Vordiplom für Wirtschaftswissenschaften
Allgemeine Informatik II – SS 2003
5. August 2003
Beispiellösung
Bearbeitungszeit: 120 Minuten

Name:	
Vorname:	
Matrikelnummer:	

Aufgabe	Punkte	Bewertung
1	**7**	
a)	1	
b)	2	
c)	4	
2	**7**	
3	**8**	
4	**8**	
5	**10**	
6	**10**	
a)	4	
b)	6	
7	**10**	
8	**12**	
a)	2	
b)	2	
c)	4	
d)	4	
9	**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 und nicht mit Rotstift schreiben
Die Angabe besteht aus 10 Seiten (ohne Titelblatt)!
Viel Erfolg!!!

Aufgabe 1 (Quicksort) **(7 Punkte)**

Gegeben sei das folgende Zahlen-Feld:

17	5	12	8	7	1	18	13	9	15	3	16

a) 1 Punkt
Ein wichtiger Schritt beim Quicksort-Algorithmus ist die Wahl des Elements (Pivot-Element, Bezugselement), mit dem die Partionierung durchgeführt wird. Ist die Wahl des Elementes **1** in obigem Array als Pivot-Element günstig? Antwort bitte mit Begründung versehen!

Lösung: 1 ist das kleinste Element und damit extrem ungünstig, weil der Teilungsschritt ein einelementiges und ein (n-1)-elementiges liefern würde

b) 2 Punkte
Als Pivotelement könnte man auch den Median der Elemente im Array verwenden. Warum ist dies für die Effizienz des Quicksort eher ungünstig?

Lösung:

Zur Bestimmung des Medians müsste das Array sortiert sein - danach ist der Quicksort aber überflüssig!

c) 4 Punkte
Als Pivotelement wird für die folgende Aufgabe das Element 13 festgelegt. Geben Sie an, wie das Feld nach dem ersten Aufteilungsschritt (Bestimmung der beiden Teilfelder) aussieht! Hierzu sind unten die Zahlen, wie sie im Algorithmus aus der Vorlesung umgestellt werden, und auch die Teilungsgrenze einzutragen!

3	5	12	8	7	1	9	13	18	15	17	16

Aufgabe 2 (Rekursion) **(7 Punkte)**

Gegeben sei das folgende Oberon-Programm:

```
MODULE Rek;
  IMPORT Write;

  PROCEDURE Rekursi(a,b,n: INTEGER);
  BEGIN
    Write.String("a = "); Write.Int(a,0);
    Write.String(", b = "); Write.Int(b,0);
    Write.String(", n = "); Write.Int(n,0);
    Write.Ln;
    IF (a <= 1) OR (b <= 1) THEN RETURN END;
    IF (a = b) THEN RETURN
    ELSIF (a < b ) THEN Rekursi(b, a, n)
    ELSE Rekursi(a - b, b, n+1);
    END
  END Rekursi;

BEGIN
  Rekursi(9,15,0);
END Rek.
```

Schreiben Sie genau auf, welche Ausgaben das Programm produziert!

Lösung:

```
a = 9, b = 15, n = 0
a = 15, b = 9, n = 0
a = 6, b = 9, n = 1
a = 9, b = 6, n = 1
a = 3, b = 6, n = 2
a = 6, b = 3, n = 2
a = 3, b = 3, n = 3
```

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

Schreiben Sie eine **rekursive** Oberon-Prozedur, die als Parameter eine nicht-negative ganze Zahl erhält und als Ergebnis deren Quersumme (Summe der dezimalen Ziffern dieser Zahl) zurückliefert!

Lösung:

```
PROCEDURE QuerSum(x:INTEGER):INTEGER;
BEGIN
   IF x < 0 THEN RETURN -1 (*ERROR*) END;

   IF x < 10 THEN RETURN  x
   ELSE
      RETURN (x MOD 10) + QuerSum(x DIV 10)
   END;
END QuerSum;
```

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

Schreiben Sie eine Oberon-Prozedur, die als Parameter ein Array von ganzen Zahlen sowie die Anzahl dieser Zahlen erhält und über einen weiteren Parameter diese Zahlen in Dezimalform, durch jeweils einen Doppelpunkt getrennt, in einem Array von Zeichen (**ARRAY OF CHAR**) liefert! Sie dürfen davon ausgehen, dass das Array der Zeichen hinreichend gross ist, um die Zahlen wie auch den Trenner aufzunehmen.

