KV Software Engineering
Übungsaufgaben SS 2005

Martin Glinz, Silvio Meier,
Nancy Merlo-Schett, Katja Gräfenhain

Übung 2

Aufgabe 1 (7 Punkte) – Systematisches Programmieren

Die folgende Java-Klasse wurde von einem Informatik-Ingenieur mit der Absicht geschrieben, dass sie möglichst wenig Speicher in Form von lokalen Variablen verbraucht. Die Klasse implementiert einen Sortieralgorithmus, welcher ein gegebenes Integer-Array zwischen dem Index 0 und j aufsteigend sortiert.

```
/*
 * Sortiert array.
 */
public class S {
  /*
   * Sortiert array, ab index j.
   */
  public void s(int[] mn, int j) {
    // Init: i mit der Länge des Arrays
    int i = j;

    while (i > 0) {
      // j iterator index
      j = i - 1;
      while (j >= 0) {
        if (mn[j] > mn[i]) {
           // Tausche Elemente, ohne eine
          // zusaetzliche lokale Variable
           mn[j] = mn[i] + mn[j]; mn[i] = mn[j] - mn[i];
           mn[j] = mn[j] - mn[i];
         } j = j - 1;
      } i = i - 1;
    }
  }
}
```

Diskutieren Sie stichwortartig dieses Programm hinsichtlich der systematischen Programmierung. Gehen Sie dabei auf die Benennung von Variablen, Methoden und Klassen ein. Behandeln Sie anschliessend die Kommentierung des Codes und mögliche Nebeneffekte durch die Programmierung. Zeigen Sie weitere Probleme auf, die Sie bzgl. der Systematik der Programmierung finden.

Aufgabe 2 (6 Punkte) – Ablaufkonstrukte

Von folgendem Programmstück wird behauptet, es würde $x = 2^n$ berechnen, wenn vor der Ausführung $z = n$ und $n > 0$ gilt, wobei die Variablen x und z vom Typ Integer sind. Beweisen Sie, dass das stimmt mit Hilfe einer Schleifeninvariante.

```
x = 1;
while (z > 0) {
        x = 2 * x;
        z = z -1;
}
```

Aufgabe 3 (7 Punkte) – Rekursion

Gegeben sei die rekursive Funktion zur Berechnung einer Integer-Potenz. Sind ein Wert x und ein nicht-negativer Integer n gegeben, soll x^n berechnet werden. Die folgende Implementierung soll diese Funktion umsetzen. Beweisen oder falsifizieren Sie die Korrektheit dieser Implementierung. Geben Sie dazu die Rekursionsformel an. Der Beweis soll nicht formal sondern mit strengen Argumenten geführt werden (Tipp: Rekursionsformel und –verankerung). Sollte die Lösung falsch sein, korrigieren Sie sie und begründen Sie die Korrektheit Ihrer Korrektur.

```
int power (int r, int n) {
      if (n==0) {
            return 1;
      }
      // n is even
      else if (n%2 == 0) {
            return power (r*r, n/2);
      }
      // n is odd
      else {
            return r * power (r, n/2);
      }
}
```