Praktikum Grundlagen der Programmierung

Prof. Dr. Harald Räcke, J. Kranz, A. Reuss

WS 2017/18

Übungsblatt 2

Abgabefrist: 06.11.2017 um 05:00 Uhr

Σ Punkte: 20

Bitte laden Sie sich vor der Bearbeitung dieses Blattes **die neueste Version** von MiniJava von der Praktikumsseite herunter!

Aufgabe 2.1 (P) Grammatik Integer

Ab Java SE 7 können beliebig viele Unterstriche zwischen Ziffern in numerischen Literalen auftreten. Diese Unterstriche haben keine semantische Bedeutung und dienen lediglich der Lesbarkeit. Beispiel: `int x = 100_000;` wird interpretiert wie `int x = 100000;`.

In der Vorlesung wurde eine Grammatik für ganze Zahlen mit führenden Nullen angegeben. Geben Sie eine Grammatik für ganze Zahlen *ohne* führende Nullen und mit den ab Java SE 7 erlaubten Unterstrichen für numerische Literale an. Beachten Sie, dass die Unterstriche nur zwischen Ziffern, nicht aber am Anfang oder Ende eines numerischen Literals erlaubt sind, siehe `http://docs.oracle.com/javase/7/docs/technotes/guides/language/underscores-literals.html`.

Lösungsvorschlag 2.1

```
1 Number ::= -? PDigit (Underscore Digit)* | 0
2 Underscore ::= _*
3 PDigit ::= 1 | ... | 9
4 Digit ::= 0 | PDigit
```

Aufgabe 2.2 (P) Reguläre Ausdrücke

Geben Sie unter Verwendung der in der Vorlesung eingeführten Operatoren für reguläre Ausdrücke *, ? und | einen regulären Ausdruck für a^+ an, wobei α^+ bedeutet, dass mindestens einmal und beliebig oft hintereinander ein durch den regulären Ausdruck α erzeugter Text auftritt.

Geben Sie reguläre Ausdrücke für die folgenden Textmuster an, wobei `letter` einen beliebigen Buchstaben im Text beschreibt:

1. Alle Texte, die kein b enthalten.
2. Alle Texte, die mit a oder b beginnen und mit c enden.
3. Alle Texte, die mit a beginnen und mit b enden oder mit b beginnen und mit a enden.

Es gilt `letter` $::= a|\ldots|z$.

Lösungsvorschlag 2.2

```
a+ : a a*

1. (a | c | ... | z)*
2. (a|b) letter* c
3. (a letter* b) | (b letter* a)
```

Aufgabe 2.3 (P) Meiern

In dieser Aufgabe wollen wir das Würfelspiel „Meiern“ implementieren, das auf den folgenden Regeln basiert:

> Für das Spiel braucht man zwei Würfel. Ein Spieler tritt gegen den Computer an. In jeder Spielrunde würfelt der Spieler mit beiden Würfeln. Die beiden gewürfelten Zahlen kombiniert er zu einer zweistelligen Zahl, so dass die größere der beiden als 10er Stelle und die kleinere von beiden als 1er-Stelle interpretiert wird.
>
> Für die Rangfolge der Würfe gilt:
>
> Der höchste Wurf (21), der sogenannte „Meier“, ist durch keinen anderen Wurf zu überbieten. Ein Spieler, der einen „Meier“ würfelt, hat sofort gewonnen.
> Kurz nach dem „Meier“ folgt der *6er-Pasch* (66), gefolgt von den anderen Paschen in absteigender Reihenfolge. Auf den kleinsten Pasch (11) folgen dann die normalen Würfe in absteigender numerischer Reihenfolge von 65 bis hinab zum kleinstmöglichsten Wurf (31).
> Es wird so lange abwechselnd von Spieler und Computer gewürfelt, bis der aktuelle Spieler den Wurf des vorherigen Spielers nicht mehr überbietet und deshalb das Spiel verliert (Ausnahme ist der „Meier“ wie zuvor beschrieben).

Schreiben Sie ein `MiniJava`-Programm, mit dem man „Meiern“ gegen den Computer spielen kann. Der Spieler soll abwechselnd über Dialogboxen über seine und die Würfe des Computers informiert werden.

Realisieren Sie diese Aufgabe Schritt für Schritt:

- Die Klasse `MiniJava` stellt Ihnen eine Methode dice() zur Verfügung, die Ihnen einen zufälligen Zahlenwert von 1 bis 6 liefert.
- Um eine Zahl zwischen 1 und 6 zu würfeln, verwenden Sie beispielsweise den folgenden Code: `int zahl; zahl = dice();`.
- Werfen Sie zwei Würfel und bestimmen Sie den Wert des Wurfes.
- Behandeln Sie die besondere Rangfolge der Würfe.
- Erweitern Sie das Programm, so dass die Würfe von Spieler und Computer unterschieden werden können.
- Merken Sie sich den Wurf des vorherigen Spielers.
- Denken Sie daran, den Spieler über die Würfe und über Sieg oder Niederlage zu informieren.

