Prof. Andreas Goerdt
M. Pippig, M. Riedel, U. Schönke

TU Chemnitz
Wintersemester 2003/2004

# Algorithmen und Programmierung

## 9. Übung – Lösungsvorschläge

1. Aufgabe:

   Der erste Teil der Invariante

   $$\mathrm{ggT}(a,b) = \mathrm{ggT}(x_l, y_l)$$

   wurde bereits in der 5. Übung (Aufgabe 2a, Euklidischer Algorithmus mit Subtraktionen) bewiesen. Es bleibt also noch

   $$x_l \cdot v_l + y_l \cdot u_l = 2ab$$

   zu beweisen. Dabei seien wie gehabt $x_l$, $y_l$, $u_l$ und $v_l$ die Werte der Variablen *x*, *y*, *u* und *v* nach dem *l*-ten Schleifendurchlauf.

   **Induktionsanfang:**

   Für $l = 0$ gilt: $x_0 = a$, $y_0 = b$, $u_0 = a$, $v_0 = b$

   Also ist

   $$\begin{aligned} x_0 \cdot v_0 + y_0 \cdot u_0 &= a \cdot b + b \cdot a \\ &= 2ab \end{aligned}$$

   und die Invariante gilt.

   **Induktionsschritt:**

   Für $l \geq 1$ gilt nach Voraussetzung (Invariante):

   $$x_{l-1} \cdot v_{l-1} + y_{l-1} \cdot u_{l-1} = 2ab$$

   Wir betrachten nun den *l*-ten Schleifendurchlauf:

   Falls $x_{l-1} < y_{l-1}$, dann ist

   $$\begin{aligned} x_l = x_{l-1} \quad &, \quad y_l = y_{l-1} - x_{l-1} \\ u_l = u_{l-1} \quad &, \quad v_l = v_{l-1} + u_{l-1} \end{aligned}$$

   Einsetzen ergibt

   $$\begin{aligned} x_l \cdot v_l + y_l \cdot u_l &= x_{l-1} \cdot (v_{l-1} + u_{l-1}) + (y_{l-1} - x_{l-1}) \cdot u_{l-1} \\ &= x_{l-1} \cdot v_{l-1} + x_{l-1} \cdot u_{l-1} + y_{l-1} \cdot u_{l-1} - x_{l-1} \cdot u_{l-1} \\ &= x_{l-1} \cdot v_{l-1} + y_{l-1} \cdot u_{l-1} \\ &= 2ab \end{aligned}$$

und die Invariante gilt folglich.

Falls andererseits $x_{l-1} \geq y_{l-1}$, dann ist

$$\begin{array}{lll} x_l = x_{l-1} - y_{l-1} & , & y_l = y_{l-1} \\ u_l = u_{l-1} + v_{l-1} & , & v_l = v_{l-1} \end{array}$$

Einsetzen ergibt

$$\begin{array}{rcl} x_l \cdot v_l + y_l \cdot u_l & = & (x_{l-1} - y_{l-1}) \cdot v_{l-1} + y_{l-1} \cdot (u_{l-1} + v_{l-1}) \\ & = & x_{l-1} \cdot v_{l-1} - y_{l-1} \cdot v_{l-1} + y_{l-1} \cdot u_{l-1} + y_{l-1} \cdot v_{l-1} \\ & = & x_{l-1} \cdot v_{l-1} + y_{l-1} \cdot u_{l-1} \\ & = & 2ab \end{array}$$

und die Invariante gilt folglich.

Sei $n$ die Anzahl der Schleifendurchläufe. Wir betrachten nun den letzten Schleifendurchlauf: Laut Invariante gilt

$$\text{ggT}(x_n, y_n) = \text{ggT}(a,b) \quad \textbf{und} \quad x_n \cdot v_n + y_n \cdot u_n = 2ab$$

Da es sich um den letzten Schleifendurchlauf handelt, gilt weiterhin

$$x_n = y_n$$

Mit dem Wert der Variable $x$ wird folglich der $\text{ggT}(a,b)$ zurückgeliefert.

Der zweite zurückgelieferte Wert ist $\frac{u+v}{2}$. Betrachten wir den zweiten Teil der Invariante genauer, dann sehen wir:

$$\begin{array}{rcl} x_n \cdot v_n + y_n \cdot u_n & = & 2ab \\ x_n \cdot v_n + x_n \cdot u_n & = & 2ab \\ x_n \cdot (v_n + u_n) & = & 2ab \\ v_n + u_n & = & \frac{2ab}{x} \\ \frac{v_n + u_n}{2} & = & \frac{ab}{\text{ggT}(a,b)} \end{array}$$

Der Algorithmus liefert also als zweites das kleinste gemeinsame Vielfache von $a$ und $b$ zurück.

2. Aufgabe:

   a)

In Beispiel 1 „wandert“ *i* durch das gesamte Feld bis zum letzten Element, da sich kein größeres Element als dieses im Feld befindet, und bleibt dort stehen. *j* bleibt dagegen sofort auf dem letzten Element stehen, da es das Pivotelement selbst ist. Nun sind aber *i* und *j* gleich, weshalb *i* inkrementiert und *j* dekrementiert wird. *i* bekommt also den Wert „Länge des Feldes “, welcher jedoch außerhalb der Feldgrenzen (0 bis Länge-1) liegt.

