Middle East Technical University

Department of Computer Engineering

CENG 242

Programming Language Concepts
2023-2024 Spring
Programming Exam 1

Due: 16 March 2024 23:59
Submission: **via ODTUClass**

1 General Specifications

- There are two questions in this Programming Exam, for each question you will implement a given Haskell function.
- The signatures of the functions, their explanations and specifications are given in the following section. Read them carefully.
- Make sure that your implementations comply with the function signatures.
- You may define helper function(s) as needed.

2 Functions

2.1 findInList (40 pts.)

This function takes two lists as arguments: the first one is the pattern and the second one is the list in which you'll be searching for the pattern, subject to the following rules:

1. Each item in the pattern must appear in the second list in the same order that they appear in the pattern.
2. The elements of the pattern needn't appear adjacently in the second list, there may be some interleaving elements not related to the pattern.
3. If there are more than one element in the second list that could fit the pattern, take the one that appears first (see example 2 below).
4. You only need to find the first full appearance of the pattern, you may ignore other occurrences, if there are any.

5. It is guaranteed that there is at least one complete occurrence of the pattern in the second list (so you don't need to consider what to do if the pattern doesn't exist in the second list).

6. You will return the indices of the elements of the pattern in a list, in the same order as the elements to which they belong. (the indices start at 0).

This function has the signature:

```
findInList :: (Eq a) => [a] -> [a] -> [Int]
```

Here are some example function calls:

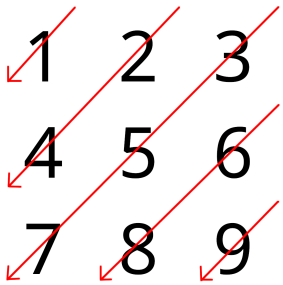
```
ghci> findInList [1,2,3] [1,5,2,6,7,8,3,5]
[0,2,6]

ghci> findInList [5,2,9,5] [2,2,9,5,4,5,9,2,2,5,2,9,9,7,5,2]
[3,7,11,14]

ghci> findInList "Store" "So long, and thanks for all the tea"
[0,13,21,22,30]
```

2.2 findInMatrix (60 pts.)

In this function, we will search for a pattern (like the first function), but unlike the first function this time our function works on 2D matrices, which are represented as nested lists. The order in which you should process the elements of the matrices are as follows:

- We will follow diagonals that trace from the top right to the bottom left of the matrix, and go through each element of the diagonal in order when we process that diagonal.
- We will start from the top left such diagonal, and go down and to the right through the diagonals, processing each one as we go.

Here is an example to demonstrate the order:



the order in which you should visit the elements of the above matrix would be:

```
1 2 4 3 5 7 6 8 9
```

Here are some more examples to help you better understand the traversal order:

```
    [[A, B, C],
      D, E, F],
      G, H, I],
      J, K, L]]
Proper order: A B D C E G F H J I K L

      [[1, 2, 3, 4, 5],
       [6, 7, 8, 9, 10]]
Proper order: 1 2 6 3 7 4 8 5 9 10
```

Go through both the pattern matrix and the second matrix in this order, and match the elements of the pattern to the elements of the second matrix as you encounter them (just like in the first question). All the rules from the first question for the searching apply, except for rule 6. You will return a matrix (instead of a list) of the same shape as the pattern, where each element is the found index of the corresponding element of the pattern.

This function has the signature:

```
findInMatrix :: (Eq a) => [[a]] -> [[a]] -> [[Int]]
```

Here is an example function call:

```
ghci> findInMatrix [[1,5],[3,4]] [[1,2,3,4],[1,2,3,4],[5,6,7,8]]
[[(0,0),(2,0)],[(1,2),(1,3)]]
```

3 Regulations

1. **Implementation and Submission:** The template file named "PE1.hs" is available in the Virtual Programming Lab (VPL) activity called "PE1" on OdtuClass. At this point, you have two options:

   - You can download the template file, complete the implementation and test it with the given sample I/O (and also your own test cases) on your local machine. Then submit the same file through this activity.
   - You can directly use the editor of VPL environment by using the auto-evaluation feature of this activity interactively. Saving the code is equivalent to submit a file.

   If you work on your own machine, make sure that your implementation can be compiled and tested in the VPL environment after you submit it.
   There is no limitation on online trials or submitted files through OdtuClass. The last one you submitted will be graded.

2. **Cheating: We have zero tolerance policy for cheating.** People involved in cheating (any kind of code sharing and codes taken from internet included) will be punished according to the university regulations.

3. **Evaluation:** Your program will be evaluated automatically using "black-box" technique so make sure to obey the specifications. No erroneous input will be used. Therefore, you don't have to consider the invalid expressions.

- **Important Note:** The given sample I/O's are only to ease your debugging process and NOT official. Furthermore, it is not guaranteed that they cover all the cases of required functions. As a programmer, it is your responsibility to consider such extreme cases for the functions. Your implementations will be evaluated by the official test cases to determine your ***actual*** grade after the deadline.