universität innsbruck

Functional Programming | WS 2020 | LVA 703025

Exercise Sheet 4, 10 points | Deadline: Wednesday, November 4, 2020, 6am

- Please write all the Haskell code into a single .hs-file and upload it in OLAT.
- You can use the template .hs-file that is provided on the proseminar page.
- Your .hs-file should be compilable with ghci.
- Don't forget to mark your completed exercises in OLAT.

Exercise 4.1 *Divide-and-Conquer* — 3 p.

The *Greatest Common Divider* (GCD) of two non-negative integers is the largest integer that divides both numbers.

For example, the GCD of 12 and 8 is 4 and the GCD of 25 and 20 is 5. As a special case, the GCD of 0 and 0 is defined as 0.

Euclid's algorithm is an efficient algorithm for computing the GCD of two integers. The algorithms is based on the principle that the GCD of two numbers does not change if the larger number is replaced by its difference with the smaller number. This results in two equations:

$$\gcd(a,b) = a \qquad \text{if } a = b \text{ or } b = 0 \tag{1}$$
$$\gcd(a,b) = \gcd(a-b,b) \qquad \text{if } a > b > 0 \tag{2}$$

1. Write a function `gcd :: Integer -> Integer -> Integer` that recursively computes the GCD of two non-negative integers using Equations (1) and (2). (1 point)
2. The `gcd` implementation of the previous question is not very efficient.

   Euclid's algorithm uses the *modulo* function instead of subtraction.

   $$\gcd(a,b) = \gcd(\operatorname{mod}(a,b),b) \qquad \text{if } b \neq 0 \tag{3}$$

   Implement a second function `gcd_eff :: Integer -> Integer -> Integer` that computes the GCD recursively using *Euclid's algorithm.*

   (Hint: think carefully about when to stop the computation; you sometimes need to swap the two arguments of the GCD.) (1 point)
3. Remember that the *modulo* of two integers a and b can be written as:

   $$\operatorname{mod}(a,b) = a - \operatorname{div}(a,b) \cdot b,$$

   where $\operatorname{div}(a,b)$ is the integer division of a and b.

   Use this equation to argue why it is allowed to replace the subtraction of Equation 2 by the modulo-operation in Equation 3. (1 point)

Note: Haskell already ships its own version of `gcd`. To test your program hide these by adding `import Prelude hiding (gcd)` at the top of your program (as seen in the template .hs-file).

Exercise 4.2 *Divide-and-Conquer* **4 p.**

The *Least Common Multiple* (LCM) of two positive integers is the smallest common multiple or positive integer that is divisible by both numbers.

For example, the LCM of 4 and 10 is 20 and the LCM of 5 and 3 is 15. Check for yourself that this is true.

In this exercise you will implement three algorithms for finding the LCM of two numbers.

1. One method for finding the LCM is by calculating the multiples of each input number until a common multiple is found. For 4 and 10 we get:

   Multiples of 4: $4, 8, 12, 16, \mathbf{20}$

   Multiples of 10: $10, \mathbf{20}$

   Write a function `lcm :: Integer -> Integer -> Integer` that computes the LCM of inputs `a` and `b` using the above described method. Write and use an auxiliary function `lcm_aux :: Integer -> Integer -> Integer -> Integer -> Integer` that recursively calculates the LCM of `a`, `b`. `lcm_aux` takes *four* arguments: `ma`, `mb`, `a`, `b`, where `ma`, `mb` are the multiples of `a` and `b` respectively.

   (Hint: in each recursion step only calculate the next multiple of the currently smallest multiple, see example above.) (1 point)

2. Instead of computing the multiple of each input until the LCM is found we can compute the multiple of one of the two inputs until it is divisible by both inputs. For 5 and 3 we get:

   First multiple $1 * 5 = 5$ is not divisible by 3

   Second multiple $2 * 5 = 10$ is not divisible by 3

   Third multiple $3 * 5 = 15$ **is divisible** by 3

   Write a function `lcm2 :: Integer -> Integer -> Integer` that computes the LCM of inputs `a` and `b` using the above described method. Write and use an auxiliary function `lcm2_aux :: Integer -> Integer -> Integer -> Integer` that recursively calculates the LCM of `a`, `b`. `lcm2_aux` takes *three* arguments: `m`, `a`, `b`, where `m` is the current multiple.

   The choice of input to calculate the multiple of (`a` or `b`) can have a huge impact on the performance. Make sure your implementation always uses the most efficient one. (1 point)

3. A third method for computing the LCM is by using the GCD. Write a third function `lcm3 :: Integer -> Integer -> Integer` that uses one of the GCD implementations of exercise 4.1.

   If you were not able to finish exercise 4.1 you can use the default `gcd` implementation provided by Prelude, make sure to remove the line `import Prelude hiding (gcd) from your script`. (1 point)

4. Which of the three LCM implementations (`lcm`, `lcm2` or `lcm3`) do you prefer?

   In GHCi you can display statistics about time and memory usage of an expression by setting `:set +s` (you only have to execute this once). All subsequent evaluations will now display the statistics.

   (Hint: execute each of your LCM implementations with sufficiently large inputs to see a clear difference.) (1 point)

Note: Haskell already ships its own version of `lcm`. To test your program hide these by adding `import Prelude hiding (lcm)` at the top of your program (as seen in the template .hs-file).

Exercise 4.3 *Enumeration* 3 p.

Some of the tallest mountains on the earth are listed below:

Name	Height in m
Everest	8848
K2	8611
Lhotse	8586
Makalu	8481
ChoOyu	8188

1. Define an enumeration type Mountain and implement a function `isHigher :: Mountain -> Mountain -> Bool` by using pattern matching on two arguments and where no comparison of heights is performed during execution. How many equations does your implementation require roughly if there are n mountains (instead of just 5)? (2 points)

2. Define a function mapping a mountain to its height in meters `altitude :: Mountain -> Integer` and use this function to implement the function `isHigher :: Mountain -> Mountain -> Bool` in a better way. How many equations does your implementation require now for n mountains?

   (1 point)