word list ["tea", "eat"]. **1 pts**

5. wordAnagrams takes a word and the result of the previous function and returns a list of anagrams for the given word. For example, if the parameters are "ate" and the result given above, it should return the words "tea" and "eat". **2 pts**

6. charCountsSubsets that takes character counts and returns all possible subsets of these counts. For example, the character counts of the word "all" is 1 of 'a', 2 of 'l', and the subsets of the character counts such as : [[ ], ['a',1], ['l',1], ['l',2], [('a',1),('l',2)], [('a',2),('l',1)]] **2 pts**

7. subtractCounts takes two mappings of character counts and returns a new mapping where counts in the second map are subtracted from the counts in the first map. For example, if your first map is 1 of 'a', 3 of 'e', 2 of 'l', and your second map is 1 of 'a', 1 of 'l', the result should be 3 of 'e', 1 of 'l'. You can assume that the second map is a subset of the first map. **2 pts**

Example usage

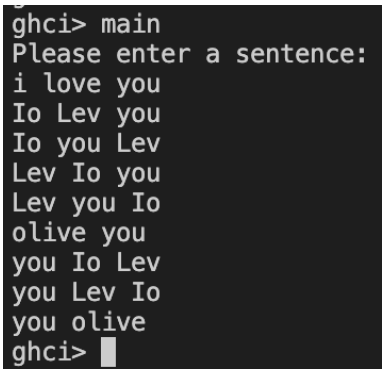

Figure 1: Example for “I love you”

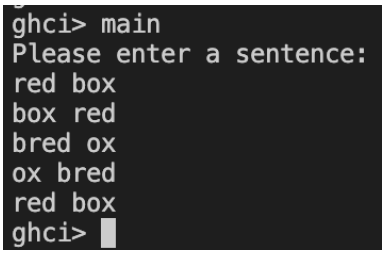

Figure 2: Example for “red box”

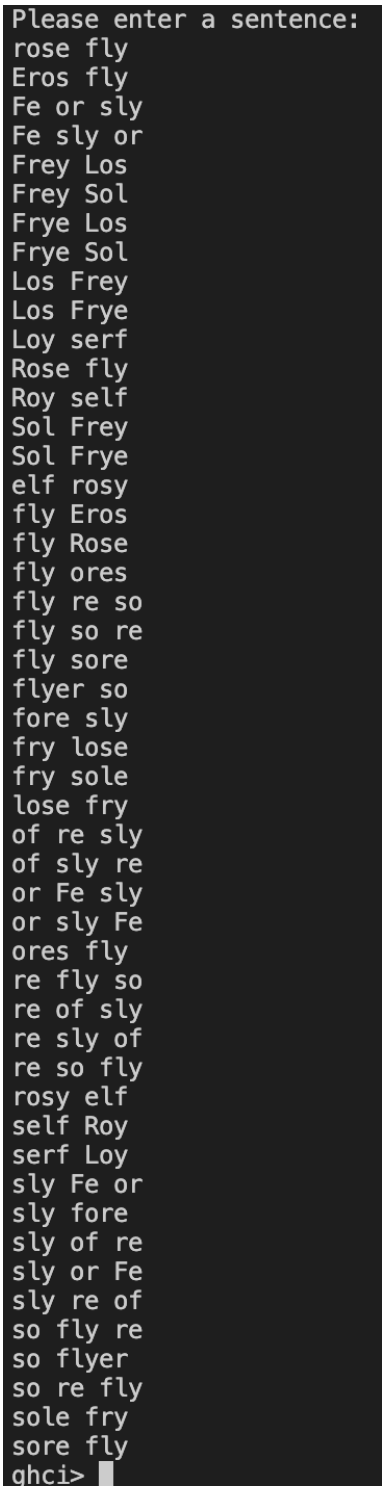

Figure 3: Example for “rose fly”