Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer

Kennzahl: _____

Kennwort: _____

Arbeitsplatz-Nr.: _____

Frühjahr

2005

46114

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

- Prüfungsaufgaben -

Fach: **Informatik (Unterrichtsfach)**

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

Anzahl der gestellten Themen (Aufgaben): 2

Anzahl der Druckseiten dieser Vorlage: 7

Bitte wenden!

Thema Nr. 1

Aufgabe 1: Verwaltung von Schülerinnen und Schülern (80 Minuten)

Es soll ein Programm zur Verwaltung von Schülerinnen und Schülern entwickelt werden. Dazu soll die Klasse Schüler erstellt werden, die folgende Informationen speichert:

- Name des Schülers
- Vorname des Schülers
- Geschlecht des Schülers (m, w)
- Liste mit erzielten Noten dieses Schülers
- Laufende Nummer des Schülers

Die Klasse soll einen Konstruktor besitzen, der ein neues Objekt anlegt. Dem Konstruktor sollen der Vor- und Nachname des Schülers sowie dessen Geschlecht übergeben werden. Die laufende Nummer soll vom Programm selbst vergeben werden. Dabei muss darauf geachtet werden, dass kein Schüler eine bereits vergebene Nummer erhält.

Außerdem soll die Klasse über eine Methode zum Hinzufügen einer Note sowie zur Ausgabe des Objekts verfügen. Letztere soll zunächst die Durchschnittsnote des Schülers berechnen und die relevanten Informationen in einer Zeile an die Standardausgabe übergeben. Das könnte so aussehen:

Beispiel 1:

22 Hopper, Grace (w) 1,3

Beispiel 2:

23 Zuse, Konrad (m) 1,3

a. Spezifizieren Sie diese Klasse, indem Sie die Signaturen des Konstruktors und der Methode in einer Programmiersprache Ihrer Wahl angeben. Überlegen Sie sich außerdem, welche Werte für die jeweiligen Parameter zulässig sind, bzw. bei welchen Parametern eine Ausnahme generiert werden soll. (20 Minuten)

b. Implementieren Sie diese Klasse. Achten Sie darauf, dass die Implementierung zu der von Ihnen erstellten Spezifikation aus Teilaufgabe a. passt. Insbesondere sollen ungültige Parameter erkannt und mit einer Ausnahme behandelt werden. (20 Minuten)

Des Weiteren soll eine Klasse Schülerliste implementiert werden. Diese soll über einen Konstruktur verfügen, der eine leere Liste generiert. Außerdem sollen zwei Methoden implementiert werden, mit denen der Liste ein Schüler hinzugefügt bzw. ein Schüler aus der Liste entfernt werden kann. Beim Entfernen soll der entsprechende Schüler durch seine laufende Nummer spezifiziert werden. Eine weitere Methode soll die Schülerliste auf der Standardausgabe ausgeben. Dazu soll die Liste zunächst bzgl. Vor- und Nachname sortiert werden.

Fortsetzung nächste Seite!

c. Spezifizieren Sie die Klasse Schülerliste (analog a)! (10 Minuten)

d. Implementieren Sie die Klasse entsprechend Ihrer Spezifikation. Greifen Sie bei der Implementierung der Ausgabe auf einen Sortieralgorithmus Ihrer Wahl zurück, den Sie als bereits implementierte Methode verwenden können. (30 Minuten)

Aufgabe 2: Maximum eines Feldes (50 Minuten)

Gegeben sei ein Feld von Ganzzahlen der Länge n. Gesucht ist die Position des größten Elements in diesem Feld. Für diese Aufgabe sollen zwei Algorithmen entwickelt und anschließend miteinander verglichen werden.

a. Entwickeln Sie zunächst die Funktion maxIterativ, welche die Aufgabe mit einem iterativen Programmieransatz löst. (10 Minuten)

b. Entwickeln Sie die Funktion maxRekursiv, welche die Aufgabe mittels einer rekursiven Funktion löst, die das Feld in jedem Rekursionsschritt in zwei Hälften teilt, jeweils das Maximum der beiden Hälften errechnet und anschließend das Maximum der beiden Ergebnisse bildet. Überlegen Sie sich zunächst ein geeignetes Rekursionsende. (20 Minuten)

c. Entwickeln Sie ein Hauptprogramm, das
- solange Werte von der Standardeingabe liest, bis das Eingabeende erreicht wird,
- diese in einem Feld ablegt,
- das Feld mit Hilfe der obigen Funktion sortiert und
- das sortierte Feld über die Standardausgabe ausgibt.

