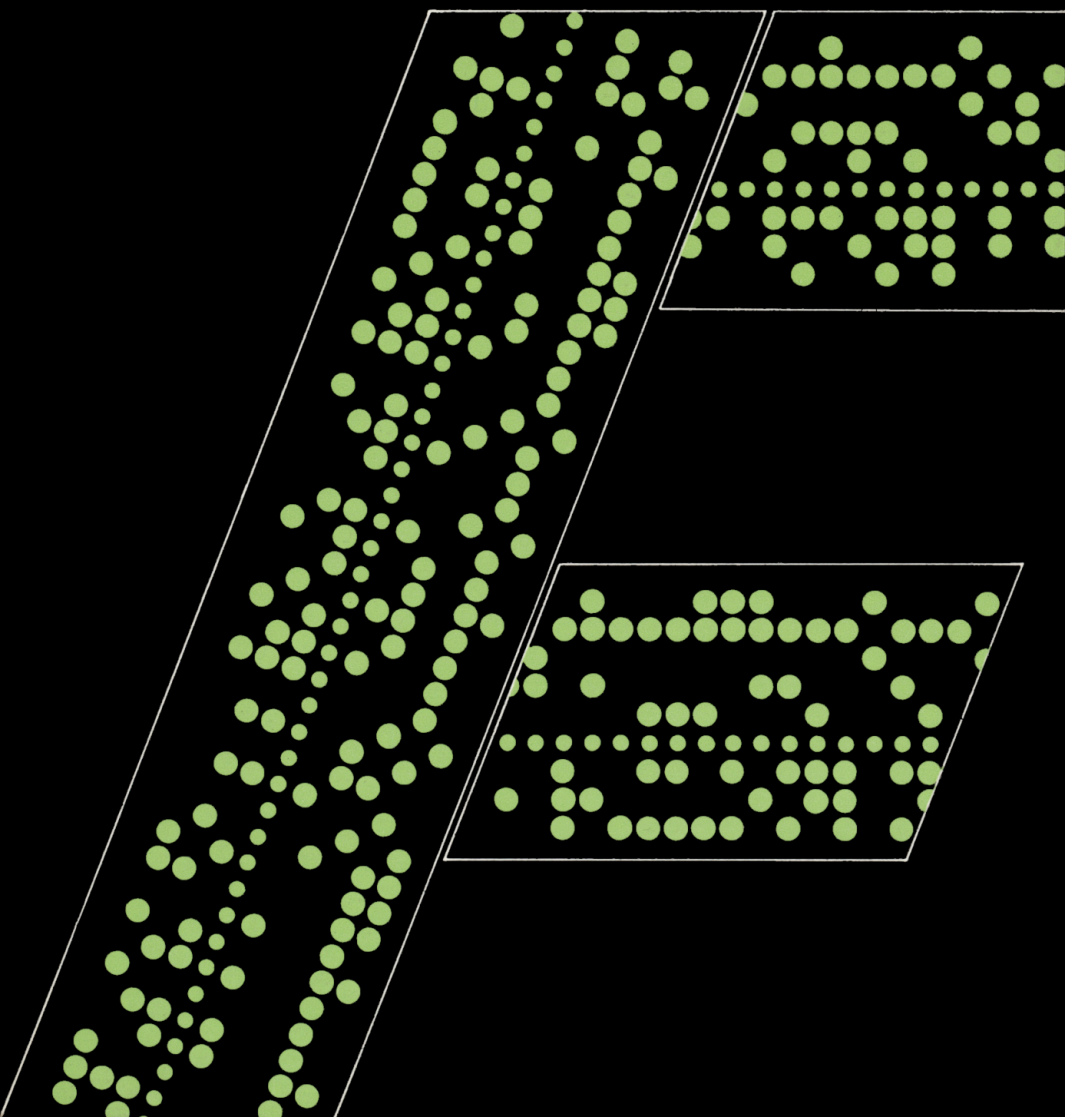
Programmierung und Nutzung von Rechenanlagen

Teil 3: FORTRAN



Programmierung und Nutzung von Rechenanlagen

Teil 3: FORTRAN

2. Auflage

Verlag Die Wirtschaft Berlin

Autoren:

Dr. Karl Hantzschmann, Dresden

Dipl.-Math. Hans-Peter Leidhold, Leipzig

Leiter des Autorenkollektivs:

Dr. Jürgen Bormann, Dresden

Lektor:

Gerhard Ziese, Berlin

1055 Berlin, Am Friedrichshain 22
Lizenz-Nr. 122; Druckgenehmigungs-Nr. 195/505/75
ES 21 A/LSV 0394
Einbandentwurf: U. Hilbert, Berlin
Printed in the German Democratic Republic
Schreibsatz: Verlag Die Wirtschaft, Berlin
Druck: Druckerei Tribüne, Berlin
Bestell-Nr. 674 747 6
EVP 4,50 Mark

Inhaltsverzeichnis

4. Einführung in die Programmiersprache FORTRAN

4.0. Einleitung

Nachdem Sie im Abschnitt „Beschreibung eines Programmablaufes" (vgl. 1.3.) die prinzipiellen Möglichkeiten und Besonderheiten der Arbeitsweise mit Programmiersprachen kennengelernt haben, sollen Sie in diesem Abschnitt des Lehrbuches einen Vertreter dieser Programmiersprachen, die Sprache FORTRAN, anwenden lernen.
FORTRAN („FORmula TRANslator") wird ähnlich wie ALGOL 60 vorwiegend für die Programmierung wissenschaftlich-technischer Aufgabenstellungen angewendet und hat sich im praktischen Einsatz hervorragend bewährt. Die kontinuierliche Entwicklung dieser Sprache führte vom ersten Übersetzungsprogramm Mitte der fünfziger Jahre über Zwischenvarianten zu FORTRAN IV, das gegenwärtig eine der am weitesten verbreiteten Programmiersprachen sein dürfte. 1964 gipfelte diese Entwicklung in der standardisierten Festlegung der Sprache FORTRAN in Anlehnung an FORTRAN IV. Parallel dazu entstand in Annäherung an die Möglichkeiten von FORTRAN II die standardisierte eingeschränkte Variante für kleinere und mittlere Rechenanlagen Basic FORTRAN. Seitdem wurden auch diese Standardvarianten in einigen Punkten beträchtlich weiterentwickelt.
Ein Großteil der in der DDR eingesetzten EDVA verfügt über Möglichkeiten zur Nutzung dieser Sprache. Hierzu zählen insbesondere die meisten importierten Anlagen. FORTRAN gewinnt weiterhin erheblich an Bedeutung, weil auch im Rahmen des Einheitlichen Systems der Elektronischen Rechentechnik (ESER) der sozialistischen Länder diese Sprache neben anderen Programmiersprachen Berücksichtigung findet. Für die Nutzung der Anlagen ES 1020 und ROBOTRON 21 steht im Plattenbetriebssystem (DOS/ES) ein FORTRAN-Compiler für die heute allgemein übliche Variante eines Basis-FORTRAN zur Verfügung. Außerdem gehört FORTRAN zu den Systemunterlagen der im VEB Kombinat ROBOTRON entwickelten und produzierten Prozeßrechner.
Der in diesem Abschnitt des Hochschullehrbuches dargestellte Stoff ist deshalb auch in Anlehnung an die im DOS/ES realisierte Variante von Basis-FORTRAN ausgewählt worden. Die Darstellung umfaßt alle wesentlichen Elemente dieser Sprache. Einige nicht in den Lehrstoff aufgenommene Elemente sind in einem Anhang, der außerdem alle zum Text gehörenden Tabellen enthält, als Ergänzun-

gen aufgeführt. Bezüglich einiger nicht berücksichtigter unwesentlicher Details in den Sprachmöglichkeiten und der mit der Nutzung der ESER-Anlagen im DOS/ES zusammenhängenden Fragen (Anwendung des Betriebssystems, Grenzen für die Anwendung von Sprachelementen in Abhängigkeit von der Hauptspeicherkapazität u. a.) sei auf die Anwenderbeschreibung des VEB Kombinat ROBOTRON verwiesen. Im Lehrstoff sind neben den von pädagogischen Gesichtspunkten gewählten Bezeichnungen für die Elemente der Sprache die allgemein üblichen Begriffe mit Schlüsselwörtern in Klammern hinzugefügt. Letztere werden auch in der ESER-Sprachbeschreibung benutzt. Das Erlernen von Basis-FORTRAN in der ESER-Variante soll dazu beitragen, die effektive Nutzung des Einheitlichen Systems der Elektronischen Rechentechnik in der Studentenausbildung bestmöglich vorzubereiten. Wegen der Besonderheiten anderer in der DDR genutzter FORTRAN-Implementierungen, speziell der Abweichungen von dieser ESER-Variante, sei auf die Anwenderdokumentationen der jeweiligen Anlage verwiesen.
Die Darstellung des Lehrstoffs erfolgt in einer Form, die ein leichtes Erlernen der Programmiersprache und einen guten Zugang zur gebräuchlichen FORTRAN-Literatur ermöglicht. Von einer formalen Darstellung der Syntax (Grammatik der Sprache) wurde weitgehend Abstand genommen (vgl. hierzu 4.1.2.), die semantische (inhaltliche) Interpretation der Sprachelemente wird ausführlich unter Einbeziehung zahlreicher Beispiele erläutert. Soweit die Verständlichkeit der Darstellung nicht darunter leidet, erscheinen im Text häufig vorkommende Begriffe im Wiederholungsfall durch Abkürzungen (ohne Anhängen von Deklinationsendungen). Um das Erlernen der Sprache zu erleichtern, sind neben den zur Erläuterung des Lehrstoffes im Text enthaltenen Beispielen unter gesonderten Gliederungspunkten jeweils mehrere vollständige Programmbeispiele zusammengestellt. Dieses Prinzip wird von Anfang an beachtet. Deshalb wurden die dazu benötigten Elemente einer einfachen Ein- und Ausgabe und Fragen der Struktur einfacher Programme gegenüber der eigentlichen Stoffolge vorgezogen. Die Beispiele sind nach methodischen Gesichtspunkten zur Veranschaulichung bestimmter Sprachelemente aufgebaut und stellen nicht in jedem Falle die effektivsten Programme dar. Da man Programmieren nur durch aktive Beschäftigung mit den Sprachelementen und nicht lediglich durch Lesen der Sprachbeschreibung erlernen kann, sei Ihnen mit Nachdruck empfohlen, die angegebenen Beispiele intensiv zu studieren und die sich anschließenden Aufgaben selbständig ohne vorzeitige Zuhilfenahme der Lösungen zu bearbeiten. Alle Programmbeispiele und die in den Lösungen enthaltenen Programme wurden auf einer Anlage getestet. Die mit [Z] gekennzeichneten Aufgaben sind als Zusatzaufgaben zur weiteren fakultativen Vertiefung Ihrer Kenntnisse gedacht. Entsprechend der unterschiedlichen Bedeutung, die bestimmte FORTRAN-Elemente für technisch orientierte gegenüber ökonomisch orientierten Fachrichtungen haben, wird im Einklang mit dem Stufenprogramm für die Hochschul- und Universitätsausbildung auf dem Gebiet der EDV/IV (Stufe 3) folgende Stoffzusammenstellung empfohlen:

Technisch orientierte Fachrichtungen studieren gründlich 4.1., 4.2., 4.3. (außer 4.3.2.2.), 4.4.4.1. und 4.5.
Ökonomisch orientierte Fachrichtungen studieren gründlich 4.1., 4.2., 4.3., 4.4., 4.5.3. und 4.5.4.
Der jeweils übrige Stoff ist nur überblicksmäßig zu behandeln.

In Übereinstimmung damit ist auch ein Teil der Aufgaben jeweils einer Fachrichtung (Kennzeichnung [T] oder [Ö]) zugeordnet. Im Gegensatz zu dem gründlich zu studierenden Lehrstoff besteht das Ziel eines überblicksmäßigen Studierens einzelner Gliederungspunkte nicht im aktiven Beherrschen der darin enthaltenen Sprachelemente; Sie sollen diese Elemente in FORTRAN-Programmen aber wenigstens verstehen können. Die mit [Z] gekennzeichneten Kapitel sind als fakultativer zusätzlicher Lehrstoff gedacht, mit dem Sie nach Abschluß der obligatorischen Programmiersprachenausbildung Ihre Kenntnisse über Basis-FORTRAN erweitern können.

4.1. Grundelemente

4.1.1. FORTRAN-Zeichen und Schlüsselwörter

Zur Formulierung eines Programmes in einer Programmiersprache bedarf es wie in jeder natürlichen Sprache eines Alphabets, dessen Bestandteile nach bestimmten Regeln zu sinnvollen Formulierungen gruppiert werden. In der Programmiersprache FORTRAN stehen dazu die folgenden *FORTRAN-Zeichen* zur Verfügung:

Buchstaben:

Großbuchstaben des lateinischen Alphabets A, B, C, ..., Z
Währungszeichen $

Ziffern:

Ø, 1, 2, 3, 4, 5, 6, 7, 8, 9

Sonderzeichen:

, (Komma)	+ (plus)
. (Punkt)	– (minus)
= (gleich)	* (Stern)
((öffnende Klammer)	/ (Schrägstrich)
) (schließende Klammer)	␣ (Leerzeichen).
' (Apostroph)	

Buchstaben und Ziffern bilden die Menge der alphanumerischen Zeichen. In der Basis-FORTRAN-Implementierung im DOS/ES stellt $ einen Buchstaben dar. Achten Sie besonders auf die eindeutige Unterscheidung der Zeichen O (Buchstabe O) und Ø (Ziffer 0). Das Leerzeichen wollen wir in den folgenden Darlegungen im Interesse einer eindeutigen Schreibweise durch das Symbol ␣ charakterisieren.

Einige Wortbildungen aus diesem Zeichenvorrat fungieren als *FORTRAN-Schlüsselwörter.* Sie werden zur Charakterisierung von Anweisungen benötigt und sind im Anhang (A 4/2) vollständig zusammengestellt. Ihre Erläuterung soll im jeweiligen Anwendungsfall erfolgen, soweit im Lehrstoff darauf Bezug genommen wird. Als Beispiele seien genannt:

```
INTEGER     DO     GOTO
FUNCTION    IF     END
```

Z

4.1.2. Bemerkungen zur syntaktischen Beschreibung

Im Interesse einer leichten Verständlichkeit und in Anlehnung an die originale Definition von FORTRAN werden im gesamten Lehrstoff die Sprachelemente explizit charakterisiert. Auf die Darstellung der Syntax (Grammatik der Sprache) in der BACKUS-Notation wurde aus diesen Gründen bewußt verzichtet.
Während die einfachen Sprachelemente in elementarer Weise rein verbal definiert werden, sind die höheren Sprachelemente von Basis-FORTRAN durch ihre kompletten Endkonstruktionen eingeführt. Zu ihrer Charakterisierung bedienen wir uns der folgenden Begriffe:

Notationskonstante (NK)

Eine Notationskonstante ist entweder eines der FORTRAN-Zeichen I F H X E A ( ) , = . oder ein Schlüsselwort, das in der angegebenen Form an der entsprechenden Stelle gesetzt werden muß.
Beispiele: X , INTEGER

Notationsvariable (NV)

Eine Notationsvariable besteht aus einem oder einer Folge von Kleinbuchstaben ohne oder mit (tiefgestelltem) Index und muß vom Programmierer entsprechend ihrer eigenen syntaktischen Definition durch eine spezielle Realisierung ersetzt werden. Als Indizes kommen natürliche Zahlen oder Kleinbuchstaben vor.
Beispiele: m, m_1, m_k, fl

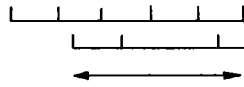
Liste variabler Länge

Sie hat die Form LE, LE, ... , LE
Alle Listenelemente LE sind entweder von der Gestalt NV oder (NV) oder NV (NV) (NV sei indizierte Notationsvariable).
Die Anzahl der Listenelemente ist im Rahmen gewisser Grenzen beliebig und größer als Null.
Beispiele: $n_1, n_2, \ldots, n_i$ $\quad$ $n_1\,(i_1), n_2\,(i_2), \ldots, n_k\,(i_k)$

Höhere Sprachelemente von Basis-FORTRAN werden nun durch eine syntaktische Kette, d. h. eine Aufeinanderfolge von NK und | oder NV und | oder Listen variabler Länge, in ihrer expliziten Form definiert. Die Definitionen der NV sind an den benötigten Stellen im Text im allgemeinen verbal angegeben.

Beispiel

GOTO (m_1, m_2, ..., m_n), i



Während vom Programmierer für die NV m_1, m_2, ..., m_n, i spezielle Realisierungen einzusetzen sind, müssen bei der Anwendung dieses Sprachelementes die NK in der fest vorgegebenen Form und Stellung belassen werden, z.B.
GOTO (2,14,8,3),K (s. 4.3.1.3.).

Zur vollständigen theoretischen Beschreibung der Syntax einer Programmiersprache verwendet man in der Theorie der Sprachen eine *Metasprache*. Diese bedient sich außer solchen Elementen wie Notationskonstante und Notationsvariable noch einiger zusätzlicher metasprachlicher Zeichen, die natürlich keine Bestandteile der zu beschreibenden Sprache sind. Wir wollen uns dies an einem einfachen Beispiel in der BACKUS-Notation ansehen:

marke ::= ziffer | marke ziffer
ziffer ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Unter Benutzung der metasprachlichen Zeichen

::= (definiert als)
| (oder)

werden die Sprachelemente ziffer und marke definiert. Eine Marke ist demnach eine unmittelbare Aufeinanderfolge von Ziffern. Zur vollständigen Charakterisierung des Sprachelementes marke kommt noch seine verbal gegebene Semantik hinzu, die Aufschluß über die inhaltliche Bedeutung und die Besonderheiten bei der Verwendung gibt. Erst Syntax und Semantik zusammen charakterisieren das Sprachelement marke eindeutig (s. 4.1.5.).

4.1.3. Namen und implizite Typfestlegung

Alle Größen eines Programms (Variable, Felder, Funktionen, Subruotinen), auf die erst an späterer Stelle eingegangen werden kann, müssen in geeigneter Weise bezeichnet sein. Dazu werden *Namen* eingeführt, für deren Bildung folgende Regeln verbindlich sind:

Es dürfen bis zu sechs Buchstaben und Ziffern in beliebiger Folge aneinandergereiht werden.
Das erste Zeichen muß ein Buchstabe sein.

