## Compiler

# Blatt 6 (Abgabe)

Prof. Dr. Oliver Braun

Fakultät für Informatik und Mathematik
Hochschule München

Letzte Änderung: 21.06.2017 15:31

Diese Aufgabe bildet die Grundlage für den Schein. Sie können diese Aufgabe alleine oder in einer Zweiergruppe bearbeiten. Um den Schein zu bekommen, müssen Sie die Lösung bis zum Praktikumstermin am 26.06.17 bzw. 27.06.17 in das von mir zur Verfügung gestellte GitHub-Repository gepusht haben. Am jeweiligen Praktikumstermin zeigen und erläutern Sie mir dann Ihre Lösung.

Nutzen Sie für Fragen zur Aufgabe die Praktikumstermine und/oder Issues in Ihrem GitHub-Repository, die Sie mir zuweisen.

Ihr Projekt wird mit jedem Push automatisch auf meinem Jenkins gebaut und getestet.

## Repository und Gruppeneinteilung

Für das Repository und die Gruppeneinteilung folgen Sie dem Classroom-Link. Im ersten Schritt erzeugen Sie ein neues Team oder treten Sie einem bestehenden Team bei. Maximale Teamgröße ist 2. Wenn Sie ein Team neu erzeugt haben, wird Ihnen außerdem ein neues Repository erzeugt. Wenn Sie dem Team beitreten, bekommen Sie Zugriffsrechte auf das bereits existierende Repository.

Damit ich Ihnen Ihren GitHub-Account auch zuordnen kann, erzeugen Sie als **erstes** eine Datei mit dem Namen `stud.yaml` und ihren Daten, in der Form:

```yaml
- name: Miriam Musterfrau
  email: mimu@hm.edu
  github: KleineHexe16
- name: Max Mustermann
  email: mamu@hm.edu
  github: KleinerMann17
```

**Achtung:**

- Verwenden Sie genau die E-Mail-Adresse, die von Ihnen im ZPA2 steht, also Ihre `@hm.edu`-Adresse. Anderenfalls kann ich Sie nicht zuordnen und Sie können auch keinen Schein bekommen.
- Die Datei wird von mir automatisiert verarbeitet, daher ist es **zwingend** notwendig, dass Sie die Struktur wie angegeben einhalten. Wenn Sie alleine arbeiten, tragen Sie sich trotzdem mit einem Spiegelstrich eingerückt ein.

## Der Markdown-To-HTML-Compiler

Initial befindet sich im Repository ein erster Ansatz für einen Markdown-to-HTML-Compiler an dem Sie weiter arbeiten sollen. Der Compiler kann schon ein bisschen was, macht manches ganz okay und anderes nicht ganz optimal. Zum Teil muss die Implementierung geändert werden, wenn Sie weitere Features hinzufügen.

Sie können den Compiler, wie gewohnt, mit `stack build` bauen und folgendermaßen nutzen:

```
md2html

Usage: md2html COMMAND [-o|--output OUTFILE] INFILE
  Markdown-To-HTML-Compiler

Available options:
  -o,--output OUTFILE      Ausgabe in Datei umleiten
  INFILE                   Quell-Datei im Markdown-Format
  -h,--help                Show this help text

Available commands:
  tokens                   gescannte Tokens ausgeben
  ast                      durch den Parser erzeugten AST ausgeben
  html                     HTML erzeugen
  dot                      AST in dot-Syntax ausgeben
```

Der Compiler besteht aus einem Frontend (Scanner und Parser) und zwei Backends (Generierung von HTML, Generierung von einer Dot-Repräsentation)

Entwickeln Sie sowohl das Frontend als auch **beide** Backends gemäß den folgenden Anforderungen weiter.

### Anforderungen

Das Frontend muss eine Teilmenge von Markdown akzeptieren. Als Grundlage dient die CommonMark Spezifikation in der Version 0.27 (2016-11-18). Hilfreich um das Verhalten

einzelner Markierungen zu verstehen oder größere Blöcke auszuprobieren ist der Dingus.

**Achtung:** Ein Problem von Markdown ist, dass sich die vorhandenen Markdown-Implementierungen (Editoren, Generatoren, …) nicht einheitlich verhalten. Ihre Implementierung muss sich so verhalten wie die o.g. Spezifikation dies vorschreibt. Z.B. reicht bei manchen Editoren schon ein Einrücken um ein Zeichen für den nächsten Listenlevel. Nach der Spezifikation reicht das nicht.

Die Zwischenrepräsenation soll, wie schon im ersten Ansatz, weiter als Baum- bzw. Graphstruktur sehr nahe am zu generierenden HTML sein. Sie finden die aktuelle Implementierung als `AST`-Datentyp im Modul `Types`.

Das HTML-Backend muss das passende HTML so generieren, dass es von `http://validator.w3.org/` akzeptiert wird. Das Dot-Backend muss eine Zeichenkette generieren, die von `dot`, z.B. unter `http://www.webgraphviz.com/`, akzeptiert wird.

Der fertige Compiler muss mindestens mit folgenden Teilen der Spezifikation umgehen können (Sie müssen nicht alle möglichen Sonderfälle und Ausnahmen implementieren, die angegegeben Beispiele zeigen das gewünschte Verhalten):

- ATX headings, Beispiele 32, 33, 34, 37, 38, 39
- Indented Code Blocks, Beispiele 76, 77, 78, 80, 81
- Paragraphs, Beispiele 180 - 187
- Blank Lines
- Code Span, Beispiele 312, 316
- Emphasis and Strong Emphasis, nur mit `*`, Beispiele 328, 329, 332, 333, 355, 356, 358, 370, 371, 373
- Links, Beispiele 456, 457, 458, 470,
- Images, Beispiele 541, 547, 550
- Hard Line Breaks, Beispiele 603, 607, 608, 609,
- List Items, Bullet List Marker nur `-`, Ordered List Marker nur Zahlen gefolgt von einem `.`, Beispiele 217, 225, 226, 230, 243, 244, 255, 256
- Lists, insbesondere müssen sich Listen verschachteln lassen können, Beispiele 264, 267, 268, 271, 272
- Link Reference Definitions, Beispiele 166, 167, 170

Tipp: Es bietet sich an mindestens für alle genannten Beispiele Tests zu schreiben.