Heike und Manfred Fillinger

Ein Tiny-Pascal-Compiler

Teil 1

Pascal-Compiler benötigen im Normalfall einen Speicherausbau von mindestens 32 KByte sowie ein Floppy-Disk-Laufwerk. Der hier vorgestellte Tiny-Pascal-Compiler wurde für den TRS-80 Level-II (16 KByte) entworfen. Er ist ebenso für das Video-Genie 3003 gebrauchsfähig. Ein Pascal-Programm wird in Basic übersetzt. Dabei stellt der Compiler die Datentypen, Funktionen und Prozeduren des Level-II-Basic von Tandy zur Verfügung.

In diesem Beitrag soll das Leistungsvermögen des Tiny-Pascal-Compiler aufgezeigt werden. Die Syntax der Programmiersprache Pascal wird daher nur im Rahmen dieser Forderung beschrieben. Für darüber hinausgehende Information sei auf die Fachliteratur verwiesen. Zunächst die Elemente von Tiny-Pascal.

Bezeichner

Alle Größen, mit denen in einem Programm gearbeitet wird, erhalten einen Namen. Diese Größen sind Variablen, Konstanten und Prozeduren. Die Zusammensetzung der Bezeichner unterliegt bestimmten Regeln. Für die Darstellung dieser Regeln haben sich die nachfolgenden Syntax-Diagramme bestens bewährt. Syntax-Diagramme werden in Pfeilrichtung gelesen. An den Verzweigungspunkten wird geprüft, ob die untersuchte Einheit (in diesem Falle der Bezeichner) den gezeigten Richtungen und Eigenschaften entspricht. Jeder Bezeichner beginnt mit einem Buchstaben, welchem dann in beliebiger Reihenfolge Buchstaben und Ziffern angehängt werden können (*Bild 1*).
Zulässige Bezeichner sind zum Beispiel:
A
A7
TAG
BLINDWIDERSTAND3
XX3YY
Unzulässige Bezeichner sind zum Beispiel:
Z-AUS
7A
BLIND WIDERSTAND

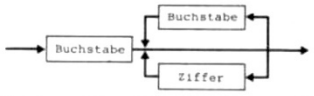


Bild 1. Pascal-Bezeichner bestehen aus Buchstaben oder Buchstaben gefolgt von Buchstaben oder Ziffern

Bild 2. Der formale Aufbau eines Pascal-Programmes

Der Programmaufbau

Ein Pascal-Programm besteht aus zwei Teilen, einem Programm-Kopf und einem Block (*Bild 2*).
Im Programm-Kopf folgt nach dem Wortsymbol PROGRAM der Programmname. Der Tiny-Pascal-Compiler übersetzt diesen Programm-Kopf in eine Basic-REM-Zeile.
Im Block erscheinen nacheinander die Deklarationsteile für Konstanten, Variablen und Prozeduren. Daran schließt der Anweisungsteil, eingeschlossen in die Schlüsselwörter „BEGIN" und „END", an (*Bild 3*).

Die Vereinbarung von Konstanten

Der Deklarationsteil (Vereinbarungsteil) beginnt mit der Definition der konstanten Größen, wie z. B. der Zahl PI! Als Konstante können ganze Zahlen, reelle Zahlen und Zeichenketten deklariert werden (*Bild 4*).
Beispiele für die Konstanten-Deklaration:

```
CONST
PI = 3.1415;
KORREKTUR =13;
FARBE = 'ROT';
```

Die Vereinbarung von Variablen

Eine Variablen-Deklaration wird durch das Wortsymbol VAR eingeleitet. Anschließend erhalten alle Variablen einen Namen, dem nach einem Doppelpunkt

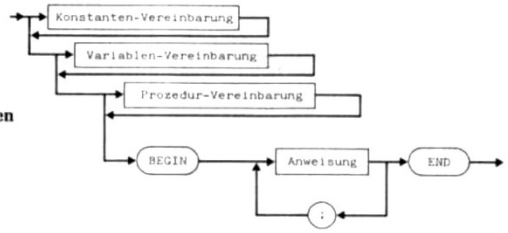


Bild 3. Die Struktur eines Blockes. Zu allen „unbekannten" Elementen des Syntax-Diagrammes gibt es wiederum ein Syntax-Diagramm

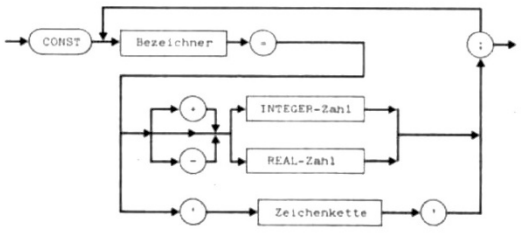


Bild 4. Die Vereinbarung einer Konstanten regelt dieses Diagramm

eine Typenangabe folgen muß (*Bild 5*).
INTEGER: Für ganze Zahlen von −32768...+32767;
REAL: Gleitkommazahlen mit Vorzeichen und 6 gültigen Ziffern; (Wahlweise, Gleitkommazahlen mit Vorzeichen und 16 gültigen Ziffern) Exponent +/− 36;
STRING: Für Zeichenketten mit einer maximalen Länge von 255 Zeichen.

