Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer

Kennzahl: _____

Kennwort: _____

Arbeitsplatz-Nr.: _____

Frühjahr

2001

46114

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

- Prüfungsaufgaben -

Fach: **Informatik (nicht vertieft studiert)**

Einzelprüfung: **Algorithmen/Datenstrukt./Progr.-meth.**

Anzahl der gestellten Themen (Aufgaben): 2

Anzahl der Druckseiten dieser Vorlage: 6

Bitte wenden!

Thema Nr. 1

Vorbemerkung:

Die in der folgenden Aufgabe geforderten Programmierungsteile dürfen Sie wahlweise entweder in *Java* oder in *Pascal* ausführen. Dabei soll die Kartei, auf die sich die Programme beziehen, als ein globaler Parameter behandelt werden.

Aufgabe:

Es soll eine einfache elektronische Bibliothekskartei erstellt und mit den wichtigsten Funktionen ausgestattet werden. Für jedes Buch sind dabei die folgenden vier Daten vorzusehen:

- Familienname des Autors
- Vorname des Autors
- Buchtitel
- Signatur für den Aufbewahrungsort

Alle Daten sollen als *strings* dargestellt werden. Wenn Sie mit *Pascal* arbeiten, dürfen Sie annehmen, dass für diese Daten jeweils 50 Zeichen genügen; in *Java* ist hierfür keine Einschränkung erforderlich.

Die Zahl der in der Kartei jeweils erfassten Titel sei n.

Die Einträge in der Kartei seien eindeutig durch die in ihr erfassten Buchtitel, d.h. zu einem Buchtitel sei jeweils höchstens ein Eintrag in der Kartei enthalten.

Im Einzelnen sind folgende Teilaufgaben zu bearbeiten:

1. Beschreiben Sie genau, mit welcher Datenstruktur Sie die Kartei darstellen werden, wenn der Zugriff primär über den Familiennamen des Autors erfolgen soll, und zwar mit einer Komplexität, deren Ordnung im allgemeinen $O(\log n)$ und nur im schlechtesten Fall $O(n)$ sei! Beachten Sie dabei aber, dass der Name eines Autors im Allgemeinen mehrfach auftritt und ein Zugriff nur in Verbindung mit dem Buchtitel eindeutig ist (s.o.)!

2. Programmieren Sie einschließlich der für die Aufgabe erforderlichen Datenstruktur (nach 1.) die folgenden Prozeduren (*Pascal*) bzw. Methoden (*Java*):

   a) Eine Prozedur (*Pascal*) bzw. eine Instanz-Methode (*Java*) ***eintragen*** mit 4 *string*-Parametern für den Familiennamen des Autors, den Vornamen des Autors, den Buchtitel und die Signatur. Falls dieser Titel in der Kartei noch nicht enthalten ist, soll er eingefügt werden. Andernfalls bleibt die Kartei unverändert.

Fortsetzung nächste Seite!

b) Eine Funktionsprozedur (*Pascal*) bzw. eine Instanz-Methode (*Java*) ***suche*** mit 2 *string*-Parametern für den Familiennamen des Autors und den Buchtitel. Als Ergebnis soll die zugehörige Signatur geliefert werden, wenn das gesuchte Buch in der Kartei erfasst ist. Andernfalls soll das Ergebnis der string *„nicht vorhanden"* sein.

3. Die Datenstruktur soll nun so ergänzt werden, dass auch ein Zugriff über den Buchtitel allein, der ja eindeutig ist, möglich wird. Auch für die Komplexitätsordnung dieses sekundären Zugriffs wird gefordert, dass sie im Allgemeinen *O(log n)* und nur im schlechtesten Fall *O(n)* sei. Beschreiben Sie genau, wie Sie diese ergänzende Datenstruktur darstellen!

4. Beschreiben Sie genau, wie das Programm ***eintragen*** (2.a) erweitert werden muss, damit die durch die sekundäre Zugriffsstruktur ergänzte Datenstruktur jederzeit konsistent bleibt!

5. Programmieren Sie die in Teilaufgabe 4 beschriebene Erweiterung für die Prozedur (*Pascal*) bzw. die Instanz-Methode (*Java*) ***eintragen***!

