Ludwig-Maximilians-Universität München
Institut für Informatik
Prof. Dr. Christian Böhm
Annahita Oswald, Bianca Wackersreuther

München, 11.12.2009

Einführung in die Programmierung

WS 2009/10

Übungsblatt 8: Korrektheit, Komplexität von imperativen Programmen

Besprechung: 21./23.12.2009 und 07./08.01.2010

Ende der Abgabefrist: Montag, 21.12.2009 10:00 Uhr.

Hinweise zur Abgabe:
Geben Sie bitte Ihre gesammelten Lösungen zu diesem Übungsblatt in einer Datei `loesung08.zip` unter `http://www.pst.ifi.lmu.de/uniworx/` ab.
Bitte beachten Sie, dass die Aufgabe 8-2 nicht in die Bonusregelung eingeht. Bereiten Sie diese aber bitte trotzdem vor, damit Sie der Übung optimal folgen können.

Aufgabe 8-1 *Hoare-Kalkül* **20 Punkte**

Beweisen Sie mit den Mitteln des Hoare-Kalküls die partielle Korrektheit des folgenden Programmstücks. Halten Sie sich dabei *exakt* an die Notation aus dem Skript. Geben Sie Ihre Lösung in einer Datei `hoare.txt` ab.

Hinweise:
Die minimale Schleifeninvariante lautet `result = i*x && i <= y`
`x`, `y`, `i`, `result` seien vom Typ **`int`**.

```
// Vorbedingung PRE: (x >= 1 && y >= 1)
result = x;
i = 1;
while(i < y)
{
  i = i+1;
  result = result+x;
}
// Nachbedingung POST: (result = x*y)
```

Aufgabe 8-2 *Komplexität von Sortieralgorithmen* **0 Punkte**

- **SimpleSort – Sortieren durch einfaches Vertauschen**
  Gegeben ist ein (unsortiertes) Array a der Länge n. Jedes Element an Position i von a (mit $i = 0, \ldots, n-1$) wird nun nacheinander mit allen Elementen der Positionen $j = i+1, i+2, \ldots, n-1$ verglichen. Falls das Element an Position j kleiner ist als das Element an Position i, werden die beiden Elemente vertauscht.

  Implementieren Sie in einer Klasse `SimpleSort` eine Methode
  `public static void sort(int[] a),`
  die ein übergebenes Array vom Typ `int[]` nach obigem Verfahren sortiert.

- **CountSort – Sortieren durch Abzählen**
  Zum Sortieren eines Arrays a der Länge n kann man sich folgende Beobachtung zu Nutze machen: In einem sortierten Array kann die Position eines Elements durch Abzählen aller kleineren Elemente des Arrays ermittelt werden. Gibt es in einem Array $i-1$ Elemente, die kleiner sind als ein Element e, so steht dieses Element e an Position i des sortierten Arrays. Trotz dieser simplen Idee erfordert die Möglichkeit des Vorkommens gleicher Elemente ein trickreiches Vorgehen.

  Implementieren Sie in einer Klasse `CountSort` eine Methode
  `public static void sort(int[] a),`
  die ein übergebenes Array vom Typ `int[]` nach obigem Prinzip sortiert.

- **SelectionSort – Sortieren durch direktes Auswählen**
  Gegeben ist ein (unsortiertes) Array a der Länge n. Für jede Position $i = 0, \ldots, n-2$ von a wird nun nacheinander das kleinste Element im Teilarray $a_i, \ldots, a_{n-1}$ gesucht und mit dem Element an Position i vertauscht.

  Implementieren Sie in einer Klasse `SelectionSort` eine Methode
  `public static void sort(int[] a),`
  die ein übergebenes Array vom Typ `int[]` nach obigem Verfahren sortiert.

- **BubbleSort – Sortieren durch direktes Austauschen**
  Gegeben ist ein (unsortiertes) Array a der Länge n. Für $i = 0, \ldots, n-1$ werden für $j = n-1, \ldots, i+1$ paarweise die Elemente mit den Positionen $j-1$ und j miteinander verglichen. Falls das Element an Position j kleiner ist als das Element an Position $j-1$, werden die Elemente vertauscht.

  Implementieren Sie in einer Klasse `BubbleSort` eine Methode
  `public static void sort(int[] a),`
  die ein übergebenes Array vom Typ `int[]` nach obigem Verfahren sortiert.

Die Klasse `Laufzeit` ermöglicht eine empirische Untersuchung des Laufzeitverhaltens der implementierten Sortierverfahren. Sie können für die Konstanten `START`, `STOP` und `SCHRITT` auch mit anderen Werten experimentieren. Was beobachten Sie?

Die Klasse `Laufzeit` finden Sie auf der Webseite zur Vorlesung.