K.-H. BACHMANN

ALGOL-PROGRAMMIERUNG

MIT VARIANTE FÜR ROBOTRON 300

VEB DEUTSCHER VERLAG DER WISSENSCHAFTEN

K.-H. BACHMANN

ALGOL-PROGRAMMIERUNG

MIT VARIANTE FÜR ROBOTRON 300

ALGOL-PROGRAMMIERUNG

MIT VARIANTE FÜR ROBOTRON 300

VON

K.-H. BACHMANN

Mit 8 Abbildungen

Dritte, berichtigte und erweiterte Auflage

VEB DEUTSCHER VERLAG DER WISSENSCHAFTEN

BERLIN 1969

ES 19 B

PRINTED IN THE GERMAN DEMOCRATIC REPUBLIC
LIZENZ-NR. 206 · 435/81/69
GESAMTHERSTELLUNG: VEB LEIPZIGER DRUCKHAUS, LEIPZIG (III/18/203)

INHALTSVERZEICHNIS

I. ALGOL 60

§ 1. Programmbeschreibung in ALGOL

Zur Programmierung eines Problems für einen bestimmten Rechenautomaten ist im allgemeinen eine beträchtliche Arbeit zu leisten. Das Resultat ist ein Maschinenprogramm, das an Hand von vorbereiteten Beispielen am Automaten noch zu prüfen ist. Derartige Programme sind nur für einen Automatentyp (Automaten mit gleichem Befehlsschlüssel) geeignet; soll das gleiche Problem mit einem anderen Automaten behandelt werden, so muß es nochmals programmiert werden. Diese Sachlage führte zu dem Wunsch, eine einheitliche, für beliebige Automaten verwendbare Schreibweise für Programme zu besitzen. Aus dieser Schreibweise werden die Programme in der Regel vom betreffenden Automaten selbst mittels eines Übersetzungsprogramms in ein Maschinenprogramm umgeformt. Dieser Vorgang wird auch als automatisches Programmieren bezeichnet.

Es existiert bisher keine allgemein als verbindlich anerkannte Formulierungssprache zur Beschreibung von Programmen, die man in dieser mehr maschinenunabhängigen Form oft auch als Algorithmen bezeichnet. Es wurde eine große Zahl von Sprachen für verschiedenartige Zwecke konstruiert, man bezeichnet sie auch als problemorientierte Sprachen. Beispiele sind die Sprachen ALGOL (algorithmic language) für Zwecke der numerischen Mathematik und COBOL (common business oriented language) für kommerzielle Zwecke. Die Sprache ALGOL, die hier näher betrachtet werden soll, wurde ab 1958 in einer internationalen Gemeinschaftsarbeit entwickelt und 1960 in einer bis auf später vorgenommene kleinere Korrekturen endgültigen Fassung ALGOL 60 veröffentlicht. Der korrigierte Bericht über ALGOL 60 ist [44]. Bei der Entwicklung von ALGOL wurden die hauptsächlich für die automatische Übersetzung in IBM-Rechenanlagen entworfene Sprache FORTRAN und die algorithmische Schreibweise von RUTISHAUSER berücksichtigt [24], [49], [50].

An eine problemorientierte Sprache sind eine Anzahl von Forderungen zu stellen, damit sie dem vorgesehenen Zweck dienen kann. Solche Forderungen sind etwa:

1. Die Sprache muß eindeutig sein, d. h., einer Beschreibung eines Programms in der betreffenden Sprache muß genau eine Vorschrift zur Verarbeitung gegebener Daten zugeordnet sein. Dabei ist es jedoch zulässig und auch nicht zu vermeiden, daß diese Vorschrift an verschiedenen Automaten verschieden realisiert wird.

2. Die Sprache muß eine vollständige Beschreibung eines Programms erlauben, d. h., alle zur Lösung eines Problems notwendigen Angaben müssen in der Sprache darstellbar sein.
3. Die Sprache soll bezüglich einer Problemklasse vollständig sein. Diese Forderung besagt, daß eine unabhängig von der betreffenden Sprache formulierte Problemklasse durch Programme lösbar sein muß, die in der Sprache formuliert sind.
4. Die Sprache soll, soweit es die bisher angegebenen Forderungen zulassen, die in der betreffenden Problemklasse übliche Symbolik weitgehend beibehalten.
5. Texte der Sprache sollen einfach herzustellen und zu lesen sein.

Ein Programm für einen Rechenautomaten ist eine Folge von Befehlen, die in einer bestimmten Reihenfolge abgearbeitet werden. Entsprechend besteht ein ALGOL-Programm im wesentlichen aus Sprachelementen, die als Anweisungen bezeichnet werden und ebenso in einer bestimmten Reihenfolge abgearbeitet werden. In Maschinenprogrammen sind die einzelnen Befehle meist in adressierbaren Speicherzellen untergebracht, damit ein Sprung zu einem solchen Befehl erfolgen kann; dementsprechend können in ALGOL Anweisungen durch Vorsetzen einer Marke gekennzeichnet werden. Man bezeichnet solche Anweisungen als markierte Anweisungen. Die erwähnte Reihenfolge, in der die einzelnen Anweisungen eines ALGOL-Textes abgearbeitet werden, ist die, in der sie notiert sind; es kann jedoch durch eine Sprunganweisung veranlaßt werden, daß diese Reihenfolge durchbrochen und bei einer markierten Anweisung, die Ziel der Sprunganweisung ist, fortgefahren wird. Zur Niederschrift eines Textes einer Sprache werden Zeichen benutzt, die für die Sprache ALGOL 60 als ALGOL-Zeichen oder Grundsymbole bezeichnet werden sollen. Grundsymbole sind die kleinen und großen lateinischen Buchstaben, die Ziffern 0–9, die Zeichen $+$, $-$, $\times$, / zur Bezeichnung der vier Grundrechenarten, die Klammern (und) und eine Anzahl weiterer Symbole, von denen hier einige mit einer deutschen Aussprache (vgl. [58]) und einer kurzen (unvollständigen) Erläuterung ihrer Anwendung angegeben seien. Weitere Grundsymbole werden von Fall zu Fall eingeführt.

Grundsymbol	Aussprache	Anwendung
	Semikolon	Trennzeichen zwischen Anweisungen
	Doppelpunkt	Trennzeichen zwischen Marke und Anweisung in einer markierten Anweisung
	Komma	Trennzeichen zwischen Elementen einer Liste

:=	ergibt sich aus	Wertzuweisungszeichen zur Zuweisung eines Wertes an eine Variable
begin	Beginn	Anfangszeichen einer aus mehreren Anweisungen zusammengesetzten Anweisung
end	Ende	Endzeichen einer zusammengesetzten Anweisung
go to	Sprung nach	Sprungzeichen in einer Sprunganweisung
if	wenn	Zeichen vor einer Bedingung in einer bedingten Anweisung
then	dann	Zeichen nach einer Bedingung in einer bedingten Anweisung
else	sonst	Zeichen zur Trennung von zwei auf Grund einer Bedingung auszuwählenden Anweisungen, die Teil einer bedingten Anweisung sind
[	eckige Klammer auf	Trennzeichen zwischen dem Namen für eine indizierte Variable und der Liste der zugeordneten Indizes
]	eckige Klammer zu	Schlußzeichen hinter einer Liste von Indizes

Hierzu sei bemerkt, daß bedingte Anweisungen solche Anweisungen sind, deren Ausführung von der Erfüllung einer hinter dem Zeichen **if** stehenden Bedingung abhängt. Zur Formulierung von Bedingungen können u. a. *Vergleichsaussagen* (*Relationen*) benutzt werden, in denen die Relationszeichen $<$, $\leqq$, $=$, $\neq$, $\geqq$ und $>$ als Grundsymbole Verwendung finden. Einige der ALGOL-Zeichen sind durch halbfett gedruckte englische Worte dargestellt, meist ist die englische Aussprache dieser Worte statt der angegebenen deutschen Aussprache beim Lesen von ALGOL-Programmen üblich. In Manuskripten werden die hier durch halbfetten Druck hervorgehobenen Grundsymbole durch Aufschreiben der gleichen Worte und Unterstreichen gekennzeichnet. Es sei noch darauf hingewiesen, daß die meisten ALGOL-Zeichen nicht nur die angegebene Verwendung finden, sondern auch in anderen später zu besprechenden Zusammenhängen in einem ALGOL-Programm vorkommen können. Die folgende Einführung in ALGOL 60 besteht aus verschiedenen Stufen, für einfache Programme genügt die Kenntnis der Stufen 1 und 3.

1. Stufe: Elementare Anweisungen

Eine numerische Rechnung in einem programmgesteuerten Rechenautomaten läuft im wesentlichen so ab, daß in Speicherzellen, die Variablen zugeordnet sind, gewisse Verschlüsselungen für Zahlen gespeichert werden. Diese Zahlen werden durch arithmetische Verknüpfungen ebenso verschlüsselter Zahlen gewonnen. Sie werden als *Werte* angesehen, die die Variablen annehmen können; der Wertebereich ist endlich. Der Vorgang der Zuordnung eines Wertes zu einer Variablen wird als *Wertzuweisung* bezeichnet, die Vorschrift zur Bildung eines Zahlenwertes ist ein *arithmetischer Ausdruck*. Der Wertebereich für Zahlenwerte in ALGOL ist theoretisch die Menge der reellen Zahlen, bei praktischen Anwendungen jedoch stets eine endliche Teilmenge. Eine Wertzuweisung wird in ihrer einfachsten Form in ALGOL geschrieben als

Variable := *arithmetischer Ausdruck*.

Zur Darstellung einer Variablen wird in ALGOL eine Folge von Buchstaben oder Ziffern benutzt, die mit einem Buchstaben beginnen muß, eine solche Folge heißt Name der Variablen. Es treten einfache Variable auf, deren Darstellung allein aus ihrem Namen besteht, und indizierte Variable, zu deren Darstellung auf den Namen eine in eckige Klammern eingeschlossene Liste von Indizes folgt. Darstellungen von Variablen sind also z. B. *x*, *y*, *a*1, *D*, *alpha*, *a*[*i*,*k*], *a*[1,2] usw. Dazu ist zu bemerken, daß als Index ein beliebiger arithmetischer Ausdruck auftreten kann und daß zur Bildung arithmetischer Ausdrücke neben Variablen auch Zahlen und Funktionen direkt verwendet werden können.

Hier soll zunächst nicht auf den formalen Aufbau der Sprache eingegangen, sondern nur an Beispielen erläutert werden, wie ALGOL-Programme zu schreiben sind. Dabei wird unter Berücksichtigung der oben formulierten Forderung Nr. 4 die in der Arithmetik gebräuchliche Schreibweise weitgehend übernommen und nur von Fall zu Fall auf Abweichungen hingewiesen.

Beispiele für Wertzuweisungen sind:

$y := 1$	Der einfachen Variablen y wird der Wert 1 zugeordnet.
$z := sin(x)$	Der einfachen Variablen z wird der Wert der Funktion sin x zugeordnet, der sich aus dem der unabhängigen Variablen x zugeordneten Wert ergibt.
$f := max(a,b)$	Falls in einer später zu erläuternden Art eine Funktion *max* von zwei Variablen definiert ist, wird der einfachen Variablen f der sich aus den aktuellen Werten von a und b ergebende Wert der Funktion *max* zugeordnet. Dabei steht

die kürzere Bezeichnung „aktueller Wert" für den Begriff „einer Variablen zur betreffenden Zeit zugeordneter Wert".

Bemerkung. Funktionen sind stets so zu schreiben, daß auftretende Argumente in runde Klammern eingeschlossen sind. Zwei oder mehr Argumente werden durch Komma getrennt, die Argumente können auch arithmetische Ausdrücke sein.

pi := 3.141 59 Der einfachen Variablen *pi* wird der Wert 3.141 59 zugeordnet.

Bemerkung. Der Dezimalpunkt wird in ALGOL zur Trennung von ganzem und gebrochenem Teil einer Dezimalzahl benutzt.

$r := sqrt(x \uparrow 2 + y \uparrow 2)$

Der einfachen Variablen r wird der aktuelle Wert von $x^2 + y^2$ zugeordnet.

Bemerkung. Die Potenzierung wird in ALGOL statt durch Höhersetzen des Exponenten durch Einschieben eines ALGOL-Zeichens ↑ dargestellt. Die Funktion mit dem Namen *sqrt* (von square root) bezeichnet die positive Quadratwurzel aus dem aktuellen Wert des Arguments.

$a[i,k] := a[i,k] + q \times a[j,k]$

Der indizierten Variablen $a[i,k]$ wird der Wert des rechtsstehenden Ausdruckes zugeordnet, wobei zur Berechnung dieses Ausdrucks der vorher $a[i,k]$ zugeordnete Wert zu verwenden ist.

Bemerkung. Indizierte Variable bezeichnen eine einzelne Variable aus einem Feld von Variablen, z. B. Komponenten von Matrizen oder Vektoren. Als Indizes können auch arithmetische Ausdrücke auftreten, die ganzzahlig zu runden sind.

$i := i + 1$ Der aktuelle Wert der einfachen Variablen i ist um 1 zu erhöhen.

$x := a \uparrow (2 \times b \uparrow 2 + d/e \times (f + h))/b$

Der auf der rechten Seite stehende Ausdruck wird in der üblichen arithmetischen Notation als

$$\frac{a^{2b^2+\frac{d}{e}(f+h)}}{b}$$

geschrieben.

Bemerkung. Bei der Berechnung arithmetischer Ausdrücke sind Prioritätsregeln zu beachten:

1. Eingeklammerte Teilausdrücke werden bei der Berechnung als Variable (ohne Namen) angesehen, ihr Wert ist der Wert des Teilausdrucks.
2. $\uparrow$ wird vor $\times$ und /, $\times$ und / werden vor $+$ und $-$ bearbeitet, sonst werden die Operationszeichen von links nach rechts abgearbeitet. Die Werte der Operanden werden ebenfalls in der notierten Reihenfolge berechnet. (Die Operanden können Funktionen sein, deren Werte erst zu berechnen sind, bei Variablen und Zahlen erübrigt sich die Berechnung).

Operationszeichen dürfen in ALGOL nicht weggelassen werden.

Die Berechnung des letzten Ausdrucks sei an einem Zahlenbeispiel verfolgt. Vor der Berechnung liege folgende Wertzuordnung vor:

$a = 2, \quad b = 2, \quad d = 3, \quad e = 2, \quad f = -5, \quad h = 3.$

Zuerst ist der umfassendere Klammerausdruck zu berechnen. In ihm wird zunächst $b \uparrow 2$ berechnet, der Wert dieses Teilausdrucks ist 4. Danach wird $2 \times b \uparrow 2 = 8$ berechnet, dann $d/e = 1.5$, danach $f + h = -2$, daraus $d/e \times (f + h) = -3$, erst dann folgt die Addition, die dem Klammerausdruck den Wert 5 zuordnet. Die Potenzierung $a \uparrow 5$ erfolgt wieder vor der Division durch b, der Wert des ganzen Ausdrucks wird somit 16.

$x := a \uparrow (-2)$

Bemerkung. Auf ein Operationszeichen muß stets eine Variable, Zahl, Funktion oder ein eingeklammerter Teilausdruck folgen, $a \uparrow -2$ ist nicht zulässig.

Einer Wertzuweisung entspricht in einem zugeordneten Maschinenprogramm im allgemeinen eine Folge von Befehlen. Die Wertzuweisung in ALGOL wird als Einheit angesehen, d. h., es ist nicht möglich, irgendwo innerhalb der Wertzuweisung mit ihrer Abarbeitung zu beginnen. Entsprechendes gilt auch für andere ALGOL-Anweisungen. Eine Ausnahme bildet die zusammengesetzte Anweisung, die in mancher Beziehung ebenfalls als Einheit angesehen wird, in deren Inneres jedoch gesprungen werden kann. Eine *zusammengesetzte Anweisung* besteht aus einer Folge von Anweisungen, z. B. Wertzuweisungen, die jeweils durch ein Semikolon getrennt sind. Vor der ersten Anweisung der Folge steht das ALGOL-

Zeichen **begin**, hinter der letzten Anweisung das ALGOL-Zeichen **end**. Ein Beispiel für eine zusammengesetzte Anweisung ist

begin $y := (x + 0.16)/(0.46 \times x + 0.70);\quad y := 0.5 \times (y + x/y);$
$y := 0.5 \times (y + x/y)$ **end**.

Diese Anweisung liefert eine gute Näherung y für $\sqrt{x}$, sofern $\frac{1}{4} \leqq x \leqq 1$ gilt.

An einem Zahlenbeispiel sei die Abarbeitung erläutert. Hat x vor Beginn der Anweisung den Wert 0.36, so ergeben sich in den drei Teilanweisungen folgende Werte für y:

$y := 0.600739371534;\quad y := 0.600000454986;\quad y := 0.600000000000.$

In der zweiten Teilanweisung wird zur Berechnung des rechtsstehenden Ausdrucks der erste Wert für y verwendet, in der dritten der zweite Wert für y.

Soll ein Sprung zu einer Anweisung führen, so ist die Anweisung zu markieren. Die einfachste Form einer markierten Anweisung ist

Marke: *Anweisung*.

Als Marken finden Namen oder (vorzeichenlose) ganze Zahlen Verwendung. *Sprunganweisungen* haben die Form

go to *Marke*.

Auch diese Form ist nur die einfachste Art von Sprunganweisungen, sie genügt aber fast immer; im allgemeinen kann die Marke durch einen sogenannten Zielausdruck ersetzt werden. Die Ausführung einer Sprunganweisung geschieht einfach so, daß die mit der betreffenden Marke markierte Anweisung als nächste Anweisung ausgeführt wird. Das setzt voraus, daß es genau eine solche Anweisung im betreffenden ALGOL-Text gibt. Sprunganweisungen treten häufig als Teil einer komplizierteren Anweisung – der *bedingten Anweisung* – auf, von der es zwei Arten gibt:

1. Art: **if** R **then** A_1
2. Art: **if** R **then** A_1 **else** A_2.

In dieser Notierung steht R für eine Relation (allgemeiner für einen später zu behandelnden logischen oder Booleschen Ausdruck), A_1 und A_2 sind Anweisungen, die auch markiert sein können. A_1 darf allerdings nicht wieder eine bedingte Anweisung sein. Es tritt also in den bedingten Anweisungen ein ähnlicher Fall wie bei den zusammengesetzten Anweisungen auf, daß eine Anweisung andere Anweisungen enthält, zu denen gesprungen werden darf. Das sollte, da hierbei Schwierigkeiten im Verständnis des Programms auftreten können, jedoch möglichst vermieden werden. Die Ausführung einer bedingten Anweisung geschieht so, daß geprüft wird, ob die Relation R erfüllt ist. Ist das der Fall, so wird A_1

ausgeführt, damit ist die Anweisung abgearbeitet. Andernfalls wird bei der 1. Art zur folgenden Anweisung übergegangen, bei der 2. Art A_2 ausgeführt, womit die Anweisung in diesem Fall auch abgearbeitet ist.

Als Beispiel für die Verwendung dieser Anweisungen sei eine zusammengesetzte Anweisung gebracht, die die Lösung einer quadratischen Gleichung $x^2 + p \cdot x + q = 0$ beschreibt, p und q seien reelle Zahlen, die Lösungen $x_1 = u_1 + i \cdot v_1$, $x_2 = u_2 + i \cdot v_2$ können reell oder komplex sein. Im folgenden wird $u1$ statt u_1, $v1$ statt v_1 usw. geschrieben, damit in ALGOL zulässige Namen für die Variablen auftreten:

```
begin D := 0.25 × p ↑ 2 − q;
      if D < 0 then go to A;
      u1 := −0.5 × p + sqrt(D);  u2 := −0.5 × p − sqrt(D);
      v1 := 0;  v2 := 0;  go to B;
A:    u1 := −0.5 × p;  v1 := sqrt(−D);
      u2 := −0.5 × p;  v2 := −sqrt(−D);
B:    D := 0 end
```

Diese zusammengesetzte Anweisung enthält manches Überflüssige. So ist die markierte Anweisung „*B*: $D := 0$“ nur angeführt worden, damit zur Anweisung „**go to** *B*“, die ein Überspringen der vier darauffolgenden Anweisungen veranlassen soll, eine mit *B* markierte Anweisung existiert. Wird eine neue Art von Anweisung, die *leere Anweisung*, eingeführt, so läßt sich statt dessen

```
...;  B: end
```

schreiben. Die leere Anweisung tritt im Schriftbild überhaupt nicht in Erscheinung, sie ist nichts anderes als eine weggelassene Anweisung und eigentlich nur zu dem Zweck eingeführt, um zum Ende einer zusammengesetzten Anweisung springen zu können. Weiter fällt auf, daß der Ausdruck „$-0.5 \times p$“ zweimal berechnet werden muß. Das kann verbessert werden, wenn Wertzuweisungen eingeführt werden, in denen mehreren Variablen gleichzeitig derselbe Wert zugewiesen wird. Die betreffenden Variablen werden dazu alle durch jeweils ein ALGOL-Zeichen := getrennt links vom bewertenden Ausdruck notiert. Damit läßt sich die behandelte Anweisung einfacher wie folgt formulieren:

```
begin D := 0.25 × p ↑ 2 − q;
      if D < 0 then go to A;
      u1 := −0.5 × p + sqrt(D);  u2 := −0.5 × p − sqrt(D);
      v1 := v2 := 0;  go to B;
A:    u1 := u2 := 0.5 × p;  v1 := sqrt(−D);  v2 := −sqrt(−D);
B:    end
```

Die Anweisung läßt sich auch gänzlich ohne Sprunganweisung formulieren, auch die mehrfache Berechnung der Teilausdrücke „$-0.5 \times p$“ und „*sqrt(D)*“ (d. h. $\sqrt{D}$) läßt sich vermeiden.

Eine geeignete Form ist etwa

```
begin D := 0.25 × p ↑ 2 − q;
      u1 := u2 := −0.5 × p;  v1 := sqrt(abs(D));
      if D ≧ 0 then
        begin u1 := u1 + v1;  u2 := u2 − v1;  v1 := v2 := 0 end
        else  v2 := −v1
end
```

Der darin auftretende Ausdruck *sqrt(abs(D))* dient zur Berechnung von $\sqrt{|D|}$, allgemein wird in ALGOL *abs(a)* statt $|a|$ geschrieben, wobei *a* ein beliebiger arithmetischer Ausdruck ist. Eine Bemerkung ist noch zur mehrfachen Wertzuweisung zu machen. Es sei z. B.

$$i := v[i] := i + 1$$

eine zweifache Wertzuweisung, der aktuelle Wert von i vor Abarbeitung dieser Anweisung sei 2. Dann bedeutet dies die Zuweisung des Wertes 3 an die Variable i und an die Variable $v[2]$. Es sind die auf der linken Seite einer Wertzuweisung in Indizes vorkommenden Ausdrücke in jedem Fall vor der Zuweisung des Wertes der rechten Seite an die einzelnen Variablen zu berechnen. Daher ist nicht der neue Wert von i, d. h. 3, in $v[i]$ zu berücksichtigen, sondern der alte Wert 2.

Der mehrfachen Wertzuweisung entsprechend sind auch mehrfache Markierungen möglich, d. h., es können beliebig viele jeweils durch einen Doppelpunkt getrennte Marken vor eine Anweisung geschrieben werden. Eine Sprunganweisung mit einer dieser Marken als Zielausdruck führt zu der betreffenden Anweisung.

In den hier als Beispiel angeführten Anweisungen traten bereits mehrfach Funktionen auf. Im allgemeinen sind solche Funktionen in besonderen Funktionserklärungen zu definieren, einige sogenannte *Standardfunktionen*, zu deren Bezeichnung festgesetzte Namen dienen, werden jedoch nicht gesondert definiert, sondern haben eine festgelegte Bedeutung. Es handelt sich dabei um elementare Funktionen eines Arguments, das ein arithmetischer Ausdruck ist, der im folgenden mit E bezeichnet sei. Diese Funktionen sind:

abs(E) zur Berechnung von $|E|$

sign(E) $sign(E) = \begin{cases} +1 & \text{für } E > 0 \\ 0 & \text{für } E = 0 \\ -1 & \text{für } E < 0 \end{cases}$

sqrt(E) zur Berechnung von $+\sqrt{E}$ für $E \geqq 0$, sonst undefiniert

sin(E) zur Berechnung des Sinus von E (E im Bogenmaß)

cos(E) zur Berechnung des Cosinus von E (E im Bogenmaß)

arctan(E) zur Berechnung des Hauptwertes von arctan(E), d. h.

$$-\frac{\pi}{2} < \arctan(E) < +\frac{\pi}{2}$$

ln(E) zur Berechnung des natürlichen Logarithmus von E für $E > 0$, sonst undefiniert

exp(E) Zur Berechnung von e^E

entier(E) zur Berechnung der größten ganzen Zahl $\leqq E$

Bemerkung. Mit dem in den Erläuterungen angeführten E ist stets der aktuelle Wert des arithmetischen Ausdrucks E gemeint.

Es seien noch einige zusammengesetzte Anweisungen als Beispiele angegeben:

1. **begin** $z := 125 \times y$; $y := z - entier(z)$ **end**

Dieses Beispiel beschreibt eine Anweisung zur Erzeugung einer neuen Pseudozufallszahl y aus einer alten Pseudozufallszahl y. Ist y anfangs ein ungerades Vielfaches von 2^{-10}, so ergeben sich durch fortlaufende Anwendung dieser Anweisung 256 in $0 < y < 1$ gleichverteilte Zahlen, bei der 256-ten Anwendung entsteht wieder die Ausgangszahl. Eine z. B. von $y = {}^1/_{1024} = 0.0009765625$ ausgehende Folge liefert folgende Werte bei sechsmaliger Anwendung:

$y := 0.1220703125$; $y := 0.2587890625$; $y := 0.3486328125$;
$y := 0.5791015625$; $y := 0.3876953125$; $y := 0.4619140625$.

Eine solche Art der Erzeugung von Pseudozufallszahlen ist der Arbeitsweise von im Dualsystem arbeitenden Rechenautomaten angepaßt.

2. **begin** $i := 1$; $s := 0$; $1 : s := s + a[i] \times b[i]$; $i := i + 1$;
if $i \leqq n$ **then go to** 1 **end**

Bemerkung. Die Zahl 1 tritt in diesem Beispiel sowohl als Marke als auch als Wert einer Variablen auf. Um eventuell mögliche Verwechslungen bei

komplizierteren Programmen auszuschließen, ist bei vielen ALGOL-Übersetzungsprogrammen die Verwendung ganzer Zahlen als Marken nicht zulässig. Es ist daher zweckmäßig, nur Namen als Marken zu verwenden.

Die Anweisung beschreibt die Berechnung des Skalarproduktes s aus zwei Vektoren mit je n Komponenten $a[1]$ bis $a[n]$ und $b[1]$ bis $b[n]$.

3. **begin** $i := 1$; $A: k := i + 1$;
B: $w := a[i,k]$; $a[i,k] := a[k,i]$; $a[k,i] := w$;
$k := k + 1$; **if** $k \leqq n$ **then go to** B;
$i := i + 1$; **if** $i < n$ **then go to** A **end**

Die Anweisung beschreibt die Transponierung einer quadratischen Matrix mit Elementen $a[i,k]$ mit $1 \leqq i, k \leqq n$.

2. Stufe: Laufanweisungen

In den beiden letzten Beispielen treten zu wiederholende Programmteile (Zyklen) auf. Bei der Berechnung des Skalarproduktes ändert sich die Variable i von 1 bis n jeweils um 1, darauf wird jedesmal das Produkt $a[i] \times b[i]$ gebildet und zu s addiert. Dieser Vorgang läßt sich in ALGOL kürzer beschreiben:

begin $s := 0$; **for** $i := 1$ **step** 1 **until** n **do** $s := s + a[i] \times b[i]$ **end**

Die zweite Anweisung innerhalb dieser zusammengesetzten Anweisung ist eine sogenannte *Laufanweisung*, die ausdrückt, daß die hinter **do** stehende Anweisung $s := s + a[i] \times b[i]$ für die Werte $i = 1$ bis $i = n$ wiederholt auszuführen ist. Zu ihrer Formulierung werden die ALGOL-Zeichen **for**, **step**, **until** und **do** gebraucht, die als „für", „Schritt", „bis" und „führe aus" ausgesprochen werden können.

