Dr. Annette Bieniusa
Mathias Weber, M. Sc.
Peter Zeller, M. Sc.

**TU Kaiserslautern**
**Fachbereich Informatik**
**AG Softwaretechnik**

# Übungsblatt 2: Software-Entwicklung 1 (WS 2017/18)

Ausgabe: Montag, 30.10.17
Abgabe: Montag, 06.11.17

## Hinweise zu Einreichaufgaben

Bitte beachten Sie folgende Hinweise für dieses und die weiteren Übungsblätter:

1. Einzureichende Abgaben sind Montags vor 12:00 Uhr Mittags über das Exclaim System abzugeben.
2. Sie können sich im Exclaim System unter https://softech.cs.uni-kl.de/exclaim mit Ihrem Stats-Account einloggen und Dateien zu den einzelnen Übungen hochladen.
3. Sie können Grafiken auch von Hand anfertigen und dann eine gescannte oder abfotografierte Kopie hochladen.
4. Laden Sie nur Dateien in den folgenden Formaten hoch: PDF, JPG, PNG, Textdatei (UTF-8). Verwenden Sie auf keinen Fall Formate, welche spezielle Programme benötigen (**kein** Microsoft Word, **kein** Open-/LibreOffice o.ä., **kein** Pages).
5. Wenn Sie Java-Programme einreichen, dann müssen komplette Java-Dateien hochgeladen werden, die vom Java-Übersetzer akzeptiert werden. Bitte laden Sie einzelne Dateien hoch, keine Archive, keine kompilierten Dateien. Programme, die nicht kompilieren, werden nicht korrigiert und mit **0 Punkten** bewertet.
6. Vermeiden Sie den Einsatz integrierter Entwicklungsumgebungen wie beispielsweise Eclipse. Verwenden Sie einen einfachen Editor, wie den in der Vorlesung demonstrierten und auf dem ersten Übungsblatt eingeführten Editor Visual Studio Code. In der Klausur stehen Ihnen die Hilfsmittel von IDEs schließlich auch nicht zur Verfügung.

## Hinweise zu den Übungen

Wegen der Feiertage finden am Dienstag und Mittwoch (31. Oktober und 1. November) keine Übungen und keine Vorlesung statt. Am Donnerstag und Freitag stehen die Tutoren zu den normalen Terminen von Gruppe 4-9 zur Verfügung. Nutzen Sie diese Zeit, falls Sie Probleme mit den Aufgaben von diesem Blatt oder denen von Blatt 1 haben. Sie müssen sich in dieser Woche nicht an die Termine halten, für die Sie registriert sind, sondern können zu beliebigen Terminen kommen.

Erst ab der folgenden Woche (ab 7. November) fängt dann der normale Übungsbetrieb an und es gilt Anwesenheitspflicht. In dieser Woche werden Sie auch einen Termin mit Ihrem Tutor für die Abnahmen ausmachen, welche in der darauf folgenden Woche (ab 13. November) beginnen.

## Aufgabe 1 Kontextfreie Grammatiken (9 Punkte)

Die formale Sprache L ist durch die Grammatik $\Gamma = (N, T, \Pi, U)$ mit

$$
\begin{aligned}
N &= \{S, U, V\} \\
T &= \{a, b, c, d\} \\
\Pi &= \left\{ \begin{array}{lcl} S & \rightarrow & c \\ U & \rightarrow & VU \\ U & \rightarrow & Sb \\ V & \rightarrow & d \\ V & \rightarrow & Ua \end{array} \right\}
\end{aligned}
$$

definiert.

a) Welches ist der kürzeste Satz der Sprache L? Geben Sie alle Sätze der Sprache (mit Ableitung) an, die bis zu fünf Buchstaben besitzen.

b) Überprüfen Sie, ob Ihre Ableitungen aus Aufgabenteil a *Linksableitungen* sind. Falls nicht, geben Sie zu den entsprechenden Sätzen auch eine Linksableitung an. Können Sie für einen der Sätze auch eine andere Linksableitung angeben?

c) Zeichnen Sie einen Syntax-Baum[1] für den Satz "`cbacbacb`".

d) Ist die Grammatik $\Gamma$ eindeutig? Wenn nicht, geben Sie ein Gegenbeispiel an.

