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

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?
Antwort: Es sollte bezgl. der Ordnung der Elemente das "Mittlere" sein – dies zu bestimmen, würde die Effizienz des Quicksort allerdings erheblich verschlechtern!

b) 2 Punkte
Nennen Sie zwei Strategien (durch Angabe der Zahl in obigem Feld), wie man "praktisch" das Bezugselement wählt!
Antwort: Man nimmt das in der Mitte des Feldes stehende oder man nimmt vom ersten, mittleren und letzten Element im Feld das bzgl. der Ordnung dieser drei Elementen "Mittlere".

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!

Bezugselement: **9**

1	5	9	8	7	12	18	13	17	15	16

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:

```
MODULE bsp;

   IMPORT Write, ASCII;

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

   BEGIN
     IF ( n < 0 ) OR (n >= LEN(a)) THEN RETURN END;
     IF a[n] # ASCII.nul
     THEN
        PrintReverse(a, n+1);
        Write.Char(a[n]);
     END;
   END PrintReverse;

BEGIN
   PrintReverse("ein neger mit gazelle zagt im regen nie", 2);
   Write.Ln;

END bsp.
```

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!

```
a=64, b=24
a=24, b=64
a=16, b=24
a=8, b=16
a=0, b=8
8
```

(b) (2 Punkte)
Hat das Ergebnis der Prozedur *Rekursi* eine mathematische Bedeutung? Falls ja: welche? (Kurze Begründung!)
Antwort: Die Prozedur bestimmt den größten gemeinsamen Teiler (ggt) der beiden Parameter;
wenn $a < b$, so ist der $ggt(b\ MOD\ a, a) = ggt(a, b)$; denn sei $r = b\ MOD\ a$, dann ist $b = x * a + r$; ist t der $ggt(a, b)$, so teilt t auch r, also bleibt der ggt erhalten;
wenn $a > b$, dann ist $b\ MOD\ a = b$, beim rekursizen Aufruf werden dann beide Parameter vertauscht;
wenn $a = b$, dann ist $b\ MOD\ a = 0$, d.h. ggt(a,b) = a = b

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;
    VAR
      str: Streams.Stream;
      i,sum,n: INTEGER;
  BEGIN
    n:= 0; sum := 0;
    Strings.Open(str, a);
    Read.IntS(str,i);
    WHILE str.count > 0 DO
      INC(n);
      sum := sum + i;
      Read.IntS(str, i);
    END;
    IF n > 0 THEN
       RETURN sum / n
    ELSE
       RETURN 0.0
    END;
  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!

Antwort:

- Textuelle Trennung in DEFINITION- und Implementierungsmodul
- Im DEFINITIONS MODUL steht nur das, was zur Benutzung der Leistungen des Moduls notwendig ist – das und nur das

b) **(4 Punkte)**

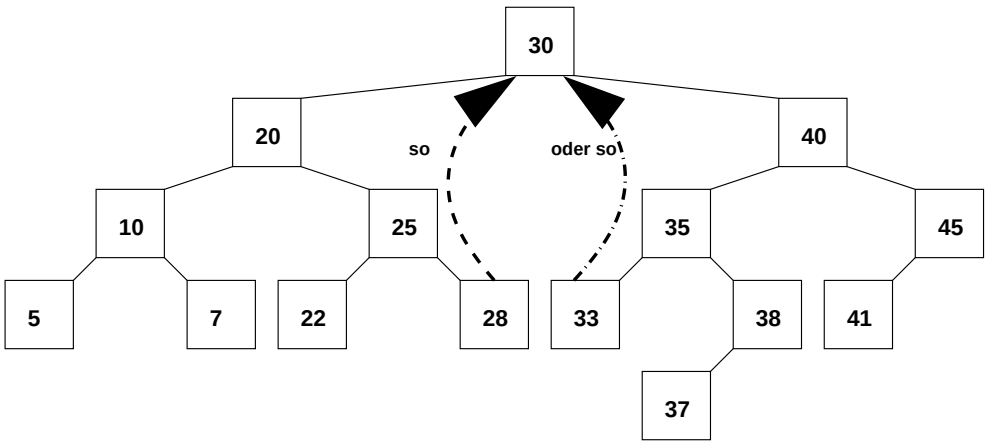
Erläutern Sie, wie **Abstrakte Datentypen** mit den Konzepten von Oberon umgesetzt werden kann!

Antwort:

- Das Modul-Konzept (s.o.)
- Konzept der Typerweiterung
- Basis-Record (bzw. Zeiger): Objects.ObjectRec, Objects.Object

Aufgabe 6 (Bäume) **(12 Punkte)**

Gegeben sei folgender Binär-Baum:



a) (2 Punkte)
Ist dieser Baum nach Höhe ausgeglichen? (Antwort mit Begründung!)
Antwort: ja, für **jeden** Knoten gilt: die Höhe seines linken und die seines rechten Teilbaums unterscheiden sich um maximal 1

b) (2 Punkte)
Ist dieser Baum nach Gewicht ausgeglichen? (Antwort mit Begründung!)
Antwort: nein, die Anzahl der Knoten im linken Teiilbaum des Knotens “40” ist 4, die im rechten 2 $\rightarrow$ Differenz 2

(c) (4 Punkte)
Geben Sie an, wie dieser Baum bei **preorder**-Traversierung mit Ausgabe der Knoteninhalte erscheint!
Lösung:
30 20 10 5 7 25 22 28 40 35 33 38 37 45 41

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

siehe Skizze

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:

```
MODULE ToLower;
  IMPORT ASCII, Args := UnixArguments, Write;

  CONST Length = 256;
  VAR arg: ARRAY Length OF CHAR;
  PROCEDURE Lower(VAR str: ARRAY OF CHAR);
    VAR i: INTEGER;
  BEGIN
    i := 0;
    WHILE (i < LEN(str) )& (str[i] # ASCII.nul) DO
       IF (str[i] >= "A") & (str[i] <= "Z") THEN
          str[i] := CHR(ORD(str[i]) - ORD("A") + ORD("a"));
       END;
       INC(i);
    END
  END Lower;
BEGIN
  Args.Init("filename");
  IF ~ Args.GetArg(arg) THEN
     Args.Usage;
  ELSE
    Args.AllArgs;
    Lower(arg);
    Write.String(arg); Write.Ln;
  END
END ToLower.
```

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

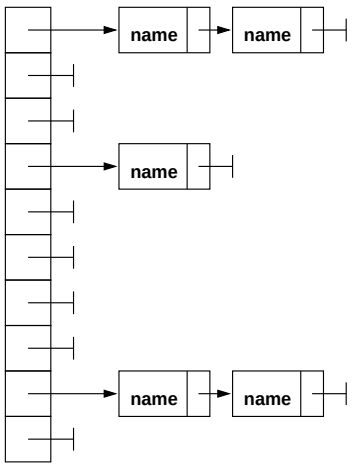
Gegeben sei ein Binärbaum wie folgt:

```
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; na-
me: Name);
    VAR p: HashEntry;
BEGIN
    NEW(p);
    p.name := name;
    p.next := h[pos];
    h[pos] := p;
END InsertHash;
```

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);
   VAR pos: INTEGER;
BEGIN
   IF root # NIL THEN
      MakeHash(root.right, h);
      pos := Hash(root.name, TableLength);
      InsertHash(h, pos, root.name);
      MakeHash(root.left, h);
   END
END MakeHash;
```