Musterlösung zu Übungsblatt 4

Praktische Informatik I

17. November 1999

Der Aufgabentext ist in normaler Schrift gegeben, die Lösungen in kursiver Schrift.

1 Syntax und Semantik [3 Punkte]

(a) [2 Punkte]

Gegeben sind folgende Variablen:

int a=1; byte b=1; short c=1;

Welche der nachfolgenden Ausdrücke enthalten Fehler? Geben Sie bei jedem Fehler an, ob es sich um einen semantischen oder syntaktischen Fehler handelt. Geben Sie eine Vermutung an, wie der Ausdruck korrekt aussehen sollte!

(i) $5 * 7||30\%12 - 39/3 - 3 * 12 - 2 = 0$

Syntaxfehler: || kann nur zwei boolean-Werte verknüpfen (keine Zahlen).

Syntaxfehler: zwischen zwei Werten darf kein Zuweisungsoperator stehen. Entweder muss links vom "=" ein Variablenname stehen, oder es muss statt des Zuweisungsoperators ein Vergleichsoperator verwendet werden.

Korrekt ist als Ausdruck z.B. $(5 * 7|30\%12 - 39/3 - 3 * 12 - 2) == 0$*, dies entspricht* $(35|\text{-}45) == 0$*, also* $\text{-}13 == 0$*, also false. Bemerkung: "==" hat hoehere Priorität als "|"!*

(ii) $a + + == a$

Dieser Ausdruck ist syntaktisch korrekt.

Was die Semantik angeht: Der Ausdruck wertet immer zu true aus, und macht i.A. wohl nicht viel Sinn.

(iii) $c = 100000 + 3 * a * b * 1/a/b$

Dieser Ausdruck ist syntaktisch korrekt, semantisch aber falsch, weil der short-Variablen c ein zu grosser Wert zugewiesen wird (300003).

Eine korrekte Form des Ausdrucks ist z.B. $c = 10000 + 3 * a * b * 1/a/b$ *(30003 < 32768).*

(iv) $1 + 5 == 6 == 5 + 1$

Syntaxfehler: Der Vergleich von zwei integers (1+5==6) liefert einen boolean-Wert. Dieser soll dann mit einem integer verglichen werden (true==6), und dies ist in Java nicht erlaubt.

Eine korrekte Form ist z.B. $1 + 5 == 6$ *oder auch* $(1 + 5 == 6)\&\&(6 == 5 + 1)$.

Bewertungsvorschlag: Je einen halben Punkt, wenn in (i) beide Syntaxfehler, in (ii) die syntaktische Korrektheit, in (iii) der semantische Fehler und in (iv) der Syntaxfehler erkannt wurden. Ein halber Punkt (insgesamt) Abzug, wenn keine korrigierten Fassungen angegeben wurden.

(b) [1 Punkt] Korrigieren Sie die folgenden deutschen Sätze und geben Sie an, wo die Semantik und wo die Syntax falsch ist!

(i) "Ein Elefant is ein lilafarbenes, großes Tier."

Syntax: "ist" statt "is"

Semantik: Ein Elefant ist im Allgemeinen grau, nicht lila.

Korrigierte Fassung: "Ein Elefant ist ein graues (oder auch: graufarbenes), großes Tier."

(ii) "Wir programmiere in Java, weil Java als total veraltete Programmiersprache total veraltet ist."

Syntax: "programmieren" statt "programmiere"

Semantik: Man kann vielleicht viel über Java sagen, aber sicher nicht, dass es veraltet ist. Außerdem wird "total veraltet" unnötigerweise wiederholt.

Korrigierte Fassung: "Wir programmieren in Java, weil Java als ganz neue Programmiersprache total cool ist!" (oder so ähnlich...)

Bewertungsvorschlag: Ein Punkt, wenn je ein Syntaxfehler und ein semantischer Fehler richtig erkannt und korrigiert wurden. Ein halber Punkt ab zwei erkannten Fehlern.

2 Java-Syntax [8 Punkte]

Die Java-Syntax unterstützt die drei Wiederholungsanweisungen *for, while* und *do - while.* Die drei Schleifen sind durch folgende Syntax-Regeln definiert (siehe auch Schader/Schmidt-Thieme, S.513ff):

While-Anweisung:
while (Ausdruck) Anweisung

Do-Anweisung:
do Anweisung while (Ausdruck);

For-Anweisung:
for (For-Init$_{opt}$; Ausdruck$_{opt}$; For-Update$_{opt}$) Anweisung

For-Init:
Anweisungsausdrucks-Liste
Lokale Variablendeklaration

For-Update:
Anweisungsausdrucks-Liste

Anweisungsausdrucks-Liste:
Anweisungsausdruck
Anweisungsausdrucks-Liste, Anweisungsausdruck

Zeigen Sie: Die in der Java-Syntax enthaltenen drei Wiederholungsanweisungen *for, while* und *do* sind äquivalent. Mit anderen Worten: eine While-Anweisung läßt sich auch mit Hilfe einer Do-Anweisung und einer For-Anweisung ausdrücken und umgekehrt.