Beispiel:

```
VAR
RESONANZFREQUENZ  : REAL;
INDUKTIVITÄT      : REAL;
KAPAZITÄT         : REAL;
BASIS             : INTEGER;
EINGABE,FUNKT     : STRING;
NUM,RESERVE       ARRAY (15)
                  OF STRING;
MATRIX:           ARRAY (9,19)
                  OF REAL;
```

Für die Feldvariablen NUM und RESERVE wurde nach dem vorstehenden Beispiel ein eindimensionales Feld mit einer maximalen Tiefe von 16 (0...15) Elementen vereinbart. Für die Feldvariable MATRIX wurde ein zweidimensionales Feld mit einer maximalen Tiefe von 10 mal 20 Elementen vereinbart (*Bild 6*).

Die Prozedur-Vereinbarung

Prozeduren sind Unterprogramme und stellen ein wichtiges Mittel zur Strukturierung von Programmen dar. Durch Verwendung von Prozeduren kann eine Gesamtaufgabe in abgegrenzte Teilaufgaben gegliedert werden. Das hat die folgenden Vorteile:

- Das Programm wird übersichtlicher.
- Ein Programmabschnitt wird nur einmal geschrieben, kann aber mehrfach ausgenutzt werden.
- Es lassen sich bereits vorhandene Prozeduren als Bausteine für neue Programme einsetzen.

Die Deklaration von Prozeduren erfolgt nach den Vereinbarungsschritten für Konstanten und Variablen. Sie wird eingeleitet durch das Schlüsselwort PROCEDURE, gefolgt von einem Bezeichner (Prozedur-Name) und dem zwischen BEGIN und END geklammerten Anweisungsteil (*Bild 7*).
Beispiel:

```
PROCEDURE FRES;
 BEGIN
  RESONANZFREQUENZ:=1/
  (2*PI*SQRT
  (INDUKTIVITAET*KAPAZITÄT))
 END (* FRES *);
```

RESONANZFREQUENZ,PI,INDUKTIVITÄT und KAPAZITÄT müssen global also im Hauptprogramm deklariert sein. Im Gegensatz zu Standard-PASCAL läßt der Tiny-Pascal-Compiler wegen des begrenzten Speicherplatzes die Kommunikation zwischen Hauptprogramm und der Prozedur sowie die Prozedur-Interne-Verarbeitung nur über global deklarierte Variablen und Konstanten zu. Es ist somit weder eine Parameterliste noch ein interner Deklarationsteil zulässig.

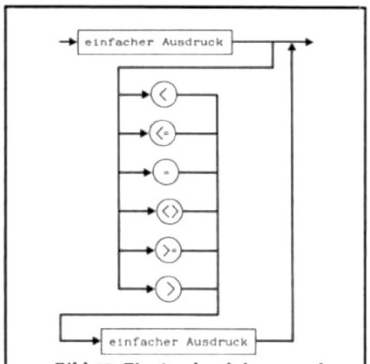


Bild 10. Ein Ausdruck kann auch zusammengesetzt sein

Die Anweisungen von Tiny Pascal

Der Anweisungsteil wird durch die Schlüsselwörter BEGIN und END geklammert. *Bild 8* zeigt das Syntax-Diagramm der Anweisung. *Bild 9* zeigt das Diagramm zum Tiny-Pascal-Konstrukt „Variable".
Beispiel:

```
RADIUS
MATRIX (2, INDEX+KORR)
```

Bild 10 zeigt die Konstruktion von „Ausdruck". Ein Ausdruck kann ein einfacher Ausdruck sein, oder ein einfacher Ausdruck, gefolgt von einem Vergleichsoperator und einem zweiten einfachen Ausdruck.
Beispiel:

```
ECKFREQUENZ/SPERRFREQUENZ > 1
```

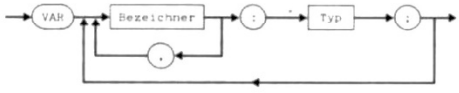


Bild 5. So gibt man dem System bekannt, daß eine Variable gebraucht werden soll

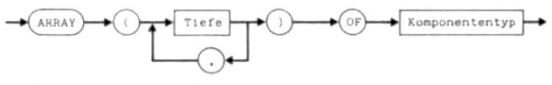


Bild 6. Ein ganzes Feld von Daten eines bestimmten Typs, ein „Array" wird so vereinbart