Eine Laufanweisung kann auf mehrere Arten gebildet werden, die am häufigsten vorkommende Form ist

for $V := a_1$ **step** a_2 **until** a_3 **do** A

Dabei steht V für eine beliebige Variable (Laufvariable), a_1, a_2 und a_3 stehen für beliebige arithmetische Ausdrücke, A für eine beliebige Anweisung. Ist der Ausdruck a_2 stets positiv, so wird die Anweisung A ausgeführt, wenn $a_3 \geqq V$ ist. Nach jeder Ausführung von A wird V um a_2 erhöht, und A wird, falls immer noch $a_3 \geqq V$ ist, wiederholt ausgeführt. Die angegebene Anweisung zur Transponierung

einer quadratischen Matrix schreibt sich als Laufanweisung

```
for i := 1 step 1 until n − 1 do
    for k := i + 1 step 1 until n do
        begin w := a[i,k]; a[i,k] := a[k,i]; a[k,i] := w end
```

Bemerkung. Die in einer Laufanweisung vorkommende Anweisung kann wieder eine Laufanweisung sein. Sollen mehrere Anweisungen wiederholt werden, so sind sie durch Einklammern mit **begin** und **end** zu einer einzigen (zusammengesetzten) Anweisung zusammenzufassen.

Weitere Beispiele mit Laufanweisungen sind:

1.
```
for i := 1 step 1 until n do
    for k := 1 step 1 until m do
        begin c[i,k] := 0;
            for j := 1 step 1 until l do
            c[i,k] := c[i,k] + a[i,j] × b[j,k] end
```

Diese Anweisung beschreibt die Multiplikation einer (n,l)-Matrix mit Elementen $a[i,j]$ mit einer (l,m)-Matrix mit Elementen $b[j,k]$. Das Resultat ist die (n,m)-Matrix mit Elementen $c[i,k]$.

2.
```
begin for j := 1 step 1 until n − 1 do
    begin for i := j + 1 step 1 until n do
        begin q := −a[i,j]/a[j,j];
            for k := j + 1 step 1 until n + 1 do
            a[i,k] := a[i,k] + q × a[j,k]
        end
    end;
    x[n] := a[n,n + 1]/a[n,n];
    for k := n − 1 step − 1 until 1 do
    begin s := a[k,n + 1]; for j := k + 1 step 1 until n do
                               s := s − a[k,j] × x[j];
        x[k] := s/a[k,k]
    end
end
```

Bemerkung. Ist der hinter **step** stehende Ausdruck a_2 negativ, so wird die hinter **do** stehende Anweisung ausgeführt, wenn $a_3 \leqq V$ ist.

Die aus drei Teilanweisungen bestehende zusammengesetzte Anweisung beschreibt die Lösung eines linearen Gleichungssystems

$$\sum_{k=1}^{n} a_{ik}x_k = a_{i,n+1} \quad (i = 1, \ldots, n)$$

mittels des Gaußschen Eliminationsverfahrens. Die beiden letzten Teilanweisungen können auch durch eine einzige ersetzt werden:

for $k := n$ **step** $-$ 1 **until** 1 **do**
begin $s := a[k,n+1]$; **for** $j := k+1$ **step** 1 **until** n **do**
$s := s - a[k,j] \times x[j]$;
$x[k] := s/a[k,k]$
end

Hierbei tritt die Besonderheit auf, daß innerhalb der innersten Laufanweisung „**for** $j := k+1$ **step** 1 **until** n **do** ..." im Fall $k = n$ die Laufvariable j den Anfangswert $n+1$ hat. Die Relation $V \leqq a_3$, d. h. hier $j \leqq n$, zur Ausführung der hinter **do** stehenden Anweisung ist somit nicht erfüllt, die Wirkung der Laufanweisung ist die einer leeren Anweisung, s behält den Wert von $a[n,n+1]$, $x[n]$ ergibt sich daher ebenso wie oben angegeben.

Im allgemeinen ist es auch zulässig, daß der in einer Laufanweisung hinter **step** stehende Ausdruck a_2 den Wert 0 annehmen kann. Eine allgemeine Vorschrift zur Abarbeitung von

for $V := a_1$ **step** a_2 **until** a_3 **do** A

kann wieder in ALGOL dargestellt werden und lautet

begin $V := a_1$; M: **if** *sign*$(a_2) \times (V - a_3) > 0$ **then go to** *Ende*;
A; $V := V + a_2$; **go to** M;
Ende: **end**

Aus dieser Darstellung folgen die oben angegebenen Regeln zur Ausführung der Anweisung A und auch, daß A stets ausgeführt wird, wenn a_2 den Wert 0 hat.

Als weiteres Beispiel sei eine Anweisung zur Berechnung des Wertes f eines Polynoms $f(x) = \sum_{k=1}^{n} a_k \cdot x^k$ mittels des Hornerschen Schemas gegeben:

begin $b[n] := a[n]$; **for** $k := n - 1$ **step** $-$ 1 **until** 0 **do**
$b[k] := a[k] + b[k+1] \times x$; $f := b[0]$ **end**

Da stets nur eine Komponente des Vektors b im vorkommenden Ausdruck $a[k] + b[k+1] \times x$ gebraucht wird, lassen sich alle Komponenten bei auto-

matischer Rechnung in der gleichen Speicherzelle speichern, die zuletzt den gesuchten Funktionswert enthält. Die Anweisung vereinfacht sich dadurch zu

begin $f := a[n]$; **for** $k := n - 1$ **step** -1 **until** 0 **do**

$f := a[k] + f \times x$ **end**

Auch für $n = 1$ und $n = 0$ liefert diese Anweisung die richtigen Funktionswerte ($a_1 \cdot x + a_0$ bzw. a_0). Ein nach der automatischen Übersetzung bei den meisten Rechenautomaten kürzeres Maschinenprogramm erhält man mit

begin $f := 0$; **for** $k := n$ **step** -1 **until** 0 **do** $f := a[k] + f \times x$ **end**

Bemerkung. Indizierte Variable sollten so wenig wie möglich verwendet werden, da bei vielen Übersetzungssystemen die Adressenberechnung für indizierte Variable zusätzliche Befehle im erzeugten Maschinenprogramm erfordert. Weiter sei darauf hingewiesen, daß stets deutlich zwischen der Ziffer 0 und dem Buchstaben *O* unterschieden werden muß, zweckmäßiger Weise wird *O* (Buchstabe *O*) nicht als Name benutzt, Entsprechendes gilt für das Multiplikationszeichen × und den Buchstaben *x*. Oft wird statt × auch * geschrieben.

Soll für $n = 1$ und $n = 0$ die Abarbeitung der Laufanweisung vermieden werden, so kann z. B. geschrieben werden:

begin $f := 0$; **if** $n > 1$ **then**

begin for $k := n$ **step** -1 **until** 0 **do** $f := a[k] + f \times x$ **end**

else if $n = 1$ **then** $f := a[1] \times x + a[0]$ **else** $f := a[0]$

end

Bemerkung. Es ist nicht zulässig, daß in der bedingten Anweisung

if R **then** A_1 **else** A_2

die Anweisung A_1 eine Laufanweisung ist; gegebenenfalls ist eine Laufanweisung durch Einklammern in **begin** und **end** zu einer „zusammengesetzten Anweisung" zu machen. Der Grund dafür ist, daß

if R **then for** $i := 1$ **step** 1 **until** n **do if** R_1 **then** A_3 **else** A_2

sowohl durch

if R **then begin for** $i := 1$ **step** 1 **until** n **do if** R_1 **then** A_3 **end else** A_2

als auch durch

if R **then begin for** $i := 1$ **step** 1 **until** n **do if** R_1 **then** A_3 **else** A_2 **end**

interpretiert werden könnte. Es ist jedoch erlaubt, in

if R **then** A_1

für A_1 eine Laufanweisung zu verwenden, d. h., der als Beispiel für

eine mögliche Zweideutigkeit angegebenen Anweisung wird eindeutig die zweite Bedeutung zugewiesen. Als Merkregel kann man aufstellen, daß sich das Zeichen **else** auf das näherliegende Zeichen **then** bezieht, d. h. die Alternative zu A_3 und nicht zu einer nach A_3 abschließenden Laufanweisung einleitet.

Die im letzten Beispiel vorkommende Bedingung „**if** $n = 1$" kann bei der Realisierung in Rechenautonaten auf Schwierigkeiten stoßen, wenn n als Resultat einer anderen Rechnung entstanden ist, bei der durch Rundungsfehler statt des gewünschten Wertes $n = 1$ ein etwas davon verschiedener Wert auftritt. Theoretisch ist dann die Bedingung erfüllt, praktisch jedoch nicht. Das läßt sich vermeiden, wenn vorher festgelegt wird, daß die Variable n nur ganzzahlige Werte annehmen darf und bei jeder Wertzuweisung an n entsprechend gerundet wird, Man bezeichnet eine solche Festlegung als Typerklärung oder Typvereinbarung; durch sie wird einer Variablen ein Typ zugeordnet (ganzzahlig oder reell). Allgemein dienen *Erklärungen* oder *Vereinbarungen* zur Zuordnung gewisser Eigenschaften zu vorkommenden Namen.

3. Stufe: Typ- und Felderklärungen

In ALGOL 60 sind alle vorkommenden Namen mit Ausnahme der Namen für Marken und für sogenannte formale Parameter zu erklären, d. h., sie sind in einer Erklärung aufzuführen. Auch Namen für Standardfunktionen und Standardprozeduren werden nicht erklärt. Eine Erklärung ist innerhalb eines Blockes gültig. Ein Block ist eine Anweisung, die wie eine zusammengesetzte Anweisung aus einer Folge von in **begin** und **end** eingeschlossenen Anweisungen besteht. Zwischen dem Zeichen **begin** und der ersten dieser Anweisungen sind noch Erklärungen für gewisse innerhalb des Blockes vorkommende Namen notiert. Die Zeichen **begin** und **end** erhalten dadurch eine weitere Bedeutung, sie dienen sowohl zur Kennzeichnung zusammengesetzter Anweisungen als auch zur Kennzeichnung von Blöcken. In ALGOL werden Variable, Felder von Variablen sowie die noch zu besprechenden Prozeduren (Funktionen und Unterprogramme) und Verteiler mit einem gemeinsamen Ausdruck Größen genannt und durch Namen bezeichnet. Größen, deren Namen in einem Block erklärt (vereinbart) werden, heißen in diesem Block lokale Größen. Ein Name der lokalen Größe hat außerhalb des Blocks keine Bedeutung, sofern er dort nicht nochmals erklärt ist. In einer Erklärung kann mehreren Namen die gleiche Eigenschaft zugeordnet werden, diese Namen sind dann stets in einer Liste durch Kommata getrennt zusammengefaßt. Ein Block gilt als Anweisung, kann demnach innerhalb eines anderen Blocks ebenso wie in einer bedingten Anweisung, Laufanweisung oder zusammengesetzten

Anweisung vorkommen. Da Blöcke verschachtelt werden können, ist die Bezeichnung lokale Größe relativ. Ist etwa in einem Block B_1 ein weiterer Block B_2 als Anweisung enthalten und wird in B_1 ein Name a als Bezeichnung für eine ganzzahlige Variable a erklärt, so ist a bezüglich B_1 lokal, nicht jedoch bezüglich B_2; a heißt dann in B_2 global. Es ist allerdings auch möglich, in B_2 den Namen a neu zu erklären, er hat dann bei jedem Auftreten in B_2 diese neue Bedeutung. Man sollte jedoch versuchen, in einem Programm das Auftreten gleicher Namen für verschiedene Größen zu vermeiden; Ausnahmen sollten nur die formalen Parameter machen (Argumente in Funktionserklärungen bzw. Erklärungen von Unterprogrammen). Die am Blockanfang stehenden Erklärungen sind jeweils durch Semikolon getrennt. Auch dieses Zeichen erhält dadurch eine weitere Bedeutung.

Neben den bereits erwähnten Typerklärungen gibt es in ALGOL 60 Felderklärungen, Verteilererklärungen sowie Funktions- und Unterprogrammerklärungen, die beiden letzten sind nur wenig verschieden, werden daher meist gemeinsam als Prozedurerklärungen bezeichnet. Besonders diese Prozedurerklärungen, deren Zweck die Bezeichnung ganzer Algorithmen durch einen Namen ist, ermöglichen den vielseitigen Einsatz von ALGOL.

Die einfachsten Erklärungen sind die *Typerklärungen* von der Form

real L oder **integer** L.

Dabei sind **real** und **integer** ALGOL-Zeichen mit der empfohlenen deutschen Aussprache „reell" und „ganzzahlig"; L steht für eine Liste von Namen, von denen jeder einzelne dadurch als Bezeichnung einer einfachen (nicht indizierten) Variablen definiert wird, deren Wertebereich die Menge der reellen bzw. ganzen Zahlen ist. Diese Definition ist nur innerhalb des Blockes gültig, in dem die Erklärung steht. Eine dritte Art von Typerklärungen ist

Boolean L.

Diese Erklärung definiert die Name der Liste L als Bezeichnungen zweiwertiger Variabler, deren Definitionsbereich die Menge {**true**, **false**} ist. Für die ALGOL-Zeichen **Boolean**, **true** und **false** ist als deutsche Aussprache „Boolesch", „richtig" und „falsch" empfohlen worden. Diese zweiwertigen Variablen dienen zur Durchführung von Rechnungen im Aussagenkalkül der mathematischen Logik, mit ihnen können Aussagen, die in ALGOL als *Boolesche Ausdrücke*[1]) bezeichnet werden, gebildet werden, die als Werte ebenfalls nur **true** und **false** annehmen können. Die Booleschen Ausdrücke kommen in Bedingungen hinter dem ALGOL-Zeichen **if** vor, die bisher erwähnten Relationen zwischen arithmetischen Aus-

[1]) Nach GEORGE BOOLE (englischer Mathematiker, 1815–1864).

drücken gelten in ALGOL ebenfalls als Boolesche Ausdrücke, die den Wert **true** annehmen, wenn die Relation erfüllt ist, andernfalls den Wert **false**.

Felderklärungen dienen zur Definition der Wertebereiche für die Indizes von indizierten Variablen und zur Festlegung der zu ihrer Bezeichnung verwendeten Namen. Die Gesamtheit der indizierten Variablen mit einem Namen heißt ein Feld, je nach Anzahl der Indizes können wir von eindimensionalen Feldern (Vektoren), zweidimensionalen Feldern (Matrizen), dreidimensionalen Feldern usw. sprechen. Eine Felderklärung kann z. B. die Form

array $L[G]$

haben. Das Grundsymbol **array** wird „Feld" ausgesprochen, L steht wieder für eine Liste von Namen, G für eine Grenzenliste. L enthält die Namen von Feldern, die gleiche Dimension und gleiche Wertebereiche für die Indizes haben. Die Grenzenliste besteht aus einer der Dimension entsprechenden Anzahl von durch jeweils ein Komma getrennten Grenzenpaaren. Jedes Grenzenpaar hat die Form „$a_1 : a_2$", wobei a_1 und a_2 ganzzahlige arithmetische Ausdrücke sind. Ist $a_{n1} : a_{n2}$ das n-te Grenzenpaar einer Grenzenliste, so muß der Wert i_n des n-ten Index einer indizierten Variablen mit einem Namen aus der zugehörigen Feldliste L der Bedingung $a_{n1} \leqq i_n \leqq a_{n2}$ genügen. Andernfalls darf die indizierte Variable nicht verwendet werden; alle Ausdrücke und Anweisungen, in denen eine solche unzulässige Variable vorkommt, gelten als nicht definiert. Um den Wertebereich für die indizierten Variablen zu definieren, sind die Felderklärungen jeweils mit Typerklärungen verbunden und haben dementsprechend eine der Formen

real array $L[G]$; **integer array** $L[G]$ oder **Boolean array** $L[G]$.

Es ist möglich, in der ersten Form das Zeichen **real** wegzulassen. Weiter können mehrere Felderklärungen gleichen Typs zu einer einzigen Felderklärung, z. B. der Form

array $L_1[G_1], L_2[G_2], \ldots, L_n[G_n]$

vereinigt werden. In die Grenzenpaare dürfen stets nur globale Veränderliche und Konstante eingehen.

Zur Erläuterung der Verwendung von Typ- und Felderklärungen seien einige der bereits gebrachten Beispiele nochmals in Form von Blöcken notiert:

1. **begin real** x,y; $y := (x + 0.16)/(0.46 \times x + 0.70)$;
   $y := 0.5 \times (y + x/y)$; $y := 0.5 \times (y + x/y)$
   end

2. **begin integer** i; **real** s; **array** $a,b[1:n]$;
   $i := 1$; $s := 0$; $1 : s := s + a[i] \times b[i]$; $i := i + 1$;
   if $i \leqq n$ **then go to** 1
   end

3. **begin integer** i,j,k; **array** $a[1:n, 1:l]$;
   array $b[1:l, 1:m]$; **array** $c[1:n, 1:m]$;
   for $i := 1$ **step** 1 **until** n **do**
   for $k := 1$ **step** 1 **until** m **do**
   begin $c[i,k] := 0$;
   for $j := 1$ **step** 1 **until** l **do**
   $c[i,k] := c[i,k] + a[i,j] \times b[j,k]$
   end
   end

In ALGOL 60 sind Programme Blöcke, in denen alle vorkommenden Namen erklärt sind, oder zusammengesetzte Anweisungen ohne zu erklärende Namen. Die Blöcke unter 2. und 3. sind hiernach keine ALGOL-Programme, da die Namen n, l und m in ihnen nicht erklärt sind. Diese Namen dürfen in den Blöcken auch nicht erklärt werden, da sie in Grenzenpaaren vorkommen und darin keine lokalen Größen zulässig sind. Das zweite Beispiel läßt sich etwa in folgender Weise zu einem Programm erweitern:

begin integer n; $n := 100$;
begin integer i; **real** s; **array** $a,b[1:n]$;
$i := 1$; $s := 0$; $A : s := s + a[i] \times b[i]$; $i := i + 1$;
if $i \leqq n$ **then go to** A
end
end

Dieses Programm ist ein Block mit der lokalen Größe n, der aus zwei Anweisungen besteht; die zweite ist wieder ein Block mit den lokalen Größen i, s, a und b sowie der ebenfalls als lokale Größe geltenden Marke A, durch die hier die oben benutzte Marke 1 ersetzt wurde, um ein Beispiel für einen nicht besonders zu erklärenden Namen zu geben. Die Variable n ist bezüglich dieses im ersten Block enthaltenen inneren Blockes global, kann danach in einer Felderklärung

array $a,b[1:n]$

benutzt werden. Darüber hinaus ist noch zu bemerken, daß jeder zur Berechnung von Indexgrenzen benutzten Variablen beim Eintritt in den betreffenden Block bereits ein Wert zugeordnet sein muß. Das geschieht im Beispiel durch die Anweisung n: = 100.

Auch allen anderen Variablen muß vor ihrer Verwendung ein Wert zugeordnet sein, da sonst nicht mit ihnen gerechnet werden kann. Aus diesem Grunde ist der im ersten Beispiel angegebene Block nur formal ein Programm, da der Variablen x bei Ausrechnung des ersten Ausdrucks kein Wert zugeordnet ist. Diese Wertzuweisung kann außerhalb des Blocks geschehen, dann darf x aber nicht im Block erklärt werden. Ein sinnvolles Programm ergibt sich etwa durch

begin real x; $x := 0.5$;

begin real y; $y := (x + 0.16)/(0.46 \times x + 0.70)$;
$y := 0.5 \times (y + x/y)$; $y := 0.5 \times (y + x/y)$
end

end

Bei komplizierten Programmen muß man sehr aufmerksam sein, um nicht in einer Anweisung eine unbewertete Größe zu benutzen. Später zu besprechende besondere Anweisungen dienen zur Einstellung der benötigten Anfangswerte durch Ablesen von einem Eingabemedium.

Bei Eintritt in einen Block sind die im Block lokalen Größen unbewertet. Eine Ausnahme bilden nur die als **own** erklärten Variablen und Felder (vgl. 6. Stufe). Bei Austritt aus dem Block verlieren die Erklärungen ihre Gültigkeit, die betreffenden Größen sind dann nicht definiert.

4. Stufe: Prozeduren

Zur Definition von Funktionen dienen *Funktionserklärungen*, die zum Namen einer Funktion eine Vorschrift zur Berechnung der Funktion zuordnen. Eine Funktion hängt von einer Anzahl von Argumenten ab, die bei jeder Anwendung der Funktion andere Variable oder Ausdrücke sein können. Es kann z. B. einmal $sin(x + 1)$ und ein andermal $sin\ (y)$ gebraucht werden, jedesmal ist die gleiche Funktion, d. h. die gleiche Rechenvorschrift (der gleiche Algorithmus), gemeint, sie wird nur mit anderen Parametern ausgeführt. In der Funktionserklärung wird die Rechenvorschrift unter Verwendung freier oder *formaler Parameter* notiert, die bei der aktuellen Ausführung entweder durch aktuelle Parameter ersetzt werden oder mit Werten aktueller Parameter bewertet werden. Beim klassischen Funktionsbegriff kann man die Rechenvorschrift mit Variablen notieren, denen vor der jeweiligen Berechnung des Funktionswertes bestimmte Werte zugewiesen werden. Der in ALGOL verwendete Funktionsbegriff ist allgemeiner, er ist aus der in Rechenautomaten üblichen Unterprogrammtechnik entwickelt worden, in ihm können z. B. auch formale Parameter als formale Marken oder als formale Namen für Unterprogramme auftreten, die dann bei jeder Ausführung durch die aktuelle Marke bzw. den aktuellen Unterprogrammnamen zu ersetzen sind.

Die für Funktionserklärungen verwendete Schreibweise sei an einem Beispiel erörtert:

```
real procedure wurzel (x); value x; real x;
    begin real y; y := (x + 0.16)/(0.46 × x + 0.70);
                  y := 0.5 × (y + x/y); wurzel := 0.5 × (y + x/y) end
```

Neu eingeführt wurden die Grundsymbole **procedure** (Prozedur) und **value** (Wert). Zur Kennzeichnung, daß es sich um eine Funktion mit reellem Wertebereich handelt, ist vor das Zeichen **procedure** ein Zeichen **real** zu setzen. Die Funktion wird mit dem Namen „*wurzel*" bezeichnet. Auf diesen Namen folgt in Klammern eingeschlossen der formale Parameter x; im allgemeinen kann eine ganze Liste von formalen Parametern (Namen) auf den Prozedurnamen folgen. Dieser formale Parameter x wird durch die Angabe **value** x als ein formaler Parameter charakterisiert, dem vor jeder Abarbeitung des die Funktion definierenden Algorithmus der Wert eines aktuellen Parameters zuzuweisen ist. Die darauf folgende *Spezifikation* **real** x ist zwar keine Erklärung, da sie sich nicht auf eine Variable bezieht, sondern auf einen formalen Parameter, sie hat aber eine gleichartige Wirkung, die bei der folgenden Erläuterung des Prozeduraufrufs zu besprechen ist. Der die Funktionserklärung abschließende Block enthält die Anweisung

$$wurzel := 0.5 \times (y + x/y),$$

durch deren Ausführung der aktuelle Funktionswert festgelegt wird. Dabei wird der Funktionsname „*wurzel*" wie eine einfache Variable von Typ **real** behandelt.

Funktionen werden bei der Bildung von Ausdrücken wie Variable benutzt. Die Funktion „*wurzel*" kann etwa in einer Wertzuweisung vorkommen, z. B. in

$$z := x/y - wurzel(x \uparrow 2 + y \uparrow 2) + y.$$

Die Ausführung dieser Wertzuweisung läßt sich durch folgenden Block beschreiben:

```
begin real Hilfsvariable, wurzel; Hilfsvariable := x/y;
    begin real x1; x1 := x ↑ 2 + y ↑ 2;
        begin real y; y := (x1 + 0.16)/(0.46 × x1 + 0.70);
                      y := 0.5 × (y + x1/y);
                  wurzel := 0.5 × (y + x1/y)
        end
    end; z := Hilfsvariable − wurzel + y
end[1])
```

[1]) Man beachte, daß y in der abschließenden Anweisung nicht mit dem lokalen y im innersten Block übereinstimmt; dieses hat nach dem Verlassen des Blocks keine Bedeutung mehr.

Allgemein wird beim *Prozeduraufruf*, der durch Erkennen des Prozedurnamens und einer anschließenden in Klammern gesetzten Liste von *aktuellen Parametern* (Namen oder Ausdrücke) eingeleitet wird, der durch die Prozedurerklärung gegebene Algorithmus abgearbeitet. Dieser Algorithmus wird dabei als Block angesehen, der den aufrufenden Namen ersetzt, wobei gegebenenfalls einige Namen (formale Parameter) innerhalb des Algorithmus durch andere Namen, Ausdrücke oder Zeichenfolgen zu ersetzen sind. Es ist dabei gleichgültig, ob es sich um eine Funktionserklärung oder eine Unterprogrammerklärung handelt; im Gegensatz zu Unterprogrammen haben Funktionen einen Wert vom Typ **real**, **integer** oder **Boolean**, und beim Funktionsaufruf wird der Funktion dieser Wert zugeordnet. Die Anzahl der aktuellen Parameter muß mit der Anzahl der formalen Parameter übereinstimmen. Beim Aufruf werden die aktuellen Parameter den formalen Parametern in der notierten Reihenfolge zugeordnet.

Gewisse formale Parameter können als *Parameter mit Anfangswert* definiert werden, indem sie in der Prozedurerklärung in einer auf das Zeichen **value** folgenden Liste aufgeführt werden. Solchen Parametern wird vor Abarbeitung des Algorithmus der Wert des zugeordneten aktuellen Parameters zugewiesen. Diese formalen Parameter müssen innerhalb der Prozedurerklärung spezifiziert sein, d. h., ihnen wird eine Eigenschaft zugeordnet, die mit den Eigenschaften des betreffenden aktuellen Parameters verträglich sein muß. In ALGOL werden als Werte angesehen: reelle Zahlen (ganze Zahlen als Spezialfall) und Wahrheitswerte (**true** und **false**) als Werte von einfachen und indizierten Variablen, geordnete Mengen von Werten indizierter Variablen als Werte von Feldern, Marken als Werte von Zielausdrücken. Ein Parameter mit Anfangswert kann daher als **real**, **integer**, **Boolean**, **(real) array**, **integer array**, **Boolean array** oder **label** (Marke) spezifiziert werden. Die Wertzuweisung an diese Parameter geschieht nun vor Ausführung des in jedem Fall als Block anzusehenden Algorithmus, und zwar außerhalb dieses Blockes. Bei als **real**, **integer** oder **Boolean** spezifizierten Parametern denkt man sich die formalen Parameter als Variable entsprechenden Typs erklärt, denen sogleich die Werte der aktuellen Parameter zugewiesen werden. Damit hierbei keine Schwierigkeiten auftreten, sind die Namen der formalen Parameter gegebenenfalls passend zu ändern. Im aufgeführten Beispiel ist zur Beschreibung der Ausführung eine solche Änderung vorzunehmen. Der Parameter x wird bei Ausführung durch die reellwertige Variable $x1$ ersetzt, damit der Block

begin real $x1$; $x1 := x \uparrow 2 + y \uparrow 2$; **begin** ... **end end**

notiert werden kann. Würde die Bezeichnung x des formalen Parameters für die bei der Ausführung auftretende Variable übernommen, so bedeutete das im soeben notierten Block, daß die Variable x neu erklärt würde und der Ausdruck $x \uparrow 2$

$+ y \uparrow 2$ nicht gebildet werden könnte, weil dieser Variablen kein Wert zugeordnet wäre. Einem als Feld spezifizierten formalen Parameter (durch **array** usw.) darf nur der Name eines (erklärten) Feldes als aktueller Parameter zugeordnet sein; handelt es sich um einen Parameter mit Anfangswert, so ist dieser Parameter als Feld mit den gleichen Grenzen wie der entsprechende aktuelle Parameter erklärt zu denken. Diesem Feld wird der Wert des aktuellen Feldes zugeordnet, d. h., jeder seiner indizierten Variablen wird der Wert der entsprechenden indizierten Variablen des aktuellen Feldes zugewiesen. Dieser Vorgang läßt sich in ALGOL gegebenenfalls nach Abänderung des Namens des formalen Parameters wie bei einfachen Variablen beschreiben. Ist ein Parameter mit Anfangswert als **label** spezifiziert, so ist ihm ein Zielausdruck als aktueller Parameter zuzuordnen. Der Wert dieses Zielausdruckes (eine Marke) ersetzt vor Ausführung des Algorithmus in dessen ALGOL-Notation (dem *Prozedurrumpf*) den zugeordneten formalen Parameter überall.