In Beispiel 2 bleibt *i* sofort auf dem ersten Element stehen, da es sich um das Pivot-Element handelt. *j* wandert bis zum ersten Element durch, da sich kein kleineres als dieses im Feld befindet. *i* und *j* sind wieder gleich, weshalb *i* inkrementiert und *j* dekrementiert wird. *j* hat nun den Wert -1, welcher wieder außerhalb der Feldgrenzen liegt.

b) Der Unterschied zum ursprünglichen Algorithmus besteht in den veränderten Startwerten. Als Pivotelement wird das letzte Element des Feldes festgelegt und j mit `feld.length-1` initialisiert.

Der Algorithmus funktioniert dann nicht, wenn das Pivotelement zufällig das kleinste Element im Feld ist. In der ersten inneren `do-while`-Schleife wird *j* solange dekrementiert, bis das *j*-te Feldelement kleiner gleich dem Pivotelement ist. Da es sich jedoch um eine `do-while`-Schleife handelt, wird zuerst *j* dekrementiert und danach der Test durchgeführt. Somit wird das letzte Feldelement nicht mit dem Pivotelement (sich selbst) verglichen und bleibt nicht dort stehen. Da sich jedoch kein kleineres als das letzte Element im Feld befindet, wird *j* komplett durch das Feld wandern und schließlich auf nicht existierende Feldelemente zugreifen (negative Indizes).

3. Aufgabe:

Damit zwei Matrizen miteinander multipliziert werden können, muss die Anzahl der Spalten der ersten Matrix mit der Anzahl der Zeilen der zweiten Matrix übereinstimmen. Es können also nur $m \times n$-Matrizen mit $n \times p$-Matrizen multipliziert werden. Die Ergebnismatrix hat die Dimension $m \times p$.

```
import Prog1Tools.IOTools;
public class MatrizenMult {

  public static void main(String[] args) {
    int m, n, p, i, j, k;
    int [][] a, b, c;

    // Dimensionen der Matrizen (m×n,n×p) einlesen
    m = IOTools.readInteger("Bitte m eingeben :");
    n = IOTools.readInteger("Bitte n eingeben :");
    p = IOTools.readInteger("Bitte p eingeben :");

    // Matrizen a und b initialisieren
    // ...
```

```
    // evtl. Ausgabe von a und b
    // ...

    // Ergebnismatrix c erzeugen und initialisieren
    c = new int[m][p];
    for (i=0; i < m; i++)
      for (j=0; j < p; j++)
        c[i][j] = 0;

    // Matrizenmultiplikation
    for (i=0; i < m; i++)
      for (k=0; k < p; k++)
        for (j=0; j < n; j++)
          c[i][k] = c[i][k] + a[i][j] * b[j][k];

    // Ausgabe von c
    for (i=0; i < m; i++) {
      for (j=0; j < p; j++)
        System.out.print(c[i][j] + "␣");
      System.out.println();
    }
    return;
  }
}
```

4. Aufgabe:

```
public class Swaps{

  public static int[] swappedCopy (int[] a) {

    int[] ergebnis;

    ergebnis = new int[a.length];

    for(int i=0; i<a.length; i++){
      ergebnis[i] = a[a.length-1-i];
    }
    return ergebnis;
  }
```

```
  public static void swap (int[] a) {
    int halb;
    int hilf;
    halb = a.length / 2;

    for (int i=0; i<halb; i++) {
      // i-tes Element von links mit
      // i-tem Element von rechts tauschen
      hilf = a[i];
      a[i] = a[a.length-1 - i];
      a[a.length-1 - i] = hilf;
    }
  }

  public static void main(String[] args) {

    int[] feld;

    // feld mit new in gewuenschter Groesse erzeugen, z.B.
    feld = new int[10];

    // feld mit Werten belegen und gleich ausgeben, z.B.
    for (int i=0; i<feld.length; i++){
      feld[i] = i+1;
      System.out.println("feld["+i+"]=␣"+feld[i]);
    }

    int[] kopie = new int[feld.length];
    kopie = swappedCopy(feld);
    // kopie enthaelt jetzt gespiegelte Elemente von feld
    // Ausgabe:
    for (int i=0; i<kopie.length; i++){
      System.out.println("kopie["+i+"]=␣"+kopie[i]);
    }

    swap(feld);
    // jetzt ist feld selbst gespiegelt
    // Ausgabe
    for (int i=0; i<feld.length; i++){
      System.out.println("feld["+i+"]=␣"+feld[i]);
    }
  }
}
```

5. Aufgabe:

   Beim ersten Aufruf `max(x,y,z)` wird die erste Methode mit zwei `int`- und einem `double`-Parameter verwendet. Java verwendet in solchen Fällen immer die Methode, bei der die wenigsten Typkonvertierungen notwendig sind. (Verlustbehaftete Konvertierungen (z.B. `double` nach `int`) werden natürlich nicht ausgeführt.)

   Der zweite Aufruf `max(x,y)` ergibt jedoch bereits einen Compile-Fehler: Da bei der dritten und vierten Variante von `max` zur Anwendung auf zwei `int`-Variablen gleich viele Typkonvertierungen notwendig sind, verweigert Java die Compilierung wegen Mehrdeutigkeit („`reference to max is ambiguous`“).

6. Aufgabe:

   a) Beim Übersetzen erhält man eine Fehlermeldung, da die Variable *k* nicht initialisiert ist („`variable k might not have been initialized`“).

   b) Das Übersetzen geht problemlos, beim Ausführen wird fünf mal die Zeile
   `Feld: 0`
   ausgegeben, da Feldelemente automatisch mit 0 (bei `int`) initialisiert werden.

   c) Das Übersetzen geht problemlos, beim Ausführen wird
   `Klasse: 0`
   ausgegeben, da Klassenvariablen automatisch initialisiert werden.