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

TU Kaiserslautern
Fachbereich Informatik
AG Softwaretechnik

Lösungshinweise/-vorschläge zum Übungsblatt 3: Software-Entwicklung 1 (WS 2017/18)

Die Hinweise und Vorschläge in diesem Dokument sollen der Lösungsfindung dienen und erheben demnach weder Anspruch auf Vollständigkeit noch Korrektheit. Sollten Sie Fehler finden würden wir uns freuen wenn Sie uns diese mitteilen. (Kontaktdaten finden Sie auf unserer Webpräsenz.)

Aufgabe 1 Sprachen und Grammatiken (12 Punkte)

In dieser Aufgabe verwenden wir eine Mengennotation für Sprachen. ...

a) Geben Sie jeweils eine Grammatik für die folgenden Sprachen an: ...

$$\Gamma_0 = (\{S,T\},\{a,b\},\left\{\begin{array}{lcl} S & \rightarrow & aS \mid aT \\ T & \rightarrow & bT \mid bb \end{array}\right\},S)$$

$$\Gamma_1 = (\{S,T\},\{a,b\},\left\{\begin{array}{lcl} S & \rightarrow & aSb \mid aTb \\ T & \rightarrow & bT \mid bb \end{array}\right\},S)$$

$$\Gamma_2 = (\{S\},\{a,b\},\{S \rightarrow abS \mid \epsilon\},S)$$

$$\Gamma_3 = (\{S,T\},\{a,b\},\left\{\begin{array}{lcl} S & \rightarrow & aSaaa \mid T \\ T & \rightarrow & bbT \mid bb \end{array}\right\},S)$$

b) Geben Sie die Sprachen für folgende Grammatiken in Mengen-Schreibweise an: ...

$$L_3 = \{a^n \mid n \geq 1\}$$
$$L_4 = \{a^n b a^n \mid n \geq 0\}$$
$$L_5 = \{(ab)^n aca(ba)^n \mid n \geq 0\}$$

c) Geben Sie zur folgenden Beschreibung eine Grammatik und eine Definition in Mengennotation an:

Jedes Wort der Sprache ist eine mindestens einelementige Folge von a gefolgt von einer möglicherweise leeren Folge von x, die zur linken und rechten durch ein einzelnes l bzw. r begrenzt ist.

Beispiel: Die Worte $alr, aaalxr, alxxxxr$ sind in der Sprache und die Worte $lar, arl, xrl, lxxr, allrr$ sind **nicht** in der Sprache enthalten.

$\Gamma_c = (N,T,P,S)$

$N = \{S,X\}$

$T = \{a,l,x,r\}$

$P = \{S \rightarrow aS \mid alXr,\ X \rightarrow xX \mid \epsilon\}$

Mengenschreibweise:

$L_c = \{a^n l x^m r \mid n \geq 1 \text{ und } m \geq 0\}$

Aufgabe 2 Java Grammatik (10 Punkte)

a) Entscheiden Sie für die folgenden 10 Eingaben, ob sie zu der Sprache gehören, die durch das gegebene Syntax-Diagramm mit Startsymbol "Anweisungen" definiert wird. Dabei steht der Terminalknoten "Zahl" für alle Zahlen bestehend aus Ziffern 0-9 und der Terminalknoten "Bezeichner" für Zeichenreihen aus den Buchstaben a-z, die nicht andersweitig als Terminalknoten verwendet werden. Leerzeichen und Zeilenumbrüche werden ignoriert.

Erklären Sie bei Eingaben, die nicht zur Sprache gehören, was der Fehler ist und wie dieser korrigiert werden könnte.

1.
```
System.out.println(2 == 3);
```

Gehört zur Grammatik und wird vom Java-Übersetzer akzeptiert.

