Vordiplom für Wirtschaftswissenschaften
Allgemeine Informatik II – WS 2003/2004
10. Oktober 2003

Beispiellösung

Bearbeitungszeit: 120 Minuten

Name:	
Vorname:	
Matrikelnummer:	

Aufgabe	Punkte	Bewertung
1	**7**	
a)	1	
b)	2	
c)	4	
2	**11**	
3	**14**	
4	**12**	
a)	6	
b)	6	
5	**8**	
6	**8**	
7	**12**	
8	**8**	
a)	2	
b)	2	
c)	4	
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:

150	16	12	8	7	180	1	13	9	5	3	170

a) 1 Punkt
Ein wichtiger Schritt beim Quicksort-Algorithmus ist die Wahl des Elements (Pivot-Element, Bezugselement), mit dem die Partitionierung durchgeführt wird. Ist die Wahl des Elementes **180** in obigem Array als Pivot-Element für den ersten Partitionierungsschritt dafür günstig? Antwort bitte mit Begründung versehen!
Lösung:

Nein, 180 ist das größte Element und würde zu einer sehr ungleichen Partitionierung (1 zu n-1) führen.

b) 2 Punkte
Als Pivotelement könnte man auch den auf eine ganze Zahl gerundeten Mittelwert (Durchschnittswert) der Elemente im Array verwenden. Warum ist dies für die Effizienz des Quicksort in obigem Beispiel eher ungünstig?
Lösung:

Auch nein, da dieser Mittelwert von den drei großen Zahlen dominiert wird und wiederum zu einer sehr ungleichen Partionierung (3 zu n -3) führen würde!

c) 4 Punkte
Als Pivotelement wird für die folgende Aufgabe das Element 12 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, wie auch die Teilungsgrenze einzutragen!

3	5	9	8	7	1	180	13	12	16	150	170

Aufgabe 2 (Rekursion) **(11 Punkte)**

Gegeben ist folgendes Oberon-Programm mit der rekursiven Prozedur *Rekursi*:

```
MODULE Rekursion;
  IMPORT Write;

  VAR k1,k2: INTEGER;

  PROCEDURE Rekursi(i,j: INTEGER; VAR k1,k2: INTEGER);
  BEGIN
     IF (i<=0) OR (j <=0) THEN RETURN END;
     Write.Int(i*k1,0);  Write.String(" | ");
     IF (k1*i < k2*j) THEN INC(k1); Rekursi(i,j,k1,k2);
     ELSIF (k1*i > k2*j) THEN INC(k2); Rekursi(i,j,k1,k2)
     ELSE RETURN END;
     Write.String(" | "); Write.Int(j*k2,0);
  END Rekursi;
BEGIN
  k1:=1; k2:=1;
  Rekursi(4,3,k1,k2);
  Write.Ln;
END Rekursion.
```

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

Lösung:

```
4 | 4 | 8 | 8 | 12 | 12 |   | 12 | 12 | 12 | 12 | 12
```

Aufgabe 3 (File-I/O, Argumentverarbeitung) (14 Punkte)

Schreiben Sie ein Oberon-Programm *Head*, das an die Standardausgabe die ersten Zeilen von den Dateien ausgibt, deren Namen als Argumente übergeben werden. Die Anzahl der auszugebenden Zeilen ist per Voreinstellung auf 5 zu setzen, kann aber als Option beim Aufruf geändert werden. Die Aufruf-Syntax ist also:

```
Head [-zahl] file ...
```

Bei nicht existierenden oder nicht lesbaren Dateien soll eine Fehlermeldung an die Diagnoseausgabe ausgegeben werden. Eventuell notwendige Konstantenvereinbarungen können frei (aber sinnvoll) gewählt werden!
Lösung: (Fortsetzung nächstes freies Blatt)

```
MODULE Head;
   IMPORT UA := UnixArguments, UF:=UnixFiles, Streams, Write, Read;
   CONST
      FileNameLength = 256;
      LineLength = 1024;
   TYPE
      FileName = ARRAY FileNameLength OF CHAR;
      Line     = ARRAY LineLength OF CHAR;
   VAR
      arg: Streams.Stream;
      file: FileName;
      line: Line;
      flag: CHAR;
      number,i: INTEGER; endflags, gotfile: BOOLEAN;
BEGIN
   number := 5;
   endflags := FALSE; gotfile := FALSE;
   UA.Init("[-zahl] file ...");
   WHILE ~ endflags & UA.GetFlag(flag) DO
      CASE flag OF
      | "0" .. "9":  UA.UngetOpt;
                     UA.Fetch(arg);
                     Read.IntS(arg,number);

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

Fortsetzung Aufgabe 3:

```
   WHILE UA.GetArg(file) DO
       gotfile := TRUE;
       IF UF.Open(arg,file,UF.read,1,NIL) THEN
         Write.String(file); Write.String(" : "); Write.Ln;
         i:= 0;
         Read.LineS(arg,line);
         WHILE (i < number) & ~arg.eof DO
           Write.Line(line);
           INC(i);
           Read.LineS(arg,line);
         END;
     ELSE
       Write.StringS(Streams.stderr,file);
       Write.StringS(Streams.stderr," does not exist!");
       Write.LnS(Streams.stderr);
     END;
   END;
   IF ~gotfile THEN UA.Usage; END;
END Head.
```

Aufgabe 4 (Rekursion) **(12 Punkte)**

Durch folgende Typ-Vereinbarung wird eine dynamische Liste definiert:

```
TYPE List = POINTER TO ListRec;
     ListRec = RECORD
                 content: INTEGER; (*Eintrag*)
                 next: List;
               END;
```

a) **6 Punkte**
Schreiben Sie eine rekursive Prozedur, die alle Einträge, die ein Vielfaches von 3 sind, **vom Ende der Liste her** an die Standardausgabe ausgibt!
Lösung:

```
PROCEDURE PrintFromE(l: List);
BEGIN
  IF l = NIL THEN RETURN END;
  PrintFromE(l.next);
  IF (l.content MOD 3 = 0) THEN Write.Int(l.content,4); END;
END PrintFromE;
```

b) **6 Punkte**
Schreiben Sie eine rekursive Prozedur, die als Ergebnis die Anzahl der Listenelemente liefert, deren Eintrag mit einer an die Prozedur übergebenen Zahl übereinstimmt!
Lösung:

```
PROCEDURE Count(l: List; el: INTEGER):INTEGER;
BEGIN
  IF l = NIL THEN RETURN 0
  ELSE
     IF l.content = el
     THEN RETURN 1 + Count(l.next, el)
     ELSE RETURN 0 + Count(l.next,el)
     END
  END;
END Count;
```

Aufgabe 5 (Dynamische Datenstrukturen) (8 Punkte)

Für eine große Handelskette mit Niederlassungen in sehr vielen Orten sollen die Umsatzdaten aller Niederlassungen verwaltet werden. Jede Niederlassung wird durch Angabe des Ortes (Ortsnamen sind maximal 20 Zeichen lang) und durch Angabe der Straße (Straßennamen sind ebenfalls maximal 20 Zeichen lang) eindeutig identifiziert; zu diesen beiden Angaben wird noch der Umsatz (Typ: REAL) geführt. In den meisten Orten gibt es nur eine, in einigen aber zwei und mehr Niederlassungen. Da der Zugriff auf diese Daten immer über die Festlegung des Ortes erfolgt, soll als grundlegende Datenstruktur ein sortierter Binärbaum verwendet werden; darin kommt jeder Ort genau einmal vor.

Geben Sie nach diesen Vorgaben die notwendigen Vereinbarungen an!

Lösung:

```
CONST
  Laenge = 21;
TYPE
  Name = ARRAY Laenge OF CHAR;
  Baum = POINTER TO BaumRec;
  Niederlassung = POINTER TO NiederlassungRec;
  BaumRec = RECORD
              ort: Name;
              nl: Niederlassung;
              l,r: Baum;
            END;
  NiederlassungRec =
            RECORD
              strasse: Name;
              umsatz: REAL;
              next: Niederlassung
            END;
```

Aufgabe 6 (Streams) (8 Punkte)

Gegeben sei folgende Typ-Vereinbarung:

```
TYPE Buffer = ARRAY 1024 OF CHAR;
```

Schreiben Sie eine Prozedur, die als Parameter ein Objekt von diesem Typ, gefüllt mit ganzen Zahlen (getrennt durch Leeraum), erhält, von diesen Zahlen die Summe bildet und diese am Ende anfügt (durch Leeraum von der letzten Zahl getrennt). Sie können davon ausgehen, dass die Summe noch Platz in diesem Objekt hat!

Lösung:

```
PROCEDURE CheckSum(VAR b: ARRAY OF CHAR);
  VAR s,i: INTEGER; str: Streams.Stream;
BEGIN
  s:=0;
  Strings.Open(str, b);
  Read.IntS(str,i);
  WHILE ~str.eof DO
     s := s + i;
     Read.IntS(str,i);
  END;
  Write.CharS(str, " ");
  Write.IntS(str,s,0);
END CheckSum;
```

Aufgabe 7 (Listen) **(12 Punkte)**

Durch folgende Typ-Vereinbarung ist eine Liste definiert:

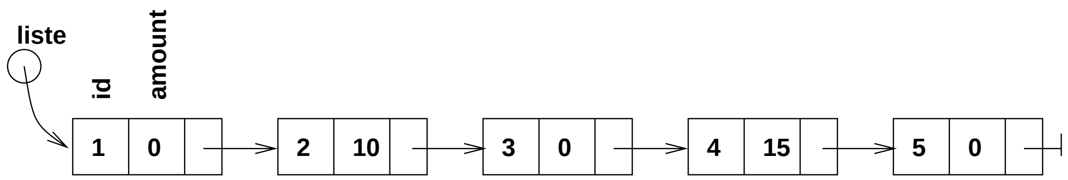
```
TYPE List = POINTER TO ListRec;
     ListRec = RECORD
                  id, amount: INTEGER;
                  next: List;
                END;
```

Schreiben Sie eine Prozedur

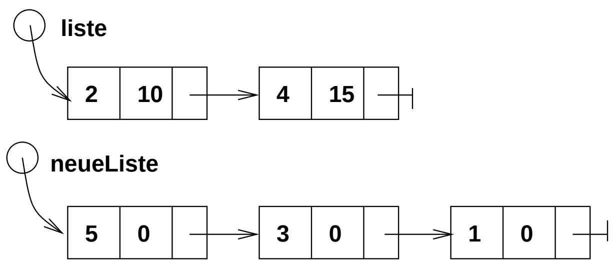
```
PROCEDURE Chain(VAR l: List; amount:INTEGER):List;
```

die alle Elemente mit einem als Parameter übergebenen Wert der Komponente `amount` aus der als Parameter übergebenen Liste `l` entfernt und diese in einer neuen Liste zusammenfügt. Letztere soll von der Prozedur zurückgegeben werden. Die Reihenfolge der Elemente muss nicht erhalten bleiben!

Beispiel:



neueListe := Chain(liste, 0) $\hookrightarrow$



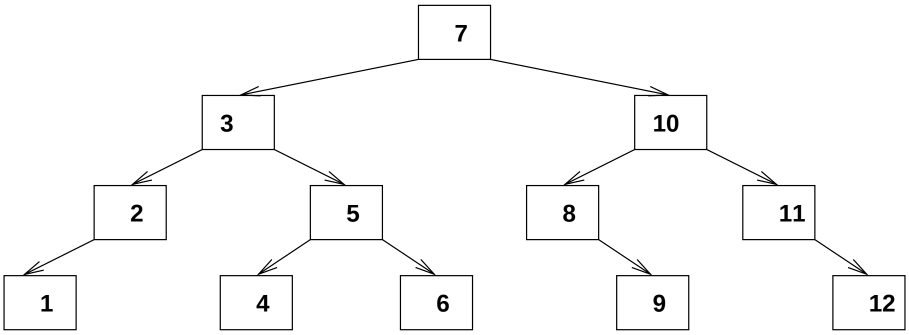
Lösung nächstes Blatt

Lösung Aufgabe 7:

```
 PROCEDURE Chain(VAR l: List; amount: INTEGER):List;
  VAR p,q, newlist: List;
 BEGIN
   IF l = NIL THEN RETURN NIL END;
   newlist := NIL;
   WHILE l.amount = amount DO
      p := l;
      l := l.next;
      p.next := newlist;
      newlist := p;
   END;
   q := l;
   WHILE q.next # NIL DO
     IF q.next.amount = amount THEN
         p := q.next;
         q.next := p.next;
         p.next := newlist;
         newlist := p;
     ELSE
         q := q.next;
     END;
  END;
  RETURN newlist;
END Chain;
```

Aufgabe 8 (Bäume) **(8 Punkte)**

Gegeben ist folgender Binärbaum:



a) 2 Punkte
Ist dieser Baum vollständig ausgeglichen? (Antwort mit kurzer Begründung)

Lösung: Ja, denn für jeden Knoten gilt: Die Anzahl der Knoten in seinem linken Teilbaum und die Anzahl derer in seinem rechten Teilbaum differieren um maximal 1!

b) 2 Punkte
Ist dieser Baum nach Höhe ausgeglichen? (Antwort mit kurzer Begründung)

Lösung: Ja, denn für jeden Knoten gilt: Die Höhe seines linken Teilbaums und die seines rechten Teilbaums differieren um maximal 1!

c) 4 Punkte

Tragen Sie in obigen Baum die Zahlen von 1 bis 12 so ein, dass er bzgl. dieser Zahlen nach inorder sortiert ist!

Lösung: siehe oben