Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
Institut für Simulation und Graphik
Prof. Dr. Klaus-Dietz Tönnies

Magdeburg, 16.01.03

Übungsklausur zur Vorlesung

"Einführung/Algorithmen und Datenstrukturen"

Aufg. 1	Aufg. 2	Aufg. 3	Summe
(von 11)	(von 8)	(von 8)	(von 27)

Blattzahl:

Unterschrift:

Allgemeine Hinweise
Schreiben Sie auf jedes Blatt Ihren Namen, Ihre Matrikelnummer und eine Seitennummer. Die Programme sind gut zu kommentieren! Beginn und Ende der Lösung zu einer Aufgabe sind durch einen waagerechten Strich deutlich zu kennzeichnen.

Zugelassene Hilfsmittel: nur Schreibmaterial
Nicht zugelassene Hilfsmittel: Mitschriften, Bücher, Skript, Taschenrechner, Handy

Zum Bestehen der Klausur sind 13 Punkte erforderlich.

Aufgabe 1 (11 Punkte)
Komplexe Zahlen sind Zahlen mit einem Realteil und einem Imaginärteil. Eine Komplexe Zahl k wird durch $k = (a+i*b)$ angegeben. „*a*“ ist der Realteil und „*b*“ der Imaginärteil, „*i*“ ist eine Konstante mit $i^2 = -1$. Folgende Rechenregeln gelten für komplexe Zahlen $k_1=a_1+i*b_1$, $k_2=a_2+i*b_2$:

Betrag: $|k_1| = sqrt(a_1^2+b_1^2)$ (“sqrt” ist die Quadratwurzel)
Multiplikation: $k_1*k_2 = (a_1*a_2-b_1*b_2) + i*(a_1*b_2+a_2*b_1)$

a. Definieren Sie das Interface für einen abstrakten Datentyp “Komplexe Zahl”, der es erlaubt, komplexe Zahlen zu multiplizieren sowie den Betrag einer komplexen Zahl zu bilden.
b. Schreiben Sie eine Klasse, die dieses Interface durch den Datentyp **double** implementiert und neben einem geeigneten Konstruktor und den genannten Methoden auch noch das Quadrat der komplexen Zahl bilden kann.
c. Erweitern Sie die Klasse um eine Methode *wurzel()*, die dann die Quadratwurzel der komplexen Zahl zurückgibt, wenn der Imaginärteil Null ist. Andernfalls soll eine Ausnahmebedingung „InvalidInputForSquareRoot“ ausgelöst werden.

Aufgabe 2 (8 Punkte)

a. Erklären Sie kurz das Prinzip des binären Suchens in einer sortierten Folge *F[0..m-1]* von ganzen Zahlen.
b. Geben Sie eine *rekursive **und*** eine *iterative* Java-Methode zum binären Suchen eines Elementes in *F* an. Der Rückgabewert der Methode ist die Position des ersten gefundenen Elementes oder *-1*, falls das Element nicht gefunden wurde. Die Elemente seien ganzzahlig.

Aufgabe 3 (8 Punkte)

a. Entwickeln Sie in Pseudocode eine Methode *reverse()* auf einer einfach verketteten Liste, die die Referenzen zwischen Listenelementen so ändert, dass jeweils der ehemalige Nachfolgeknoten auf seinen ehemaligen Vorgängerknoten zeigt (d.h., die Reihenfolge der Listenelemente umdreht).
   Ein Ihnen zur Verfügung stehender Datentyp *EList* (einfach verkettete Liste) stellt Ihnen dazu das folgende Interface mit den aus der Vorlesung bekannten Bedeutungen der Methoden zur Verfügung:

```
public interface EList {
        public int size();
        public boolean isEmpty();
        public Position first();
        public void setFirst(Position p);
        public boolean isFirst(Position p);
        public Position after(Position p);
        public Object getElement(Position p);
}
```

   Weiterhin steht die Klasse *Node* zur Verfügung:

```
public class Node implements Position{
        private Object element;
        private Node next;
        …
        public Node setNext(Node n) { next = n;}
}
```

b. Schreiben Sie in Pseudocode eine Methode *before(p)*, die den Vorgängerknoten von *p* für die oben genannte einfach verkettete Liste zurückgibt.
c. Berechnen und begründen Sie den Aufwand für *before(p).*