2.
```
x = 42;
```

Gehört zur Grammatik, aber der Java-Compiler beschwert sich, weil x nirgendwo definiert ist.

```
Test.java:4: error: cannot find symbol
      x = 42;
      ^
symbol:   variable x
location: class Test
```

3.
```
int x = 3;
x == 3;
```

Gehört nicht zur Grammatik. Ein Vergleich ist ein Ausdruck, aber kein Statement.

```
Test.java:4: error: not a statement
  x == 3;
    ^
```

4.
```
int i = 40;
int f = 2;
int if = i + f;
```

Gehört nicht zur Sprache, weil "if" ein Schlüsselwort ist, und somit nicht als Bezeichner verwendet werden kann. Der Java-Übersetzer findet hier gleich 6 Fehler. Meistens ist dabei der erste Fehler der entscheidende und die anderen nur Folgefehler.

```
Test.java:6: error: not a statement
          int if = i + f;
          ^
```

5.
```
int x = 1;
if (x = 1)
  x = 2;
else {
  x = 3;
}
```

Gehört nicht zur Sprache, weil `x = 1` kein Ausdruck ist. Wahrscheinlich war `x == 1` gemeint. Der Java-Compiler sieht das etwas anders, für ihn ist eine Zuweisung auch ein Ausdruck, welcher den zugewiesenen Wert als Ergebnis liefert. Allerdings passen die Typen hier nicht; die Bedingung muss vom Typ **`boolean`** sein.

```
Test.java:4: error: incompatible types: int cannot be converted to boolean
  if (x = 1)
        ^
```

6.
```
{int x=1
;}
{boolean x = 1 == 2
;}
```

Diese Anweisungen gehören zur Sprache und werden auch von Java akzeptiert. Alleinstehende Blöcke, ohne **`if`** oder **`while`**, werden aber nur sehr selten verwendet. Außerdem sollte man sein Java-Programm so nicht formatieren.

7.
```
if (x < 0);
    b = 1;
else
    b = 2;
```

Das Semikolon in der ersten Zeile ist eine eigene, leere Anweisung, mit der die If-Anweisung abgeschlossen ist. Das “else” hat somit kein passendes “if” mehr. Entsprechend sieht die Fehlermeldung von javac auch wie folgt aus:

```
Test.java:6: error: 'else' without 'if'
        else
        ^
```

8.
```
{{{;}{{{};}}{};};}
```

Dieses Beispiel ist nicht typisch für Java, da unnötige Klammern und alleinstehende Blöcke verwendet werden. Es gehört aber zu der Sprache und wird auch von javac akzeptiert.

9.
```
int x = 3;
if (x < 3)
  if (x < 10)
    x = 1;
else
  x = 2;
System.out.println(x);
```

Dieses Beispiel gehört zur Sprache und wird vom Java Compiler akzeptiert. Allerdings machen die fehlenden geschweiften Klammern und die falsche Einrückung das Programm schlecht lesbar. Geht man nach den Einrückungen, so wäre wohl die Ausgabe “1” erwartet. Die Ausgabe ist jedoch “3”.

10.
```
int x = 1;
while (x < 100)
    if (x < 90)
      x = x * 2;
    else
      x = x + 1;
System.out.println(x);
```

Dieses Beispiel gehört zur Sprache, und wird von Java akzeptiert. Allerdings sollte man die geschweiften Klammern bei einer While- und If-Anweisung immer setzen, damit das Programm besser lesbar ist.

b) Testen Sie die Anweisungen aus Teil a) in einem Java-Programm. Verwenden Sie hierzu das Programm "Hello.java" von Aufgabenblatt 1 und ersetzen Sie die Anweisung `System.out.println("Hello World!")` durch die Anweisungen aus Teil a). Vergleichen Sie die Fehlermeldungen des Java-Übersetzers mit Ihren Ergebnissen aus Teil a). Für diese Teilaufgabe muss keine Lösung hochgeladen werden.

Siehe Lösung zu a)

c) Wandeln Sie das Syntaxdiagramm in eine kontextfreie Grammatik um, welche die gleiche Sprache beschreibt.