Lösungsvorschlag 2.3

Die Lösung befindet sich in der Datei `Meiern.java`.

Aufgabe 2.4 (P) Lustige Sieben

In dieser Aufgabe wollen wir das Würfelspiel *Lustige Sieben* als ein Programm namens `LustigeSieben.java` schreiben. Es gibt einen Spieler, der gegen die Bank spielt. Der Spieler startet mit einem Guthaben von 100. Der Spieler setzt einen Teil seines Guthabens auf nur eines der Felder des Spielfelds. Die Bank würfelt mit zwei Würfeln. Dazu steht Ihnen die Methode `int dice()` der Klasse `MiniJava` zur Verfügung. Anschließend zahlt die Bank entsprechend folgender Regel an den Spieler dessen Gewinn aus:

- Der dreifache Einsatz wird ausgezahlt, falls die Summe der beiden Würfel 7 ergibt und der Spieler auf die 7 gesetzt hat;
- der doppelte Einsatz wird ausgezahlt, falls die Summe der beiden Würfel genau der gewählten Zahl des Spielfeldes entspricht;
- der einfache Einsatz wird zurückgezahlt, falls sich das Würfelergebnis auf derselben Längsseite wie die gewählte Zahl befindet.

Beispiel: Wenn insgesamt 4 Augen gewürfelt werden, so erhält der Spieler für die Wahl der 4 den doppelten Einsatz, seinen Einsatz zurück, wenn der Spieler auf die $2, 3, 5$ oder 6 gesetzt hat, und verliert andernfalls seinen Einsatz an die Bank.

Das Spielfeld sieht wie folgt aus:

7	
2	8
3	9
4	10
5	11
6	12

Das Programm fragt so lange nach der gewählten Zahl und dem Einsatz, bis das Guthaben von 100 des Spielers aufgebraucht ist oder der Spieler die 0 zur Beendigung des Glücksspiels eingegeben hat. Nach jeder Runde sollen das Würfelergebnis und das Guthaben des Spielers ausgegeben werden.

Hinweis: Achten Sie darauf, Eingaben auf ihre Gültigkeit hin zu überprüfen!

Hilfestellung: Implementieren Sie die Aufgabenstellung in kleinen Schritten so weit Sie kommen:

- Lassen Sie den Spieler zuerst nur einmal und ohne Einsatz spielen – Vergleichen Sie den Tipp mit dem Würfelergebnis!
- Lassen Sie den Spieler einen beliebigen Betrag auf eine Zahl setzen – Geben Sie den Gewinn/Verlust aus!
- Legen Sie einen Kontostand für den Spieler an und verrechnen Sie den Gewinn/Verlust mit dem Kontostand. Verhindern Sie eine Überschreitung des Kontos!
- Lassen Sie wiederholt neue Spiele mit dem aktualisierten Kontostand zu!
- Für die Dialoge können Sie die Methode `readInt(String txt)` verwenden, die ein Dialogfeld mit der Nachricht `txt` erzeugt und eine Eingabe erwartet.

Lösungsvorschlag 2.4

Die Lösung befindet sich in der Datei `LustigeSieben.java`.

Die Hausaufgabenabgabe erfolgt über Moodle; beachten Sie, dass es mehrere Moodle-Seiten gibt (um Ihre und unsere Verwirrung zu erhöhen). Bitte geben Sie Ihren Code als UTF8-kodierte (ohne BOM) Textdatei(en) mit der Dateiendung `.java` ab. Geben Sie **keine** Projektdateien Ihrer Entwicklungsumgebung ab. Geben Sie **keinen** kompilierten Code ab (`.class`-Dateien). Geben Sie Ihren Code **nicht** als Archiv (z.B. als `.zip`-Datei) ab. Achten Sie darauf, dass Ihr Code kompiliert. Auf diesem Blatt gibt auch Aufgaben, zu denen die Lösung kein Programm ist. Geben Sie diese Aufgaben in einer einzelnen, korrekt orientierten und gut lesbaren DIN-A4-PDF-Datei ab. Hausaufgaben, die sich nicht im vorgegebenen Format befinden, werden nur mit Punktabzug oder gar nicht bewertet.

Aufgabe 2.5 (H) Mailefonnummernadressen [4 Punkte]

Gegeben seien $\texttt{letter} ::= a|\ldots|z$, $\texttt{special} ::= \_\,|\,.\,|-$ sowie $\texttt{digit} ::= 0|\ldots|9$ als Spezifikation für (Klein-)Buchstaben, Sonderzeichen (Unterstrich, Punkt, Minuszeichen) und Ziffern.

