Vordiplom für Wirtschaftswissenschaften
Allgemeine Informatik II – SS 2001
1. August 2001
Bearbeitungszeit: 120 Minuten

Name:	
Vorname:	
Matrikelnummer:	

Aufgabe	Punkte	Bewertung Teilaufgaben	Gesamt
1	**8**		
a)	2		
b)	2		
c)	4		
2	**10**		
3	**8**		
4	**8**		
5	**8**		
a)	4		
b)	4		
6	**12**		
a)	2		
b)	2		
c)	4		
d)	4		
7	**10**		
8	**16**		
a)	8		
b)	8		
Summe:	**80**		

Bitte benutzen Sie für die Lösungen den freigelassenen Platz nach der jeweiligen Angabe; sollte dieser nicht ausreichen, verwenden Sie bitte die Rückseite, wobei die Zuordnung zur jeweiligen Aufgabe deutlich erkennbar sein muss!

Bitte nicht mit Bleistift schreiben
Die Angabe besteht aus 9 Seiten (ohne Titelblatt)!
Viel Erfolg!!!

Aufgabe 1 (Quicksort) **(8 Punkte)**

Gegeben sei das folgende Zahlen-Feld:

17	5	12	8	7	9	18	13	1	15	16

Ein wichtiger Schritt beim Quicksort-Algorithmus ist, das Feld über ein "Bezugselement" (nennen wir es **x**) in zwei Teilfelder aufzuteilen, so dass alle Elemente im linken Teilfeld kleiner oder gleich **x**, im rechten Teilfeld größer oder gleich **x** sind.

a) 2 Punkte
Welche Eigenschaft sollte das Bezugselement haben, damit der Algorithmus "optimal" arbeitet?

b) 2 Punkte
Nennen Sie zwei Strategien (durch Angabe der Zahl in obigem Feld), wie man "praktisch" das Bezugselement wählt!

c) 4 Punkte
Geben Sie an, wie das Feld nach dem ersten Aufteilungsschritt (Bestimmung der beiden Teilfelder mit obiger Eigenschaft) aussieht! Hierzu sind unten die Zahlen wie sie im Algorithmus umgestellt werden wie auch die Teilungsgrenze einzutragen!

Aufgabe 2 (Rekursion) **(10 Punkte)**

Schreiben Sie eine **rekursive** Oberon-Prozedur, die als Argumente eine Zeichenfolge sowie eine ganze Zahl als Index in diese Zeichenkette erhält und die alle Zeichen von "hinten her" bis einschließlich des Zeichens, das durch das zweite Argument bestimmt wird, ausgibt.

Prozedurkopf:

```
PROCEDURE PrintReverse(a: ARRAY OF CHAR; n : INTEGER);
```

Ein Aufrufbeispiel:

```
PrintReverse("ein neger mit gazelle zagt im regen nie", 2);
```

Dieser liefert:

```
ein neger mi tgaz ellezag tim regen n
```

Lösung:

Aufgabe 3 (Rekursion) **(8 Punkte)**

Gegeben sei das folgende Oberon-Programm:

```
MODULE Aufgabe3;
IMPORT Write;

PROCEDURE Rekursi(a,b : INTEGER) : INTEGER;
  BEGIN
  Write.String("a="); Write.Int(a,1);
  Write.String(", b="); Write.Int(b,1);
  Write.Ln;
  IF (a=0) THEN
         RETURN b;
  ELSE
         RETURN Rekursi(b MOD a, a);
  END;
END Rekursi;

BEGIN
  Write.Int(Rekursi(64,24),0);
  Write.Ln;
END Aufgabe3.
```

(a) (6 Punkte)

Schreiben Sie genau auf, welche Ausgaben das Programm produziert!

(b) (2 Punkte)

Hat das Ergebnis der Prozedur *Rekursi* eine mathematische Bedeutung? Falls ja: welche? (Kurze Begründung!)

Aufgabe 4 (Streams) **(8 Punkte)**

Schreiben Sie im unten vorgegebenen Modul-Rahmen eine Prozedur, die als Argumente durch Leerzeichen oder Tabulatorzeichen getrennte Ziffernfolgen erhält und den Mittelwert dieser als Zahlen interpretierten Ziffernfolgen liefert.

```
MODULE Aufgabe4;
  IMPORT Streams, Strings, Read, Write;
  TYPE
    Field = ARRAY 100 OF CHAR;
  VAR
    f: Field;
  PROCEDURE Average(a: ARRAY OF CHAR): REAL;
  (* HIER BEGINNT IHRE LOESUNG: *)













  END Average;

BEGIN
  f := "1 2    3";
  Write.Real(Average(f), 16);
  Write.Ln;
END Aufgabe4.
```

