Assignment 2

Due: 11:59PM EST, Nov 30 2017

All questions must either be answered individually, or in a team of at most 3 members.

1. [20 points] Define a function **replic lt** that replicates each element in **lt** into a list. If the element is **k**, the resulting list contains k copies of the same element. Error should be returned if any element is a negative number. **You must define this function using the higher-order function of "map"**

e.g. >replic [2,3,4,7,6]
[[2,2],[3,3,3],[4,4,4,4], [7,7,7,7,7,7,7], [6,6,6,6,6,6]]

2. [25 points] Given the following definition of the propositional formula:

```
data Formula
        = Atom Bool                     -- atomic formula
        | And Formula Formula           -- f /\ f
        | Or Formula Formula            -- f \/ f
        | Implies Formula Formula       -- f -> f
        | Not Formula                   -- not(f)
```

(1) Write a Haskell function **collect_atoms f** that computes all boolean primitives of a propositional formula f.

e.g. >collect_atoms (And (Implies (Atom True) (Atom False)) (Not (Atom False)))
[True, False, False]

(2) Write a Haskell function **eval f** to evaluate term f according to standard definitions of propositional logic.

e.g. >eval (And (Implies (Atom True) (Atom False)) (Not (Atom False)))
False

3. [10 points] Read the following Prolog program:

```
mysterious([],0).
mysterious([X],X).
mysterious([X,Y|Xs], Res) :-
        mysterious(Xs, Res1),
        Res is X + Res1.
```

(a) **[5 points]** Explain what this program does.
(b) **[5 points]** If the query is mysterious([2,3,4,5,6,7,8,9, 10, 11],L), what will be the result?

4. [15 points] Draw the derivation tree for the query reach(a,X), where

```
reach(X,Y) :- edge(X,Y).
reach(X,Y) :- reach(Y,Z), edge(Z,X).
edge(X,Y)  :-  edge1(X,Y)
edge(X,Y)  :-  edge1(Y,X)
edge1(a,b).
edge1(b,c).
```

5. [30 points] Use flex to build a propositional logic evaluator (in the same grammar as in Question 2), which reads in-fix logic expressions from standard input and writes the computed result on standard output. Here are some requirements:

1. The calculator terminates when control-D is pressed
2. Constants are True and False
3. Boolean connectives are `/\` (for conjunction), `\/` (for disjunction), `->` (for implication), and `not` (for negation)
4. Parentheses are possible in the input. If parentheses are not used, all operators are left associative
5. White spaces, tabs or new lines, are possible
6. If any character other than those listed above is seen in the input, e.g. #, the calculator responds with an error message: "Invalid character: #". Error messages should be sent to stderr. If parentheses are not matched, you also need to print out an error message

Here are some examples:

User input	Calculator output
`True /\ False`	`False`
`True /\ not False`	`True`
`True /\ (False -> True)`	`True`
`(True \/ False)`	`True`

Define the calculator only with flex (Hint: you need to explicitly maintain a stack for computing the arithmetic expressions.)

Submission Instructions:

- Write down the answers to questions 1, 2 in a file named assignment2.hs.
- Write down the answers to questions 5 in a file named assignment2.l.
- Write down all other answers in a **txt**, **doc**, or **pdf** file, and name it assignment2.suffix where suffix is one of the above.
- Write a README file (text file, do not submit a .doc file) which contains

    - You name(s) and email address(es). PLEASE list your team members if any.
    - Whether your Haskell/flex programs were tested on department machines.
    - Briefly describe anything special about your submission that the TA should take note of.

- 4. Place all files under one directory with a unique name (such as [userid]_2 for assignment 2, e.g. davidL_2).
- 5. Tar the contents of this directory using the following command.
  tar –cvf [directory_name].tar [directory_name]
  e.g. tar -cvf davidL_2.tar davidL_2/
- 6. Upload your tared file on mycourses.

Each team only needs to submit one copy. It does not matter which member on the team submits.