Übungsblatt 4

Vorwort

Wir haben uns in der vergangenen Vorlesung Parser näher angeschaut. In den praktischen Übungen soll es weniger um das "erfinden" eines neuen Parser-Kombinators gehen, als um die Anwendung.

Set-Up

Da wir nun zum ersten mal mit externen dependencies arbeiten, müssen wir diese zunächst installieren. Dies ist ein sehr wichtiger Schritt, da nahezu alle weiteren Übungszettel dieses Vorgehen voraussetzen.

Als erstes sollten Sie die Quellen von `stack` mit einem `stack update` aktualisieren. Anschließend erstellen sie ein neues Stack-Projekt mittels `stack new <Projektname>`. Sie erhalten ein Verzeichnis mit dem Projektnamen, in dem ein kleines "Hello World" liegt. Als erstes sollte man immer über die `projektname.cabal` drüber schauen. Hier werden nachher auch alle dependencies eingetragen.

Bei mir sieht das z.B. so aus (Projekt heisst "parser"):

```
name:                parser
version:             0.1.0.0
synopsis:            Initial project template from stack
description:         Please see README.md
homepage:            https://github.com/githubuser/parser#readme
license:             BSD3
license-file:        LICENSE
author:              Author name here
maintainer:          example@example.com
copyright:           2016 Author name here
category:            Web
build-type:          Simple
-- extra-source-files:
cabal-version:       >=1.10

library
  hs-source-dirs:      src
  exposed-modules:     Lib
  build-depends:       base >= 4.7 && < 5
  default-language:    Haskell2010

executable parser-exe
```

```
  hs-source-dirs:      app
  main-is:             Main.hs
  ghc-options:         -threaded -rtsopts -with-rtsopts=-N
  build-depends:       base
                     , parser
  default-language:    Haskell2010

test-suite parser-test
  type:                exitcode-stdio-1.0
  hs-source-dirs:      test
  main-is:             Spec.hs
  build-depends:       base
                     , parser
  ghc-options:         -threaded -rtsopts -with-rtsopts=-N
  default-language:    Haskell2010

source-repository head
  type:     git
  location: https://github.com/githubuser/parser
```

Generell sollte man hier ordentliche Angaben machen, da viele tools dieses automatisch lesen und überall eintragen. Insbesondere sollte man nicht seine private Email-Adresse nehmen, sondern z.b. die Techfak-Adresse, die Studiumsbezogen ist.

Wir haben hier 3 Bereiche: `library`, `parsers-exe` und `test-suite`. Library ist das, was wir meistens schreiben - eine Sammlung von Funktionen, die das eigentlich tun. Dann haben wir die executable; diese enthält die `main` und ruft meist in wenig Code unsere library auf, nachdem sie z.b. Parameter/Dateien/... gelesen hat und kümmert sich um die Ausgabe. Somit können wir für spätere Zwecke (Projekte) die Library 1:1 wiederverwenden. Die test-suite ignorieren wir für den Moment. Wir kommen in einer separaten Vorlesung noch einmal auf Tests zu sprechen.

Eine editierte Variante könnte etwa so aussehen:

```
name:                parser
version:             0.1.0.0
synopsis:            A little parser for generic CSV-Files
description:         Please see README.md
homepage:            https://github.com/Drezil/FFPiHaskell_parser#readme
license:             BSD3
license-file:        LICENSE
author:              Stefan Dresselhaus
maintainer:          sdressel@techfak.uni-bielefeld.de
copyright:           2016 Stefan Dresselhaus
```

```
category:            Tool
build-type:          Simple
-- extra-source-files:
cabal-version:       >=1.10

library
  hs-source-dirs:      src
  exposed-modules:     Lib
  build-depends:       base >= 4.7 && < 5
                     , attoparsec
  default-language:    Haskell2010

executable parser-exe
  hs-source-dirs:      app
  main-is:             Main.hs
  ghc-options:         -threaded -rtsopts -with-rtsopts=-N
  build-depends:       base
                     , parser
  default-language:    Haskell2010

test-suite parser-test
  type:                exitcode-stdio-1.0
  hs-source-dirs:      test
  main-is:             Spec.hs
  build-depends:       base
                     , parser
  ghc-options:         -threaded -rtsopts -with-rtsopts=-N
  default-language:    Haskell2010

source-repository head
  type:     git
  location: https://github.com/Drezil/FFPiHaskell_parser
```

Änderungen die gemacht wurden:

- Daten ausgefüllt
- attoparsec als dependency der library hinzugefügt
- github-links angepasst (sofern man github verwendet)

Nachdem man das ganze nun gespeichert hat, reicht ein `stack build` um alle dependencies herunterzuladen, kompilieren und installieren. Anschließend kann man mit `stack exec parser-exe` das Programm ausführen.

Ein simpler CSV-Parser

Sie sollten aus ihrem Studium bereits die EBNF kennen. Eine (simple) CSV-Datei besitzt folgende EBNF:

```
csv-file       = { row }
row            = field-list, eol
field-list     = field, [ ",", field-list ]
field          = [ whitespace ], field-value, [ whitespace ]
field-value    = quoted-string | bare-string
quoted-string  = '"', quoted-content, '"'
quoted-content = { quoted-char }
quoted-char    = (any char except '"' or eol)
bare-string    = { bare-char }
bare-char      = (any char except ',' or eol without whitespace at beginning/end)
whitespace     = space-char, { space-char }
space-char     = " " | "\t"
eol            = "\n"
```

Kurzes recap: `{ .. }` bedeutet 1 oder mehr, `[ .. ]` sind optional, `A | B` heißt, entweder A oder B, `A, B, C` bedeutet zunächst A, dann B, dann C.

Datenstrukturen

Überlegen sie sich zunächst, wie eine Datenstruktur aussehen könnte und definieren sie diese. Inhalt sind vorerst nur Strings. Sie brauchen keine Zahlen/Daten/... zu erkennen.

Parser

Schreiben sie einen Parser, der einen CSV-String in diese Datenstruktur parsed und geben sie diese aus (`deriving Show` auf der Datenstruktur reicht). Ein paar Testbeispiele für CSV-Dateien finden sie auf github/im Lernraum.

Bonus

Natürlich ist das nur ein simpler CSV-Parser. Folgende Features wären für einen echten Einsatz noch Wünschenswert:

- sicherstellen, dass alle "rows" gleich lang sind und ggf. mit Fehlermeldung abbrechen
- einen "Header" mit einlesen, der die einzelnen Spalten beschreibt
- Quotation nicht nur als "blabla'bla", sondern auch als 'blabla"bla' zulassen,"bla " bla" auch entsprechend parsen.