

### Musterlösung

Aufgaben sind in schwarzer Times-New-Roman- und Arial-Schrift, Lösungen in *blauer Monotype-Corsiva-Schrift* gehalten.

Prüfer: Prof. Dr. Karlheinz Hug

Prüfungsdauer: 90 Minuten

Zugelassene Hilfsmittel: Alle (Skripten, Übungsmaterial, Literatur,...)

Anzahl der Aufgabenseiten: 11

Aufgabe	Punkte		Aufgabe	Punkte	
	möglich	erreicht		möglich	erreicht
1	18		6	10	
2	6		7	18	
3	6		8	14	
4	10		9	19	
5	7				
<b>Summe möglich:</b>		<b>108</b>	<b>Summe erreicht:</b>		
			<b>Note:</b>		

- Zum „Bestehen“ der Probepfprüfung sind **45**, für die Note „Eins“ **90** Punkte erforderlich.
- Bearbeiten Sie die Aufgaben auf dazu freigelassenen Stellen! Tragen Sie in Tabellen und an punktierten Stellen ... im Text passende Angaben ein! Der Platz reicht normalerweise; vermeiden Sie Extrablätter!
- Aufgabe 3 bezieht sich auf das Modul von Aufgabe 2. Die anderen Aufgaben, meist auch Teilaufgaben, sind unabhängig voneinander lösbar.
- Sie haben am Ende der Probepfprüfung diese Musterlösung erhalten und können damit Ihre Ergebnisse selbst bewerten.



**Viel Erfolg!**

### Aufgabe 1 Syntax beschreiben

(Punkte: 18 |.....) (1) Eine kleine Sprache ist in EBNF-Notation durch folgende Regeln gegeben.

$$S = Z \mid Z \text{ „+“ } Z.$$

$$Z = \text{ „1“ } \mid \text{ „2“ } \mid \text{ „3“}.$$

Schreiben Sie alle Wort er dieser Sprache auf!

*Sprache = {1, 2, 3, 1+1, 1+2, 1+3, 2+1, 2+2, 2+3, 3+1, 3+2, 3+3}*

Vereinfachen Sie die erste Regel mittels einer Option!

$$S = Z [ \text{ „+“ } Z ].$$

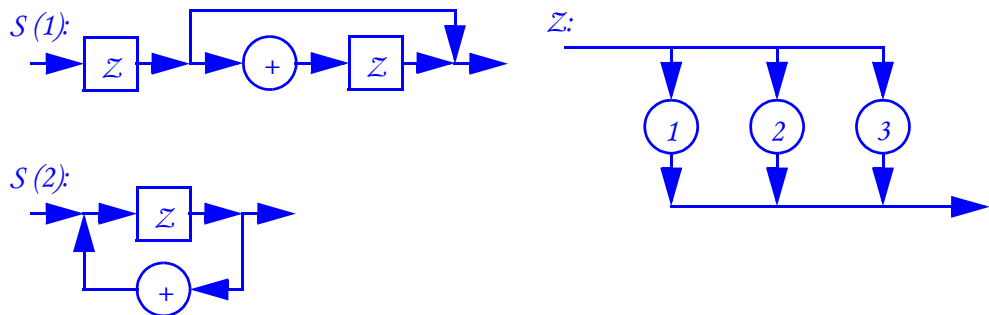
(2) Die erste Regel in (1) wird durch

$$S = Z \{ \text{ „+“ } Z \}.$$

ersetzt. Beschreiben Sie verbal, wie die Wort er dieser Sprache aussehen!

*Jedes Wort beginnt mit einer der Ziffern 1, 2, 3, gefolgt von beliebig vielen (auch null) Summanden der Form +1, +2 oder +3.*

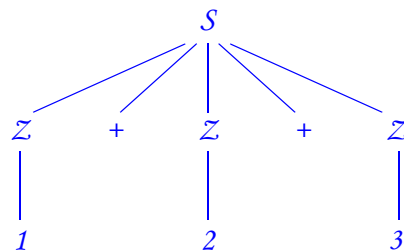
(3) Geben Sie zu den EBNF-Regeln aus (1) - (2) entsprechende Syntaxdiagramme an!



