Lab Sheet 3

Pattern Matching, Type Classes, and Currying

STS Software Technology Systems

Functional Programming - Winter 2022/2023 - November 09, 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 Recall the cons operator `(:)`, which takes an element `x` and a list `xs` and returns a list which is `xs` with `x` inserted at the front. Also recall that a list uses the cons operator internally: `[1,2,3]` is nothing else than `1:(2:(3:[]))`. What are the results of the following Haskell expressions?
Note: The `let pattern = input in output` matches the expression `input` against `pattern` and returns the value of `output`.
Hint: Not all expressions are well-formed lists.

a) `let [x] = 1:[] in x`

b) `let [x] = "1"in x`

c) `let [x] = 1 in x`

d) `let (x:xs) = [1..10] in (x, xs)`

e) `let (x:(y:ys)) = [[1,2],[3,4,5]] in (x, y, ys)`

f) `let (x:xs) = [1,2]:[3,3,3,3] in x`

g) `let [x, 2] = [2..3] in x`

Task 2 Write the following function definitions to a file (or use the enclosed *Errors.hs*).

```
myHead :: [a] -> a
myHead x:xs = x

myData :: [a]
myData = read "[1,2,3]"

apply_f_twice :: Int -> Int -> Int -> Int
apply_f_twice f x = f (f x)
```

Load the file using GHCi (`:load`), look at the error messages, understand the problem and fix all errors. Repeat until the file loads with no errors.
Hint: Try to fix the functions one after another.

Task 3 Rewrite the following function definitions using pattern matching

a) `myAnd` Computes the result of the logical and ($\wedge$) of the two Boolean variables.

```
myAnd :: Bool -> Bool -> Bool
myAnd a b = a && b
```

b) `multComplex` multiplies two complex numbers of the form $c = a + ib$. Remember that $c_1c_2 = (a_1a_2 - b_1b_2) + i(a_1b_2 + b_1a_2)$. Each complex number $a + ib$ is represented by the tuple `(a,b)`.

Hint: Your implementation should not use `fst` and `snd` anymore.

```
multComplex :: (Floating a) => (a,a) -> (a,a) -> (a,a)
multComplex x y = (fst x * fst y - snd x * snd y, fst x * snd y + snd x * fst y)
```

Task 4 Given the following descriptions of functions, provide reasonable type signatures (contracts).
Hint: Use GHCi's `:i` to get information about type classes.

- `isCapital`: Determine whether the specified letter is a capital letter.
- `average`: Return the mean value from a given list.
- `getMax`: Return the greatest element of a given list.
- `det`: Returns the determinant of a given square matrix.

Task 5 * This definition of `double` relies on currying. Explain how it works.

```
double :: (Num n) => n -> n
double = mult 2
  where mult x y = x * y
```

Task 6 * The Haskell Prelude provides a function flip, which takes a two-argument function `f` and returns another two-argument function taking its argument in the opposite order. For example,

$$\texttt{(flip (-)) 1 2} \equiv \texttt{(-) 2 1} \equiv \texttt{2 - 1} \equiv \texttt{1}.$$

Hint: The expression `(-)` is the subtraction operator passed as a function.

The description above can be turned directly into a function in Haskell, like the below implementation `verboseFlip`:

```
verboseFlip :: (a -> b -> c) -> (b -> a -> c)
verboseFlip f = g
  where g x y = f y x
```

Alternatively, one can implement the same function using currying, as in the following implementation of `curryFlip`:

```
curryFlip :: (a -> b -> c) -> b -> a -> c
curryFlip f x y = f y x
```

Your task is to explain why the two implementations are equivalent.

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