Vordiplom Wirtschaftswissenschaften

Lösung zur Allgemeine Informatik II

SS 2000

4. August 2000

Aufgabe		Punkte	Bewertung
1		8	
2		8	
3	a)	8	
	b)	8	
4		8	
5		6	
6	a)	8	
	b)	8	
7	a)	3	
	b)	3	
	c)	6	
	d)	6	
Summe		**80**	

☞ Bearbeitungszeit: 120 Min.

Aufgabe 1 (8 Punkte)

Argumentverarbeitung:

Schreiben Sie ein Oberon-Programm `Count`, das sowohl die Zahl der beim Aufruf auf der Kommandozeile gesetzten *flags* als auch die der “echten” Argumente ausgibt! Von der Einhaltung der bekannten Unix-Konventionen kann ausgegangen werden, ein `--` kommt nicht vor!

Bespiele:

`Count -xy -c file`: 3 Flags und 1 Argument!
`Count -c file1 file2`: 1 Flag und 2 Argumente!

Lösung:

```
MODULE MyArgs;
  IMPORT AV := UnixArguments, Write;

  VAR flag: CHAR;
      arg: ARRAY 80 OF CHAR;
      flagCount, argCount: INTEGER;

BEGIN
  flagCount := 0;
  WHILE AV.GetFlag(flag) DO
     INC(flagCount);
  END;
  argCount := 0;
  WHILE AV.GetArg(arg) DO
     INC(argCount);
  END;
  Write.String("Number of flags: ");
  Write.Int(flagCount,0); Write.Ln;
  Write.String("Number of arguments: ");
  Write.Int(argCount,0); Write.Ln;

END MyArgs.
```

Aufgabe 2 (8 Punkte)

Streams

Das folgende Programm WordCount soll von der Standardeingabe *eine* Zeile (maximal 256 Zeichen) einlesen und die Anzahl der darin enthaltenen Worte ausgeben. Ein Wort besteht aus einer beliebigen Zeichenfolgen ohne *blank*, *tab* oder *newline*, so wie es auch durch die Prozeduren `Read.String` bzw. `Read.StringS` (siehe unter "Hinweis") erkannt wird.

```
MODULE WordCount;
   IMPORT Streams, Strings, Read, Write;
   TYPE
      Line = ARRAY 256 OF CHAR;
   VAR
      line, word: Line; w: INTEGER;
      str: Streams.Stream;
BEGIN
      Read.Line(line);
      w := 0;
      Strings.Open(str, line);
      Read.StringS(str, line);
      WHILE str.count > 0 DO
     INC(w);
        Read.StringS(str, line);
      END;
      Write.String("Anzahl Worte: "); Write.Int(w,0); Write.Ln;

END WordCount.
```

Das obige Programm ist zu vervollständigen! Das Lesen von Worten muss mit Hilfe der Prozedur `StringS` aus dem Modul `Read` erfolgen!

Hinweis:

1) Das Modul `Strings` enthält die Funktion: `PROCEDURE Open(VAR stream: Streams.Stream; VAR string: Array OF CHAR);` das eine Stream-Leseverbindung zu einem String herstellt.

2) `PROCEDURE StringS(s: Streams.Stream; VAR string: ARRAY OF CHAR)` aus dem Modul `Read` liest pro Aufruf ein weiteres Wort aus dem Stream `s` in das Array `string` ein.

3) Wurde nichts (mehr) aus dem Stream gelesen (Ende erreicht), so wird die Variable `count` in Streams auf 0 gesetzt (also z.B. `str.count = 0`, wenn `str` vom Typ `Stream` ist.)

Aufgabe 3 (16 Punkte)

File-I/O und Argumentverarbeitung:

(*a*) 8 Punkte

Implementieren Sie die Prozedur `EmptyLines` (siehe unten), die die Anzahl der Leerzeilen in einem durch den Parameter `str` übergebenen *Stream* bestimmt; das Lesen einer Zeile kann mit der Prozedur `LineS` aus dem Modul *Read* erfolgen. Leerzeilen sind dadurch zu erkennen, dass das erste Zeichen in der von `Read` gelieferten Zeile bereits das Null-Byte ist, der String also leer ist.