Hinweis: Sie dürfen auf die Teilergebnisse a) und b) auch dann zurückgreifen, wenn Sie diese Aufgaben noch nicht vollständig gelöst haben. (10 Minuten)

d. Vergleichen Sie Ihre Lösungen aus Aufgabe a) und b) bezüglich
- Speicherkomplexität und
- Zeitkomplexität

Wie sinnvoll ist in diesem Zusammenhang die Verwendung des Landau-Symbols O? (10 Minuten)

Fortsetzung nächste Seite!

Aufgabe 3: Verkette Listen (50 Minuten)

Im Folgenden soll die flexible Datenstruktur List in einer objektorientierten Sprache Ihrer Wahl implementiert werden. Die Klasse soll Elemente vom abstrakten Datentyp DATA speichern können und dabei so flexibel sein, dass List-Objekte sowohl FIFO-Listen (First in First Out bzw. Warteschlangen) als auch LIFO-Listen (Last In First Out bzw. Keller) sein können.
Implementieren Sie eine solche Klasse mit den folgenden Konstruktoren und Methoden:

(Lesen und Planen: 20 Minuten)

Konstruktor:	Erzeugt eine leere Liste. Als Argument wird übergeben, ob es sich um eine Warteschlange oder einen Keller handeln soll. (5 Minuten)
Method add:	Fügt der Liste ein Objekt hinzu. (15 Minuten)
Method remove:	Entfernt ein Objekt und liefert dieses als Rückgabewert. Je nach Art der Liste (FIFO bzw. LIFO) wird das „älteste" bzw. „jüngste" Objekt in der Liste zurückgegeben. (10 Minuten)

Achten Sie bei der Implementierung darauf, dass die Zeitkomplexität des Konstruktors und der beiden Methoden in allen Fällen unabhängig von der Länge der Liste ist. Die Komplexität soll also in allen Fällen zur Klasse O (1) gehören.

Thema Nr. 2

Aufgabe 1: Baumdarstellung arithmetischer Ausdrücke

Arithmetische Ausdrücke in Infixnotation mit den binären Operatoren $+, -, *, /$ und dem unären Operator $-$ und Variablen oder Konstanten als Operanden können induktiv wie folgt definiert werden:
Basis: Einzelne Variablen und Konstanten sind arithmetische Ausdrücke.
Induktion: Wenn E_1 und E_2 arithmetische Ausdrücke sind, so sind auch

$$(E_1 + E_2),\ (E_1 - E_2),\ (E_1 * E_2),\ (E_1 / E_2) \text{ und } (-E_1)$$

arithmetische Ausdrücke.

Entsprechend dieser Definition kann ein arithmetischer Ausdruck durch einen Ausdrucksbaum dargestellt werden, dessen Blätter einzelne Variablen oder Konstanten sind und dessen innere Knoten die jeweiligen Teilausdrücke repräsentieren.

a) Geben Sie den Ausdrucksbaum zum arithmetischen Ausdruck $(y * (-(x + 9)))$ an!
b) Definieren Sie in einer Programmiersprache Ihrer Wahl eine Datenstruktur zur Darstellung arithmetischer Ausdrücke, die zu jedem Ausdruck den Operator und die Teilausdrücke, auf die der Operator angewendet wird, angibt.
c) Spezifizieren Sie in einer Programmiersprache Ihrer Wahl eine rekursive Funktion inorder(), die aus einem gegebenen Ausdrucksbaum, der als Parameter übergeben wird, den zugehörigen arithmetischen Ausdruck in Infixnotation ausgibt.

Achten Sie in allen Teilaufgaben auf eine korrekte Klammerung der Teilausdrücke!

Aufgabe 2: Warteschlangen als abstrakter Datentyp

Eine Warteschlange ist eine lineare Datenstruktur zur Ablage von Elementen des gleichen Typs mit der Eigenschaft, dass nur am Ende der Datenstruktur Elemente eingefügt und nur am Anfang der Datenstruktur Elemente entnommen werden können (FIFO-Prinzip). Dementsprechend stehen die folgenden beiden elementaren Operationen zur Verfügung:

enqueue (e): Einfügen eines Elementes e am Ende der Warteschlange.
e=dequeue(): Entnahme eines Elementes e am Anfang der Warteschlange (falls vorhanden).

Gehen Sie dabei davon aus, dass jedes Warteschlangenelement einen einzelnen Integerwert enthält.

a) Realisierung von Warteschlangen mit einfach verketteten Listen in einer imperativen Programmiersprache Ihrer Wahl:

   α) Definieren Sie eine Knoten-Datenstruktur zur Verwendung in der Listenrealisierung.
   β) Geben Sie eine Funktion `make_element()` an, die einen Knoten der Liste mit zugehörigem Integerwert, der als Parameter übergeben wird, anlegt.

Fortsetzung nächste Seite!

