Vordiplomklausur zur
Allgemeinen Informatik I/II

27. August 2001

Prof. Dr. H. Neumann
Dr. K. Murmann

Vorname: ____________________ Name: ____________________

Matrikel-Nr.: ____________________

Hinweise

- Lösungen bitte in die Aufgabenblätter schreiben.
- Zur Lösung der 10 Aufgaben stehen Ihnen 3 Stunden (180 Minuten) Bearbeitungszeit zur Verfügung. Insgesamt können Sie 150 Punkte erreichen.
- Erlaubte Hilfsmittel: Skript, Vorlesungsmitschrift und ein Modula-2-Buch.
  Elektronische Hilfsmittel sind nicht erlaubt!

Aufgabe	1	2	3	4	5	6	7	8	9	10	$\sum$
Punkte	10	10	20	15	10	15	25	10	10	25	150
erreicht											
Korrekteur											

1. Aufgabe 10

Antworten Sie auf die folgenden Fragen einfach mit **ja** oder **nein** (Bewertung: **1** Punkt für eine richtige Antwort, **-1** Punkte für eine falsche Antwort, **0** Punkte falls keine Antwort gegeben wurde) :

1. Die Sprache $L = \{1^{7n} \mid n \geq 0\}$ wird von einem endlichen Automaten akzeptiert.
2. Eine `FOR`-Schleife läßt sich stets in eine äquivalente `WHILE`-Schleife transformieren.
3. In einem vollständigen Binärbaum hat jeder Knoten genau zwei Nachfolgerknoten und genau einem Vorgängerknoten.
4. Eine Sprache L mit endlich vielen Wörtern ist regulär.
5. Für einen arithmetischen Ausdruck in Postfix-Notation müssen Klammern als Strukturierungsmittel verwendet werden.
6. Eine Hash-Funktion sollte möglichst surjektiv sein.
7. Innerhalb eines Prozedurrumpfes dürfen von den Eingabeparametern nur die `VAR`-Parameter auf der linken Seite einer Zuweisung stehen.
8. In einem vollständigen binären Suchbaum in dem 2^n Zahlen vom Typ `CARDINAL` gespeichert sind, lässt sich das Maximum dieser Zahlen stets mit dem Aufwand $O(n)$ berechnen.
9. Bei Divide-and-Conquer Verfahren wird die Datenmenge jeweils in separate Teilmengen aufgeteilt und anschliessend jede Teilmenge durch rekursiven Aufruf der Methode bearbeitet.
10. Die Funktion `ORD` ist für die elmentaren Datentypen `CARDINAL`, `BOOLEAN`, `INTEGER` und `REAL` definiert.

Lösung:

```
1. ja       2. ja
3. nein     4. ja
5. nein     6. ja
7. nein     8. ja
9. ja      10. nein
```

2. Aufgabe 10

Beantworten Sie die folgenden Fragen mit **ja** oder **nein** ohne Begründung (Bewertung: **2** Punkt für eine richtige Antwort, **-2** Punkte für eine falsche Antwort, **0** Punkte falls keine Antwort gegeben wurde) :

1. $f(n) = \frac{\log n}{n}$. Gilt $f \in O(n^{-2})$?
2. $f(n) = (\log n)^2$. Gilt $f \in O(\log(n))$?
3. $f(n) = n\log(n^2) + n^2$. Gilt $f \in O(n^2)$?
4. $f(n) = 15n$. Dann ist $f \in O(n\log(n))$.
5. $f(n) = 10^n$. Dann ist $f \in O(2^n)$.

Lösung:

1. **nein**
2. **nein**
3. **ja**
4. **ja**
5. **nein**

3. Aufgabe 20

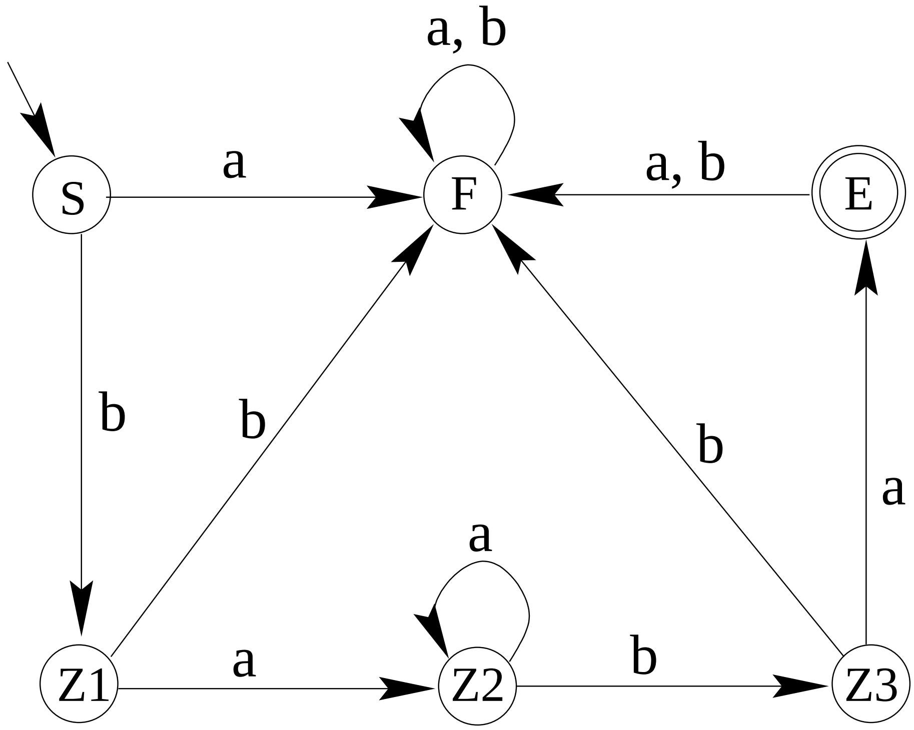
Es sei $L = \{ba^n ba \mid n \in \mathbb{N}\}$. ($\mathbb{N} = \{1, 2, 3, \ldots\}$)

1. Bestimmen Sie einen regulären Ausdruck α mit $L(\alpha) = L$. 4
2. Konstruieren Sie einen endlichen Automaten M, der L akzeptiert. Geben Sie den Zustandsgraphen des Automaten an. 4
3. Finden Sie eine Grammatik G mit $L(G) = L$ und notieren Sie diese in EBNF. 4
4. Finden Sie eine reguläre Grammatik G mit $L(G) = L$. 4
5. Geben Sie ein Syntaxdiagramm an, das genau die Wörter w aus der Sprache L beschreibt. 4

Lösung:

(a) baa*ba

(b) Zustände: {S, Z1, Z2, Z3, F, E}
Startzustand: S
Endzustand: E
Alphabet: {a, b}

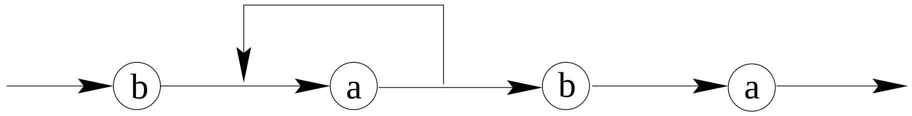


(c) L := 'b' 'a' {'a'} 'b' 'a'.

(d) Terminalsymbole: { a, b}
Nichterminalsymbole: { S, X, Y, Z}
Startsymbol: S
Produktionsregeln:

`S -> bX, X -> aX, X -> aY, Y -> bZ, Z -> a`

(e) Syntaxdiagramm:



4. Aufgabe

Gegeben sei das folgende Modula-2-Programmstück:

```
CONST  NMAX = 10;
VAR    arr     : ARRAY[1..NMAX] OF CARDINAL;
       p, i    : CARDINAL;

.
.
(* Vorbelegung von ARRAY arr *)
FOR i:= 1 TO NMAX DO
    arr[i] := ....
END;

(* Test: Ist arr aufsteigend sortiert ? *)
.
.
.
```

Es soll getestet werden ob die im Feld `arr` gespeicherten Zahlen aufsteigend sortiert sind. Ist dies der Fall, so soll die Variable `p` den Wert `NMAX` bekommen, sonst soll in der Variablen `p` der kleinste Index `i` mit `arr[i] > arr[i+1]` gespeichert werden.
Realisieren Sie diesen Test jeweils mittels einer `FOR`-, einer `REPEAT`-, sowie mittels einer `WHILE`-Schleife (jeweils 5 Punkte).