$\Gamma = (N, T, \Pi, \textit{Anweisungen})$, mit:

$$N = \{\textit{Anweisungen, Anweisung, Ausdruck, Operator, Typ, Zuweisung, If-Anweisung, While-Anweisung, Print}\}$$

$$T = \{+, *, <, ==, \{, \}, (,), ;, =, \textit{Bezeichner, Zahl, boolean, int, if, else, while, System.out.println}\}$$

$$\Pi = \left\{ \begin{array}{rcl}
\textit{Anweisungen} & \rightarrow & \epsilon \mid \textit{Anweisung Anweisungen} \\
\textit{Anweisung} & \rightarrow & \textit{Zuweisung} \\
& \mid & \textit{If-Anweisung} \\
& \mid & \textit{While-Anweisung} \\
& \mid & \textit{Print} \\
& \mid & ; \\
& \mid & \{\ \textit{Anweisungen}\ \} \\
\textit{Zuweisung} & \rightarrow & \textit{Typ}\ \textbf{\textit{Bezeichner}} = \textit{Ausdruck}\ ; \\
& \mid & \textbf{\textit{Bezeichner}} = \textit{Ausdruck}\ ; \\
\textit{If-Anweisung} & \rightarrow & \textbf{\textit{if}}\ (\ \textit{Ausdruck}\ )\ \textit{Anweisung}\ \textbf{\textit{else}}\ \textit{Anweisung} \\
& \mid & \textbf{\textit{if}}\ (\ \textit{Ausdruck}\ )\ \textit{Anweisung} \\
\textit{While-Anweisung} & \rightarrow & \textbf{\textit{while}}\ (\ \textit{Ausdruck}\ )\ \textit{Anweisung} \\
\textit{Print} & \rightarrow & \textbf{\textit{System.out.println}}\ (\ \textit{Ausdruck}\ )\ ; \\
\textit{Ausdruck} & \rightarrow & \textbf{Bezeichner} \\
& \mid & \textbf{Zahl} \\
& \mid & \textit{Ausdruck Operator Ausdruck} \\
& \mid & (\ \textit{Ausdruck}\ ) \\
\textit{Operator} & \rightarrow & + \mid * \mid < \mid == \\
\textit{Typ} & \rightarrow & \textbf{boolean} \mid \textbf{int}
\end{array} \right\}$$

Aufgabe 3 Aufsteigend sortierte Zahlen (3 Punkte)

Schreiben Sie ein Java-Programm "IsStronglyMonotonicIncreasing3", welches drei double-Werte x, y, und z nimmt und "true" ausgibt, wenn die drei Zahlen von links nach rechts immer größer werden (also $x < y < z$ gilt).

Beispiel:

```
> java IsStronglyMonotonicIncreasing3 1 2 3
true
> java IsStronglyMonotonicIncreasing3 4.1 4.2 4.2
false
```

Zur Erinnerung: Die Funktion `Double.parseDouble(String s)` *wandelt einen* `String` *in einen* **`double`** *Wert um.*

```java
public class IsStronglyMonotonicIncreasing3 {
    public static void main(String[] args) {
        double x = Double.parseDouble(args[0]);
        double y = Double.parseDouble(args[1]);
        double z = Double.parseDouble(args[2]);
        System.out.println(x < y && y < z);
    }
}
```

Aufgabe 4 Wochentage (5 Punkte)

Schreiben Sie ein Programm "Wochentag", welches den Wochentag für ein bestimmtes Datum ausgibt. Dabei soll das Datum über 3 Programmparameter angegeben werden:

- Parameter 1 ist der Tag des Monats (1-31)
- Parameter 2 ist der Monat (1 für Januar bis 12 für Dezember)
- Parameter 3 ist das Jahr (positive Zahl)

Der entsprechende Wochentag soll ausgeben werden (deutsche Sprache, erster Buchstabe groß geschrieben), zum Beispiel:

```
> java Wochentag 13 11 2017
Montag
> java Wochentag 15 11 2017
Dienstag
```