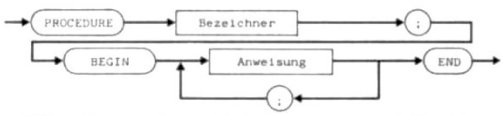


Bild 7. Die Prozeduren, die Unterprogramme, bilden den wichtigsten Punkt in der Pascal-Philosophie

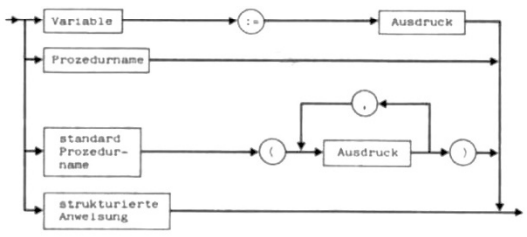


Bild 8. Das ist der Aufbau des Sprachelementes „Anweisung"

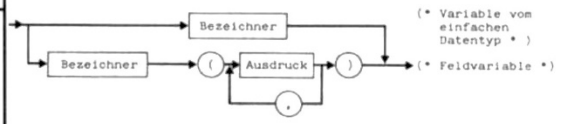


Bild 9. Eine Variable kann auch als Komponente eines Feldes auftreten, deshalb ist ihr Diagramm kompliziert

Ein einfacher Ausdruck besteht aus einem oder mehreren Termen, die durch additive Operatoren verbunden sind (*Bild 11*).

Beispiel:
LÄNGE+13-KORR
X OR Y
Terme wiederum verknüpfen Faktoren mit Hilfe multiplikativer Operatoren (*Bild 12*).

Beispiel:
B AND C
A * B
Faktoren sind dabei folgendermaßen definiert (*Bild 13*).
Die vorzeichenlose Konstante zeigt *Bild 14*.

Es gilt folgende Prioritätshierarchie unter den Operatoren:
1. Potenzen A↑B
2. Negation – Wert
3. *, /
4. +, –
5. <, >, =, <=, >=, <>
6. NOT
7. AND
8. OR

Da eine Deklaration von bool'schen Datentypen in Tiny-Pascal nicht vorgesehen ist, können die Vergleichsoperatoren und die logischen Operatoren nur in Verbindung mit strukturierten Anweisungen sinnvoll eingesetzt werden.
Bei Zeichenketten (STRING) ist der Operator „+" für die Verkettung von Zeichenketten vorgesehen. Die Vergleichsoperatoren dienen dem Vergleich zweier Zeichenketten; diese werden Zeichen für Zeichen von links nach rechts verglichen.
An dieser Stelle sei auf die Syntax beim Kommentar hingewiesen. Ein Kommentar wird zwischen (*...*) eingefaßt. Im Anweisungsteil wird ein Kommentar vom Tiny-Pascal-Compiler überlesen. Im Deklarationsteil wird ein Kommentar in eine Basic-REM-Zeile übersetzt, wenn dieser alleine in einer Pascal-Programm-Zeile steht. Darüber hinaus darf im Deklarationsteil ein Kommentar nur noch hinter dem Semikolon angeordnet werden.

Die Wertzuweisung

In Pascal wird eine Wertzuweisung durch := symbolisiert. Die Variable auf der linken Seite soll den Wert des auf der rechten Seite stehenden Ausdrucks annehmen. Beispiel: *Bild 15* zeigt ein Struktogramm zur Flächenberechnung eines Kreises.

```
PROGRAM KFLAECHE;
(* BERECHNET KREISFLAECHE *)
CONST
   PI = 3.1415;
VAR
   DURCHM, FLAECHE: REAL;
BEGIN
   READ(DURCHM);
   FLAECHE := SQR(DURCHM) *PI/4;
   WRITE('DURCHMESSER: ', DURCHM)
   WRITELN
   WRITE ('FLAECHE:',FLAECHE)
END. (* KFLAECHE *)
```

Output-Beispiel:
? 27
DURCHMESSER: 27
FLAECHE: 572.539

Die Prozeduranweisung

Die Prozeduranweisung dient zum Aufruf einer Folge von Anweisungen. Der Aufruf erfolgt durch die Nennung des Prozedurnamens. Beispiel: *Bild 16* zeigt ein Struktogramm mit Prozeduraufruf, *Bild 17* zeigt die Prozedur als Struktogramm.