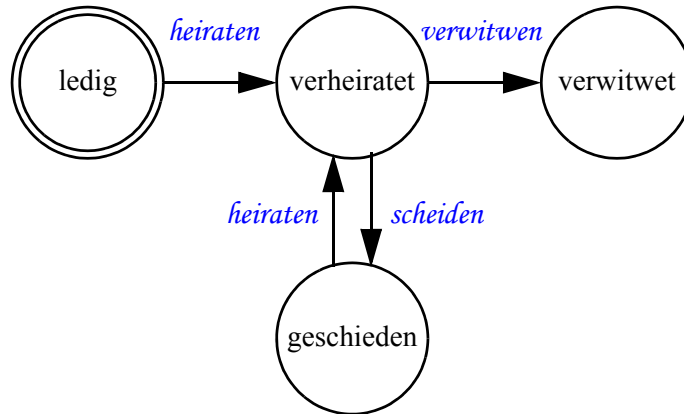
(4) Geben Sie zu dem Wort

„1+2+3“

nach den EBNF-Regeln aus (2) einen Zerteilungsbaum an!



- (5) Geben Sie zum Zustandsdiagramm einen regulären EBNF-Ausdruck an, der alle möglichen Reihenfolgen von Zuständen mit Anfangszustand „ledig“ spezifiziert!

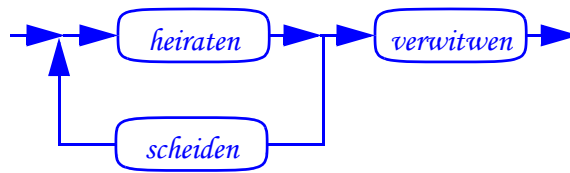


*ledig verheiratet { geschieden verheiratet } verwitwet*

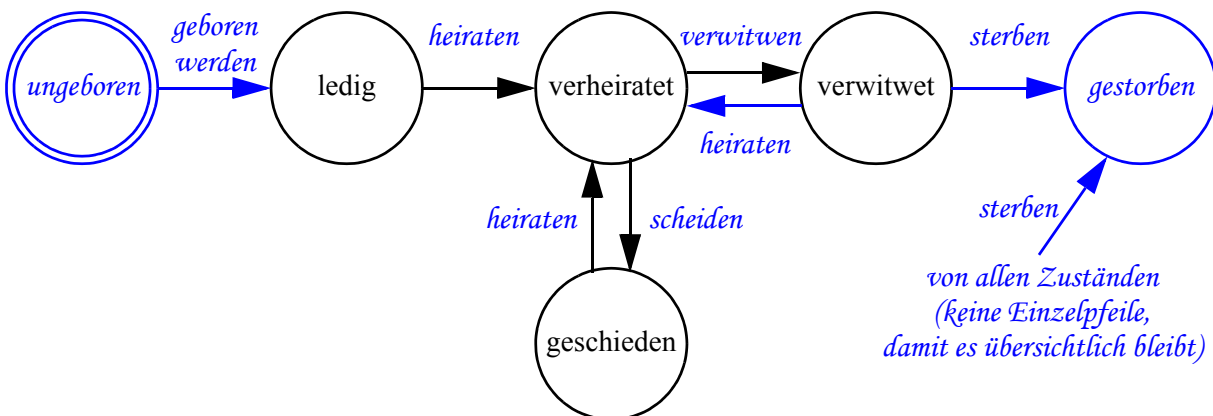
- (6) Beschriften Sie die Pfeile im Zustandsdiagramm von (5) mit passenden Namen für Aktionen und geben Sie einen regulären EBNF-Ausdruck an, der alle möglichen Reihenfolgen von Aktionen spezifiziert!