Einige Beispiele sollen das erläutern:

korrekte Namen	unzulässige Namen
A	1A
M1Ø	INTEGRAL
SUMME	22
ALPHA	M-1
S11	

Die Gestaltung von Namen obliegt völlig dem Ermessen des Programmierers, praktisch ist ein Anlehnen an die bei der Formulierung des Algorithmus verwendete Symbolik zu empfehlen.
In den Fällen, wo Namen solche Programmgrößen repräsentieren, denen in irgendeiner Weise während der Programmabarbeitung Werte zugewiesen werden, muß für diese Größen eine *Typfestlegung* erfolgen. Der Typ gibt an, von welcher Art der Wert der durch den Namen festgelegten einfachen Variablen, der Funktion oder des festgelegten Feldelementes ist. Wenn wir künftig allgemein vom Typ eines Namens sprechen, wollen wir darunter immer den Typ der durch ihn bezeichneten Programmgröße verstehen. FORTRAN erlaubt dazu als einfachste Möglichkeit eine *implizite Typfestlegung* für die beiden Grundtypen ganzzahlig und reell (Standard-Typfestlegung):

Ist über den Typ eines Namens explizit nichts anderes festgelegt (s. 4.2.5.), entscheidet der erste Buchstabe des Namens über den Typ. Mit I, J, K, L, M oder N beginnende Namen sind vom Typ INTEGER, alle anderen vom Typ REAL.

Danach stellen beispielsweise I, LAMBDA, M1 Namen für Variable mit ganzzahligen Werten, X, SUMME, BETA dagegen Namen für Variable mit reellen Werten einfacher Genauigkeit dar. Es gibt in FORTRAN keine reservierten Namen, d. h. auch die Zeichenfolgen der Schlüsselwörter könnten als Namen verwendet werden, was allerdings nicht zu empfehlen ist.

4.1.4. Zahlendarstellungen

In nahezu allen Programmen kommen neben den variablen Größen konstante Programmgrößen vor, deren Werte sich während der Abarbeitung eines Programms nicht ändern. Derartige Programmkonstanten können vom Typ ganzzahlig, reell oder doppeltgenau sein.
Eine *ganzzahlige Konstante* k besteht aus einer Folge von Ziffern, vor der ein Vorzeichen stehen darf. Der von der jeweiligen Anlage abhängige Wertebereich ist für die ESER-Anlagen im DOS/ES durch

$-2^{31} \leqslant k \leqslant 2^{31} - 1 \quad (2^{31} - 1 = 2\,147\,483\,647)$

gegeben.

Beispiele für ganzzahlige Konstanten:

```
     5    -2
+12345     Ø
```

Keine ganzzahligen Konstanten sind dagegen:

26.	Dezimalpunkt nicht erlaubt
8ØØØØØØØØØ	Zahlenbereich überschritten

Reelle Konstanten sind durch einen Dezimalpunkt oder den Buchstaben E als Kennzeichen für den Exponenten zur Basis 10 oder beides gekennzeichnet. Die Darstellungsmöglichkeiten lassen sich symbolisch in folgender Form angeben:

i.d

i.dEj

kEj

i,j,k sind dabei beliebige ganzzahlige Konstanten, d bezeichnet die Ziffernfolge des Dezimalteiles. Es ist zulässig, daß die Positionen i oder d leer sind. Die Darstellung mEj mit m = i.d oder m = k ist gleichbedeutend mit $m * 10^j$. Bei ESER-Anlagen können reelle Konstanten betragsmäßig zwischen ungefähr

$5{,}42 \cdot 10^{-79}$ und $7{,}21 \cdot 10^{75}$

liegen; hinzu kommt als Sonderfall die Zahl 0. Eine reelle Zahl (einfacher Genauigkeit) besitzt bis zu 7 gültige Dezimalziffern.

Beispiele

FORTRAN-Darstellung	Zahlwert
+3.4	3,4
.25	0,25
-Ø.ØØØ8	−0,0008
-12.	−12
Ø.Ø	0
3.4E1	34
.2E-2Ø	$0{,}2 \cdot 10^{-20}$
27.E-2	0,27
-3E+1	−30

Dagegen sind folgende Schreibweisen fehlerhaft:

3.14E-1.4	Exponent nicht ganzzahlig
E2	Ziffern vor E fehlen
6,27	Komma nicht erlaubt
1.1-1	Exponentenangabe erfordert E
1.23456789	Anzahl der Ziffern zu groß

FORTRAN erlaubt die Verarbeitung von reellen Konstanten mit erhöhter Stellenzahl. *Doppeltgenaue Konstanten* sind durch ein D als Exponentensymbol gekennzeichnet. Basis-FORTRAN im DOS/ES erlaubt dabei, maximal 16 gültige Dezimalziffern zu berücksichtigen.

Beispiele

FORTRAN-Darstellung	Zahlwert
8.14DØ2	814
314.159265359D-2	π mit 12 Ziffern
Ø.ØD-Ø	0

Ganze Zahlen werden intern als binäre Festkommazahlen in einem Wort, reelle Zahlen mit einfacher bzw. doppelter Genauigkeit intern als Gleitkommazahlen in einem Wort bzw. Doppelwort dargestellt (vgl. Abschnitt „Aufbau und Arbeitsweise von EDVA").

4.1.5. Marken

Jede Anweisung eines FORTRAN-Programms kann durch eine *Marke* besonders gekennzeichnet werden. Marken bestehen aus maximal 5 Ziffern und werden vor die zu markierenden Anweisungen geschrieben. Selbstverständlich muß die Zuordnung der Marken in einem Programm eindeutig sein. Führende Nullen sind bei der Bildung von Marken ohne Bedeutung, Ziffernfolgen mit dem Wert Null sind als Marke nicht zugelassen.

Beispiele

2
111
ØØ3 } äquivalent
3 }

Dagegen sind als Marken nicht erlaubt:

112233	mehr als 5 Ziffern
A1	keine Buchstaben zulässig
-8	keine Sonderzeichen zulässig
ØØ	Ziffernfolge mit Wert Null

4.1.6. Programmformular

Die Folge der zu einem Programm gehörenden *Anweisungen* (vgl. 4.2.6.) muß in ein Programmformular eingetragen werden. Jede Zeile dieses Formulars ist in eine Lochkarte abzulochen. Im Anhang (A 4/1) ist die für Basis-FORTRAN im DOS/ES verwendete Lochkartendarstellung der FORTRAN-Zeichen angegeben.
Das Programmformular (Bild 1) ist analog zur Lochkarte in 80 Spalten eingeteilt. Beim Eintragen der Programminformationen sind die folgenden Vereinbarungen zu beachten.
Jede Anweisung beginnt auf einer neuen Zeile und kann die Spalten 7 bis 72 belegen. In Spalte 6 steht in der Regel ein Leerzeichen oder eine Null. Benötigt die Dar-

FORTRAN
Programmformular

Programm _____
_____
Betrieb _____

Bearbeiter _____
Datum _____
Geprüft _____
Blatt _____ von _____

Marke		Anweisungen	Problem / Folge
2 4	6	8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72	74 76 78 80
C		TABELLIERUNG VON Y=EXP(-X**2/2)	1
		WRITE(3,10)	2
10		FORMAT(3(H1,TABELLIERUNG VON EXP(-X**2/2)))	3
		READ(1,11)H	4
11		FORMAT(F5.2)	5
		X=0.	6
		Y=EXP(-X**2.5)	7
		WRITE(3,12)X,Y	8
12		FORMAT(2(1X,F7.2,4X,F8.6))	9
		X=X+H	10
		IF(Y-1.E-5)2,1,1	11
2		STOP	12
		END	13
1 3 5		7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71	73 75 77 79

Bild 1

stellung einer Anweisung mehr als eine Zeile, müssen die Fortsetzungszeilen in Spalte 6 durch ein von Null und Leerzeichen verschiedenes beliebiges Zeichen gekennzeichnet werden. Zwischen die einzelnen Anweisungen des Programms können zur Verbesserung der Lesbarkeit und des Verständnisses Bemerkungen eingefügt werden, die auf das Programm selbst ohne Einfluß sind. Derartige Kommentarzeilen sind durch ein C in Spalte 1 als solche zu kennzeichnen und bestehen aus einer beliebigen Folge von FORTRAN-Zeichen. Wenn Anweisungen zu markieren sind, so erscheint die eindeutig zugeordnete Marke beliebig in den Spalten 1 bis 5 der Anfangszeile der entsprechenden Anweisung. Leerzeichen dürfen im Programmformular im Interesse einer guten Lesbarkeit beliebig verwendet werden. Die Spalten 73 bis 80 sind für das vom Compiler erzeugte Objektprogramm ohne Einfluß und können organisatorische Angaben zum Programm (z. B. Name des Programms, Kartennumerierung) aufnehmen.

4.1.7. Aufgaben

4.1.7./1

Prüfen Sie die angegebenen Namen auf syntaktische Richtigkeit, wobei für die als fehlerhaft erkannten Fällen die Begründung anzugeben ist.

WURZEL	ØA1	SUMME	S22
X-1	RHO␣4	THETA8	01
MAX12	ERGEBNIS	M.N	$

4.1.7./2

a) Es ist zu untersuchen, welche der Zeichenfolgen als FORTRAN-Konstanten nicht erlaubt sind (Begründung). Für die mit E und D richtigen Konstantendarstellungen ist die mathematisch übliche Zahlendarstellung anzugeben.

222.Ø	+Ø.ØE-Ø	Ø,ØØ
7.E-2	1E-A	D1
7E-2	1Ø.1.2	7.04
8Ø.2D+Ø8	.Ø3	-2.1E-Ø1
-Ø867219	Ø.Ø3	

b) Geben Sie für die folgenden Zahlen eine FORTRAN-gerechte Darstellung unter Verwendung des Exponentensymbols E an.

0,572	$88{,}23 \cdot 10^2$	36
$-33 \cdot 10^{-18}$	0,000044	− 0

4.2. Lineare Programme

Aufgabe dieses Kapitels soll es sein, die Elemente von Basis-FORTRAN kennenzulernen, mit denen bereits einfache lineare Programme geschrieben werden können. Unter linearen Programmen wollen wir dabei solche Programme verstehen, bei denen die einzelnen Anweisungen strikt in ihrer textlichen Reihenfolge im Programm-

formular abgearbeitet werden, wobei weder Programmverzweigungen noch wiederholtes Abarbeiten bestimmter Programmteile auftreten sollen.

4.2.1. Arithmetische Ausdrücke

Das Analogon zur allgemein bekannten mathematischen Formelschreibweise stellen in FORTRAN die *Ausdrücke* dar. Sie verkörpern Regeln zur Berechnung von Werten, indem die angegebenen *Operatoren* auf die im Ausdruck enthaltenen *Operanden* angewendet werden. Beim Schreiben von FORTRAN-Ausdrücken sind als Besonderheiten gegenüber der üblichen mathematischen Schreibweise zu beachten:

Die Schreibweise ist streng linear, d. h. alle Bestandteile des Ausdrucks erscheinen in einer Zeile nebeneinander ohne Hoch- oder Tiefstellen von Exponenten, Divisoren und Indizes.
Der Multiplikationsoperator * darf nicht weggelassen werden.
Zwei aufeinanderfolgende Operatoren sind stets durch Klammern zu trennen.

Als Bestandteile der arithmetischen Ausdrücke sind erlaubt:

Operatoren		Operanden
+	Addition	Konstante
–	Subtraktion	Variable
*	Multiplikation	Funktionsaufrufe (s. 4.2.2. und 4.5.2.)
/	Division	eingeklammerte Ausdrücke
**	Potenzierung	

Gegenüber dem älteren Standard ist es heute im allgemeinen zugelassen, daß Operanden beliebigen Typs durch die genannten Operatoren verknüpft werden können. Bei der Berechnung derartiger gemischter Ausdrücke macht sich dann eine Konvertierung gewisser Operanden von einem Typ in einen anderen notwendig. Wenn bei der zweistelligen Operation A op B beide Operanden unabhängig voneinander sowohl vom Typ INTEGER als auch vom Typ REAL sein können, gilt für den Typ des Ergebnisses die Aussage: Das Ergebnis ist nur dann vom Typ INTEGER, wenn sowohl A als auch B vom Typ INTEGER sind. In allen anderen Kombinationen ist das Ergebnis vom Typ REAL. Eine vollständige Übersicht über die gegebenenfalls notwendigen, durch den Compiler zu realisierenden Konvertierungen und den Typ des Ergebnisses bei beliebig zugelassenen Operanden (einschließlich doppeltgenauer Operanden) ist im Anhang (A 4/3) zusammengestellt. Bei Anwendung der Potenzoperation ist zu beachten, daß bei Exponenten vom Typ REAL die Basis nicht negativ sein darf und bei der Basis Null ein Exponent größer als Null erforderlich ist.

FORTRAN-Schreibweise	übliche Schreibweise	Bemerkungen
A/(B*C) A/B/C	$\left.\begin{matrix}\\\end{matrix}\right\} \frac{A}{B \cdot C}$	Ergebnis REAL
C*(D+E)/(F+G)	$\frac{C \cdot (D + E)}{F + G}$	Ergebnis REAL
X	X	einzelner Operand als trivialer Ausdruck
-(Ø.7+R5)**3.5	$-(0{,}7 + R5)^{3{,}5}$	
R1*5**(-B)*SIGMA	$R1 \cdot 5^{-B} \cdot SIGMA$	Ergebnis REAL
3AB	$3 \cdot A \cdot B$	fehlerhaft: * fehlt
A**-3	A^{-3}	fehlerhaft: 2 aufeinanderfolgende Operatoren
(-8.17)**E	$(-8{,}17)^{E}$	fehlerhaft: Basis negativ bei REAL-Exponent
I+17/3.Ø	$I + \frac{17}{3}$	Ergebnis REAL
π*g*H	πgH	fehlerhaft: griechische und kleine lateinische Buchstaben
PI*RADIUS**2	$PI \cdot RADIUS^2$	Variable PI beinhaltet Zahlenwert π

Die Berechnung der arithmetischen Ausdrücke geschieht nach den für die arithmetischen Operationen üblichen Regeln unter Zugrundelegung der aktuellen numerischen Werte der Operanden. Im Normalfall erfolgt die Berechnung eines Ausdruckes von links nach rechts unter Beachtung notwendiger Typkonvertierungen und der Prioritäten

1. Klammern
2. Potenzierungen
3. Multiplikationen/Divisionen
4. Additionen/Subtraktionen.

Mit Hilfe der Klammern kann also die Reihenfolge der Auswertung wie üblich verändert werden.
Aufeinanderfolgende Potenzbildungen erfolgen abweichend von der genannten Reihenfolge von rechts nach links:

A**B**C bedeutet $A^{(B^C)}$.

4.2.2. Standardfunktionen

Zur Vereinfachung der Programmierarbeit besteht die Möglichkeit, die Werte von oft benötigten mathematischen Funktionen direkt mit Hilfe von *Standardfunktionen* bereitzustellen. So existieren beispielsweise Standardfunktionen COS,

EXP, SQRT zur Berechnung von Cosinus-, Exponentialfunktions- und Wurzelwerten für reelle Argumente. Standardfunktionen können unmittelbar als Operanden in Ausdrücken auftreten. Der Ausführung der vorgesehenen Operationen geht dann erst die Berechnung des Wertes der Standardfunktion voraus, indem automatisch ein entsprechendes Maschinenprogramm zur Durchrechnung des Algorithmus der benötigten Funktion aktiviert wird.
Der Funktionsaufruf wird im Ausdruck in der Form

$f(p_1, p_2, \ldots, p_n)$

geschrieben, wobei die p_i (i = 1(1)n) als Parameter die Argumente der Funktion angeben und f den Namen der Funktion symbolisiert.

Beispiele

FORTRAN-Schreibweise	übliche Schreibweise	Bemerkungen
`COS(X)**2`	$\cos^2(X)$	
`X+SQRT(Z-2.Ø)`	$X + \sqrt{Z-2}$	Schreibweise 2.Ø erspart interne Konvertierung
`IABS(I-J)`	$\lvert I - J \rvert$	Typ INTEGER } für Argument und
`ABS(X-Y)`	$\lvert X - Y \rvert$	Typ REAL } Funktionswert
`EXP(X)*COS(A*X)/ SQRT(1.Ø+SIN(B*X)**2)`	$\frac{e^X \cdot \cos AX}{\sqrt{1+\sin^2 BX}}$	
`AMAX1(A,B,C,D)`	max (A,B,C,D)	

Die Beispiele zeigen, daß als Argumente auch arithmetische Ausdrücke verwendet werden können.
Eine vollständige Zusammenstellung aller in Basis-FORTRAN vorhandenen Standardfunktionen mit Angaben zur Anzahl und zum Typ der Parameter und des Funktionswertes finden Sie im Anhang (A 4/4).

4.2.3. Ergibtanweisungen

Um den Wert eines berechneten Ausdrucks einer Variablen zuweisen zu können, bedarf es einer *Ergibtanweisung* in der Form

v = a.

a ist dabei der zu berechnende Ausdruck, v die Variable, nach der der Ergebniswert zur weiteren Verwendung übertragen werden soll. Dabei ist es nicht notwendig, daß der Typ des mit dem Ausdruck ermittelten Wertes mit dem der Variablen übereinstimmt. Analog zur Berechnung der gemischten Ausdrücke wird erforderlichenfalls eine Typkonvertierung zwischengeschaltet. Dies bedeutet bei der Konvertierung eines INTEGER-Wertes in einen REAL-Wert lediglich die Änderung der Zahlendarstellung in der Rechenanlage, in der umgekehrten Richtung dagegen

ein Herauslösen des ganzen Teils der reellen Zahl und dessen Umwandlung in eine INTEGER-Zahl. Analog wird verfahren, wenn für den Ausdruck doppeltgenaue Werte entstehen oder die Variable v für doppeltgenaue Werte vereinbart worden ist.