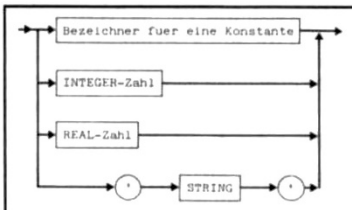


Bild 14. Die vorzeichenlose Konstante

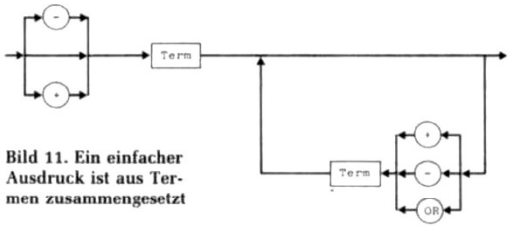


Bild 11. Ein einfacher Ausdruck ist aus Termen zusammengesetzt

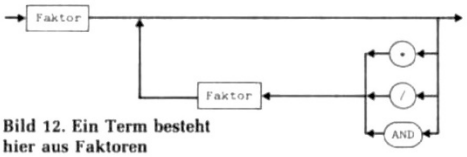


Bild 12. Ein Term besteht hier aus Faktoren

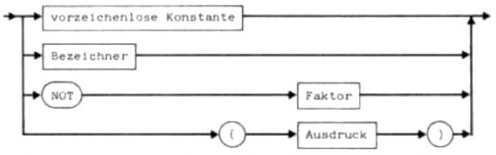


Bild 13. Faktoren wiederum können Ausdrücke enthalten. Ausdrücke können andererseits alle Elemente enthalten, die ab Bild 10 angesprochen sind. Das Diagramm enthält also rekursive Elemente

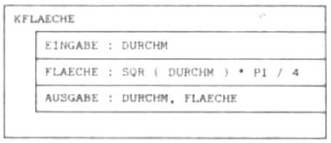


Bild 15. Das Struktogramm eines ersten Programmes

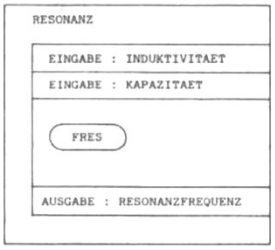


Bild 16. Das Struktogramm eines Programmes, das die Resonanz-Frequenz eines Schwingkreises ermitteln soll. Es benutzt die Prozedur FRES

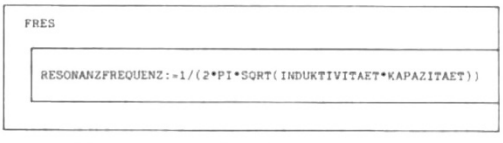


Bild 17. FRES berechnet die Resonanzfrequenz

```
PROGRAM RESONANZ;
(* BERECHNET DIE RESONANZFREQUENZ *)

CONST
 PI = 3.1415;

VAR
 RESONANZFR, IND, KAP: REAL;

PROCEDURE FRES;
 BEGIN
  RESONANZFR:=1/(2*PI*SQRT(IND*KAP));
 END; (*FRES*)

BEGIN (*HAUPTPROGRAMM*)
 WRITE('INDUKTIVITAET IN HENRY:');
 READ(IND);
 WRITELN;
 WRITE('KAPAZITAET IN FARAD:');
 READ(KAP);

 FRES;

 WRITELN('RESONANZ:=',RESONANZFR,'HZ');
END. (* RESONANZ *)
```

Output-Beispiel:
INDUKTIVITAET IN HENRY: 0.136E-3
KAPAZITAET IN FARAD: 120E-12
RESONANZ: =1.24587E+06 HZ

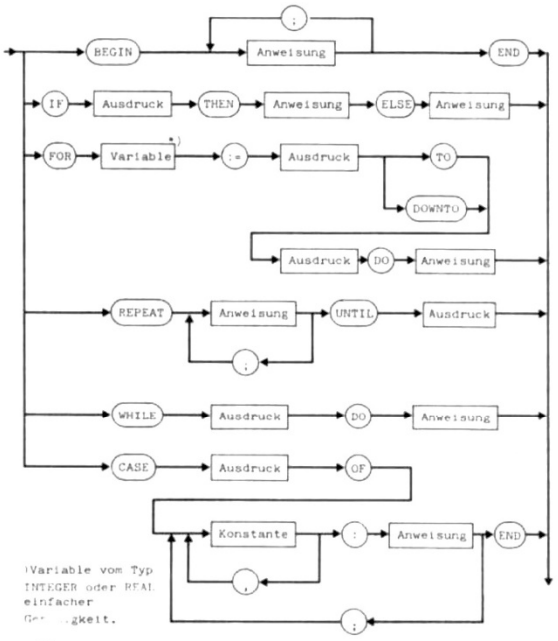


Bild 18. Das Syntax-Diagramm zur strukturierenden Anweisung ist relativ komplex

Die Tiny-Pascal-Standard-Prozeduren

Die Standard-Prozeduren READ und WRITE dienen zur Ein- und Ausgabe von Daten. Die Dateneingabe erfolgt durch die Anweisung READ(....). Mit der Anweisung READ(RADIUS,HOEHE, SPEZ) werden drei Werte eingelesen und den Variablen RADIUS, HOEHE und SPEZ zugeordnet.