Formale Parameter, die nicht in der auf **value** folgenden Liste von Namen aufgeführt sind, werden anders behandelt. Sie werden vor Ausführung des Algorithmus im Prozedurrumpf überall durch die zugeordneten aktuellen Parameter ersetzt, und zwar durch die betreffende Zeichenfolge (Namen oder Ausdruck), und nicht durch den Wert des aktuellen Parameters. Während die oben beschriebene Zuweisung eines Wertes an einen formalen Parameter als *Wertaufruf* (call by value) bezeichnet wird, spricht man bei dieser Substitution vom *Namensaufruf* (call by name). Diese durch Zeichenfolgen zu ersetzenden Parameter seien hier als *symbolische Parameter* bezeichnet, sie halten einen Platz für den aktuellen Parameter frei. Kommen in den aktuellen Parametern Namen vor, die auch als Bezeichnung formaler Parameter mit Anfangswert oder lokaler Größen des Prozedurrumpfes auftreten, so sind diese formalen und lokalen Größen durch passend abgeänderte Bezeichnungen zu ersetzen. Ein Prozeduraufruf kann nur ausgeführt werden, wenn die vorzunehmenden Substitutionen und Bewertungen zu Anweisungen führen, die mit den ALGOL-Regeln verträglich sind. Gegebenenfalls sind die substituierenden Zeichenfolgen in Klammern einzuschließen. Symbolische Parameter können ebenfalls spezifiziert werden, es ist jedoch nicht vorgeschrieben. Neben den bereits erwähnten Spezifizierungen können noch folgende verwendet werden: **procedure**, **real procedure**, **integer procedure**, **Boolean procedure**, **switch** (Verteiler) und **string** (Zeichenreihe). Es ist erforderlich, daß die durch die Spezifikation eines formalen Parameters angegebenen Eigenschaften sowohl mit der Verwendung des Parameters im Prozedurrumpf als auch mit den Eigenschaften des beim Prozeduraufruf zugeordneten aktuellen Parameters verträglich sind. Zum Beispiel kann ein als **array** spezifizierter formaler Parameter nicht durch eine als **real** erklärte aktuelle Variable ersetzt werden. Auf die Begriffe „Verteiler" und

„Zeichenreihe" wird später eingegangen. Hier sei noch erwähnt, daß auch parameterlose Prozeduren zulässig sind, es treten dann bei der Erklärung keine formalen Parameter auf, entsprechend auch keine aktuellen Parameter beim Aufruf.

Zur Erläuterung sollen einige weitere Funktionserklärungen aufgestellt und analysiert werden. Dabei sollen die Funktionsnamen so gewählt werden, daß sie möglichst kurz sind und bereits eine gewisse Aussage über die Funktion enthalten.

1. **real procedure** *max*(*x*,*y*); **value** *x*, *y*; **real** *x*, *y*;
   if $x < y$ **then** $max := y$ **else** $max := x$

Bemerkung. Der Prozedurrumpf braucht nicht die Form eines Blockes zu haben, er wird bei Ausführung der Prozedur trotzdem als Block angesehen.

Die Funktion *max* mit den zwei Argumenten *x* und *y* (reelle Zahlen) erhält den Wert der größeren der beiden Zahlen, bei Gleichheit ihren gemeinsamen Wert. Die Ausführung eines Aufrufs „*max*(*a*,*b*)" geschieht in der Art

begin real *x*, *y*; $x := a$; $y := b$;
    begin if $x < y$ **then** $max := y$ **else** $max := x$ **end**
end

Dabei wird *max* wie eine globale reelle Veränderliche behandelt, die außerhalb dieses Blocks erklärt ist. Man beachte jedoch, daß *max* keine Variable, sondern eine Funktion bezeichnet, also nicht außerhalb des Prozedurrumpfes wie eine Variable behandelt werden kann; auch im Prozedurrumpf darf *max* in isolierter Form nur auf der linken Seite einer Wertzuweisung vorkommen.

2. **real procedure** *poly* (*a*,*n*,*x*); **value** *n*, *x*; **array** *a*;
   integer *n*; **real** *x*;
   begin integer *k*; **real** *t*; $t := 0$;
   for $k := n$ **step** -1 **until** 0 **do** $t := a[k] + t \times x$;
   $poly := t$
   end

Diese Funktionserklärung gibt einen Algorithmus (Hornersches Schema) zur Berechnung von $poly\,(a,n,x) = \sum_{k=0}^{n} a_k \cdot x^k$ an (vgl. [2]). Bei einem Aufruf muß der erste aktuelle Parameter Name eines eindimensionalen Feldes sein, der zweite und dritte Parameter müssen arithmetische Ausdrücke sein, wobei der zweite

einen ganzzahligen Wert haben soll. So bedeutet etwa $f := poly\ (A,\ 10,\ sqrt(u \uparrow 2 + v \uparrow 2) + 1)$ die Berechnung von $f = \sum_{k=0}^{10} A_k(\sqrt{u^2 + v^2} + 1)^k$. Die Ausführung läßt sich beschreiben durch

```
begin real poly;
    begin integer n; real x; n := 10; x := sqrt(u ↑ 2 + v ↑ 2) + 1;
        begin integer k; real t; t := 0;
            for k := n step −1 until 0 do t := A[k] + t × x;
            poly := t
        end
    end; f := poly
end
```

Sind in der Erklärung alle drei Parameter symbolische Parameter, so erfolgt die Ausführung nach

```
begin real poly;
    begin integer k; real t; t := 0;
        for k := 10 step − 1 until 0 do
        t := A[k] + t × (sqrt(u ↑ 2 + v ↑ 2) + 1);
        poly := t
    end; f := poly
end
```

Bei der automatischen Übersetzung kann die Verwendung symbolischer Parameter an Stelle von Parametern mit Anfangswert zu kürzeren Maschinenprogrammen führen, jedoch kann dadurch in Spezialfällen eine wesentlich längere Rechenzeit entstehen. Im vorliegenden Beispiel ist mit symbolischem Parameter x der Ausdruck $sqrt(u \uparrow 2 + v \uparrow 2) + 1$ elfmal zu berechnen; falls x ein Parameter mit Anfangswert ist, nur einmal. Bei Feldern ist es im allgemeinen zweckmäßiger, symbolische Parameter zu benutzen, da für Felder mit Anfangswert nochmals der gleiche Speicherraum benötigt wird wie für das aktuelle Feld. Es kann jedoch Fälle geben, in denen der Aufbau des Programms die Einführung eines Feldes als formaler Parameter mit Anfangswert als zweckmäßig erscheinen läßt.

In der Formulierung der Funktionserklärung für die Funktion *poly* wird die lokale Veränderliche t eingeführt. Sofern man auf eine solche Veränderliche verzichten wollte und den Prozedurrumpf

```
begin integer k; poly := 0; for k := n step − 1 until 0 do
                                poly := a[k] + poly × x
end
```

schriebe, so wäre das eine in ALGOL 60 unzulässige Formulierung. Der Name *poly* bezeichnet nämlich eine Funktion von drei Veränderlichen, aber nicht eine Variable. Er muß jedoch in der Funktionserklärung mindestens einmal auf der linken Seite einer Wertzuweisung vorkommen, damit der Funktion ein Wert zugeordnet werden kann. Jedes andere Vorkommen des Prozedurnamens innerhalb des Prozedurrumpfes bewirkt einen Prozeduraufruf, dazu muß jedoch die vorgeschriebene Anzahl aktueller Parameter mit notiert werden. Man bezeichnet einen solchen Aufruf der gleichen Prozedur während ihrer Ausführung als *rekursiven Aufruf*. Ein Beispiel dafür ist

```
3. real procedure POLY(a,n,x); value x; array a;
                                integer n; real x;
     if n > 0 then POLY := a[n] × x ↑ n + POLY(a,n − 1,x)
              else POLY := a[0]
```

Die Ausführung sei an Hand der Anweisung $F := POLY(B,2, Z \uparrow 2)$ erläutert. Entsprechend dem bisherigen Vorgehen ergibt sich folgende Durchführung:

```
begin real POL;
    begin real x; x := Z ↑ 2;
        begin if 2 > 0 then POL := B[2] × x ↑ 2 +
                                   POLY(B, 2 − 1,x)
                               else POL := B[0]
        end
    end; F := POL
end
```

Die hierin auftretende Funktion $POLY\ (B, 2 - 1, x)$ ist nach dem gleichen Schema auszuwerten, dabei tritt eine Funktion $POLY(B, 2 - 1 - 1, x)$ auf. Wird diese ausgewertet, so hat die Relation $2 - 1 - 1 > 0$ den Wert **false**, und die Auswertung der an sich auftretenden Funktion $POLY(B, 2 - 1 - 1 - 1, x)$ erfolgt nicht, sondern es ergibt sich der Wert von $POLY(B, 2 - 1 - 1, x)$ – der natürlich gleich dem Wert von $POLY(B, 0, x)$ ist – zu $B[0]$. Daraus ergibt sich wiederum der Wert von $POLY(B, 2 - 1, x)$ zu $B[1] \times x \uparrow 1 + B[0]$ und schließlich der Wert von $POLY(B, 2, x)$ zu $B[2] \times x \uparrow 2 + B[1] \times x \uparrow 1 + B[0]$.

Der rekursive Aufruf von Prozeduren führt oft zu kurzen Formulierungen in ALGOL, denen jedoch meist komplizierte Maschinenprogramme und lange

Rechenzeiten entsprechen. Daher sollte man diese Art der Programmierung möglichst vermeiden (vgl. [48]). Bei vielen ALGOL-Übersetzern ist ein rekursiver Aufruf auch nicht zugelassen. Die Beschreibung der Ausführung eines rekursiven Prozeduraufrufs durch eine ALGOL-Anweisung kann nicht in Form eines gestreckten Programms geschehen, da im allgemeinen nicht bekannt ist, wie oft der rekursive Aufruf erfolgt. Eine den Ablauf des Algorithmus „*POLY*" beschreibende Prozedur kann etwa folgende Form haben:

```
4. real procedure POLY1(a, n, x); value x; integer n; real x;
     begin array Keller[0 : n − 1]; integer j; real POLYH;
         if n > 0 then
         begin for j := 0 step 1 until n − 1 do
               Keller[j] := a[n − j] × x ↑ (n − j) end
         POLYH := a[0];
         if n > 0 then
         begin for j := n − 1 step − 1 until 0 do
               POLYH := Keller[j] + POLYH end
         POLY1 := POLYH
     end
```

Bemerkung. Formale Parameter können als globale Größen im Prozedurrumpf auch in Felderklärungen vorkommen.

Das lokale Feld „*Keller*" nimmt Zwischenresultate auf, die in der Prozedur *POLY* bei jedem der rekursiven Aufrufe entstehen. Diese Zwischenresultate werden bei der Bewertung der lokalen Hilfsvariablen *POLYH* benutzt, die nacheinander die Werte annimmt, die bei den einzelnen rekursiven Aufrufen der Funktion *POLY* zugeordnet werden.

```
5. integer procedure ggT(a,b, Ausgang); integer a, b; label Ausgang;
       begin integer r;
             if a ≦ 0 then go to Ausgang;
             if b ≦ 0 then go to Ausgang;
             if a < b then begin r := a; a := b; b := r end
A: r := a − b × entier(a/b);
             if r = 0 then ggT := b else
             begin a := b; b := r; go to A end
       end
```

Bemerkung. Statt *entier*(*a*/*b*) kann auch $a \div b$ geschrieben werden. Das Operationszeichen $\div$ für ganzzahlige Division ist nur für ganzzahlige Operanden definiert und liefert das ganzzahlige Resultat *sign*(*a*/*b*) $\times$ *entier*(*abs*(*a*/*b*)).

Die Funktion *ggT* bezeichnet den größten gemeinsamen Teiler der natürlichen Zahlen *a* und *b*. Ist eine der beiden Zahlen keine natürliche Zahl, so wird zur Anweisung mit der formalen Marke „*Ausgang*" gesprungen, z. B. bewirkt *ggT*(−3, 1, *B*) einen Sprung zu der mit „*B*" markierten Anweisung. Das angewandte Verfahren zur Bestimmung des größten gemeinsamen Teilers ist der bekannte Euklidische Algorithmus.

Während Funktionen bei der Bildung von Ausdrücken auftreten, werden Unterprogramme durch eine *Unterprogrammanweisung* aufgerufen (auch als *Prozeduranweisung* bezeichnet). Eine Unterprogrammanweisung besteht aus einem Namen, gefolgt von einer eingeklammerten Liste aktueller Parameter, unterscheidet sich in der Schreibweise also nicht von einer Funktion. Der Name ist einem Algorithmus zugeordnet, der in einer *Unterprogrammerklärung* definiert sein muß, die genauso wie eine Funktionserklärung aufgebaut ist, nur ist das erste Zeichen der Erklärung das ALGOL-Zeichen **procedure** ohne ein vorgesetztes Typzeichen (**real**, **integer**, **Boolean**). Die Ausführung erfolgt in der beim Funktionsaufruf erörterten Weise.

Es seien einige Beispiele für Unterprogrammerklärungen angeführt:

1. **procedure** *add*(*a*1, *a*2, *b*1, *b*2, *r*1, *r*2);
   real *a*1, *a*2, *b*1, *b*2, *r*1, *r*2;
   begin *r*1 := *a*1 + *b*1; *r*2 := *a*2 + *b*2 **end**

2. **procedure** *Mult*(*a*1, *a*2, *b*1, *b*2, *r*1, *r*2);
   real *a*1, *a*2, *b*1, *b*2, *r*1, *r*2;
   begin *r*1 := *a*1 × *b*1 − *a*2 × *b*2;
   *r*2 := *a*2 × *b*1 + *a*1 × *b*2 **end**

Die Unterprogramme „*add*" und „*Mult*" dienen zur Addition bzw. Multiplikation zweier komplexer Zahlen $a1 + i \cdot a2$ und $b1 + i \cdot b2$ mit dem Resultat $r1 + i \cdot r2$. Mit Hilfe dieser beiden Unterprogrammerklärungen kann die Berechnung des Wertes eines komplexen Polynoms

$$p1 + i \cdot p2 = \sum_{k=0}^{n} (A1_k + i \cdot A2_k) \cdot (u + i \cdot v)^k$$

in folgender Weise formuliert werden:

```
3. procedure polyK(A1, A2, n, u, v, p1, p2);
        value n, u, v;  array A1, A2;  integer n;
        real u,v, p1, p2;
        begin procedure add(a1, a2, b1, b2, r1, r2);
              real a1, a2, b1, b2, r1, r2;
              begin r1 := a1 + b1;  r2 := a2 + b2 end;
              procedure mult(a1, a2, b1, b2, r1, r2);
              real a1, a2, b1, b2, r1, r2;
              begin real t;  t := a1 × b1 − a2 × b2;
                    r2 := a2 × b1 + a1 × b2;  r1 := t end;
              integer k;  p1 := p2 := 0;
              for k := n step − 1 until 0 do
              begin mult (p1, p2, u, v, p1, p2);
              add(A1[k], A2[k], p1, p2, p1, p2)
              end
        end
```

Im Gegensatz zu Funktionserklärungen sind bei Unterprogrammerklärungen die Resultate im allgemeinen in der Liste formaler Parameter aufzuführen. Während eine Funktion in ALGOL 60 stets eine einwertige Funktion ist, können bei Unterprogrammen mehrere Resultate auftreten. Auch Funktionen, bei denen Felder als Resultate auftreten, sind im Sinne von ALGOL 60 keine Funktionen, sondern Unterprogramme. Diese gesonderte Anführung der Resultate bedingt manchmal eine etwas langwierige Schreibweise. Das dritte Beispiel zeigt, wie innerhalb des Prozedurrumpfes, der ein Block ist, wieder Prozedurerklärungen vorkommen können. Es zeigt weiter die Möglichkeit, in verschiedenen Prozedurerklärungen gleichartig bezeichnete formale Parameter zu verwenden, die keinerlei Beziehung zueinander haben. Es sei weiter darauf hingewiesen, daß der gleiche aktuelle Parameter mehrfach bei Aufruf einer Prozedur vorkommen kann, z. B. $p1$ und $p2$ in *mult*($p1, p2, u, v, p1, p2$). Die Ausführung dieses Unterprogramms geschieht wie

```
begin real t;  t := p1 × u − p2 × v;
               p2 := p2 × u + p1 × v;  p1 := t end
```

Falls die zusätzliche lokale Variable t in *mult* gegenüber *Mult* nicht eingeführt würde, erhielte man nicht das gewünschte Produkt, denn *Mult*($p1, p2, u, v, p1, p2$) ergibt die Ausführung

```
begin p1 := p1 × u − p2 × v;
      p2 := p2 × u + p1 × v end,
```

in der bei der zweiten Anweisung die Variable $p1$ bereits einen neuen Wert hat.

Es sind in ALGOL 60 einige *Standardunterprogramme* vorgesehen, deren Aufruf in einem Programm erfolgen kann, ohne daß der zugehörige Algorithmus in einer Unterprogrammerklärung definiert wird. Der Grund für die Einführung dieser Standardunterprogramme ist, daß die Funktion dieser Algorithmen nicht mit den bisher behandelten ALGOL-Anweisungen beschreibbar ist [46]. Es handelt sich um Unterprogramme zur Beschreibung von Eingabe- und Ausgabevorgängen an Rechenautomaten und eine weitere Standardfunktion mit dem Namen *length* zur Festellung der Anzahl von Zeichen in einer Zeichenreihe.

Hier seien nur die Unterprogramme *inreal*, *outreal*, *inarray* und *outarray* erläutert. Jedes dieser Unterprogramme ist zum Aufruf mit zwei aktuellen Parametern zu verbinden. Der erste Parameter ist ein arithmetischer Ausdruck, dessen Wert eine natürliche Zahl sein muß, die einen numerierten Eingabe- oder Ausgabekanal des betreffenden Rechenautomaten bezeichnet. Bei *inreal* und *outreal* muß der zweite aktuelle Parameter der Name einer als **real** oder **integer** erklärten Variablen sein. Die Wirkung von *inreal* ist, daß eine auf dem Eingabemedium nach gewissen vom betreffenden Rechenautomaten abhängenden Vorschriften dargestellte reelle Zahl der Variablen als Wert zugewiesen wird. Das Eingabemedium muß mit dem durch den ersten Parameter bezeichneten Kanal verbunden sein und wird automatisch um eine Zahl weitergestellt. Die Wirkung von *outreal* besteht in der entsprechenden Ausgabe des Wertes der Variablen. Der zweite Parameter der Unterprogramme *inarray* und *outarray* ist Name eines Feldes. Durch *outarray* wird veranlaßt, daß der Wert des bezeichneten Feldes ausgegeben wird, dieser Wert ist eine geordnete Menge reeller Zahlen. Die Ordnung ist bei einem n-dimensionalen Feld dadurch gegeben, daß $a[k_1, k_2, \ldots, k_n]$ Vorgänger von $a[j_1, j_2, \ldots, j_n]$ ist, wenn der erste von einem Index j_p verschiedene Index $k_p < j_p$ ist. Sie entspricht also bei ein- und zweidimensionalen Feldern der üblichen Darstellung von Vektoren bzw. von aus Zeilenvektoren zusammengesetzten Matrizen. Durch *inarray* wird eine entsprechende Eingabe, d. h. eine Wertzuweisung an alle einzelnen Variablen des Feldes veranlaßt. Eine Erklärung des Feldes muß in jedem Fall vorangegangen sein. Die Struktur des Feldes ist nur durch die Erklärung bestimmt und nicht aus der Anordnung der Werte auf dem Eingabe- oder Ausgabemedium ersichtlich. Weitere Unterprogramme zur Eingabe und Ausgabe einzelner Zeichen sind mit *insymbol* und *outsymbol* bezeichnet (vgl. S. 42).

Ein Programm in ALGOL 60 ist ein Block oder eine zusammengesetzte Anweisung. Da alle vorkommenden Namen zu erklären sind, treten Programme in Form zusammengesetzter Anweisungen üblicherweise nicht auf. Programme unter Anwendung der Standardunterprogramme sind z. B.

1. **begin** *outreal*(1, 3.14159) **end**

Durch dieses Programm wird über Kanal 1 die Zahl 3.14159 ausgegeben.

2. **begin integer** n; *inreal*(2,n);
begin array $a, b, c[1 : n,1 : n]$; **integer** i, k, j;
inarray(2,a); *inarray*(2,b);
for $i := 1$ **step** 1 **until** n **do**
for $k := 1$ **step** 1 **until** n **do**
begin $c[i,k] := 0$;
for $j := 1$ **step** 1 **until** n **do**
$c[i,k] := c[i,k] + a[i,j] \times b[j,k]$
end; *outarray*(1,c)
end
end

Mit diesem Programm wird die Eingabe zweier (n,n)-Matrizen über Kanal 2 mit Elementen $a[i,j]$ und $b[j,k]$ sowie ihre Multiplikation mit dem Resultat $c[i,k]$ und dessen Ausgabe über Kanal 1 beschrieben. Auf dem Eingabemedium sind nacheinander die Zahl n sowie die Zahlen $a[1,1]$, $a[1,2]$, ..., $a[n,n]$, $b[1,1]$, $b[1,2]$, ..., $b[n,n]$ angeordnet.

5. Stufe: Verteiler

Außer Typerklärungen, Felderklärungen und Prozedurerklärungen sind in ALGOL 60 noch Verteilererklärungen vorgesehen. Durch eine Verteilererklärung wird einem Vektor von Zielausdrücken ein Name (Verteilername) zugeordnet. Ein Zielausdruck ist ein Ausdruck, dem als Wert eine Marke zugeordnet ist. Der einfachste Zielausdruck wird durch eine Marke dargestellt, jedoch ist auch ein Verteiler ein Zielausdruck. Ein Verteiler wird als Verteilername mit einem nachfolgenden in eckige Klammern eingeschlossenen positiven ganzzahligen arithmetischen Ausdruck notiert, ist also äußerlich nicht von einer einfach indizierten Variablen unterscheidbar. Der Wert eines Verteilers ist gleich dem Wert des Zielausdruckes, der die Komponente des Vektors in der Verteilererklärung ist, die dem Indexwert entspricht. Zur Darstellung von Verteilererklärungen wird das neue Grundsymbol **switch** (Verteiler) eingeführt, das Wertzuweisungszeichen := wird mit einer weiteren Bedeutung verwendet. Ein Beispiel für eine Verteilererklärung ist **switch** $V := A, B, C, D, E$. Kommt innerhalb des Gültigkeitsbereichs dieser Verteilererklärung (d. h. im Block, an dessen Anfang die Verteilererklärung steht, solange nicht der Name V in einem internen Block neu erklärt ist) ein Zielausdruck $V[i]$ vor, so wird dadurch die i-te Marke aus der Liste von Marken A, B, C, D, E

ausgewählt. $V[i]$ ist also nur für $i = 1, 2, \ldots, 5$ definiert. Eine Sprunganweisung, deren Zielausdruck ein nicht definierter Verteiler ist, wird wie eine leere Anweisung ausgeführt. In der Verteilerliste rechts des Zeichens := können beliebige Zielausdrücke stehen, die jeweils bei ihrer Auswahl durch den Verteiler ausgewertet werden. Ein Beispiel für den Einsatz eines Verteilers ist:

begin integer i, j; **switch** $S := A, B, S[j], A$;
A: *inreal*(1,i); *inreal*(1,j); **go to** $S[i + j]$;
B: *outreal*(2,$i - j$) **end**

Dieses Programm wird von den Werten des in den Eingabekanal 1 gelegten Eingabemediums gesteuert, für die nur die Zahlen 1 und 2 zugelassen werden. Ist das Eingabemedium z. B. ein Lochstreifen mit den Zahlen

2, 2, 2, 1, 1, 2,

so werden durch das Programm folgende Anweisungen ausgeführt[1]):

$i := 2$
$j := 2$
go to $S[4]$ = **go to** A
$i := 2$
$j := 1$
go to $S[3]$ = **go to** $S[j]$ = **go to** $S[1]$ = **go to** A
$i := 1$
$j := 2$
go to $S[3]$ = **go to** $S[j]$ = **go to** $S[2]$ = **go to** B
outreal(2, -1)

Es wird also eine Zahl -1 über Kanal 2 ausgegeben und das Programm beendet. Wenn auch dieses Programm keinen praktischen Nutzen hat, so zeigt es doch gut die Wirkungsweise des Verteilermechanismus; insbesondere die Möglichkeit des „rekursiven Aufrufs" des gleichen Verteilers, der bei Auswertung eines Zielausdrucks mehrfach dieselbe Verteilererklärung benutzen kann.

Es sei darauf hingewiesen, daß die in der Liste der Zielausdrücke einer Verteilererklärung vorkommenden Namen keine globalen Größen zu sein brauchen, wie es bei Felderklärungen innerhalb der Feldgrenzen erforderlich ist. In der Regel werden es Marken sein, die innerhalb des betreffenden Blocks Anweisungen markieren, also lokal sind. Ein Sonderfall tritt auf, wenn innerhalb des Blockes

[1]) Hierbei drückt das Gleicheitszeichen die Äquivalenz der nebeneinanderstehenden Anweisungen aus, es handelt sich nicht um das Relationszeichen aus ALGOL 60.

ein weiterer Block eingeschlossen ist, in dem eine Marke nochmals erklärt wird (d. h. eine Anweisung markiert). Ergibt sich diese Marke innerhalb dieses inneren Blockes als Resultat der Auswertung des im umfassenden Block erklärten Verteilers, so erfolgt der Sprung zur entsprechend markierten Anweisung im umfassenden Block.

Beispiel.
```
begin switch V := L, M;  integer x;
        L: inreal(x);
        M: begin integer y;  inreal(y);
                if x > y then go to V[x];
                L: outreal(y)
           end
end
```

Die Sprunganweisung **go to** $V[x]$ wird z. B. im Fall $x = 1$ und $y = 0$ als Sprung zur Anweisung „L: *inreal*(x)" ausgeführt, nicht zur Anweisung „L: *outreal*(y)", die bei $x \leqq y$ ausgeführt wird. Da in ALGOL 60 eine Sprunganweisung zu einem nicht definierten Wert eines Verteilers wie eine leere Anweisung behandelt wird, wird L: *outreal*(y) auch ausgeführt, wenn x weder den Wert 1 noch den Wert 2 hat.

6. Stufe: Ergänzungen

In ALGOL 60 sind weitere – seltener angewandte – Bildungen möglich. Typ- und Felderklärungen können durch ein vorgesetztes Zeichen **own** (eigen) erweitert werden. Die dadurch als **own** erklärten Namen für Variable und Felder erhalten die zusätzliche Eigenschaft, daß ihnen bei Eingang in einen vorher schon einmal abgearbeiteten Block der Wert zugeordnet wird, den sie beim letztmaligen Verlassen dieses Blocks hatten. Schwierigkeiten treten bei als **own** erklärten Feldern auf, wenn sich die Feldgrenzen, die jeweils beim Eintritt in den Block berechnet werden und dann bis zum Verlassen des Blocks gültig sind, ändern. Dadurch können auch bei als **own** erklärten Feldern nicht mit Werten belegte indizierte Variable auftreten. In vielen Übersetzungssystemen wird **own** nicht oder nur mit Einschränkungen verwendet.

