Homework 3

Due Thursday, 2/14/2019
Happy Valentine's Day

Important: For this assignment I would like you to turn in a zipped folder with two files. The first file should be a LaTeX document with the complete solutions to all problems (including code). The second file should have suffix "hs" and contain all of the Haskell programs you are writing for this assignment. All functions should be named exactly as specified in this assignment. We will be testing your code by loading this file and running our code on it. The first line of your Haskell file should be:

```
module Hmwk3 where
```

Putting this line in will allow us to automatically test your program. Leaving it out will result in all of our tests failing, and you receiving a correspondingly low score, making all of us sad! As usual, test your program by loading the "hs" file once you have fully commented your code.

Name your two files `Hmwk3.hs` and `Hmwk3.pdf` (where you should replace *OurNames* by the last names of the people working together on the assignment).

When you are ready to turn in your homework, compress your folder into a zip file. Then go to `https://submit.cs.pomona.edu/2019sp/cs131` and click on the assignment 3 on that page and follow the instructions to submit your work.

Be sure to test your programs one last time before submitting. I've seen students mess up when commenting code in a way that causes everything to break. Code that doesnt compile will get very little credit. We will be using some automatic testing, and code that doesnt compile brings everything to a halt. With a large class we will not have time to go in to manually tweak your code to make it work.

1. (10 points) **Parsing** Please do problem 4.2 from Mitchell, page 83.

   Example 4.2 specifies that multiplication and division have higher precedence than addition and subtraction, and that operators of the same precedence are left associative, e.g., 6 - 2 - 1 is interpreted as being equivalent to (6 - 2) - 1 and *not* 6 - (2 - 1).

2. (10 points) **Custom Haskell Control Structures**

   You will find it extremely helpful to read Sections 1 and 2 (pp. 1 – 16) of Simon Peyton Jones' paper, "Tackling the Awkward Squad: monadic input/output, concurrency, exceptions, and foreign-language calls in Haskell" before tackling this problem. You will find a link to it next to the Lecture 6 notes.

   One of the claimed advantages of first-class functions is the ability to write custom control structures. This question will explore this by asking you to write some familiar control structures.

   Below, we provide code for the `whileIO` and `ifIO` control structures implemented in the IO monad. We also provide an example of its usage which prints the integers between 0 and 3 inclusive.

```
import Control.Monad.ST
import Data.IORef

ifIO :: IO Bool -> IO a -> IO a -> IO a
ifIO b tv fv = do { bv <- b;
```

```
                             if bv then tv else fv}

        whileIO :: IO Bool -> IO () -> IO ()
        whileIO b m = ifIO b
                      (do {m; whileIO b m})
                      (return ())

        whileTest = do {v <- newIORef 0;
                        whileIO (do{ x <- readIORef v;
                                     return (x<4)})
                                (do{ x<-readIORef v;
                                     print x;
                                     writeIORef v (1+x) })}
```

IORef is in the standard Haskell library and supports mutable variables in the IO monad. It is described at http://www.haskell.org/ghc/docs/7.6.2/html/libraries/base/Data-IORef.html.

`untilIO` is a very similar control structure to `whileIO`, except the loop condition test 1) executes after the loop body and 2) causes loop exit if true instead of false, as is the case for `whileIO`. Please implement `untilIO` with the provided type signature and also create `untilTest` to print the integers between 0 and 3 inclusive.

```
untilIO:: IO () -> IO Bool -> IO ()
```

For this problem, please turn in the code given here along with the functions you write so that the TA's will be able to grade it without cutting and pasting code.

3. (10 points) **Parsing Tuples**

   Given the following BNF:

```
        <exp> ::= ( <tuple> ) | a
        <tuple> ::= <tuple>, <exp> | <exp>
```

   (a) Draw the parse tree for ((a,a),a,(a)).

   (b) Write a lexer for terms of this form. The tokens are simply "a", "(", ")", and ",". The tokens generated by the lexer should be from the following type:

```
        data Tokens = AToken | LParen | RParen | Comma | Error String |
                      EOF deriving (Eq,Show)
```

   where `EOF` marks the end of the token list. The main lexer function should be of the form:

```
        getTokens :: [Char] -> [Tokens]
```

   As an example, you should get the following results when testing `getTokens`:

```
*Main> getTokens "((a,a),a,(a,a))"
[LParen,LParen,AToken,Comma,AToken,RParen,Comma,AToken,Comma,
LParen,AToken,Comma,AToken,RParen,RParen,EOF]
```

To be continued in your next homework!