Verwenden Sie zur Berechnung keine Datums-Funktionen der Standardbibliothek, sondern statt dessen die folgende Formel (Zellers Kongruenz):

$$h = \left(q + \left\lfloor \frac{13(m+1)}{5} \right\rfloor + K + \left\lfloor \frac{K}{4} \right\rfloor + \left\lfloor \frac{J}{4} \right\rfloor - 2J \right) \bmod 7$$

Hierbei ist:

- h ist der Wochentag (0 = Samstag, 1 = Sonntag, 2 = Montag, …, 6 = Freitag)
- q ist der Tag des Monats (1-31)
- m ist der Monat (März bis Dezember wie üblich von 3 bis 12; Januar und Februar werden als 13. und 14. Monat des vorherigen Jahres betrachtet)
- K ist das Jahr im Jahrhundert ($year$ mod 100).
- J ist das Jahrhundert ($\lfloor year/100 \rfloor$).
- Der Ausdruck "x mod y" ist der Rest der Division von x geteilt durch y, entspricht also für positive Zahlen dem Java-Ausdruck `x % y`.

  Hinweis: Um auch negative `x`-Werte korrekt zu behandeln kann statt dessen der Ausdruck `Math.floorMod(x, y)` verwendet werden.
- Der Ausdruck $\left\lfloor \frac{x}{y} \right\rfloor$ ist die Division mit abrunden auf die nächst kleinere ganze Zahl, entspricht also für positive Zahlen dem Java-Ausdruck `x / y`.

```java
public class Wochentag {

    public static void main(String[] args) {
        int tag = Integer.parseInt(args[0]);
        int monat = Integer.parseInt(args[1]);
```

```
        int jahr = Integer.parseInt(args[2]);

        int q = tag;
        int m = monat;
        if (m < 3) {
            m = m + 12;
            jahr = jahr - 1;
        }
        int K = jahr % 100;
        int J = jahr / 100;

        int h = Math.floorMod(q + (13 * (m + 1) / 5) + K + K / 4 + J / 4 + 5 * J, 7)
    ;

        switch (h) {
        case 0:
            System.out.println("Samstag");
            break;
        case 1:
            System.out.println("Sonntag");
            break;
        case 2:
            System.out.println("Montag");
            break;
        case 3:
            System.out.println("Dienstag");
            break;
        case 4:
            System.out.println("Mittwoch");
            break;
        case 5:
            System.out.println("Donnerstag");
            break;
        case 6:
            System.out.println("Freitag");
            break;
        }

    }
}
```

Aufgabe 5 Logarithmus berechnen (4 Punkte)

Schreiben Sie ein Java-Programm "Logarithmus", welches zwei `int` Zahlen x und b als Programm-Parameter nimmt und die größte `int` Zahl n berechnet, so dass $b^n \leq x$ gilt. Geben Sie n auf der Konsole aus. Sie können davon ausgehen, dass $b > 1$ und $x \geq 1$ gilt. Verwenden Sie für diese Aufgabe keine mathematischen Funktionen aus der Java Standard-Bibliothek.

```
public class Logarithmus {
  public static void main(String[] args) {
    int x = Integer.parseInt(args[0]);
    int b = Integer.parseInt(args[1]);

    int n = 0;
    int p = 1;
    while (p <= x) {
      n = n + 1;
      p = p * b;
   }
   System.out.println(n-1);
 }
}
```

Aufgabe 6 Programm-Parameter und Terminierung (6 Punkte)

Für diese Aufgabe ist nur Teil e) abzugeben. Die anderen Teilaufgaben sollten Sie in der Abnahme erklären können.