Wie bereits erwähnt wurde, können zweiwertige Aussagen oder *Boolesche Ausdrücke* (auch als *logische Ausdrücke* bezeichnet) gebildet werden. Ihre Elemente sind einfache oder indizierte Variable, zweiwertige (Boolesche) Funktionen, Relationen oder eingeklammerte Boolesche Ausdrücke. Sie können die Werte **true** und **false** annehmen. Wie bei arithmetischen Ausdrücken gibt es spezielle Operationszeichen zur Verknüpfung von zwei Operanden und ein Operationszeichen $\neg$ (gesprochen „nicht") zur Anwendung auf einen Operanden. Diese letzte Operation

(Negation) hat den Effekt, als Resultat den negierten Wert des Operanden (**true** statt **false** bzw. **false** statt **true**) zu ergeben. Für die zweistelligen Operationen gibt die folgende Tabelle Auskunft über die Bildung des Resultatwertes:

Operationszeichen	Aussprache	Wert des ersten Operanden	Wert des zweiten Operanden	Wert des Resultates
$\wedge$	und	**false**	**false**	**false**
		false	**true**	**false**
		true	**false**	**false**
		true	**true**	**true**
$\vee$	oder	**false**	**false**	**false**
		false	**true**	**true**
		true	**false**	**true**
		true	**true**	**true**
$\supset$	impliziert	**false**	**false**	**true**
		false	**true**	**true**
		true	**false**	**false**
		true	**true**	**true**
$\equiv$	äquivalent	**false**	**false**	**true**
		false	**true**	**false**
		true	**false**	**false**
		true	**true**	**true**

Bei der Bildung Boolescher Ausdrücke aus den erwähnten Elementen sind wie bei arithmetischen Ausdrücken Vorrangregeln in der Verknüpfung von Operanden zu beachten. Bei Operationszeichen mit gleichem Rang geschieht die Verknüpfung weiter links stehender Operanden zuerst. Die Rangfolge ist

1. $\neg$; 2. $\wedge$; 3. $\vee$; 4. $\supset$; 5. $\equiv$.

Beispiele.

1. $\neg a \vee a \wedge b$

   Sei Wert(a) = **true**, Wert(b) = **false**. Die Auswertung ergibt
   Wert($\neg a$) = **false**
   Wert($a \wedge b$) = **false**
   Wert($\neg a \vee a \wedge b$) = **false**

Durch Einsetzen der anderen möglichen Wertekombinationen für a und b kann man sich überzeugen, daß dieser Ausdruck dem Ausdruck $a \supset b$ äquivalent ist.

2. $a \wedge b \wedge c \vee \neg a$

   Sei Wert(a) = **true**, Wert(b) = **false**, Wert(c) = **false**. Es folgt
   Wert($\neg a$) = **false**
   Wert($a \wedge b$) = **false**
   Wert($a \wedge b \wedge c$) = **false**
   Wert($a \wedge b \wedge c \vee \neg a$) = **false**

3. $1 \leqq x \wedge x \leqq 2$

   Hier treten als Elemente $1 \leqq x$ und $x \leqq 2$ auf. Der Ausdruck erhält nur den Wert **true**, wenn beide Elemente den Wert **true** haben, d. h. x im Intervall $1 \leqq x \leqq 2$ liegt. Derartige mehrstellige Relationen können demnach in ALGOL 60 mittels des Zeichens $\wedge$ in der angegebenen Weise geschrieben werden.

Ähnlich wie bedingte Anweisungen können in ALGOL 60 auch *bedingte Ausdrücke* gebildet werden. Die bisher verwendeten Ausdrücke werden als *einfache Ausdrücke* bezeichnet, es gibt einfache arithmetische Ausdrücke, einfache Boolesche Ausdrücke und einfache Zielausdrücke. Ist E ein einfacher Ausdruck, A ein beliebiger (einfacher oder bedingter) Ausdruck des gleichen Typs, B ein Boolescher Ausdruck, so ist

if B **then** E **else** A

ein bedingter Ausdruck dieses Typs. Als Typ wird dabei die Eigenschaft bezeichnet, ein arithmetischer, Boolescher oder Zielausdruck zu sein. Ein eingeklammerter Ausdruck zählt als einfacher Ausdruck.

Beispiele.

1. **if** $x < 1$ **then** $a + b$ **else** $a - b$
2. **if** $a \wedge b$ **then** (**if** $x < 1$ **then** $x - 1$ **else** $abs(x)$) **else** $abs(x) - 1$
3. **if if** $u < v$ **then** $a \wedge b$ **else** $a \vee b$ **then** $x < 1$ **else** $y < 1 \vee a$

Der erste Ausdruck ist ebenso wie der zweite ein arithmetischer Ausdruck, der dritte ein Boolescher Ausdruck. Analysiert man den dritten Ausdruck, so hat er die Form

if B **then** E **else** A

mit $B =$ **if** $u < v$ **then** $a \wedge b$ **else** $a \vee b$
$E = x < 1$
$A = y < 1 \wedge a$

Die Bewertung eines bedingten Ausdruckes geschieht so, daß zuerst der Boolesche Ausdruck B ausgewertet wird, er hat entweder den Wert **true** oder den Wert **false**. Ist sein Wert **true**, so hat der gesamte Ausdruck den Wert des Ausdrucks E hinter **then**, andernfalls den Wert des Ausdrucks A hinter **else**. Im dritten Beispiel wird der Wert von B entweder gleich dem Wert von $a \wedge b$ (falls $u < v$) oder gleich dem Wert von $a \vee b$ (falls $u \geqq v$). Der gesamte Ausdruck hat entweder den Wert $x < 1$ oder den Wert von $y < 1 \wedge a$, je nachdem, ob B den Wert **true** oder **false** hat.

Bedingte Ausdrücke können außer für arithmetische Ausdrücke in Relationen überall an Stelle einfacher Ausdrücke eingesetzt werden. Diese Möglichkeit kann in Prozeduren eventuell die Notation vereinfachen, da bedingte Ausdrücke auch aktuelle Parameter sein können.

Bei *Laufanweisungen* war bisher nur eine Form

for $V := a_1$ **step** a_2 **until** a_3 **do** A

behandelt worden. An Stelle des **step-until**-Elementes „a_1 **step** a_2 **until** a_3" kann eine ganze Liste von Laufelementen auftreten. Solche Laufelemente haben entweder die Form eines arithmetischen Ausdrucks, die bereits erwähnte Form oder die Form „a **while** b". Dabei ist a ein arithmetischer Ausdruck und b ein Boolescher Ausdruck. Das Zeichen **while** (solange) ist ein neu einzuführendes Grundsymbol.

Die Ausführung geschieht so, daß der Variablen V nacheinander die sich aus den einzelnen Laufelementen ergebenden Werte zugewiesen werden und jedesmal die Anweisung A ausgeführt wird. Ist ein Laufelement ein einzelner arithmetischer Ausdruck, so wird V der Wert dieses Ausdrucks zugewiesen und A unbedingt ausgeführt. Hat ein Laufelement die Form eines **step-until**-Elementes, so erfolgt die Wertzuweisung an V und die Ausführung von A in der oben bereits beschriebenen Art. Hat schließlich ein Laufelement die Form „a **while** b", so erhält V vor jeder Ausführung von A den Wert von a zugewiesen; b wird ausgewertet und A dann ausgeführt, wenn b den Wert **true** hat. Darauf wird der gesamte Vorgang wiederholt, sofern nicht durch eine im Innern von A vorkommende Sprunganweisung die Laufanweisung verlassen wird. Hat b den Wert **false**, so wird zum nächsten Element der Liste übergegangen, falls „a **while** b" nicht das letzte Element der Liste ist. Dann wird die gesamte Laufanweisung beendet. Ist die Laufanweisung dadurch beendet worden, daß sie über eine im Innern von A gelegene Sprunganweisung verlassen wird, so behält V den letzten zugewiesenen Wert. Wird jedoch die Laufanweisung nach Abarbeitung der gesamten Liste der Laufelemente beendet, so ist der Wert von V undefiniert.

Beispiel. **for** $x := 0$ **step** 0.001 **until** 1, 1.01 **step** 0.01 **until** 10,
$x + 0.1$ **while** $exp(-x) \geqq 0.0000000001$ **do**
outreal(2, $exp(-x)$)

Diese Anweisung beschreibt die Tabellierung von e^{-x} über Ausgabekanal 2 mit verschiedenen Schrittweiten (0.001 für $0 \leqq x \leqq 1$, 0.01 für $1 < x \leqq 10$ und 0.1 für $x > 10$). Die Ausführung wird abgebrochen, wenn $exp(-x) < 0.0000000001$ ist.

Um Zahlen verschiedener Größenordnung bequem schreiben zu können, wurde noch das Grundsymbol $_{10}$ zur Notierung von Zehnerpotenzen eingeführt. Eine Zahl $a \cdot 10^b$ kann damit in ALGOL 60 in der Form $a_{10}b$ geschrieben werden, b ist eine ganze Zahl mit oder ohne Vorzeichen, a eine beliebige Zahl mit oder ohne Dezimalpunkt. Auch die Form $_{10}b$ für die Zahl 10^b in der üblichen Schreibweise ist zulässig. Damit kann das oben angeführte letzte Laufelement auch in der Form

$x + 0.1$ **while** $exp(-x) \geqq {}_{10} - 10$

geschrieben werden.

Als aktuelle Parameter können in ALGOL 60 auch Zeichenreihen vorkommen. Um sie zu kennzeichnen, werden zwei verschiedene Apostrophe ‘ und ’ eingeführt, die Anfang und Ende der Zeichenreihe angeben. Zwischen ihnen steht eine beliebige Folge von Zeichen mit Ausnahme dieser beiden Apostrophe. Es ist auch möglich, darin statt eines Zeichens wieder eine Zeichenreihe zu schreiben.

Beispiele. ‘abc’
‘10,1 ␣ 10,2’
‘a‘BC’ 10‘1”

Bemerkung. Das ALGOL-Zeichen ␣ ist als Zeichen für einen Zwischenraum innerhalb von Zeichenreihen bestimmt. Es kann etwa bei der Ausgabe von Zahlen auf einem Druckstreifen für das Freilassen einer Zeile benutzt werden. Eine Bedeutung dieses Zeichens ist nicht festgelegt.

In den Standardunterprogrammen *insymbol* und *outsymbol* wird eine Zeichenreihe als aktueller Parameter benutzt. Diese Unterprogramme haben drei Parameter, der erste ist ein arithmetischer Ausdruck (mit natürlicher Zahl als Wert), der zweite eine Zeichenreihe. Durch den ersten Parameter wird ein Eingabe- bzw. Ausgabekanal ausgewählt. In *insymbol* ist der dritte Parameter eine Variable, in *outsymbol* ist er ein arithmetischer Ausdruck, der natürliche Zahlen als Werte annehmen kann. Die Wirkung von *insymbol* besteht darin, daß das nächste Zeichen

vom Eingabemedium gelesen und von links nach rechts mit den Zeichen verglichen wird, die in der Zeichenreihe als zweitem aktuellen Parameter enthalten sind. Stimmt es mit dem n-ten Zeichen überein, so wird der mit dem dritten Parameter bezeichneten Variablen die natürliche Zahl n als Wert zugewiesen. Kommt keine Übereinstimmung zustande, so wird der Variablen die Zahl 0 zugeordnet. Durch *outsymbol* wird das dem Wert (n) des dritten aktuellen Parameters entsprechende n-te Zeichen aus der Zeichenreihe (zweiter aktueller Parameter) herausgesucht und über den ausgewählten Ausgabekanal ausgegeben

Beispiele.

1. *insymbol*(1,'0123456789, + −' n)
   bewirkt Zuweisung einer der Zahlen 1 bis 13 an n; wird z. B. ein Zeichen + vom Eingabemedium gelesen, so wird n der Wert 12 zugeordnet.

2. **begin real** x; **integer** z; $z := 0$;
   for $x := 0$ **step** $_{10}-3$ **until** 1, 1.01 **step** 0.01 **until** 10,
   $x + 0.1$ **while** $exp(-x) \geqq {}_{10}-10$ **do**
   begin *outreal*(2, $exp(-x)$); $z := z + 1$;
   if $z = 10$ **then**
   begin *outsymbol*(2, '␣', 1); $z := 0$ **end**
   end
   end

   Dieses Programm beschreibt die gleiche Tabellierung wie oben mit Einschieben eines Zwischenraumes nach jedem zehnten Wert.

Es sei noch bemerkt, daß für nicht in ALGOL 60 vorgesehene Zeichen negative Zahlen als Codes zur Ein- und Ausgabe mit *insymbol* bzw. *outsymbol* entsprechend der verfügbaren Rechenanlage und ihrem Verwendungszweck festgelegt werden können.

In ALGOL 60-Programmen können noch erläuternde Texte nach bestimmten Vorschriften eingeschoben werden. Dazu ist das Grundsymbol **comment** (Bemerkung) eingeführt worden. Hinter jedem Semikolon oder **begin** kann dieses Zeichen mit einer nachfolgenden Zeichenfolge, die mit einem Semikolon endet, eingeschoben werden. Die Zeichenfolge selbst darf demnach kein Semikolon enthalten. Hinter einem Zeichen **end** kann ebenfalls ein erläuternder Text eingefügt werden, der kein Semikolon, **end** oder **else** enthalten darf. Eines dieser Zeichen muß auf diesen Text folgen, sofern das Programm nicht abgeschlossen ist. Im Parameterteil einer Prozedur kann jedes die Parameter trennende Komma durch)*B*:(ersetzt werden, wobei *B* eine beliebige Folge von Buchstaben ist. Diese Schreibweise dient

zur Erläuterung des darauf folgenden aktuellen oder formalen Parameters, z. B. *add*(*a*1,*a*2,*b*1,*b*2) *Resultat:* (*r*1,*r*2).

Werden ALGOL 60-Programme veröffentlicht, so geschieht das meist in Form einer Prozedurerklärung. Jedes Programm kann durch Vorsetzen eines Zeichens **procedure** und eines Namens zu einer parameterlosen Prozedur gemacht werden. Es ist jedoch zweckmäßig, gewisse Größen als Parameter einzuführen. Abschließend sei als Beispiel eine Funktionserklärung mit Erläuterungen angegeben:

```
real procedure cosinus(x);
    comment Die Funktion cosinus(x) entspricht der Standardfunktion cos(x),
            der abweichende Name ist gewählt, da Standardfunktionen nicht
            erklärt werden;
    value x; real x;
    begin comment Das Argument wird zunächst auf das Intervall 0 ≦ y ≦ 2 × pi
                  mit pi := 3.14159265 reduziert, es wird ausgenutzt, daß cos(x)
                  eine gerade Funktion ist;
        real y, z; integer q; switch V := A, B, B, A;
        y := abs(x);
        y := y − entier(y/6.28318531) × 6.28318531;
        q := entier(y/1.57079633) + 1;
        comment q gibt den Quadranten des Arguments y an, dieses wird im
                folgenden auf 0 ≦ y ≦ pi/8 reduziert. cos(y) wird durch
                Taylorsche Entwicklung 1 − y ↑ 2/2 + y ↑ 4/24 − y ↑ 6/720
                approximiert, durch zweimalige Anwendung von cos(2 × y)
                = 2 × cos(y) × cos(y) − 1 zurücktransformiert;
        if y > 3.14159265 then y := 6.28318531 − y;
        if y > 1.57079633 then y := 3.14159265 − y;
        y := y × y/16;
        comment Beginn des Hornerschen Schemas für Berechnung der Taylor-
                schen Entwicklung;
        z := 1/24 − y/720;
        z := z × y − 0.5; z := z × y + 1;
        comment Beginn der Rücktransformation;
        z := 2 × z × z − 1; z := 2 × z × z − 1;
        go to V[q]; comment Der Verteiler V dient zur Festlegung des Vor-
                            zeichens des Resultats;
    B: cosinus := −z; go to C;
    A: cosinus := z;
    C: end cosinus
```

Die wichtigsten Regeln zur Bildung von ALGOL 60-Programmen seien zum Abschluß zusammengestellt:

1. Ein ALGOL 60-Programm ist ein Block oder eine zusammengesetzte Anweisung, eingeschlossen in **begin** und **end**.
2. Am Anfang eines Blockes stehen Erklärungen zur Festlegung von Eigenschaften vorkommender Namen. Namen sind mit einem Buchstaben beginnende Zeichenfolgen aus Buchstaben und Ziffern. Sie dienen zur Bezeichnung von Variablen, Feldern, Marken, Prozeduren und Verteilern.
3. Der eigentliche Inhalt des Programms wird durch eine oder mehrere Anweisungen beschrieben, die im Block auf die Erklärungen folgen.
4. Anweisungen sind: Wertzuweisungen, Sprunganweisungen, leere Anweisungen, bedingte Anweisungen, Laufanweisungen, Unterprogrammanweisungen, zusammengesetzte Anweisungen und Blöcke.
5. Erklärungen gelten innerhalb eines Blockes. Da in einem Block ein weiterer Block eingeschlossen sein kann, ist eine Neuerklärung des gleichen Namens dort möglich. Nach Beendigung des eingeschlossenen Blocks gilt dann jedoch die alte Erklärung wieder, der Gültigkeitsbereich hat ein „Loch" (solche Konstruktionen sollten vermieden werden). In einem Block erklärte Namen heißen bezüglich des Blocks lokal. Ein Name darf nicht mehrfach im gleichen Blockkopf (Erklärungsteil des Blocks) erklärt werden.
6. Erklärungen sind Typerklärungen der Form **real** L, **integer** L oder **Boolean** L; Felderklärungen der Form **array** $L_1[G_1]$, $L_2[G_2]$, ..., $L_n[G_n]$, in denen statt **array** auch **integer array** oder **Boolean array** stehen kann; Prozedurerklärungen und Verteilererklärungen. L bzw. L_i bezeichnen Listen von durch Komma getrennten Namen, die G sind Listen von ebenfalls durch Komma getrennten Grenzenpaaren. Ein Grenzenpaar besteht aus zwei ganzzahligen arithmetischen Ausdrücken, die Grenzen für den entsprechenden Index einer zugeordneten indizierten Variablen angeben. Diese arithmetischen Ausdrücke dürfen nicht mit lokalen Variablen oder Funktionen gebildet werden. Eine indizierte Variable hat den Namen eines Feldes, gefolgt von einer Liste von Indizes, die in eckige Klammern eingeschlossen sind; sie bezeichnet eine Komponente eines Feldes. Prozedurerklärungen beginnen mit **procedure** (Unterprogrammerklärungen) oder **real procedure**, **integer procedure** bzw. **Boolean procedure** (Funktionserklärungen) und einem Namen. Auf den Namen folgt eine Liste von in Klammern eingeschlossenen verschiedenen Namen (formale Parameter). Diese Liste kann auch entfallen. Der Prozedurrumpf ist eine Anweisung – meist ein Block – zur Definition des Unterprogramms (der Funktion). Darin

vorkommende formale Parameter werden bei Ausführung durch aktuelle Parameter ersetzt, wenn sie nicht in der Prozedurerklärung als Parameter mit Anfangswert definiert werden (durch Aufnahme in eine Liste hinter dem Zeichen **value**, sie sind dann auch zu spezifizieren). Diese Parameter erhalten nur zu Beginn der Ausführung den Wert des betreffenden aktuellen Parameters zugeordnet. Die für formale Parameter benutzten Namen dürfen mit Namen anderer im Programm – aber nicht im Prozedurrumpf – verwendeter Größen übereinstimmen. Verteilererklärungen haben die Form **switch** $V := L$, wobei V ein Name und L eine Liste von Zielausdrücken ist.

7. Wertzuweisungen haben die Form $V_1 := V_2 := \cdots := V_n := E$. Dabei sind V_1 bis V_n einfache (als **real**, **integer** oder **Boolean** erklärte) Variable oder indizierte Variable (Komponenten von Feldern) gleichen Typs. E ist ein arithmetischer oder Boolescher Ausdruck, der mit geringfügigen Einschränkungen in der üblichen mathematischen Notation niedergeschrieben ist. Für V_i kann im Inneren einer Funktionserklärung auch der Funktionsname stehen. Durch die Wertzuweisung wird den linksstehenden Größen der Wert des rechtsstehenden Ausdrucks zugewiesen (bei Booleschen Ausdrücken müssen links als **Boolean** oder **Boolean procedure** erklärte Namen stehen).

8. Sprunganweisungen haben die Form **go to** D, wobei D eine Marke oder ein Zielausdruck ist. Sie veranlassen, daß das Programm bei der entsprechend markierten Anweisung fortgesetzt wird. Ist $D = V[i]$ (Verteiler), so wird der i-te Zielausdruck aus der entsprechenden Verteilererklärung verwendet.

9. Anweisungen können durch Vorsetzen von einer oder mehreren Marken (Namen oder ganze Zahlen) mit jeweils einem Doppelpunkt markiert werden.

10. Bedingte Anweisungen der Form **if** B **then** A dienen zur bedingten Ausführung der unbedingten Anweisung A (wenn der Boolesche Ausdruck B den Wert **true** hat). Bedingte Anweisungen der Form **if** B **then** A_1 **else** A_2 dienen je nach dem Wert von B zur Auswahl zwischen der unbedingten Anweisung A_1 und der Anweisung A_2 (letztere, wenn B den Wert **false** hat).

11. Laufanweisungen (meist in der Form **for** $V := a_1$ **step** a_2 **until** a_3 **do** A) dienen zur mehrfachen Ausführung einer Anweisung A. Eine Laufvariable V nimmt dabei verschiedene Werte an.

12. Unterprogrammanweisungen rufen Unterprogramme über ihren Namen auf. Eine Liste aktueller Parameter, die in Klammern eingeschlossen sind, enthält Informationen über die spezielle Ausführung (vgl. 6. und 16.).

13. Zusammengesetzte Anweisungen unterscheiden sich von Blöcken dadurch, daß in ihnen keine Erklärungen vorkommen. In Blöcken auftretende Marken

(an markierten Anweisungen) gelten als lokale Größen, ein Sprung in das Innere eines Blocks ist daher nicht möglich, in das Innere einer zusammengesetzten Anweisung kann gesprungen werden.

14. Leere Anweisungen bestehen aus der leeren Zeichenreihe, sie dienen zur Anbringung von Marken, z. B. am Ende eines Blocks.
15. Als Trennzeichen zwischen Erklärungen, Anweisungen, Spezifikationen und Wertlisten (Liste hinter **value**) dient das Semikolon.
16. Es empfiehlt sich, in Prozedurerklärungen (vgl. 6.) auftretende formale Parameter zu spezifizieren. Eine Spezifikation besteht aus einem Spezifikator, gefolgt von einer Liste formaler Parameter. Ein formaler Parameter darf höchstens einmal spezifiziert werden, Parameter mit Anfangswert müssen spezifiziert werden. Als Spezifikatoren können auftreten: **real**, **integer**, **Boolean**, **array**, **real array**, **integer array**, **Boolean array**, **procedure**, **real procedure**, **integer procedure**, **Boolean procedure**, **label**, **switch** und **string**. Spezifikationen erläutern die Art der betreffenden formalen Parameter. Die beim Aufruf einer Prozedur vorkommenden aktuellen Parameter sollen die durch Spezifikation zugeordneter formaler Parameter angegebenen Eigenschaften haben. Spezifikationen sind keine Erklärungen, ihre Wirkung ist bei Parametern mit Anfangswert der einer entsprechenden Erklärung gleich. Feldgrenzen werden dabei gegebenenfalls vom aktuellen Parameter übernommen. Spezifikationen stehen in der Prozedurerklärung direkt vor dem Prozedurrumpf.
17. Im Zweifelsfall wähle man statt einer eleganten Notation lieber eine, deren Eindeutigkeit klar ist, auch wenn sie umständlicher erscheint.

§ 2. Anwendung von ALGOL

Soll ein in ALGOL geschriebenes Programm auf einem dafür geeigneten Rechenautomaten bearbeitet werden, so muß es in einer wohlbestimmten Form eingabefertig vorbereitet werden. Im allgemeinen wird das Programm auf Lochstreifen oder Lochkarten umgesetzt und in dieser Form in die Maschine eingegeben, übersetzt und abgearbeitet. Entsprechend sind die einzelnen ALGOL-Zeichen auf dem Eingabemedium durch Codes dargestellt, die normalerweise aus einem einzelnen Eingabezeichen bestehen, jedoch in Sonderfällen auch mehrere Eingabezeichen umfassen können.

Aus praktischen Erwägungen ist es zweckmäßig, zur Herstellung eines eingabefertigen ALGOL-Programms eine Schreibmaschine zu benutzen, da man dann Tippfehler sofort erkennt. Die Schreibmaschine ist mit einer Zusatzeinrich-

tung versehen, die automatisch einen Lochstreifen herstellt, in den mit jedem Tastenanschlag eine Lochkombination gestanzt wird. Vielfach verwendet werden Fernschreibmaschinen, die einen international genormten Lochstreifencode haben, den Fünfkanal-Telegraphen-Code. Nachteilig ist, daß mit 5 Kanälen nur $2^5 = 32$ Kombinationen erzeugbar sind, während das ALGOL-Alphabet 116 Zeichen umfaßt. Daher sind Lochstreifen mit 7 oder 8 Kanälen besser geeignet, die ebenfalls mit speziellen Schreibmaschinen hergestellt werden können. Auch mit derartigen Lochstreifen werden ALGOL-Zeichen wie **begin** und **procedure** nicht durch ein einziges Eingabezeichen, sondern durch eine Folge von Eingabezeichen dargestellt, da jedem Tastenanschlag ein Zeichen entspricht und das Schriftbild mehrere Anschläge erfordert. Schreibmaschinen, die für jedes ALGOL-Zeichen nur ein Eingabezeichen auf einem Lochstreifen erzeugen, gibt es bisher nicht. Die Erzeugung von Lochkarten mit Hilfe eines Schreiblochers geschieht im Prinzip ebenso wie die Herstellung eines Lochstreifens. Der Vorteil des Lochstreifens gegenüber der Lochkarte liegt im geringeren Platzbedarf bei der Aufbewahrung, Lochkarten bieten dagegen die Möglichkeit, Programmteile in beliebiger Reihenfolge zu Programmen zusammenstellen zu können.

Es können nicht beliebige ALGOL-Programme zur Übersetzung und Bearbeitung auf einen gegebenen Rechenautomaten zugelassen werden. Selbstverständlich sind Beschränkungen durch die Speicherkapazität des Rechenautomaten, die die Anzahl der Variablen, den Umfang der Felder und die Länge des ALGOL-Programms begrenzen. Darüber hinaus sind in den meisten Übersetzungssystemen Einschränkungen in den zulässigen Konstruktionen von ALGOL-Programmen vorgesehen, z. B. sind als **own** erklärte Felder nicht zulässig, oder der rekursive Aufruf von Prozeduren ist nicht gestattet usw. Zu jedem Übersetzungssystem sind diese Einschränkungen ebenso wie die Beschränkungen durch die Speicherkapazität dem Benutzer mitzuteilen. Eine gegenüber ALGOL 60 eingeschränkte Sprache ist die Sprache IFIP SUBSET ALGOL 60. Ihre Einschränkungen sind auch in der sogenannten ALCOR-Konvention enthalten [2], [27]. ALCOR ist eine Abkürzung für ALGOL-Converter. Diese ALCOR-Konvention enthält neben Einschränkungen der Sprache ALGOL 60 auch einen Eingabecode, der der Verwendung von Fünfkanallochstreifen besonders angepaßt ist und sich nur wenig vom internationalen Telegraphencode Nr. 2 unterscheidet, der im Fernschreibverkehr benutzt wird. Es ist auch möglich, in einem speziellen ALCOR-Code gelochte Lochkarten zur Eingabe zu verwenden, sofern die Eigenschaften des betreffenden Rechenautomaten das zulassen. Die ALCOR-Gruppe, die ca. 40 Institutionen als Mitglieder umfaßt, benutzt Übersetzungssysteme, die auf den ALCOR-Konventionen basieren. Obwohl die einzelnen Mitglieder sehr verschiedenartige Rechenautomaten verwenden, sind die Programme überall ver-

wendbar, wenn gewisse Verschiedenheiten in Zahlendarstellung, Eingabe und Ausgabe berücksichtigt werden.