1. Geben Sie eine Grammatik zur Spezifikation zu lässiger Telefonnummern gemäß folgender Regeln an. Eine zulässige Telefonnummer besteht aus einer optionalen Länder-Vorwahl gefolgt von einer Vorwahl des Wohnortes gefolgt von einer Anschlussnummer. Enthält die Telefonnummer eine Länder-Vorwahl, so wird dieser entweder ein Pluszeichen oder eine Doppelnull vorangestellt. Enthält die Telefonnummer keine Länder-Vorwahl, so wird ihr eine einfache Null vorangestellt. Verwenden Sie die folgenden Grundbausteine:

   Länder-Vorwahl: $\texttt{lvw} ::= 49\,|\,37\,|\,1876\ldots$

   Orts-Vorwahl: $\texttt{ow} ::= 89\,|\,99\ldots$

   Anschlussnummer: $\texttt{nr} ::= \texttt{digit}^+$

   Beispiele für zulässige Telefonnummern nach diesen Vorgaben sind 0891234567 und 0049891234567.

2. Geben Sie eine Grammatik zur Spezifikation zu lässiger E-Mail-Adressen gemäß folgender Regeln an. Ein zulässiger *Name* ist eine nichtleere Zeichenkette, die

   - aus beliebig vielen Buchstaben (`letter`) besteht.
   - Zusätzlich dürfen Sonderzeichen (`special`) enthalten sein, sofern jeweils direkt vor und direkt hinter ihnen ein Buchstabe steht.

   Eine zulässige Adresse ist dann wie folgt gekennzeichnet.

   - Die Adresse enthält genau einmal das Zeichen @.
   - Vor dem @ steht ein zulässiger Name.
   - Hinter dem @ steht ein zulässiger Name, der mindestens einen Punkt enthält.

Lösungsvorschlag 2.5

```
1. telefonnummer ::= (((+|00) lvw) | 0) ow nr

2. name ::= letter letter* (special letter letter*)*
   mailadresse ::= name @ name . name
```

Teil 2. mit der Interpretation, dass mehrere Sonderzeichen hintereinander erlaubt sind:

```
2. name ::= letter+ (special* letter+)*
   mailadresse ::= name @ name special* . special* name
```

Korrekturbemerkung:

1. 1 Punkt; fast richtige Lösung gibt 0.5 Punkte.
2. 3 Punkte; je kleinem Fehler 1 Punkt Abzug (bspw. letter* statt letter+)

Aufgabe 2.6 (H) Reguläre Ausdrücke [4 Punkte]

Geben Sie unter Verwendung der in der Vorlesung und im Praktikum für reguläre Ausdrücke eingeführten Operatoren *, ?, | und + reguläre Ausdrücke für die folgenden Textmuster an, wobei `letter` einen beliebigen Buchstaben im Text beschreibt:

1. Es kommen mindestens zwei, maximal vier a's im Text vor.
2. Alle a's stehen an *geraden Positionen*, d. h. der Buchstabe a kann nur an 2., 4., 6., 8. usw. Stelle stehen.
3. Vor (d. h. links von) jedem a im Text kommt irgendwo (d. h. an beliebiger Stelle) ein einzelnes b vor. *Einzeln* bedeutet dabei, dass weder direkt links daneben noch direkt rechts daneben ein weiteres b steht.

Es gilt `letter` $::= a|\ldots|z$, `bbisz` $::= b|\ldots|z$, `cbisz` $::= c|\ldots|z$.

Lösungsvorschlag 2.6

```
1. bbisz* a bbisz* a bbisz* a? bbisz* a? bbisz*
2. bbisz | (bbisz letter)* | (bbisz letter)* bbisz
3. bbisz* | (bbisz* cbisz)? b (cbisz bbisz*)? a letter*
```

Korrekturbemerkung:

Punkte (jeweils halbe Punktzahl, falls nahezu korrekt):

1. 1 Punkt
2. 1 Punkt
3. 2 Punkte

Aufgabe 2.7 (H) 3 und 7 [4 Punkte]

Schreiben Sie ein MiniJava-Programm, welches die Summe aller positiven Zahlen kleiner n bildet, die durch 3 oder 7 teilbar sind. Die Zahl n soll vom Nutzer eingelesen werden. Gibt der Nutzer dabei eine negative Zahl ein, soll dieser um die Eingabe einer neuen Zahl gebeten werden.

Gibt der Benutzer z.B. die Zahl 25 ein, müssen die Zahlen 3, 6, 7, 9, 12, 14, 15, 18, 21 und 24 summiert werden, was 129 ergibt. Das Ergebnis soll ausgegeben werden.