Tip: Beweisen Sie zunächst die Äquivalenz von while und do, indem Sie die Funktion einer Do-Anweisung mit Hilfe einer While-Anweisung simulieren und umgekehrt. Zeigen Sie anschließend die Äquivalenz von while und for in analoger Art und Weise.

Zunächst soll gezeigt werden, dass die While- und Do-Anweisungen äquivalent sind:

Hierzu ist zu bemerken, dass der einzige semantische Unterschied zwischen einer while *und einer* do *Schleife der ist, dass bei einer* do *Schleife die Anweisung* in jedem Fall mindestens einmal *ausgeführt wird, bei einer* while *Schleife aber dann nicht, wenn der Ausdruck von Anfang an zu false auswertet.*

Simulieren einer do-while *Schleife mit einer While-Anweisung:*

```
Anweisung    //Die Anweisung auf jeden Fall einmal ausführen
while (Ausdruck) Anweisung
```

Simulieren einer while *Schleife mit einer Do-Anweisung:*

```
if (Ausdruck) do Anweisung while (Ausdruck);
```

(ist Ausdruck von Anfang an false, wird nichts ausgeführt, wie bei einer do-while *Schleife)*

⇒ while *und* do *Schleifen sind äquivalent.*

Als nächstes wird gezeigt, dass die For- und While-Anweisungen äquivalent sind:

Simulieren einer for *Schleife mit einer While-Anweisung:*

For-Init;$_{opt}$
while (Ausdruck) { //bzw. while(true), wenn kein Ausdruck (opt) angegeben
  Anweisung
  For-Update;$_{opt}$
}

Simulieren einer while *Schleife mit einer For-Anweisung:*

for (; Ausdruck;) Anweisung

⇒ for *und* while *Schleifen sind äquivalent.*

Aus der Äquivalenz von do und while und der Äquivalenz von while und for folgt auch die Äquivalenz von for und do.

Bemerkung: Der Ausdruck in der for *Schleife ist wie bei* while *diejenige Bedingung, die für das* Fortsetzen *der Schleife erfüllt sein muß.*
Bemerkung: Anweisung kann entweder eine einzelne, mit Semikolon abgeschlossene Java-Anweisung sein, oder auch ein ganzer Codeblock (in geschweiften Klammern).

Bewertungsvorschlag: Je zwei Punkte für jede "Simulation", ein Punkt Abzug, wenn nicht explizit aus den zwei gezeigten Äquivalenzen die dritte gefolgert wird.

3 Java-Programmierung [9 Punkte]

Ein bekanntes Rätsel ist das Damenproblem im Schach (C.F.Gauß, 1850): 8 Damen sind so auf das Spielbrett zu stellen, dass sie sich gegenseitig nicht schlagen können. Wir betrachten eine Vereinfachung des Problems mit 8 Türmen. Formal ausgedrückt haben wir eine 8 × 8 Matrix mit Elementen aus {0,1}. Eine 1 gibt an, dass auf einem Feld ein Turm steht, eine 0 gibt an, dass das Feld frei ist. Eine Matrix stellt eine Lösung des Turmproblems dar, wenn in jeder Zeile und in jeder Spalte höchstens eine 1 zu finden ist.

Entwerfen Sie ein Java-Programm (Application), das feststellt, ob eine gegebene Matrix eine Lösung des Turmproblems darstellt.

Anmerkung: Benutzen Sie den Klassennamen `NTuermeTester`! Das Programm muss als Kommandozeilenparameter den Namen einer Datei erwarten, die eine 8 × 8 Matrix der obigen Form enthält. Das Programm muss als Ergebnis "ja" ausgeben, falls die Matrix eine korrekte Lösung des Turmproblems darstellt, anderenfalls soll "nein" ausgegeben werden.
Hinweise zur Ein/Ausgabe in Java finden Sie auf den Webseiten der Übung. Die Testdateien `ntuermetest1.txt` und `ntuermetest2.txt` finden Sie ebenfalls im Web. Hier ein Beispiel für einen Kommandozeilenaufruf und die Ergebnisausgabe des von Ihnen zu entwickelnden Programmes:

```
> java NTuermeTester ntuermetest1.txt
 ja
```

Das folgende Programm liest eine 8 × 8 Matrix der oben genannten Form aus der Textdatei, deren Namen per Kommandozeile übergeben wurde und prüft, ob es sich dabei um eine Lösung zum Turmproblem handelt.