Die wesentlichsten Einschränkungen der Sprache IFIP SUBSET ALGOL 60 gegenüber ALGOL 60 seien im folgenden angegeben. Für eine vollständige Darstellung sei auf [2] oder [45] verwiesen:

1. Erklärungen mit **own** sind nicht zugelassen.
2. Ein rekursiver Aufruf von Prozeduren ist nicht möglich.
3. Als Marken sind nur Namen (keine ganzen Zahlen) erlaubt.
4. Alle formalen Parameter sind zu spezifizieren.
5. Eine Sprunganweisung zu einem undefinierten Verteiler wird nicht wie eine leere Anweisung behandelt, sondern ist undefiniert (ein Programm mit einer solchen Anweisung ist demnach nicht korrekt). Es sind nur einfache, nicht eingeklammerte Zielausdrücke zugelassen.
6. Potenzierung ist nicht mit ganzzahliger Basis und negativem ganzzahligen Exponenten möglich.
7. Eine Laufvariable muß eine einfache Variable sein.
8. In einer Verteilerliste dürfen nur Marken auftreten.
9. Als aktuelle Parameter, die symbolische Parameter ersetzen, können nur Namen oder Zeichenreihen benutzt werden, also keine Ausdrücke.
10. Das Alphabet enthält nur kleine Buchstaben, das Zeichen ÷ für ganzzahlige Divisionen ist weggelassen, entsprechend ist diese Operation nicht zugelassen.
11. Namen, die in den ersten 6 Zeichen übereinstimmen, werden als gleich angesehen.

Nachfolgend seien die im ALCOR-Code benutzten Eingabezeichen einschließlich der zugeordneten Lochkombination auf dem Fünfkanallochstreifen angegeben. Mit Ausnahme der sogenannten Betriebszeichen sind jeder Lochkombination zwei Zeichen zugeordnet, ein Buchstabe und ein anderes Zeichen. Welches Zeichen gemeint ist, wird dadurch festgelegt, daß die Fernschreibmaschine entweder durch das Betriebszeichen „.Buchstabenumschaltung" in den Zustand „buchstabenseitig" oder durch das Betriebszeichen „Ziffernumschaltung" in den Zustand „ziffernseitig" versetzt wird. Dieses Verfahren ist dasselbe wie bei jeder handelsüblichen Schreibmaschine für das Schreiben kleiner oder großer Buchstaben mit der gleichen Taste durch Betätigung der Umschalttaste. Weitere Betriebszeichen sind „Wagenrücklauf", „Zeilenvorschub" und „Zwischenraum" Auf dem 17,5 mm breiten Lochstreifen ist außer den fünf das Zeichen bestimmenden Spuren noch eine Taktspur vorhanden. Für ein zu stanzendes Zeichen ist eine

Zeile vorgesehen, deren Lage durch ein Loch in der Taktspur zwischen der zweiten und dritten Zeichenspur fixiert ist.

Die Zuordnung zwischen Lochkombinationen und Zeichen ist:

gelochte Spuren	Zeichen buchstabenseitig	Zeichen ziffernseitig
1 – 2	a	–
1 – 4 – 5	b	×
– 2 – 3 – 4	c	:
1 – 4	d	(unzulässig)
1	e	3
1 – 3 – 4	f	[
– 2 – 4 – 5	g	]
3 – 5	h	10
– 2 – 3	i	8
1 – 2 – 4	j	;
1 – 2 – 3 – 4	k	(
– 2 – 5	l	)
– 3 – 4 – 5	m	
– 3 – 4	n	,
– 4 – 5	o	9
2 – 3 – 5	p	0
1 – 2 – 3 – 5	q	1
– 2 – 4	r	4
1 – 3	s	'
– 5	t	5
1 – 2 – 3	u	7
– 2 – 3 – 4 – 5	v	=
1 – 2 – 5	w	2
1 – 3 – 4 – 5	x	/
1 – 3 – 5	y	6
1 – 5	z	+
– 4	„Wagenrücklauf“	
– 2	„Zeilenvorschub“	
1 – 2 – 3 – 4 – 5	„Buchstabenumschaltung“	
1 – 2 – 4 – 5	„Ziffernumschaltung“	
– 3	„Zwischenraum“	

Die Lochkombination „1–4“ ist ziffernseitig unzulässig, da diese bei Übermittlung des Lochstreifens über einen Fernschreibkanal den Namensgeber des Emp-

fängers auslöst. Die „leere Lochkombination", bei der nur das Taktloch gelocht ist, ist ebenfalls unzulässig. Abb. 1 zeigt ein Stück eines Lochstreifens.

Für gewisse ALGOL-Zeichen ist eine Darstellung durch mehrere Zeichen im ALCOR-Code vorgesehen. Die Zuordnung ist folgende:

ALGOL-Zeichen	ALCOR-Code	ALGOL-Zeichen	ALCOR-Code
↑	'power'	∧	'and'
<	'less'	¬	'not'
≦	'not greater'	:=	:=
=	'equal'	␣	„Zwischenraum"
≧	'not less'	‘	'('
>	'greater'		')'
≠	'not equal'	**begin**	'begin'
≡	'equiv'	**end**	'end'
⊃	'impl'	**go to**	'go to'
∨	'or'	usw.	

Die in ALGOL durch unterstrichene englische Worte dargestellten Zeichen werden im ALCOR-Code durch entsprechende in Apostroph eingeschlossene Worte dargestellt (Wortsymbole). Erlaubt das zur Herstellung benutzte Gerät nur das Schreiben großer Buchstaben, so repräsentieren diese die kleinen.

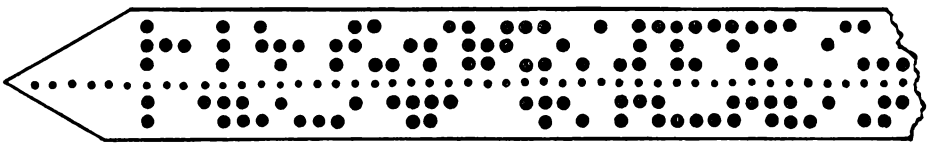

Abb. 1. Lochstreifen

Ein in ALGOL vorliegendes Programm ist vor seiner Anwendung an einer Rechenanlage zu überprüfen, ob es die für die Rechenanlage bestehenden Einschränkungen – z. B. die der ALCOR-Konvention – erfüllt. Danach wird es in eingabefertiger Form auf Lochstreifen oder Lochkarten gestanzt und nochmals geprüft. Die Rechenanlage stellt daraus mittels eines besonderen Leseprogramms und eines Übersetzungsprogramms ein Maschinenprogramm her, das dann im Speicher der Maschine zur anschließenden Rechnung bereitsteht oder auch ausgegeben wird. Bei Maschinen mit kleiner Speicherkapazität ist meist nur die zweite Möglichkeit vorgesehen, das ausgegebene Programm ist dann vor Beginn der eigentlichen Rechnung wieder einzugeben. Trotz sorgfältiger Prüfung können in

einem Programm noch Fehler enthalten sein. Vom Übersetzungsprogramm wird dann eine Fehlermeldung gegeben. Mit Hilfe eines speziellen Korrekturprogramms läßt sich ein derartiger Fehler häufig sofort beseitigen. Oft wird auch nach entdeckten Fehlern in der Übersetzung fortgefahren, um weitere Fehler ebenfalls zu finden. Dabei handelt es sich jedoch nur um Verletzungen der gegebenen Regeln, nicht um inhaltliche Fehler im Rechenverfahren.

Für die Eingabe von Daten und die Ausgabe von Resultaten sind häufig Standardprozeduren vorgesehen, die von den oben erläuterten Standardprozeduren *inreal*, *inarray*, *insymbol*, *outreal*, *outarray* und *outsymbol* abweichen. In der ALCOR-Konvention werden z. B. *read*(V) für die Bewertung der Variablen V durch Eingabe und *print*(V) für das Ausdrucken des Wertes von V benutzt.

Umfangreiche Programme setzen sich meist aus Teilprogrammen zusammen. ALGOL 60 bietet die Möglichkeit, solche Teilprogramme mit einem Namen zu versehen und in der Form einer Unterprogrammerklärung in ein größeres Programm einzufügen. Die Abarbeitung eines solchen Unterprogramms wird durch einen Unterprogrammaufruf veranlaßt. Hat man eine Anzahl dieser ALGOL-Prozeduren in einer Unterprogrammbibliothek gesammelt und liegen sie dort eingabefertig vor, so besteht die Programmierungsarbeit im wesentlichen im Schreiben eines Rahmenprogramms. Bei der Eingabe sind die verschiedenen Programmteile zusammenzufügen, was bei auf Karten gelochten Programmen einfach durch Aufeinanderlegen der verschiedenen Kartenstöße erfolgen kann. Bei Lochstreifeneingabe sind die einzelnen Teile entweder zusammenzukleben, einzeln nacheinander einzugeben oder in einer besonderen Kopiereinrichtung auf einen gemeinsamen Streifen zu kopieren. In Unterprogrammen kommen häufig globale Größen vor. Es ist dann dafür zu sorgen, daß im Rahmenprogramm diese globalen Größen erklärt werden und gegebenenfalls auch die notwendigen Bewertungen vor Eintritt in das Unterprogramm erfolgen. Solche Größen können auch als Parameter in das Unterprogramm eingeführt werden.

Als Beispiel sei ein ALGOL-Programm zur Bestimmung aller Nullstellen eines Polynoms mit reellen Koeffizienten $P(z) = \sum_{k=0}^{m} c_k \cdot z^k$ entwickelt. Als Verfahren werde ein Verfahren stärksten Abstiegs benutzt, das eine Modifikation des Newtonschen Verfahrens darstellt [23]. Zunächst wird ein Hilfsprogramm entwickelt, das die Bestimmung des Funktionswertes eines Polynoms $f(z) = \sum_{k=0}^{n} a_k \cdot z^k$ erlaubt. Es ist zweckmäßig, z nicht von vornherein als komplex anzunehmen, sondern zwei verschiedene Verfahren für reelles z und komplexes z zu verwenden. Eine Boolesche Variable „*komplex*“ soll den Wert **true** annehmen, wenn mit z als komplexer Zahl gerechnet wird. Wird mit der reellen Zahl $z = x$ gerechnet,

so habe *komplex* den Wert **false**. Diese zweite Variante kann mit dem üblichen Hornerschen Schema ausgeführt werden, für das in § 1 eine **real procedure** *poly(a,n,x)* erklärt wurde. Für die erste Variante kann ebenfalls ein abgewandeltes Hornersches Schema nach COLLATZ [31] benutzt werden. Es beruht auf der Division $f(z) : (z^2 - p{\cdot}z - q)$, wenn $f(u + i{\cdot}v)$ zu berechnen ist und $p = 2{\cdot}u$, $q = -(u^2 + v^2)$ gesetzt wird. Man erhält

$$f(z) = g(z) \cdot (z^2 - p{\cdot}z - q) + a_1'{\cdot}z + a_0'.$$

Es folgt für $z = u + i{\cdot}v$

$$\begin{aligned} f(u + i{\cdot}v) = g(u + i{\cdot}v) \cdot (u^2 - v^2 - p{\cdot}u + u^2 + v^2 + i \cdot [2{\cdot}u{\cdot}v - p{\cdot}v]) \\ + a_1'{\cdot}(u + i{\cdot}v) + a_0' = a_1'{\cdot}u + a_0' + i{\cdot}a_1'{\cdot}v = Rf + i{\cdot}If. \end{aligned}$$

Zur Durchführung der Division kann ein Rechenschema benutzt werden:

		a_n	a_{n-1}	a_1	a_0
			pA_n	pA_2	pA_1
			qA_{n+1}	qA_3	qA_2
$A_{n+1} = 0$		$A_n = a_n$	A_{n-1}	A_1	A_0

Jedes A_i $(i = 0, \ldots, n-1)$ ist Summe der darüberstehenden reellen Zahlen. Es gilt $g(z) = \sum_{k=2}^{n} A_k \cdot z^{k-2}$, $a_1' = A_1$, $a_0' = A_0 - p{\cdot}A_1$. Daraus folgt

$$\begin{aligned} Rf &= A_1 \cdot (u - p) + A_0 = A_0 - A_1 \cdot u; \\ If &= A_1{\cdot}v. \end{aligned}$$

Ein ALGOL-Unterprogramm für diese Rechnung kommt mit drei Hilfsvariablen und einer Laufvariablen aus:

```
procedure kompoly(a, n, u, v, Rf, If); value n, u, v; array a;
        integer n; real u, v, Rf, If;
        begin integer k; real t1, t2, t3, p, q;
              t1 := 0; t2 := a[n]; p := 2 × u; q := -(u × u + v × v);
              for k := n - 1 step -1 until 0 do
                  begin t3 := a[k] + p × t2 + q × t1;
                        t1 := t2; t2 := t3
                  end;
                  Rf := t2 - t1 × u; If := t1 × v
        end
```

Soll nach Bestimmung einer reellen Nullstelle z_1 oder eines Paares konjugierter komplexer Nullstellen $z_1 = x_1 + i{\cdot}y_1$, $z_2 = x_1 - i{\cdot}y_1$ das Polynom durch

Division durch $z - z_1$ bzw. $(z - z_1) \cdot (z - z_2)$ auf ein Polynom $(n - 1)$-ten bzw. $(n - 2)$-ten Grades reduziert werden, so ist es zweckmäßig, die Prozeduren für die Ausführung des Hornerschen Schemas sowohl im reellen als auch im komplexen Fall abzuändern. Die Koeffizienten des Polynoms $g(z)$ sind ja im komplexen Fall gerade die Koeffizienten des gesuchten Polynoms, entsprechend wird auch im reellen Fall im Hornerschen Schema eine Division durch $z - x$ ausgeführt. Das Rechenschema hat die Gestalt

	a_n	a_{n-1}	a_1	a_0
		$A_n x$	$A_2 x$	$A_1 x$
$A_n = a_n$		A_{n-1}	A_1	A_0

Es gilt $f(z) = g^*(z) \cdot (z - x) + A_0$ mit $g^*(z) = \sum_{k=1}^{n} A_k \cdot z^{k-1}$. Für $A_0 = 0$ geht die Division ohne Rest auf, x ist Nullstelle von $f(z)$. Beide Rechenschemata können in einem Unterprogramm zusammengefaßt werden, in dem die Variable „*komplex*" eine globale Größe ist.

```
procedure Hornerschema(a, n, u, v) Resultat :( Rf, If, A);
        value n, u, v;  array a, A;  integer n;  real u, v, Rf, If;
        begin integer k;  real p, q;
            if komplex
            then begin A[n + 1] := 0;  A[n] := a[n];  p := 2 × u;
                       q := -(u × u + v × v);
                       for k := n - 1 step - 1 until 0 do
                       A[k] := a[k] + p × A[k + 1] + q × A[k + 2];
                       Rf := A[0] - A[1] × u;  If := A[1] × v
                 end
            else begin A[n] := a[n];
                       for k := n - 1 step -1 until 0 do
                       A[k] := a[k] + u × A[k + 1];
                       Rf := A[0];  If := 0
                 end
        end
```

Das Programm soll so geschrieben werden, daß eine Nullstelle eines Polynoms $\sum_{k=0}^{r} b_k \cdot z^k = P_r(z)$ bestimmt wird und danach dieses Polynom in ein Produkt aus einem linearen bzw. quadratischen Faktor und einem Faktor $(n - 1)$-ten bzw. $(n - 2)$-ten Grades zerlegt wird, der dann $P_r(z)$ ersetzt. Anfangs gilt $r = m$ und

$b_k = c_k$ $(k = 0, \ldots, m)$. Weiter sei noch $c_m = 1$ vorausgesetzt, um den Sonderfall $c_m = 0$ von vornherein auszuschließen. In einem Teil 1 des Programms wird die Ableitung $P_r'(z) = \sum_{k=0}^{r-1} (k+1) \cdot b_{k+1} \cdot z^k$ gebildet. Dieser und die folgenden Teile seien zunächst ohne Berücksichtigung von Erklärungen notiert:

Teil 1: **for** $k := 0$ **step** 1 **until** $r - 1$ **do** $B[k] := (k + 1) \times b[k + 1]$;

In *Teil* 2 wird mit der Durchführung des Newtonschen Verfahrens begonnen, dabei wird von einem gegebenen Anfangswert $z0 = x0 + i \cdot y0$ ausgegangen:

Teil 2: *Hornerschema*(*b*, *r*, *x*0, *y*0, *Rf*, *If*, *A*);

$M := Rf \times Rf + If \times If$;

Teil 21: **if** $M <$ *delta* **then go to** *Kontrolle*;

Hornerschema(*B*, $r - 1$, *x*0, *y*0, *Rfa*, *Ifa*, *A*);
$N := Rfa \times Rfa + Ifa \times Ifa$;
if $N > M \times$ *epsilon*
then begin if *komplex*
then begin $x1 := x0 - (Rf \times Rfa + If \times Ifa)/N$;
$y1 := y0 - (If \times Rfa - Rf \times Ifa)/N$
end
else begin $x1 := x0 - Rf/Rfa$; $y1 := y0$ **end**
end
else begin $x0 := x0 + d$; **go to** *Teil* 2 **end**;

Bemerkung. Ist $M = |P_r(z_0)|^2 < \delta$, so wird z_0 als genügend gute Näherung für eine Nullstelle angesehen. Diese wird im Programmteil *Kontrolle* nochmals geprüft. Ist $N = |P_r'(z_0)|^2 > \varepsilon \cdot M$, so wird ein Schritt des Newtonschen Verfahrens ausgeführt; es ist dadurch wegen $M \geqq \delta > 0$ die Ausführbarkeit der Division $P_r(z_0)/P_r'(z_0)$ gesichert, und es gilt $|P_r(z_0)/P_r'(z_0)| < \sqrt{1/\varepsilon}$. Ist die Bedingung für N nicht erfüllt, so wird $x0$ um eine gegebene Konstante d abgeändert und von vorn begonnen. Wird ε genügend klein gewählt, so ist die Bedingung $N > \varepsilon \cdot M$ höchstens in einander fremden Umgebungen der Nullstellen von P_r' verletzt.

In *Teil* 3 des Programms wird geprüft, ob $|P_r(z_1)| < 0.8\ |P_r(z_0)|$ ist. Ist das der Fall, so wird das Newtonsche Verfahren fortgesetzt. Andernfalls wird die Newtonsche Korrektur so lange halbiert, bis ein absolut kleinerer Funktionswert als $|P_r(z_0)|$ gefunden wird. Sollte das nach z. B. 100 Schritten noch nicht geschehen sein, so wird zu einem Programmteil „Sonderfall" gegangen.

```
Teil 3: ZV := 0;
        Hornerschema(b, r, x1, y1, Rf, If, A);
        M1 := Rf × Rf + If × If;
if M1 < 0.64 × M
then Teil 31: begin x0 := x1;  y0 := y1;  M := M1;  go to Teil 21 end
else Halbierung: begin ZV := ZV + 1;  x1 := 0.5 × (x0 + x1);
                       y1 := 0.5 × (y0 + y1);
                       Hornerschema(b, r, x1, y1, Rf, If, A);
                       M1 := Rf × Rf + If × If;
                       if M1 < M
                       then begin if M1 < 0.64 × M
                                  then go to Teil 31
                                  else go to Teil 4
                            end
                       else begin if ZV < 100
                                  then go to Halbierung
                                  else go to Sonderfall
                            end
                  end;
```

Der vierte Programmteil enthält eine weitere Änderung gegenüber dem ursprünglichen Newtonschen Verfahren, wenn $|P_r(z_1)| < 0.8\,|P_r(z_0)|$ nicht, aber $|P_r(z_1)| < |P_r(z_0)|$ erreicht wurde, wie es in der Umgebung einer Nullstelle von $P_r'(z)$ auftreten kann. Es wird dann geprüft, ob für $z_2 = z_0 + 2i(z_1 - z_0)$ ein absolut kleinerer Funktionswert erhalten wird. Ist das der Fall, so wird das Verfahren mit diesem z_2 an Stelle von z_0 neu begonnen, sonst mit z_1.

```
Teil 4: x2 := x0 − 2 × (y1 − y0);  y2 := y0 + 2 × (x1 − x0);
        Merke := komplex;  komplex := true;
        Hornerschema(b, r, x2, y2, Rf, If, A);
        M2 := Rf × Rf + If × If;
    if M2 < M1
  then begin x0 := x2;  y0 := y2;  M := M2;  go to Teil 21 end
  else begin komplex := Merke;  x0 := x1;  y0 := y1 go to Teil 2 end;
```

Der Programmteil *Kontrolle* dient zur Prüfung der gefundenen Näherung z_0 durch Einsetzen in das Polynom $P(z)$. Außerdem werden die Koeffizienten b_k des nächsten Polynoms $P_r(z)$ fixiert, die gefundene Nullstelle bzw. das Paar konjugiert komplexer Nullstellen wird ausgegeben. Ist $|P(z_0)|^2 \geqq \delta$, so wird das gesamte Verfahren nochmals auf $P(z)$ statt auf $P_r(z)$ angewandt. Dabei wird einer

Booleschen Variablen „*Pruefung*" der Wert **true** erteilt, die Werte der A_k und r werden aufbewahrt, bis z_0 genügend korrigiert ist.

```
Kontrolle: if Pruefung then begin r := H;  for k := 0 step 1 until r do b[k] := h[k];
                    Pruefung := false;  go to Ausgabe
                    end;
           if komplex then begin r := r - 2;  for k := 0 step 1 until r do
                                              h[k] := b[k] := A[k + 2] end
                      else begin r := r - 1;  for k := 0 step 1 until r do
                                              h[k] := b[k] := A[k + 1] end
           Hornerschema(c, m, x0, y0, Rf, If, A);
           M := Rf × Rf + If × If;
           if M < delta then

Ausgabe:   begin if komplex
                 then begin outreal(1, x0);  outreal(1, y0);
                            for j := 1 step 1 until 7 do outsymbol(1,'komplex',j)
                      end
                 else begin outreal(1, x0);
                        for j := 1 step until 5 do outsymbol(1,'reell',j)
                      end;
                      if r > 0.5
                      then begin x0 := y0 := 0; komplex := false;  go to Teil 1 end
                      else go to Ende
           end;
           Pruefung := true;  H := r;  go to Teil 0;
```

Teil 0 ist vor *Teil* 1 zu setzen; darin wird $r = m$ und $P_r(z) = P(z)$ gesetzt.

```
Teil 0: r := m;  for k := 0 step 1 until r do b[k] := c[k];
```

In den Programmteil *Sonderfall* wird eingetreten, wenn kein absolut kleinerer Funktionswert als $P_r(z_0)$ aufzufinden ist, obwohl $|P_r(z_0)|^2 \geqq \delta$ ist. Es ist zu vermuten, daß dieses Verhalten auf Rundungsfehler zurückzuführen ist, die sich besonders stark bemerkbar machen, wenn $|P_r'(z_0)|$ sehr groß ist. Aus diesem Grunde wird geprüft, ob $|P_r'(z_0)|^2 \cdot \varepsilon_1 > |P_r(z_0)|^2$ mit einem kleinen $\varepsilon_1 > 0$ gilt. In diesem Fall wird zum Programmteil *Kontrolle* übergegangen, andernfalls wird das Programm mit Fehleranzeige beendet.

```
Sonderfall: if N × epsilon1 > M then go to Kontrolle;
            for j = 1 step 1 until 6 do outsymbol(1,'Fehler',j);  Ende:
```

Obwohl es an sich nicht sinnvoll ist, die einfachen Fälle $r = 2$ und $r = 1$ (quadratische bzw. lineare Gleichung) nach diesem Verfahren zu behandeln, seien diese Fälle nicht gesondert programmiert, da das zusätzliche Entscheidungen und Programmteile erfordert. In das Programm ist noch ein Teil zur Eingabe bzw. zur Berechnung der c_k aufzunehmen, alle Namen sind zu erklären. Außerdem sind Anfangswerte für *x*0, *y*0, *komplex* und *Pruefung* einzustellen.

```
Programm:
begin integer m;  inreal(2,m);
      begin real x0, y0, Rf, If, Rfa, Ifa, delta, epsilon, N, M, x1, y1,
                 x2, y2, M1, M2, d, epsilon1;
            integer k, r, ZV, j, H;  Boolean komplex, Merke, Pruefung;
            array b, c, h[0:m], A[0:m + 1], B[0:m - 1];
            procedure Hornerschema ...;
            inreal(2, delta);  inreal(2, epsilon);  inreal(2, d);
            inreal(2, epsilon1);  inarray(2, c);  Pruefung := false;
            x0 := y0 := 0;  komplex := false;  Teil 0: ...;
            Teil 1: ...;  Teil 2: ...;  Teil 3: ...;  Teil 4: ...;
            Kontrolle: ...;  Sonderfall: ...;
Ende: end
end.
```

Jedes Programm läßt sich auch als Prozedurrumpf auffassen und in eine Prozedurerklärung einbetten. Werden formale Parameter eingeführt, so sind diese nicht mehr zu erklären, sondern zu spezifizieren. Im vorliegenden Fall könnte man schreiben:

```
procedure Nullstellenbestimmung (c, m, delta, epsilon, epsilon1, d);
          value m, delta, epsilon, epsilon1, d;  array c;
          integer m;  real delta, epsilon, epsilon1, d;
          begin real x0, y0, Rf, If, Rfa, Ifa, N, M, x1, y1, x2, y2, M1, M2;
                integer k, r, ZV, j, H;  Boolean komplex, Merke, Pruefung;
                array b, h[0:m], A[0:m + 1], B[0:m - 1];
                procedure Hornerschema ...;
                Pruefung: = false ; ...;
   Ende: end
```

In einem weiteren Beispiel sei die genäherte Lösung der Anfangswertaufgabe für eine gewöhnliche Differentialgleichung erster Ordnung nach dem Runge-Kutta-Verfahren programmiert. Die Differentialgleichung sei $z' = f(x,z)$, gegeben sei der Anfangswert *za* an der Stelle *xa*. Das Verfahren werde mit der Schritt-

weite $h > 0$ ausgeführt, je zwei Schritte seien zu einem Doppelschritt zusammengefaßt. Die genäherte Integration von einem Ausgangspunkt (x_0, z_0) führt zu folgendem Formelsatz:

$$f_0 = f(x_0, z_0)$$

$$z_1 = z_0 + \frac{h}{2} \cdot f_0, \qquad x_1 = x_0 + \frac{h}{2}, \quad f_1 = f(x_1, z_1)$$

$$z_2 = z_0 + \frac{h}{2} \cdot f_1, \qquad x_2 = x_0 + \frac{h}{2}, \quad f_2 = f(x_2, z_2)$$

$$z_3 = z_0 + h \cdot f_2, \qquad x_3 = x_0 + h, \quad f_3 = f(x_3, z_3)$$

$$z_4 = z_0 + \frac{h}{6} \cdot f_0 + \frac{h}{3} \cdot f_1 + \frac{h}{3} \cdot f_2 + \frac{h}{6} \cdot f_3, \qquad x_4 = x_0 + h, \quad f_4 = f(x_4, z_4)$$

$$z_5 = z_4 + \frac{h}{2} \cdot f_4, \qquad x_5 = x_4 + \frac{h}{2}, \quad f_5 = f(x_5, z_5)$$

$$z_6 = z_4 + \frac{h}{2} \cdot f_5, \qquad x_6 = x_4 + \frac{h}{2}, \quad f_6 = f(x_6, z_6)$$

$$z_7 = z_4 + h \cdot f_6, \qquad x_7 = x_4 + h, \quad f_7 = f(x_7, z_7)$$

$$z_8 = z_4 + \frac{h}{6} \cdot f_4 + \frac{h}{3} \cdot f_5 + \frac{h}{3} \cdot f_6 + \frac{h}{6} \cdot f_7, \qquad x_8 = x_4 + h, \quad f_8 = f(x_8, z_8)$$

Zusätzlich wird berechnet

$$Z_8 = z_0 + \frac{h}{3} \cdot f_0 + \frac{4h}{3} \cdot f_4 + \frac{h}{3} \cdot f_8.$$

Ist $Z_8 - z_8$ genügend klein, so wird das Verfahren fortgesetzt, andernfalls wird der Schritt mit halber Schrittweite wiederholt. Ist $Z_8 - z_8$ sehr klein, so kann die Schrittweite verdoppelt werden.