Beispiele

X=Ø X=Ø.Ø	Wert Null wird der reellen Variablen X zugeordnet (Darstellung als reelle Konstante erspart Konvertierung)
X=-X	Vorzeichenumkehr bei X
F=A*B*SIN(GAMMA)*Ø.5	
I=I+1	zum Wert von I wird 1 addiert und das Ergebnis wieder I zugeordnet (Erhöhung des Wertes von I um 1)
M=SUMME	ganzer Teil von SUMME ergibt M (SUMME vom Typ REAL, M vom Typ INTEGER)
K=17/3	Ergebnis ist K=5 (K vom Typ INTEGER)
X+Y=(3.5+A)/B	fehlerhaft: links vom Gleichheitszeichen keine Ausdrücke erlaubt
-T=U+V	
M1=M2=RHO**4	fehlerhaft: Mehrfachzuweisungen nicht möglich

4.2.4. Einfache Ein- und Ausgabe

Obwohl für die Programmierung von Ein- und Ausgabevorgängen ein gesondertes Kapitel vorgesehen ist, sollen bereits an dieser Stelle die einfachsten Möglichkeiten zusammengestellt werden. Das Kennenlernen von *Ein- und Ausgabeanweisungen* (E/A-Anweisungen) sowie der damit im Zusammenhang stehenden *Formatanweisung* wird uns schon jetzt in die Lage versetzen, komplette FORTRAN-Programme zu schreiben und damit die Übungsbeispiele effektiver zu gestalten. In 4.4. werden wir dann auf dieser Grundlage aufbauend die in Basis-FORTRAN bestehenden Möglichkeiten vervollständigen und eine detaillierte Darstellung der Ein- und Ausgabevorgänge bringen.
Zum Einlesen von Daten von einem externen Speichermedium in den Hauptspeicher steht die Anweisung

READ(d,f)l

zum Ausgeben die Anweisung

WRITE(d,f)l

zur Verfügung. Die mit den Schlüsselwörtern READ und WRITE beschriebenen Anweisungen sind ausführbare Anweisungen (zu den Begriffen ausführbare und nichtausführbare Anweisung vgl. 4.2.6.). Durch den Buchstaben d, für den entweder eine natürliche Zahl (positive ganzzahlige Konstante ohne Vorzeichen) oder eine Variable vom Typ INTEGER stehen kann, wird das am Transfer beteiligte externe Speichermedium symbolisiert, wobei diese Zuordnung natürlich anlagenabhängig ist. Wir wollen uns zunächst auf das Einlesen der Daten über einen

Kartenleser und das Ausdrucken der Daten über einen Zeilendrucker beschränken und dabei für alle Übungsbeispiele folgende Zuordnung treffen:

READ(1,f)l Einlesen von 80-spaltigen Lochkarten

WRITE(3,f)l Ausdrucken auf einem Zeilendrucker.

Mit dem Buchstaben f wird der Bezug zu einer den E/A-Anweisungen zugeordneten Formatanweisung hergestellt, wenn die zu übertragenden Daten in einer aufbereiteten Form gelesen oder ausgedruckt werden sollen. Die Art der Aufbereitung wird durch die Formatanweisung festgelegt, deren Marke in den E/A-Anweisungen unter f angegeben ist.
Die am Datentransfer beteiligten Programmgrößen werden in Form einer Liste l *(Übertragungsliste)* angegeben. Als Listenelemente, die durch Komma voneinander zu trennen sind, lassen wir vorerst nur die Variablen zu. So bewirken beispielsweise

`READ(1,1ØØ)X,SUMME,I,M1` das Einlesen von vier Werten, die in der durch die Formatanweisung mit der Marke 1ØØ beschriebenen Art auf einer Lochkarte gelocht sind und das Zuordnen dieser Werte zu den Variablen X,SUMME,I,M1 in der angegebenen Reihenfolge,

`WRITE(3,2ØØ)A,B,C` das Ausdrucken der Werte von A,B,C in einer Zeile mittels Zeilendrucker in der durch die Formatanweisung mit der Marke 2ØØ festgelegten Darstellungsform.

Die Menge der durch eine READ-Anweisung aus einer Lochkarte eingelesenen bzw. durch eine WRITE-Anweisung in eine Zeile gedruckten Daten wollen wir als *Datensatz* bezeichnen. Dieser Begriff wird später noch etwa verallgemeinert werden (s. 4.4.1. und 4.4.5.). Im folgenden wollen wir uns einige wichtige Möglichkeiten zur Datenaufbereitung mit Hilfe einer Formatanweisung ansehen.
Die mit dem Schlüsselwort FORMAT gebildete nichtausführbare Anweisung hat in der einfachsten Variante die Form

FORMAT $(b_1, b_2, \ldots, b_n)$

Die b_i (i = 1(1)n) sind *Datenfeldbeschreibungen (Formate)* zur Festlegung der Darstellungsform für die auf der Lochkarte gelochten Eingangsdaten bzw. für die auf einem Zeilendrucker auszudruckenden Daten. Wir wollen uns zunächst auf das Kennenlernen einiger wichtiger Formate beschränken.

a) I-Format

Das *I-Format* in der Form

Iw

dient der Festlegung, wie ganze Zahlen auf Lochkarten gelocht oder auf Papier ausgedruckt werden sollen.
Die Konstante w (natürliche Zahl aus $1 \leqslant w \leqslant 254$) bezeichnet dabei die Feldweite, d. h. die Anzahl der insgesamt zum Datenfeld gehörenden Zeichen.

Einige Beispiele sollen das I-Format zur Ein-/Ausgabe von ganzen Zahlen erläutern.

Format I4 zur Eingabe:

Zeichenfolge auf dem Eingabemedium (Datenfeld)	Wert der eingegebenen Zeichenfolge
3876	3876
␣+1Ø	10
␣−␣3	−3
2␣6␣	2060
−8␣8	−808
␣␣␣␣	0

w=4

Das Datenfeld muß selbstverständlich eine ganze Zahl enthalten. Führende Leerzeichen sind bedeutungslos, alle übrigen werden als Nullen interpretiert. Das Pluszeichen kann entfallen.

Format I4 zur Ausgabe:

Wert der auszugebenden Variablen	ausgegebene Zeichenfolge (Datenfeld)
27	␣␣27
−777	−777
−1	␣␣−1
0	␣␣␣Ø
11228	****

␣
w=4

Die auszugebende ganze Zahl darf einschließlich eines Vorzeichens die Feldweite w nicht überschreiten, anderenfalls wird das Sonderzeichen * ausgegeben. Wenn Ziffern und Vorzeichen die Feldweite w nicht ausfüllen, wird nach links mit Leerzeichen aufgefüllt.

b) F-Format

Das *F-Format*

Fw.d $(1 \leqslant w \leqslant 254, 0 \leqslant d \leqslant w)$

(d und w ganzzahlige Konstanten ohne Vorzeichen) wird zur Übertragung reeller Zahlen einfacher Genauigkeit ohne Exponent verwendet. w bezeichnet wieder die Feldweite, d gibt die Anzahl der Dezimalstellen rechts vom Punkt an. Die Wirkungsweise des F-Formats wollen wir uns wieder an einigen Beispielen ansehen.

Format F6.2 zur Eingabe:

eingegebene Zeichenfolge	interner Wert
␣␣6.23	6,23
␣␣␣623	6,23
␣␣-␣36	-0,36
␣␣␣␣-1	-0,01
87.1␣␣	87,10
w=6	d=2

Wenn in der eingegebenen Zeichenfolge kein Dezimalpunkt vorkommt, wird seine Position durch d festgelegt. Anderenfalls bleibt die Angabe von d wirkungslos. Leerzeichen werden als Nullen interpretiert, das Vorzeichen + kann entfallen.

Format F6.2 zur Ausgabe:

Wert der auszugebenden Variablen	ausgegebene Zeichenfolge
12,3	␣12.30
-56,387	-56.39
0,0	␣␣0.0␣
-3672,416	******
	w=6

Die Anzahl der auszugebenden Zeichen einschließlich des Dezimalpunktes und des Minuszeichens darf die Feldweite w nicht übersteigen. Für den Druck der Zahl legt d die Stellenzahl des gebrochenen Teiles fest. Hat der interne Wert der Zahl weniger als d Dezimalstellen, erfolgt ein Ergänzen mit Nullen, liegen mehr als d Stellen gebrochener Teil vor, wird auf d Stellen gerundet.

c) H-Format

Das *H-Format* in der Form

$wHc_1 c_2 \ldots c_w \qquad (1 \leqslant w \leqslant 254\,,\ c_i$ beliebige Zeichen)

dient der Übertragung von Zeichenfolgen, die aus beliebigen Zeichen des FORTRAN-Alphabets aufgebaut sein können (Literale). Die Konstante w (natürliche Zahl) gibt an, wieviel beliebige Zeichen hinter dem H vorhanden sein müssen (Leerzeichen sind mitzuzählen). Der Einlesevorgang mit dem H-Format bewirkt, daß die im Format stehenden w Zeichen c_i (i = 1(1)w) durch w eingegebene Zeichen ersetzt werden. Beim Ausgabevorgang dagegen gelangen die im Format stehenden w Zeichen über den Zeilendrucker zur Ausgabe.

Beispiele

	Format vor E/A	übertragene Zeichenfolge	Format nach E/A
Eingabe	9HDIFFERENZ 6H⌴⌴⌴⌴⌴⌴	⌴SUMME⌴⌴⌴ ⌴DATUM	9H⌴SUMME⌴⌴⌴ 6H⌴DATUM
Ausgabe	16HZWISCHENERGEBNIS	ZWISCHENERGEBNIS	16HZWISCHENERGEBNIS

d) X-Format:

Mit Hilfe des *X-Formats*

wX $(1 \leqslant w \leqslant 254)$

ist es möglich, bei der Eingabe w Zeichen der Lochkarte zu übergehen und bei der Ausgabe w Leerzeichen einzusetzen.

Der gesamte Ein- und Ausgabevorgang mit aufbereiteten Daten läuft unter Regie der Formatliste ab, so daß direkt von einer *Formatsteuerung* gesprochen werden kann. Wir wollen uns die für das Zusammenwirken der formatgebundenen E/A-Anweisungen und der Formatanweisung gültigen Regeln unmittelbar an einigen einfachen Beispielen erarbeiten.

Beispiel 1

```
   READ(1,1Ø)I,A,B
1Ø FORMAT(I4,F1Ø.3,F1Ø.3)
```

Den drei Variablen I, A und B sind über die Marke 1Ø die Formate I4, F1Ø.3 und F1Ø.3 zugeordnet. Die Zuordnung erfolgt generell in der Reihenfolge ihrer Niederschrift von links nach rechts. Selbstverständlich müssen die Übertragungslistenelemente und die Formate verträglich sein. Zur Variablen I vom Typ INTEGER gehört das Format I4, zu den Variablen A und B vom Typ REAL gehören die Formate F1Ø.3. Der aus drei Daten bestehende einzulesende Datensatz ist in eine Lochkarte gelocht worden und zwar der ganzzahlige Wert für I in die Spalten 1 bis 4, die reellen Werte für A und B in die Spalten 5 bis 14 und 15 bis 24.

Beispiel 2

```
   READ(1,2Ø)I,J,K
2Ø FORMAT(3I6)
```

Falls gleiche Formate mehrfach hintereinander in der Formatliste vorkommen, können sie mit einem *Wiederholungsfaktor* zusammengefaßt werden. So sind FORMAT(3I6) und FORMAT(I6,I6,I6) äquivalente Anweisungen. Dieser Wiederholungsfaktor (natürliche Zahl) kann vor dem I-Format, dem F-Format und weiteren noch zu behandelnden Formaten auftreten. Die einzulesenden drei ganzen Zahlen stehen auf einer Lochkarte in den Spalten 1 bis 6, 7 bis 12 und 13 bis 18.

Beispiel 3

```
   READ(1,21)I,J,K
21 FORMAT(I6)
```

Wenn nach vollständiger Abarbeitung der Formatliste in der Übertragungsliste noch nicht berücksichtigte Variable übrigbleiben, werden die Formate der Formatliste, beginnend hinter der öffnenden Klammer, erneut benutzt. Mit dem einmaligen Abarbeiten der Formatliste ist jeweils ein Datensatz festgelegt, die Ein- bzw. Ausgabe beginnt dann mit einer neuen Lochkarte bzw. Druckzeile. Die Werte für I,J,K sind also jeweils auf einer Lochkarte in den Spalten 1 bis 6 gelocht.

Beispiel 4

```
   READ(1,22)I,J,K
22 FORMAT(I4,6X,I6,4X,I1Ø)
```

Den X- und H-Formaten entspricht kein Element der Übertragungsliste. Die ganzzahligen Werte für die Variablen I,J und K sollen jetzt auf einer Lochkarte in den Spalten 1 bis 4, 11 bis 16 und 21 bis 30 stehen. Die X-Formate bewirken, daß beim Einlesen der Lochkarte die Spalten 5 bis 10 und 17 bis 20 übergangen werden.

Beispiel 5

```
   WRITE(3,3Ø)Y,Z
3Ø FORMAT(1X,4HY␣=␣,F1Ø.3,5X,4HZ␣=␣,F1Ø.3)
```

Die Werte der reellen Variablen Y und Z werden in aufbereiteter Form auf einer Zeile ausgedruckt. Vor diesem Druckvorgang erfolgt ein Papiervorschub auf die nächste Zeile, der durch die Angabe 1X als erstes Element der Formatliste veranlaßt wird. Folgt auf 1X ein weiteres X-Format, so können diese zusammengefaßt werden, etwa 1X, 7X zu 8X. Mit dieser eingeschränkten Variante einer Papiervorschubsteuerung wollen wir uns vorerst begnügen, eine vollständige Behandlung der Druckervorschubsteuerung wird in 4.4.2.3. vorgenommen werden. Das Format 4HY␣=␣ bewirkt den Druck der Zeichen Y = , anschließend erscheint der Wert der Variablen Y im Format F1Ø.3. Nach 5 Leerzeichen Zwischenraum folgen die Angaben für Z.

```
Y = xxxxxx.xxx     Z = xxxxxx.xxx
└┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┘
   (Wert von Y)      (Wert von Z)
```

Beispiel 6

```
   WRITE(3,4Ø)LOHN
4Ø FORMAT(1X, 2ØX,13HBRUTTOLOHN␣=␣,F7.2,5H␣MARK)
```

Nach einem Papiervorschub auf die nächstfolgende Druckzeile wird

```
                     BRUTTOLOHN = xxxx.xx MARK
└┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┘
```

gedruckt. Nach dem letzten Element der Übertragungsliste noch abzuarbeitende H- und X-Formate der Formatliste werden noch ausgeführt. Die Formate 1X, 2ØX hätten wir auch zu 21X zusammenfassen können.

4.2.5. Explizite Typfestlegung für Variable

In 4.1.3. haben wir die Möglichkeit kennengelernt, Namen mit Hilfe des Anfangsbuchstabens implizit den Typ REAL oder INTEGER zuzuordnen. Außer dieser bequemen Regelung ist es aber auch möglich, durch eine *Typanweisung* davon abweichende Festlegungen zu treffen. Diese explizite Typzuordnung hat gegenüber der vom Anfangsbuchstaben abhängenden den Vorrang. Folgende Typanweisungen sind möglich:

INTEGER $n_1, n_2, \ldots, n_i$

REAL $n_1, n_2, \ldots, n_j$

DOUBLE PRECISION $n_1, n_2, \ldots, n_k$

Die n_q symbolisieren Namen von Variablen, Feldern und Funktionen. An dieser Stelle interessieren vorerst nur Typanweisungen für einfache Variable. Allen Variablen, die in der Liste nach dem jeweiligen FORTRAN-Schlüsselwort aufgeführt sind, wird damit als Typ ganzzahlig, reell oder doppeltgenau zugeordnet. Damit ist für die Verwendung der Variablen, beispielsweise als Operanden in Ausdrücken, eindeutig festgelegt, in welcher Weise sie zu verarbeiten sind. Die Typanweisungen sind nichtausführbare Anweisungen.

Beispiele

```
INTEGER OMEGA,RHO,A
DOUBLE PRECISION SUMME,M1
REAL I,REST
```

Wenn im Programm die Variablen A,B,I, J,M1,REST,SUMME,RHO,OMEGA und XX vorkommen, dann sind A,J,RHO,OMEGA vom Typ INTEGER, ferner B,I,REST,XX vom Typ REAL und M1,SUMME doppeltgenaue Variable. Die explizite Festlegung von REST ist überflüssig.

4.2.6. Struktur einfacher Programme

Wie bereits in 4.1.6. erwähnt, besteht jedes FORTRAN-Programm aus Anweisungen. Wir haben dabei zwischen den *ausführbaren* und *nichtausführbaren Anweisungen* zu unterscheiden. Die ausführbaren Anweisungen legen die auszuführenden Operationen fest. Dazu zählen die bisher behandelten Ergibtanweisungen und die Anweisungen zur Ein- und Ausgabe von Daten. Die nichtausführbaren Anweisungen hingegen werden bei der unmittelbaren Abarbeitung des Programms übergangen, sie enthalten nur Festlegungen für die ausführbaren Anweisungen. Als Beispiele haben wir bereits die Formatanweisung zur Formfestlegung für ein- oder auszugebende Daten und die Typanweisung zur expliziten Typfestlegung kennengelernt. Mit Ausnahme der Formatanweisungen müssen alle nichtausführbaren Anweisungen im Programm vor der Gesamtheit der ausführbaren Anweisungen auf-

geführt sein. Formatanweisungen können an beliebigen Stellen des Programms erscheinen.
Die Abarbeitung eines Programms beginnt immer mit der ersten ausführbaren Anweisung.
Um für den FORTRAN-Compiler das Ende des Programmtextes zu kennzeichnen, muß jedes Programm mit der Endzeile END abgeschlossen sein.
Nach diesen einführenden Bemerkungen über einfachste Programme werden wir die komplette Struktur von FORTRAN-Programmen in 4.5.6. eingehend behandeln.

4.2.7. Beispiele

Beispiel 1

Von einem geraden Kreiskegel, dessen Bestimmungsdaten auf einer Lochkarte in den Spalten 1 bis 10 (Radius), 11 bis 20 (Höhe) und 31 bis 40 (Dichte) verfügbar sind, sollen Masse und Oberfläche berechnet werden. Falls auf der Lochkarte die Zahl der Dezimalstellen nicht durch einen Dezimalpunkt festgelegt ist, soll die gelochte Ziffernfolge als reelle Zahl mit drei Stellen rechts vom Dezimalpunkt interpretiert werden.

