Lab Sheet 7

User-defined Types

STS Software Technology Systems

Functional Programming - Winter 2022/2023 - December 07, 2022 - Schupp/Lübke

How to succeed with the labs and exercises?

Labs and exercise sheets are published every week on the course homepage at StudIP. As described in the first lecture, each successfully completed lab and exercise earns you bonus points towards your final score in this semester's exam. Keep in mind that you only get bonus points if you would pass the exam without them. **Cheating does not help you - but we will!**

How to complete a lab successfully?

In the lab, you will solve the tasks on lab sheet. You are encouraged to talk to your neighbors and find solutions together, as well to ask the tutor for help. **Towards the end of the session, the tutor will briefly discuss your solutions with you. To pass the lab, you should complete two thirds of the tasks (rounding half up)**.

How to complete an exercise successfully?

In order to complete an exercise sheet successfully, you must upload your answers using INGInious **before the deadline** printed on the exercise sheet. We will not consider any solutions handed in after the deadline! Furthermore, you must solve and hand in the exercises **individually** and your Haskell code **must compile** and **pass certain amounts of tests** as specified. During the exercise session, we develop possible solutions together. Please participate! We encourage you to ask and answer questions from fellow students.

Technically, Haskell files you submit using INGInious must have the format as specified in the task sheets (usually ".hs", ".lhs", or ".txt"). Furthermore, INGInious will only consider your last submission. Therefore, if you first submit successfully (your code compiles and tests are passed) and afterwards unsuccessfully (your code does not compile or certain tests fail again), your last submission counts, and - if it does not compile - will therefore be ignored. Make sure your last submission was successful!

How to get additional information?

We encourage you to discuss past and present exercise sheets with us. Either approach us during the exercise session, or visit us during the weekly office hours. We are also available via e-mail or on the StudIP forum. We try to reply as quick as possible and in general, you should get a reply the next weekday, but we cannot guarantee this.

Task 1 Try to load the following Haskell code in GHCi.
You can find it in the (enclosed) file "Errors.hs".

```
type Event a = Lecture | Lab a | exercise deriving Show

type Professor = String
newtype Title = Title String
newtype Course a = Course Professor Title [Event a] deriving Show

type PathToFile = String
data EventDescription = Lab {
     topic :: String
    ,assistants :: [String]
    } deriving Show

isLab :: Event a -> Bool
isLab (Lab _ ) = True

giveMeAllLabs :: Course EventDescription -> [Event EventDescription]
giveMeAllLabs (Course es) = filter isLab es

labs = giveMeAllLabs (Course "Schupp" (Title "Functional␣Programming") [Lecture])
```

Fix all errors. Test that `labs` returns the expected result.
Hint: The keyword deriving instructs the Haskell compiler to automatically make a user-defined type an instance of some type class(es) by employing a default implementation (if a suitable one exists). This is only possible, if all subtypes of the user-defined type are also instances of of the given type class(es). For example, writing

```
data Foo = Foo Int Bool deriving Eq
```

is fine, because Int and Bool are themselves instances of Eq. In contrast

```
data A = A1 Int | A2 Float
data B = B A String deriving Eq
```

does not compile, because `A` is not an instance of Eq You'll learn more about this in the next lecture.

Task 2 Represent the following fictitious course description as a value of type `Course EventDescription` from Task 1.

Esoteric Programming Languages

In this Lecture, Prof. Brain and assistant Pinky explain programming languages of the future. The course consists of at least a lecture and an exercise. Furthermore, Pinky offers labs on Brainfuck[1] and Taxi[2]

Task 3 Consider the type definitions of Task 1. Write a function that takes a value of type `Course EventDescription` and returns the list of all lab topics. Test your implementation with your solution from Task 2.

Task 4 Follow the design recipe for data definitions from the lecture and design a type according to the following specification.

[1] https://esolangs.org/wiki/Brainfuck
[2] https://esolangs.org/wiki/Taxi

The type represents either an error, parameterized by an error id and an error description, or a non erroneous result of some kind. The error id must be an integral number and the description must be a string.

Task 5 * Consider your type definition of an error in Task 4. Write a function `bindE` that takes a value `v` of the type you defined and a function `f`. `bindE` applies `f` to the wrapped value in `v` if it represents a non erroneous value. Otherwise it returns the original error. Explain how this function would be used.

* This is an advanced, optional task, but we strongly advise you to give it a try!