Project 01

Due: Monday, March 18th 11:59pm

1 Description

In this project you will create an expression calculator. The calculator will prompt the user for an expression in prefix notation and calculate the result of the inputted expression. A history will be kept so that previous results can be used in expressions. You are expected to handle possible errors in the user's input.

2 Details

When ran, the program should immediately prompt the user for an expression. It will evaluate that expression, printing an error message if an error is encountered. A simple generic message "Invalid Expression" is alright.

There should be a history, a simple list of values. Every time the program succeeds in evaluating the expression, the result will be added to the history and printed to the screen. Remember, this is a functional language. The history should be a parameter to your eval loop function. So, adding a value to history should involve consing it with the existing history. When the value is printed to screen, you should include its history id (or index). The id will always be the order it was added to history.

So, the first value will be id 1, the second id 2, and so on. We can use the `(!?)` operator to do this. Using the cons operator to construct the history will result the values being in reverse order of their id. That is, the most recent value will be first in the list. This can be remedied by reversing the list before giving it to `(!?)`.

2.1 Expressions

An expression is a value, binary operation, or a unary operation.

+ – A binary operator that adds together the result of two expressions.

***** – A binary operator that multiplies together the result of two expressions.

/ – A binary operator that divides (integer division) the result of the first expression by the result of the second. Don't forget that the user might try to divide by zero. This is an error.

- – A unary operator that negates the value of an expression. There is no subtraction. So, to subtract we can add to a negative number.

$n – n should be an integer. A value specifying to use the history value corresponding to id n.

any integer – A value representing a literal integer

Expressions are in prefix notation and read from left to right. So, if I wanted to evaluate $2 * \$1 + \$2$, I would right `+*2$1$2`. White space can be used to divide tokens (like to numbers), but other wise is insignificant. In the previous example I could also write `+ *2 $1 $2`. It is an error, if the expression is evaluated, but there are still text remaining. For instance, `+1 2 2` is an error. The second 2 is extraneous and not part of the expression.

2.2 Some Tips

- Don't try to do everything in one function.
- Create functions that evaluates a particular type of expression and yields a two element list. The first element would be the result of the evaluation, and the second the remaining characters in the expression.
- Solve the problem in stages. First think about how you would write this program in a language you are familiar with. Maybe write some psudocode. Translate the algorithm to use the functional way of computation. Then, write the scheme code.
- Test your functions individually.

2.3 Some useful functions

You can search for functions by name or type using hoogle.haskell.org.

- `fromIntegral`
- `isDigit` (Requires you to "import Data.Char")
- `isSpace` (Requires you to "import Data.Char")
- `toLower` (Requires you to "import Data.Char")
- `getLine`
- `print`
- `putStr/putStrLn`
- `read`
- `span`
- `map`

In addition, you may want to "import Data.Maybe" to gain the Maybe type for error handling.

3 What to Turn in

Upload your submission as a zip archive containing the following:

- Source code (.hs file(s))
- Readme (Plain text document)
  - List the files included in the archive and their purpose
  - Explain how to compile and run your project
  - Include any other notes that the TA may need
- Write-up (Microsoft Word or pdf format)
  - How did you approach the project?
  - How did you organize the project? Why?
  - What problems did you encounter?
  - How did you fix them?
  - What did you learn doing this project?
  - If you did not complete some feature of the project, why not?
    - What unsolvable problems did you encounter?
    - How did you try to solve the problems?
    - Where do you think the solution might lay?
      - What would you do to try and solve the problem if you had more time?

4 Grading

The grade for this project will be out of 100, and broken down as follows:

Followed Specifications	50
Following the Functional Paradigm (tail-recursion, no mutable variables, etc.)	20
Correct Output (User interaction, calculations work, etc.)	20
Write-up	10

If you were not able to complete some part of the program discussing the problem and potential solutions in the write-up will reduce the points deducted for it. For example, suppose there is a bug in your code that sometimes allows two customers to approach the same worker, and could not figure out the problem before the due date. You can write 2-3 paragraphs in the write-up to discuss this issue. Identify the error and discuss what you have done to try to fix it/find the problem point, and discuss how you would proceed if you had more time. Overall, inform me and the TA that you know the problem exists and you seriously spend time trying to fix the problem. Normally you may lose 5 points (since it is a rare error) but with the write-up you only lose 2. These points can make a large difference if the problem is affecting a larger portion of the program.