Das FORTRAN-Programm hat dann das folgende Aussehen:

```
C MASSE UND OBERFLAECHE EINES GERADEN KREISKEGELS
      REAL MASSE
      READ(1,1Ø) RADIUS,HOEHE,DICHTE
   1Ø FORMAT(2F1Ø.3,1ØX,F1Ø.3)
      MASSE = 3.1416*RADIUS**2*HOEHE*DICHTE/3.Ø
      OBERFL = 3.1416*RADIUS*(RADIUS+SQRT(RADIUS**2+HOEHE**2))
      WRITE(3,11) MASSE,OBERFL
   11 FORMAT(1X, 8HMASSE␣=␣,F1Ø.3,1ØX,14HOBERFLAECHE␣=␣,F1Ø.3)
      STOP
      END
```

Nur die Variable MASSE muß als REAL vereinbart werden, für alle anderen vorkommenden Variablen ist der Typ REAL implizit festgelegt. Die erste Zeile des Programms ist eine Kommentarzeile und fungiert als Programmüberschrift. Die zur Ausgabeanweisung gehörende Formatliste beginnt mit 1X, um den Druck der Ergebnisse auf der nächsten Zeile anzufangen. Im H-Format wird der erläuternde Text für die beiden Resultatwerte angegeben, so daß der Ergebnisdruck folgendes Aussehen bekommt:

```
MASSE = xxxxxx.xxx          OBERFLAECHE = xxxxxx.xxx
      (Wert von MASSE)                  (Wert von OBERFL)
```

Die Anweisung STOP beendet die Ausführung des Programms (vgl. dazu Anhang A4/6). END zeigt für den Compiler das physische Ende des zu übersetzenden Programms an.

Beispiel 2

Es sollen die Lösungen x und y des linearen Gleichungssystems

$Ax + By = E$
$Cx + Dy = F$

ermittelt werden. Die vier Koeffizienten sind in eine Lochkarte in die Spalten 1 bis 8, 9 bis 16, 17 bis 24, 25 bis 32 und die Elemente der rechten Seite in eine folgende Lochkarte in die Spalten 1 bis 8 und 9 bis 16 gelocht worden.

FORTRAN-Programm

```
C LOESUNG EINES LINEAREN GLEICHUNGSSYSTEMS
      READ(1,2Ø) A,B,C,D,E,F
   2Ø FORMAT(4F8.2)
      DELTA = A*D-B*C
C DELTA WIRD VERSCHIEDEN VON NULL VORAUSGESETZT
      X = (D*E-B*F)/DELTA
      Y = (A*F-C*E)/DELTA
      WRITE(3,21)
   21 FORMAT(1X,28HKONTROLLDRUCK␣EINGANGSDATEN:)
      WRITE(3,22) A,B,E,C,D,F
      WRITE(3,23) X,Y
   22 FORMAT(1X,2ØX,F1Ø.2,1ØX,F1Ø.2,1ØX,F1Ø.2)
   23 FORMAT(1X,11HLOESUNGEN:␣,F1Ø.2,1ØX,F1Ø.2)
      STOP
      END
```

Beim Eingabevorgang wird nach Abarbeitung der vier Formate F8.2 der erste Eingabedatensatz abgeschlossen. Der Eingabevorgang wird mit dem Einlesen der nächsten Lochkarte fortgesetzt, wobei die Formatliste erneut von links nach rechts bis zum vollständigen Abarbeiten der Übertragungsliste zur Wirkung kommt. Die Berechnung der Lösungen x und y erfolgt nach der Cramerschen Determinantenregel. Eingangsdaten und Lösungswerte erscheinen im Ergebnisdruck in der Form

```
KONTROLLDRUCK EINGANGSDATEN:
                     xxxxxx.xx        xxxxxx.xx        xxxxxx.xx
                     xxxxxx.xx        xxxxxx.xx        xxxxxx.xx
LOESUNGEN: xxxxxx.xx          xxxxxx.xx
```

Erst zu einem späteren Zeitpunkt (siehe 4.4.5.) werden wir in der Lage sein, die drei Ausgabeanweisungen zu einer zu vereinigen. Im Gegensatz zu den bisherigen Beispielen zeigt die Anweisung WRITE (3,21), daß die Liste der auszugebenden Variablen auch leer sein kann. Es wird dann nur die im H-Format der Formatanweisung 21 stehende Folge von FORTRAN-Zeichen gedruckt.

Beispiel 3

Ein Geldbetrag kleiner als 100 M (Angabe in vollen Mark in den Spalten 79 und 80 einer Lochkarte) soll so in Münzen von 20 M, 5 M und 1 M zerlegt werden, daß die Anzahl der Münzen minimal ausfällt.

FORTRAN-Programm

```
C MINIMALE ZERLEGUNG EINES GELDBETRAGES
      INTEGER BETRAG
      READ(1,5Ø) BETRAG
   5Ø FORMAT(78X,I2)
      MARK2Ø = BETRAG/2Ø
      M = BETRAG-MARK2Ø*2Ø
C M IST NUR HILFSGROESSE
      MARK5 = M/5
      MARK1 = M-MARK5*5
      WRITE(3,51) BETRAG,MARK2Ø,MARK5,MARK1
   51 FORMAT(1X,I2,5H␣M␣=␣,I1,22H␣ZWANZIGMARKMUENZEN␣+␣,I1,
     12Ø H␣FUENFMARKMUENZEN␣+␣,I1,15H␣EINMARKMUENZEN)
C ANWEISUNG 51 MIT FORTSETZUNGSZEILE
      STOP
      END
```

Alle im Programm verwendeten Variablen sind ganzzahlig. Die im Algorithmus enthaltenen Divisionen liefern ganzzahlige Resultate, also nur den ganzzahligen Anteil der Quotienten. Als Ergebnisdruck entsteht

```
xx M = x ZWANZIGMARKMUENZEN + x FUENFMARKMUENZEN + x EINMARKMUENZEN
```

4.2.8. Aufgaben

4.2.8./1

Schreiben Sie die folgenden Formelausdrücke als FORTRAN-Ausdrücke.

$\frac{a^2}{b\,|c|}$ $\qquad$ $|\max(z1,z2,z3,z4,z5) - \min(y1,y2,y3,y4)|$

$A + Bx + Cx^2 + Dx^3 + Ex^4$ $\qquad$ $e^{-(x-1)^2} - \alpha(x-1)^2$

$\sqrt{\cos^2 2x + \sin^2 5x}$ $\qquad$ $(x^i + 2)^N$

$\frac{\tanh \pi x}{\sqrt{1+x^3}}$ $\quad e^{-|x-a|}$

4.2.8./2

Welche Ergebnisse werden für X, Y und Z durch das folgende Programm ermittelt, wenn für die Eingangsgrößen A, B und C in den ersten 15 Spalten einer Lochkarte die Zeichenfolge ␣␣1Ø␣-2Ø␣␣3ØØ gelocht ist.

```
   READ(1,1Ø) A,B,C
1Ø FORMAT(3F5.2)
   X = ((C+B)/C*3.Ø*A)**2/(C*B/((C-B+1.Ø)*2))
   Y = EXP(B+2.Ø*A)*SQRT((C-B)**2-(A+2.Ø)**2)
   Z = SQRT(ABS(X+Y))
   WRITE(3,11) X,Y,Z
11 FORMAT(1X,F1Ø.2,5X,F1Ø.2,5X,F1Ø.2)
   STOP
   END
```

4.2.8./3

Prüfen Sie das nachfolgende Programm auf syntaktische Richtigkeit und begründen Sie die ermittelten Fehler.

```
1 REAL N
2 READ(1,3) A,B,PARAMETER,SIGMA,N
3 FORMAT(2F1Ø.2,6X,F6.1,F6.1,1ØX,I2)
4 F = A*EXP(-ABS(B*N))/SQRT(1.ØØØ+SINSIGMA**2)
5 G-1 = (ABS(F)**1.7+3)/3.Ø-(A+B*N)*N**2
6 F = AMAXØ(F,G-1,PARAMETER)
7 WRITE(3,8) F
8 FORMAT(1X,3HF=␣,F1Ø,2)
9 STOP
```

4.2.8./4

Es ist ein FORTRAN-Programm zur Umrechnung von Polarkoordinaten in kartesische Koordinaten aufzustellen.

4.2.8./5

Die quadratische Gleichung $Ax^2 + Bx + C = 0$ ist mit Hilfe eines FORTRAN-Programms zu vorgegebenen Koeffizienten (in den Spalten 1 bis 8, 9 bis 16 und 17 bis 24 einer Lochkarte für F-Format mit 2 Dezimalstellen gelocht) zu lösen. Dabei kann die Existenz reeller Wurzeln vorausgesetzt werden.

4.3. Verzweigte Programme

4.3.1. Anweisungen zur Programmverzweigung

Mit den bisher besprochenen FORTRAN-Sprachelementen kann die Auswertung von Formeln – Berechnung arithmetischer Ausdrücke und Zuweisungen ihrer Werte an einfache Variable – sowie eine einfache Form der Ein- und Ausgabe programmiert werden. Programmverzweigungen auf der Grundlage von Entscheidungen, die wesentlich die Fähigkeit der Programmsteuerung einer Datenverarbei-

tungsanlage ausmachen, können wir noch nicht handhaben. Basis-FORTRAN kennt hierzu die Sprunganweisung (GOTO-Anweisung) und die arithmetische Wennanweisung (IF-Anweisung).

4.3.1.1. Unbedingte Sprunganweisung (GOTO-Anweisung)

Die *unbedingte Sprunganweisung* bietet eine Möglichkeit, die Ausführung der Anweisungen eines Programms gemäß ihrer textlichen Reihenfolge zu unterbrechen. Sie hat die Form

GOTO m

wobei m die Marke einer ausführbaren Anweisung und GOTO ein Schlüsselwort ist. GOTO m bewirkt, daß die Programmabarbeitung mit der mit m markierten Anweisung fortgesetzt wird.

Es sei bemerkt, daß die im Programmtext unmittelbar nach einer Sprunganweisung stehende Anweisung markiert sein muß, da sie nur durch einen Sprung erreicht werden kann.

Beispiel

Aus hier nicht näher interessierenden Gründen (uns fehlt noch die Motivation für Sprünge) soll ein aus den aufeinanderfolgenden Teilen 1, 2, 3 bestehendes Programm in der Reihenfolge 1, 3, 2 durchlaufen werden. Das läßt sich durch Einfügen von Sprunganweisungen und Markierungen, wie in Bild 2 angegeben, verwirklichen.

```
C PRINZIPBEISPIEL
      Programmteil 1
      GOTO 3
  2   Programmteil 2
      GOTO 5
  3   Programmteil 3
      GOTO 2
  5   STOP
      END
```

Bild 2

4.3.1.2. Arithmetische Wennanweisung (IF-Anweisung)

Die arithmetische Wennanweisung dient der Verzweigung eines Programms in Abhängigkeit vom Vorzeichen eines arithmetischen Ausdrucks. Sie hat die Form

IF (a) m_1, m_2, m_3

Dabei sind: IF ein Schlüsselwort, a ein arithmetischer Ausdruck, m_1, m_2, m_3 Marken ausführbarer Anweisungen. Die Abarbeitung der Wennanweisung beginnt mit der Berechnung von a. Ergibt sich ein Wert kleiner als Null, so wird das Programm mit der Anweisung fortgesetzt, die die Marke m_1 trägt. Ist der Wert von a gleich Null oder größer als Null, so erfolgt die Fortsetzung bei m_2 bzw. m_3. Die arithmetische Wennanweisung bestimmt also eine von drei Fortsetzungen (vgl. Bild 3).

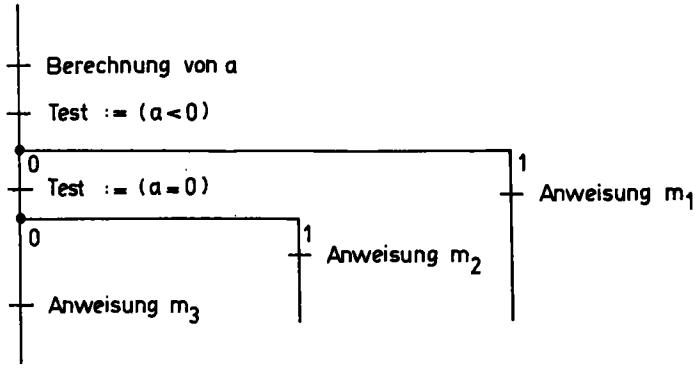


Bild 3

Beispiele korrekter Wennanweisungen:

```
IF (X-2) 5,7,2
IF (A*B + 3.5) 4,4,5
IF (EXP(X/2)*DCOS(V)) 1,2,3
```

Fehlerhaft sind:

```
IF (A+B) 4,3,I      Variable als Marke
IF (C) 2,14         fehlende Marke
```

Auch die einer Wennanweisung im Programmtext folgende Anweisung ist nur erreichbar, wenn sie eine Marke trägt.

Programmbeispiel

Die Funktion $y = e^{-\frac{x^2}{2}}$ soll mit der Schrittweite $h > 0$, beginnend bei $x = 0$, solange tabelliert werden, bis $y < 10^{-5}$ eingetreten ist.

FORTRAN-Programm

```
C TABELLIERUNG VON Y = EXP(-X**2/2)
      WRITE (3,1Ø)
   1Ø FORMAT (1X,29HTABELLIERUNG␣VON␣EXP(-X**2/2))
      READ (1,11) H
   11 FORMAT (F5.2)
      X = Ø.Ø
    1 Y = EXP(-X*X*Ø.5)
      WRITE (3,12) X, Y
   12 FORMAT (21X, F7.2, 4X, F8.6)
      X = X + H
      IF (Y - 1.E-5) 2,1,1
    2 STOP
      END
```

Nach dem Druck der Programmüberschrift auf einer neuen Zeile wird H eingelesen. Es folgt die Berechnung des ersten Paares X, Y. Im Ausdruck für Y wurde aus Gründen der Rechenzeit die Multiplikation mit 0.5 statt der Division durch 2 verwendet. Der Druck der Werte von X und Y erfolgt jeweils nach Zeilenvorschub ab Druckstelle 21, zwischen X und Y erscheinen vier Leerstellen. Die Wennanweisung führt bei $Y \geqslant 10^{-5}$ zurück zur Fortsetzung der Tabellierung (Marke 1), ansonsten zum Abschluß des Programms (Marke 2).

4.3.1.3. Berechnete Sprunganweisung (berechnete GOTO-Anweisung)

Die *berechnete Sprunganweisung* dient zur Mehrfachverzweigung und zur bequemen Organisation eines variablen Rücksprungs aus einem mehrfach genutzten Programmstück. Sie entspricht dem variablen Konnektor der PAP-Technik und hat die Form

GOTO $(m_1, m_2, \ldots, m_n)$, i

wobei $m_1, m_2, \ldots, m_n$ Marken ausführbarer Anweisungen sind und i eine einfache ganzzahlige Variable ist. Hat i einen der Werte $1, 2, \ldots, n$, etwa k, so erfolgt die Programmfortsetzung mit der mit m_k markierten Anweisung (vgl. Bild 4). Andernfalls ist die berechnete Sprunganweisung leer, d. h. es wird mit der unmittelbar danach stehenden Anweisung fortgefahren.

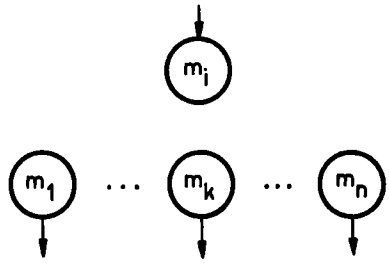


Bild 4

Beispiel

```
GOTO (1,3,15), L
```

Hat L den Wert 3, so wird im Programm mit der durch 15 markierten Anweisung fortgefahren. Wenn L = 1 ist, erfolgt die Fortsetzung bei der Marke 1, wenn L = 2 ist, bei der Marke 3. Falls L keinen der Werte 1, 2, 3 hat, kommt die auf GOTO (1, 3, 15), L im Programmtext folgende Anweisung zur Abarbeitung.

Programmbeispiel

Die Funktion y(x) habe die Definition

$$y = \begin{cases} 0{,}5\,x & \text{für } 0 \leqslant x < 4 \\ 2+\sqrt{4-(x-6)^2} & \text{für } 4 \leqslant x < 8 \\ 6-0{,}5\,x & \text{für } 8 \leqslant x \leqslant 12 \end{cases}$$

Zu einem vorgegebenen Wert x, $0 \leqslant x \leqslant 12$, ist y(x) zu ermitteln.

```
C PROGRAMMAUSSCHNITT
      I = X/4 + 1
C ZUWEISUNG DES GANZEN TEILS DES WERTES VON X/4 + 1 AN I
      GOTO (4,5,6,6), I
    4 Y = Ø.5*X
      GOTO 3Ø
    5 Y = 2.Ø + SQRT(4.Ø - (X - 6.Ø)** 2)
      GOTO 3Ø
    6 Y = 6 - Ø.5*X
   3Ø .........
```

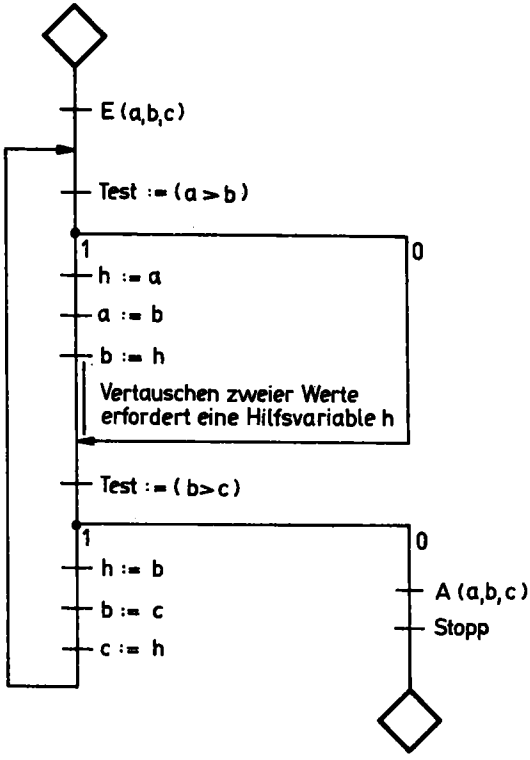
Die Anweisung I = X/4 + 1 ergibt genau für jedes X des ersten Intervalls $0 \leqslant X < 4$ den Wert 1, entsprechend 2 für das zweite und 3 für das dritte angegebene Intervall mit Ausnahme von X = 12. Für X = 12 lautet I = 4. Deshalb mußte die 6 auch noch als vierte Marke in die Liste hinter GOTO aufgenommen werden.

