PLC: Homework 5 [75 points]

Due date: Wednesday, March 1, 10:30pm
3 point bonus if you turn in by Tuesday, February 28th, 10:30pm

About This Homework

The goal of this homework is to learn more about writing programs that manipulate linguistic artifacts, where those artifacts are represented as data in the programming language. We will be working with the HTML and CSS representations we have seen in class starting Feb. 16.

The first thing you should do for this homework is copy the files from the `hw/hw5` subdirectory of the class repository, to a new `hw5` subdirectory of your personal repository, similarly to previous homeworks. You will just be modifying two of these files, namely `html.agda` and `html-thms.agda`, but we will refer to others of them in the instructions below.

There is extra credit possible for this assignment, so you can earn over 75 points (but 75 is a perfect score).

Partners Allowed

For this homework, you may work by yourself or with one partner (no more). See the instructions for `hw2` for how to create the `ack.txt` and `partner.txt` files that are required if you work with a partner

All files should go in a hw5 subdirectory (which you create) of your personal repository. Please call the subdirectory exactly that. Do not call it `Homework 4` or `Hw5` or other variations, or we will penalize you 5 points.

How to Turn In Your Solution

Make sure you have done a "Subversion add" on the `hw5` directory you created in your personal repository, and on the `.agda` files you have to submit for this assignment, and then do a "Subversion commit" on your personal repository directory to submit them. You can commit as many times as you want up to the deadline. If you commit after the deadline, we will use the last existing version before the deadline. Work submitted after the deadline will not count.

How to Check Your Solution

As for previous homeworks, you might want to make sure you have correctly submitted your solution via Subversion. Again, just go to the URL for your personal repository using a web browser, log in with your HawkId and password, and you can see what has been submitted.

Make sure each of the files you submit can be checked by Agda without any holes, yellow highlighting, or red highlighting. If any file does not check this way, we will penalize

you 5 points total (if multiple files do not check, we will still just penalize you 5 points). Any problems you do not solve should simply be removed from the `.agda` files, so that there are no holes (question marks) in the files.

How To Get Help

You can post questions in the `hw5` section on Piazza, or elsewhere on Piazza. See the course's Google Calendar, linked from Piazza, for the locations and times for office hours, including evening Skype office hours for Prof. Stump.

1 Reading

Please read Sections 5.1 and 5.3 of the book.

2 Provided code, and compiling

The code I am providing you is in `html.agda`, which has definitions of `html` and `css` datatypes, and operations on those; and `hmain.agda`, which defines `main` entry point to the `hmain` program, handles command-line arguments, and contains testcases. **Please do not modify hmain.agda, as we will assume it does not contain any of your code.** All your modifications should be made to `html.agda`, or to `html-thms.agda` (which I am splitting out separately because Agda does not like compiling files with holes in them).

To complete this assignment, you will need to compile `hmain.agda` to an executable program, called `hmain.exe` (Windows) or else just `hmain` (Linux, Mac). In emacs, if you are in a buffer visiting `hmain.agda`, you type Control-c Control-x Control-c to compile. Then you may be asked for "backend", and you should type in "GHC" and hit enter. Then the Agda compiler will be invoked to generate Haskell code, and the Haskell compiler will then (automatically) be run on that generated code to create the executable.

Once you have compiled `hmain`, you can run it from your terminal or command shell, like this on Windows:

```
hmain not 1
```

And like this on Mac and Linux:

```
./hmain not 1
```

The command-line arguments are

- either "indented" or "not" to control whether the html is printed indented (you must implement this, as described below) or unindented (which I am providing)
- a number for test data. Tests are numbered 1 through 4. Presently, most of them do not do anything interesting (you will change that in the problems below).

3 Indented printing of html [15 points]

When trying to debug code or other computer-language artifacts generated by a program, it is generally a good idea to invest in decent printing functions, for printing out those artifacts in as readable a form as possible. This is because for debugging, you will likely spend a good bit of time trying to read through output your code has generated, and so it is wise to print that output in a nice format (so you can read it easily).

