universität innsbruck

Functional Programming	WS 2023/2024	LVA 703025
Exercise Sheet 9, 10 points		Deadline: Tuesday, December 12, 2023, 8pm

- Mark your completed exercises in the OLAT course of the PS.
- You can start from `template_09.hs` provided on the proseminar page.
- Upload your `*.hs` files in OLAT. (Upload each file separately, and do not use zip, etc.)
- Your `*.hs` files must be compilable with `ghci`.

Exercise 1 *Scope of Variable/Function Names* **4 p.**

The following exercises are about the scope of variables and functions.

1. In the Haskell program below, analyze the scope of `radius` in the three functions `operationA`, `operationB`, and `operationC`. Moreover, state which `radius` (global or local) each function refers to and justify your answers. (1 point)

```haskell
radius :: Double
radius = 10  -- global radius

computeVolume :: Double -> Double
computeVolume rad = (4/3)*pi*rad^3

operationA :: Double -> Double
operationA radius = computeVolume radius

operationB :: Double
operationB  = computeVolume radius

operationC :: Double -> Double
operationC = computeVolume
```

2. Analyze the implementation of `reverseList` in the program below. Does it work as expected? Perform the same variable renaming as in the slides from week 9. (1 point)

```haskell
reverseList :: [a] -> [a]
reverseList xs =
    let reverseListAux xs ys = case xs of
            (x:xs) -> reverseListAux xs (x:ys)
            _ -> ys
    in reverseListAux xs []
```

3. Given the following program:

```haskell
squareRootTwo :: Double -> Integer -> Double
squareRootTwo guess n
    | n == 0 = guess
    | otherwise = squareRootTwo ((guess + 2/guess) / 2) (n-1)

squareRootTwoA :: Double -> Integer -> Double
squareRootTwoA guess n
    | n == 0 = guess
    | otherwise = squareRootTwoA ((guess + 2/guess) / 2) (n-1) where n=n

squareRootTwoB :: Double -> Integer -> Double
squareRootTwoB guess n
    | n == 0 = guess
    | otherwise = let n = n-1 in squareRootTwoB ((guess + 2/guess) / 2) n
```

(a) Consider the funtion `squareRootTwo` above which approximates $\sqrt{2}$ based on an initial guess for n iterations. Do `squareRootTwoA` and `squareRootTwoB` work as expected? Justify your answers. (1 point)

(b) Is it considered good practice to have global and local variables/functions of the same name? (1 point)

Exercise 2 *Modules and Property-Based Testing with LeanCheck* **6 p.**

The easiest way to install additional packages for Haskell is the Haskell Tool Stack, called `stack` on the command line. If `stack` is not installed on your system, then please do so.[1] If you installed GHC via `ghcup`, then this is possible by invoking `ghcup install stack`.

1. First work through the `LeanCheck` README[2] and then its tutorial[3] so that you are able to use the package and answer basic questions about it. (2 points)

2. Install the `LeanCheck` package for *property-based testing* via (0.5 points)

   ```
   $ stack install leancheck
   ```

   Make sure that the package is actually available by starting GHCi via

   ```
   $ stack ghci
   ```

   and then entering

   ```
   ghci> import Test.LeanCheck
   ghci> :t holds
   holds :: Testable a => Int -> a -> Bool
   ```

3. Define a module `Tree` that exports the type `Tree` (and its constructors) from Sheet 05 and also the functions `fillXs` and `splitAtLevel`. (0.5 points)

4. Write a `Listable` instance for `Tree a`. (1 point)

5. Use `LeanCheck`'s `check` function to test whether the following property holds:

   > For arbitrary integers `i`, trees `t` and `s`, and lists of trees `ss`, we have that whenever `splitAtLevel i t == (s, ss)`, then also `fillXs s ss == (t, ss)`. (2 points)

   Hint: You can do this by following these steps:

   (a) Import `LeanCheck` and your module `Tree`.

   (b) Insert the `Listable` instance for `Tree a` from above.

   (c) Implement a function

   ```haskell
   prop_splitAtLevel_implies_fillXs ::
     Int -> Tree Int -> Tree Int -> [Tree Int] -> Bool
   ```

[1] https://docs.haskellstack.org/en/stable/install_and_upgrade/
[2] https://github.com/rudymatela/leancheck/blob/master/README.md
[3] https://github.com/rudymatela/leancheck/blob/master/doc/tutorial.md

that encodes the property from above. Note that `LeanCheck` provides the notation `==>` for logical implication. That is, `x ==> y` means "*whenever* `x`*, then also* `y`".

(d) Use `LeanCheck`'s `check` function to test your property.