4.3.1.4. Beispiele

Beispiel 1

Die drei Variablen a, b, c sind nach steigenden Werten auf Platz zu sortieren, d. h. durch Vertauschen soll $a \leqslant b \leqslant c$ erreicht werden.
Es werden also benachbarte Fehlstände durch Vertauschen beseitigt. Falls $a > b$ ist, erfolgt der Austausch $a \leftrightarrow b$, anschließend, falls erforderlich, $b \leftrightarrow c$ und eventuell noch $a \leftrightarrow b$. Hätte man auf den Zyklus verzichtet, müßte ein dritter bedingter Austausch vorgesehen werden.

Programmablaufplan



FORTRAN-Programm

```
C SORTIEREN DREIER WERTE AUF PLATZ
      READ (1,1Ø) A, B, C
   1Ø FORMAT (3F15.5)
    5 IF (A - B) 2,2,1
    1 H = A
      A = B
      B = H
    2 IF (B - C) 4,4,3
    3 H = B
      B = C
      C = H
      GOTO 5
    4 WRITE (3,2Ø) A, B, C
   2Ø FORMAT (5X, F15.5)
      STOP
      END
```

Beispiel 2

Die Funktion

$$z = \begin{cases} \dfrac{4}{x^2+y^2}\ln\left(1+x+\dfrac{x^2+y^2}{4}\right) & \text{für } 0 \leqslant x \leqslant 1 \\ \dfrac{2x}{y}\arctan\dfrac{y}{x+2} & \text{für } 1 < x \leqslant 2 \end{cases}$$

ist für x = 0(0.2)2 und 15 einzulesende Werte $y = y_i$ (i = 1(1)15) zu berechnen. Es ist jeweils y mit den dazugehörigen Werten von z auszudrucken. Einen PAP zu dieser Aufgabe finden Sie im Abschnitt „Beschreibung eines Programmablaufs" (vgl. Aufgabe 13) dieses Lehrbuchs.

FORTRAN-Programm

```
C FUNKTIONSTABELLIERUNG Z = Z(X,Y)
      I = 1
    9 READ (1,1Ø) Y
   1Ø FORMAT (F12.4)
      WRITE (3,2Ø) Y
   2Ø FORMAT (1X,4HY␣=␣, F12.4)
      X = Ø.Ø
    7 IF (X - 1.1) 3,3,4
    3 A = (X*X+Y*Y)/4
      Z = ALOG(1+X+A)/A
      GOTO 5
    4 Z = 2*X/Y*ATAN(Y/(X+2))
    5 WRITE (3,3Ø) Z
   3Ø FORMAT (21X, F12.4)
      X = X + Ø.2
```

```
      IF (X - 2.1) 7,7,8
    8 I = I + 1
      IF (I- 15) 9,9,6
    6 STOP
      END
```

Auf den Druck von Y folgen jeweils die elf Z-Werte untereinander, gegenüber der Y-Zeile um mindestens 20 Druckstellen eingerückt.

Allgemeine Bemerkung

Die Vergleiche $X \leqslant 1$ und $X \leqslant 2$ wurden durch $X \leqslant 1.1$ bzw. $X \leqslant 2.1$ ersetzt, da X als REAL-Zahl durch seine additive Erzeugung mit Hilfe der (infolge Konvertierung ins Dualsystem und endlicher Zahlwortlänge) in der Rechenanlage rundungsfehlerbehafteten Schrittweite 0.2 nicht exakt die Werte 1 bzw. 2 annehmen muß. Der Sicherheitszuschlag beträgt die halbe Schrittweite. Diese Maßnahme ist immer zweckmäßig, wenn eine REAL-Größe zur Steuerung benutzt wird.

Beispiel 3

Der Funktionswert y = sin x soll für ein vorgegebenes x (Bogenmaß!) nach der Potenzreihenentwicklung

$$y = x - \frac{x^3}{3!} + \frac{x^5}{5!} - + \ldots$$

mit einem relativen Fehler von höchstens 10^{-8} berechnet und gedruckt werden.

Vorbemerkung: Jedes Glied der Reihe läßt sich rekursiv aus seinem Vorgänger berechnen. Mit der Schreibweise $y = g_1 + g_3 + g_5 + \ldots$ gilt

$$g_1 = x$$

und

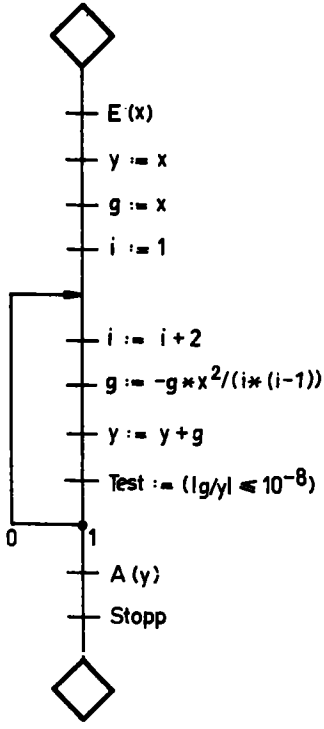
$$g_i = -g_{i-2} \cdot \frac{x^2}{(i-1) \cdot i} \qquad (i = 3, 5, \ldots)$$

FORTRAN-Programm

```
C SIN(X)
      READ (1,1Ø) X
   1Ø FORMAT (F9.6)
      Y = X
      G = X
      I = 1
    3 I = I + 2
      G = -G*X*X/(I*(I-1))
      Y = Y + G
      IF (ABS(G/Y)-1.E-8) 5,5,3
    5 WRITE (3,11) X,Y
```

```
11 FORMAT (1X,4HSIN(,F9.6,4H)␣=␣, F9.6)
   STOP
   END
```

Programmablaufplan



Der Genauigkeitstest bezieht sich auf den absoluten Betrag des Quotienten aus letztem Glied und Näherungswert. Im Hinblick auf die Rechenzeit wäre es günstiger, für den konstanten Faktor x^2 zur Berechnung von g eine Hilfsvariable einzuführen.
Bei diesem Beispiel kam es uns auf die Programmierung eines Iterationszyklus an. In der praktischen Programmierung berechnet man einen Sinuswert mit Hilfe einer Standardfunktion.

4.3.1.5. Aufgaben

4.3.1.5./1

Je nach der Größe des Wertes einer Variablen a soll ein Programm an unterschiedlichen Stellen fortgesetzt werden:

Wertebereich	Marke der Fortsetzung
$a \leqslant -1{,}5$	1Ø
$-1{,}5 < a < 1{,}5$	2Ø
$1{,}5 \leqslant a < 3$	3Ø
$3 \leqslant a < 4{,}5$	4Ø
$4{,}5 \leqslant a$	5Ø

Formulieren Sie diese Programmverzweigung

a) mit Hilfe arithmetischer Wennanweisungen
b) mit Hilfe einer berechneten Sprunganweisung!

4.3.1.5./2

Vorgegeben ist eine unbekannte Anzahl von Meßdaten $a > 0$, je ein Wert pro Lochkarte. Den Abschluß bilde ein Wert $a = 0$. Gesucht sind die Anzahl n der Meßdaten sowie die Prozentzahlen p, q, r der auf die Intervalle $0 < a \leqslant 2$, $2 < a \leqslant 5$, $5 < a$ entfallenden Meßdaten.

4.3.1.5./3

Ö

Gegeben sind ein Darlehensbetrag DARL, der Betrag der Rückzahlungsrate RATE sowie ein Jahreszinsfuß ZI in %. Das Darlehen wird durch konstante monatliche Ratenzahlungen getilgt. Der monatlichen Zinsenberechnung liegt als Zinsfuß 1/12 des Jahreszinsfußes zugrunde.
Ermitteln Sie, nach wieviel Monaten das Darlehen getilgt ist, sowie die Höhe der eventuellen Überzahlung im letzten Tilgungsmonat.

4.3.1.5./4

T

Die Stromstärke J in einem elektrischen Schwingkreis mit ohmschem Widerstand R, Induktivität L und Kapazität C zum Zeitpunkt t berechnet sich gemäß

$$J = \begin{cases} A\, e^{-\frac{Rt}{2L}} \cos\left(\sqrt{\frac{1}{LC} - \frac{R^2}{4L^2}}\, t - \varphi\right) & \text{für } \frac{1}{LC} > \frac{R^2}{4L^2} \\ e^{-\frac{Rt}{2L}} (A + Bt) & \text{für } \frac{1}{LC} = \frac{R^2}{4L^2} \\ A\, e^{-\left(\frac{R}{2L} + \sqrt{\frac{R^2}{4L^2} - \frac{1}{LC}}\right)t} + B\, e^{-\left(\frac{R}{2L} - \sqrt{\frac{R^2}{4L^2} - \frac{1}{LC}}\right)t} & \text{für } \frac{1}{LC} < \frac{R^2}{4L^2} \end{cases}$$

Stellen Sie für vorgegebene Werte von R, L, C sowie t, A, B, φ ein FORTRAN-Programm zur Berechnung von J auf.

Z

4.3.1.5./5

Wir betrachten ein Zweipersonenspiel, bei dem 9 Münzen in drei Zeilen angeordnet sind, zwei in der ersten, drei in der zweiten, vier in der dritten Zeile.

Spielregeln:

1. Es werden abwechselnd Münzen entfernt
2. Jeder Zug muß aus Münzen einer Zeile bestehen und mindestens eine Münze umfassen
3. Gewinner ist, wer zuletzt zieht.

Stellen Sie ein FORTRAN-Programm für eine optimale Spielweise des Nachziehenden bis zur Spielentscheidung auf. Im Wechsel erfolgen Lochkarteneingabe des Zuges und Druck des Antwortzuges, beide bestehend aus Zeilennummer und Zugumfang.

Anleitung:

Bei richtiger Strategie gewinnt stets der Anziehende. Nur bei einem Fehler des Anziehenden gewinnt der Nachziehende bei folgender Strategie:

1. Erzeugung zweier gleichlanger Zeilen bei leerer restlicher Zeile
2. Wenn das nicht möglich ist, Erzeugung der Anzahlen 1, 2, 3
3. Bei Vorliegen von 1, 2, 3 (richtige Strategie des Anziehenden!) garantiert kein Zug den Gewinn, also ist ein beliebiger auszuführen.

4.3.2. Anweisungen zur Arbeit mit indizierten Variablen

Zahlreiche Rechenprozesse führen eine bestimmte Folge von Operationen in zyklischer Wiederholung für gleichartige Größen aus. Solche Größen sind zum Beispiel ein Satz von Meßwerten oder eine Menge von Verflechtungskoeffizienten in einem linearen Produktionsmodell. Man faßt sie in der Mathematik als Vektor oder Matrix zusammen und unterscheidet sie durch Anfügen von Indizes an den

gemeinsamen Namen. FORTRAN nennt eine solche geordnete Gesamtheit ein Feld, das einzelne Element indizierte Variable oder Feldelement.
Zur bequemen Steuerung zyklischer Rechenprozesse dient die Schleifenanweisung.

4.3.2.1. Felder

4.3.2.1.1. Definition von Feldern

Ein *Feld* ist bestimmt durch seinen Namen, seine Dimension und seine Größe. Die *Dimension* entspricht der Anzahl der Indizes. Basis-FORTRAN gestattet ein-, zwei- und dreidimensionale Felder, die man als Vektor, Matrix und übereinanderliegende Matrizen auffassen kann.
Die Definition eines Feldes muß stets explizit (hinsichtlich Dimension und Größe) zu Beginn des Programms, vor der ersten ausführbaren Anweisung und vor einer eventuellen Anweisungsfunktionsdefinition (s. 4.5.2.2.) erfolgen.
Felder werden entweder durch eine Feldanweisung oder eine Typanweisung oder eine Globalanweisung (vgl. 4.5.5.) definiert. Die *Feldanweisung* hat die Form

DIMENSION $n_1(i_1), n_2(i_2), \ldots, n_k(i_k)$

Dabei sind: DIMENSION ein FORTRAN-Schlüsselwort, $n_1, \ldots, n_k$ Namen, hier *Feldnamen* genannt, $i_1, \ldots, i_k$ *Indexlisten*. Entsprechend der Dimension des Feldes besteht eine Indexliste aus bis zu drei natürlichen Zahlen, die durch Kommas getrennt werden. Diese Konstanten stellen die oberen Grenzen der Laufbereiche der einzelnen Indizes dar, während die unteren Grenzen stets 1 lauten. Die einzelnen Indizes können unabhängig voneinander jeden Wert ihres Laufbereichs annehmen, so daß sich die *Größe* des Feldes, die Anzahl seiner Elemente, als Produkt der oberen Indexgrenzen ergibt. Der *Typ* eines Feldes n_j ist entweder implizit gegeben oder n_j (ohne Indexliste!) ist in einer entsprechenden Typanweisung anzuführen, die aber nicht vor der Feldanweisung stehen darf.

Beispiel

```
DIMENSION I(1Ø), A(3,2), T(2,3,2)
```

definiert drei Felder I, A, T unterschiedlicher Dimension mit 10, 6, 12 Elementen. Erscheinen die Feldnamen anschließend in keinen Typanweisungen, handelt es sich bei I um ein ganzzahliges Feld, bei A und T um reelle Felder. Würden die Typanweisungen

```
REAL I
DOUBLE PRECISION T
```

folgen, so wären I und A reelle Felder und T ein doppeltgenaues Feld.
Bei der Felddefinition mittels *Typanweisung* wird wie bei der Feldanweisung an jeden Feldnamen die Indexliste angefügt.

Beispiele

```
REAL I(1Ø), A(3,2)
DOUBLE PRECISION T(2,3,2)
INTEGER C(5,7), B, D, I(5)
```

Die beiden ersten Beispiele sind zusammen eine äquivalente Formulierung des obigen Beispiels mit den zwei zusätzlichen Typanweisungen. Im dritten Beispiel werden Felder C und I und einfache Variable B, D in einer Typanweisung definiert.
Es sei besonders darauf hingewiesen, daß ein Feld nur auf genau eine der drei genannten Arten definiert werden kann.

4.3.2.1.2. Feldelemente

Die Niederschrift eines *Feldelements* – auch *indizierte Variable* genannt – erfolgt so, daß dem Feldnamen die Indizes folgen, die wie in der Felddefinition in Klammern eingeschlossen und durch Kommas getrennt werden.
Beim obigen Beispiel

```
DIMENSION I(1Ø), A(3,2), T(3,2,2)
```

haben wir es demnach mit folgenden Feldelementen zu tun:

```
     I(1)     I(2)   ..........   I(1Ø)

A(1,1)   A(1,2)          T(1,1,1)   T(1,1,2)   T(1,2,1)   T(1,2,2)
A(2,1)   A(2,2)          T(2,1,1)   T(2,1,2)   T(2,2,1)   T(2,2,2)
A(3,1)   A(3,2)          T(3,1,1)   T(3,1,2)   T(3,2,1)   T(3,2,2)
```

Von großer Wichtigkeit für die Programmierung ist die Festlegung über die Anordnung der Feldelemente bei interner Speicherung. Die Speicherung erfolgt grundsätzlich in aufeinanderfolgende Speichereinheiten, auch Speicherwörter genannt. Die Elemente eines eindimensionalen Feldes werden in natürlicher Reihenfolge gespeichert, beim zweidimensionalen Feld erfolgt die Anordnung spaltenweise und beim dreidimensionalen Feld wird so verfahren, daß ein vorderer Index schneller läuft als ein hinterer. Also lautet die Speicherreihenfolge der Felder A und T:

```
A(1,1)  A(2,1)  A(3,1)  A(1,2)  A(2,2)  A(3,2)

T(1,1,1)  T(2,1,1)  T(3,1,1)  T(1,2,1)  T(2,2,1)  T(3,2,1)  T(1,1,2)
T(2,1,2)  T(3,1,2)  T(1,2,2)  T(2,2,2)  T(3,2,2)
```

Es sei noch bemerkt, daß ein Feld vom Typ DOUBLE PRECISION pro Feldelement zwei Speichereinheiten belegt und damit insgesamt den doppelten Speicherbedarf eines REAL- oder INTEGER-Feldes gleicher Größe hat.
Indizes dürfen die Gestalt eines ganzzahligen Ausdrucks der folgenden sieben Arten, eines sogenannten *Indexausdrucks*, haben:

Indexausdruck	Beispiel
c	4
v	I
v + c	N + 3
v − c	M − 7
k * v	3 * I
k * v + c	4 * N + 2
k * v − c	2 * M − 1Ø

Dabei bezeichnen c und k natürliche Zahlen und v eine einfache ganzzahlige Variable. Der Wert jedes Indexausdrucks muß im Laufbereich des zugehörigen Index liegen, d. h. zwischen 1 (einschließlich) und der oberen Grenze des Laufbereichs (einschließlich).

Beispiele

FORTRAN-Schreibweise	übliche Schreibweise
C(7)	C_7
MAT(2*N-1, 15)	$MAT_{2n-1,15}$
X(I, J, K-1)	$X_{ij,k-1}$
WERT(3*I)	$WERT_{3i}$

Fehlerhaft sind die Schreibweisen:

C(5.Ø)	5.Ø nicht ganzzahlig
A(-I, J)	−I kein Indexausdruck
V(2 + J,Ø)	2 + J müßte lauten J + 2 Ø außerhalb des Laufbereichs
B(I * 3 - 2, C(1))	I * 3 − 2 müßte lauten 3 * I − 2 C(1) kein Indexausdruck
C(-9 + L)	−9 + L kein Indexausdruck

Programmbeispiel

Gegeben sei die zweireihige Matrix

$$\underline{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix},$$

deren Inverse

$$\underline{A}^{-1} = \begin{bmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{bmatrix}$$

ermittelt werden soll.

