Exercise Sheet 2

Design Recipes and Testing

STS Software Technology Systems

Functional Programming - Winter 2022/2023 - November 08, 2021 - 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.

DEADLINE: 08:00, November 14, 2022

2022-11-14T08:00:00+01:00

In this exercise sheet, you practice designing by recipe as described in the lecture. You also write (and run) simple tests with QuickCheck. In the accompanying file `Ex02.hs`, you find a template where you can fill out the missing parts (indicated using *[TODO]*). Think about properties of your functions that are reasonable for testing. We encourage you to implement tests with random parameters, but for this exercise, unless otherwise stated, it is enough to implement the examples as tests.

⚠ Submission - Important Note ⚠

For your submission, consider the following rules:

- Do NOT rename any of the provided source files (e.g., `Ex02.hs`) to ensure that the source files keep the same name as the modules defined inside them (e.g., `module Ex02 where`)
- Do NOT change the left-hand sides of any given function definition (e.g., in `add a b = undefined`, do not change the `add a b` part) There may be exceptions to this rule, but in any case:
  - The name of the function must remain unchanged
  - The number of arguments of the function must remain unchanged
  - The types of arguments must remain unchanged
- Do NOT change the (80 character long) lines between individual functions or any of the design recipe keywords (e.g., `-- CONTRACT`)
- Define your function examples as follows:
  `example␣[function]␣[some number] = [function] [arg1] [arg2] .... == [result]`
  Example: `example␣add␣1 = add 2 3 == 5`
  → your examples must be code, not comments
- Define your function tests as follows:
  `prop␣[function]␣[prop description] [arg1] [arg2] ... = .....`
  Example: `prop␣add␣reference a b = (add a b) == (a + b)`
- Zip all your submission files together into a `.zip`-archive named `Ex02␣[FirstName]␣[LastName].zip`
  Example: `Ex02␣Ole␣Luebke.zip`

Task 1 Write a complete design recipe for `euclDistance` that computes the Euclidean distance between two points. You need at least two examples, and three tests. The Euclidean distance between two points (x_0, y_0), (x_1, y_1) is calculated as $\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}$

Task 2 The two-argument function `iff` returns True exactly when both Boolean operands have the same value. Otherwise it returns False. Construct a design recipe for the `iff` function and implement it using:

a) Conditional expressions (`iffC`).

b) Guarded equations (`iffG`).

c) Pattern matching (`iffP`).

d) Simple logical connectives (`&&`, `||`, `not`) (`iffB`)

In the definition section(s) of the design recipe, you are not allowed to use `==` or `/=`.
Implement one non-trivial test property for `iff` that tests whether the result of `iff a b` is the same as for `a == b`.

Task 3 The function `tail` from the Prelude produces an error if applied to an empty list:

```
Prelude> tail []
*** Exception: Prelude.tail: empty list
```

Provide a function `myTail` that behaves like `Tail` except that `myTail` returns no error for the empty list. Think about a useful return value. Follow the design recipe and implement `myTail` using:

a) Conditional expressions (`myTailCond`).

b) Guarded equations (`myTailGuard`).

c) Pattern matching (`myTailMatch`).

Hint: Use the library functions `null` and `tail`.

Task 4 Read through the documentation and implementation of `foo` and `bar` in the accompanying file `Ex02.hs`. If you do not know what e.g. `toUpper` does, ask hoogle[1]. Try them out in GHCi. What would be better names? Rename both functions and complete the design recipes.

Task 5 An overloaded function `f` takes a polymorphic argument and returns a value of the same polymorphic type. In its definition, `f` uses the following functions and operators:

- `+`
- `<`

Provide the most general type (contract) for `f`.
Hint: If given an operator, function, or type class name, GHCi's `:i` command provides helpful information.

Upload your edited "Ex02.hs" file as a `zip` file using INGInious (*https://inginious.sts.tuhh.de*[2,3]). If your name contains any non-ASCII[4] characters (e.g., ä, ö, ü, ß), please replace

[1] http://www.haskell.org/hoogle/
[2] Only reachable from TUHH network or via VPN
[3] There are issues with Firefox, please use a Chromium-based Browser (e.g., Chromium, Google Chrome, Microsoft Edge)
[4] https://en.wikipedia.org/wiki/ASCII

them with the appropriate sequence of ASCII characters (ae, oe, ue, ss). The zip file should be named as follows:

`Ex02_[FirstName]_[LastName].zip`
(Example: Ex02_Ole_Luebke.zip)

DO NOT rename the files or any of the predefined functions or arguments. For INGInious to accept your submission as correct, it must at least pass the tests for:

- **Task 1**,
- *3* out of *4* sub-tasks of **Task 2**,
- *2* out of *3* sub-tasks of **Task 3**

If any tests failed for a specific sub-task, you can get further information about a counterexample for these tests in the test results section on INGInious. Additionally, you also have to comply with the stated requirements for the uniqueness, file types, and naming conventions.

Please remember, that you are not guaranteed to get the bonus point when INGInious judges your submission as correct. We also perform manual checks (e.g., for free text answers, plagiarism, ...) that determine the final result. You will be informed about the outcome via e-mail.