Vordiplom für Wirtschaftswissenschaften
Allgemeine Informatik II – SS 2002
30. Juli 2002
Bearbeitungszeit: 120 Minuten
BEISPIELLÖSUNG

Name:	
Vorname:	
Matrikelnummer:	

Aufgabe	Punkte	Bewertung
1	**8**	
a)	2	
b)	6	
2	**8**	
3	**8**	
4	**8**	
5	**8**	
6	**16**	
a)	8	
b)	8	
7	**12**	
a)	2	
b)	2	
c)	4	
d)	4	
8	**12**	
a)	6	
b)	6	
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 12 Seiten (ohne Titelblatt)!
Viel Erfolg!!!

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

Gegeben sei das folgende Zahlen-Feld:

12	5	17	8	7	18	9	13	16	1	15

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.

Teilaufgabe a) **(2 Punkte)**

Nennen Sie zwei Strategien mit Bezug auf obiges Feld, wie man „praktisch" das Bezugselement wählt!

Lösung:
Man nimmt das mittlere Element im Array, also 18

Man nimmt vom ersten, mittleren und letzten Element den Median, also 15

Teilaufgabe b) **(6 Punkte)**

Geben Sie an, wie das Feld bei der Wahl von **12** als Bezugselement 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!

1	5	9	8	7	18	17	13	16	12	15

Aufgabe 2 (Rekursion) **(8 Punkte)**

Gegeben sei folgendes Oberon-Programm mit einer rekursive Prozedur:

```
MODULE rek;
  IMPORT Write;
  VAR x,y: INTEGER;

  PROCEDURE rekursi(b,n: INTEGER):INTEGER;
  BEGIN
    Write.String("b = "); Write.Int(b,4);
    Write.String(" | n = "); Write.Int(n,0); Write.Ln;
    IF n <= 0 THEN RETURN  1 END;
    IF (n MOD 2 = 1) THEN
      RETURN b * rekursi(b, n-1)
     ELSE
       RETURN rekursi(b*b,n DIV 2)
     END;
  END rekursi;

BEGIN
  x := 4;
  x := rekursi(x,5);
  Write.String("Ergebnis: "); Write.Int(x, 0); Write.Ln;
END rek.
```

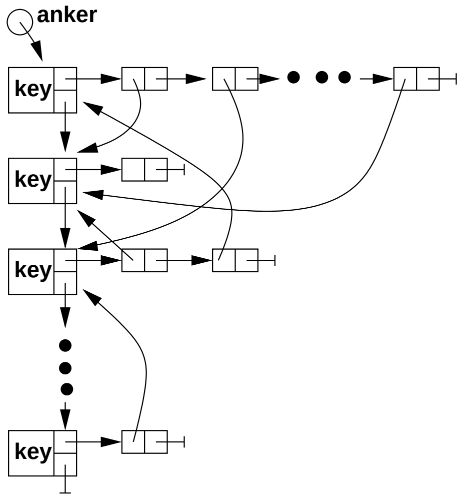
Geben Sie exakt an, was dieses Programm bei der Ausführung an die Standardausgabe ausgibt!

Lösung:

```
b =    4 | n = 5
b =    4 | n = 4
b =   16 | n = 2
b =  256 | n = 1
b =  256 | n = 0
Ergebnis: 1024
```

Aufgabe 3 (Dynamische Datenstrukturen) (8 Punkte)

Durch folgende Skizze wird eine dynamische Datenstruktur beschrieben:



Die Record-Komponente **key** ist dabei eine Folge von maximal 25 Zeichen.

Aufgabe: Geben Sie eine vollständige Typ-Vereinbarung in Oberon, so dass die Variable *anker* in der Skizze vereinbart werden kann!
Lösung:

```
MODULE list;
  TYPE
    List1 = POINTER TO List1Rec;
    List2 = POINTER TO List2Rec;
    List1Rec = RECORD
                  key: ARRAY 26 OF CHAR;
                  next1 : List1;
                  next2 : List2;
               END;
    List2Rec = RECORD
                  next1: List1;
                  next2: List2;
               END;
BEGIN
END list.
```

Aufgabe 4 (Argumentverarbeitung) **(8 Punkte)**

Ein Oberon-Programm soll folgende Aufruf-Struktur erhalten:

progname `[-x] [-n] file`

Die eckigen Klammern bedeuten hier *optional*.