Mit $d = a_{11}a_{22} - a_{12}a_{21}$ gelten die Formeln:

$\alpha_{11} = a_{22}/d,\ \alpha_{12} = -a_{12}/d,\ \alpha_{21} = -a_{21}/d,\ \alpha_{22} = a_{11}/d.$

FORTRAN-Programm

```
C INVERSE EINER ZWEIREIHIGEN MATRIX
      DIMENSION A(2,2), ALPHA(2,2)
      READ (1,1Ø) A
   1Ø FORMAT (4F15.5)
      D = A(1,1)*A(2,2) - A(1,2)*A(2,1)
      ALPHA(1,1) = A(2,2)/D
      ALPHA(1,2) = -A(1,2)/D
      ALPHA(2,1) = -A(2,1)/D
      ALPHA(2,2) = A(1,1)/D
      WRITE (3,2Ø) ALPHA(1,1), ALPHA(1,2)
   2Ø FORMAT (1X, F15.5, 3X, F15.5)
      WRITE (3,2Ø) ALPHA(2,1), ALPHA(2,2)
C AUSGABE DER MATRIX ALPHA IN UEBLICHER ANORDNUNG
      STOP
      END
```

Allgemeine Bemerkung

Ein Feldname kann ebenso wie eine Variable in einer Übertragungsliste stehen. READ (1,1Ø) A liest das Feld A in der Speicherreihenfolge ein, also spaltenweise. Das ist bei der Herstellung der Primärdatenträger zu beachten.

4.3.2.2. Äquivalenzanweisung (EQUIVALENCE-Anweisung)

Mit Hilfe der Äquivalenzanweisung kann man zwei oder mehrere Variable als äquivalent erklären. Das ist erforderlich, wenn man in einem Programm

a) für die gleiche Größe zwei oder mehrere verschiedene Namen benutzen möchte,
b) zwei oder mehreren verschiedenen Variablen denselben Speicherplatz zuweisen möchte.

Die zweite Möglichkeit setzt natürlich voraus, daß die einzelnen Variablen im Programm nicht zugleich benötigt werden, denn ein Speicherplatz kann zu jedem Zeitpunkt nur einen Wert enthalten. Diese Möglichkeit spielt insbesondere bei der Arbeit mit größeren Feldern eine Rolle, bei der häufig die begrenzte Hauptspeicherkapazität zu besonderen Maßnahmen zwingt.
Die *Äquivalenzanweisung* ist eine nichtausführbare Anweisung und hat die Form

EQUIVALENCE $(l_1), (l_2), \ldots, (l_m)$

Dabei ist EQUIVALENCE ein Schlüsselwort, und die l_i sind Listen der Gestalt

$v_1, v_2, \ldots, v_k \qquad (k \geqslant 2)$

wo v_j entweder eine einfache oder eine indizierte Variable mit Konstanten als Indizes (Feld muß vorher vereinbart worden sein!) darstellt. Allen v_j einer Liste l_i wird der gleiche Speicherplatz zugewiesen. Ist v_j ein Feldelement, so werden die übrigen Elemente dieses Feldes entsprechend ihrer Anordnung (vgl. 4.3.2.1.2.) vor oder hinter diesem Speicherplatz gespeichert. Kommt ein v_j vom Typ DOUBLE PRECISION vor, so orientiert die Äquivalenzanweisung auf die erste (linke) Speichereinheit von v_j.

Beispiel

```
DIMENSION A(2), Y(3)
EQUIVALENCE (A(1), Y(1), C)
```

Damit haben A(1) und Y(1) und C sowie A(2) und Y(2) je einen gemeinsamen Speicherplatz, während der Platz von Y(3) nicht mehrfach genutzt wird.
Als Besonderheit muß noch erwähnt werden, daß Feldelemente zwei- und dreidimensionaler Felder in einer Äquivalenzanweisung auch mit nur einem Index stehen können. Dieser Index ist die Nummer des Elementes in der festen Speicherreihenfolge (vgl. 4.3.2.1.2.) bei Zählung der Feldelemente ab eins. In Formeln:

$$a(i,\ j) \mathrel{\hat{=}} a(i + (j - 1)m)$$
$$b(i,\ j,\ k) \mathrel{\hat{=}} b(i + (j - 1)m + (k - 1)mn)$$

Dabei sind m und n die oberen Grenzen des ersten und zweiten Index.

Beispiel

```
DIMENSION A(2,5), B(3,2,2), E(7)
DOUBLE PRECISION C, D(3)
EQUIVALENCE (A(2,1), B(5), C), (R, E(1), D(1))
```

Belegung der zugehörigen Speichereinheiten:

Speichereinheit	k	k+1	k+2	k+3	k+4	k+5
Belegung	B(1,1,1)	B(2,1,1)	B(3,1,1)	B(1,2,1) A(1,1)	B(2,2,1) A(2,1)	B(3,2,1) A(1,2) C

k+6		k+10	k+11	k+12
B(1,1,2) A(2,2)	...	B(2,2,2) A(2,4)	B(3,2,2) A(1,5)	A(2,5)

Speichereinheit	1	1+1	1+2	1+3	1+4	1+5	1+6
Belegung	R E(1)	E(2)	E(3)	E(4)	E(5)	E(6)	E(7)
	D(1)		D(2)		D(3)		

Eine praktische Anwendung der Äquivalenzanweisung unter dem Aspekt der mehrfachen Speicherausnutzung folgt in 4.3.2.4. Hier sei noch ein praktisches Beispiel zum Aspekt mehrerer Bezeichnungen für eine Größe angeführt:
Benötigt man bei einem Problem eine Spalte einer Matrix (Tabelle) sehr häufig, so faßt man diese Spalte als Vektor auf, gibt ihr einen neuen Namen und legt eine entsprechende Äquivalenz fest, zum Beispiel

```
DIMENSION A(2Ø,12), V(2Ø)
EQUIVALENCE (A(1,8), V(1))
```

V ist hiermit als achte Spalte von A definiert worden (V(1) = A(1,8), ..., V(20) = A (20,8)). Da nur noch ein Index auszuwerten ist, wird so die Verarbeitung beschleunigt.
Es ist selbstverständlich, daß man mit einer Äquivalenzanweisung keine Widersprüche in der Speicherzuordnung erzeugen darf. Fehlerhaft ist demnach zum Beispiel

```
EQUIVALENCE (A(1), X(4)), (A(5), X(1Ø))
```

da die erste Liste die Zuordnung (A(5), X(8)) bewirkt im Widerspruch zur zweiten Liste.

4.3.2.3. Schleifenanweisung (DO-Anweisung)

4.3.2.3.1. Einfache DO-Schleife

Beginnen wir mit einem Beispiel. Von n Werten a_i (i = 1(1)n) sei die Summe $s = a_1 + a_2 + \ldots + a_n$ und die Summe der Quadrate $t = a_1^2 + a_2^2 + \ldots + a_n^2$ zu berechnen. Wir geben dazu einen Programmausschnitt mit der uns vertrauten Sprungsteuerung und einen mit einer DO-Schleife an:

```
  S = Ø.Ø                  S = Ø.Ø
  T = Ø.Ø                  T = Ø.Ø
  I = Ø                    DO 1 I = 1,N
1 I = I + 1                S = S + A(I)
  S = S + A(I)           1 T = T + A(I)**2
  T = T + A(I)**2            ........
  IF (I - N) 1,2,2
2 ........
```

Die Schleifenanweisung DO 1 I = 1,N steuert die Abarbeitung der darunter stehenden Anweisungen des Schleifenbereichs, dessen letzte die Marke 1 trägt.
Es wird an diesem Beispiel schon offensichtlich, welche Vereinfachung der Programmierung möglich ist, wenn sich die Schleifenanweisung anwenden läßt.
Die allgemeine Form der *Schleifenanweisung* lautet:

DO m i = n_1, n_2, n_3 oder DO m i = n_1, n_2

DO ist ein Schlüsselwort, m die Marke einer ausführbaren Anweisung, i eine einfache ganzzahlige Variable *(Steuervariable)*, n_1, n_2, n_3 sind natürliche Zahlen oder einfache ganzzahlige Variable (Steuerparameter: *Anfangswert, Testwert, Schritt-*

weite), die stets positive Werte haben müssen. Fehlt n_3, so bedeutet das automatisch die Schrittweite 1. Die Steuerparameter müssen außerdem die Bedingung Anfangswert ⩽ Testwert erfüllen. Die mit m markierte Anweisung muß im Programmtext (nicht notwendig unmittelbar) der Schleifenanweisung folgen und bildet die letzte Anweisung des *Schleifenbereichs* – auch *DO-Bereich* genannt –, der unmittelbar hinter der Schleifenanweisung beginnt. Diese letzte Anweisung darf keine

GOTO –
IF –
DO –
STOP –
PAUSE – oder
RETURN-Anweisung

sein (PAUSE-Anweisung s. Anhang A 4/6, RETURN-Anweisung s. 4.5.2.1.1. und 4.5.3.1.). Mit Hilfe der *Leeranweisung*

CONTINUE

die ohne Einfluß auf den Algorithmus des programmierten Problems ist, kann das stets erreicht werden, indem man sie als letzte Anweisung des Schleifenbereichs setzt.
Der Ablauf der Abarbeitung einer *DO-Schleife*, wie man für Schleifenanweisung und Schleifenbereich zusammen auch sagt, läßt sich durch folgende FORTRAN-Anweisungen symbolisch erläutern:

```
    I = N1
1   Anweisungen
    des
    Schleifenbereichs
    IF (I + N3- N2) 2,2,3
2   I = I + N3
    GOTO 1
3   ........
```

Der Schleifenbereich wird also mindestens einmal abgearbeitet. Seine Abarbeitung wird solange fortgesetzt, wie der Wert der Steuervariablen I den Testwert N2 nicht überschreitet.

Beispiele für Schleifenanweisungen

Schleifenanweisung	Werte der Steuervariablen
DO 1 I = 1,1Ø	I = 1(1)10
DO 25 K = 5,75,1Ø	K = 5(10)75
DO 1Ø I = 1,N	I = 1(1)N
DO 2Ø J = 2,M,N	J = 2,2+N,2+2N, ... solange J ⩽ M gilt

Fehlerhaft sind:

```
DO 1Ø, I = 1,1Ø        Komma hinter der Marke
DO 2Ø K = 1.Ø,M+4      1.0 keine ganzzahlige Konstante
                       M+4 keine Variable
```

Beispiele für DO-Schleifen

a)
```
    S = Ø.Ø
    DO 1 I = 1,N
  1 S = S + A(I)*B(I)
```

In diesem Beispiel wird die Produktsumme $S = A_1B_1 + \ldots + A_nB_n$ bzw. das Skalarprodukt der Vektoren $\underline{A}$ und $\underline{B}$ mit je N Komponenten berechnet.

b)
```
    DO 1 I = 1,N
    IF (X(I) - 1Ø) 2,2,3
  2 Y(I) = 2*X(I)
    GOTO 1
  3 Y(I) = -2*X(I) + 4Ø
  1 CONTINUE
```

Bei dieser Funktionstabellierung ist die Leeranweisung CONTINUE erforderlich, da hier die GOTO-Anweisung ein Ziel im Schleifenbereich haben muß.

c)
```
3 DO 1 I = 1,1ØØØ
    GLIED = GLIED*X/I
    Y = Y + GLIED
  1 IF (GLIED/Y - 1.E-8) 2,3,3
  2 ........
```

Dieses Beispiel ist nicht korrekt, da die letzte Anweisung des Schleifenbereichs eine IF-Anweisung ist. Mit folgender Änderung ist der Fehler beseitigt:

```
    DO 1 I = 1,1ØØØ
    GLIED = GLIED*X/I
    Y = Y + GLIED
    IF (GLIED/Y - 1.E-8) 2,1,1
  1 CONTINUE
  2 ........
```

Inhaltlich geht es hier um den zyklischen Teil der Berechnung von $y = e^x$ nach der Potenzreihe $1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$ mit der Abbruchschranke 10^{-8}.

Beim letzten Beispiel wird die DO-Schleife durch einen Sprung aus der Schleife heraus beendet. Die Steuervariable I bleibt in diesem Fall mit ihrem letzten Wert vorm Absprung außerhalb der DO-Schleife definiert. Das gilt nicht, wenn die Beendigung einer DO-Schleife durch Ausschöpfen des Wertebereichs der Steuervariablen erfolgt.
Ein Sprung in eine DO-Schleife, also unter Umgehung der zugehörigen Schleifenanweisung, ist nicht zulässig (Ausnahme bei DO-Nestern, vgl. 4.3.2.3.2.).
Schließlich dürfen in einem Schleifenbereich die Werte der Steuerparameter nicht verändert werden.

Bevor wir uns Schachtelungen von DO-Schleifen zuwenden, betrachten wir noch zwei Beispiele.

Beispiel 1

FORTRAN-Programm zur Ermittlung des größten von n ($n \geq 2$) Zahlenwerten $a_1, a_2, \ldots, a_n$.

```
C MAXIMUM VON N ZAHLEN
      DIMENSION A(999)
      REAL MAX
      READ (1,1Ø) N
   1Ø FORMAT (I3)
      READ (1,11) (A(I), I = 1,N)
   11 FORMAT (5F15.3)
      MAX = A(1)
C ANFANGSWERT FUER MAX
      DO 3 I = 2,N
      IF (A(I) - MAX) 3,3,4
    4 MAX = A(I)
C MAX IST GROESSTWERT VON A(1) BIS A(I)
    3 CONTINUE
      WRITE (3,2Ø) MAX
   2Ø FORMAT (1X,6HMAX␣=␣, F15.3)
      STOP
      END
```

Allgemeine Bemerkungen

1. Da eine dynamische Felddefinition, d. h. eine Definition der Größe des Feldes erst zur Laufzeit, etwa in der Form DIMENSION A(N) nach der Anweisung READ (1,1Ø) N , nicht möglich ist, hilft man sich bei der Arbeit mit Feldern allgemein so, daß ein größtes Feld definiert wird (hier mit 999 Elementen), welches dann im allgemeinen nur zum Teil belegt wird (hier mit $N \leq 999$ Elementen).

2. Die Anweisung READ (1,11) (A(I), I = 1,N) hat als Übertragungsliste ein sogenanntes implizites DO (s. 4.4.2.1.), welches einer Aufzählung der Elemente von A entspricht. Die DO-Schleife

   ```
      DO 1 I = 1,N
    1 READ (1,11) A(I)
   ```

   ist unserer Formulierung insofern nicht äquivalent, als jede READ-Anweisung einen neuen Datensatz (Lochkarte) einliest und damit nur eine Zahl pro Lochkarte eingegeben werden kann.

Beispiel 2

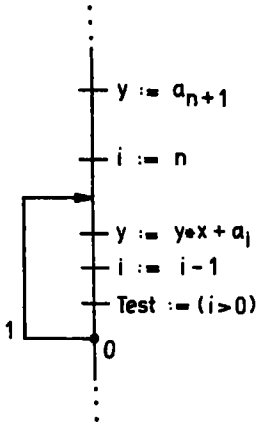
Es sei die Berechnung eines Funktionswertes des Polynoms

$$y = a_{n+1}x^n + a_n x^{n-1} + \ldots + a_2 x + a_1$$

durchzuführen (die Indizierung der Koeffizienten wurde mit 1 begonnen, da FORTRAN nur Indizes größer als Null gestattet). Zweckmäßig rechnen wir nach dem HORNERschen Schema

$$y = (\ldots((a_{n+1}x + a_n)x + a_{n-1})\ldots)x + a_1$$

oder in PAP-Darstellung



Der zugehörige FORTRAN-Programmausschnitt lautet

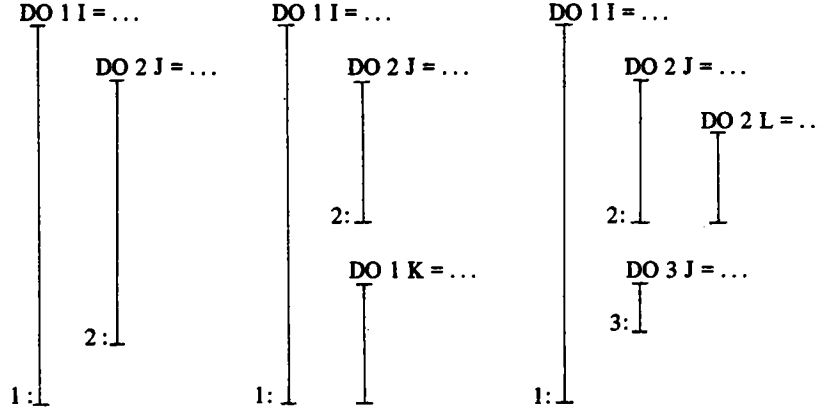
```
C PROGRAMMAUSSCHNITT
      Y = A(N+1)
      DO 1 K = 1,N
      I = N + 1 - K
    1 Y = Y*X + A(I)
```

Allgemeine Bemerkung

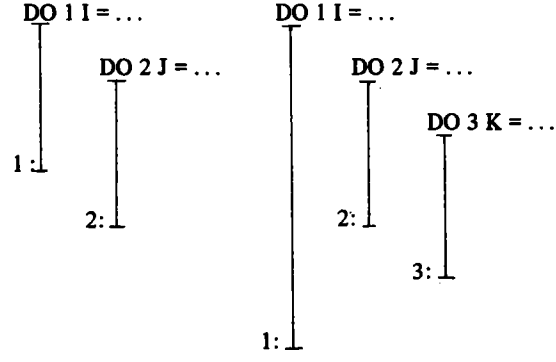
Wegen der geforderten Positivität der Schrittweite n_3 einer Schleifenanweisung ist ein „Rückwärtszählen" der Steuervariablen nicht möglich. Man hilft sich, indem man die eigentliche Steuervariable (im Beispiel I) im Schleifenbereich als „komplementäre" Variable aufbaut. I nimmt im Beispiel nacheinander die gewünschten Werte N, N–1, . . . , 2, 1 an.

