PLC: Workout 5 [90 points]

Due date: Wednesday, April 8th by midnight

About This Homework

This assignment is about parsing with the `alex` and `happy` tools (lexer- and parser-generator) for Haskell. You will write regular expressions and grammars. It is rather tricky to get these exactly right, so you should test carefully. (I am providing some testcases.)

Note on the pandemic: while attempting to preserve the quality of the course during this difficult time, I do not want to add course stress to your difficulties if you get sick (as statistically, many are likely to do). Therefore, students who contract COVID-19 are excused from an assignment missed due to the disease and may elect either to receive the average of their other grades in place of the missing one, or to complete it when they are recovered (including possibly taking an incomplete). This can be applied to one or two missed assignments. If more than two assignments are missed, though, the student will need to make up those beyond two, possibly taking an incomplete. (Of course, other University-recognized excuses for missing a deadline for an assignment will also be recognized, but addressed probably just with extensions.)

How to Turn In Your Solution

Please submit your solution via ICON. The required files for this assignment are these (you can just turn in your whole `workout5` directory):

- `RegexProblems/Tokens.x`
- `Grammar1/Grammary.y`
- `Grammar2/Grammary.y`

Please use exactly the file names we are requesting. We will require you to resubmit your homework with a 5-point penalty if the names are not exactly as we are requesting. This is for purposes of grading scripts. It is ok if ICON adds a number to your file name on multiple submission (which is allowed up to the deadline).

Partners Allowed

You may work alone or with one partner. You should both turn in your solution to the assignment, which we expect will be the same (but is allowed to be different, if you worked together but then you decided to add to your solution – or whatever the scenario). Also, you need to turn in a file called `partner.txt` which lists your partner's name. This will let us know that you worked with that person (lest we incorrectly think you plagiarized another student's similar submission).

How To Get Help

You can post questions in the `workouts` section on Piazza.

The course staff will be holding office hours by Zoom, at times to be announced on Piazza. (In-person meetings are currently forbidden by University policy in response to the epidemic.) My times will likely change from what they were earlier in the semester, so please check the course calendar:

`https://calendar.google.com/calendar/embed?src=a5d6qokrert25ce093iksp8np0%40group.calendar.google.com&ctz=America%2FChicago`

1 Reading

Read Chapters 1 and 2 of *Verified Functional Programming in Agda*, which is freely available on the UI campus (or UI VPN) here:

`https://dl-acm-org.proxy.lib.uiowa.edu/doi/book/10.1145/2841316`

2 Regular expressions [50 points]

In the subdirectory `RegexProblems` (of `workout5`), you will find a `Tokens.x` file. You will add regular expressions there to solve the following problems. A solution for `p0` is provided for you already as an example. You will just add your regular expressions in a similar way as the example one, to `Tokens.x`. (All your solutions can be added to just this one `Tokens.x` file and it should work correctly.) Positive testcases are provided in files with names like `p0-yes.txt`, and negative ones (that should cause your lexer to report an error) in files like `p0-no1.txt`. Note that lexers are expected to return a sequence of tokens, and so input files that have matching strings one after the other will return lists of length greater than 1 of matches. Note also that you need to compile using `cabal v1-build`. [10 points each]

- **p0:** match one 'a' followed by zero or more 'b's. (*already done for you, just an example*)
- **p1:** match zero or more 'c's followed by one or more 'd's.
- **p2:** match one or more 'e' or 'f' characters.
- **p3:** match a single lowercase character ('a' through 'z'), then match zero or more characters that are either lowercase characters or numeric digits ('0' through '9').
- **p4:** match strings quoted by either single or double quotes, containing zero or more characters that are either upper or lowercase letters or space.
- **p5:** match any nonempty sequence of X and Y characters where there is an even number (including 0) of X characters.

You will find the expected output for all the `yes` tests in `expected-output.txt`.

3 Grammars [40 points]

Each of the following problems has its own subdirectory. You will only need to modify the productions part of the `Grammar.y` file in that directory (`Main.hs`, `Tokens.x`, and other files do not need to be modified). You may wish to refer to the lecture materials for March 10th, where you will find a reasonably representative example grammar. Again, compile using `cabal v1-build`.

- **Grammar1:** add productions to `Grammar1/Grammar.y` to recognize simple Haskell type expressions built from the unit type `()`, parentheses, and right-associative arrow types. Positive testcases are given in `yes1.tp`, etc. [15 points]
- **Grammar2:** add productions to `Grammar2/Grammar.y` to recognize a simple fragment of an imperative programming language, as follows:
  - a `Prog` is a list of `FunCalls`, where each `FunCall` is followed by a semicolon
  - a `FunCall` is either a simple identifier (i.e., variable name like "x"), or else a function call, which consists of an identifier, then a left parenthesis, then a comma-separated list of zero or more `FunCalls`, and then a right parenthesis.

  For example, a simple `Prog` is:

  ```
  a;
  f(b,g(c));
  ```

  The `FunCalls` are `a` and `f(b,g(c))`. Notice that the second `FunCall` is nested. Positive testcases are again `yes1.prog`, etc. [25 points]