Lösungsvorschlag 2.7

```
public class ThreeAndSeven extends MiniJava {
  public static void main(String[] args) {
    int n = read("Geben Sie eine Zahl n >= 0 ein.");
    while (n < 0) {
      n = read("Was verstehen Sie an '>=0' nicht?");
    }

    int sum = 0;
    int i = 0;
    while (i < n) {
      if (i % 3 == 0 || i % 7 == 0) {
        sum += i;
      }
      i += 1;
    }

    write("Die Summe ist " + sum + ".");
  }
}
```

Aufgabe 2.8 (H) Prima Zahlen [4 Punkte]

In dieser Aufgabe soll ein Programm geschrieben werden, welches für eine Zahl $n > 0$ testet, ob n eine Primzahl ist. Die Zahl n soll durch den Benutzer eingegeben werden. Gibt der Benutzer eine ungültige Zahl ein, soll dieser um erneute Eingabe gebeten werden. Das Programm soll ausgeben, ob es sich bei n um eine Primzahl handelt.

Lösungsvorschlag 2.8

```
public class PrimTest extends MiniJava {
  public static void main(String[] args) {
    int n = read("Geben Sie eine Zahl n > 0 ein.");
    while (n <= 0) {
```

```
      n = read("Das war wohl nix! Bitte n > 0 eingeben.");
    }

    // Contains the number of divisors (excluding 1 and n)
    int divisors = 0;

    // Search for proper divisors in [2; n[
    int i = 2;
    while(i < n) {
      if(n % i == 0)
        divisors += 1;
      i += 1;
    }

    // A number is prime iff it doesn't have any divisors
    // (excluding 1 and n)
    if(divisors == 0) {
      write("Primzahl!");
    } else {
      write("Keine Primzahl!");
    }
  }
}
```

Aufgabe 2.9 (H) Potenztabelle [4 Punkte]

In dieser Aufgabe geht es darum, eine Latex-formatierte Potenztabelle auf der Konsole auszugeben. Das Programm soll dazu zunächst die Größe der Tabelle vom Benutzer einlesen. Der Benutzer gibt eine Zahl $n > 0$ ein, die für die Anzahl Zeilen und Spalten verwendet wird. Gibt der Benutzer eine ungültige Zahl ein, soll er um erneute Eingabe gebeten werden. Die Tabelle soll in der i-ten Zeile und j-ten Spalte den Wert i^{j-1} enthalten (wir zählen Zeilen und Spalten ab 1).

Gibt der Benutzer z.B. die Zahl 5 ein, so soll die Ausgabe wie folgt aussehen:

```
\begin{tabular}{lllll}
1 & 1 & 1 & 1 & 1\\
1 & 2 & 4 & 8 & 16\\
1 & 3 & 9 & 27 & 81\\
1 & 4 & 16 & 64 & 256\\
1 & 5 & 25 & 125 & 625\\
\end{tabular}
```

Machen Sie sich dazu zunächst mit Tabellen in Latex vertraut[1]. Ein Beispieldokument finden Sie außerdem unter `https://goo.gl/8jEYNd`. Melden Sie sich auf `https://sharelatex.`

[1] `https://en.wikibooks.org/wiki/LaTeX/Tables`

`tum.de` an, um schnell und einfach ein eigenes Latex-Projekt zum Testen zu erstellen. Nutzen Sie `writeLineConsole("text")` bzw. `writeConsole("text")`, um den String `"text"` (und ggf. einen Zeilenumbruch) auf der Konsole auszugeben.

Hinweis: Sie müssen in Ihrem Programm nicht auf Überläufe achten.

Hinweis: Verwenden Sie `writeConsole("\\")`, um einen einzelnen Backslash (\) auf der Konsole auszugeben.

Lösungsvorschlag 2.9

```
public class PotencyTable extends MiniJava {
  public static void main(String[] args) {
    int n = read("Geben Sie eine Zahl n >= 0 ein.");
    while (n < 0) {
      n = read("Die Zahlen >= 0 sind die ohne Minuszeichen...");
    }

    // We generate the header which needs to contain an
    // 'l' for each column.
    writeConsole("\\begin{tabular}{");
    int i = 0;
    while (i < n) {
      writeConsole("l");
      i += 1;
    }
    writeLineConsole("}");

    // We generate the table line-by-line.
    i = 1;
    while (i <= n) {
      int j = 0;
      while (j < n) {
        // We separate the fields by a '&'.
        if (j > 0)
          writeConsole(" & ");

        // We compute the potency using 'j'
        // many multiplications of 'i'.
        int pot = 1;
        int k = 0;
        while (k < j) {
          pot *= i;
          k += 1;
        }

        writeConsole(pot);
        j += 1;
      }
      // Each line needs to end in '\\'.
```

```
40        writeLineConsole("\\\\");
41        i += 1;
42      }
43
44      writeLineConsole("\\end{tabular}");
45    }
46  }
```