4.3.2.3.2. Verschachtelte DO-Schleifen

Von *verschachtelten DO-Schleifen* spricht man dann, wenn im Schleifenbereich einer (äußeren) DO-Schleife weitere (innere) DO-Schleifen vorkommen. Grundsätzlich gilt, daß sich zwei DO-Schleifen nicht überlappen dürfen.
Betrachten wir zunächst den Fall einer inneren DO-Schleife. Dann muß diese ganz im äußeren Schleifenbereich enthalten sein. Entsprechend sind bei zwei inneren DO-Schleifen nur die Fälle möglich, daß entweder beide (nicht notwendig unmittelbar) hintereinander stehen oder eine ganz im Schleifenbereich der anderen liegt. Die möglichen Anordnungen bei mehr als drei verschachtelten DO-Schleifen sind damit offensichtlich.
Man spricht von einem *DO-Nest,* wenn jede DO-Schleife dem Schleifenbereich der vorangegangenen DO-Schleife angehört.
Wir bringen einige schematische Beispiele richtiger und falscher Schachtelungen:



Diese Beispiele sind korrekt.
Fehlerhaft infolge Überlappungen der Schleifenbereiche sind:



Bei den obigen Beispielen wurde schon angedeutet, daß eine Anweisung letzte Anweisung für mehrere DO-Schleifen sein kann.
Nunmehr einige konkrete Programmausschnitte:

a)
```
      DO 1 I = 1,M
         DO 1 K = 1,N
    1    C(I,K) = A(I,K) + B(I,K)
```

Dieses Beispiel beschreibt die Addition zweier Matrizen A und B vom Typ (M, N). Die innere Schleife wurde aus Gründen der Übersichtlichkeit eingerückt.

b)
```
      DO 1 I = 1,M
      S = Ø.Ø
         DO 2 K = 1,N
    2    S = S + A(I,K)*B(K)
    1 C(I) = S
```

Dieses Beispiel beschreibt die Multiplikation einer Matrix A vom Typ (M, N) mit einem Vektor B mit N Komponenten. Einen PAP hierzu finden Sie im Abschnitt „Beschreibung eines Programmablaufs" (vgl. Beispiel 1.4.9.3.) dieses Lehrbuchs.

c)
```
      DO 1 I = 1,N
      S = Ø.Ø
         DO 2 K = 1,M
         IF (A(I,K) - 2Ø.Ø) 2,2,3
    2    S = S + A(I,K)
      AM(I) = S/M
      GOTO 1
    3 AM(I) = 1.E1Ø
    1 CONTINUE
```

Hier werden die Mittelwerte AM(I) von N Meßreihen zu je M Meßwerten berechnet. A(I, K) ist der K-te Meßwert der I-ten Meßreihe. Enthält eine Meßreihe einen Wert >20,0, so wird sie verworfen (AM(I) = 1.E1Ø).
Das letzte Beispiel führte mittels Sprung aus der innersten DO-Schleife in die übergeordnete Schleife (Marke 3). Allgemein gilt: Das Sprungziel muß entweder außerhalb der Schachtelung oder in einer übergeordneten DO-Schleife liegen, also einer solchen, die die zu verlassende DO-Schleife umfaßt.

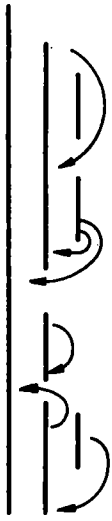


Bild 5 zulässige Sprünge

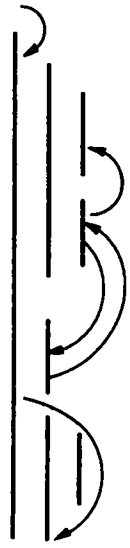


Bild 6 unzulässige Sprünge

Wir veranschaulichen in Bild 5 zulässige und in Bild 6 unzulässige Sprünge.
Ein Sprung ins Innere einer DO-Schleife ist also – wie bekannt – nicht zulässig. Es existiert eine Ausnahme. Von jeder innersten DO-Schleife eines DO-Nestes kann aus dem DO-Nest herausgesprungen und nach Abarbeitung von „äußeren" Anweisungen zu einer Anweisung des verlassenen Schleifenbereiches zurückgekehrt werden. Bedingung ist, daß dabei außerhalb des DO-Nestes weder die Steuervariable noch die Steuerparameter wertemäßig verändert werden und dort nicht erneut ein derartiger Sprung aus einer DO-Schleife vorliegt.
Wichtig ist noch die Festlegung, daß eine Anweisung, die letzte Anweisung für mehrere DO-Schleifen ist, durch einen Sprung nur von der innersten dieser DO-Schleifen erreicht werden kann.

4.3.2.4. Beispiele

Beispiel 1

Zu n Meßwerten $a_1, a_2, \ldots, a_n$ werden gesucht ($n < 1000$):

a) das arithmetische Mittel $M = (\sum_{i=1}^{n} a_i)/n$

b) die Standardabweichung $S = \sqrt{\sum_{i=1}^{n} (a_i - M)^2/(n-1)}$

c) der Prozentsatz der Werte a_i mit $|a_i - M| < S$

Bei Zugrundelegung dieser Gestalt der Formel für S würden wir nacheinander drei Schleifen (je eine für M, S, Prozentsatz) einrichten müssen. Die nachfolgende Umformung ermöglicht die simultane Berechnung der Summen in M und S und spart damit eine Schleife ein.

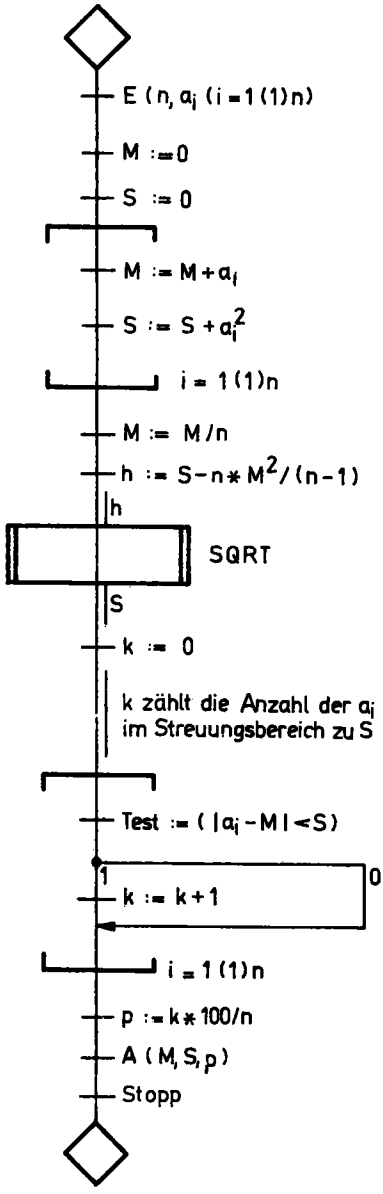
$$
\begin{aligned}
S^2 &= \Sigma (a_i - M)^2/(n-1) \\
&= (\Sigma a_i^2 - 2M\Sigma a_i + \Sigma M^2)/(n-1) \\
&= (\Sigma a_i^2 - 2MnM + nM^2)/(n-1) \\
&= (\Sigma a_i^2 - nM^2)/(n-1)
\end{aligned}
$$

Es folgt das FORTRAN-Programm und anschließend finden Sie einen Programmablaufplan zu dieser Aufgabe.

FORTRAN-Programm

```
C STATISTISCHE AUSWERTUNG
      REAL A(999), M
      READ (1,1Ø) N
   1Ø FORMAT (I3)
      READ (1,11) (A(I), I = 1,N)
   11 FORMAT (5F15.3)
      M = Ø.Ø
      S = Ø.Ø
      DO 1 I = 1,N
      M = M + A(I)
    1 S = S + A(I)**2
      M = M/N
      S = SQRT((S - N*M*M)/(N - 1))
      K = Ø
      DO 3 I = 1,N
      IF (ABS(A(I) -M)-S) 2,3,3
    2 K = K + 1
    3 CONTINUE
      P = K*1ØØ/N
C K STATT P ERGAEBE GANZZAHLIGEN PROZENTSATZ
      WRITE (3,2Ø)
   2Ø FORMAT (1X,23HSTATISTISCHE␣AUSWERTUNG)
      WRITE (3,21) M, S, P .
   21 FORMAT (1X,7H␣␣MW␣=␣, F15.4, 5X, 5HST␣=␣, F15.4, 5X,
     114HSTR␣IN␣PRZT␣=␣, F7.2)
      STOP
      END
```

Programmablaufplan



Die Ausgabe beginnt auf einer neuen Zeile mit dem Programmtitel ab Druckstelle 1 der ersten Zeile. Die Folgezeile enthält ab Druckstelle 3 die drei Werte mit Text.

Beispiel 2

Zu berechnen ist die Summe

$S = \underline{p}^T \cdot \underline{A} \cdot \underline{e}$

der Materialkosten eines Produktionsprogramms mit den gegebenen Größen:

e_j zu produzierende Menge (in Mengeneinheiten ME) des j-ten Erzeugnisses (j = 1(1)n)

$a_{i,j}$ erforderliche Menge (in ME) des i-ten Materials zur Herstellung einer ME des j-ten Erzeugnisses (i = 1(1)m, j = 1(1)n)

p_i. Preis (in M) einer ME des i-ten Materials (i = 1(1)m)

Die $a_{i,j}$ bilden die Matrix $\underline{A}$, die e_j bzw. p_i die Vektoren $\underline{e}$ bzw. $\underline{p}$. $\underline{p}^T$ ist als Transposition des Spaltenvektors $\underline{p}$ ein Zeilenvektor. Rechenfolge : $\underline{p}^T \cdot (\underline{A} \cdot \underline{e})$.
Den Teilaufgaben Matrix · Vektor und Vektor · Vektor begegneten wir bereits in 4.3.2.3.2. und 4.3.2.3.1.
Es soll nur Platz für die Matrix $\underline{A}$ und einen Vektor in Anspruch genommen werden.

Lösung

$\underline{p}$ wird nach der Berechnung von $\underline{A} \cdot \underline{e}$ auf den Platz von $\underline{e}$ eingelesen, den er überschreiten kann (bei m > n). Der Zwischenergebnisvektor $\underline{r} = \underline{A} \cdot \underline{e}$ läßt sich auf jede Spalte von $\underline{A}$, im Beispiel die erste, speichern, da nach jeder Verarbeitung einer Zeile von $\underline{A}$ diese nicht mehr benötigt wird. Wir erreichen diese Nacheinandernutzung der Speicherplätze mit einer Äquivalenzanweisung.

FORTRAN-Programm

```
C MATERIALKOSTEN
      DIMENSION A(5Ø,5Ø), E(5Ø), P(5Ø), R(5Ø)
      EQUIVALENCE (A(1), R(1)), (E(1), P(1))
      READ (1,2Ø) M, N
   2Ø FORMAT (2I2)
      READ (1,3Ø) ((A(I,J), J = 1,N), I = 1,M)
      READ (1,3Ø) (E(J), J = 1,N)
   3Ø FORMAT (1ØF8.2)
      DO 1 I = 1,M
      S = Ø.Ø
C DIESE ANWEISUNG DARF NICHT VOR DO 1 I = 1,N STEHEN,
C DA SONST FORTLAUFEND SUMMIERT WUERDE
      DO 2 J = 1,N
    2 S = S + A(I,J)*E(J)
    1 R(I) = S
      READ (1,3Ø) (P(I), I = 1,M)
```

```
   S = Ø.Ø
   DO 3 I = 1,M
 3 S = S + P(I)*R(I)
   WRITE (3,4Ø) S
4Ø FORMAT (1X,17HMATERIALKOSTEN␣:␣, F12.2, 5H␣MARK)
   STOP
   END
```

Allgemeine Bemerkung:

Die Anweisung READ (1,3Ø) ((A(I,J), J = 1,N), I = 1,M) hat als Eingabeliste ein geschachteltes implizites DO (s. 4.4.2.1.). Dieses entspricht in der Auswertungsfolge völlig zwei geschachtelten DO-Schleifen, indem zu jedem Wert des „äußeren" Index I alle Werte des „inneren" Index J abgearbeitet werden. Damit liest diese Anweisung die Matrix A zeilenweise ein.

4.3.2.5. Aufgaben

4.3.2.5./1

Welchen Wert hat die Variable A nach Abarbeitung der folgenden Verschachtelung von DO-Schleifen (DO-Nest)?

```
   A = Ø
   DO 5 I = 1,3
      DO 5 J = 2,3
      L = 3 - J
         DO 5 K = 1,4,2
5        A = -K*(A+I+L-2)
```

4.3.2.5./2

Geben Sie die im folgenden Programmstück enthaltenen drei syntaktischen Fehler und den Steuerfehler an.
Machen Sie sich nach deren Beseitigung an einem Testbeispiel klar, welche Reihenfolge der Elemente A(1) bis A(N) erzeugt wird.

```
1 DO 7 I = 2,N,1
2    DO 7 L = 1,N+1-I
3    IF (A(1+L) - A(L)) 4,2,2
4    H = A(L)
5    A(L) = A(L+1)
6    A(L+1) = H
7    GOTO 2
```

4.3.2.5./3

Erweitern Sie das Beispiel 1 aus 4.3.2.3.1. so, daß das Maximum und das Minimum von n (⩾ 2) Zahlen $a_1, a_2, \ldots, a_n$ ermittelt und ausgegeben werden.

Ö

4.3.2.5./4

Ein Betrieb stellt aus p Materialien in der ersten Produktionsstufe s Zwischenprodukte und daraus in der zweiten Produktionsstufe t Endprodukte her. Die Matrizen der Verbrauchsnormen lauten $\underline{A} = (a_{i,j})$ (i=1(1)p, j=1(1)s) und $\underline{B} = (b_{j,k})$ (j=1(1)s, k=1(1)t). Dabei gibt $a_{i,j}$ die Mengeneinheiten des i-ten Materials zur Herstellung einer Mengeneinheit des j-ten Zwischenprodukts an. Analog vermittelt $b_{j,k}$ zwischen Zwischen- und Endprodukten.
Es ist ein FORTRAN-Programm aufzustellen, das den Materialbedarfsvektor $\underline{r} = (r_i)$ bei vorgegebenem Vektor $\underline{e} = (e_k)$ der Mengen der Endprodukte ermittelt. Versuchen Sie aus Platzgründen den Speicherplatz für $\underline{A}$ und $\underline{r}$ einzusparen.

Anleitung: Orientieren Sie sich an Beispiel 2 in 4.3.2.4.

T

4.3.2.5./5

Ein gestaffeltes lineares Gleichungssystem

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + \ldots \qquad\qquad + a_{1,n}x_n &= b_1 \\ a_{i,i}x_i + a_{i,i+1}x_{i+1} + \ldots \quad a_{i,n}x_n &= b_i \\ a_{n,n}x_n &= b_n \end{aligned}$$

ist in üblicher Weise durch Rückwärtselimination zu lösen. Es wird vorausgesetzt, daß alle Hauptdiagonalelemente $a_{i,i} \neq 0$ sind.
Programmieren Sie die Eingabe so, daß alle angegebenen Koeffizienten (einschließlich der b_i) in der Reihenfolge der Gleichungen fortlaufend eingelesen werden.

Z

4.3.2.5./6

Gegeben sind zwei sortierte Zahlenfolgen

(a_i), i = 1(1)m, mit $a_i \leqslant a_{i+1}$ für i = 1(1)m − 1 und
(b_j), j = 1(1)n, mit $b_j \leqslant b_{j+1}$ für j = 1(1)n − 1

Schreiben Sie ein FORTRAN-Programm, das die sortierte Mischfolge

(c_k), k = 1(1)m + n, mit $c_k \leqslant c_{k+1}$ für k = 1(1)m + n − 1

aufbaut. Vorher ist unbedingt ein Programmablaufplan aufzustellen.

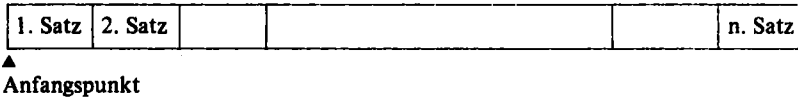
4.4. Ein- und Ausgabe von Daten

Die Programmiersprache FORTRAN enthält zahlreiche Sprachelemente zur Beschreibung des Datentransports von externen Speichermedien in interne und umgekehrt. Wir haben bereits in 4.2.4. die Struktur der mit einer Formatanweisung gekoppelten Ein- und Ausgabemöglichkeit kennengelernt. Ziel der folgenden Dar-

legungen soll es sein, das bisher Kennengelernte zu vervollständigen und teilweise zu verallgemeinern. Außerdem soll auch auf die formatfreie Datenübertragung und die Möglichkeiten der Datenorganisation eingegangen werden.

4.4.1. Dateibezug

Auf den externen Speichermedien werden die Daten zu *Dateien* zusammengefaßt. Um einen eindeutigen Bezug zwischen einem FORTRAN-Programm und den zugehörigen Daten herstellen zu können, ist jede dieser Dateien durch eine Dateinummer charakterisiert. Die Zuordnung zwischen Dateinummer und einem Ein- oder Ausgabegerät wird vom Betriebssystem der Anlage gesteuert, das Programm bleibt also (bis auf die Geräteklasse) unabhängig von einem speziellen Gerät. Für unsere Beispiele hatten wir die Nummer d = 1 der Eingabedatei für einen Kartenleser und die Nummer d = 3 der Ausgabedatei für einen Drucker zugeordnet.
Jede Datei besteht aus einer Menge von Datensätzen, wie wir sie bereits in 4.2.4. kennengelernt hatten. Für die Anordnung dieser Sätze innerhalb der Datei ist eine feste Reihenfolge gültig:



▲
Anfangspunkt

Jedem Satz mit Ausnahme des letzten folgt eindeutig ein nächster Satz, jeder Satz mit Ausnahme des ersten hat genau einen vorhergehenden Satz. In der im ESER realisierten Basis-FORTRAN-Variante bestehen zwei verschiedene Möglichkeiten, derartige Dateien zu verarbeiten:

a) sequentiell
Sequentielle Verarbeitung bedeutet, daß die Sätze einer Datei nur in einer bestimmten Reihenfolge – der fortlaufenden Zählfolge der Sätze – übertragen werden können und damit keine separate Zugriffsmöglichkeit mittels Kennzeichnung zu einem einzelnen Satz besteht, wie es für den Direktzugriff charakteristisch ist. Für die meisten externen Speichermedien stellt dies die normale Arbeitsweise dar (Lochkarte, Lochband, Magnetband). Hilfsanweisungen erlauben dabei für Magnetband und -platte das Rücksetzen einer Datei auf den Anfangspunkt oder um einen Satz (s. dazu 4.4.6.). Die folgenden Erläuterungen werden sich ausschließlich auf diese Form des Datentransfers beziehen.

b) mit direktem Zugriff
Mit der im wesentlichen für Plattendateien geschaffenen Möglichkeit kann jederzeit zu jedem beliebigen Satz einer Datei direkt Zugriff genommen und dieser übertragen werden. Dateien für den direkten Zugriff müssen mit einer DEFINE FILE-Anweisung speziell definiert werden. Ihre Verarbeitung kann dann mit READ- und WRITE-Anweisungen, die gegenüber der sequentiellen

Arbeitsweise etwas modifiziert sind, erfolgen. Eine Übersicht über die Charakteristika der Arbeit mit direktem Zugriff finden Sie im Anhang (A 4/8).