Lösung

1. `FOR`-Schleife:

   ```
   p := NMAX;
   FOR i := NMAX-1 TO 1 BY -1 DO
      IF arr[i] > arr[i+1] THEN
         p := i;
      END;
   END;
   ```

2. `REPEAT`-Schleife:

   ```
   p := 0;
   REPEAT
      INC(p);
   UNTIL (p = NMAX) OR (arr[p] > arr[p+1]);
   ```

3. `WHILE`-Schleife:

   ```
   p := 1;
   WHILE (p < NMAX) & (arr[p] <= arr[p+1]) DO
      INC(p);
   END;
   ```

5. Aufgabe 10

Gegeben sei das folgende Modula-2-Programm.

```
MODULE aufgabe;
   FROM InOut IMPORT WriteInt, WriteLn;

   VAR  a, b, c, d, z : INTEGER;

   PROCEDURE F(VAR a, b : INTEGER; c : INTEGER) : INTEGER;
   BEGIN
      b := a + c - b - d;
      IF c < b THEN
         RETURN b - c
      ELSE
         RETURN c - b
      END
   END F;

BEGIN
   a := 3;      b := 1;
   c := 7;      d := b;
   z := F(b, d, a) + c;

   WriteInt(a,3); WriteLn;
   WriteInt(b,3); WriteLn;
   WriteInt(c,3); WriteLn;
   WriteInt(d,3); WriteLn;
   WriteInt(z,3); WriteLn

END aufgabe.
```

1. Welche Ausgaben erzeugt das Programm? 5

2. Kernstück des Programms ist die Funktionsprozedur `F`, die als Ergebnis ihrer Ausführung einen Wert vom Typ `INTEGER` liefert. Verändern Sie das Programm derart, dass `F` eine normale Prozedur (ohne Rückgabewert) wird, die das Ergebnis über einen formalen Referenz-Parameter liefert. Passen Sie die Aufrufe und Berechnungen im Modul-Rumpf entsprechend an. 5

Lösung

1. Ausgabe: 3 1 7 2 8

2. ```
MODULE aufgabe;
  FROM InOut IMPORT WriteInt, WriteLn;

  VAR a, b, c, d, z : INTEGER;

  PROCEDURE F(VAR a, b : INTEGER; c : INTEGER; VAR e : INTEGER); (* <= *)
  BEGIN
    b := a + c - b - d;
    IF c < b THEN
      e := b - c;   (* <= *)
    ELSE
      e := c - b;   (* <= *)
    END
  END F;

BEGIN
  a := 3;   b := 1;
  c := 7;   d := b;
  F(b, d, a, z);   (* <= *)
  z := z + c;   (* <= *)

  WriteInt(a,3); WriteLn;
  WriteInt(b,3); WriteLn;
  WriteInt(c,3); WriteLn;
  WriteInt(d,3); WriteLn;
  WriteInt(z,3); WriteLn

END aufgabe.
```
```

6. Aufgabe 15

Die Funktion $\Phi : \mathbb{N} \to \mathbb{N}$ definiert durch

$$\Phi(1) = 1, \quad \Phi(n) = \sum_{l=1}^{n-1} \Phi(l)\Phi(n-l) \quad \text{falls} \quad n > 1$$

ist durch die folgende Modula-2-Prozedur `F` implementiert:

```
PROCEDURE F(n : CARDINAL) : CARDINAL;
VAR l, summe: CARDINAL;
BEGIN
     IF  n = 1 THEN
         RETURN 1
     ELSE
         summe := 0;
         FOR l := 1 TO n-1 DO
             summe := summe + F(l)*F(n-l)
         END;
         RETURN summe
     END;
END F;
```

1. Wie groß ist der Wert von $\Phi(4)$? Führen Sie hierzu eine Handsimulation für den Prozeduraufruf `F(4)` durch!

   Es bezeichne $T(F,n)$ die Zahl der Inkarnationen der Funktionsprozedur `F` die zur Berechnung von $\Phi(n)$ durch `F` notwendig sind! Wie groß ist $T(F,4)$? 5

2. Schreiben sie eine Modula-2-Prozedur `SmartF` zur Berechnung Φ. Dabei sollen zur Berechnung von $\Phi(n)$ durch `SmartF` nur noch höchstens etwa halb so viele, genau $\lceil T(F,n)/2 \rceil$, Inkarnationen der Funktionsprozedur `SmartF` notwendig sein. 10