Aufgabe: Schreiben Sie den kompletten Anfang eines Oberon-Programms, das die **vollständige** Argumentverarbeitung durchführt:

- bei irgendeiner Abweichung von obiger Aufrufstruktur soll die übliche *usage*-Meldung ausgegeben und das Programm beendet werden,
- für die weitere Verarbeitung soll die Information hinterlassen werden, welche der Optionen gesetzt war!

Lösung:

```
MODULE bsp;
  IMPORT UA := UnixArguments, Write;
  TYPE
    Argument = ARRAY 256 OF CHAR;
  VAR
    arg: Argument;
    xflag, nflag: BOOLEAN;
    flag: CHAR;
BEGIN
  UA.Init("[-x] [-n] file");
  xflag := FALSE; nflag := FALSE;
  WHILE UA.GetFlag(flag) DO
     CASE flag OF
     | "x" : xflag := TRUE;
     | "n" : nflag := TRUE;
     ELSE    UA.Usage;
     END;
  END;
  IF ~ UA.GetArg(arg) THEN UA.Usage END;
  UA.AllArgs;
END bsp.
```

Aufgabe 5 (File-I/O, Streams) (8 Punkte)

In einer Datei *daten* werden zeilenweise Daten in folgender Struktur abgelegt:
`anzahl einzelpreis kommentar`
Beispiel für den Inhalt der Datei *daten*:

```
2  3.5 Einzelne Artikel, die vorher 3.9 kosteten

5   6.0 Alte CD's
4   2.5 Batterien
```

Sie können davon ausgehen, dass die Datei genau diese Struktur besitzt, Leerzeilen sind allerdings zulässig.
Aufgabe: Schreiben Sie ein vollständiges Oberon-Programm, dass den Namen der Datei der Eingabedaten als einziges Argument bekommt und folgenden Wert berechnet und an die Standardausgabe ausgibt:

$$\sum anzahl * einzelpreis$$

```
MODULE prog;
  IMPORT Streams, UF := UnixFiles, UA := UnixArguments,
         Strings, Read, Write;
  CONST LineLength = 80;
  TYPE Line = ARRAY LineLength OF CHAR;
  VAR
    anzahl: INTEGER; epreis, sum: REAL;
    line: Line;
    fstr, lstr: Streams.Stream;
    file: ARRAY 256 OF CHAR;
BEGIN
    IF UA.GetArg(file) THEN END;
    IF ~UF.Open(fstr, file, UF.read, 1, NIL) THEN
       HALT(1);
    END;
    sum := 0.0; anzahl := 0; epreis := 0.0;
    Read.LineS(fstr, line);
    WHILE ~ fstr.eof DO
       Strings.Open(lstr, line);
       Read.IntS(lstr, anzahl); Read.RealS(lstr, epreis);
       sum := sum + anzahl * epreis;
       anzahl := 0; epreis := 0.0;
       Read.LineS(fstr,line);
    END;
    Write.Real(sum,16); Write.Ln;
END prog.
```

Aufgabe 6 (Listen) **(16 Punkte)**

Gegeben sei folgende Typ-Vereinbarung:

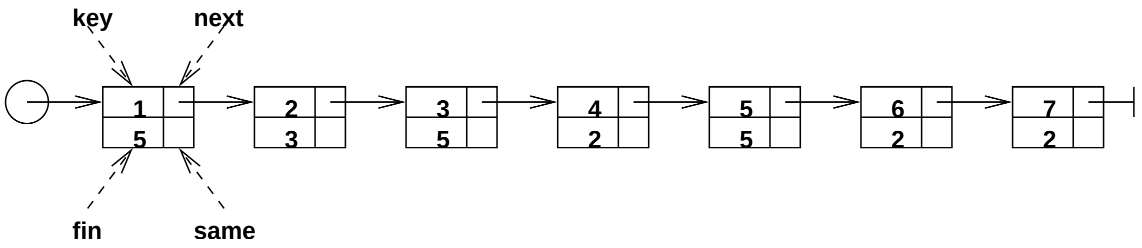
```
TYPE List = POINTER TO ListRec;
     ListRec = RECORD
                 key, financialStanding: INTEGER;
                 same: List;
                 next: List;
               END;
```

Über die Komponente `next` wird eine Liste von Datensätzen aufgebaut. Die Komponente `key` sei ein Schlüssel (z. B. zur Identifizierung eines Lieferanten) und die Komponente `financialStanding` sei ein weiteres Merkmal (hier z. B. die Bonität des Lieferanten).

