Philipps  Universität Marburg

Deadline: 03.02.2014

Submit either at the exercise session on 03.02.2014 or in box 137 outside the toilets of Kern D.

Anonymous questionnaire about exercises of the lecture "Programming languages and types"

- DO NOT write your name or student number (Matrikelnummer) on this page.
- DO NOT staple this page together with other pages of your homework submission.

The questionnaire is for statistics only and is in no way connected to the evaluation of your homework submission. By filling out this questionnaire, you declare to have answered as truthfully as your knowledge permits.

1. *Outside* the exercise session, how many hours did you spend doing the homework?

   _____ hours.

2. In your opinion, how many hours would it take an average student to do the homework?

   _____ hours.

3. How many lecture sessions did you attend this week?

   ☐ None. ☐ One session. ☐ Two sessions.

4. Did you attend the exercise session on 27.01.2014?

   ☐ Yes. ☐ No.

5. Please cross *all* statements that are true about you.

   ☐ I understand records and variants.
   ☐ I understand subtyping.
   ☐ I can explain the reason behind subtying rules.
   ☐ I know how to hide runtime errors in Java code.

Philipps Universität Marburg

Deadline: 03.02.2014

Submit either at the exercise session on 03.02.2014 or in box 137 outside the toilets of Kern D.

Matrikelnummer:	
Name:	

Homework 12 of the lecture "Programming languages and types"

Solutions are available one week after deadline at
`https://www.uni-marburg.de/fb12/ps/teaching/ws13/plt-solutions/`

1. This is an example of using records and variants in programming:

   `https://github.com/klauso/PLT2013/blob/master/exercises/ex12.ml`

   This is a summary of German taxation classes:

   `http://steuerklassen.biz/`

   Please write an OCaml program that puts each person in the appropriate taxation class. You do not have to make the program compile; it is more important to convey an understanding of the computation process.

   Here is the interface of the function. It also appears online at

   `https://github.com/klauso/PLT2013/blob/master/exercises/ex12.ml#L196`

   Please mail the code to `pllecture@mathematik.uni-marburg.de`.

```
type geld = float

type ehegatte = { einkommen : geld }

type familienstand =
  [ `ledig | `verheiratet of ehegatte | `geschieden | `verwitwet ]

type person =
  { familienstand : familienstand ; kinder : int ; lohn : geld }

let steuerklasse : person -> int
  = fun person -> ... (* please complete the function definition. *)
```

2. The Liskov Substitution Principle is an important paradigm of object-oriented design. You can read more about it here:

   `http://de.wikipedia.org/wiki/Liskovsches_Substitutionsprinzip`

   In March 1996, Robert C. Martin rephrased the principle as follows.[1]

   > Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it.

   The subtyping rules in `25-subtyping_and_systemf.pdf` has a similar purpose. If $S <: T$, then we should be able to freely substitute terms of type S into places expecting type T without causing runtime type errors.

   Let's consider an example from `ex12.ml`.

```
type mensch         = { familienstand : familienstand }
type familienmensch = { familienstand : familienstand ; kinder : int }
```

   A value `michel : mensch` can only be used by projecting `michel.familienstand`. Since any value `wilhelm` of type `familienmensch` has the field `familienstand` of the same type, it will always be safe to use `wilhelm` as if it has the type `mensch`. Thus we are justified to call `familienmensch` a subtype of `mensch`. This is the motivation behind the rule S-RcdWidth, which says that a record type S is a subtype of T if S has all the fields of T and possibly more. (Sadly, OCaml does not implement S-RcdWidth.)

$$\{l_i : T_i \ ^{i\in 1..n+k}\} \quad <: \quad \{l_i : T_i \ ^{i\in 1..n}\} \qquad \text{(S-RcdWidth)}$$

   As a second example, consider the imaginary variants

```
type MFS = (* moderner Familienstand *)
  [ `ledig | `verheiratet of ehegatte | `geschieden | `verwitwet ]

type TFS = (* traditioneller Familienstand *)
  [ `ledig | `verheiratet of ehegatte | `verwitwet ]
```

   The only way to use a value `m : MFS` is to pattern-match on it. Any program pattern-matching on `m` has to provide for all 4 situations of singlehood, marriage, divorce and widowhood.

```
match m with
    `ledig             -> ...
  | `verheiratet gatte -> ...
  | `geschieden        -> ...
  | `verwitwet         -> ...
```

   Now a value `t : TFS` describes the situation of a traditional family, when divorce wasn't invented yet. If a program receives `t` instead of `m`, then it knows already how to handle the 3 situations possible with `t`—in fact, it knows more than that, and should run without any type error. Thus `TFS` is a subtype of `MFS`. The rule S-VariantWidth generalizes the preceding discussion, and declares a variant type U to be the subtype of V whenever V has all cases of U and possibly more. (OCaml uses square brackets `[]` instead of angle brackets `<>` for variants.)

$$< l_i : T_i \ ^{i\in 1..n} > \quad <: \quad < l_i : T_i \ ^{i\in 1..n+k} > \qquad \text{(S-VariantWidth)}$$

[1] `http://www.objectmentor.com/resources/publishedArticles.html`

(a) Explain the reasoning behind the subtype relation

$$(\texttt{mensch} \to \texttt{int}) \quad <: \quad (\texttt{familienmensch} \to \texttt{int}).$$

Which rule in `25-subtyping_and_systemf.pdf` is about this situation?
(Hint: how does a program make use of a function?)

(b) Let the variant types `MFS` and `TFS` about marital status (Familienstand) be defined as on page 2. Consider the following record types.

```
type moderner       = { familienstand : MFS }
type traditioneller = { familienstand : TFS }
```

Is `moderner` a subtype of `traditioneller`, or is `traditioneller` a subtype of `moderner`? Justify your answer.

3. (Optional) Slide 25 of `25-subtyping_and_systemf.pdf` describes a faulty subtyping rule in Java. Written in Java syntax, the rule is:

$$\frac{S \mathrel{<:} T}{S\texttt{[]} \mathrel{<:} T\texttt{[]}} \qquad \text{(S-ArrayJava)}$$

For example, if `Employee` is a derived class of `Person`, then `javac` would accept all programs using an array `Employee[]` as if it were an array `Person[]`. Write a Java program to exploit S-ArrayJava. The program should compile without problems, but should produce a runtime error when executed.

Hints:

i. One possible program has fewer than 10 lines.

ii. One possible runtime error is called `ArrayStoreException`.