Lösung:

```
PROCEDURE DigitSeq(zahlen: ARRAY OF INTEGER; n: INTEGER;
                   VAR seq: ARRAY OF CHAR);
  VAR
    str: Streams.Stream;
    i: INTEGER;
BEGIN
  Strings.Open(str, seq);
  i:= 0;
  Write.IntS(str, zahlen[i], 0);
  WHILE (str.count > 0) & (i < n-1) DO
   Write.CharS(str,":");
   INC(i);
   Write.IntS(str, zahlen[i], 0);
  END;
END DigitSeq;
```

Aufgabe 5 (Argumentverarbeitung) (10 Punkte)

Schreiben Sie ein Oberon-Programm, das folgende Aufrufstruktur entsprechend den in der Vorlesung vorgestellten Unix-Konventionen besitzt:

```
progname [-r] [-zahl] file1 file2
```

`zahl` ist dabei eine nichtnegative ganze Zahl, die als Voreinstellung (*default*) 1 sei. `file1` und `file2` sind beliebige Dateinamen, die auch mit einem Minuszeichen (!) beginnen können.
Schreiben Sie die komplette Argumentverarbeitung; zusätzlich dazu soll das Programm ausgeben, ob die Option `-r` gesetzt war, welche Zahl verwendet wurde und welche Dateinamen übergeben wurden. Falsche Argumente oder zuviele / zuwenige Argumente sollen zu der üblichen *Usage*-Meldung führen. Der Anfang des Programms ist vorgegeben:

Lösung:

```
MODULE Argv;
   IMPORT UA := UnixArguments, Streams, Write, Read;
   CONST
      FileNameLength = 256;
   TYPE
      FileName = ARRAY FileNameLength OF CHAR;
   VAR
      arg: Streams.Stream;
      file1, file2: FileName;
      flag: CHAR; number: INTEGER; endflags: BOOLEAN;
BEGIN
   number := 1;
   endflags := FALSE;
   UA.Init("[-r] [-zahl] file1 file2");
   WHILE ~ endflags & UA.GetFlag(flag) DO
      CASE flag OF
      | "r":         Write.String("r-flag gesetzt"); Write.Ln;
      | "0" .. "9":  UA.UngetOpt;
                     UA.Fetch(arg);
                     Read.IntS(arg,number);
                     Write.Int(number,0); Write.Ln;

      | "-":         endflags := TRUE;
      ELSE
         UA.Usage;
```

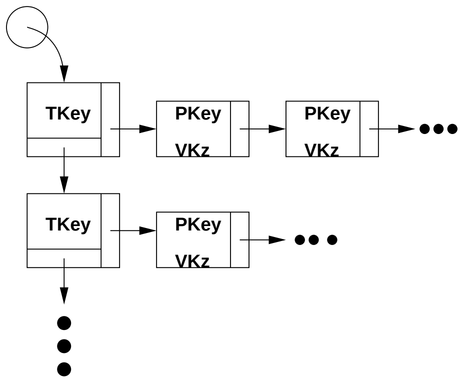
```
      END
    END;
    IF UA.GetArg(file1) & UA.GetArg(file2) THEN
        Write.String("1. Datei: "); Write.String(file1); Write.Ln;
        Write.String("2. Datei: "); Write.String(file2); Write.Ln;
        UA.AllArgs();
    ELSE
        UA.Usage;
    END;
END Argv.
```

Aufgabe 6 (Dynamische Datentypen) (10 Punkte)

Ein Konzern hat eine Anzahl von Tochtergesellschaften, die wie heute so üblich variiert. Diese Tochtergesellschaften werden über einen dreistelligen alphanumerischen Schlüssel (bezeichnet als „TKey“) identifiziert.
Jede dieser Tochtergesellschaften produziert eine variable Anzahl verschiedener Produkte. Die einzelnen Gesellschaften identifizieren ihre Produkte über einen zweistelligen alphanumerischen Schlüssel (bezeichnet als „PKey“); die Schlüsselmengen der einzelnen Gesellschaften müssen nicht disjunkt sein. Für jedes Produkt ist für den Konzern eine ganzzahlige, positive Verkaufskennzahl (bezeichnet als „VKz“) von Bedeutung. Für ein Auswertungsprogramm muss dieser Sachverhalt in **einer** Datenstruktur definiert werden.

a) 4 Punkte
Geben Sie in grafischer Form („Kästchen, Pfeile, Beschriftungen, ...“) eine Beschreibung der oben beschriebenen Datenstruktur bei drei Tochtergesellschaften, die jeweils 3 Produkte produzieren!

Lösung:



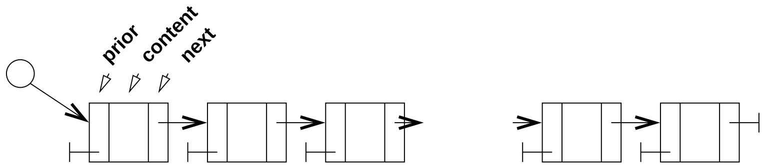
b) 6 Punkte
Geben Sie alle notwendigen Oberon-Vereinbarungen an, so dass eine Variable zur Aufnahme des obigen Sachverhalts deklariert werden kann!

```
MODULE ds;
  CONST TKEY = 3; PKEY = 2;
  TYPE
    TData = POINTER TO TDataRec;
    PData = POINTER TO PDataRec;
    TDataRec = RECORD
                   tkey: ARRAY TKEY+1 OF CHAR;
                   prod: PData;
                   next: TData
               END;
```

```
    PDataRec = RECORD
                  pkey: ARRAY PKEY+1 OF CHAR;
                  vkz: INTEGER;
                  next: PData
               END;
END ds.
```