Die Ausgabeanweisung WRITE('VOLUMEN:',VOL,' GEWICHT:',GEW) bewirkt die folgende Ausgabe von Texten und Werten: VOLUMEN: XXXXXX GEWICHT: YYYYYY. Die Ausgabe auf Grund der Anweisung WRITE beginnt immer an der aktuellen Schreibposition.

Um die Ausgabe in Zeilen zu gliedern, steht die Anweisung WRITELN(.....) (schreibe Zeile) zur Verfügung.

Zur Formatierung der Ausgabe stehen die von Basic-Level-II her bekannten Anweisungen TAB(n) und PRINT USING zur Verfügung. Die in PASCAL oft übliche Formatierung über :FELDBREITE ist nicht möglich.

Durch die Standard-Prozeduren READ#(.....) und WRITE#(.....) können Daten von einem Kassetten-Recorder gelesen bzw. auf einen Kassetten-Recorder aufgezeichnet (geschrieben) werden.

Weitere Standard-Prozeduren:
Ausgabe-Formatierungs-Prozeduren:

TAB(N)	– bewegt Cursor zur Bildschirmposition n
USING Zeichenkette, Ausdruck	– Ausdruck wird in einem Format ausgegeben, das von Zeichenkette bestimmt wird.

Grafische Prozeduren:

CLS	– Bildschirm löschen
RESET(X,Y)	– Bildpunkt dunkel tasten
SET(X,Y)	– Bildpunkt hell tasten

Sonder-Prozeduren:

POKE Adresse,Wert	– lädt den Wert in die adressierte Speicherzelle
OUT Kanal,Wert	– sendet den Wert zum Ausgabe-Kanal

Näheres zur Wirkung dieser Prozeduren ist aus dem „Handbuch für Basic, Level II" zu ersehen.

Strukturierende Anweisungen

Bild 18 zeigt den Aufbau der strukturierenden Anweisungen.

Folgen hinter den Schlüsselwörtern THEN und ELSE sowie DO und den CASE-Marken mehrere Anweisungen oder eine strukturierende Anweisung – außer der Verbundanweisung – so sind diese mit BEGIN und END zu klammern. Bei Standard-PASCAL ist diese Klammerung nur bei einer Folge von Anweisungen erforderlich.

Die Verbundanweisung

Die Verbundanweisung ist eine Folge von Anweisungen, die eine Einheit bilden soll. Sie wird zwischen BEGIN und END geklammert.

Ein- oder zweiseitige Auswahl

Die Auswahl der Verzweigung hängt von der Auswertung des Ausdruckes ab. Ist das Ergebnis der Auswertung wahr, so wird die Anweisung hinter dem Schlüsselwort THEN ausgeführt. Ist das Ergebnis der Auswertung unwahr, so wird die Anweisung hinter dem Schlüsselwort ELSE ausgeführt. Der ELSE-Fall kann auch fortgelassen werden. Man erhält dann die einseitige Auswahl.

Beispiel: *Bild 19* zeigt das Struktogramm eines Quadratwurzelprogrammes.

```
PROGRAM QUADRATWURZEL;
(* PRUEFT ARGUMENT, BER. WURZEL,*)
(* WENN ARGUMENT POSITIV *)
VAR
 ARG,WURZEL : REAL;
BEGIN
 WRITE('ARGUMENT:');
 READ(ARG);
 IF ARG >= 0
  THEN
   BEGIN
    WURZEL:=SQRT(ARG);
    WRITELN(WURZEL);
   END
  ELSE
   WRITELN('NEGATIVES ARGUMENT')
END. (* QUADRATWURZEL *)
```

Output-Beispiel:

```
 ARGUMENT : 193
 13.8924
 ARGUMENT: - 193
 NEGATIVES ARGUMENT
```

Zählschleife

Die Ausführung einer FOR-Anweisung geschieht nach folgenden Schritten:

a) Auswertung der Ausdrücke vor und hinter dem Schlüsselwort TO bzw. DOWNTO zur Bestimmung des Anfangs- und Endwertes für die Zählvariable. Ist bei TO bzw. DOWNTO der Anfangswert größer bzw. kleiner als der Endwert, so ist die FOR-Anweisung bereits ausgeführt.

b) Es wird der Zählvariablen der Anfangswert zugewiesen.

c) Die auf das Schlüsselwort DO folgende Anweisung wird ausgeführt.

d) Es wird im Falle von TO der Wert der Zählvariablen um eins erhöht, im Falle von DOWNTO um eins vermindert.

e) Es wird geprüft, ob der Wert der Zählvariablen bei TO größer, bei DOWNTO kleiner als der Endwert ist. Ist dies der Fall, ist die FOR-Anweisung vollständig ausgeführt. Andernfalls wird bei Schritt c) fortgefahren.