Das Verfahren kann in ALGOL 60 in folgender Weise dargestellt werden:

```
begin real ha, xa, xe, za, delta, epsilon, hm; inreal(2, xa); inreal(2, xe);
    inreal(2,za); inreal(2ha); inreal(2,hm); inreal(2,delta);
    inreal(2,epsilon);
    begin real v, z, x0, z0, z4, Z8, F, h, d3, c;
            integer i; array d, e[0:2]; Boolean Durchlauf 2;
            h := ha;
```

```
Anfang:  x := xa;  z := za;  d[0] := d[1] := h/2;  d[2] := h;
         d3 := e[0] := h/6;  c := e[1] := e[2] := h/3;  F := f(x,z);
         comment f bezeichnet eine globale Funktion;
      A: xa := x;  za := Z8 := z;  Durchlauf2 := false;
      B: x0 := x;  z0 := z4 := z;  Z8 := Z8 + c × F;
         if Durchlauf2 then Z8 := Z8 + h × F;
         for i := 0 step 1 until 2 do
         begin x := x0 + d[i];  z := z0 + d[i] × F;
               z4 := z4 + e[i] × F;  F := f(x,z)
         end;
         x := x0 + h;  z := z4 + d3 × F;  F := f(x,z);
         if ¬ Durchlauf2 then begin Durchlauf 2 := true; go to B end;
Kontrolle: Z8 := Z8 + c × F;
         if abs(Z8 − z) < epsilon
         then begin if h < m then begin h := 2 × h;  go to Anfang end
              end;
         if abs(Z8 − z) > delta
         then begin h := h/2;  go to Anfang end;
         outreal(1,x);  outreal(1,z);  outsymbol(1,'␣', 1);
         if x < xe then go to A
   end
end
```

Bei jeder Anwendung dieses Blocks ist er in einen weiteren Block einzuordnen, in dem die globale Funktion f erklärt ist. Zum Beispiel kann $z' = (x - z) \cdot z$ integriert werden, wenn ein Programm

```
begin real procedure f(x,z);  value x, z;  real x, z;
                      f := (x − z) × z;
   begin ...
         (obiges Verfahren)
   end
end
```

programmiert und ein entsprechender Satz von Anfangswerten zur Eingabe über Kanal 2 vorbereitet wird.

Das Verfahren läßt sich leicht zur Lösung von Differentialgleichungssystemen erster Ordnung erweitern. Das zu lösende System laute

$$z_k' = f_k(x, z_1, \ldots, z_n) \quad \text{für} \quad k = 1, \ldots, n.$$

An Stelle des oben benutzten $Z8 - z$ muß bei der Prüfung eine passende Norm des Vektors mit den Komponenten $Z8_1 - z_1, \ldots, Z8_n - z_n$ treten, z. B. $\max_k |Z8_k - z_k|$. Zur Darstellung werde jetzt die Form einer Unterprogrammerklärung gewählt:

```
procedure RKSystem (ha, xa, xe, n, za, delta, epsilon, hm, f);
        value ha, xa, xe, delta, epsilon, hm, n;
        real ha, xa, xe, delta, epsilon, hm; integer n;
        array za; real procedure f;

    begin real x, x0, h, d3, c;
        array z, z0, z4, Z8, F[1:n];
        array d, e[0:2];
        integer i, k; Boolean Durchlauf2;
        h := ha;

Anfang: x := xa; for k := 1 step 1 until n do z[k] := za[k];
        d[0] := d[1] := h/2; d[2] := h; d3 := e[0] := h/6;
        c := e[1] := e[2] := h/3;
        for k := 1 step 1 until n do F[k] := f(x, z, k);
    A:  xa := x; for k := 1 step 1 until n do za[k] := Z8[k] := z[k];

        Durchlauf2 := false;

    B:  x0 := x; for k := 1 step 1 until n do
                 begin z0[k] := z4[k] := z[k];
                     Z8[k] := Z8[k] + c × F[k];
                     if Durchlauf 2 then Z8[k] := Z8[k] + h × F[k]
                 end;
        for i := 0 step 1 until 2 do
        begin x := x0 + d[i]; for k := 1 step 1 until n do
                              begin z[k] := z0[k] + d[i] × F[k];
                                  z4[k] := z4[k] + e[i] × F[k]
                              end;
              for k := 1 step 1 until n do F[k] := f(x,z,k)
        end;

        x := x0 + h; for k := 1 step 1 until n do z[k] := z4[k] + d3 × F[k];
        for k := 1 step 1 until n do F[k] := f(x,z,k);
        if ¬Durchlauf2 then begin Durchlauf2 := true; go to B end
```

```
Kontrolle: for k := 1 step 1 until n do Z8[k] := Z8[k] + c × F[k];
           for k := 1 step 1 until n do
               if abs(Z8[k] − z[k]) ≧ epsilon then go to K1;
           if h < hm then begin h := 2 × h;  go to Anfang end;
       K1: for k := 1 step 1 until n do
               if abs(Z8[k] − z[k]) > delta
               then begin h := h/2;  go to Anfang end;
           outreal(1,x);  outarray(1,z);  outsymbol(1, '␣', 1);
           if x < xe then go to A
       end
```

Soll z. B. die Differentialgleichung $y^{IV} = (1 + x - x^2)\, y$ genähert integriert werden, so führt man diese zunächst auf ein System von Gleichungen erster Ordnung zurück:

$$\begin{aligned} z_1 &= y \quad \text{oder} \quad z_1' = z_2 \\ z_2 &= y' \qquad\quad\ \ z_2' = z_3 \\ z_3 &= y'' \qquad\quad\ \ z_3' = z_4 \\ z_4 &= y''' \qquad\quad\ \ z_4' = (1 + x - x^2)\, z_1 \end{aligned}$$

An Anfangsbedingungen seien gegeben $y(0) = y''(0) = 0$, $y'(0) = y'''(0) = 1$. Die Genauigkeit sei durch $\delta = 10^{-5}$ und $\varepsilon = 10^{-8}$ bestimmt, als Anfangsschrittweite und auch als maximale Schrittweite werde 0.1 gewählt; die Integration soll bis $xe = 1$ durchgeführt werden.

Es entsteht folgendes Programm:

```
begin real procedure f(x,z,k); value x, k; real x; array z; integer k;
          begin switch V := A1, A2, A3, A4;
              go to V[k];
              A1: f := z[2];  go to A5;
              A2: f := z[3];  go to A5;
              A3: f := z[4];  go to A5;
              A4: f := (1 + x − x × x) × z[1];
              A5: end
      array ZA[1:4];
      procedure RKSystem ...;
      ZA[1] := ZA[3] := 0;  ZA[2] := ZA[4] := 1;
      RKSystem (0.1, 0, 1, 4, ZA, 10−5, 10−8, 0.1, f)
end
```

Die in der Unterprogrammerklärung *RKSystem* häufig vorkommende Anweisung „**for** $k := 1$ **step** 1 **until** n **do** $F[k] := f(x,z,k)$" kann vorteilhaft als Unterprogramm betrachtet werden, dadurch wird das gesamte Programm kürzer. Dieses Unterprogramm könnte als „*Funktion*(x,z,n,F)" aufgerufen werden. Der letzte der formalen Parameter von *RKSystem* müßte dann „*Funktion*" lauten und als **procedure** (nicht als **real procedure**) spezifiziert werden. Für das betrachtete Beispiel ergibt sich ein entsprechend geändertes Programm:

```
begin procedure P(X,Z,N,R); value X,N; real X; integer N; array Z,R;
      begin R[1] := Z[2]; R[2] := Z[3]; R[3] := Z[4];
            R[4] := (1 + X - X × X) × Z[1]
      end;
      array ZA[1:4];
      procedure RKSystem (ha, xa, xe, n, za, delta, epsilon, hm, Funktion); ...;
      ZA[1] := ZA[3] := 0; ZA[2] := ZA[4] := 1;
      RKSystem(0.1, 0, 1, 4, ZA, 10-5, 10-8, 0.1, P)
end
```

Die Wahl der Namen formaler Parameter für ein Unterprogramm ist völlig frei, es empfiehlt sich trotzdem, um den Leser eines Programms vor Verwechslungen zu schützen, möglichst nicht solche Namen zu benutzen, die im Programm noch an anderer Stelle auftreten. Es hätte das Unterprogramm ohne weiteres auch in der Form *Funktion*(x,z,n,F), in der es aufgerufen wird, erklärt werden können, nur bezeichnen die Namen x, z, n und F dann einmal formale und einmal aktuelle Parameter. Eine solche Schreibweise erleichtert mitunter das Verständnis, darf jedoch nicht dazu führen, keinen Unterschied zwischen aktuellen und formalen Parametern zu machen. An der Erklärung der Prozedur P ist noch bemerkenswert, daß der formale Parameter N im Prozedurrumpf nicht vorkommt. Ähnliches kann bei speziellen Erklärungen für formale Unterprogramme häufig auftreten. Keinesfalls darf der betreffende Parameter jedoch im formalen Parameterteil weggelassen werden, da stets die Anzahl aktueller und formaler Parameter übereinstimmen muß.

ALGOL-Unterprogramme, die häufig benutzt werden, können auch direkt als Maschinenunterprogramme formuliert werden. Das hat besonders dann Vorteile, wenn dadurch wesentliche Einsparungen an Rechenzeit, eventuell auch an Speicherraum erzielt werden. Die Erklärung eines solchen Maschinenunterprogramms erfolgt so, daß an Stelle des Prozedurrumpfes eine nicht in ALGOL geschriebene Zeichenfolge tritt, die mit einem Semikolon abzuschließen ist. Diese Zeichenfolge besteht bei Anwendung der ALCOR-Konvention nur aus dem Symbol 'code'

(dieses Symbol gehört nicht zu den ALGOL-Zeichen). Es muß dann innerhalb des benutzten Übersetzungssystems dafür gesorgt werden, daß das Maschinenunterprogramm in die Maschine eingegeben wird. Seine Form muß die in ALGOL geforderte Parameterzuordnung gestatten, natürlich muß auch der erforderliche Speicherplatz freigehalten werden. Beim Aufruf muß das Maschinenunterprogramm auffindbar sein, nach seiner Beendigung ist wieder in das übersetzte Programm zu springen. Für den ALGOL-Programmierer macht diese Art der Einführung die Verwendung von Maschinenunterprogrammen besonders einfach, Bei der Übersetzung des ALGOL-Programms müssen allerdings die zusätzlichen Vorschriften für die Eingabe der Maschinenunterprogramme beachtet werden.

Als Beispiel wurde angenommen, ein dem Unterprogramm *RKSystem* ähnliches Unterprogramm *rkscode* zur genäherten Lösung von Differentialgleichungssystemen liege vor. Soll nun das Randwertproblem

$$y'' + (a_1 x + a_0)\, y' + (b_2 x^2 + b_1 x + b_0)\, y = 0.$$
$$y(0) = c, \quad y(1) = d$$

gelöst werden, so kann dies unter Benutzung dieses Maschinenunterprogramms geschehen. Es werden dazu zwei Lösungen $y_1(x)$ und $y_2(x)$ der Differentialgleichung mit $y_1(0) = y_2(0) = c$ und $y_1'(0) = 0$, $y_2'(0) = 1$ bestimmt. Da die Differentialgleichung linear ist, ist mit $y_1(x)$ und $y_2(x)$ auch $k_1 y_1(x) + k_2 y_2(x)$ eine Lösung. Die Konstanten k_1 und k_2 sind aus

$$k_1 y_1(1) + k_2 y_2(1) = d$$
$$k_1 \qquad + k_2 \qquad = 1$$

zu bestimmen. Für $y_1(1) = y_2(1)$ ist die Randwertaufgabe nicht lösbar, falls nicht $y_1(1) = y_2(1) = d$ ist; dann gibt es eine ganze Lösungsschar, k_1 und k_2 ergeben sich zu

$$k_1 = (d - y_2(1))/(y_1(1) - y_2(1))$$
$$k_2 = (y_1(1) - d)/(y_1(1) - y_2(1)).$$

Mit den Anfangsbedingungen $y_3(0) = c, y_3'(0) = k_2$ wird die Integration nochmals zur Kontrolle wiederholt.

Das benutzte Unterprogramm *rkscode* diene zur Integration des Gleichungssystems $z_i' = f_i(x, z_1, \ldots, z_n)$ für $i = 1, \ldots, n$ mit dem Runge-Kutta-Verfahren. Als formale Parameter treten auf:

ha = Anfangswert der Schrittweite, $ha > 0$
hm = Maximalwert der Schrittweite, $hm = 2^r ha$ (r ganze nicht negative Zahl)
xa = Anfangswert für x
xe = Endwert für x, $xe > x$, $xe - x$ ganzzahliges Vielfaches von *hm*

n = Anzahl der Gleichungen
za = Anfangsvektor, $za[i] = z_i\,(xa)$
ze = Resultatvektor, $ze[i] = \bar{z}_i(xe)$ ($\bar{z}_i$ seien die durch das Verfahren erzeugten Näherungen)
del = Toleranz für gewünschte Genauigkeit
eps = Fehlermaß bei Prüfung zur Verdoppelung der Schrittweite, $eps < 10^{-2} \cdot del$
fk = Unterprogrammname zur Erzeugung der f_i mit 4 Parametern
1. Parameter: x, 2. Parameter: Vektor z mit Komponenten z_i
3. Parameter: n, 4. Parameter: Vektor f mit Komponenten f_i
dr = Boolesche Variable mit Wert **true**, wenn Zwischenresultate zu drucken sind, sonst **false**.

Gegenüber dem Unterprogramm *RKSystem* sei die Verdoppelung der Schrittweite nur an Stellen erlaubt, die sich von *xa* durch ein ganzzahliges Vielfaches von *hm* unterscheiden. Die Ausgabe der Näherungen erfolge nur, wenn *dr* den Wert **true** hat. Ein Programm zur Lösung des betrachteten Problems kann in folgender Form in der ALCOR-Notation geschrieben werden.

```
'begin' 'procedure' rkscode (ha, hm, xa, xe, n, za, ze, del, eps, fk, dr);
'value' ha, hm, xa, xe, n, del, eps, dr;
'real' ha, hm, xa, xe, n, del, eps; 'boolean' dr; 'array' za, ze;
'procedure' fk; 'code';
'real' a0, a1, b0, b2, c, d, k1, k2; 'array' anf, end1, end2[1:2];
'procedure' fkt (x, z, m, f); 'value' x, m;
'real' x; 'integer' m; 'array' z, f;
'begin' f[1] := z[2];
        f[2] := - (a1 × x + a0) × z[2]
                - (b2 × x × x + b1 × x + b0) × z[1]
'end';

inreal(2, a0); inreal(2, a1);
inreal(2, b0); inreal(2, b1); inreal(2, b2);
inreal(2, c); inreal(2, d); anf[1] := c; anf[2] := 0;
rkscode (0.1, 0.1, 0, 1, 2, anf, end1, 10-5, 10-8, fkt, 'false');
anf[1] := c; anf[2] := 1;
rkscode(0.1, 0.1, 0, 1, 2, anf, end2, 10-5, 10-8, fkt, 'false');
'if' abs(end1[1] - end2[1]) < 10-4 'then' 'go to' sonder;

    k1 := (d - end2[1])/(end1[1] - end2[1]);
    k2 := (end1[1] - d)/(end1[1] - end2[1]);
```

```
druck: anf[1] := c;  anf[2] := k2;
       rkscode (0.1, 0.1, 0. 1, 2, anf, end1, 10-5, 10-8, fkt, 'true');
       'go to' ende;
sonder: 'if' abs(end1[1] - d) > 10-4 'then' 'go to' unloesbar 'else'
        k2 := 0.5; 'go to' druck;
unloesbar: outsymbol(1, '('u')', 1)
ende:
'end'
```

Bei der Programmierung in ALGOL 60 für eine bestimmte Rechenanlage empfiehlt es sich, Besonderheiten der Rechenanlage und des Übersetzungssystems zu berücksichtigen. Das gilt vor allem dann, wenn ein ALGOL-Programm häufiger benutzt werden soll. Solche Besonderheiten können etwa bei der Behandlung indizierter Variabler in Laufanweisungen auftreten. Bei manchen Übersetzungssystemen wird die Änderung einer Laufvariablen besonders einfach, wenn sie in Indexlisten nur an letzter Stelle als Variable steht und in der Laufanweisung jeweils um eine Konstante erhöht wird. Es ist dann nämlich auch die Adresse der betreffenden indizierten Variablen nur um diese Konstante zu erhöhen, wenn die Laufvariable geändert wird. Am Beispiel der Matrizenmultiplikation sei das erläutert. Es war bereits die Anweisung

```
for i := 1 step 1 until n do
    for k := 1 step 1 until m do
        begin c[i,k] := 0;
              for j := 1 step 1 until l do
                  c[i,k] := c[i,k] + a[i,j] × b[j,k]
        end
```

angegeben worden. Die einzige, nur an letzter Stelle einer Indexliste auftretende Laufvariable ist k. Um die mit der Änderung von k auftretenden Adressenänderungen möglichst häufig vorkommen zu lassen, müßte die Laufanweisung mit der Laufvariablen k in den beiden anderen Laufanweisungen eingeschlossen sein. In der angegebenen Form wird sie n-mal durchlaufen, die Laufanweisung mit der Laufvariablen j jedoch $n \cdot m$-mal. Im vorliegenden Fall sind Umordnungen einfach möglich, einmal können die Laufanweisungen in der Form **for** i ... **do** **for** j **do** ... **for** k angeordnet werden, ein andermal in der Form **for** j **do** ... **for** i ... **do** ... **for** k ... (vgl. [1]).

1. Form: **for** $i := 1$ **step** 1 **until** n **do**
begin for $k := 1$ **step** 1 **until** m **do** $c[i,k] := 0$;
for $j := 1$ **step** 1 **until** l **do**
for $k := 1$ **step** 1 **until** m **do**
$c[i,k] := c[i,k] + a[i,j] \times b[j,k]$
end

2. Form: **begin for** $i := $ **step** 1 **until** n **do**
for $k := 1$ **step** 1 **until** m **do** $c[i,k] := 0$;
for $j := 1$ **step** 1 **until** l **do**
for $i := 1$ **step** 1 **until** n **do**
for $k := 1$ **step** 1 **until** m **do**
$c[i,k] := c[i,k] + a[i,j] \times b[j,k]$
end

Ist ein Übersetzungssystem so beschaffen, daß bei jedem Vorkommen einer indizierten Variablen eine komplizierte Adressenrechnung ausgeführt wird, so sollte man versuchen, die Anzahl dieser Vorkommen weitgehend zu verringern. Man wird dann die Matrizenmultiplikation schreiben:

for $i := 1$ **step** 1 **until** n **do**
for $k := 1$ **step** 1 **until** m **do**
begin real s; $s := 0$
for $j := 1$ **step** 1 **until** l **do**
$s := s + a[i,j] \times b[j,k]$;
$c[i,k] := s$
end

Es kann auch versucht werden, an Stelle von Matrizen nur mit Vektoren zu hantieren, um kürzere Rechenzeiten zu erreichen. Dann ist jedoch meist die Anfertigung einer Code-Prozedur zweckmäßiger, da der Vorteil von ALGOL darin liegt, Probleme schnell und übersichtlich zu programmieren. Dieser Vorteil geht jedoch verloren, wenn man spezielle Eigenschaften einer bestimmten Rechenanlage in der ALGOL-Notation stark berücksichtigt, um sehr schnell ablaufende Programme zu erhalten. Das hier angeführte Beispiel der Matrizenmultiplikation sollte nur als Anregung bei der Programmierung ähnlicher Probleme dienen, für die Matrizenmultiplikation wird meistens eine Code-Prozedur existieren.

Weitere Programmierungsbeispiele werden hier nicht gebracht. In Form von ALGOL-Prozeduren geschriebene Programme werden laufend veröffentlicht, z. B. in [19], [20], [21], man vgl. auch [1], [2], [11], [14]. Zu einer Rechenanlage gehört eine Unterprogrammbibliothek, die auch in ALGOL-Programme einfügbare

Maschinenunterprogramme enthält, sofern die Rechenanlage zur Verarbeitung von ALGOL-Programmen vorgesehen ist. Bei der Programmierung eines speziellen Problems kann in vielen Fällen weitgehend auf diese Unterprogrammbibliothek zurückgegriffen werden, die Programmierungsarbeit wird dadurch weiter erleichtert; die Anzahl der Fehlerquellen wird verringert, da die Unterprogramme geprüft sind; das Programm wird kürzer und übersichtlicher.

Bei jeder Programmierung ist darauf zu achten, daß die Resultate möglichst weitgehend kontrolliert werden, da auch in Rechenautomaten Rechenfehler vorkommen können. Näherungsverfahren sollten so beschaffen sein, daß sie mit Fehlerabschätzungen oder wenigstens Fehlerschätzungen verbunden sind. Daher sollte man bei dem Einsatz eines Unterprogramms stets prüfen, wie weit diese Gesichtspunkte im Unterprogramm berücksichtigt sind. Auch Sonderfälle, wie etwa das Verschwinden eines Nenners, sollen stets beachtet werden. Die hier wiedergegebenen Programme erfüllen diese Bedingungen nicht vollständig. So enthält das Programm zur Berechnung der Nullstellen eines Polynoms weder Fehlerabschätzungen noch Fehlerschätzungen, statt dessen nur eine Kontrolle einer gefundenen Näherung durch Berechnen des Polynomwertes, wobei nicht berücksichtigt wird, daß diese Berechnung mit dem Hornerschen Schema wesentliche Rundungsfehler erzeugen kann. Das Programm zur Lösung von Differentialgleichungssystemen berechnet nur ein Maß für die Größenordnung des Fehlers. Beim letzten Programm zur Lösung der Randwertaufgabe wird die gefundene Näherungslösung nicht noch einmal kontrolliert. Da die Näherungslösung gedruckt wird, kann man an den ausgedruckten Resultaten ablesen, ob die Randwertaufgabe tatsächlich gelöst wurde. Daher kann auf eine solche Kontrolle auch verzichtet werden.

Die mit der Bestimmung der Fehler von Näherungsverfahren zusammenhängenden Probleme fallen in das Arbeitsgebiet der Numerischen Mathematik. Dazu gehört z. B. auch das Problem der Stabilität einer Rechnung. Bei instabilen Verfahren treten starke Fehlerfortpflanzungen auf, die im Laufe einer längeren Rechnung zu vollständig falschen Resultaten führen können. Hier kann auf derartige Fragen nicht eingegangen werden, es sei jedoch betont, daß ihre Klärung im Hinblick auf eine weitere Automatisierung von Rechenvorgängen außerordentlich wichtig ist. Probleme, die von einem rein theoretischen Standpunkt aus als vollständig gelöst anzusehen sind, können in numerischer Betrachtung durchaus offen sein. Das gilt z. B. für die Bestimmung der Nullstellen von Polynomen einer Veränderlichen, deren Existenz durch den Fundamentalsatz der Algebra gewährleistet ist. Die numerische Lösung kann jedoch in Spezialfällen immer wieder Schwierigkeiten bereiten, besonders dann, wenn man noch die Fehler der Nullstellen wissen möchte. Auch der Einfluß von Ungenauigkeiten der Koeffizienten

spielt hierbei eine Rolle, man vgl. z. B. [56]. Da derartige Fragen bei vielen Problemen nur teilweise gelöst sind, hat die Numerische Mathematik ein sehr umfangreiches Aufgabengebiet, dessen Bewältigung Voraussetzungen für eine weitere Automatisierung wissenschaftlicher und technischer Berechnungen schafft. Aus der umfangreichen Literatur seien hier nur [1], [4], [18] und [34] erwähnt.

Für die ALGOL-Programmierung ist die Arbeitsweise ebenso wie bei der Maschinenprogrammierung; man kann mehrere Arbeitsgänge unterscheiden:

1. Auswahl eines numerischen Verfahrens.
2. Darstellung in ALGOL.
3. Korrekte Niederschrift entsprechend den Regeln des benutzten Übersetzungssystems.
4. Auswahl numerischer Beispiele mit bekannten Resultaten.
5. Übersetzung des Programms und Prüfung an den ausgewählten Beispielen.

Beim fünften Arbeitsgang wird man häufig noch Fehler feststellen, die sowohl formaler Art (nicht korrekte Notierung) als auch Fehler in der Formulierung des Verfahrens (unberücksichtigte Sonderfälle, mehrfach ausgenutzte Variable, nicht eingestellte Anfangswerte usw.) sein können. Es kann sein, daß gewisse Fehler bei den ausgewählten Beispielen gar nicht in Erscheinung treten, daher sollen die Beispiele so gewählt werden, daß ein möglichst weiter Bereich erfaßt wird. Nur ein mehrfach einwandfrei gelaufenes Programm sollte in eine Programmbibliothek aufgenommen werden.

II. ALGOL-VARIANTE ROBOTRON 300

Die zur Arbeit mit dem Rechenautomaten Robotron 300 einsetzbare Variante von ALGOL wird hier näher betrachtet. Es handelt sich dabei um eine Erweiterung der auf Seite 52 beschriebenen Sprache IFIP SUBSET ALGOL 60. Die Einschränkung, daß eine Potenzierung mit ganzzahliger Basis und negativem ganzzahligen Exponenten nicht möglich ist, entfällt; ebenso darf das Zeichen ÷ für ganzzahlige Division verwendet werden. Zusätzlich werden weitere Standardfunktionen sowie Standardprozeduren zur Ein- und Ausgabe eingeführt. Zur Anwendung dieser ALGOL-Variante sind einige Kenntnisse über den Aufbau der Rechenanlage erforderlich.

§ 3. Kurze Beschreibung der Rechenanlage Robotron 300

Die Rechenanlage verarbeitet alphanumerische Zeichen, jedes alphanumerische Zeichen ist eine Zusammenfassung von acht bits. Die einzelnen bits eines Zeichens werden parallel verarbeitet, die Zeichen selbst in Serie. Der Hauptspeicher der Anlage ist ein Ferritkernspeicher, der 40000 Zeichen aufnehmen kann; jede dieser 40000 Stellen ist adressierbar. Es können Folgen von Zeichen automatisch verarbeitet werden, die in Stellen mit aufeinanderfolgenden Adressen gespeichert sind. Die Zeichenfolge wird entweder durch ein markiertes Zeichen (Zeichen mit Wortmarke) oder durch eines von drei Sonderzeichen (Satzmarke, Gruppenmarke, Blockmarke) abgeschlossen. Die Verarbeitung von Zeichenfolgen wird wie üblich durch Befehle veranlaßt. Das Befehlssystem ist ein Einadreßsystem, im allgemeinen gibt die Adresse eines Befehls die Stelle an, mit der beginnend die zu verarbeitende Zeichenfolge gespeichert ist.

