Project 3

Due: Mar 30, before midnight.

Important Reminder: As per the course Academic Honesty Policy, cheating of any kind will minimally result in your letter grade for the entire course being reduced by one level.

This document first provides the aims of this project. It then lists the requirements as explicitly as possible. It then hints at how these requirements can be met. Finally, it describes how it can be submitted.

Aims

The aims of this project are as follows:

- To give you exposure to recursive and functional programming.
- To expose you to Haskell programming.

Requirements

Use the Glasgow Haskell Compiler ghc to implement all the functions specified in Main.hs or in the skeleton files referenced by `Main.hs`.

When a function requirement specifies a restriction on how that function should be implemented, your implementation **must** meet that restriction.

Provided Files

You should use the provided prj3-sol directory as a starting point for your project by copying it into your `i571/submit` directory. It contains the following files:

Main.hs
A skeleton file which contains specifications for the some of the functions you are required to write. For each function it contains a dummy implementation with a `"TODO"` error. It also contains QuickCheck tests for these functions as well as a `main` function which runs all of the tests for the entire project.

EvalIntExpr.hs
This is a separate skeleton file for implementing a single function `evalIntExpr`.

EvalIdExpr.hs
This is a separate skeleton file for implementing a single function `evalIdExpr`.

EvalMaybeExpr.hs
This is a separate skeleton file for implementing a single function `evalMaybeExpr`.

PostfixExpr.hs
This is a separate skeleton file for implementing a single function `postfixExpr`.

TestUtils.hs
A file containing utilities used for testing. You should not need to modify this file.

README

A template `README` which you should complete and submit.

Hints

To get started:

1. Simply copy the provided `prj3-sol` directory into your `~/i571/submit` directory.
2. Install `QuickCheck` using:

   ```
   $ cabal update
   $ cabal install --lib QuickCheck
   ```

   This may take a few minutes.

Keep the following points in mind while working on your project:

- Review the class Haskell slides. In particular, the member function covered in class will be useful for exercise #3 and the do syntactic sugar will be useful for exercise #7.
- Details of specific syntax for different Haskell constructs are available in this Haskell Cheat Sheet.
- This document may help you understand Haskell error messages.
- If you do not understand the specs for a function, please look at the provided tests for examples of the expected use of the function.
- When writing recursive functions, use the structure of the data to guide your code. Typically your code will contain one equation for each variant of the data.
- Hints for each function are available in the provided skeleton files.
- If you want to experiment with functions or expressions, simply type the expression or function into `ghci`. For functions, you will need to define the function within a `let`. You will either need to type the entire definition within a single line using something like:

  ```
  ghci> let f x = a * x + b where { a = 2; b = 3 }
  ```

  or turn on multiline mode using:

  ```
  ghci> :set +m
  ```

- You should not need to use Haskell library functions other than those mentioned explicitly in this project or covered in class.
- The project can be run from within ghci:

  ```
  # change to project directory
  $ cd ~/i571/submit/prj3-sol
  ```

  and start ghci:

  ```
  -- start GHC REPL
  $ ghci
  GHCi, version ...

  -- initial load of file
  ghci> :l "Main.hs"
  [1 of 6] Compiling EvalIdExpr       ( EvalIdExpr.hs, interpreted )
  ```

```
...
[6 of 6] Compiling Main             ( Main.hs, interpreted )
Ok, six modules loaded.

-- Running all the tests via the main function should not
-- produce any output as they are currently set up to Skip
ghci> main
ghci>
```

As you develop, you will iterate steps similar to the following:

```
-- reload all code; repeat each time source files are changed
*Main> :r
Ok, six modules loaded.

-- run a specific function for an input chosen by you.
ghci> member 2 [1, 2, 3]
True

-- run all the provided tests for member
ghci> testMember
"******* member"
+++ OK, passed 1 test.
...

-- run all tests after completing project after ensuring that all
-- tests in Main.hs are set to Run.
ghci> main
"******* toSingletonLists"
...
"******* test postfixExpr"
+++ OK, passed 1 test.
...
```

- You can also run all tests from the command-line using **`runghc Main.hs`** while in the project directory. This command will return silently if you run it on the project as distributed since all test suites are set to **`Skip`** but you should see the tests results for exercises as you set their test suites to **`Run`** (or **`Only`**).

Submission

Before submitting your project, ensure that all test suites are set to **`Run`**. Update your README to specify the status of your project and document any known issues.

Submit using a procedure similar to that used in your previous project. Note that unpacking your zip file must create a **`prj3-sol`** directory containing **`Main.hs`** which should be set up to run all the tests.

[The project will be set up on gradescope by early next week].