Screen-scraping
Informatics 1 – Introduction to Computation
Functional Programming Tutorial 4

Banks, Cooper, Heijltjes, Korte, Lehtinen, Melkonian, Vlassi-Pandi, Wadler, Yallop

Week 5
due 4pm Wednesday 21 October 2020
tutorials on Friday 23 October 2020

Attendance at tutorials is obligatory; please send email to lambrose@ed.ac.uk if you cannot join your assigned tutorial.

Good Scholarly Practice: Please remember the good scholarly practice requirements of the University regarding work for credit. You can find guidance at the School page

http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct.

This also has links to the relevant University pages. Please do not publish solutions to these exercises on the internet or elsewhere, to avoid others copying your solutions.

1 Installation Process

To be able to run the optional part of **Tutorial 4** on your personal machine, you first need to install the `HTTP` package.

- For `Linux` and `OS X` run the following commands:

  ```
  cabal update
  cabal install --lib HTTP
  ```

  Note: The above steps assume that you have already installed `Haskell` successfully.

- For `Windows` this package requires a `Unix` compatibility toolchain to be installed. To do so:
  - If you have `Windows 10`, we suggest that you install the Windows Subsystem for Linux. The easiest way is to install it is from the Windows Store as described in the previous link.

    Note: The Linux Subsystem mounts a new file system within the existing Windows partition; consequently to access your C drive you need to `cd /mnt/c`.
  - If you have an earlier version of `Windows` we suggest that you either install Git-bash or Cygwin by following the instructions in the corresponding links.

  Once you have installed one of the above tools, you have to open them and install `Haskell` in them as described in **Tutorial 1**, *BUT* this time follow the instructions for `Linux`! After the installation has been done successfully, you may run:

  ```
  cabal update
  cabal install --lib HTTP
  ```

2 Basic Screen Scraper

A "screen scraper" is a tool used to extract data from web sites, by looking at their source. In this exercise, you will write one of the most hated screen scrapers: one that extracts email addresses. Why is it hated? Because people use screen scrapers like that to collect email addresses to send spam to. However, in this exercise we will show you a *useful* purpose of the email screenscraper!

We are going to be extracting names and emails from web pages written in HTML (HyperText Markup Language). For instance, from the following HTML:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
  <head>
     <title>FP: Tutorial 4</title>
  </head>
  <body>
     <h1>A Boring test page</h1>
     <h2>for tutorial 4</h2>
     <a href="https://course.inf.ed.ac.uk/inf1a">Inf1A Learn</a><br>
     <b>Lecturer:</b> <a href="mailto:wadler@inf.ed.ac.uk">Philip Wadler</a><br>
     <b>TA:</b> <a href="mailto:cchirita@exseed.ed.ac.uk">Claudia-Elena Chirita</a>
   </body>
</html>
```

We are going to extract a list of the "`<a>`" elements, which contain URLs (Uniform Resource Locators). If a URL begins with `http:` it is an address of a web page; if it begins with `mailto:` the rest of it is an email address. For the document above, here is the list of links (each one contains some extra data at the end, which is an artifact of the technique we use):

```
["https://course.inf.ed.ac.uk/inf1a\">Inf1A Learn",
 "mailto:wadler@inf.ed.ac.uk\">Philip Wadler",
 "mailto:cchirita@exseed.ed.ac.uk\">Claudia-Elena Chirita"]
```

From this list, we will in turn extract a list of names and email addresses:

```
[("Philip Wadler","wadler@inf.ed.ac.uk"),
 ("Claudia-Elena Chirita","cchirita@exseed.ed.ac.uk")]
```

The file `Tutorial4.hs` contains the test html-document and the lists above: `testHTML`, `testLinks`, and `testAddrBook`.

Notice that the type of `testLinks` is `[Link]` and the type of `testAddrBook` is `[(Name,Email)]`. In other words: `testLinks` is a list of `Link`s, and `testAddrBook` is a list of pairs of a `Name` with an `Email`. If you look in the file `Tutorial4.hs` you will find the following type declarations:

