Klausur Allgemeine Informatik II

SS 1999

10. Juli 1999

Name ______________________________

Vorname ______________________________

Matrikel-Nr. ______________________________

Studiengang ______________________________

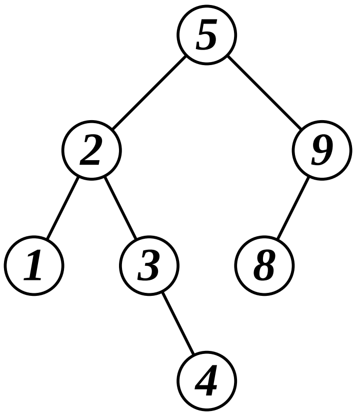
Semester ______________________________

Aufgabe		Punkte	Bewertung
1	(binäre Bäume)	18	
2	(Ring)	18	
3	(EBNF)	12	
4	(Rekursion)	18	
5	(Hash-Verfahren)	16	
6	(Typ-Erweiterungen)	18	
Summe		**100**	

Diese Klausur dient der Erlangung eines Übungsscheines. Die Vordiploms-Prüfung für Elektrotechnik findet an einem anderen Termin statt.

Aufgabe 1 (18 Punkte)

Gegeben sei folgender binärer Baum, bei dem ganze Zahlen als Schlüssel dienen:



(*a*) Bitte kreuzen Sie das jeweils zutreffende an:

Aussage	Korrekt	Falsch
Der Baum ist bezüglich der Höhe ausgeglichen.		
Der Baum ist nach dem Gewicht ausgeglichen.		
Der Baum ist sortiert.		

(*b*) Wie würde der Baum aussehen, wenn der Knoten mit der Nummer 2 gelöscht werden würde entsprechend der üblichen Löschprozedur für sortierte binäre Bäume?

(*c*) Zeichnen Sie auf, wie der vorgegebene Baum aussehen würde, wenn ein Element mit dem Schlüssel 7 hinzukommen würde.

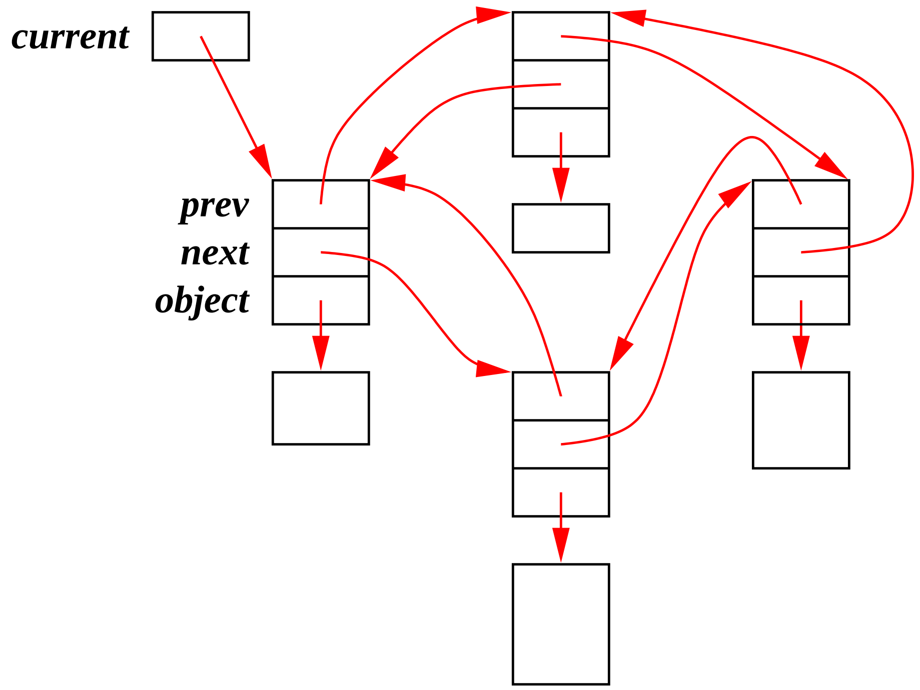
(*d*) Geben Sie die Schlüssel des Baumes in Postorder an.

Aufgabe 2 (18 Punkte)

Gegeben sei die Datenstruktur eines Ringes:

```
TYPE
   Element = POINTER TO ElementRec;
   ElementRec =
      RECORD
         prev, next: Element;
         object: Objects.Object;
      END;
   Ring =
      RECORD
         current: Element;
      END;
```

Bei einem Ring sind alle Elemente ringförmig angeordnet. Im Unterschied zur doppelt verketteten Liste gilt somit für alle Elemente, daß sowohl *prev* und *next* jeweils ungleich **NIL** sind:



Bei einem Ring ohne Elemente wäre *current* gleich **NIL**.