Teilaufgabe 6a) **(8 Punkte)**

Schreiben Sie eine Prozedur, die als ersten Parameter einen Zeiger auf ein Element dieser Liste und als zweiten Parameter einen Wert für die Komponente `financialStanding` erhält. Die Prozedur soll einen Zeiger auf das erste gefundene Listenelement liefern, dessen Komponente `financialStanding` mit dem zweiten Parameter übereinstimmt. Die Suche soll an dem als ersten Parameter übergebenem Listenelement beginnen. Falls kein solches Listenelement existiert, soll `NIL` geliefert werden.

Beispiel:



- Übergibt man an die Prozedur einen Zeiger auf den Knoten mit `key = 2` und `bon = 5`, so liefert sie einen Zeiger auf den Knoten mit `key = 3`;
- übergibt man an die Prozedur einen Zeiger auf den Knoten mit `key = 2` und `bon = 3`, so liefert sie einen Zeiger auf den Knoten mit `key = 2`

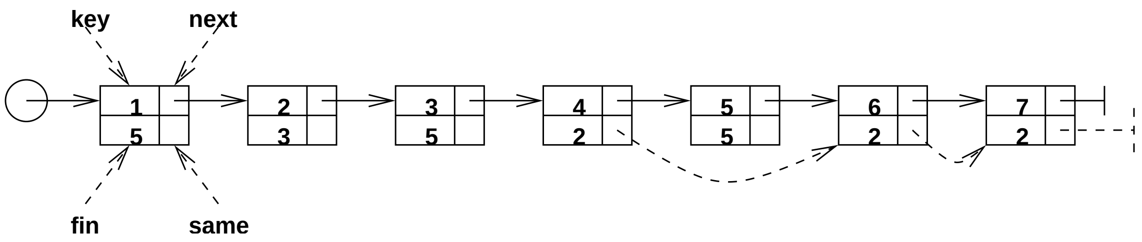
Lösung von Teilaufgabe 6a):

```
PROCEDURE GetNext(p: List; b: INTEGER):List;
BEGIN
  WHILE (p # NIL) DO
     IF p.financialStanding = b THEN
          RETURN p
     ELSE
        p := p.next;
     END;
   END;
   RETURN NIL;
 END GetNext;
```

Teilaufgabe 6b) **(8 Punkte)**

Unter Nutzung der Prozedure `GetNext` der Teilaufgabe a) sollen Sie nun eine Prozedur **Join** schreiben, die als ersten Parameter den Anker einer Liste (vom Typ `List`, siehe oben) und als zweiten Parameter einen Wert für die Komponente `financialStanding` erhält und alle Listenelemente mit diesem Wert über die Komponente `same` analog zur Verkettung über die Komponente `next` verkettet.