Bei Notierung eines Zeichens in Form einer Bitfolge werden im allgemeinen nur sechs bits notiert, sie werden von links nach rechts als *v-bit*, *u-bit*, *8-bit*, *4-bit*, *2-bit*, *1-bit* bezeichnet. Für Eingabezwecke kann ein Lochstreifen benutzt werden, der als Achtkanalstreifen aufgebaut ist. Auf diesem Lochstreifen werden bits mit dem Wert 1 wie üblich durch ein Loch dargestellt, bits mit dem Wert 0 bleiben ungelocht. Die Anordnung der einzelnen bits erfolgt in Spuren, zwischen der dritten und vierten Spur liegt eine Taktspur. Es besteht folgende Zuordnung:

Spur 1: 1-bit
Spur 2: 2-bit
Spur 3: 4-bit

Taktspur

Spur 4: 8-bit
Spur 5: Prüfbit
Spur 6: *u*-bit
Spur 7: *v*-bit
Spur 8: Wortmarkenbit

Das Prüfbit wird so gebildet, daß die Gesamtzahl der bits je Zeichen ungerade ist (einschließlich Prüfbit und Wortmarkenbit). Das Wortmarkenbit wird besetzt, wenn das betreffende Zeichen als Ende eines Wortes (spezielle Zeichenfolge) markiert ist oder eines der erwähnten Sonderzeichen ist. Bei der Niederschrift einzelner Zeichen werden selbstverständlich nicht die Bitfolgen notiert, sondern die darzustellenden Zeichen selbst. Die 64 verfügbaren Zeichen sind in der folgenden Tabelle so zusammengestellt, daß jeweils 16 Zeichen in einer Zeile nebeneinander stehen (entsprechend den Bitfolgen 0000, 0001, ..., 1111 in der üblichen Anordnung); die einzelnen Zeilen entsprechen den Kombinationen 00, 01, 10 und 11 von *v*-bit und *u*-bit:

0	1	2	3	4	5	6	7	8	9	□	#	(		▼	]
+	a	b	c	d	e	f	g	h	i	~		;	!	▼▼	"
−	j	k	l	m	n	o	p	q	r	≈	)	*	=	<	?
	/	s	t	u	v	w	x	y	z	≋		%	△	>	[

In der Rechenanlage gespeicherte Zeichen können mit Hilfe einer am Steuerpult der Anlage angeschlossenen Schreibmaschine geschrieben werden oder mittels eines Zeilendruckers ausgedruckt werden. Die Zeichen können weiter in der oben beschriebenen Form auf Lochstreifen oder in einer hier nicht näher zu erörternden Art auf Lochkarten gestanzt werden. Werden Zeichen über die Schreibmaschine ausgeschrieben, so werden mit Wortmarke markierte Zeichen unterstrichen, falls ein entsprechender Ausgabebefehl verwendet wird. Das Zeichen □ veranlaßt die Betätigung der Leertaste, erscheint also nicht im Schriftbild. Bei Ausgabe über den Schnelldrucker werden die kleinen Buchstaben durch große ersetzt (0 und o sind dann nicht mehr unterscheidbar), die Zeichen □, ~, ≈, ≋, ▼, ▼▼ und △ erscheinen als Leerzeichen oder führen auf einen Druckerfehler. Im Hauptspeicher dienen die Zeichen ~, ≈ bzw. ≋ als Satzmarke, Gruppenmarke bzw. Blockmarke, wenn sie zusätzlich durch eine Wortmarke gekennzeichnet sind.

Die Zeichenverarbeitung im Inneren der Maschine erfolgt in der *Zentraleinheit*, die neben dem Hauptspeicher Steuerwerk und Rechenwerk umfaßt. Zur Bedienung der Anlage ist ein *Steuerpult* mit Anzeigetafel und der bereits erwähnten Schreibmaschine, über die auch Informationen eingegeben werden können, vorgesehen. Weitere Eingabegeräte sind Lochstreifen- und Lochkartenleser, die Ausgabe erfolgt über die bereits erwähnten Geräte. Die Speicherkapazität kann durch Anschluß von Zusatzkernspeicher, Magnetbandgeräten und Magnettrommelspeichern gesteigert werden.

Zur Übersetzung und Verarbeitung von ALGOL-Programmen sind Zentraleinheit, Steuerpult mit Schreibmaschine, Lochstreifenleser, Schnelldrucker und Zusatzkernspeicher für 10000 Zeichen erforderlich. Um größere Programme übersetzen zu können, sind Zwischenausgaben erforderlich. Hierzu werden üblicherweise Magnetbandgeräte benutzt, jedoch ist auch die Verwendung des Lochstreifenstanzers möglich; dabei erhöht sich der Zeitbedarf beträchtlich. Der Einsatz von Magnetbandgeräten hat den weiteren Vorteil, daß der Wechsel zwischen den verschiedenen Programmteilen des ALGOL-Übersetzers automatisch und schnell erfolgt.

Die Rechenanlage kann ganze Zahlen und Gleitkommazahlen verarbeiten. Die Zahlen sind als vorzeichenbehaftete Dezimalzahlen mit einer unter gewissen Einschränkungen beliebigen Stellenzahl dargestellt. Zu Rechnungen wird ein Akkumulator benutzt, der 120 Stellen aufnehmen kann. Die höchste Stelle einer Zahl trägt eine Wortmarke, adressiert wird die Zahl an ihrer niedrigsten Stelle. Diese Stelle trägt bei ganzen Zahlen auch das Vorzeichen, das als Minuszeichen im v-bit erscheint, das Pluszeichen wird weggelassen. Gleitkommazahlen enthalten in den beiden niedrigsten Stellen den Exponenten, die restlichen Stellen bilden die ganzzahlige Mantisse. Sowohl Exponent als auch Mantisse sind vorzeichenbehaftet, wie bei ganzen Zahlen beschrieben. Für den Exponenten e gilt somit $-99 \leqq e \leqq 99$. In der hier behandelten ALGOL-Variante sind sämtliche Zahlen zehnstellige Gleitkommazahlen (8 Stellen Mantisse und 2 Stellen Exponent). Die Zahl π wird also maschinenintern genähert dargestellt durch 3̲14159270p, dabei ist die Zeichenfolge 0p Darstellung des Exponenten -7. Ganze Zahlen werden ebenfalls als Gleitkommazahlen mit dem Exponenten 0 dargestellt, der Bereich für darstellbare ganze Zahlen g ist daher $|g| < 10^8$. Zur Speicherung logischer Werte sind in der Rechenanlage 20 elektronische Schalter (Selektoren 0, ..., 19) und vier Handschalter (Wahlschalter 0, ..., 3) vorhanden. Die elektronischen Schalter können automatisch aus- bzw. eingeschaltet werden, die Wahlschalter nur von Hand. Der einem Schalter zugeordnete Wert ist **true**, wenn der Schalter eingeschaltet ist, **false**, wenn der Schalter ausgeschaltet ist.

§ 4. Einschränkungen der ALGOL-Variante

Die Einschränkungen der ALGOL-Variante gegenüber der Sprache ALGOL 60 seien hier im einzelnen aufgezählt, ihre Reihenfolge entspricht der Anordnung des revidierten ALGOL-Berichtes [44].

1. Es sind nur kleine Buchstaben zugelassen (*a*, *b*, ..., *z*).
2. Die Zeichen **own** und ⌴ sind nicht zugelassen.
3. Namen, die in den ersten sechs Zeichen übereinstimmen, werden als gleich angesehen.
4. Zahlen müssen den durch die maschineninterne Darstellung (s. o.) bedingten Einschränkungen hinsichtlich ihres Absolutbetrages genügen.
5. Zeichenreihen können aus den maschineninternen Zeichen mit Ausnahme der Zeichen ~, ≈, ≋, ▼, ▼▼ und △ gebildet werden. Die Zeichenfolgen '('und')' stellen die ALGOL-Zeichen ' und ' dar, daher dürfen sie im Inneren einer eigentlichen Zeichenkette nicht vorkommen.
6. Der Absolutbetrag eines Index muß kleiner als 10^5 sein.
7. Der Wert eines einfachen arithmetischen Ausdrucks ist höchstens dann vom Typ **integer**, wenn das Resultat und alle bei seiner Auswertung entstehenden Zwischenresultate (mit Ausnahme der Zwischenresultate bei der Berechnung des Argumentes der Standardfunktion *entier*) absolut kleiner als 10^8 bleiben.
8. Eine Potenz $a \uparrow i$ (a beliebiger arithmetischer Ausdruck, i ganzzahliger arithmetischer Ausdruck) hat den gleichen Typ wie a, wenn i eine vorzeichenlose nichtnegative ganze Zahl ist, sonst den Typ **real**.
9. Als Marken sind nur Namen zugelassen.
10. Bedingte Zielausdrücke sind nicht zugelassen.
11. Eine Sprunganweisung ist undefiniert, wenn der Zielausdruck ein Verteiler ist, dessen Wert undefiniert ist.
12. In Laufanweisungen darf zwischen **for** und := nur eine einfache Variable stehen. Bei Beendigung einer Laufanweisung behält diese Variable ihren letzten Wert.
13. Aktuelle Parameter, die symbolische Parameter ersetzen (call by name) dürfen nur Namen, Zeichenketten oder indizierte Variable sein. Im letzteren Fall werden jedoch die Indizes wie bei Wertaufruf (call by value) behandelt. Namen von Standardprozeduren für Ein- und Ausgabe sind als aktuelle Parameter nicht zugelassen.

14. Als aktuelle Parameter für Wertaufruf sind nur Ausdrücke zugelassen, jedoch nicht parameterlose Funktionen. Daher dürfen Namen von Feldern als aktuelle Parameter nur im Namensaufruf verwendet werden.
15. Bei Namensaufruf müssen Art und Typ des aktuellen Parameters mit Art und Typ des zugeordneten formalen Parameters übereinstimmen. Bei Wertaufruf gilt Entsprechendes mit der Ausnahme, daß aktuelle Parameter vom Typ **integer** formalen Parametern vom Typ **real** zugeordnet werden dürfen.
16. Eine Prozedur darf während der Ausführung des Prozedurkörpers, während der Auswertung aktueller mit Namen aufgerufener Parameter oder während der Auswertung von Erklärungen innerhalb der Prozedur nicht nochmals aufgerufen werden.
17. Erklärungen mit **own** entfallen.
18. Der Absolutbetrag des Wertes einer Feldgrenze muß kleiner als 10^5 sein.
19. Auf der rechten Seite einer Verteilererklärung dürfen nur Marken vorkommen. Diese Marken müssen in dem Block, in dessen Kopf die Verteilererklärung enthalten ist, lokal oder global sein, sie dürfen also nicht erst in einem inneren Block als Ziel (Marke vor :) auftreten.
20. Sämtliche formalen Parameter müssen spezifiziert werden.

§ 5. Erweiterungen der ALGOL-Variante

Gegenüber ALGOL 60 wurden zusätzliche Standardfunktionen und Standardprozeduren sowie die neuen Zeichen **code** und **wait** eingeführt. Das Zeichen **wait** ist für den Inhalt des Programms ohne Bedeutung, es hat nur den Zweck, die Übersetzung zeitweilig unterbrechen zu können. Das Zeichen **code** ist im ALGOL-Programm als Prozedurkörper für solche Prozeduren zu verwenden, die im Maschinencode programmiert vorliegen. Diese Prozeduren werden während des Programmlaufs bei erstmaligem Aufruf eingelesen. Sie sind entweder auf Magnetband oder auf Lochstreifen gespeichert und werden entsprechend automatisch eingelesen oder durch eine Schreibmaschinenausschrift angefordert.

Im folgenden bezeichne E bzw. E_i einen arithmetischen Ausdruck, B einen logischen Ausdruck, I bzw. I_i einen ganzzahligen arithmetischen Ausdruck. An Standardfunktionen treten zu denen von ALGOL 60 hinzu:

tan(E)	für den tangens von E.
arcsin(E)	für den Hauptwert des arcus sinus von E.
arccos(E)	für den Hauptwert des arcus cosinus von E.
arc(E_1, E_2)	für den im Bogenmaß angegebenen Polwinkel

	eines Punktes mit den kartesischen Koordinaten E_1 und E_2. Dabei ist $0 \leqq arc(E_1, E_2) < 2\pi$, $arc(0, 0)$ ist undefiniert.
$div(I_1, I_2)$	für den ganzen Teil von I_1/I_2.
$res(I_1, I_2)$	für $I_1 - div(I_1, I_2) \times I_2$ (Rest der Division I_1/I_2).
$max(E_1, ..., E_n)$	für den größten Wert von $E_1, ..., E_n$.
$min(E_1, ..., E_n)$	für den kleinsten Wert von $E_1, ..., E_n$.
$sel(I)$	für die Stellung des Schalters I mit $0 \leqq I \leqq 23$. Dabei sind die Schalter 0, ..., 19 die Selektoren 0, ..., 19; die Schalter 20, ..., 23 die Wahlschalter 0, ..., 3.
$sel(I, B)$	für die Stellung des Schalters I mit $0 \leqq I \leqq 19$. Ist B **true**, so wird der Schalter zusätzlich (nach Abfrage) eingeschaltet; ist B **false**, ausgeschaltet.

Die neu eingeführten Standardfunktionen haben folgende Typen:

real procedure	*tan*, *arcsin*, *arccos*, *arc*, *max*, *min*;
integer procedure	*div*, *res*;
Boolean procedure	*sel*.

Die für Ein- und Ausgabezwecke eingeführten *Standardprozeduren* haben die Namen *read*, *print*, *write*, *input* und *output*. Dabei dient *read* zum Einlesen von Werten über Lochstreifen, *print* zum Ausdrucken von Werten bzw. Zeichenreihen über Schnelldrucker, *write* zum entsprechenden Ausschreiben mittels der Steuerpultschreibmaschine, *input* zum Einlesen von Werten über ein durch den ersten Parameter zu spezifizierendes Gerät, *output* zur entsprechenden Ausgabe. Der bei *input* bzw. *output* das Gerät spezifizierende erste Parameter ist eine ganze Zahl (Kanal), Kanal 1 bezeichnet den Lochstreifenstanzer, Kanal 2 den Eingabeteil der Steuerpultschreibmaschine. Die Standardprozeduren können mit beliebig vielen Parametern notiert werden. Für *read* sind als Parameter Variable (auch indizierte Variable) und Feldnamen zugelassen, für *print* und *write* arithmetische und logische Ausdrücke sowie Zeichenreihen, für *print* zusätzlich noch Feldnamen. Während bei *read*, *write*, *print* und *input* verschiedene Arten von Parametern in einem aktuellen Parameterteil vereint sein können, sind bei *output* nur Parameter gleichen Typs in einem aktuellen Parameterteil zugelassen, abgesehen vom ersten Parameter für die Kanalnummer und dem zweiten Parameter für ein *Format*. Anhand eines Beispiels sei die Wirkungsweise der einzelnen Standardprozeduren erläutert:

```
begin real x,y; integer i,k,n; array a[1:3,1:3];
      read(x,y); write(x,y); input(2,n);
      print(n, 'produkt □=', x × y); print(n × x, n = 1);
      for i := 1 step 1 until 3 do
      for k := 1 step 1 until 3 do a[i,k] := i + k;
      print(a); output(1, 'e', a); output(1, 'l', n ≠ 1)
end
```

Die Anweisung *read*(*x*,*y*) veranlaßt, daß zwei Zeichengruppen vom Lochstreifen gelesen werden. Diese Zeichengruppen werden geprüft, ob sie reelle oder ganze Zahlen darstellen, entsprechend in die interne Zahlendarstellung umgeformt und danach zur Bewertung der Variablen *x* und *y* benutzt. Der verwendete Lochstreifen ist ein Achtkanalstreifen (vgl. § 6), er wird mit einer speziellen Schreibmaschine hergestellt. Als Trennzeichen zwischen den einzelnen Zeichengruppen (Eingabeinformationen) dient das Zeichen Wagenrücklauf/Zeilenvorschub. In der internen Zeichendarstellung (vgl. § 3) entspricht diesem das Zeichen Satzmarke (geschrieben als ≃). Die Eingabeinformation ist für Zahlen die in ALGOL 60 übliche Zahlendarstellung (z. B. 0.1; 25; .2; $_{10}-5$; -7.5; $-0{,}25_{10}-3$ usw.). Sollen *x* und *y* die Werte 0.1 und $5_{10}-2$ zugeordnet werden, so ist bei Herstellung des Lochstreifens mit der Eingabeschreibmaschine zu schreiben:

```
0.1
5 # −2
```

Das Zeichen $_{10}$ wird bei Ein- und Ausgabe prinzipiell durch das Zeichen # dargestellt. Sollen logische Werte eingegeben werden, so sind die Zeichenfolgen 'true' bzw. 'false' auf den Eingabestreifen zu lochen.

Die Anweisung *write*(*x*,*y*) veranlaßt die Ausschrift

```
.10000000#00      .500000000# −01
```

mittels der Steuerpultschreibmaschine. Reelle Zahlen werden stets in Form normierter Gleitkommazahlen ausgegeben, d. h., sie beginnen mit . und einer von 0 verschiedenen Ziffer, gegebenenfalls davor noch ein Minuszeichen. Eine Ausnahme bildet die Zahl 0, bei der die ausgegebene Mantisse die Form .00000000 hat; der Exponententeil kann von Fall zu Fall verschieden sein. Ganze Zahlen werden in der üblichen Schreibweise ausgegeben. Die gleiche Darstellung wird auch bei Ausgabe über den Schnelldrucker mittels *print* benutzt. Im einzelnen ist der Aufbau (das Ausgabeformat) ausgegebener Zeichenreihen folgender:

Reelle Zahlen werden mit 14 Zeichen ausgegeben, das erste Zeichen ist − bei negativen Zahlen, Zwischenraum bei nichtnegativen Zahlen; das zweite Zeichen ist der Dezimalpunkt (.); drittes bis zehntes Zeichen werden von Dezimalziffern

besetzt; das elfte Zeichen ist #, das zwölfte Zeichen ist Vorzeichen des Exponenten (− oder Zwischenraum); dreizehntes und vierzehntes Zeichen sind Dezimalziffern (Exponent). Ganze Zahlen werden mit neun Zeichen ausgegeben. Hat eine ganze Zahl $m \leqq 8$ geltende Stellen, so besetzen die m Ziffern die letzten Stellen, das Vorzeichen wird direkt davor gesetzt (− oder Zwischenraum). Logische Werte werden als 'true' oder 'false' ('TRUE' bzw. 'FALSE' auf Schnelldrucker) ausgegeben. Zeichenreihen werden ohne Berücksichtigung der begrenzenden ALGOL-Zeichen ‘ und ’ ausgegeben. Zwischen je zwei auszugebenden Zeichengruppen, deren Ausgabe durch einen Aufruf von write bzw. print veranlaßt wird, werden drei Zwischenräume eingeschoben. Reicht bei Ausgabe über den Schnelldrucker der Platz auf einer Zeile nicht aus, so wird mit der betreffenden Zeichengruppe eine neue Zeile begonnen. Bei Beendigung einer Zeile auf der Steuerpultschreibmaschine erfolgen Zeilenvorschub und Wagenrücklauf automatisch.

Die Anweisung *input*(2,*n*) bewirkt die Ausschrift einer Meldung

sr − eingabe:

mittels der Steuerpultschreibmaschine. Danach geht der Automat in einen Zustand über, der eine Eingabe über diese Schreibmaschine verlangt. Nach Eingabe einer Zahl, deren Darstellung mit dem Typ des zweiten Parameters von *input* verträglich sein muß, wird der Programmablauf fortgesetzt. Die einzugebende Zahl ist durch eine Satzmarke ($\simeq$) abzuschließen. Hier kann z. B. eine die ganze Zahl 2 darstellende Zeichenfolge ($2\simeq$) eingetippt werden. Das Schreibmaschinenprotokoll hat danach die Form

sr − eingabe: $2\simeq$

Zwischenräume werden bei der Eingabe übergangen.

Die Ausführungen der Anweisungen

print(*n*, ‘*produkt* □=’, $x \times y$); *print*($n \times x$, $n = 1$)

bewirkt den Druck folgender Zeilen mit dem Schnelldrucker:

```
          2     PRODUKT =      .50000000#−02
.20000000#  00        'FALSE'
```

Mit jeder neuen Anweisung *print* wird eine neue Schnelldruckerzeile begonnen, ebenso mit jeder neuen Anweisung *write* eine neue Schreibmaschinenzeile. Die Anweisung *print*(*a*) veranlaßt den Druck der Matrix *a* mittels Schnelldrucker. Zum Druckbild ist zu bemerken, daß die Einzelwerte eines Feldes in lexikographischer Reihenfolge ausgegeben werden (vgl. Erläuterung zu *outarray* auf S. 35). Bei n-dimensionalen Feldern werden die Werte mit übereinstimmenden $n - 1$ ersten Indizes in einer Ausgabegruppe zusammengefaßt. Vor jede solche Ausgabegruppe

wird eine Zeile mit diesen Indizes gedruckt, bei einer zu großen Dimension werden entsprechend mehrere Zeilen mit Indizes gedruckt. Die Ausgabegruppe selbst wird in einer dem Typ entsprechenden Anordnung gedruckt, als erster der Wert mit dem kleinstmöglichen letzten Index. Bei eindimensionalen Feldern (Vektoren) gibt es nur eine Ausgabegruppe, eine Indexzeile wird nicht gedruckt. Zweidimensionale Felder (Matrizen) werden demnach in der üblichen Schreibweise (zeilenweise) mit zwischengeschobenen Zeilennummern gedruckt. Für jeden Index sind sechs Zeichen vorgesehen, sonst entspricht das Druckbild eines Index dem einer ganzen Zahl. Hier ergibt sich folgende Anordnung:

```
    1
.20000000# 01   .30000000# 01   .40000000# 01
    2
.30000000# 01   .40000000# 01   .50000000# 01
    3
.40000000# 01   .50000000# 01   .60000000# 01
```

Werden Felder mit einer Anweisung *read* eingelesen, so ist die gleiche lexikographische Anordnung der Werte auf dem Eingabestreifen zu wählen, Indizes sind in diesem Fall nicht zu lochen. Die Matrix *a* kann mit der Anweisung *output*(1, '*e*', *a*) in der für eine solche Eingabe erforderlichen Form gestanzt werden. Die abschließende Anweisung *output*(1, '*l*', $n \neq 1$) veranlaßt Stanzen der Zeichenfolge 'true' □≃. Das Format '*e*' bzw. '*l*' als zweiter aktueller Parameter von *output* dient zur Bezeichnung der hier eingeführten Ausgabeformate für Zahlen und logische Werte.

§ 6. Herstellung von Eingabestreifen

Eingabelochstreifen werden mit einer Schreibmaschine hergestellt, die Achtkanallochstreifen in einem Code locht, der weitgehend mit dem Interncode der Rechenanlage Robotron 300 übereinstimmt. Es ist der Code des Schreibautomaten Optima 528. Da mit diesem Schreibautomaten sowohl große als auch kleine Buchstaben und eine Anzahl von Sonderzeichen geschrieben werden können, sind ebenso wie bei den bereits beschriebenen internationalen Fünfkanal-Fernschreibcode Umschaltzeichen vorgesehen (vgl. S. 49). Für die Herstellung von ALGOL-Eingabelochstreifen dürfen weder große Buchstaben noch die Sonderzeichen △ und ⊙ (Irrung Satz) verwendet werden. Die folgende Tabelle enthält die Zuordnung zwischen ALGOL-Zeichen und der bei der Herstellung des Lochstreifens

zu benutzenden Schreibweise, Buchstaben und Ziffern werden in der Tabelle nicht aufgeführt.

ALGOL-Zeichen	Text auf Schreibmaschine	ALGOL-Zeichen	Text auf Schreibmaschine
true	'true'		
false	'false'	10	#
+	+		
−	−	:=	:=
×	*	**go to**	'go to'
/	/	**if**	'if'
÷	'div'	**then**	'then'
↑	'power'	**else**	'else'
=	=	**for**	'for'
<	<	**do**	'do'
>	>	**step**	'step'
≦	'notgreater'	**until**	'until'
≧	'notless'	**while**	'while'
≠	'notequal'	**begin**	'begin'
≡	'equiv'	**end**	'end'
⊃	'impl'	**Boolean**	'boolean'
∨	'or'	**integer**	'integer'
∧	'and'	**real**	'real'
¬	'not'	**array**	'array'
(	(	**string**	'string'
)	)	**label**	'label'
[	[	**value**	'value'
]	]	**switch**	'switch'
‘	'('	**comment**	'comment'
	')'	**code**	'code'
		wait	'wait'

Einige ALGOL-Zeichen können auch in anderer Form geschrieben werden, und zwar < als 'less', = als 'equal', > als 'greater', ↑ als **. Das Zeichen **wait** darf nur nach ; oder **end** verwendet werden. Der Zweck von **wait** ist, einen Teillochstreifen abzuschließen. Es ist dadurch möglich, ein Programm auf mehrere Lochstreifen zu schreiben. Wird während der Übersetzung ein Zeichen **wait** erreicht, so wird die Übersetzung unterbrochen, der nächste Teilstreifen kann eingelegt werden. Danach wird die Rechenanlage neu gestartet. Es ist wichtig, jeden

Lochstreifen – auch Teilstreifen – mit einem Zeichen Wagenrücklauf/Zeilenvorschub abzuschließen, da anderenfalls der Eingabevorgang nicht abgeschlossen wird. Die Tasten für Zeilenvorschub und für Wagenrücklauf allein dürfen nicht benutzt werden.

§ 7. Ablauf der Übersetzung

Der Ablauf der Übersetzung wird hier nur grob skizziert; für Einzelheiten sei auf die Bedienungsanleitung verwiesen. Nach Einlegen des Magnetbandes mit dem Übersetzungsprogramm und des Lochstreifens mit dem ALGOL-Programm wird die Maschine gestartet. Über die Steuerpultschreibmaschine wird der Text

titel:

ausgeschrieben. Hierauf ist ein aus vier Zeichen bestehender Titel einzutippen, der im Lauf der Übersetzung und Rechnung auf sämtliche ausgegebenen Protokolle gedruckt wird. Während der folgenden Übersetzung wird der Lochstreifen zeilenweise, d. h. jeweils bis zu einem Zeichen Wagenrücklauf/Zeilenvorschub, gelesen. Die Zeilen werden gezählt, über Schnelldrucker wird der ALGOL-Text mit Zeilennummern versehen zur Kontrolle ausgedruckt. Nach Beendigung der Übersetzung folgt die Schreibmaschinenausschrift

ausgabe:

Hierauf kann entweder ja oder nein eingetippt werden. Im ersten Fall wird das erzeugte Maschinenprogramm ausgegeben, im zweiten Fall beginnt die Rechnung. Der Beginn der Rechnung wird durch Ausschreiben des zu Beginn eingegebenen Titels gefolgt von „start" angezeigt. Es kann dann durch Betätigung des Startknopfes die Ausführung des übersetzten ALGOL-Programms ausgelöst werden; vorher ist gegebenenfalls ein Datenlochstreifen einzulegen. Nach Beendigung der Rechnung folgt die Schreibmaschinenschrift

ende ...
... wiederholen?:

Hierbei deutet ... den Titel des Programms an; nach Eintippen von ja bzw. nein wird die Rechnung wiederholt oder die Abarbeitung des Programms eingestellt. Bei Wiederholung wird wie oben beschrieben

... start

über die Schreibmaschine ausgegeben, es kann dann gegebenenfalls ein neuer Datenstreifen eingelegt werden; danach wird wie oben verfahren.

§ 8. Fehlererkennung

Die Übersetzung läuft in zwei Phasen ab; in jeder Phase können gewisse Fehler im zu übersetzenden Programm entdeckt werden. Es ist jedoch auch möglich, daß sich Fehler erst während der Rechnung bemerkbar machen. In jedem Fall werden Fehlermeldungen über den Schnelldrucker ausgegeben. In Sonderfällen kann es vorkommen, daß auch regelwidrige Eingangstexte übersetzt und gerechnet werden, die dann vorliegende Interpretation des Eingangstextes kann in solchen Fällen mit dem vom Programmierer beabsichtigten – jedoch inkorrekt notierten – Effekt übereinstimmen.

Werden bei der Übersetzung Fehler bemerkt, so wird die betreffende Phase zu Ende geführt, um eventuell noch weitere Fehler entdecken zu können. Das erzeugte Programm ist dann nicht verwertbar; es folgt die Schreibmaschinenausschrift

programm fehlerhaft

Danach wird die Übersetzung abgebrochen. Ein Fehler kann bei dem Versuch, die Übersetzung weiterzuführen, in gewissen Fällen weitere Fehlermeldungen hervorrufen, da dem Übersetzungsprogramm Informationen fehlen können. Es ist daher in jedem Einzelfall zu überprüfen, ob der angezeigte Fehler tatsächlich vorliegt. Die Fehlermeldungen werden bei Ablauf der ersten Übersetzungsphase zwischen die einzelnen Protokollzeilen gedruckt; sie stehen daher automatisch an der Programmstelle, an der sich der Fehler erstmalig bemerkbar macht. Diese Stelle stimmt im allgemeinen nicht mit der Stelle überein, an der der Fehler gemacht wurde; z. B. werden fehlende Erklärungen erst am Ende des Programms gemeldet. Während der zweiten Übersetzungsphase werden eine Zeilennummer, die Nummer des Semikolons in der betreffenden Zeile und die Nummer einer Fehlergruppe ausgedruckt. Das angegebene Semikolon ist das vor dem Erkennen des angezeigten Fehlers zuletzt aufgetretene Semikolon; der Fehler selbst kann also auch in einer der folgenden Zeilen erkannt worden sein.