Hinweis: Blättern Sie kurz eine Seite weiter - die dortigen IMPORT-Anweisungen liefern einen "Fingerzeig"!

```
PROCEDURE EmptyLines(str: Streams.Stream): INTEGER;
    VAR
      count: INTEGER;
      line: ARRAY 256 OF CHAR;
BEGIN
    count := 0;
    Read.LineS(str, line);
    WHILE ~ str.eof DO
      IF line[0] = ASCII.nul THEN
         INC(count);
      END;
      Read.LineS(str, line);
    END;
    RETURN count
END EmptyLines;
```

(*b*) 8 Punkte

Auch wenn Sie Teilaufgabe a) nicht gelöst haben sollten, können Sie mit dieser Aufgabe fortfahren! Unter Verwendung der Prozedur aus a) sollen Sie nun ein Oberon-Programm `Count` schreiben das wie folgt aufgerufen werden kann:

`Count [file1 file2 ...]`

Wird *kein* Dateiname angegeben, so werden die Leerzeilen in der *Standardeingabe* bestimmt und ausgegeben! Kommt mindestens ein Dateiname, so werden die Anzahl der Leerzeilen in den angegegebenen Dateien bestimmt und der Dateiname gefolgt von der Anzahl der Leerzeilen pro Datei ausgegeben. Sollte eine der angegebenen Dateien nicht zum Lesen geöffnet werden können, soll das Programm mit Exit-Status 1 terminieren. Ergänzen Sie das Programm:

```
MODULE Count;
   IMPORT ASCII, Streams, UF := UnixFiles, AV := UnixArguments, Process,
      Read, Write;
  VAR
     file: ARRAY 256 OF CHAR;
     str: Streams.Stream;

  PROCEDURE EmptyLines(str: Streams.Stream): INTEGER;

BEGIN

    IF ~AV.GetArg(file) THEN
       Write.Int(EmptyLines(Streams.stdin),0);
       Write.Ln;
    ELSE
       REPEAT
      IF ~UF.Open(str,file,0,1,NIL) THEN
        Process.Exit(1);
         ELSE
        Write.String(file);
           Write.Int(EmptyLines(str),4);
         END;
      Write.Ln;
       UNTIL ~AV.GetArg(file);
    END

END Count.
```


Aufgabe 4 (8 Punkte)

Rekursion

Gegeben sei folgendes Programm:

```
MODULE Rekursi;
  IMPORT Write;
  VAR x,y,z: INTEGER;

  PROCEDURE Rek(n: INTEGER; VAR b, res: INTEGER);
  BEGIN
     Write.String("n = "); Write.Int(n,0); Write.Ln;
     Write.String("b = "); Write.Int(b,0); Write.Ln;
     Write.String("res = "); Write.Int(res,0); Write.Ln;
     Write.String("----------"); Write.Ln;
     IF n = 1 THEN
     res := res *b;
     RETURN
     END;

     IF ODD(n) THEN
     res := res * b;
     Rek(n-1,b,res)
     ELSE
     b := b * b;

        Rek(n DIV 2, b, res)
     END;
  END Rek;
BEGIN

  x := 6; y := 2; z := 1;
  Rek(x,y,z);
  Write.String("----------"); Write.Ln;
  Write.String("x = "); Write.Int(x,0); Write.Ln;
  Write.String("y = "); Write.Int(y,0); Write.Ln;
  Write.String("z = "); Write.Int(z,0); Write.Ln;
END Rekursi.
```

Geben Sie exakt an, was dieses Programm auf die Standardausgabe schreibt!

```
n = 6
b = 2
res = 1
----------
n = 3
b = 4
res = 1
----------
n = 2
b = 4
res = 4
----------
n = 1
b = 16
res = 4
----------
----------
x = 6
y = 16
z = 64
```

Aufgabe 5 (6 Punkte)

Dynamische Datenstrukturen

In einem Buchhaltungssystem werden Konten geführt, die am Beispiel eines Bankkontos wie folgt aufgebaut seien:

Soll		Bank	Haben
Anfangsbestand	6000,00	Verbindlichkeiten	4000,00
Kasse	2000,00	Kasse	1000,00
Kasse	3000,00		
Waren	4000,00		

Allgemein bestehe also ein Konto aus einer Konto-Bezeichnung sowie einer linken und rechten Seite (SOLL und HABEN). Die Einträge im SOLL und im HABEN bestehen aus einer Kontenbezeichnung und einem Betrag; davon können beliebig viele (vorab nicht bekannt) enthalten sein.

