Technische Universität München
Fakultät für Informatik
R. Kuntschke, I. Nagy,
Dr. M. Schneider

SS 2006
Lösungsvorschläge zu Blatt 8
30. Juni 2006

Übungen zu Einführung in die Informatik I

Aufgabe 31 Nichtstrikte Funktionen (Lösungsvorschlag)

Sowohl der bedingte Ausdruck als auch die Konjunktion und Disjunktion werden in Ocaml gemäß der „Lazy"-Strategie ausgewertet. Im Falle der Konjunktion (Disjunktion) heißt das, dass das zweite Argument nur ausgewertet wird, wenn das erste `true` (`false`) ist. Bei dem bedingten Ausdruck wird je nachdem ob die Bedingung erfüllt oder nicht erfüllt ist nur der entsprechende Zweig ausgewertet.

Aufgabe 32 Primfaktorzerlegung (Lösungsvorschlag)

a) Kleinster Teiler:

```
public static int kleinsterTeiler (int n){
        boolean b = false;
        int d = 2;
        while ((n % d) != 0)
            d++;
        }
        return d;
}
```

b) PrimfaktorZerlegung:

```
public static void primfaktorZerlegung (int n) {
        if (n>=2) {
                int d = kleinsterTeiler(n);
                        System.out.println(d);
                while (d < n) {
                        n = n / d;
                        d = kleinsterTeiler(n);
                        System.out.println(d);
                }
        }
}
```

Aufgabe 33 Berechnung von π nach der Monte-Carlo-Methode (Lösungsvorschlag)

a) Rekursive Version:
In der rekursiven Version wurde die Anzahl der Stichproben auf 1000 heruntergesetzt, da bei höheren Zahlen ein Stackoverflow von Java auftreten kann.

```
import java.util.Random;

public class MonteCarlo_rek {

    public static int anzahl = 1000;
    public static Random generator = new Random();

    public static int versuch(int rest_anzahl, int treffer) {
            double x = generator.nextDouble();
            double y = generator.nextDouble();

            return rest_anzahl > 0
                ? ((x*x + y*y) < 1.0
                   ? versuch(rest_anzahl - 1, treffer + 1)
                   : versuch(rest_anzahl - 1, treffer))
                : treffer;
    }

    public static void main(String argv[]) {

        int treffer = versuch(anzahl, 0);
        double pi = 4 * ((double) treffer) / ((double) anzahl);

        System.out.println("Ermittelter Wert von PI: " + pi);
    }
}
```

b) Iterative Version:

```
import java.util.Random;

public class MonteCarlo {

    public static int anzahl = 1000;

    public static void main(String argv[]) {

        Random generator = new Random();

        int treffer = 0;

        for (int i = 0; i < anzahl; i++) {
            double x = generator.nextDouble();
            double y = generator.nextDouble();

            if ((x*x + y*y) < 1.0)
                treffer++;
        }

        double pi = 4 * ((double) treffer) / ((double) anzahl);

        System.out.println("Ermittelter Wert von PI: " + pi);
    }
}
```

Aufgabe 34 Potenz und allgemeine Wurzel (Lösungsvorschlag)

a) Potenz:

```
public static double powerInt (double x, int n){
        double result = 1.0;
        if (n<0) {
                x = 1/x;
                n = -n;
                }
        while (n > 0) {
                result = result*x;
                n = n-1;
        }
        return result;
}
```

b) Wurzel:

```
public static double abs (double x) {
        if (x<0) return -x;
        else return x;
}

public static double wurzel (double x, int n, double
   epsilon){
        double upper = x;
        double lower = 0;
        double mid = 0.5*(upper + lower);
        while (abs(upper-lower)> epsilon) {
                if (powerInt(mid, n) > x) upper = mid;
                else lower = mid;
                mid = 0.5*(upper + lower);
        }
        return mid;
}
```

Aufgabe 35 Binomialkoeffizienten und Pascalsches Dreieck (Lösungsvorschlag)

Eine mögliche Lösung könnte folgendermaßen aussehen:

```
public class PascalDreieck
{
  public static void main(String args [])
  {
    int n;
    n = Integer.parseInt(args[0]);
    System.out.println("Pascal-Dreieck␣mit␣n␣Zeilen,␣n␣=␣" + n);
    PascalDreieck p = new PascalDreieck(n);
  }

  PascalDreieck(int eingabe)
  {
    dreieck(eingabe);
```

```
    viereck(eingabe);
  }

  void dreieck(int x)
  {
    int i, k;
    for (i = 0; i < x; i++)
    {
      for(k = 0; k <= i; k++)
        System.out.print(binomkoeff(i, k) + "␣␣");
      System.out.println();
    }
  }

  int binomkoeff(int n, int k)
  {
    if ((k == 0) || (k == n))
      return 1;
    else
      return binomkoeff(n - 1, k) + binomkoeff(n - 1, k - 1);
  }

  void viereck(int n)
  {
    int[][] rechteck = new int[n][(2 * n) - 1];

    // Feld initialisieren
    for (int z = 0; z < n; z++)
      for (int s = 0; s < (2 * n) - 1; s++)
        rechteck[z][s] = 0;

    // oberste "Ecke" des Dreiecks mit 1 belegen
    rechteck[0][n - 1] = 1;

    // Rechenschema anwenden
    for (int z = 1; z < n; z++)
      for (int s = n - z - 1; s <= n + z; s += 2)
        // letzte Zeile gesondert behandeln!
        rechteck[z][s] =
          ((z == (n - 1) && s == 2 * n - 2) ? 0 : rechteck[z - 1][s +
              1]) +
          ((z == (n - 1) && s == 0) ? 0 : rechteck[z - 1][s - 1]);

    System.out.println();
    for (int z = 0; z < n; z++)
    {
      for (int s = 0; s < (2 * n) - 1; s++)
        if (rechteck[z][s] != 0)
          System.out.print(rechteck[z][s] + "␣␣");
      System.out.println();
    }
  }
}
```