*heiraten { scheiden heiraten } verwitwen*

- (7) Geben Sie zum EBNF-Ausdruck aus (6) ein entsprechendes Syntaxdiagramm an!



- (8) **Hausaufgabe nach Probepfprüfung** (Extrapunkte: 22): Ergänzen Sie das Zustandsdiagramm um die Zustände „ungeboren“ und „gestorben“, passende Pfeile und Beschriftungen der neuen Pfeile und führen Sie die Teilaufgaben (5), (6), (7) für das so erweiterte Zustandsdiagramm aus!



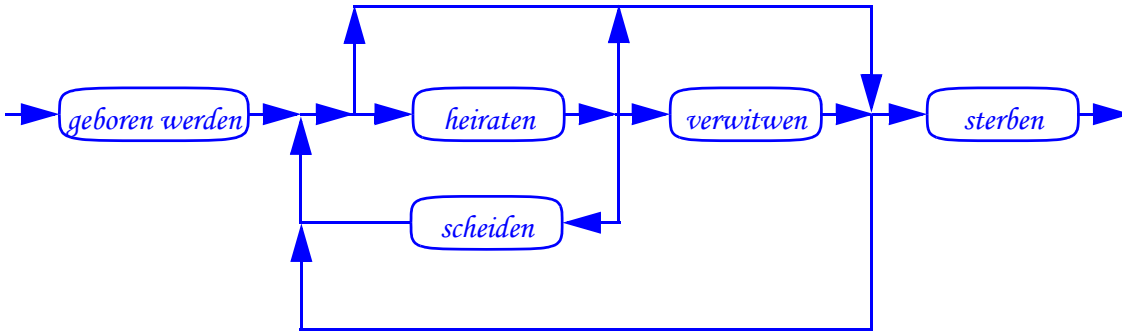
*von allen Zuständen  
(keine Einzelfeile,  
damit es übersichtlich bleibt)*

*Auch Verwitwete können wieder heiraten! Weil „sterben“ in jedem Zustand möglich ist, werden die regulären Ausdrücke relativ komplex;*

*(5): ungeboren [ ledig [ verheiratet { ( geschieden | verwitwet ) verheiratet } [ geschieden | verwitwet ] ] ] gestorben*

(6): *geboren werden* [ *heiraten* [ { (*scheiden* | *verwitwen*) *heiraten* }  
 [ *scheiden* | *verwitwen* ] ] ] *sterben*

(7):



## Aufgabe 2 Reihung vertraglich spezifizieren

(Punkte: 6 |.....) Erganzen Sie die *Cleo*-Spezifikation des Moduls, das eine Reihung modelliert, mit Invarianten, Vor- und Nachbedingungen und Kommentaren zu Bedingungen! Element ist ein beliebiger Typ. Die Reihung enthalt Eintrage von Elementen; sie realisiert eine Abbildung von Indizes auf Elemente.

**Programm 2.1**  
 Cleo: Reihung

```

MODULE Array
    QUERIES
        Capacity : NATURAL (* Konstante Maximalzahl von Eintragen *)
        Item (IN index : NATURAL) : Element (* Falls Position index gultig, zugehoriges Element *)
        PRE
            index < Capacity (* Position index gultig *)
        POST
            (* Has (result) auskommentiert, da Item elementar, Has abgeleitet *)
        END
        Has (IN x : Element) : BOOLEAN (* Ist das Element x enthalten? *)
        POST
            result IMPLIES
                EXISTS index : NATURAL IT_HOLDS (index < Capacity) AND (Item (index) = x)
            END
        END
    INVARIANTS
        Capacity > 0 (* Nie leer *)
    ACTIONS
        Put (IN index : NATURAL; IN x : Element) (* Fuge x an der Position index ein. *)
        PRE
            index < Capacity (* Position index gultig *)
        POST
            Has (x); (* x ist enthalten *)
            Item (index) = x (* x steht an Position index *)
        END
    END Array
    
```

### Aufgabe 3 Reihung mit Zusicherungen testen

(Punkte: 6 |.....)