Beispiel: *Bild 20* zeigt das Struktogramm für ein „Einmaleins"-Programm.

```
PROGRAM EINMALEINS;
(*BERECHNET DAS KLEINE EINMALEINS*)
VAR
 WERT,ZVAR1,ZVAR2 : INTEGER;

BEGIN
 FOR ZVAR1 := 1 TO 10 DO
  BEGIN (*ÄUSSERE SCHLEIFE*)
   FOR ZVAR2 := 1 TO 10 DO
    BEGIN (*INNERE SCHLEIFE*)
    WERT := ZVAR1 * ZVAR2;
    WRITELN(ZVAR1,'*', ZVAR2,':=',WERT)
    END;
   WRITELN
  END
END. (*EINMALEINS*)
```

Output-Beispiel:

```
1 * 1 := 1
1 * 2 := 2
1 * 3 := 3
1 * 4 := 4
1 * 5 := 5
1 * 6 := 6
1 * 7 := 7
1 * 8 := 8
1 * 9 := 9
1 * 10 := 10

2 * 1 := 2
.....
.....
9 * 9  := 81
9 * 10 := 90

10 * 1  := 10
10 * 2  := 20
10 * 3  := 30
10 * 4  := 40
10 * 5  := 50
10 * 6  := 60
10 * 7  := 70
10 * 8  := 80
10 * 9  := 90
10 * 10 := 100
```

Wiederholanweisung mit Endbedingung

Die Ausführung der REPEAT-Anweisung erfolgt nach folgenden Schritten:

a) Ausführung der hinter dem Schlüsselwort REPEAT stehenden Anweisung bzw. Anweisungen.

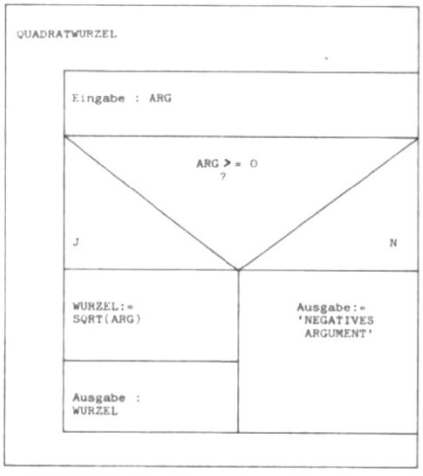


Bild 19. Das Struktogramm zur Berechnung einer Quadratwurzel

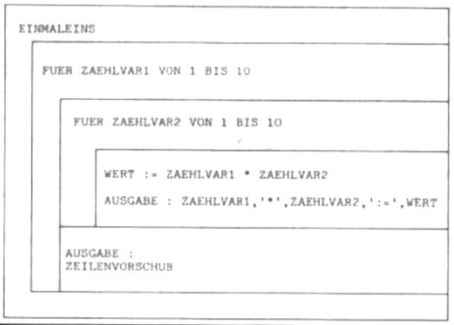


Bild 20. Das Struktogramm zu „Einmaleins" enthält die Zählschleife gleich zweimal und verschachtelt

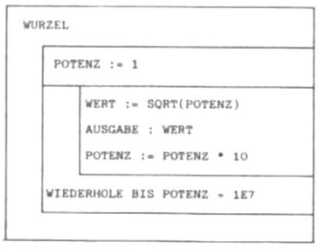


Bild 21. Ein Struktogramm mit Wiederhol-Schleife und Abfrage am Ende

b) Auswertung der Endbedingung (Ausdruck). Ist das Resultat der Auswertung unwahr, wird zu Schritt a) zurückgekehrt. Ist das Resultat wahr, so ist die REPEAT-Anweisung vollständig ausgeführt.