## Aufgabe 2 Schreiben von Grammatiken (7 Punkte)

a) Geben Sie eine kontextfreie Grammatik für E-Mail Adressen an. Dabei sollen folgende Punkte beachtet werden:

   1. Die Adresse enthält genau einmal das Zeichen '@'.
   2. Die restlichen Teile der Adresse können folgende Zeichen enthalten: Buchstaben (a-z), Zahlen, Bindestriche, Unterstriche und Punkte.
   3. Es dürfen keine zwei Punkte aufeinander folgen, es darf kein Punkt direkt vor oder direkt nach dem '@' Zeichen kommen und es darf kein Punkt am Anfang oder Ende der Adresse stehen.
   4. Hinter dem '@' Zeichen muss mindestens ein Punkt vorkommen.
   5. Vor dem '@' muss mindestens ein Zeichen stehen.

b) *(Zusatzaufgabe)* Sind alle oben genannten Punkte korrekt, das heißt gelten sie für alle gültigen E-Mail Adressen?

## Aufgabe 3 Parser (8 Punkte)

In dieser Aufgabe werden Sie ein Java-Programm schreiben, welches erkennt, ob eine bestimmte Eingabe ein gültiges Femto-Programm ist. Solch ein Programm, welches eine formale Sprache einliest, wird Parser genannt.

Sie werden vermutlich erst im Laufe des Semesters verstehen, wie dieses Programm funktioniert. In dieser Aufgabe geht es lediglich darum, die vorgegeben Regeln zu befolgen, um die Femto-Grammatik in ein entsprechendes Java-Programm zu übersetzen.

[1] Syntaxbäume werden erst in der Vorlesung am Donnerstag behandelt.

Wir betrachten hier die folgende Grammatik für Femto:

$\Gamma = (N, T, \Pi, \texttt{Programm})$

$N = \{\texttt{Programm}, \texttt{VereinbarungsListe}, \texttt{WertVereinbarung}, \texttt{Ausdruck}, \texttt{Operator}\}$

$T = \{\texttt{print}, \texttt{semikolon}, \texttt{typeInt}, \texttt{bezeichner}, \texttt{gleich}, \texttt{zahl}, \texttt{klammerAuf}, \texttt{klammerZu}, \texttt{plus}, \texttt{mult}\}$

$$\Pi = \left\{ \begin{array}{rcl} \texttt{Programm} & \rightarrow & \texttt{VereinbarungsListe print Ausdruck semikolon} \\ \texttt{VereinbarungsListe} & \rightarrow & \texttt{WertVereinbarung VereinbarungsListe} \\ & | & \epsilon \\ \texttt{WertVereinbarung} & \rightarrow & \texttt{typeInt bezeichner gleich Ausdruck semikolon} \\ \texttt{Ausdruck} & \rightarrow & \texttt{zahl} \\ & | & \texttt{bezeichner} \\ & | & \texttt{klammerAuf Ausdruck Operator Ausdruck klammerZu} \\ \texttt{Operator} & \rightarrow & \texttt{plus} \\ & | & \texttt{mult} \end{array} \right\}$$

a) Laden Sie sich die Vorlagen `Parser.java` und `FemtoParser.java` herunter und speichern Sie die Dateien im gleichen Ordner. Öffnen Sie ein Terminal, wechseln Sie mit `cd` zu diesem Ordner und übersetzen Sie die Datei `Parser.java` mit dem Befehl:

```
javac FemtoParser.java
```

Es sollte kein Fehler auftreten und eine Datei `FemtoParser.class` im gleichen Ordner erstellt werden. Sie können diesen Befehl nach jedem der folgenden Schritte verwenden, um sicherzustellen, dass Sie ein gültiges Java Programm geschrieben haben.

b) Vervollständigen Sie die Datei nach der Anleitung, welche Sie am Ende des Übungsblattes finden. Ihr Code soll in der Datei `FemtoParser.java` nach der Zeile "`public class FemtoParser extends Parser {`" und vor der Zeile mit "`public static void main(String[] args) {`" stehen.