Geben Sie die nötigen Typ-Vereinbarungen in Oberon an, so dass eine Variable angelegt werden kann, die ein konkretes Konto aufnehmen kann. Die Kontenbezeichnung sei hier vereinfachend als ganzzahlige Nummer gegeben, die Beträge können als reelle Zahlen gewählt werden.

Beispiel: `VAR bank: Konto;`

Lösung:

```
TYPE
     Eintrag    = POINTER TO EintragRec;
     EintragRec = RECORD
                    kontenId: INTEGER;
                    betrag: REAL;
                    next: Eintrag;
                  END;
     Konto      = RECORD
                    soll, haben: Eintrag;
                  END;
```


Aufgabe 6 (16 Punkte)

Dynamische Datenstrukturen (Listen)

Sie erhalten in einer Datei volkswirtschaftliche Daten in folgender Struktur:

`year month amount`

Die Datensätze in dieser Datei sind bezogen auf die Zeitangaben aufsteigend sortiert. Zum Lesen dieser Datensätze steht eine Prozedur zur Verfügung, die jeweils eine Zeile aus der Datei liest und die Eingabe in die entsprechenden Variablen aufspaltet:

```
PROCEDURE GetData(VAR year, month: INTEGER; VAR amount: REAL):BOOLEAN;
(* TRUE, wenn Datensatz erfolgreich gelesen,
 * FALSE, sonst
 *)
```

Diese Prozedur ist **nicht** zu realisieren!

Mit Hilfe dieser Prozedur sind die Daten in eine Liste einzulesen, so dass die Elemente ebenfalls aufsteigend bezüglich Zeitangaben sortiert sind.

Die Typ-Vereinbarungen für die Liste seien wie folgt vorgegeben:

```
TYPE
     List =    POINTER TO ListRec;
     ListRec = RECORD
                 year, month: INTEGER;
                 amount: REAL;
                 next: List;
               END;
```

Aufgaben siehe die nächsten Seiten!

(*a*) 8 Punkte

Realisieren Sie die Prozedur `BuildList`, die mit Hilfe der Prozedur `GetData` die Liste aufbaut (die bezgl. Zeitangaben aufsteigend sortiert ist) und via Parameter einen Zeiger auf das erste Element zurückliefert!

<u>Lösung:</u>

```
PROCEDURE BuildList(VAR head: List);
    VAR
       tail, p: List;
       year, month: INTEGER;
       amount: REAL;
BEGIN
       head := NIL; tail := NIL;
       WHILE GetData(year, month, amount) DO
           NEW(p);
           p.year := year;
           p.month := month;
           p.amount := amount;
           p.next := NIL;
           IF tail = NIL THEN
              head := p; tail := p;
           ELSE
              tail.next := p;
           tail := p;
           END
       END;
    END BuildList;
```


(*b*) 8 Punkte

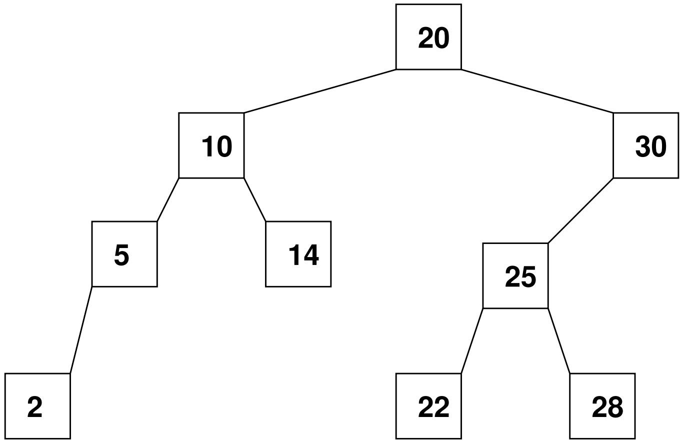
Realisieren Sie eine Prozedur `AvgAmount`, die den Durchschnittswert der in der Liste enthaltenen Beträge (Komponente *amount*) ermittelt.

```
PROCEDURE AvgAmount(list: List; VAR avg: REAL): BOOLEAN;
(* liefert FALSE, wenn Liste leer *)
    VAR p: List; sum: REAL; nr: INTEGER;
BEGIN
    sum := 0.0; nr := 0;
    p := list;
    WHILE p # NIL DO
       sum := sum + p.amount;
       INC(nr);
       p := p.next;
    END;
    IF nr > 0 THEN
        avg := sum / nr;
        RETURN TRUE
    ELSE
        RETURN FALSE
    END;
END AvgAmount;
```