Während der ersten Übersetzungsphase können die in der folgenden Tabelle erläuterten Fehlermeldungen gedruckt werden

Fehlermeldung	Fehlerart
UEBERLAUF 01	Der vorhandene Speicherplatz reicht zur Übersetzung nicht aus
UEBERLAUF 02	Die Struktur des ALGOL-Programms ist zu kompliziert
UEBERLAUF 03	Abbruch der Übersetzung wegen andauernder Auswirkung früherer Fehler
FEHLER 04	Falscher Zahlenaufbau

FEHLER 05 Unzulässiges ALGOL-Zeichen verwendet
FEHLER 06 Markierung durch einen bereits erklärten oder zur Markierung verwendeten Namen; Benutzung einer Marke, die kein Name ist
FEHLER 07 Falsche Klammerstruktur
FEHLER 08 Name in unzulässigem Zusammenhang
FEHLER 09 Syntaktischer Fehler in Ausdruck oder in Wertzuweisung
FEHLER 10 Variable hinter **for** ist keine einfache Variable
FEHLER 11 Falscher Aufbau einer Prozedurerklärung
FEHLER 12 Zeichen **code** nicht als Prozedurkörper verwendet
FEHLER 13 Spezifikationszeichen **procedure** in falschem Zusammenhang
FEHLER 14 Formaler Parameter nicht spezifiziert
FEHLER 15 Erklärung nicht am Blockanfang
FEHLER 16 Nachwirkung eines früheren Fehlers
FEHLER 17 Abbruch der Übersetzung als Wirkung früherer Fehler
ALGOL-TEXT FALSCH Nicht verwertbares Zeichen
FEHLER 18 vom Lochstreifen gelesen,
daher Abbruch der Übersetzung
FEHLER 19 Zeichen **else** in falschem Zusammenhang

Die am Ende der ersten Übersetzungsphase mögliche Fehlermeldung

DEKL? ...

enthält einen sechsstelligen Namen, der im Programm vorkam, ohne daß er an dieser Stelle erklärt war.

Die Fehlerarten, die in der zweiten Übersetzungsphase erkannt werden können, sind in zehn Gruppen eingeteilt, die mit 0 bis 9 numeriert sind. Den einzelnen Gruppen entsprechen folgende Fehler:

0: Objektprogramm zu umfangreich
1: Syntaktischer Fehler im ALGOL-Programm
2: Typ eines Operanden nicht mit Operationszeichen verträglich
3: Typen in Wertzuweisung nicht verträglich
4: Aktueller Parameter einer Standardfunktion oder Standardprozedur hat falschen Typ
5: Unzulässige Namen in Verteilerliste
6: Anzahl aktueller Parameter falsch
7: Unzulässige aktuelle Parameter in Eingabeprozedur
8: Ausdruck als linke Seite einer Wertzuweisung
9: Syntaktischer Fehler im ALGOL-Programm

Tritt bei der Abarbeitung des übersetzten Programms ein Fehler auf, so wird eine Fehlermeldung über die Steuerpultschreibmaschine ausgeschrieben und mit der Ausnahme von Eingabefehlern zusätzlich über Schnelldrucker ausgedruckt. Anschließend wird der Programmlauf angehalten. Als Fehlerausdrucke sind möglich: ZELLE LEER; ADRESSE > 39999; DIVISOR = 0; HS VOLL; SPRUNG IN BLOCK; FELDGRENZEN; FELDLÄNGE; PARAMETERTYP; AUSDRUCK ALS PARAMETER; PARAMETERANZAHL; INDEX ZU KLEIN; INDEX ZU GROSS; WERT UNBESTIMMT; ARGUMENT < 0; ZAHL ZU GROSS; KANAL NR.; SEL NR. und als Fehlerausschriften bei Eingabefehlern „eingabefehler : ls", „eingabefehler : sr" und „eingabefehler : typ". Die Fehlermeldungen sind in den meisten Fällen direkt verständlich; daher seien nur einige Fehler erläutert:

HS VOLL	tritt auf, wenn der vorhandene Speicherplatz zur Fortsetzung der Rechnung nicht ausreicht.
FELDGRENZEN	tritt auf, wenn eine obere Feldgrenze kleiner als die zugehörige untere Grenze ist.
FELDLÄNGE	tritt auf, wenn der Speicherplatz nicht ausreicht, um ein neu erklärtes Feld aufzunehmen.
KANAL NR.	tritt auf, wenn eine nicht vorgesehene Kanalnummer in *input* bzw. *output* verwendet wird.
SEL NR.	tritt auf, wenn in der Standardfunktion *sel* eine unzulässige Selektornummer verwendet wird.

Zu den Eingabefehlern ist zu bemerken, daß die Ausschriften erfolgen, wenn ein eingegebener Wert inkorrekt aufgebaut ist – dabei bezieht sich ls auf die Lochstreifeneingabe, sr auf die Schreibmaschineneingabe – oder einen Typ hat, der mit dem Typ des in der Eingabeprozedur stehenden Parameters nicht verträglich ist.

Eine weitere Fehlerklasse wird durch die Schreibmaschinenausschrift „anzeige" gekennzeichnet. Dadurch wird auf eine Fehleranzeige am Steuerpult hingewiesen.

§ 9. Beispiele

Im folgenden sind zwei Beispiele wiedergegeben, als erstes ein Programm zur Lösung der quadratischen Gleichung

$$x^2 + px + q = 0.$$

Die Eingangsdaten p und q werden auf Lochstreifen gelocht; die Rechnung wird mit jeweils neuen Daten solange wiederholt, bis einmal $p = 99999999$ gelesen

wird. Im zweiten Beispiel wird die auf Seite 44 behandelte Funktion *cosinus* benutzt. Das Programm veranlaßt das Ausdrucken einer Tabelle von *cos x* für $x = 0°, \ldots, 180°$ in Schritten von 1°. Wiedergegeben sind Programmniederschrift (Abb. 2 und Abb. 6), Datenniederschrift beim ersten Beispiel (Abb. 3), Übersetzungsprotokoll des Schnelldruckers (Abb. 4 und Abb. 7) und Rechenresultate (Abb. 5 und Abb. 8). Die benutzten Titel sind *qugl* und *cosi.*

```
'begin''real'd,p,q,u,v;
print('('loesung von x'power'2 + p*x + q = 0')');
print('(' ')');
a:read(p,q);'if'p=99999999'then''goto'b;
print('('p  =')',p,'('q  =')',q);
d:=0.25*p'power'2-q;
u:=-0.5*p;v:=sqrt(abs(d));
'if'd'notless'0'then'
print('('x1 =')',u+v,'('x2 =')',u-v)'else'
'begin'print('('x1 =')',u,'('+ i*')',v);
       print('('x2 =')',u,'('- i*')',v)
'end'; print('(' ')');'goto'a;
b:'end';'wait'
```

Abb. 2. Programmniederschrift *qugl*

```
-2
 1
-0.7
 0.1
 0.1
- .459#-1
 5.4#-1
 1.018#-1
#1
 3.4#1
  .6
  .8
-#12
  74#22
  99999999
    0
```

Abb. 3. Datenniederschrift *qugl*

QUGL

```
01  'BEGIN' 'REAL' D,P,Q,U,V;
02  PRINT('('LOESUNG VON X'POWER'2 + P*X + Q = 0')');
03  PRINT('(' ')');
04  A:READ(P,Q);'IF'P=99999999'THEN''GOTO'B;
05  PRINT('('P  =')',P,'('Q  =')',Q);
06  D:=0.25*P'POWER'2-Q;
07  U:=-0.5*P;V:=SQRT(ABS(D));
08  'IF'D'NOTLESS'U'THEN'
09  PRINT('('X1 =')',U+V,'('X2 =')',U-V)'ELSE'
10  'BEGIN'PRINT('('X1 =')',U,'('+ I*')',V);
11         PRINT('('X2 =')',U,'('- I*')',V)
12  'END'; PRINT('(' ')');'GOTO'A;
13  B:'END';'WAIT'
```

Abb. 4. Übersetzungsprotokoll *qugl*

```
LOESUNG VON X'POWER'2 + P*X + Q = 0

P  =   - 20000000# 01    Q  =     .10000000# 01
X1 =     10000000# 01    X2 =     .10000000# 01

P  =   -.70000000# 00    Q  =     .10000000# 00
X1 =     .50000000# 00    X2 =     .20000000# 00

P  =     .10000000# 00    Q  =     .45900000#-01
X1 =     .17000000# 00    X2 =     .27000000# 00

P  =     .54000000# 00    Q  =     .10180000# 00
X1 =   -.27000000# 00    + I*     .17000000# 00
X2 =   -.27000000# 00    - I*     .17000000# 00

P  =     .10000000# 02    Q  =     .34000000# 02
X1 =   -.50000000# 01    + I*     .30000000# 01
X2 =   -.50000000# 01    - I*     .30000000# 01

P  =     .60000000# 00    Q  =     .80000000# 00
X1 =   -.30000000# 00    + I*     .84261498# 00
X2 =   -.30000000# 00    - I*     .84261498# 00

P  =   -.10000000# 13    Q  =     .74000000# 24
X1 =     .50000000# 12    + I*     .70000000# 12
X2 =     .50000000# 12    - I*     .70000000# 12
```

Abb. 5. Rechenresultate *qugl*

```
'begin''integer'i;
    'real''procedure' cosinus(x);
        'comment'die funktion cosinus(x)entspricht der standard-
                 funktion cos(x), der abweichende name ist
                 erforderlich,da standardfunktionen nicht
                 erklaert werden;
        'value'x;'real'x;
        'begin''comment' das argument wird zunaechst auf  das
                         intervall0'notgreater'y'notgreater'
                          2*pi mit  pi:=3.14159265reduziert,
                         es wird ausgenutzt,dass cos (x)eine
                         gerade funktion ist;
                    'real'y,z;'integer'q;'switch'v:=a,b,b,a;
                                             y:=abs(x);
                y:=y-entier(y/6.283 185 31)*6.283 185 31;
                q:=entier(y/1.57079633)+1;
               'comment'q gibt den quadranten des arguments y an,
                        dieses wird im folgenden auf 0'notgreater'
                        y'notgreater'pi/8 reduziert.cos(y)wird
                        durch taylersche entwicklung1-y'power'
                        2/2+y'power'4/24-y'power'6/720approximiert,
                        durch zweimalige anwendung von cos(2*y)=2*
                       cos(y)*cos(y)-1zuruecktransformiert;
                       'if'y'greater'3.14159265'then'y:=6.28318531
                        -y;
                        'if'y'greater'1.57079633'then'y:=3.14159265
                                -y;
                      y:=y*y/16;
                       'comment' beginn des hornerschen schemas
                                 fuer  berechnung der taylorschen
                                 entwicklung;
                     z:=1/24-y/720;
                     z:=z*y-0.5;z:=z*y+1;
                    'comment' beginn der ruecktransformation;
                     z:=2*(z*z-0.5);z:=2*(z*z-0.5);
                    'goto'v[q];'comment'der verteiler v dient zur
                    festlegung des vorzeichens des  resultats;
              b:cosinus:=-z;'goto'c;
              a:cosinus:=z;
        c:'end'cosinus;
    'for'i:=0'step'1'until'180'do'
          print(i,cosinus(3.14159265/180*i))
'end''wait'
```

Abb. 6.
Programmniederschrift *cosi*

```
                                                                    COSI
01  'BEGIN''INTEGER'I;
03      'REAL''PROCEDURE' COSINUS(X);
04          'COMMENT'DIE FUNKTION COSINUS(X)ENTSPRICHT DER STANDARD-
05                   FUNKTION COS(X). DER ABWEICHENDE NAME IST
06                   ERFORDERLICH,DA STANDARDFUNKTIONEN NICHT
07                   ERKLAERT WERDEN;
08          'VALUE'X;'REAL'X;
09          'BEGIN''COMMENT' DAS ARGUMENT WIRD ZUNAECHST AUF  DAS
10                        INTERVALL0'NOTGREATER'Y'NOTGREATER'
11                         2*PI MIT  PI:=3.14159265REDUZIERT.
12                        ES WIRD AUSGENUTZT,DASS COS (X)EINE
13                        GERADE FUNKTION IST;
14                    'REAL'Y,Z;'INTEGER'Q;'SWITCH'V:=A,B,B,A;
15                                                Y:=ABS(X);
16                  Y:=Y-ENTIER(Y/6,283 185 31)*6,283 185 31;
17                  Q:=ENTIER(Y/1,57079633)+1;
18                 'COMMENT'Q GIBT DEN QUADRANTEN DES ARGUMENTS Y AN,
19                         DIESES WIRD IM FOLGENDEN AUF 0'NOTGREATER'
20                         Y'NOTGREATER'PI/8 REDUZIERT.COS(Y)WIRD
21                         DURCH TAYLORSCHE ENTWICKLUNG,Y'POWER'
22                         2/2+Y'POWER'4/24-Y'POWER'6/720APPROXIMIERT.
23                         DURCH ZWEIMALIGE ANWENDUNG VON COS(2*Y)=2*
24                        COS(Y)*COS(Y)-1ZURUECKTRANSFORMIERT;
25                         'IF'Y'GREATER'3.14159265'THEN'Y:=6.28318531
26                         -Y;
27                         'IF'Y'GREATER'1.57079633'THEN'Y:=3.14159265
28                                 -Y;
29                        Y:=Y*Y/16;
30                        'COMMENT' BEGINN DES HORNERSCHEN SCHEMAS
31                                 FUER  BERECHNUNG DER TAYLORSCHEN
32                                 ENTWICKLUNG;
33                    Z:=1/24-Y/720;
34                    Z:=Z*Y-0.5;Z:=Z*Y+1;
35                   'COMMENT' BEGINN DER RUECKTRANSFORMATION,
36                   Z:=2*(Z*Z-0.5);Z:=2*(Z*Z-0.5);
38                  'GOTO'V[Q];'COMMENT'DER VERTEILER V DIENT ZUR
39                   FESTLEGUNG DES VORZEICHENS DES  RESULTATS;
40              B:COSINUS:=-Z;'GOTO'C;
41              A:COSINUS:=Z;
42          C:'END'COSINUS;
43      'FOR'I:=0'STEP'1'UNTIL'180'DO'
44           PRINT(I,COSINUS(3.14159265/180*I))
45  'END''WAIT'
```

Abb. 7. Übersetzungsprotokoll *cosi*

0	.10000000# 01	67	.39073112# 00	133	-.68199846# 00
1	.99984768# 00	68	.37460648# 00	134	-.69465846# 00
2	.99939076# 00	69	.35836792# 00	135	-.70710680# 00
3	.99862960# 00	70	.34202012# 00	136	-.71933974# 00
4	.99756410# 00	71	.32556816# 00	137	-.73135374# 00
5	.99619478# 00	72	.30901694# 00	138	-.74314488# 00
6	.99452184# 00	73	.29237164# 00	139	-.75470962# 00
7	.99254616# 00	74	.27563732# 00	140	-.76604436# 00
8	.99026810# 00	75	.25881892# 00	141	-.77714600# 00
9	.98768836# 00	76	.24192186# 00	142	-.78801070# 00
10	.98480776# 00	77	.22495094# 00	143	-.79863552# 00
11	.98162726# 00	78	.20791156# 00	144	-.80901700# 00
12	.97814754# 00	79	.19080892# 00	145	-.81915200# 00
13	.97437008# 00	80	.17364816# 00	146	-.82903766# 00
14	.97029576# 00	81	.15643438# 00	147	-.83867058# 00
15	.96592574# 00	82	.13917298# 00	148	-.84804812# 00
16	.96126166# 00	83	.12186922# 00	149	-.85716732# 00
17	.95630466# 00	84	.10452826# 00	150	-.86602538# 00
18	.95105644# 00	85	.87155640#-01	151	-.87461970# 00
19	.94551856# 00	86	.69756300#-01	152	-.88294756# 00
20	.93969264# 00	87	.52335800#-01	153	-.89100660# 00
21	.93358048# 00	88	.34899320#-01	154	-.89879410# 00
22	.92718384# 00	89	.17452160#-01	155	-.90630784# 00
23	.92050492# 00	90	.12000000#-06	156	-.91354554# 00
24	.91354538# 00	91	-.17452260#-01	157	-.92050492# 00
25	.90630784# 00	92	-.34899320#-01	158	-.92718384# 00
26	.89879410# 00	93	-.52335800#-01	159	-.93358048# 00
27	.89100660# 00	94	-.69756300#-01	160	-.93969264# 00
28	.88294756# 00	95	-.87155640#-01	161	-.94551856# 00
29	.87461970# 00	96	-.10452826# 00	162	-.95105660# 00
30	.86602538# 00	97	-.12186922# 00	163	-.95630482# 00
31	.85716732# 00	98	-.13917298# 00	164	-.96126166# 00
32	.84804812# 00	99	-.15643438# 00	165	-.96592574# 00
33	.83867058# 00	100	-.17364816# 00	166	-.97029576# 00
34	.82903752# 00	101	-.19080892# 00	167	-.97437008# 00
35	.81915200# 00	102	-.20791168# 00	168	-.97814754# 00
36	.80901700# 00	103	-.22495106# 00	169	-.98162726# 00
37	.79863552# 00	104	-.24192198# 00	170	-.98480776# 00
38	.78801070# 00	105	-.25881904# 00	171	-.98768836# 00
39	.77714600# 00	106	-.27563732# 00	172	-.99026810# 00
40	.76604436# 00	107	-.29237176# 00	173	-.99254616# 00
41	.75470962# 00	108	-.30901694# 00	174	-.99452200# 00
42	.74314488# 00	109	-.32556816# 00	175	-.99619478# 00
43	.73135374# 00	110	-.34202012# 00	176	-.99756410# 00
44	.71933974# 00	111	-.35836792# 00	177	-.99862960# 00
45	.70710680# 00	112	-.37460648# 00	178	-.99939076# 00
46	.69465832# 00	113	-.39073112# 00	179	-.99984768# 00
47	.68199830# 00	114	-.40673654# 00	180	-.10000000# 01
48	.66913062# 00	115	-.42261818# 00		
49	.65605906# 00	116	-.43837122# 00		
50	.64278766# 00	117	-.45399056# 00		
51	.62932038# 00	118	-.46947162# 00		
52	.61566138# 00	119	-.48480960# 00		
53	.60181502# 00	120	-.50000006# 00		
54	.58778524# 00	121	-.51503808# 00		
55	.57357646# 00	122	-.52991924# 00		
56	.55919278# 00	123	-.54463906# 00		
57	.54463906# 00	124	-.55919292# 00		
58	.52991924# 00	125	-.57357646# 00		
59	.51503808# 00	126	-.58778524# 00		
60	.49999992# 00	127	-.60181502# 00		
61	.48480954# 00	128	-.61566152# 00		
62	.46947150# 00	129	-.62932038# 00		
63	.45399042# 00	130	-.64278766# 00		
64	.43837110# 00	131	-.65605906# 00		
65	.42261818# 00	132	-.66913062# 00		
66	.40673654# 00				

Abb. 8. Rechenresultate *cosi*

LITERATURVERZEICHNIS

Bücher und Sammelbände

[1] F. L. Bauer, J. Heinhold, K. Samelson und R. Sauer, Moderne Rechenanlagen, Stuttgart 1965.

[2] R. Baumann, ALGOL-MANUAL der ALCOR-Gruppe, München-Wien 1965.

[3] B. V. Bowden (Herausgeber), Faster than thought, a symposium on digital machines, London 1953.

[4] L. Collatz, Funktionalanalysis und numerische Mathematik, Berlin-Göttingen-Heidelberg 1964.

[5] R. Goodman (Herausgeber), Annual review in automatic programming, Oxford-London-New York-Paris, Vol. 1, 1960; Vol. 2, 1961; Vol. 3, 1963; Vol. 4, 1964.

[6] B. W. Gnedenko, W. S. Koroljuk und J. L. Justschenko, Elemente der Programmierung, Leipzig 1964 (Übersetzung aus dem Russischen).

[7] F. R. Güntsch, Einführung in die Programmierung digitaler Rechenautomaten, 2. Aufl., Berlin 1963.

[8] W. Hoffmann (Herausgeber), Digitale Informationswandler, Braunschweig 1962.

[9] A. M. Jaglom und I. M. Jaglom, Wahrscheinlichkeit und Information, 3. Aufl., Berlin 1967 (Übersetzung aus dem Russischen).

[10] W. Kämmerer, Ziffernrechenautomaten, 3. Aufl., Berlin 1963.

[11] I. O. Kerner und G. Zielke, Einführung in die algorithmische Sprache ALGOL, Leipzig 1966.

[12] A. I. Kitow und N. A. Krinizki, Elektronische Digitalrechner und Programmierung, Leipzig 1962 (Übersetzung aus dem Russischen).

[13] W. Knödel, Programmieren von Ziffernrechenanlagen, Wien 1961.

[14] K. Nickel, ALGOL-Praktikum, Karlsruhe 1964.

[15] B. Randell und L. J. Russell, ALGOL 60 Implementation, London-New York 1964.

[16] K. Steinbuch (Herausgeber), Taschenbuch der Nachrichtenverarbeitung, Berlin-Göttingen-Heidelberg 1962.

[17] H. V. Wilkes, D. J. Wheeler und S. Gill, The preparation of programs for an electronic digital computer, Reading, Mass., 1951.

[18] R. Zurmühl, Praktische Mathematik für Ingenieure und Physiker, 4. Aufl., Berlin-Göttingen-Heidelberg 1963.

Zeitschriften

Die zitierten Zeitschriften enthalten laufend Algorithmen in ALGOL 60.

[19] Communications of the Association for Computing Machinery, New York.

[20] Nordisk Tidskrift for Informationsbehandling, Kopenhagen.

[21] Numerische Mathematik, Berlin-Göttingen-Heidelberg.

Originalarbeiten

[22] K.-H. Bachmann, Einige Besonderheiten des Dresdner Rechenautomaten D 1, Nachrichtentechn. Fachber. **4**, 90–91 (1956).

[23] K.-H. BACHMANN, Lösung algebraischer Gleichungen nach der Methode des stärksten Abstiegs, Z. angew. Math. Mech. **40**, 132–135 (1960).

[24] J. W. BACKUS u. a., The FORTRAN automatic coding system, Proc. West. Joint Comp. Conf., Los Angeles 1957, 188–198.

[25] J. W. BACKUS, The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM conference, Proc. intern. Conf. Inf. Proc., UNESCO, Paris 1959, 125–132.

[26] F. L. BAUER und K. SAMELSON, Maschinelle Verarbeitung von Programmsprachen, in [8], 227–268.

[27] R. BAUMANN, ALGOL-Manual der ALCOR-Gruppe, Elektronische Rechenanlagen **3**, 206–212, 259–265 (1961); **4**, 71–85 (1962).

[28] H. BOTTENBRUCH, Programmierung digitaler Systeme, in [16], 1330–1398.

[29] N. CHOMSKY, Three models for the description of language, IRE Transactions on Information theory IT-2, 113–124 (1956).

[30] Y. CHU, An ALGOL-like computer design language, Comm. ACM **8**, 607–615 (1965).

[31] L. COLLATZ, Das Hornersche Schema bei komplexen Wurzeln algebraischer Gleichungen, Z. Angew. Math. Mech. **20**, 235–236 (1940).

[32] E. W. DIJKSTRA, Recursive programming, Numer. Math. **2**, 312–318 (1960).

[33] J. EICKEL, M. PAUL, F. L. BAUER und K. SAMELSON, A syntax controlled generator of formal language processors, Comm. ACM **6**, 451–455 (1963).

[34] M. ENGELI, Automatisierte Behandlung elliptischer Randwertprobleme, Dissertation Zürich 1962.

[35] F. R. GÜNTSCH und W. HÄNDLER, Zur Simultanarbeit bei Digitalrechnern, Elektronische Rechenanlagen **2**, 117–128 (1960).

[36] W. HÄNDLER, Digitale Universalrechenautomaten in [16], 1009–1158.

[37] U. HILL, H. LANGMAACK, H. R. SCHWARZ und G. SEEGMÜLLER, Efficient handling of subscripted variables in ALGOL 60 compilers, Proc. Symp. on symbolic languages in data processing, Rom 1962, 331–340.

[38] W. HOFFMANN, Entwicklungsbericht und Literaturzusammenstellung über Ziffern-Rechenautomaten, in [8], 650–717.

[39] E. T. IRONS, The structure and use of the syntax directed compiler, in [5], Vol. 3, 207–227.

[40] L. KALMÀR, On a digital computer which can be programmed in a mathematical formula language, II. Magyar Matematikai Kongressuzs, Budapest 1960.

[41] W. KÄMMERER, Ziffernrechenautomat mit Programmierung nach mathematischem Formelbild, Jenaer Jahrbuch 1959/II.

[42] P. J. LANDIN, A correspondence between ALGOL 60 and Church's Lambda-notation, Comm. ACM **8**, 89–101, 158–165 (1965).

[43] N. J. LEHMANN, Bericht über den Entwurf eines kleinen Rechenautomaten an der Technischen Hochschule Dresden, Ber. Mathematikertagung, Berlin 1953, 262–270.

[44] P. NAUR (Herausgeber), Revised report on the algorithmic language ALGOL 60, Numer. Math. **4**, 420–453 (1963). Deutsche Übersetzung, Berlin 1966.

[45] W. L. VAN DER POEL (Herausgeber), Report on SUBSET ALGOL 60 (IFIP), Numer. Math. **6**, 454–458 (1964).

[46] W. L. VAN DER POEL (Herausgeber), Report on input-output procedures for ALGOL 60, Numer. Math. **6**, 459–462 (1964). Deutsche Übersetzung, Berlin 1966.

[47] W. L. VAN DER POEL, Microprogramming and trickology, in [8], 269–311.

[48] H. G. RICE, Recursion and iteration, Comm. ACM **8**, 114–115 (1965).

[49] H. RUTISHAUSER, Automatische Rechenplanfertigung bei programmgesteuerten Rechenmaschinen. Mitt. Inst. Angew. Math. ETH Zürich Nr. 3, Basel 1952.

[50] H. RUTISHAUSER, Maßnahmen zur Vereinfachung der Programmierung, Nachrichtentechn. Fachber. **4**, 26–30 (1956).

[51] K. SAMELSON, Probleme der Programmierungstechnik, Nachrichtentechn. Fachber. **4**, 139–140 (1956).

[52] K. SAMELSON, Probleme der Programmierungstechnik, Aktuelle Probleme der Rechentechnik, Berlin 1957, 61–68.

[53] K. SAMELSON und F. L. BAUER, Maßnahmen zur Erzielung kurzer und übersichtlicher Programme für Rechenautomaten, Z. Angew. Math. Mech. **34**, 262–272 (1954).

[54] K. SAMELSON und F. L. BAUER, Sequentielle Formelübersetzung, Elektronische Rechenanlagen **1**, 176–182 (1959).

[55] H. SCHECHER, Maßnahmen zur Vereinfachung von Rechenplänen, Z. Angew. Math. Mech. **36**, 377–395 (1956).

[56] J. H. WILKINSON, The evaluation of the zeros of ill-conditioned polynomials, Numer. Math. **1**, 150–180 (1959).

[57] K. ZUSE, Über den allgemeinen Plankalkül als Mittel zur Formulierung schematisch-kombinativer Aufgaben, Arch. Math. **1**, 441–449 (1948/49).

[58] Fachnormenausschuß Informationsverarbeitung im Deutschen Normenausschuß, ALGOL-Wörterbuch, Elektronische Datenverarbeitung 1963, 115–117.

NAMEN- UND SACHVERZEICHNIS