Beispiel: *Bild 21* zeigt ein Struktogramm mit REPEAT

```
PROGRAM WURZEL;
(* QUADRATWURZELN ALLER ZEHNERPOTENZEN VON 1-1 000 000 *)
VAR
 POT,WERT : REAL;
BEGIN
 POT := 1;
 REPEAT
  WERT := SQRT(POT);
  WRITELN('WURZEL AUS', POT,' IST: ',WERT);
  POT := POT * 10;
  WRITELN
 UNTIL POT = 1E7
END (*WURZEL*)
```

Output-Beispiel:
WURZEL AUS 1 IST : 1
WURZEL AUS 10 IST : 3.16228
WURZEL AUS 100 IST : 10
WURZEL AUS 1000 IST : 31.6228
WURZEL AUS 10000 IST : 100
WURZEL AUS 100000 IST : 316.228
WURZEL AUS 1E+06 IST : 1000

Wiederholanweisung mit Eingangsbedingung

Die WHILE-Anweisung wird wie folgt ausgeführt:

a) Auswertung der Eingangsbedingung (Ausdruck). Ist das Resultat der Auswertung unwahr, so ist die WHILE-Anweisung vollständig ausgeführt. Ist das Resultat wahr, so wird mit Schritt b) fortgefahren.

b) Ausführung der Anweisung. Anschließende Rückkehr zu Schritt a).

Beispiel: *Bild 22* zeigt ein Struktogramm mit WHILE-Struktur

Output-Beispiel:
SUMME 1 BIS 100 IST : 5050

Mehrfach Auswahl

Die Ausführung einer CASE-Anweisung läuft in folgenden Schritten ab:

a) Auswertung des Ausdrucks.

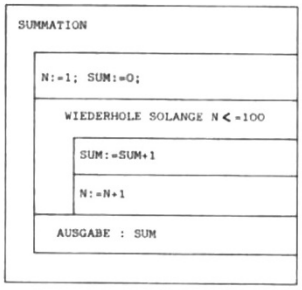
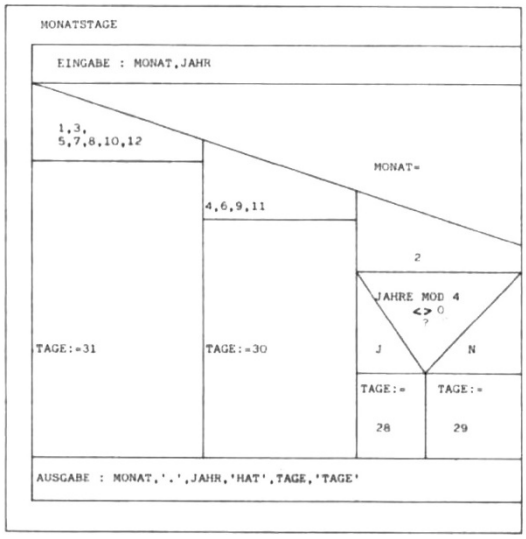


Bild 22. Ein Struktogramm mit Wiederholung und Abfrage zu Beginn

Bild 23. Struktogramm zur Berechnung der Anzahl von Monatstagen. Dabei werden die auftretenden Fälle durch Mehrfach-Abfragen ermittelt




```
PROGRAM SUMMATION;
(* SUMMATION GANZER ZAHLEN VON 1 BIS EINSCHLIESSLICH 100 *)

VAR
 N, SUM : INTEGER;
BEGIN
  N := 1; SUM := 0;
  WHILE N <= 100 DO
   BEGIN
    SUM := SUM + N;
    N := N + 1;
   END; (*WHILE*)
   WRITELN ('SUMME 1 BIS 100 IST: ', SUM)
END. (*SUMMATION*)
```

b) Stimmt der Wert des Ausdrucks mit einer der CASE-Marken (Konstanten) überein, so wird die zugehörige Anweisung ausgeführt. Ist das geschehen, so ist auch die CASE-Anweisung beendet. Ist keine passende Marke aufgelistet, so ist die CASE-Anweisung unbestimmt.

Beispiel: *Bild 23* zeigt das Struktogramm zu „Monatstage", das CASE enthält

```
PROGRAM MONATSTAGE;
(*BERECHNET DIE ANZAHL DER MONATSTAGE*)
(*GÜLTIG ZWISCHEN 1901 UND 2099*)
VAR
      JAHR,MONAT,TAGE : INTEGER;
BEGIN (*HAUPTPROGRAMM*)
      READ(MONAT,JAHR);
      CASE MONAT OF
      1,3,5,7,8,10,12 : TAGE := 31;
            4,6,9,11 : TAGE := 30;
                   2 : BEGIN
                       IF (JAHR-TRUNC(JAHR/4)*4) <> 0
                          THEN TAGE := 28
                          ELSE  TAGE := 29
                       END (*IF*)
      END; (*CASE*)
      WRITELN(MONAT,'.',JAHR,' HAT',TAGE,'TAGE')
END.
```

Output-Beispiel:

```
? 2
?? 1982
 2 . 1982 HAT 28 TAGE

? 9
?? 1980
 9 . 1980 HAT 30 TAGE

? 2
?? 1956
 2 . 1956 HAT 29 TAGE

? 3
?? 2050
 3 . 2050 HAT 31 TAGE
```

