Haskell Practice Test

Imperial College London

Department of Computing

Lexis Practice in Haskell

Friday 11 December 2020
09:00 to 10:00
ONE HOUR
(including 10 minutes planning time)

- The maximum total is **0 marks**.
- Credit will be awarded throughout for conciseness, clarity, *useful* commenting, and the appropriate use of the various language features.
- **Important:** TEN MARKS will be deducted from solutions that do not compile in the test environment, which will be the same as the lab machines. Comment out any code that does not compile before you submit.
- The examples and test cases here are not guaranteed to exercise all aspects of your code. You are therefore advised to define your own tests to complement the ones provided.
- The extracted files should be **directly inside** your gitlab repository. You can extract the archives by adjusting the following command:
  `7z x filename.7z`
- Push the final version of your code to `gitlab` before the deadline, and then go to `LabTS`, find the final/correct commit and submit it to `CATe`.

Instructions

- The (*real*) Lexis Tests consist of:
  - 10 minutes of reading time (*without computer access*)
  - 170 minutes of coding time
- Always remember: **Keep Saving Your Work!**
- Do not change any of the filenames and do not move any of the files. The excuse: "*My IDE did it!*" will not be accepted.
- If you encounter any technical issues (*e.g. your computer gets stuck*), record any evidence (*videos, photos, etc.*).
- If you encounter any other issues (*e.g. your favourite IDE does not work*), use the terminal instead.
- **We will not be providing any help on how to solve this test**. Join us in the lab session afterwards to discuss any queries regarding this test.
- If you identify any issues with the test spec, please let us know via email.
- During the last 10 minutes, make sure that your code compiles, that the files are where they are supposed to be, and that you have removed any unnecessary imports and comments. Once again, the excuse: "*My IDE did it!*" will not be accepted.
- Code in comments is not marked, but may be considered for feedback.

Haskell String Matching

In this mini exercise, you are invited to write some basic string processing functions in Haskell. See if you can write each one using just one line of code!

1. Define a function `isPrefix :: String -> String -> Bool` that returns `True` iff the first given string is a prefix of the second. For example,

   ```
   *Main> isPrefix "has" "haskell"
   True
   *Main> isPrefix "" "haskell"
   True
   *Main> isPrefix "ask" "haskell"
   False
   ```

2. Define a function `removePrefix :: String -> String -> String` that returns the result of removing a prefix (first argument) from a given string (second argument). A precondition is that the first string is a prefix of the second. For example:

   ```
   *Main> removePrefix "ja" "java"
   "va"
   *Main> removePrefix "" "java"
   "java"
   ```

3. Define a function `suffixes :: [a] -> [[a]]` that returns the lists of all suffixes of a given string in descending order of length. For example:

   ```
   *Main> suffixes "perl"
   ["perl","erl","rl","l"]
   ```

   Hint: one way to do this is by repeated application of the `tail` function.

4. Use the functions `isPrefix` and `suffixes` to define a function `isSubstring :: String -> String -> Bool` that returns `True` iff the first string is a substring of the second. For example:

   ```
   *Main> isSubstring "ytho" "python"
   True
   *Main> isSubstring "ythough" "python"
   False
   ```

5. Use the functions `isPrefix` and `suffixes` above to define a function `findSubstrings :: String -> String -> [Int]` that returns the indices of all occurrences of a given substring (first argument) in a given text string (second argument). If there are no occurrences the function should return `[]`. For example:

   ```
   *Main> findSubstrings "an" "banana"
   [1,3]
   *Main> findSubstrings "s" "mississippi"
   [2,3,5,6]
   ```

   The order of the elements in the result list is not important.

   Hint: You might consider using a list comprehension.