Lösung:

1. ```
   F(4) = F(1)*F(3) + F(2)*F(2) + F(3)*F(1)
        = 1*F(3) + F(2)*F(2) + F(3)*F(1)
        = F(1)*F(2) + F(2)*F(1) + F(2)*F(2) + F(3)*F(1)
        = 1*F(2) + F(2)*F(1) + F(2)*F(2) + F(3)*F(1)
        = F(1)*F(1) + F(2)*F(1) + F(2)*F(2) + F(3)*F(1)
        = 1*F(1) + F(2)*F(1) + F(2)*F(2) + F(3)*F(1)
        = 1 + F(2)*F(1) + F(2)*F(2) + F(3)*F(1)
        = 1 + F(1)*F(1)*F(1) + F(2)*F(2) + F(3)*F(1)
        = 1 + 1*F(1)*F(1) + F(2)*F(2) + F(3)*F(1)
        = 1 + 1*F(1) + F(2)*F(2) + F(3)*F(1)
        = 1 + 1 + F(2)*F(2) + F(3)*F(1)
        = 2 + F(1)*F(1)*F(2) + F(3)*F(1)
        = 2 + 1*F(1)*F(2) + F(3)*F(1)
        = 2 + 1*F(2) + F(3)*F(1)
        = 2 + F(1)*F(1) + F(3)*F(1)
        = 2 + 1*F(1) + F(3)*F(1)
        = 2 + 1 + F(3)*F(1)
        = 3 + (F(1)*F(2) + F(2)*F(1))*F(1)
        = 3 + (1*F(2) + F(2)*F(1))*F(1)
        = 3 + (1*F(1)*F(1) + F(2)*F(1))*F(1)
        = 3 + (1*F(1) + F(2)*F(1))*F(1)
        = 3 + (1 + F(2)*F(1))*F(1)
        = 3 + (1 + F(1)*F(1)*F(1))*F(1)
        = 3 + (1 + 1*F(1)*F(1))*F(1)
        = 3 + (1 + 1*F(1))*F(1)
        = 3 + (1 + 1)*F(1)
        = 3 + 2*1 = 5
   ```

   Ausserdem gilt

   $$T(F,1) = 1, \quad T(F,n) = 1 + 2 * \sum_{l=1}^{n-1} T(F,l) \quad \text{falls} \quad n > 1$$

   Also T(F,2) = 3, T(F,3) = 9 und **T(F,4) = 27**.
   ```

2.
```
PROCEDURE SmartF(n : CARDINAL) : CARDINAL;
VAR l, summe: CARDINAL;
BEGIN
     IF  n = 1 THEN
         RETURN 1
     ELSE
         IF ODD(n) THEN
            summe := 0;
         ELSE
            l := SmartF(n DIV 2);
            summe := l * l;
         END;
         FOR l := 1 TO (n-1) DIV 2 DO
             summe := summe + 2*SmartF(l)*SmartF(n-l)
         END;
         RETURN summe
     END;
END SmartF;
```

7. Aufgabe 25

Gegeben sei eine Liste `li` in der Zahlen vom Typ `CARDINAL` gespeichert sind.

1. Schreiben Sie eine **iterative** Prozedur

   ```
   PROCEDURE teilen(VAR li, ligerade, liungerade : LIST);
   ```

   die die Liste `li` aufteilt in zwei Listen `ligerade` bzw. `liungerade`. In die Liste `ligerade` sollen die Elemente aus `li` mit geradzahligem Eintrag gehängt werden, in die andere Liste die Elemente mit ungradzahligem Eintrag. Aufrufe von `NEW` sind nicht erlaubt. 10

2. Schreiben Sie eine **rekursive** Prozedur

   ```
   PROCEDURE umdrehen(VAR li : LIST);
   ```

   die die Reihenfolge der Elemente, d.h. die Zeiger, in der Liste `li` umdreht, ohne eine zweite Liste zu verwenden (d.h. Aufrufe von `NEW` sind nicht erlaubt). 10

3. Realisieren Sie `umdrehen` als **iterative** Prozedur. 5

Lösung:

1.
```
PROCEDURE teilen(VAR li, ligerade, liungerade : LIST);
   VAR
      p : LIST;
BEGIN
   ligerade := NIL;
   liungerade := NIL;
   WHILE li <> NIL DO
      p := li^.next;
      IF ODD(li^.data) THEN
         li^.next := liungerade;
         liungerade := li;
      ELSE
         li^.next := ligerade;
         ligerade := li;
      END;
      li := p;
   END;
END teilen;
```

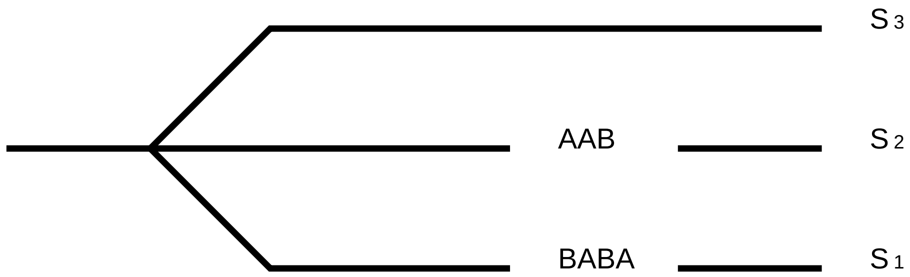
2.
```
PROCEDURE umdrehen(VAR li : LIST);
   VAR
      p : LIST;
BEGIN
   IF (li <> NIL) & (li^.next <> NIL) THEN
      p := li^.next;
      umdrehen(p);
      li^.next^.next := li;
      li^.next := NIL;
      li := p;
   END;
END umdrehen;
```

3.
```
PROCEDURE umdrehen(VAR li : LIST);
   VAR
      v, n : LIST;
BEGIN
   v := NIL;
   WHILE li <> NIL DO
      n := li^.next;
      li^.next := v;
      v := li;
      li := n;
   END;
   li := v;
END umdrehen;
```

8. Aufgabe

10

In einem Sackbahnhof mit drei Gleisen befinden sich in den Gleisen S_1 und S_2 zwei Züge mit Waggons für den Zielbahnhof A bzw. B. Gleis S_3 sei leer.



Betrachten Sie S_1, S_2, S_3 als Stapel (Stack). Erstellen Sie ein Programmstück in `Modula-2` unter Verwendung der unten angegebenen (Funktions-)Prozeduren, das zwei beliebige aus Waggons für A und B bestehende Züge so umordnet, dass anschließend in S_1 alle Waggons für A und in S_2 alle Waggons für B stehen.
Sie können davon ausgehen, dass `Stapel, Waggon, Zielbahnhof` im Hauptprogramm definiert sind.

```
PROCEDURE push(VAR S : Stapel; X : Waggon);
(* stellt X auf S ab  *)

PROCEDURE pop(VAR S : Stapel) : Waggon ;
(* Wenn S nicht leer ist, liefert pop den vordersten Waggon von S und
   entfernt ihn von S; Fehler sonst *)

PROCEDURE top(S : Stapel) : Zielbahnhof;
(* top liefert den Zielbahnhof des vordersten Waggons in S,
   ohne ihn zu entfernen *)

PROCEDURE leer(S : Stapel) : BOOLEAN;
(* leer liefert TRUE, wenn S leer; FALSE sonst *)
```

Lösung:

```
(* verteile S2 auf S1 (die B's) und S3 (die A's) *)
WHILE NOT leer(S2) DO
   IF top(S2) = A THEN
      push(S3, pop(S2));
   ELSE
      push(S1, pop(S2));
   END;
END;

(* verteile S1 auf S2 (die B's) und S3 (die A's) *)
WHILE NOT leer(S1) DO
   IF top(S1) = A THEN
      push(S3, pop(S1));
   ELSE
      push(S2, pop(S1));
   END;
END;

(* bringe alle Elemente von S3 (nur A's) nach S1 *)
WHILE NOT leer(S3) DO
   push(S1, pop(S3));
END;
```

9. Aufgabe 10

Gegeben sei eine Datenmenge in einem Feld. Diese Datenmenge soll mithilfe des Sortieralgorithmus `BubbleSort` in **aufsteigender** Reihenfolge sortiert werden. Dieser Algorithmus ist gegeben:

```
CONST laenge = 8;
TYPE FELD = ARRAY [1..laenge] OF CARDINAL;

PROCEDURE BubbleSort(VAR arr : FELD);
VAR
  index, pos, puffer: CARDINAL;
BEGIN
   FOR index := 2 TO laenge DO
       FOR pos := laenge TO index BY -1 DO
           IF arr[pos-1] > arr[pos] THEN
              puffer     := arr[pos-1];
              arr[pos-1] := arr[pos];
              arr[pos]   := puffer
           END
       END
   END
END BubbleSort;
```