Gehen Sie dann wie folgt vor und überprüfen Sie nach jedem Schritt, ob das Übersetzen mit `javac` Fehler findet. Beheben Sie die Fehler, bevor Sie mit dem nächsten Schritt fortfahren. Wenden Sie sich bei Problemen an einen Tutor.

1. Schreiben Sie den Code für das Nichtterminalsymbol `Operator`. Dieses Nichtterminalsymbol hat zwei alternative Produktionen, welche jeweils aus einem einzelnen Terminalsymbol auf der rechten Seite bestehen.
2. Schreiben Sie den Code für das Nichtterminalsymbol `Ausdruck`. Hier gibt es drei alternative Produktionen. Die unterste Alternative besteht aus einer Sequenz von Terminal- und Nichtterminalsymbolen.
3. Schreiben Sie den Code für das Nichtterminalsymbol `WertVereinbarung`. Hier gibt es nur eine Produktion.
4. Schreiben Sie den Code für das Nichtterminalsymbol `VereinbarungsListe`.
5. Schreiben Sie den Code für das Nichtterminalsymbol `Programm`. Ein Teil des Codes hierfür ist bereits in der Vorlage enthalten, ersetzen Sie diesen Teil entsprechend durch den vollständigen Code.

c) Übersetzen Sie das fertige Programm mit dem Befehl:

```
javac FemtoParser.java
```

Laden Sie dann die Datei `parser_input1.txt` herunter, und speichern Sie die Datei im gleichen Ordner wie `FemtoParser.java`. Testen Sie dann Ihr Programm mit der Datei `parser_input1.txt` als Eingabe, indem Sie den folgenden Befehl ausführen:

```
java FemtoParser < parser_input1.txt
```

Wenn Sie alles richtig gemacht haben, sollte "Eingabe akzeptiert." ausgegeben werden. Testen Sie Ihr Programm entsprechend auch mit den anderen bereitgestellten Beispielen. Bei den Beispielen 4-6 sollte Ihr Programm einen Fehler in der Eingabe erkennen und anzeigen.

d) Laden Sie die Datei `FemtoParser.java` im Exclaim System hoch. Verwenden Sie danach den entsprechenden Button um Ihre Abgabe vom System testen zu lassen.

## Parser Anleitung

Bei der Programmierung mit Java ist es wichtig, dass Sie sich an Groß- und Kleinschreibung halten. Zusätzliche Leerzeichen und Zeilenumbrüche haben keine Bedeutung, sollten aber verwendet werden um den Code übersichtlich zu halten.

Der Parser, den wir in dieser Aufgabe schreiben, liest die Eingabe von links nach rechts ein. Dabei kann der Parser zwei Aktionen durchführen:

- Er kann ein bestimmtes Terminalsymbol `t` mit der Aktion `accept(t)` akzeptieren. Wenn das aktuelle Terminalsymbol in der Eingabe ungleich `t` ist, gibt der Parser einen Fehler aus und wird beendet. Ansonsten wird mit dem nächsten Terminalsymbol fortgefahren.
- Der Parser kann mit `nextIs(t)` prüfen, ob das nächste Terminalsymbol `t` ist.

Die Vorlage für diese Aufgabe enthält bereits Java-Code für diese Aktionen. Im Folgenden wird erklärt, wie aus diesen Aktionen und Java-Sprachkonstrukten der Parser programmiert werden kann.

**Nichtterminale:** Für jedes Nichtterminal gibt es ein oder mehrere Produktionen in der Grammatik. Sei $A$ ein Nichtterminalsymbol und die Produktionen für $A$ wie folgt definiert:

$$A \to \alpha_1 \mid \ldots \mid \alpha_n$$

Hierbei ist $\alpha_i$ jeweils aus $(N \cup T)^*$.

**Code für rechte Seiten von Produktionen:** Der Parser-Code für ein $\alpha_i$ ergibt sich aus den entsprechenden Nichtterminal- und Terminalsymbolen in $\alpha_i$. Für ein Terminalsymbol $b$ verwenden wir die Aktion "`accept(b);`" um das Terminalsymbol zu akzeptieren. Für ein Nichtterminalsymbol $B$ verwenden wir den Code "`B();`". Der Code für die einzelnen Symbole wird hintereinander geschrieben.