Aufgabe 7 (18 Punkte)

<u>Binärbäume</u>

Gegeben sei folgender binärer Baum, bei dem ganze Zahlen als Schlüssel dienen:



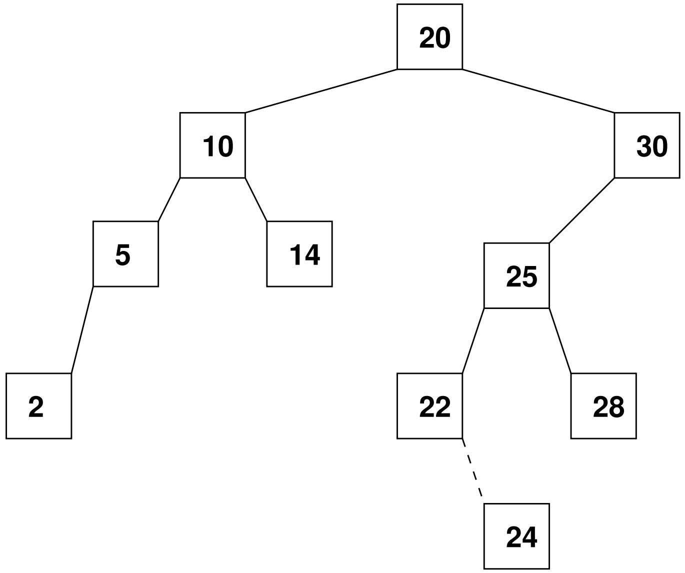
(*a*) 3 Punkte

Bitte kreuzen Sie das jeweils zutreffende an:

Aussage	Richtig	Falsch
Der Baum ist bezüglich der Höhe ausgeglichen.		X
Der Baum ist nach dem Gewicht ausgeglichen.		X
Der Baum ist nach Inorder sortiert.	X	

(*b*) 3 Punkte

Zeichnen Sie auf, wie der Ursprungsbaum aussehen würde, wenn ein Element mit dem Schlüssel 24 hinzukommen würde. Das Element soll als *Blatt* in den Baum eingefügt werden, damit der Baum nicht umstrukturiert werden muss! (Dabei soll die Ordnung im Baum erhalten werden. Er muß aber nicht nach Höhe oder Gewicht ausgeglichen bleiben, falls er das vorher war!)



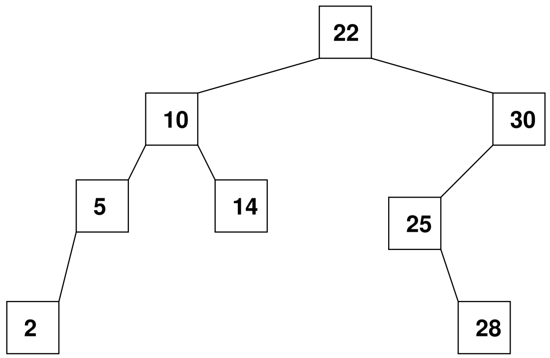
(*c*) 6 Punkte

Geben Sie die Schlüssel des vorgegebenen Baumes in **Postorder** an.

2 5 14 10 22 28 25 30 20

(*d*) 6 Punkte

Zeichnen Sie den Baum, wie er nach dem Löschen des Knotens "20" entsprechend dem in der Vorlesung behandelten Algorithmus aussieht (die Ordnung muss erhalten bleiben, eine evt. vorhandene Ausgeglichenheit jedoch nicht).



oder