Mid-term examination #1 — given Tuesday 1st October

General instructions

Closed-book, closed-notes, closed-computer, in-class exam.

Time allowed: 75 minutes.

Total points available: 150 pts.

Answer in the spaces provided.

Your name (print):

I pledge my honor that in the preparation of this assignment I have complied with the University of New Mexico Board of Regents' Policy Manual.

Please sign and date:

1.1 Haskell—infinite lists (30 pts)

1. (15 pts) Define a Haskell value `evens :: [Int]` that represents the infinite list of all positive even numbers, in ascending order starting from 2.

2. (15 pts) Explain how Haskell is able to compute with infinite lists without looping forever.

   Use the example of computing the sum of the first 100 positive even numbers in your explanation.

1.2 Haskell—list functionals (30 pts)

1. (10 pts) Write down the end result of evaluating the following Haskell expression:

```
foldr (\x -> \acc -> x:x:acc) [9] [1,2,3]
```

2. (20 pts) Use `map` and `filter` to implement a function `oddsSquared :: [Int] -> [Int]` that returns the list of the squares of the odd numbers from an input list.

   For example, `oddsSquared [2,3,4,5]` should return `[9,25]`.

   Note that you **must** use both `map` and `filter` in your answer, and one of these **must** be the outermost function call.

1.3 Haskell—datatype declarations (50 pts)

Here is the definition of a Haskell type of binary trees with exactly two sub-trees per node that extends the version discussed in class by also storing data values in the leaves as well as in the nodes:

```
data ExtTree a = ELeaf a
               | ENode a (ExtTree a) (ExtTree a)
```

where the first argument to `Node` is the data value stored there and the second and third arguments to `Node` represent the "left" and "right" sub-trees of that node, respectively.

1. (15 pts) Define an `extTreeMap` function for the `ExtTree a` datatype, with the type signature

   ```
   extTreeMap :: (a -> b) -> ExtTree a -> ExtTree b
   ```

   such that `extTreeMap f t` applies the function `f` to *all* data values stored in the tree, both in the `ELeafs` and in the `ENodes`.

2. (15 pts) Define a function `flatten :: (ExtTree a) -> [a]` that returns a list containing *all* data values stored in the tree, both in the `ELeafs` and in the `ENodes`.

   They should be ordered with the values from the left subtree first, then the value stored in the current `ENode`, then the values from the right subtree.

   For example, `flatten (ENode 1 (ENode 2 (ELeaf 3) (ELeaf 4)) (ELeaf 5))` should return `[3,2,4,1,5]`.

3. (20 pts) Define an `extTreeFold` function to fold over trees of the `ExtTree a` datatype **and** give its most general type signature.

1.4 Proving properties of programs (40 pts)

Consider a function `length :: [a] -> Int` that returns the number of elements in a list, i.e.:

```
length [] = 0
length (x:xs) = 1 + length xs
```

Now prove, by induction on lists, that

```
length (map f l) = length l
```

for all `l :: [a]` and for all `f :: a -> b`.