Hierzu wird geprüft, ob in jeder Zeile und in jeder Spalte der Matrix nur genau *ein Turm steht. Genau dann ist die Matrix eine Lösung.*

Wenn mehr als ein Turm in einer Zeile/Spalte steht, ist die Matrix offensichtlich keine Lösung, da dann ein Turm den anderen schlagen kann. Aber auch wenn wir eine Zeile/Spalte entdecken, in der kein *Turm steht wissen wir bereits, dass die Matrix keine Lösung sein kann: da wir genau acht Türme auf acht Zeilen und acht Spalten verteilen, muss in diesem Fall nämlich notwendigerweise eine andere Zeile/Spalte mit mehr als einem Turm besetzt sein. Diese Erkenntnis ist zwar nicht notwendig für die Korrektheit des Programmes, aber sie macht es effizienter.*

```
/*****************************************************************************
* Author:  Jan Peter Damm (C) 1999
* Project name:  Musterloesung NTuermeTester
* Date:  19.11.1999
* Name:  NTuermeTester.java
* Description:
* Reads an 8x8 matrix from a file and tests if it is a solution
* to the 8-Towers-Problem
*****************************************************************************/

import java.io.*;
import java.util.StringTokenizer;

class NTuermeTester {

  static int[][] board = new int[8][8]; //game board with 8x8 fields
  static BufferedReader fileIn; //input stream reading data from input file

  /*****************************************************************************
  * Name:  main
  * Description:
  * Reads an 8x8 matrix from a file and tests if it is a solution
  * to the 8-Towers-Problem.
  * Gives correct results if the input file has the correct structure
  * (i.e.  a text file containing eight rows of eight values each,
  * separated by whitespaces, each value being either 0 or 1, with
  * a total of eight 1-values in the file).
  * Output is either "ja" if the matrix is a solution, else "nein".
  * Parameters:
  * The method requires one parameter to be passed from command line,
  * which is the name of the input file.
  *****************************************************************************/
  public static void main(String[] args) {
    if (args.length != 1) { //program expects one input parameter
      System.out.println("Start with java NTuermeTester <input file name>");
    }
    try { //attempt to open input stream from file
      fileIn = new BufferedReader(new FileReader(args[0]));
    } catch (FileNotFoundException fnf) { //if file isn't found, exit
      System.out.println("File " + args[0] + "not found.");
      System.exit(1);
    }
```

```
    //Read data from input file into 2D-array board:
    String row = ""; //holds one row of input data
    for (int i=0; i<8; i++) { //read eight rows of data from file
      try {
        row = fileIn.readLine(); //read one row
      } catch (IOException ioe) { //exit if problem with i/o
        System.out.println("Couldn't read all data from file " + args[0]);
        System.exit(2);
      }
      StringTokenizer st = new StringTokenizer(row);
      for (int j = 0; j < 8; j++) { //read eight integers from row
        String s = st.nextToken(); //tokens are separated by whitespaces
        int value = Integer.parseInt(s);
        board[i][j] = value;
      }
    }

    //Check if there is exactly one tower in each row:
    for (int i=0; i<8; i++) { //row by row
      int numTowersInRow = 0; //number of towers in this row
      for (int j=0; j<8; j++) { //column by column
        numTowersInRow += board[i][j]; //if 1, increase count
      }
      if (numTowersInRow != 1) { //if not exactly one tower,
        System.out.println("nein"); //print "nein"
        System.exit(0); //and exit
      }
    }

    //Check if there is exactly one tower in each column:
    for (int j=0; j<8; j++) { //column by column
      int numTowersInCol = 0; //number of towers in this column
      for (int i=0; i<8; i++) { //row by row
        numTowersInCol += board[i][j]; //if 1, increase count
      }
      if (numTowersInCol != 1) { //if not exactly one tower,
        System.out.println("nein"); //print "nein"
        System.exit(0); //and exit
      }
    }
    //If we have come this far, it is a solution to the tower problem
    System.out.println("ja");
  }
}
```

Bewertungsvorschlag: $4\frac{1}{2}$ Punkte (50 %) auf die korrekte Funktion, wenn die gegebenen Testdateien und noch ein bis zwei weitere Testdateien korrekt erkannt werden. Im Code einen Punkt auf das Einlesen aus der Datei und je einen auf die beiden Prüfschleifen, sowie $1\frac{1}{2}$ Punkte auf gute Kommentierung.