University of Calgary
Department of Computer Science

Introduction to Programming Paradigms (CPSC 449)
Assignment 2 of 4

This assignment is to be completed individually and you will only receive credit for your own original work.

Your submission will be evaluated with respect to style and efficiency, as well as effectiveness.

Reproducing or adapting any published or unpublished material, regardless of the source, without including a proper reference to the source is considered academic misconduct and may result in very severe penalties.

Three-valued logic is a type of many-valued logic that is defined for three distinct truth values - True, False, and Unknown. This system is of particular interest in the field of database administration as it facilitates logical calculations between the standard (i.e., binary) truth values and the NULL value that is frequently stored in an attribute. For this exercise you are required to define an enumerated algebraic data type (and some support function) to be used to represent three-valued logic. You are strongly advised to do some very basic research on three-valued logic before attempting this exercise.

1. Define an enumerated algebraic data type to reflect the different possible values. I strongly recommend that you do not attempt to overload keywords like true and false; instead you should use troo, falz, and unknown (or something to that effect).

2. Write functions ternaryNOT, ternaryAND, and ternaryOR, defined for your enumerated type. Use the minimum number of patterns required - do not replicate entire truth tables or you will be penalized.

3. Now assume instead that the three truth values are being represented using the Maybe type (specifically as Maybe Bools). Write the functions ternaryNOT', ternaryAND', and ternaryOR' using this type for the arguments and the result. Consider the following type definition for ternaryNOT':

   ```
   ternaryNOT' :: (Maybe Bool) -> (Maybe Bool)
   ```

   Please note that it must be possible to "nest" these operations into more complex expressions. For instance, expressions such as `ternaryAND' (ternaryNOT' x) (ternaryNOT' y)` should be considered valid and should return a correct result.

---

4. Write a recursive function (i.e., without list comprehensions) that counts the number of even numbers in a list of integers. (Use value `mod` 2 to test parity.) Then rewrite the function such that it uses a list comprehension instead (i.e., the second version must not be a recursive function).

5. Using the [(Client, Video)] database example from the notes CPSC449-S15-05-(Advanced List Processing).pdf, write a function that takes one such database as its first argument and a string as a second argument. This function should return the a list of the rented videos that contain the second argument in their titles. You must write your own substring testing function (presumably with the definition [Char] -> [Char] -> Bool) to check and see if each video title contains the substring passed as the second argument. As a clarifying example:

   ```
   Main> listVideos dataBase "in"
   ["The Thing","The Thing","Big Trouble in Little China"]
   ```