Für $\alpha_1 = c\ S\ d$ mit $c, d \in T$ und $S \in N$ ergibt sich also beispielsweise der Code:

```
accept(c);
S();
accept(d);
```

Im folgenden wird der Code für die rechte Seite einer Produktion $A \to \alpha$ mit rechts($\alpha$) bezeichnet.

**Bedingungen für Alternativen:** Um alternative Produktionen zu unterscheiden, muss der Parser prüfen, welche Alternative er wählen soll. Dazu soll die Aktion `nextIs` verwendet werden.

Als erstes muss dazu festgestellt werden, welche Alternative mit welchen Terminalsymbolen anfangen kann.

Falls $\alpha_i = a \ldots$ mit $a \in T$, dann ist klar, dass $\alpha_i$ mit dem Terminalsymbol $a$ anfängt.

Falls ein Nichtterminal am Anfang steht, also $\alpha_i = A \ldots$ mit $A \in N$, dann kann $\alpha_i$ mit einem der Terminalsymbole anfangen, mit denen die Alternativen Produktionen von $A$ anfangen können.

Die Spezialfälle $\alpha_i = \epsilon$ und $A \Rightarrow^* \epsilon$ betrachten wir hier nicht, da sie für das Beispiel nicht relevant sind.

Falls eine Alternative mit Terminalsymbol $a$ anfangen kann, ergibt sich daraus die Bedingung "`nextIs(a)`". Wenn es mehrere mögliche Terminalsymbole am Anfang gibt, werden die einzelnen Bedingungen durch "`||`" voneinander getrennt. Sind beispielsweise die Terminalsymbole $a$ und $b$ möglich, ergibt sich der Code `nextIs(a) || nextIs(b)`.

Im folgenden wird der Code für die Bedingung einer Produktion $A \to \alpha$ mit `bedingung(`$\alpha$`)` bezeichnet.

**Code für Nichtterminale:** Wir betrachten wieder ein Nichtterminalsymbol $A$ mit den folgenden Produktionen:

$$A \to \alpha_1 \mid \ldots \mid \alpha_n$$

Falls es nur eine Produktion für $A$ gibt (also $n = 1$ und $A \to \alpha_1$), dann schreiben wir den folgenden Code:

```
void A() {
  rechts(α1)
}
```

Bei mehreren Alternativen müssen entsprechend die Bedingungen überprüft werden, um die richtige Alternative zu wählen. Für 2 Alternativen sind die Produktionen $A \to \alpha_1 \mid \alpha_2$ und der Code der folgende:

```
void A() {
  if (bedingung(α1)) {
    rechts(α1)
  } else {
    rechts(α2)
  }
}
```

Bei 3 Alternativen sind die Produktionen $A \to \alpha_1 \mid \alpha_2 \mid \alpha_3$ und der Code:

```
void A() {
  if (bedingung(α1)) {
    rechts(α1)
  } else if (bedingung(α2)) {
    rechts(α2)
  } else {
    rechts(α3)
  }
}
```

Bei mehr als drei Alternativen kann nach dem gleichen Muster fortgefahren werden.

## Beispiel:

Für die Grammatik $\Gamma = (N, T, \Pi, S)$ mit

$$\begin{aligned} N &= \{S, X, Y\} \\ T &= \{a, b, c, d\} \\ \Pi &= \left\{ \begin{array}{lcl} S & \to & X \\ & \mid & c\ S\ d \\ X & \to & a\ b\ a\ a \\ & \mid & Y\ a \\ Y & \to & b\ b \end{array} \right\} \end{aligned}$$

ergibt sich der folgende Code:

```
void S() {
  if (nextIs(a) || nextIs(b)) {
    X();
  } else {
    accept(c);
    S();
    accept(d);
  }
}

void X() {
  if (nextIs(a)) {
    accept(a);
    accept(b);
    accept(a);
    accept(a);
  } else {
    Y();
    accept(a);
  }
}

void Y() {
  accept(b);
  accept(b);
}
```