1. Führen Sie für das folgende Feld von 8 Elementen 5

17	19	20	1	7	13	4	3

   eine Handsimulation des Sortieralgorithmus durch. Geben Sie den Inhalt des Feldes am Ende eines jeden Durchlaufs der äußeren `FOR`-Schleife an (in Tabellenform).

2. Erweitern Sie nun die Prozedur `BubbleSort` so, dass die Prozedur beendet wird, sobald die Sortierschlüssel im Feld `arr` in aufsteigend sortierter Reihenfolge vorliegen, also keine Vertauschungen vorgenommen werden. 5

Lösung:

1.

17	19	20	1	7	13	4	3

20	17	10	13	1	7	4	3

20	17	13	10	7	1	4	3

20	17	13	10	7	4	1	3

20	17	13	10	7	4	3	1

20	17	13	10	7	4	3	1

20	17	13	10	7	4	3	1

2.

```
CONST laenge = 8;
TYPE FELD = ARRAY [1..laenge] OF CARDINAL;

PROCEDURE BubbleSort(VAR arr : FELD);
VAR
  index, pos, puffer: CARDINAL;
  ready : BOOLEAN;
BEGIN
   index := 2;
   ready := FALSE;
   WHILE ~ready & (index <= laenge) DO
       ready := TRUE;
       FOR pos := laenge TO index BY -1 DO
           IF arr[pos-1] > arr[pos] THEN
              puffer      := arr[pos-1];
              arr[pos-1]  := arr[pos];
              arr[pos]    := puffer
              ready := FALSE;
           END
       END
       INC(index);
   END
END BubbleSort;
```

10. Aufgabe 25

1. Schreiben Sie eine **rekursive** Funktionsprozedur 10

   ```
   PROCEDURE GanzerBaum(t : TREE) : BOOLEAN;
   ```

   die überprüft ob es sich bei bei dem Baum `t` um einen ganzen Binärbaum handelt (d.h. alle Knoten haben entweder 0 oder 2 Kinderknoten) und genau in diesem Fall den Wert `TRUE` zurück gibt.

2. Schreiben Sie eine **rekursive** Funktionsprozedur 10

   ```
   PROCEDURE Hoehe(t : TREE) : CARDINAL;
   ```

   die die Höhe des nichtleeren Baumes `t` bestimmt. Die Höhe eines Baumes ist definiert als die Länge des längsten Pfades zwischen der Wurzel und den Blattknoten.

3. Schreiben Sie eine **rekursive** Funktionsprozedur 5

   ```
   PROCEDURE LoescheBaum(t : TREE);
   ```

   die den Speicherplatz für den gesamten Baum `t` mittels `DISPOSE` freigibt.

Lösung:

1.
   ```
   PROCEDURE GanzerBaum(t : TREE) : BOOLEAN;
   BEGIN
      IF t = NIL THEN
         RETURN TRUE;
      ELSIF t^.left = NIL THEN
         RETURN t^.right = NIL;
      ELSE
         RETURN (t^.right <> NIL) & GanzerBaum(t^.left) & GanzerBaum(t^.right);
      END;
   END GanzerBaum;
   ```

2. ```
PROCEDURE Hoehe(t : TREE) : CARDINAL;
   VAR
      h1, h2 : CARDINAL;
BEGIN
   IF t^.left = NIL THEN
      h1 := 0;
   ELSE
      h1 := 1 + Hoehe(t^.left);
   END;

   IF t^.right = NIL THEN
      h2 := 0;
   ELSE
      h2 := 1 + Hoehe(t^.right);
   END;

   IF h1 > h2 THEN
      RETURN h1;
   ELSE
      RETURN h2;
   END;
END Hoehe;
```

3. ```
PROCEDURE LoescheBaum(t : TREE);
BEGIN
   IF t <> NIL THEN
      LoescheBaum(t^.left);
      LoescheBaum(t^.right);
      DISPOSE(t);
   END;
END LoescheBaum;
```