Die CP-ahnlichen Anweisungen testen das Reihungsmodul aus Aufgabe 2. Dabei ist Element gleich INTEGER. Erganzen Sie die Zusicherungen!

#### Programm 3.1

CP: Reihungstest

```

FOR index := 0 TO Array.Capacity - 1 DO
  Array.Put (index, index * index);

  ASSERT (Array.Has (index * index));

  ASSERT (Array.Item (index) = index * index)
END;
FOR index := 0 TO Array.Capacity - 1 DO
  Array.Put (index, Array.Item (index) + 1);

  ASSERT (Array.Has (index * index + 1));

  ASSERT (Array.Item (index) = index * index + 1)
END

```

### Aufgabe 4 Rekursionsformel und Schleife konstruieren

(Punkte: 10 |.....)

Erganzen Sie die Nachbedingungen der *Cleo*-Abfrage zu einer Rekursionsformel fur die Quersumme der ganzen Zahl *n*!

#### Programm 4.1

Cleo: Quersumme

```

QUERIES
  SumOfTheDigits (IN n : INTEGER) : INTEGER
  POST

  (n < 0) IMPLIES (result = SumOfTheDigits (-n));

  ((0 <= n) AND (n < 10)) IMPLIES (result = n);

  (10 <= n) IMPLIES
    (result = SumOfTheDigits (n DIV 10) + SumOfTheDigits (n MOD 10))
END

```

Skizzieren Sie einen Algorithmus, der die Rekursionsformel mit einer Bedingungs-  
schleife realisiert!

```

n := ABS (n);
result := 0;
WHILE n > 0 DO
  INC (result, n MOD 10);
  n := n DIV 10
END

```

### Aufgabe 5 Typregeln anwenden

(Punkte: 7 |.....)

Die CP-Vereinbarungen

VAR i : INTEGER; x, y : REAL;

vorausgesetzt, zeigen Sie in folgender Zuweisung die Typen aller Größen und Ausdrücke sowie Typanpassungen und Typfehler an!

```
i := (x + 4) & (y > 0)
    INT   REAL   INT   REAL   INT
           implizit
           angepasst an
           REAL
                   implizit
                   angepasst an
                   REAL
                           REAL
                           BOOLEAN
```

*Typfehler, da beide Operanden von & vom Typ BOOLEAN sein müssen*

*Zweiter Typfehler, da der Ausdruck auf der rechten Seite der Zuweisung nicht vom Typ INTEGER ist bzw. nicht an INTEGER anpassbar ist.*

Welches Werkzeug erkennt Typfehler zu welchem Zeitpunkt?

*Der Übersetzer zur Übersetzungszeit. (Typprüfung gehört zur statischen Semantik.)*

### Aufgabe 6 Sprachkonstrukte kennen

(Punkte: 10 |.....)

Begründen Sie zu den Aussagen, warum sie richtig bzw. falsch sind!

Aussage	Begründung
<i>In einer Anweisung können Ausdrücke vorkommen.</i>	<input checked="" type="checkbox"/> Richtig <input type="checkbox"/> Falsch, weil z.B. auf der rechten Seite einer Zuweisung ein Ausdruck steht: Variable := Ausdruck. Auch Bedingungen bei IF und WHILE sind Ausdrücke.
<i>In einem Ausdruck können Anweisungen vorkommen.</i>	<input type="checkbox"/> Richtig <input checked="" type="checkbox"/> Falsch, weil die Auswertung eines Ausdrucks einen Wert liefern, aber nichts verändern soll, während Anweisungen wie Zuweisungen und Aufrufe gewöhnlicher Prozeduren Werte (Zustände) von Variablen ändern, aber keine Werte liefern.

