CSCE 314 [Section 501] Programming Languages – Spring 2020

Hyunyoung Lee

Homework Assignment 4B

Assigned on Friday, March 6, 2020

Electronic submission to eCampus due **at 10:00 p.m., Monday, March 23, 2020**
By electronically submitting this assignment to eCampus by logging in to your account, you are signing electronically on the following Aggie Honor Code:
"On my honor, as an Aggie, I have neither given nor received any unauthorized aid on any portion of the academic work included in this assignment."

In this assignment, you will earn total 60 points. Here are some general instructions.

1. Read the problem descriptions and requirements carefully! There may be significant penalties for not fulfilling the requirements.

2. Some problems ask you to explain the working of your function with the given input. Your explanation must be consistent with your definition of the function. Your work will be graded not only on the correctness of your answer, but also on the consistency and clarity with which you express it.

3. This homework set is an *individual* homework, not a team-based effort. Discussion of the concept is encouraged, but actual write-up of the solutions must be done individually and your final product – the code as well as the comments and explanations – should never be shared.

4. Submit electronically exactly one file named *YourLastName-YourFirstName*-**hw4b**.hs, and nothing else, on eCampus.tamu.edu.

5. Make sure that the Haskell script (the .hs file) you submit compiles without any error when compiled using the Glasgow Haskell Compilation System (ghc or ghci) version 7 and above[1].

   If your program does not compile, you will receive very few points (more likely zero) for this assignment. To avoid receiving zero for the entire assignment, if you cannot complete defining a function correctly without compile error, you can set the function definition `undefined`, see the skeleton code provided.

6. Remember to put the head comment in your file, including your name, UIN, and *acknowledgements of any help received* in doing this assignment. Again, remember the Honor Code!

[1]Version 7 is installed in the departmental servers (linux.cse.tamu.edu and compute.cse.tamu.edu), and version 8 is what you will get if you install the Haskell system in your computer.

Keep the name and type of each function exactly the same as given in the problem statement and the skeleton code. Also, do not remove or modify the test list or the main function. If you remove or modify those, then you will be penalized for that.
This homework set is on Monadic Parsing, Chapter 13 of the Haskell textbook. Please, read the textbook sections 13.1–13.8 very carefully! Also, download the Parsing library (which is already included in the skeleton code of this homework), load it into GHCi and try out the examples in the textbook (and my lecture slides) for the basic parsers and derived primitives. This will help you understand the basic parsing functionalities such as the sequencing operation `(>>=)` for parsers and the choice operator `(<|>)` (Section 13.5). The choice parser `p <|> q` returns the result of the first parser `p` if `p` succeeds on the input; otherwise it returns the result of the second parser `q` applied to the same input.

Problem 1. (4 points) Put your full name, UIN, and *acknowledgements of any help received* in the head comment in your .hs file for this assignment.

Problem 2. (26 points) The parser for expressions discussed in section 13.8 (page 190) is of type `expr :: Parser Int`, `term :: Parser Int` and `factor :: Parser Int`. That is, the parsed result of an expression is an integer. For example, if the input string was `"2*(3+4)"`, the parsed result (by `parse expr "2*(3+4)"`) would be `[(14,"")]` (note that the `eval` function given in the end of section 13.8 extracts the integer value 14 out of this singleton list). Your task is to explain the step-by-step working of the two example expressions below.

1. (13 points)

   ```
   > parse expr "2*(3+4)"
   ```

   results in

   ```
   [(14,"")]
   ```

2. (13 points)

   ```
   > parse expr "1+2*3"
   ```

   results in

   ```
   [(7,"")]
   ```

Your explanation should be as specific as possible, strictly using the definitions of `expr`, `term` and `factor` given in section 13.8 (page 190) in the textbook.

Problem 3. (30 points) Exercise 13.5 (Modified). Use the following type `Expr`.

```
data Expr = Val Int | Add Expr Expr | Mul Expr Expr
             deriving Show
```

Your task in this problem is to modify the parser for expressions to have type `expr :: Parser Expr`, `term :: Parser Expr` and `factor :: Parser Expr`. Thus, the parsed result will be an abstract syntax tree, for example, if you do

```
> parse expr 2*(3+4)
```

it should output

```
[(Mul (Val 2) (Add (Val 3) (Val 4)),"")]
```

and

```
> parse expr "1+2*3"
```

should output

```
[(Add (Val 1) (Mul (Val 2) (Val 3)),"")]
```

Have fun!