Functional Programming
Exercises Week 9

(for December 13, 2013)

Numbers in parentheses refer to the 6th edition of the course notes.
Exercises marked with ⋆ are optional and can be ignored.

1. Read Chapter 8 of the lecture notes.

2. (Exercise 8.1) Write a parser

```
uibk_mail : (Strng.t * Strng.t, char)Parser.t
```

that accepts an email address as used for students at the university of Innsbruck, i.e.,

$$l^{+}\texttt{.}l^{+}\texttt{@student.uibk.ac.at}$$

where l is a letter. The result should be the forename and the surname as a pair, e.g.,

```
# test uibk_mail "christian.sternagel@student.uibk.ac.at";;
```

should give the result `(("christian", "sternagel"),"")`. The functions of `Parser` may (and should) be used freely.

3. (Exercise 8.4) Implement a parser `int : (int, char)Parser.t` for (decimal) integers where an integer is given by the (simplified) grammar

$$\begin{aligned} i &::= n \mid \texttt{+}n \mid \texttt{-}n \\ n &::= d^{+} \\ d &::= \texttt{0} \mid \ldots \mid \texttt{9} \end{aligned}$$

4. (Exercise 8.6) Write a parser `words : (Strng.t list, char)Parser.t` that accepts arbitrarily many sentences and returns all the words that are contained as a list of l-strings. Here a sentence is a sequence of words (i.e., lowercase or uppercase letters) that are separated by white spaces and/or commas and terminated by a full stop (`.`), question mark (`?`), or exclamation mark (`!`).

5. (Exercise 8.8) Write a parser `tag : ((Strng.t * Strng.t), char) t` that accepts a simplified version of XML tags. For this purpose let a tag be of the form

$$\texttt{<}\textit{tagname}\texttt{>}\textit{content}\texttt{</}\textit{tagname}\texttt{>},$$

where *tagname* is an arbitrary (non empty) sequence of letters, *content* is an arbitrary sequence of characters except ‘`<`’, and the first and second occurrence of *tagname* have to be identical. The result of the parser should be a pair of l-strings, where the first is the name of the tag and the second its content. E.g., `<a>bla</a>` should be accepted with result `([’a’], [’b’; ’l’; ’a’])`, whereas `<a>bla</A>` and `<a>bla</b>` should both fail.

Hint: You need not consider nested XML tags.

⋆. Consider the following grammar for propositional formulas:

$$\phi ::= p \mid (!\ \phi) \mid (\phi\ \&\ \phi)$$

a) Write a `lexer` for this grammar such that e.g.

```
# Parser.test lexer "(a & (!a) )";;
- : token list * char list =
([LPAR; ID "a"; AND; LPAR; NOT; ID "a"; RPAR; RPAR], "")
```

and

```
# Parser.test lexer "(a a)";;
- : token list * char list =
([LPAR; ID "a"; ID "ab"; RPAR], "")
```

b) Write a `parser` for this grammar such that e.g.

```
# Parser.parse parser [LPAR; ID "a"; AND; LPAR; NOT; ID "a"; RPAR; RPAR];;
- : t option = Some (And (Atom "a", Not (Atom "a")))
```

and

```
# Parser.parse parser [LPAR; ID "a"; AND; LPAR; NOT; ID "a"; RPAR; RPAR];;
- : t option = Some (And (Atom "a", Not (Atom "a")))
```

c) Give a non-left recursive grammar such that ! binds stronger than &.