Die englische Bezeichnung für Anweisung lautet *statement*.

Die deutsche Bezeichnung für *assertion* lautet *Zusicherung*.

Übersetzen Sie den folgenden Abschnitt aus dem *Component Pascal Language Report*!

*A data type determines the set of values which variables of that type may assume, and the operators that are applicable. A type declaration associates an identifier with a type. In the case of structured types (arrays and records) it also defines the structure of variables of this type. A structured type cannot contain itself.*

*Ein Datentyp bestimmt die Menge von Werten, die Variable dieses Typs annehmen können, und die anwendbaren Operatoren. Eine Typvereinbarung verbindet einen Namen mit einem Typ. Im Fall strukturierter Typen (Reihungen und Verbunde) definiert sie auch die Struktur von Variablen dieses Typs. Ein strukturierter Typ kann sich nicht selbst enthalten.*

## Aufgabe 7 Programmstucke verbessern

(Punkte: 18 |.....) Die gegebenen Programmstucke sind durch einfachere, aber aquivalente Programmstucke zu ersetzen, d.h. ihr Wert bzw. Effekt muss erhalten bleiben.

Vereinfachen Sie den booleschen Ausdruck so weit wie moglich!

$$\begin{aligned}
 &(\sim(x < 8) \ \& \ (y > 9)) \ \text{OR} \ ((x \geq 8) \ \& \ \sim(y \leq 9)) \\
 &((x \geq 8) \ \& \ (y > 9)) \ \text{OR} \ ((x \geq 8) \ \& \ (y > 9)) \\
 &(x \geq 8) \ \& \ (y > 9)
 \end{aligned}$$

*Bemerkung: Die Regel lautet:  $a \ \text{OR} \ a = a$*

Vereinfachen Sie die Anweisungen so weit wie moglich!

Anweisung	Verbesserte Fassung
<pre> IF k &lt; 3 THEN   x := 4 * a;   y := 5 * x + k;   z := 6 - y ELSE   x := 4 * a;   y := x + 7 * k;   z := 6 - y END                     </pre>	<pre> x := 4 * a; IF k &lt; 3 THEN   y := 5 * x + k ELSE   y := x + 7 * k END; z := 6 - y                     </pre> <p><i>Bemerkung: Anweisungen, die in allen Zweigen vorkommen, kann man ggf. vor- bzw. nachstellen.</i></p>
<pre> IF ~small THEN   IF ~big THEN     medium := TRUE   ELSE     medium := FALSE   END ELSE   medium := FALSE END                     </pre>	<pre> medium := ~(small OR big)                     </pre> <p><i>Bemerkung: Auswahlanweisungen, die in jedem Zweig nur ein und derselben booleschen Variablen einen Wert zuweisen, lassen sich stets auf nur eine Zuweisung reduzieren.</i></p> <pre> medium := ~small &amp; ~big                     </pre> <p><i>ist auch richtig, aber weiter zu vereinfachen.</i></p>

*Losungsweg zur dritten Teilaufgabe:*

*Zuerst vereinfachen wir die geschachtelte IF-Anweisung zu einer Zuweisung:*

```

IF ~small THEN
  medium := ~big
ELSE
  medium := FALSE
END
                    
```

*Der kreative Schritt ist, die Ausdrucke auf den rechten Seiten der Zuweisungen geschickt aquivalent umzuformen (zu erweitern). Mit  $a = \text{TRUE} \ \& \ a$  und  $\text{FALSE} = \text{FALSE} \ \& \ b$  erhalten wir:*

```

IF ~small THEN
  medium := TRUE & ~big
ELSE
  medium := FALSE & ~big
END
                    
```

*Im THEN-Zweig gilt  $\sim\text{small} = \text{TRUE}$ , im ELSE-Zweig  $\sim\text{small} = \text{FALSE}$ . Damit erhalten wir:*

```

IF ~small THEN
    medium := ~small & ~big
ELSE
    medium := ~small & ~big
END
    
```