Die Standard-Funktionen von Tiny Pascal

Dieser Tiny-Pascal-Compiler bearbeitet folgende Standardfunktionen:

		äquivalente Basic-Funktion
SQR(X)	Quadrat	X↑2
SQRT(X)	Wurzel	SQR(X)
TAN(X)	Tangens	TAN(X)
ARCTAN(X)	Arcustangens	ATN(X)
SIN(X)	Sinus	SIN(X)
COS(X)	Cosinus	COS(X)
LN(X)	Natürlicher Logarithmus	LOG(X)
EXP(X)	Exponential	EXP(X)
ABS(X)	Absolutbetrag	ABS(X)
TRUNC(X)	Abschneiden	FIX(X)
RND(X)	Zufallszahl	RND(X)
LENGTH(ZK)	Länge einer ZK	LEN(ZK)
SUBSTR(ZK,P,n)	Unterzeichenkette	MID$(ZK,P,n)
ORD(ZK)	ASCII-Code eines Zeichens	ASC(ZK)
CHR(X)	Zeichen entsprechend dem ASCII-Code	CHR$(X)
POINT(X,Y)	Grafischen Punkt abfragen	POINT(X,Y)
PEEK(Adresse)	Inhalt der Adresse	PEEK(Adresse)
INP(Kanal)	Wert am Eingabekanal	INP(Kanal)
STR(X)	ZK-Darstellung des Argumentwertes	STR$(X)
POS(n,Zeichen)	Folge von n mal Zeichen	STRING$(n,Zeichen)

„ZK" bedeutet Zeichenkette.

Näheres zur Wirkung der aufgeführten Funktionen ist aus dem „Handbuch für Basic Level II" ersichtlich.
Der Compiler selbst folgt im nächsten Heft.

Neue Laufwerke

Die Firma Epson hat ihre Produktpalette mit einer wohlabgestimmten Linie von Floppy-Laufwerken für OEM-Zwecke erweitert.
Um mit den ganz kleinen Laufwerken zu beginnen: SMD-100 heißt eine Serie von 3,5-Zoll-Laufwerken, die es wahlweise

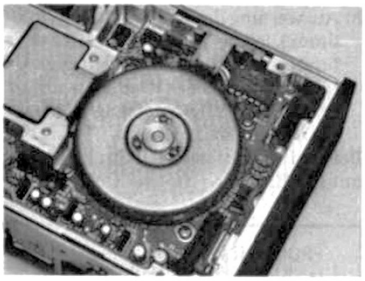


Bild 1. Der Motor der Mikrofloppy

für Batteriebetrieb oder normale Floppy-Spannungsversorgung gibt. *Bild 1* zeigt im Ausschnitt den „Direct-Drive-Motor", der für lange Lebensdauer und große Laufruhe sorgt. *Bild 2* zeigt den Kopfschlitten, der mit Stahlseilbetrieb versehen ist. In den Laufwerken werden spezielle CMOS-ICs verwendet, die den Stromverbrauch und die Wärmezufuhr drastisch senken. Die batteriebetriebenen Typen verbrauchen im „Standby" nur 0,05 W. Während eines Lese-/Schreib-Vorgangs etwa 3,2 W. Die Daten: Unformatiert können je nach Laufwerktyp 250, 500 und 1000 KByte Speicherkapazität geboten werden. Es gibt Ausführungen für Single-Side-Betrieb und Ausführungen für Double-Side-Betrieb. Die Spurenanzahlen sind 40 bzw. 80. Angesteuert werden die Laufwerke über eine Schnittstelle, die der Industrie-Standard-Schnittstelle für 5,25-Zoll-Laufwerke entspricht. Die Zugriffszeit (Track zu Track) beträgt 6 bzw. 3 ms. Das Aufzeichnungsverfahren kann sowohl FM als auch MFM sein. Nach Erscheinen dieser Laufwerke wird es wohl nur noch eine Frage der Zeit sein, wann es wirklich tragbare Computer mit Massen-Speicher für die Aktentasche geben wird. Gleichzeitig bringt Epson ebenso technologisch interessante 5,25-Zoll-Geräte. *Ro.*

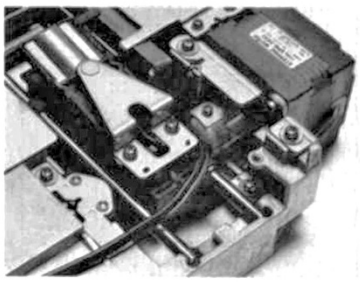


Bild 2. Das ist der Kopfschlitten der Floppy von Epson