Thema Nr. 2

Aufgabe 1: *(Systementwurf und Programmiermethodik)*

Eine Galaxie besteht aus mehreren Himmelskörpern. Es gibt hier nur zwei Arten von Himmelskörpern: Sterne und Planeten. Jeder Himmelskörper wird durch einen Namen (als Zeichenkette) identifiziert. Für Sterne wird zusätzlich ihre Helligkeitsstufe (durch eine ganze Zahl gegeben) gespeichert, für Planeten wird zusätzlich die Eigenschaft gespeichert, ob es dort Wasser gibt. Ein Planet kreist direkt um genau einen Himmelskörper und „kennt" nur den Himmelskörper, um den er direkt kreist. Jeder Planet kreist indirekt um genau einen Stern, d.h. dass für jeden Planeten die transitive Hülle der „Umkreist"-Relation immer genau einen Stern enthält.

a) Geben Sie ein passendes Klassendiagramm zur Modellierung von Galaxien an, mit den entsprechenden Klassen (d.h. `Galaxie`, `Himmelskoerper`, `Stern`, `Planet`), Aggregationen, Assoziationen und ihren Multiplizitäten!

b) Übersetzen Sie das Klassendiagramm nach Java!

c) Implementieren Sie die folgende Methode in der Klasse `Planet`!

```
public String starName() {...}
/** gibt einen String mit dem Namen des Sterns zurück, um den der Planet
(this) kreist. */
```

Fortsetzung nächste Seite!

Aufgabe 2: *(Datenstrukturen und Algorithmen)*

Die folgende Java-Klasse `ListIntElem` beschreibt die Struktur der Listenelemente einer verketteten Liste von ganzen Zahlen:

```
public class ListIntElem
{ private int data;
  private ListIntElem next;

  public int getData()
  { return data;
  }

  public ListIntElem getNext()
  { return next;
  }

  public void chainTo(ListIntElem o)
  { next = o;
  }
}
```

Die folgende unvollständige Java-Klasse soll einfach verkettete Listen implementieren:

```
public class LinkedIntList
{ private ListIntElem head;

  public int max()
  {
    ...
  }

  public void reduce()
  {  ...
  }

  public LinkedIntList reverse()
  {  ...
  }
}
```

a) Implementieren Sie die Methode `max`, die für jede nicht leere Liste `l` (d.h. `l.head != null`) die größte Zahl findet, die Attributwert eines Elements von `l` ist!

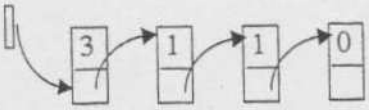
b) Eine Liste heißt aufsteigend geordnet, wenn jedes Element kleiner oder gleich seinem Nachfolger ist (sofern es den Nachfolger gibt). Implementieren Sie die Methode `reduce`, die aus einer aufsteigend geordneten Liste alle Wiederholungen entfernt!

Fortsetzung nächste Seite!

c) Implementieren Sie die Methode `reverse`, die eine Liste umdreht, d.h. eine neue Liste erzeugt, deren Elemente in umgekehrter Reihenfolge angeordnet sind!

Beispiel: Für ein Argument der Form

liefert die Methode `reverse`:



Aufgabe 3: *(Zeitkomplexität)*

Die folgende Java-Methode sortiert einen Array von ganzen Zahlen, indem durch Vertauschen benachbarter Elemente die jeweils größeren nach hinten wandern.

```
static void sort(int[] a)
{
  boolean stop = false;          //fuer den Abbruch der aeusseren Schleife
  int bound = a.length-1;          //obere Grenze fuer die innere Schleife

  while (!stop & bound>0)
  {
    stop = true;                 //stop==true bedeutet, dass die innere Schleife
                                 //(noch) keine Elemente vertauscht hat
    for (int j=0;j<bound;j++)
    {
      if (a[j] > a[j+1])         //a[j] ist groesser als a[j+1]
                                   //also vertausche
      { int h = a[j]; a[j] = a[j+1]; a[j+1] = h;
        stop = false;
      }
    }
    bound--;
  }
}
```

a) Erklären Sie die Begriffe von Best-Case- und Worst-Case-Zeitkomplexität! Erklären Sie die O-Notation, die benutzt wird, um die Zeitkomplexität von Algorithmen qualitativ zu erfassen!

Geben Sie in den folgenden Teilaufgaben b) und c) jeweils eine kurze Begründung an!

b) Für welche Eingaben tritt der Best-Case auf? Welche Best-Case-Zeitkomplexität hat der Algorithmus?

c) Für welche Eingaben tritt der Worst-Case auf? Welche Worst-Case-Zeitkomplexität hat der Algorithmus?

Fortsetzung nächste Seite!

Aufgabe 4: *(Programmierung)*

Programmieren Sie in einer von Ihnen gewählten Programmiersprache eine Funktion `potenz5`, die überprüft, ob eine natürliche Zahl $i \in N$ Potenz von 5 ist, d.h. ob es ein $x \in N$ gibt, so dass $5^x = i$. Geben Sie an, welche Programmiersprache Sie gewählt haben!