Schreiben Sie eine Prozedur **Insert**, die ein neues Element an irgendeiner Stelle in den Ring einfügt:

```
PROCEDURE Insert(VAR ring: Ring; object: Objects.Object);
```

Hinweise: Sie können das neue Element vor *ring.current* oder hinter *ring.current* einfügen. Achten Sie aber darauf, daß *ring.current* auch **NIL** sein könnte.

Aufgabe 3 (12 Punkte)

Gegeben sei folgende Grammatik in EBNF mit *Expression* als Startsymbol:

```
Expression = List | Atom .
Atom = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" .
List = "(" [ ExpressionList ] ")" .
ExpressionList = Expression { "," Expression } .
```

Stellen Sie bei folgenden Teilaufgaben jeweils fest, ob es sich um einen Satz dieser Grammatik handelt. Wenn ja, sollten Sie eine mögliche Ableitung angeben. Leerzeichen sind dabei zu ignorieren.

(*a*) `1`

(*b*) `()`

(*c*) `(1)`

(*d*) `(1 2)`

(*e*) `(1, 2)`

(*f*) `(1, 2), 3`

Aufgabe 4 (18 Punkte)

Für den größten gemeinsamen Teiler gilt folgende rekursive Beziehung für $m, n \geq 0$:

- $ggt(0, n) = n$
- $ggt(m, n) = ggt(n \text{ mod } m, m)$, falls $m > 0$

(*a*) Schreiben Sie eine rekursive Prozedur *GGT*, die direkt auf der angeführten Beziehung beruht.

(*b*) Ersetzen Sie die *tail recursion* in diesem Beispiel durch eine Schleife und formulieren Sie entsprechend eine iterative Version von *GGT*.

Aufgabe 5 (16 Punkte)

(*a*) Bitte kreuzen Sie die Eigenschaften an, die für eine gute Hash-Funktion wichtig sind und realistischerweise auch erwartet werden können:

Stetigkeit	
Surjektivität	
Injektivität	
Effiziente Berechenbarkeit	
Nicht-Determinismus	

(*b*) Für einen Geburtstagskalender haben Sie folgenden Datentyp definiert:

```
TYPE
   BirthDate = POINTER TO BirthDateRec;
   BirthDateRec =
      RECORD
         (Objects.ObjectRec)
         day: INTEGER; (* 1..31 *)
         month: INTEGER; (* 1..12 *)
         year: INTEGER; (* e.g. 1999 *)
         name: ARRAY 80 OF CHAR;
      END;
```

Geben Sie eine (gute!) Hash-Funktion für diesen Datentyp, die nur das Datum berücksichtigt (also Tag, Monat und Jahr) und eine (von der Tabellengröße unabhängige) ganze Zahl zurückliefert.

Aufgabe 6 (18 Punkte)

Gegeben seien folgende Typdefinitionen:

```
TYPE
   Partner = POINTER TO PartnerRec;
   PartnerRec = RECORD name, address, phone: ARRAY 80 OF CHAR END;
   Person = POINTER TO PersonRec;
   PersonRec = RECORD (PartnerRec) title: ARRAY 12 OF CHAR END;
   Employee = POINTER TO EmployeeRec;
   EmployeeRec = RECORD (PersonRec) office, department: ARRAY 16 OF CHAR END;
   Company = POINTER TO CompanyRec;
   CompanyRec = RECORD (PartnerRec) head: Employee END;
```

(*a*) Zeichnen Sie ein Diagramm, das die Typ-Hierarchie widerspiegelt.

(*b*) Bitte kreuzen Sie jeweils das Zutreffende an:

Aussage	Korrekt	Falsch
Employee ist eine Erweiterung von *Partner*		
Company ist eine Erweiterung von *Person*		
Person ist eine direkte Erweiterung von *Partner*		
Partner ist eine Erweiterung von *Company*		

Hinzu kommen folgende Deklarationen:

```
VAR partner: Partner; person: Person; employee: Employee; company: Company;

PROCEDURE Assign(VAR ptr: Partner; partner: Partner);
BEGIN
   ptr := partner;
END Assign;
```

Geben Sie bei folgenden Teilaufgaben jeweils an, ob der jeweilige Programmtext vom Compiler akzeptiert wird und ob er zur Laufzeit fehlerfrei durchläuft. Begründen Sie Ihre Angaben jeweils durch die Angaben der statischen und dynamischen Typen von *ptr* und *partner* innerhalb von *Assign*.

(*c*)

```
NEW(partner); NEW(employee); Assign(partner, employee);
```

(*d*)

```
NEW(company); NEW(person); Assign(company, person);
```

(*e*)

```
NEW(employee); NEW(person); Assign(employee, person);
```