Damit ergibt sich

```

medium := ~small & ~big
    
```

und mit einer de morganschen Regel

```

medium := ~(small OR big)
    
```

Bei diesem Lösungsweg in 5 Schritten ist wohl Schritt 2 der schwierigste, weil kreative. Eine einfachere, schematisch anzuwendende Lösungsmethode ist das Aufstellen einer Wahrheitstabelle der Eingangsgrößen small und big und der Ausgangsgröße medium. Die Werte für medium ergeben sich aus dem Algorithmus.

small	big	medium
FALSE	FALSE	TRUE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	FALSE

Aus der Tabelle lässt sich direkt ablesen:  $medium := \sim(small \text{ OR } big)$

Verbessern Sie die (korrekte) Implementation der Funktionsprozedur zum Testen des Sortiertseins so, dass sie mit einer Variable und zwei Zuweisungen weniger auskommt!

**Programm 7.1**

CP: Sortiertsein prüfen

```

PROCEDURE Sorted (IN x : ARRAY OF Comparable) : BOOLEAN;
    (*! Postcondition: result = the elements of x are ordered. !*)
    VAR
        result : BOOLEAN;
        i      : Index;
    BEGIN
        result := TRUE;
        i := 0;
        WHILE result & (i < LEN (x) - 1) DO
            result := (x [i] <= x [i + 1]);
            INC (i)
        END;
        RETURN result
    END Sorted;
    
```

result ist verzichtbar. Beachten Sie die Reihenfolge der Fortsetzungsbedingungen - kurze Auswertung ist hier wesentlich!

```

PROCEDURE Sorted (IN x : ARRAY OF Comparable) : BOOLEAN;
    (*! Postcondition: result = the elements of x are ordered. !*)
    VAR
        i : Index;
    BEGIN
        i := 0;
        WHILE (i < LEN (x) - 1) & (x [i] <= x [i + 1]) DO
            INC (i)
        END;
        RETURN i >= LEN (x) - 1
    END Sorted;
    
```

### Aufgabe 8 Schleifenablauf und -terminierung prüfen, Schleifen umformen

(Punkte: 14 |.....) Verfolgen Sie jeweils einige Durchläufe der beiden Schleifen und begründen Sie zu jeder Schleife, warum sie abbricht bzw. warum nicht!

Schleife 1	Schleife 2	
<pre>k := 13579; WHILE k # 0 DO   k := k - 2;   IF k &lt; 0 THEN     k := -3 * k   END END END</pre>	<pre>FOR n := 1 TO 2 DO   m := n + 1;   n := m - 2 END</pre>	
k	n	m
13579 13577 <i>... ungerade Zahl ...</i> 3 1 -1 3 ...	1 0 1 ...	2 2 ...
<input type="checkbox"/> Terminiert <input checked="" type="checkbox"/> <i>Läuft endlos</i> , weil $k$ mit einer ungeraden Zahl initialisiert wird und durch wiederholtes Vermindern um 2 ungerade bleibt, also nie gerade wird, also auch nicht 0. Sobald $k$ durch $k-2$ negativ wird, wird es anschließend durch $-3*k$ wieder positiv, bleibt aber ungerade. Die Abbruchbedingung $k = 0$ wird also nie erfüllt.	<input type="checkbox"/> Terminiert <input checked="" type="checkbox"/> <i>Läuft endlos</i> , weil: Aus $m = n + 1$ folgt $m - 2 = n + 1 - 2 = n - 1$ . Also wird die Laufvariable $n$ bei jeder Iteration explizit um 1 erniedrigt und dann implizit um 1 erhöht. Also behält $n$ seinen Anfangswert 1 und wird nie größer 2. Die Abbruchbedingung $n > 2$ wird also nie erfüllt.	