Aufgabe 5 (Modularisierung) **(8 Punkte)**

a) **(4 Punkte)**

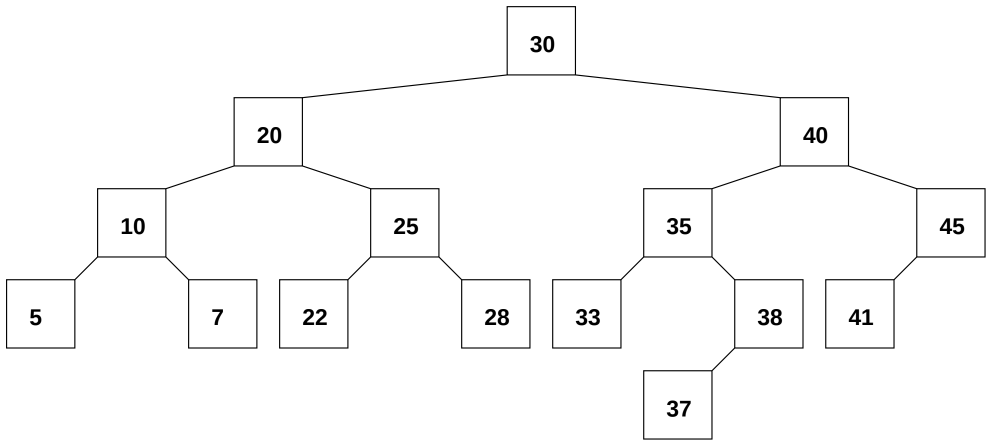
Erläutern Sie, wie das Prinzip **Information Hiding** mit den Konzepten von Oberon umgesetzt werden kann!

b) **(4 Punkte)**

Erläutern Sie, wie **Abstrakte Datentypen** mit den Konzepten von Oberon umgesetzt werden kann!

Aufgabe 6 (Bäume) **(12 Punkte)**

Gegeben sei folgender Binär-Baum:



a) (2 Punkte)
Ist dieser Baum nach Höhe ausgeglichen? (Antwort mit Begründung!)

b) (2 Punkte)
Ist dieser Baum nach Gewicht ausgeglichen? (Antwort mit Begründung!)

(c) (4 Punkte)
Geben Sie an, wie dieser Baum bei **preorder**-Traversierung mit Ausgabe der Knoteninhalte erscheint!

d) (4 Punkte)
Skizzieren Sie in obigem Bild, wie dieser Baum nach Löschen des Knotens “30” mit dem Algorithmus aus der Vorlesung aussieht!

Aufgabe 7 (Argumentverarbeitung) **(10 Punkte)**

Schreiben Sie ein Oberon-Programm, das beim Aufruf genau ein Argument erhält, die dort enthaltenen Großbuchstaben in die entsprechenden Kleinbuchstaben wandelt, die anderen Zeichen unverändert läßt und den so erzeugten String an die Standardausgabe schreibt. Eine korrekte und vollständige Argumentverarbeitung ist verlangt!

Hinweis: Sei a eine Variable vom Typ $CHAR$, die als Wert einen Großbuchstaben hat, so kann daraus wie folgt der entsprechende Kleinbuchstabe erzeugt werden:

```
a := CHR(ORD(a) - ORD("A") + ORD("a"));
```

Lösung:

Aufgabe 8 (Bäume und Hash-Organisation) (16 Punkte)

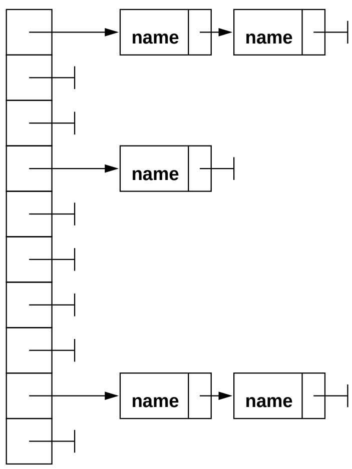
Gegeben sei ein Binärbaum wie folgt:

```
CONST NameLength = 32;
TYPE
   Name = ARRAY NameLength OF CHAR;
   Tree = POINTER TO Node;
   Node = RECORD
            key: INTEGER;
            name: Name;
            left, right: Tree;
          END;
```

Die in den Knoten enthaltenen Einträge – Komponente **name** – sollen zusätzlich in einer Hash-Organisation verwaltet werden. Dazu stehen weiterhin folgende Vereinbarungen zur Verfügung:

```
CONST TableLength = 11;
TYPE
   HashEntry = POINTER TO HashEntryRec;
   HashEntryRec = RECORD
                    name: Name;
                    next: HashEntry;
                  END;
   HashTable = ARRAY TableLength OF HashEntry;
```

Dies ergibt folgende Struktur für die Hash-Tabelle:



weiter auf der nächsten Seite ->

Aufgabe 8 (Fortsetzung)

Weiterhin sind folgende zwei Funktion zur Verwendung gegeben:

```
PROCEDURE InitHash(VAR h: HashTable);
(* h[i] := NIL, 0 <= i < TableLength *)
PROCEDURE Hash(name: Name): INTEGER;
(* returns index in HashTable for given name *)
```

a) (8 Punkte)
Schreiben Sie eine Oberon-Prozedur, die ein neues Element (Typ: *Name*) in die Hash-Tabelle einfügt – diese ist mit der Prozedur *InitHash* bereits initialisiert!
Lösung:

```
PROCEDURE InsertHash(VAR h: HashTable; pos: INTEGER;
                     name: Name);
```

b) (8 Punkte)
Schreiben Sie eine Oberon-Prozedur, die für einen Baum, wie auf der vorigen Seite definiert, bzgl. der Komponente *name* eine Hash-Organisation aufbaut!
Lösung:

```
PROCEDURE MakeHash(root: Tree; VAR h: HashTable);
```