Ludwig-Maximilians-Universität München
Institut für Informatik
Prof. Dr. M. Wirsing, M. Hammer, A. Rauschmayer

SS 2006
Übungsblatt 4

Informatik II

Zu jeder Aufgabe ist eine Datei abzugeben, deren Name rechts in der Aufgabenüberschrift steht. Stellen Sie die Dateien in ein extra Verzeichnis (mit beliebigem Namen) und packen Sie dieses zu einem ZIP-Archiv. Geben Sie dieses, wie üblich, per UniWorx ab.

Aufgabe 4-1 Insertion Sort (`InsertionSort.java`, 14 Punkte)

- Algorithmus: *Insertion Sort* ist ein Sortier-Algorithmus, bei dem eine Sequenz u von unsortierten Zahlen in eine Sequenz s von sortierten Zahlen überführt werden soll. Am Anfang ist dabei s leer, danach werden der Reihe nach alle Elemente e von u durchlaufen und so in s eingefügt, dass s sortiert bleibt. Für diese Aufgabe wollen wir ganze Zahlen aufsteigend sortieren: Man sucht also das erste Element von s, das größer ist als e und fügt e davor ein. Ist kein Element von s größer, so wird e hinten an s gehängt. Dieser Algorithmus ist sehr nahe an dem, was ein Mensch tun würde, wenn er etwas sortieren muss.
- Implementierung: Wir wollen mit Reihungen (Arrays) arbeiten und s nicht separat halten, sondern direkt in u unterbringen (sogenanntes *in-place sorting*). Dazu betrachten wir immer den Anfang von u als sortiertes Array und fügen den Rest sortiert ein. Zu Beginn hat dieses sortierte Präfix automatisch schon ein Element (da ein einelementiges Array immer sortiert ist). Das nächste einzufügende Element steht immer direkt hinter dem Präfix. Zu Veranschaulichung: Sei u ein Array mit n Elementen e_i.

  $$\underbrace{e_0, e_1, \ldots, e_{len-1}}_{\text{bereits sortiert}}, \underbrace{e_{len}, \ldots, e_{n-1}}_{\text{unsortiert}}$$

  e_{len} ist das nächste einzusortierende Element. u ist sortiert in dem Moment, wo das Präfix das gesamte u ausfüllt, es ist dann $len = n$.
- Hinweis: Sie können die `break`-Anweisung verwenden, um eine Schleife vorzeitig zu verlassen. `break` steht z.B. im Then-Fall einer `if`-Anweisung und führt oft zu lesbarerem Code, als wenn man das entsprechende Abbruchkriterium noch zusätzlich in die Schleifenbedingung packt (beispielsweise die dritte Stelle der `for`-Schleife).

```
public class BreakExample {
    public static void main(String[] args) {
        int[] myInts = { 20, 3, 2, 0, 5, 1};
        // Variante 1
        System.out.println("1. Alle Zahlen vor der ersten 0:");
        for(int i=0; i < myInts.length; i++) {
            if (myInts[i] == 0) {
                break;
            }
            System.out.println(myInts[i]);
        }
        // Variante 2
        System.out.println("2. Alle Zahlen vor der ersten 0:");
        for(int myInt : myInts) {
            if (myInt == 0) {
                break;
            }
            System.out.println(myInt);
        }
    }
}
```

Teilaufgaben:

a) Verwenden Sie das untenstehende Gerüst zur Implementierung der Prozeduren (statischen Methoden) `sort` und `insert` und versehen Sie diese Methoden ausserdem mit einem geeigneten JavaDoc-Kommentar.

```
import java.util.Arrays;

public class InsertionSortSkeleton {

    public static void sort(int[] numbers) {
        // Zu implementieren
    }

    private static void insert(int[] numbers, int index, int element, int len) {
        // Zu implementieren
    }

    public static void main(String[] args) {
        int[] numbers = { 3, 4, 3, 2, 1 };
        // Arrays.toString hilft beim Ausgeben von Arrays:
        System.out.println("Davor: " + Arrays.toString(numbers));
        sort(numbers);
        System.out.println("Danach: " + Arrays.toString(numbers));
    }
}
```

b) Fügen Sie zu den Prozeduren sinnvolle Vor- und Nachbedingungen hinzu:

- Prozedur `sort`: Eine Nachbedingung.
- Prozedur `insert`: Eine Vorbedingung.

Hinweise:

- Schreiben Sie die Vor- und Nachbedingungen in den Java-Code, mit der `assert`-Anweisung.
- Sie können beliebig aus dem Quellcode der Vorlesung abschreiben.
- Dies ist keine formale Aufgabe, es geht vielmehr darum, ein pragmatisches Gefühl für Vor- und Nachbedingungen zu entwickeln.

c) Welche Zeitkomplexität hat der Insertion Sort? Begründen Sie Ihre Antwort kurz informell.

Aufgabe 4-2 Matrizen transponieren (`MatrixTransposition.java`, 8 Punkte)

Schreiben Sie ein Programm, das Matrizen transponiert. Erweitern Sie hierzu das folgende Skelett:

```
import java.util.Arrays;

public class MatrixTranspositionSkeleton {

    public static int[][] transpose(int[][] matrix) {
        // Zu implementieren
    }

    public static void printMatrix(int[][] matrix) {
        // Zu implementieren
    }

    public static void main(String[] args) {
```

```
        int[][] matrix = {{1,2,3},{4,5,6}};
        System.out.println("Vor Transposition:");
        printMatrix(matrix);
        System.out.println("Nach Transposition:");
        printMatrix(transpose(matrix));
    }
}
```

Zu beachten:

- Implementieren Sie Matrizen als geschachtelte Reihungen.
- Welche Art von zweidimensionaler Reihung kann überhaupt sinnvoll gedreht werden? Schreiben Sie hierzu eine geeignete Vorbedingung, die ggf. eine Hilfsprozedur verwendet.
- Bei der Prozedur `printMatrix` kann ihnen die statische Methode `toString(int[])` in der Klasse `java.util.Arrays` helfen. Im Prinzip könnten Sie auch `deepToString(int[])` einsetzen, aber damit stehen die Zeilen der Matrix nicht untereinander.

Aufgabe 4-3 **Hoare-Tripel** (`HoareTripel.txt`, 8 Punkte)

Für welche der folgenden Spezifikationen sind *alle*, *bestimmte* oder *gar keine* Programme `P` *partiell* oder *total* korrekt? Geben Sie für jedes Tripel die Menge der Programme `P` an, die partiell bzw. total korrekt sind. Gilt die Korrektheit nur für bestimmte Programme, geben Sie ein Beispiel eines korrekten und eines inkorrekten Programms an!

a) {**true**} `P` {**true**}

b) {**false**} `P` {**true**}

c) {**true**} `P` {**false**}

d) $\{\phi\}$ `P` $\{\phi\}$ für eine beliebige Formel ϕ

Abgabe: Per UniWorx, bis spätestens Montag, den 29.5.2006 um 9:00 Uhr.