*Bemerkung: Eine „normale“ Zählschleife terminiert, aber in Schleife 2 ist diese Programmierregel verletzt: Die Zählvariable darf im Schleifenrumpf nur gelesen werden. Explizite Zuweisungen an die Zählvariable (zer-)stören die Logik der Zählschleife und können zu Endlosschleifen führen. Ich nenne eine Zählschleife missbraucht, wenn in ihrem Schleifenrumpf der Wert der Zählvariable explizit verändert wird. Manche Programmiersprachen verbieten den Missbrauch von Zählschleifen; CP leider nicht.*

Formulieren Sie Schleife 2 (FOR ...) in eine äquivalente WHILE-Schleife um!

```
aux := 2;
n := 1;
WHILE n <= aux DO
  m := n + 1;
  n := m - 2;
  INC (n)
END
```

Zeigen Sie in vier Schritten durch Ersetzen von Programmstücken durch äquivalente Stücke, dass folgende Schleife *nicht* terminiert!

```
mustGoOn := TRUE;
WHILE mustGoOn DO
  mustGoOn := mustGoOn OR FALSE
END
```

1. Schritt: Disjunktion mit *FALSE* weglassen:

```
mustGoOn := TRUE;
WHILE mustGoOn DO
  mustGoOn := mustGoOn
END
```

2. Schritt: *mustGoOn* behält seinen Wert, also Zuweisung weglassen:

```
mustGoOn := TRUE;
WHILE mustGoOn DO
  END
```

3. Schritt: *mustGoOn* behält nach Initialisierung seinen Wert, also sein Auftreten durch den Wert ersetzen:

```
WHILE TRUE DO
  END
```

4. Schritt: Die Fortsetzungsbedingung ist immer *TRUE*, also die Abbruchbedingung immer *FALSE*, also terminiert die Schleife nicht.

## Aufgabe 9 Programmablauf verfolgen

(Punkte: 19 |.....) Untersuchen Sie Programm 9.1 wie weiter unten beschrieben!

### Programm 9.1

CP: Unbekannter Algorithmus

```

CONST
  number = 8;

VAR
  row      : ARRAY number OF INTEGER;
  index1, index2 : INTEGER;

BEGIN
  ... (* Initialisierung der Variablen row. *) ...
  FOR index1 := 2 TO LEN (row) - 1 DO
    row [0] := row [index1];
    index2 := index1 - 1;
    WHILE row [0] < row [index2] DO
      row [index2 + 1] := row [index2];
      index2 := index2 - 1
    END;
    row [index2 + 1] := row [0]
  END
END ...
    
```

In der folgenden Tabelle ist jeder Variable eine Spalte zugeordnet. In den Zeilen (außer der Kopfzeile) werden die Werte, die die Variablen besitzen, wenn die Programmausführung die Stelle (\*!\*) erreicht hat, aufgezeichnet.

Der Programmablauf befindet sich an der Stelle (\*!\*). Verfolgen Sie ihn weiter und vervollständigen Sie die Tabelle, indem Sie jeweils die Zustände der Variablen bis zum Erreichen der Stelle (\*!\*) in eine neue Zeile eintragen! (Tragen Sie auch Zwischenwerte der Variable index2 ein und streichen Sie sie ggf. durch.)

row								index1	index2
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]		
56	79	34	12	17	8	43	6		
34	34	79	12	17	8	43	6	2	4 0
12	12	34	79	17	8	43	6	3	<del>2</del> 4 0
17	12	17	34	79	8	43	6	4	<del>3</del> <del>2</del> 1
8	8	12	17	34	79	43	6	5	<del>4</del> <del>3</del> <del>2</del> 1 0
43	8	12	17	34	43	79	6	6	<del>5</del> 4
6	6	8	12	17	34	43	79	7	<del>6</del> <del>5</del> <del>4</del> <del>3</del> <del>2</del> 1 0
								8	