So your first problem is to modify the definition of `html-to-indented` in `html.agda` so that it prints out html in a properly indented manner. For this, you should put newlines (`"\textbackslash n"` in Agda) after each start tag (like `<html>`) and before each end tag (like `</html>`). More importantly, you should indent each line that follows a newline. The indentation should start out at zero spaces, and should be incremented by one space inside each subelement. So the sample HTML which you can see by running `hmain` with command-line arguments `not` and `1`, which looks like this

```
<html> <body> <p> Go Hawks </p> <p> Big Ten </p> </body> </html>
```

should print out indented like this:

```
<html>
 <body>
  <p>
   Go Hawks
  </p>
  <p>
   Big Ten
  </p>
 </body>
</html>
```

You can find this output also as `sample-indented.html`. We will run your `hmain` with `indented` to print some html samples out in indented form (possibly not the html samples we provided you with).

[15 points]

4 Generating html [25 points]

HTML has tags `ul` for unordered lists and `ol` for ordered lists. Immediately inside elements built with these tags, you can use tag `li` for list items. If you open `output-html-list-indented.html` in a web browser, you will see basic examples.

1. Change the definition of `html-list` (in `html.agda`) so that it takes in a list of `html` values, and generates a new `html` value that represents an HTML list. That list will consist of one `li` element for each value in the input list given to `html-list`. When your code is working correctly, you should be able to generate the output in `output-html-list.html` if you run `hmain not 2`. [10 points]

2. Fill in the proof of `html-list-preserves-length` in `html-thms.agda`. [5 points]

3. Change the definition of `list-prefixes` in `html.agda` so that it takes in an input list `l` and returns a list of all the prefixes of `l`, from shortest to longest. A prefix of a list `l` is a list that begins `l`. For this problem, we will just consider nonempty prefixes. The entire list will be considered a prefix of itself. For example, the list of (nonempty) prefixes of

   ```
   'a' :: 'b' :: 'c' :: []
   ```

   is

   ```
   ('a' :: []) ::
   ('a' :: 'b' :: []) ::
   ('a' :: 'b' :: 'c' :: []) ::
   []
   ```

   You can make use of this function in one of the problems below. [10 points]

5 Styling HTML [40 points]

Here we will write some functions to add inline styles to some HTML.

1. Fill in the definition of `p-prefixes` in `html.agda` so that it returns a list of elements tagged with `p`, where each such element contains text with a prefix of the input list of characters (you can convert a list of characters back to a string using the library function `𝕃char-to-string`). The prefixes should be in order from shortest to longest.

   If you did not manage to get `list-prefixes` working in the previous problem, you should still be able to get points here by hard-coding the list of all prefixes of the string which is given as an argument to `p-prefixes` in `hmain` (but it will be a nuisance to write these out so maybe try `list-prefixes` one more time!).

   Example output is found in `p-prefixes.html`. Your code will be called when your run `hmain I 3`, where I can be "indented" or "not".

   [10 points]

2. Fill in the definition of `gradient` so that it generates a list of all triples of numbers `r , g , b` (red, green, blue) starting from the given starting values and incrementing each value by the given increment until any of the given ending values is exceeded (do not include a tuple when the ending values are actually exceeded). So for the function call

   ```
   gradient 0 0 0 10 0 0 125 0 0
   ```

   the output is

   ```
   (0 , 0 , 0) ::
   (10 , 0 , 0) ::
   (20 , 0 , 0) ::
   (30 , 0 , 0) ::
   (40 , 0 , 0) ::
   ```

```
(50 , 0 , 0) ::
(60 , 0 , 0) ::
(70 , 0 , 0) ::
(80 , 0 , 0) ::
(90 , 0 , 0) ::
(100 , 0 , 0) ::
(110 , 0 , 0) ::
(120 , 0 , 0) :: []
```

[10 points]

3. Fill in the definition of `tuples-to-decls` so that it takes in a list of (r,g,b) tuples of numbers and produces a list of pairs (in the code I am using `css-decl` for a name for pairs of strings) that look like

   ```
   "color" , "rgb(r,g,b)"
   ```

   where the r, g, and b values are the ones from the tuples.

   [10 points]

4. Fill in the definition of `zip-html-css-decls` so that it takes a list of html values and a list of `css-decls` and recurses down both lists, adding a `style` attribute for the current `css-decl` if the current html value is `tagged`. If it is not `tagged`, you can just skip that value. The resulting list should be returned. Whenever either input list becomes empty during the recursion, you can just return the empty list.

   When this is implemented, you should see output by running `hmain I 4` (I is indented or not). Sample output generated by my solution is `moses.html`.

   [10 points]