Beispiel: Verkettung aller Elemente mit `bon = 2`

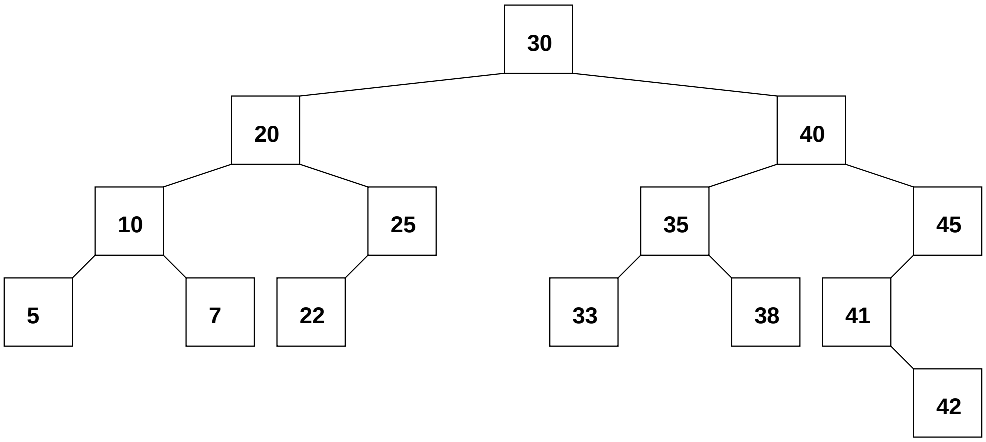


Lösung:

```
PROCEDURE Join(p: List; b: INTEGER);
  VAR q: List;
BEGIN
   IF p = NIL THEN RETURN END;
   IF p.financialStanding = b THEN
      p.same := GetNext(p.next, b);
      p := p.same;
   ELSE
      p := GetNext(p,b);
   END;
   Join(p,b);
END Join;
```

Aufgabe 7 (Bäume) (12 Punkte)

Gegeben sei folgender Binär-Baum:



Teilaufgabe 7a) (2 Punkte)
Ist dieser Baum nach Höhe ausgeglichen? (Antwort mit Begründung!)

Nein: Knoten 45 → Höhe des linken Teilbaums ist 2, die des rechten 0!

Teilaufgabe 7b) (2 Punkte)
Ist dieser Baum nach Gewicht ausgeglichen? (Antwort mit Begründung!)

Nein: Knoten 45 → Gewicht des linken Teilbaums ist 2, das des rechten 0!

Teilaufgabe 7c) (4 Punkte

Geben Sie an, wie die Knoten dieses Baumes bei **postorder**-Traversierung ausgegeben werden!

5 7 10 22 25 20 33 38 35 42 41 45 40 30

Teilaufgabe 7d) (4 Punkte)

Beschreiben Sie **zwei verschiedene** Strategien mit Bezug auf die Knoteninhalte des obigen Baumes, wie man zum Löschen des Knotens mit Inhalt “30” vorgehen kann!

Man kopiert vom linken Teilbaum den größten Knoten (also 25) an die Stelle von 30 und löscht die alte Position von 25, indem der Rechts-Zeiger von 20 auf 22 umgesetzt wird

Man kopiert vom rechten Teilbaum den kleinsten (also 33) an die Stelle von 30 und löscht die alte Position von 33, indem der Links-Zeiger von 35 auf NIL gesetzt wird

Aufgabe 8 (Bäume) **(12 Punkte)**

Gegeben sei ein Binärbaum wie folgt:

```
TYPE
   Tree = POINTER TO Node;
   Node = RECORD
            key,bal: INTEGER;
            left, right: Tree;
          END;
```

Die Komponente `key` sei wieder ein identifizierendes Merkmal (hier nicht weiter von Bedeutung); die Komponente `bal` gibt für jeden Knoten den „Balanzierungsgrad“ in folgendem Sinn an:
Sei `p` ein Zeiger auf einen Knoten des Baumes, so ist `p.bal` die Anzahl Knoten im linken Teilbaum minus Anzahl Knoten im rechten Teilbaum.

Teilaufgabe a) **(6 Punkte)**

Schreiben Sie eine Oberon-Prozedur, die für einen Teilbaum die Zahl seiner Knoten liefert! Die Kompoente `bal` darf hierzu **nicht** benutzt werden!

Lösung:

```
PROCEDURE Count(p: Tree):INTEGER;
BEGIN
  IF p = NIL THEN RETURN 0
  ELSE
    RETURN 1 + Count(p.left) + Count(p.right)
  END;
 END Count;
```

Teilaufgabe b) **(6 Punkte)**

Unter Nutzung der Prozedur `Count` aus der Teilaufgabe a) ist nun eine Prozedur zu entwickeln, die in alle Knoten eines Baums den Balanzierunggrad in die Komponente `bal` einträgt.

Lösung:

```
PROCEDURE Bal(p: Tree);
BEGIN
   IF p # NIL THEN
       p.bal := Count(p.left) - Count(p.right);
       Bal(p.left);
       Bal(p.right);
   END;
END Bal;
```