Gegeben ist das folgende Java-Programm:

```
public class Range {
    public static void main(String[] args) {
        int start = Integer.parseInt(args[0]);
        int end   = Integer.parseInt(args[1]);
        int step  = Integer.parseInt(args[2]);

        int i = start;
        while (i != end) {
            System.out.println(i);
            i = i + step;
        }
    }
}
```

a) Speichern Sie das obige Programm in einer Datei namens “Range.java” und übersetzen Sie es mit dem Java-Übersetzer.

b) Führen Sie das Programm mit dem Befehl `java Range 0 100 5` aus. Was ist die Ausgabe?

Es werden die Zahlen von 0 bis 95 in 5-er Schritten ausgegeben (eine Zahl pro Zeile).

c) Was passiert, wenn Sie andere Argumente an das Programm übergeben? Zum Beispiel nur eine Zahl, vier Zahlen, oder Buchstaben? Wenn Fehler auftreten, wo steht die Zeilen-Nummer, in der der Fehler auftrat?

Wenn man nur zwei Zahlen übergibt, bekommt man einen Fehler (`ArrayIndexOutOfBoundsException`) in Zeile 5, weil das Element mit Index 2 nicht Teil des Arrays ist.

Bei mehr als drei Zahlen, werden die zusätzlichen Zahlen einfach nicht gelesen.

Bei der Eingabe von Buchstaben, bekommt man den Fehler:

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "hello"
    at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
    at java.lang.Integer.parseInt(Integer.java:580)
    at java.lang.Integer.parseInt(Integer.java:615)
    at Plus.main(Plus.java:3)
```

Bei Laufzeit-Fehlern wird ein sogenannter “Stacktrace” unter der Fehlermeldung angezeigt. Die Zeile, die den Fehler verursacht hat, findet sich darin. Im Beispiel oben gibt die letzte Zeile mit “(Plus.java:3)” die Zeile 3 als Fehlerquelle an. Die restlichen Zeilen, die mit “at java....” starten können in der Regel ignoriert werden.

d) Für welche Eingaben terminiert das Programm, wenn man annimmt, dass es keine Überläufe gibt? Begründen Sie jeweils, warum das Programm terminiert. Wie sieht es mit Überläufen aus?

Hinweis: Was ist die Ausgabe von `java Range 0 25 4`?

Wenn Sie ein Programm von der Shell aus starten, dann können Sie es mit der Tastenkombination `Strg+C` abbrechen.

Wenn es keine Überläufe gäbe, dann würde das Programm in folgenden Fällen terminieren:

1. wenn $start == end$ gilt
2. wenn $start < end$ und $step > 0$ und $(end - start)\%step == 0$
3. wenn $start > end$ und $step < 0$ und $(start - end)\%step == 0$

Im Fall $start == end$ wird die Schleife nie betreten und das Programm terminiert.

Im Fall 2 wird der Wert von i immer größer, bis er irgendwann gleich end ist und die Schleife beendet wird. end wird auch genau getroffen, also nicht übersprungen, weil die Differenz von $start$ und end durch $step$ teilbar ist.

Fall 3 ist analog.

(Mit Überläufen kann das Programm auch terminieren, wenn $step$ in die falsche Richtung zeigt, da ab einem gewissen Punkt ein Überlauf stattfindet und man sich so dem Ziel von der anderen Seite nähert. Die Teilbarkeit ist auch nicht immer nötig, aber es ist schwer die genaue Bedingung für die Terminierung dann intuitiv zu beschreiben.)

e) Verbessern Sie das Programm so, dass es immer terminiert und es für alle bereits terminierenden Eingaben im Verhalten unverändert bleibt.

```
public class RangeOK {
    public static void main(String[] args) {
        int start = Integer.parseInt(args[0]);
        int end   = Integer.parseInt(args[1]);
        int step  = Integer.parseInt(args[2]);

        int i = start;
        while ((step > 0 && i < end) ||
               (step < 0 && i > end)) {
            System.out.println(i);
            i = i + step;
        }
    }
}
```

Wichtig zu beachten sind die Fälle für einen negativen Wert für `step` und der Fall `step == 0`.