```
type Link = String
type Name = String
type Email = String
type HTML = String
type URL = String
```

These type declarations simply define aliases for the familiar type `String`, which makes your program more readable.

Note: If you want to know more about HTML, have a look at: `http://www.w3schools.com/html/`.

Exercise 1

Write a function `sameString :: String -> String -> Bool` that returns `True` when two strings are the same, but ignores whether a letter is in upper- or lowercase. For example:

```
*Main> sameString "HeLLo" "HElLo"
True
*Main> sameString "Hello" "Hi there"
False
```

Warning: Unintuitively, the mapping between upper and lower case characters is not one-to-one. For example, the greek letter μ and the micro sign map to the same upper case letter. What does your code do on `sameString "\181" "\956"`? In this case either behaviour is acceptable, as long as the tests don't fail on input containing these characters!

Exercise 2

(a) Write a function `prefix :: String -> String -> Bool` that checks whether the first string is a prefix of the second, like the library function `isPrefixOf` that you used before, but this time it should be case-insensitive.

```
*Main> prefix "bc" "abCDE"
False
*Main> prefix "Bc" "bCDE"
True
```

(b) Check your function using the predefined test properties `prop_prefix_pos` and `prop_prefix_neg`. Why is `prop_prefix_pos` not sufficient to test your code? Think about faulty code that would pass this test.

Exercise 3

(a) Write the function `contains` as in tutorial 2, but case-insensitive. For example:

```
*Main> contains "abcde" "bd"
False
*Main> contains "abCDe" "Bc"
True
```

(b) Write a test property `prop_contains :: String -> Int -> Int -> Bool` to test your `contains` function. You can take inspiration from `prop_prefix_pos`.

Exercise 4

(a) Write a case-insensitive function `takeUntil :: String -> String -> String` that returns the contents of the second string *before* the first occurrence of the first string. If the second string does not contain the first as a substring, return the whole string. E.g.:

```
*Main> takeUntil "cd" "abcdef"
"ab"
```

(b) Write a case-insensitive function `dropUntil :: String -> String -> String` that returns the contents of the second string *after* the first occurrence of the first string. If the second string does not contain the first as a substring, return the empty string. E.g.:

```
*Main> dropUntil "cd" "abcdef"
"ef"
```

Exercise 5

(a) Write a case-insensitive function `split :: String -> String -> [String]` that divides the second argument at every occurrence of the first, returning the results as a list. The result should not include the separator. For example:

```
*Main> split "," "comma,separated,string"
["comma","separated","string"]
*Main> split "," "no comma"
["no comma"]
*Main> split "the" "to thE WINNER the spoils!"
["to "," WINNER "," spoils!"]
*Main> split "end" "this is not the end"
["this is not the ",""]
```

Your function should return an error if the first argument, the separator string, is an empty list. You will find your functions `takeUntil` and `dropUntil` useful here.

(b) Write a function `reconstruct :: String -> [String] -> String` that reverses the result of `split`. That is, it should take a string and a list of strings, and put the list of strings back together into one string, with the first string everywhere in between (but not at the start or at the end).

(c) Look at the predefined test function `prop_split` and try to understand what it does. Use it to test your `split` function.

Exercise 6

(a) Use your functions `split` and `takeUntil` to write a function `linksFromHTML :: HTML -> [Link]`. You can assume that a link begins with the string `<a href="` and ends with the string `</a>`. Don't include the separators in the results, and don't include anything in the HTML that precedes the first link. Example:

```
*Main> linksFromHTML testHTML
["https://course.inf.ed.ac.uk/inf1a\">Inf1A Learn",
 "mailto:wadler@inf.ed.ac.uk\">Philip Wadler",
 "mailto:cchirita@exseed.ed.ac.uk\">Claudia-Elena Chirita"]
```

Note: to include the character " in a string, precede it with a backslash (\), as \".

(b) Use `testLinksFromHTML` to test your function on the given sample data. Note that this test does not require QuickCheck, since it does not depend on randomly generated input.

Exercise 7

Write a function `takeEmails :: [Link] -> [Link]` which takes just the email addresses from a list of links given by `linksFromHTML`. Example:

```
*Main> takeEmails testLinks
["mailto:wadler@inf.ed.ac.uk\">Philip Wadler",
 "mailto:cchirita@exseed.ed.ac.uk\">Claudia-Elena Chirita"]
```

Exercise 8

Write a function `link2pair :: Link -> (Name, Email)` which converts a mailto link into a pair consisting of a name and the corresponding email address. The name is the part of the link after `<a href="...">` and `</a>` tags; the email address is the part in the quotes after `mailto:`. Add an appropriate error message if the link isn't a `mailto:` link. Example:

```
*Main> link2pair "mailto:john@smith.co.uk\">John"
("John","john@smith.co.uk")
```

Exercise 9

(a) Combine your functions `linksFromHTML`, `takeEmails` and `link2pair` to write a function `emailsFromHTML :: HTML -> [(Name, Email)]` that extracts all mailto links from a webpage, turns them into `(Name, Email)` pairs, and then removes duplicates from that list. Example:

```
*Main> emailsFromHTML testHTML
[("Philip Wadler","wadler@inf.ed.ac.uk"),
 ("Claudia-Elena Chirita","cchirita@exseed.ed.ac.uk")]
```

Note: the library function `nub :: [a] -> [a]` removes duplicates from a list.

(b) You can test your function with `testEmailsFromHTML`.

3 Optional Material: Pulling in live URLs

In `Tutorial4.hs` a test URL is predefined, `testURL`. Since it is just a string, you can ask GHCi to display it. Do this, and copy-paste the link into your web browser to see what page it refers to. To see the HTML of the page right-click and select 'view page source', or a similar option depending on your browser.

```
*Main> testURL
"http://www.inf.ed.ac.uk/teaching/courses/inf1/A/testpage.html"
```

The function `emailsFromURL`, which is already defined in `Tutorial4.hs`, extracts email addresses from a URL using your very own `emailsFromHTML`. Test your function `emailsFromHTML` by testing it on real URLs of your choice.

As you will have seen, `emailsFromURL` sometimes produces a rather long list of names and email addresses. Sometimes you have a vague idea of who it is you are looking for and in that case, you do not want to go through the entire list of names one-by-one. Over the next few exercise you will be implementing a function `emailsByNameFromURL` in order to find the email address of a person whose name you know.

Exercise 10

Write a function `findEmail :: Name -> [(Name,Email)] -> [(Name,Email)]` which given (part of) a name and a list of `(Name,Email)` pairs, returns a list of those pairs which match the name. Example:

```
*Main> findEmail "Claudia" testAddrBook
[("Claudia-Elena Chirita","cchirita@exseed.ed.ac.uk")]
*Main> findEmail "wadler" testAddrBook
[("Philip Wadler","wadler@inf.ed.ac.uk")]
*Main> findEmail "Fred" testAddrBook
[]
```

Exercise 11

Define the function `emailsByNameFromHTML :: HTML -> Name -> [(Name, Email)]`. This function should take an HTML string and (part of) a name, and return all `(Name,Email)` pairs which match the name.

```
*Main> emailsByNameFromHTML testHTML "Claudia"
[("Claudia-Elena Chirita","cchirita@exseed.ed.ac.uk")]
```

The function `emailsByNameFromURL`, which is already defined in `Tutorial4.hs`, uses your very own `emailsByNameFromHTML` function to extract the email address of a certain person from a live URL. Maybe you can try it on your own webpage, if you have one.

Exercise 12

If one of you, or somebody you know has their email address published on a website, check to see if `emailsByNameFromURL` can find it! Note that for the function to work properly you should have implemented correctly the function `emailsByNameFromHTML`.