γ) Geben Sie Funktionen zur Realisierung der beiden elementaren Funktionen `enqueue()` und `dequeue()` an. Geben Sie beim Entnahmeversuch aus einer leeren Warteschlange eine geeignete Fehlermeldung aus. Verwenden Sie zur Realisierung von `enqueue()` die Funktion `make_element()`.

b) Realisierung einer Warteschlange mit Hilfe eines Feldes fester Länge in einer imperativen Programmiersprache Ihrer Wahl.
Verwenden Sie für die Realisierung zwei globale Feldindices:
- der Index `head` gibt das erste belegte Element der Warteschlange an;
- der Index `tail` gibt die Feldposition an, an der das nächste einzufügende Element abgelegt wird.

Verwenden Sie für eine effiziente Ausnutzung des Feldes eine zirkuläre Implementierung (wraparound), die beim Erreichen des Feldendes zum Einfügen neuer Elemente Feldpositionen am Anfang des Feldes verwendet, wenn dort durch Entfernen von Warteschlangenelementen freie Feldpositionen entstanden sind.

α) Spezifizieren Sie die verwendeten Datenstrukturen und geben Sie eine Funktion `queue_init()` zur Initialisierung der Warteschlange an.

β) Geben Sie Funktionen zur Realisierung von `enqueue()` und `dequeue()` an und beachten Sie dabei jeweils die Möglichkeit, dass durch die Aktualisierung von `head` und `tail` das Feldende erreicht werden kann.
Geben Sie geeignete Fehlermeldungen aus, wenn `enqueue` auf eine volle Liste angewendet wird und wenn `dequeue` auf eine leere Liste angewendet wird.

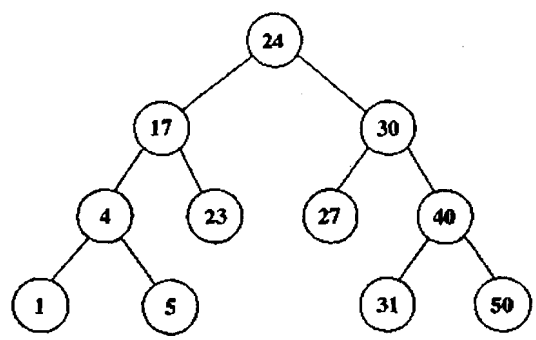
c) Diskutieren Sie Vor- und Nachteile der Realisierung von Warteschlangen mit einfach verketteten Listen und Feldern fester Länge!

Fortsetzung nächste Seite!

Aufgabe 3 (binäre Suchbäume)

Ein (unbalancierter) binärer Suchbaum ist eine dynamische Datenstruktur zur knotenorientierten Ablage von Daten mit Suchschlüsseln, welche zum Sortieren der Einträge verwendet werden. Als Suchschlüssel sollen Integerwerte verwendet werden. Jeder Knoten x enthält neben den Daten einen Suchschlüssel und einen Verweis auf den linken und den rechten Unterbaum des Knotens. Die Knoten sollen dabei so angeordnet sein, dass alle Knoten im linken Unterbaum von x Suchschlüssel besitzen, die kleiner als der Suchschlüssel von x sind, und dass alle Knoten im rechten Unterbaum von x Suchschlüssel besitzen, die größer als der Suchschlüssel von x sind. Jeder Suchschlüssel soll nur einmal in dem Suchbaum auftreten. Verwenden Sie für die folgende Realisierung eine imperative Programmiersprache Ihrer Wahl.

a) Definieren Sie eine Datenstruktur zur Realisierung eines Knotens des Suchbaumes mit Daten, Suchschlüssel und Verweisen auf den linken bzw. rechten Unterbaum.
b) Geben Sie eine rekursive Funktion `tree_lookup()` an, die für einen als Parameter übergebenen Suchschlüssel den zugehörigen Eintrag in einem Suchbaum zurückliefert.
c) Geben Sie eine rekursive Funktion `tree_insert()` an, die Daten mit einem gegebenen Suchschlüssel in einen Suchbaum als neuen Eintrag einfügt.
d) Löschen Sie aus folgendem binären Suchbaum, in dem nur die Suchschlüssel angegeben sind, nacheinander die Elemente mit Suchschlüssel 5,4,24. Geben Sie die nach jedem Löschvorgang resultierenden Suchbäume an.



e) Geben Sie eine rekursive Funktion `delete_min()` an, die aus einem Unterbaum, dessen Wurzel als Parameter übergeben wird, den Eintrag mit dem kleinsten Suchschlüssel löscht!
f) Geben Sie eine rekursive Funktion `delete()` an, die einen über den Suchschlüssel spezifizierten Eintrag aus einem Suchbaum löscht. Verwenden Sie dazu die Funktion `delete_min()` aus Aufgabenteil e)!