Aufgabe 7 (Dyn. Datenstrukturen und Rekursion) **(10 Punkte)**

Gegeben sei eine Liste folgender Struktur:



Die zugehörige Typ-Vereinbarung ist wie folgt gegeben:

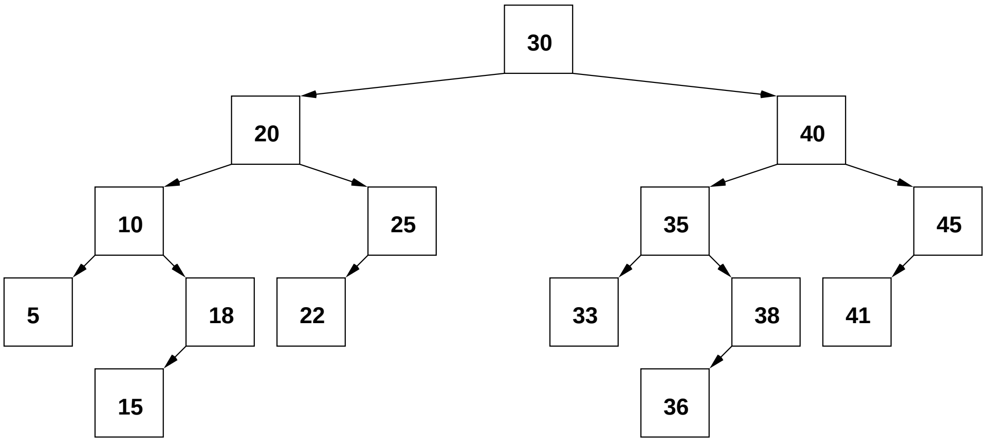
```
TYPE List = POINTER TO ListRec;
     ListRec = RECORD
                 content: INTEGER;
                 prior, next: List;
               END;
```

Die Verkettung von „links nach rechts" sei also gegeben.
Schreiben Sie eine rekursive Prozedur `BackChain`, die bei einer durch einen Parameter gegebenen Liste die Verkettung von „rechts nach links" aufbaut, so dass eine doppelt verkettete Liste entsteht!
Lösung:

```
PROCEDURE BackChain(l: List);
BEGIN
  IF l = NIL THEN RETURN END;
  IF l.next # NIL THEN
       l.next.prior := l;
       BackChain(l.next);
  END;
END BackChain;
```

Aufgabe 8 (Bäume) **(12 Punkte)**

Gegeben sei folgender Binär-Baum:



Teilaufgabe 8a) **(2 Punkte)**
Ist dieser Baum nach Höhe ausgeglichen? (Antwort mit Begründung!)

Lösung:
Ja, denn für jeden Knoten gilt: Höhendifferenz seines linken und rechten Teilbaums ist betragsmäßig kleiner oder gleich 1!

Teilaufgabe 8b) **(2 Punkte)**
Ist dieser Baum nach Gewicht ausgeglichen? (Antwort mit Begründung!)

Lösung:
NEIN, denn z.B. der linke Teilbaum des Knotens „20" hat 4 Knoten, sein rechter hat 2 Knoten, Differenz ist größer als 1

Teilaufgabe 8c) **(4 Punkte**

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

Lösung:

30 20 10 5 18 15 25 22 40 35 33 38 36 45 41

Teilaufgabe 8d) **(4 Punkte)**

Beschreiben Sie **zwei verschiedene** Strategien mit Bezug auf die Knoteninhalte des obigen Baumes, wie man zum Löschen des Knotens mit Inhalt "35" mit Aufrechterhaltung der Sortierung vorgehen kann!

Lösung:

Man kopiert vom linken Teilbaum den größten Knoten (also 33) an die Stelle von 35 und löscht die alte Position von 33, indem der Links-Zeiger von früher 35 (jetzt 33) auf NIL gesetzt wird

Man kopiert vom rechten Teilbaum den kleinsten (also 36) an die Stelle von 35 und löscht die alte Position von 36, indem der Links-Zeiger von 38 auf NIL gesetzt wird

Aufgabe 9 (Bäume und Rekursion) (8 Punkte)

Durch folgende Typ-Vereinbarung sei ein (nach **inorder**) sortierter Binär-Baum definiert:

```
TYPE Tree = POINTER TO Node;
     Node = RECORD
              content: INTEGER; (* >= 0 *)
              left, right: Tree
            END;
```

Schreiben Sie eine rekursive Prozedur, die als Parameter einen Zeiger auf die Wurzel eines solchen Baumes erhält und in aufsteigender Reihenfolge alle geradzahligen Knoteninhalte ausgibt!

Lösung:

```
PROCEDURE InorderEven(root: Tree);
BEGIN
  IF root = NIL THEN RETURN END;
  IF root.left # NIL THEN
     InorderEven(root.left)
  END;
  IF root.content MOD 2 = 0 THEN
     Write.Int(root.content,6)
  END;
  IF root.right # NIL THEN
     InorderEven(root.right)
  END;
END InorderEven;
```