4.4.2. Ein- und Ausgabeanweisungen

Wie bereits gesagt, wollen wir uns im folgenden nur auf die sequentielle Verarbeitung von Datenmengen orientieren. Die prinzipielle Arbeitsweise haben bereits die in 4.2.4. behandelten Anweisungen READ(1,f)l und WRITE(3,f)l demonstriert. Damit können in Lochkarten gelochte Datensätze in der Reihenfolge ihres Erscheinens an der Leseeinrichtung nacheinander eingelesen und Datensätze nacheinander auf einem Zeilendrucker ausgedruckt werden.

4.4.2.1. Übertragungslisten

Als Elemente einer *Übertragungsliste* l können in Erweiterung der in 4.2.4. benutzten Variante allgemein

Variable (einfache und indizierte)
Feldnamen
implizite DO's

vorkommen. Zur Erhöhung der Übersichtlichkeit dürfen mehrere der jeweils durch Komma getrennten Listenelemente mit Hilfe von Klammern zusammengefaßt werden.
Unter einem *impliziten DO* wollen wir dabei ein Listenelement der Form

(ll,i=p_1,p_2,p_3) oder (ll,i=p_1,p_2)

verstehen, wobei ll wieder eine Übertragungsliste repräsentiert. Für die Steuervariable i und die Steuerparameter p_1,p_2,p_3 (bzw. p_1,p_2) gilt das in 4.3.2.3. für die DO-Anweisung Gesagte.
Jede Übertragungsliste legt eine feste Anordnung von einfachen und l oder indizierten Variablen fest, denen in der Reihenfolge von links nach rechts Daten zuzuordnen bzw. deren Werte zu übertragen sind. Diese Anordnung wird wie folgt ermittelt:

Namen von Variablen bleiben unverändert erhalten.
Für Feldnamen wird die Folge ihrer Elemente in der Reihenfolge ihrer Speicherung eingesetzt.
Ein implizites DO der Form (ll,I=M1,M2,M3) erzeugt eine Liste von Variablen in der durch folgende DO-Schleife charakterisierten Weise

```
  DO 1 I=M1,M2,M3
1 Anordnen von ll mit jeweiligem festen I
```

(analog bei weggelassenem M3).

Wenn zum Beispiel in einem Programm DIMENSION A(3), B(4) vereinbart wurde, sind die Listen

```
A,B
A(1),A(2),A(3),B(1),B(2),B(3),B(4)
(A(I), I=1,3),(B(J),J=1,4)
```

äquivalent. Von diesen Festlegungen haben wir bereits in mehreren Programmbeispielen Gebrauch gemacht.
Implizite DO's dürfen wie in normalen DO-Schleifen geschachtelt werden.

Beispiel

Wenn M ein zweidimensionales Feld der Größe M(2,3) bezeichnet, bewirken

M	Anordnung der Feldelemente spaltenweise entsprechend Speicherung M(1,1),M(2,1), M(1,2),M(2,2),M(1,3),M(2,3)
((M(I,J),J=1,3),I=1,2)	Anordnung der Feldelemente zeilenweise M(1,1),M(1,2),M(1,3),M(2,1),M(2,2),M(2,3)

4.4.2.2. Formatgebundene und formatfreie Ein- und Ausgabeanweisungen

In FORTRAN besteht die Möglichkeit, die Übertragung von Daten unmittelbar in der internen Darstellungsform vorzunehmen oder sie durch Zuordnung einer Formatanweisung mit einer Aufbereitung der Daten zu verbinden. Folgende *sequentielle E/A-Anweisungen* stehen zur Verfügung

READ(d,f)l	READ(d)l
WRITE(d,f)l	WRITE(d)l

Je nachdem, ob der Bezug f auf eine Formatanweisung vorhanden ist oder nicht, spricht man von *formatgebundenen* oder *formatfreien E/A-Anweisungen.*
Die beiden formatgebundenen Anweisungen haben wir bereits in 4.2.4. ausführlich kennengelernt. Während diese in der Hauptsache bei der Eingabe von Primärdaten (zum Beispiel von Lochkarten) und bei der Ausgabe von Ergebnissen auf langsamen peripheren Geräten (zum Beispiel Drucker, Lochkartenstanzer) Anwendung finden, wird die Zwischenspeicherung von Daten (meist auf Magnetbändern und Platten) im allgemeinen in der internen Darstellung, also formatfrei, vorgenommen. Die Anweisung WRITE(d)l gibt die in der Übertragungsliste l aufgeführten Variablen in der Reihenfolge von links nach rechts in interner Form als nächsten Satz der Datei d aus. Umgekehrt bewirkt READ(d)l die Eingabe des nächsten Satzes der Datei d und die Zuweisung zu den Elementen von l. Ein mit Format ausgegebener Satz kann selbstverständlich nicht durch eine formatfreie READ-Anweisung eingelesen werden, ein mit READ(d)l zu lesender Satz muß vorher durch WRITE(d)l geschrieben worden sein.
Mit Ausnahme der Anweisung WRITE(d)l kann bei den übrigen E/A-Anweisungen in speziellen Anwendungsfällen die Liste l entfallen. Praktisch genutzt wird dieser Umstand bei formatgebundenen E/A-Vorgängen, wenn in der Formatliste

nur solche Formate stehen, die kein entsprechendes Element der Übertragungsliste erfordern (s. Beispiel 2 in 4.2.7.). Mit der Anweisung READ(d), die formatfrei und ohne Übertragungsliste ist, kann man einen Satz einlesen, ohne ihn irgendwelchen Variablen zuzuweisen. Der Satz wird im Prinzip übergangen und der folgende zur Verarbeitung bereitgestellt.

4.4.2.3. Druckervorschubsteuerung

Werden Daten über einen Drucker ausgegeben, besteht für den Programmierer eine wesentliche Aufgabe darin, die Daten in einem übersichtlichen Druckbild anzuordnen. Dabei müssen die gerätetechnischen Parameter wie Anzahl der Druckpositionen pro Zeile und Anzahl der Druckzeilen pro Seite unbedingt beachtet werden. Bei den im ESER verwendeten Druckern stehen im allgemeinen 120 oder 156 Druckpositionen pro Zeile und maximal etwa 65 Zeilen pro Seite zur Verfügung. Zur vertikalen Gliederung des Druckbildes werden die Möglichkeiten der Papiervorschubsteuerung genutzt. Diese erfolgt durch das erste Zeichen des auszugebenden Datensatzes entsprechend folgender Tabelle:

erstes Zeichen	Vorschubsteuerung
+	kein Papiervorschub
1	Papiervorschub auf erste Zeile der nächsten Seite
Ø	Papiervorschub auf übernächste Zeile
Leerzeichen	Papiervorschub auf nächste Zeile

Beachten Sie dabei, daß zu dem auszugebenden Datensatz außer den Werten der in der Übertragungsliste stehenden Variablen auch die in der Formatliste aufgeführten Leerzeichen des X-Formats und die beliebigen Zeichen aus dem H-Format gehören.
Am einfachsten realisiert man den zur Druckbildgestaltung erforderlichen Papiervorschub durch ein H-Format als erstes Element der Formatliste (1H␣ oder 1X bei Papiervorschub um eine Druckzeile, 1HØ bei Vorschub um zwei Druckzeilen, 1H1 bei Übergang zur nächsten Seite und 1H+, wenn kein Vorschub gewünscht wird). Folgt dieser Angabe ein weiteres H-Format, können beide zusammengefaßt werden (5H1Y␣=␣ ist beispielsweise äquivalent zu 1H1,4HY␣=␣).
Für die Anwendung der in obiger Tabelle zusammengestellten Möglichkeiten sind folgende Regeln zu beachten:
Der Papiervorschub erfolgt stets vor dem Drucken des jeweiligen Datensatzes.
Das erste den Papiervorschub steuernde Zeichen des Datensatzes wird nicht mit ausgedruckt.

4.4.3. Formatanweisung

Nachdem wir bereits in 4.2.4. die *Formatanweisung* in einer vereinfachten Variante kennengelernt haben, hier nun ihre allgemeine Form:

FORMAT(fl)

Als Elemente der *Formatliste* fl können einzelne Formate, Formatgruppen, Positionierungen und Satztrennzeichen vorkommen. Eine Zusammenstellung aller Formate werden wir in 4.4.4. vornehmen. Die Möglichkeit einer Positionierung von Datenfeldern ist im Anhang (A 4/7) aufgeführt. Als Satztrennzeichen wird der Schrägstrich / verwendet. Es ist möglich, mehrere Satztrennzeichen hintereinander zu setzen. Steht zwischen zwei Elementen der Formatliste ein Satztrennzeichen, kann das Komma als Trennzeichen zwischen den Listenelementen entfallen. Die Funktion des Satztrennzeichens werden wir in 4.4.4.3. kennenlernen.
Für eine Reihe von Formaten hatten wir bereits in 4.2.4. die Möglichkeit erläutert, zur Verkürzung der Formatliste den Formaten einen Wiederholungsfaktor voranzustellen. Er gibt an, wievielmal das Format hintereinander gelten soll. Mehrere Formatlistenelemente lassen sich durch Einklammern zu einer Gruppe zusammenfassen. Analog zu einzelnen Formaten kann auch solch eine *Formatgruppe* mit einem Wiederholungsfaktor versehen werden. Dabei darf in einer Formatgruppe keine weitere Gruppe als Listenelement auftreten, eine Schachtelung ist also nicht erlaubt.

Beispiele

mit Wiederholungsfaktoren	äquivalente Darstellung
`FORMAT(I4/3F6.1)`	`FORMAT(I4,/F6.1,F6.1,F6.1)`
`FORMAT(//2(I6,2X))`	`FORMAT(//(I6,2X,I6,2X))`
`FORMAT(2X,3(I2/2I4),15X,2F8.2)`	`FORMAT(2X,(I2/I4,I4,I2/I4,I4,I2/1I4,I4),5X,F8.2,F8.2)`

Bei der Auflösung des Wiederholungsfaktors vor einer Formatgruppe werden die Gruppenklammern als Gesamtbegrenzung nach außen gezogen.
Die Formatgruppe spielt im Ablauf der Formatsteuerung eine wichtige Rolle. Beim Vorhandensein von Formatgruppen in der Formatliste wird nämlich in dem Fall, daß nach Abarbeitung der Formatliste in der zugehörigen Übertragungsliste noch weitere Variable vorkommen, die Reihenfolge der Formatabarbeitung mit der am weitesten rechts stehenden Formatgruppe fortgesetzt. Auf diesen Sachverhalt werden wir bei der zusammenfassenden Darstellung des gesamten E/A-Vorganges (4.4.5.) noch zurückkommen.

4.4.4. Formatlistenelemente

In diesem Abschnitt wollen wir alle Möglichkeiten, die FORTRAN zur Datenfeldbeschreibung bietet, systematisch zusammenstellen und neben den bereits bekannten I-, F-, H- und X-Formaten einige weitere kennenlernen.

4.4.4.1. Datenfeldbeschreibungen für numerische Werte

Hierzu sind zu zählen:

I-Format F-Format E-Format[1]

Das E-Format in der Form

Ew.d ($1 \leq w \leq 254$, $0 \leq d \leq w$)

wird bei der Übertragung von reellen Daten einfacher oder doppelter Genauigkeit in Exponentendarstellung zwischen einem externen Speichermedium und Variablen vom Typ REAL oder DOUBLE PRECISION verwendet. Die Entscheidung zwischen einfacher oder doppelter Genauigkeit hängt vom Typ des zugehörigen Elementes der Übertragungsliste ab. d legt wie beim F-Format die Stellen rechts vom Dezimalpunkt fest, gerechnet bis zur letzten nicht zum Exponenten gehörenden Stelle. Die folgenden Beispiele sollen die Arbeit mit dem E-Format charakterisieren.

Eingabe im Format E6.2

eingegebene Zeichenfolge	interner Wert
␣-1234	$-0{,}1234 \cdot 10^{2}$
␣␣␣+1␣	$0{,}1 \cdot 10^{0}$
␣338.4	$0{,}3384 \cdot 10^{3}$
27␣+1␣	$0{,}27 \cdot 10^{11}$
-.66E2	$-0{,}66 \cdot 10^{2}$
79ØE+1	$0{,}79 \cdot 10^{2}$
86.D-8	$0{,}86 \cdot 10^{-6}$

Für die Eingabe im E-Format sind beliebige Zahlendarstellungen zugelassen. Ein Dezimalpunkt im eingelesenen Datenfeld setzt die Wirkung von d außer Kraft. Die Behandlung der Leerzeichen und Vorzeichen erfolgt analog zum F-Format.

Ausgabe im Format E8.2

auszugebender Wert	ausgegebene Zeichenfolge
6,28	Ø.63E␣Ø1
-0,02	-.2ØE-Ø1
1234,5	Ø.12E␣Ø4
$7{,}2\ 10^{-18}$	Ø.72E-17
0,0	Ø.Ø␣␣␣␣␣

1 FORTRAN besitzt zur Übertragung doppeltgenauer Zahlen noch ein spezielles D-Format. Da in der Basis-FORTRAN-Implementierung im DOS/ES dieser Datentransport auch mit dem E-Format gesteuert werden kann, wollen wir auf eine gesonderte Behandlung dieses D-Formates verzichten. Die hier und in den folgenden Ausführungen zum E-Format angegebenen Aussagen treffen vollinhaltlich auch auf das D-Format zu.

Generell steht die erste von Null verschiedene Dezimalstelle unmittelbar rechts vom Dezimalpunkt. Für den Exponententeil sind konstant vier Stellen vorgesehen, bei der Ausgabe der Null erscheinen anstelle des Exponententeiles vier Leerpositionen. Vor dem Dezimalpunkt wird, sofern die Feldweite w Platz dafür beläßt, eine Null ausgegeben. Wenn die Zahl der unbedingt auszugebenden Zeichen (Exponententeil, Dezimalpunkt, negatives Vorzeichen, d Ziffernstellen) die Feldweite w überschreitet, werden Sonderzeichen * ausgegeben. Das eventuelle Verkürzen der auszugebenden Mantisse entsprechend der Angabe von d erfolgt mit Rundung. Wird der Wert einer Variablen vom Typ DOUBLE PRECISION ausgegeben, erscheint anstatt des Exponentensymbols E ein D.
Bei Verwendung der F- und E-Formate kann durch die zusätzliche Angabe eines *Skalenfaktors* eine Modifizierung des im F-Format übertragenen Wertes um eine Zehnerpotenz bzw. seiner Mantissennormierung bei der Ausgabe im E-Format erfolgen. Die genaue Wirkung dieses Skalenfaktors in Abhängigkeit vom gewählten Format ist im Anhang (A 4/7) zusammengestellt.

4.4.4.2. Datenfeldbeschreibungen für Zeichenfolgen

Zur Übertragung von Zeichenfolgen, die aus beliebigen FORTRAN-Zeichen bestehen können,[2] stehen das H- und das A-Format zur Verfügung.

Ergänzung zum H-Format

Zu der in 4.2.4. angegebenen Darstellung $wHc_1c_2\ldots c_w$ existiert als äquivalente Darstellung $'c_1c_2\ldots c_w'$ mit der zusätzlichen Bedingung, daß innerhalb der Apostrophzeichen auftretende Apostrophe stets doppelt zu schreiben sind.

A-Format

Das *A-Format* in der Form

Aw $(1 \leqslant w \leqslant 254)$

dient der Übertragung von beliebigen Zeichenfolgen zwischen externen Speichermedien und Variablen beliebigen Typs. Wenn g die Anzahl der in einer Speichereinheit des Übertragungslistenelements unterzubringenden Zeichen angibt (bei ESER-Anlagen g=4 für Variable vom Typ INTEGER und REAL und g=8 für Variable vom Typ DOUBLE PRECISION), wird die Handhabung des A-Formats durch die folgende Übersicht charakterisiert:

Eingabe:	$w \leqslant g$	Übertragung von w Zeichen linksbündig,	Auffüllen mit $g - w$ Leerzeichen rechts
	$w > g$	Übertragung von g Zeichen rechtsbündig,	Wegfall der $w - g$ linken Zeichen

2 Die Zeichenfolgen können auch noch weitere, für die jeweilige Rechenanlage zulässige Zeichen enthalten.

Ausgabe: $g \leqslant w$ Übertragung von g Zeichen rechtsbündig, Auffüllen mit $w - g$ Leerzeichen links

$g > w$ Übertragung von w Zeichen linksbündig, Wegfall der $g - w$ rechten Zeichen

Beispiele

Format für Eingabe	eingegebene Zeichenfolge	interne Darstellung
A4	MARK	MARK
A6	MARK␣␣	RK␣␣
A2	X1	X1␣␣

Format für Ausgabe	interne Darstellung	ausgegebene Zeichenfolge
A4	OTTO	OTTO
A6	MARK	␣␣MARK
A2	ABCD	AB

4.4.4.3. Zwischenraumbeschreibung und Satztrennzeichen

Im Interesse der Vollständigkeit sei die in 4.2.4. bereits behandelte Möglichkeit zur Beschreibung von Zwischenräumen mit Hilfe des X-Formats

wX $(1 \leqslant w \leqslant 254)$

(Übergehen von w Zeichen bei der Eingabe, Einsetzen von w Leerzeichen bei der Ausgabe) an dieser Stelle noch einmal genannt.
Die Verwendung des *Satztrennzeichens* / bewirkt auf Grund seiner Bedeutung für die Definition von Datensätzen (vgl. 4.4.5.) folgendes:
n Schrägstriche am Anfang oder am Ende der Formatliste definieren n leere Sätze. Das bedeutet für die Eingabe das Übergehen von n Sätzen (zum Beispiel Lochkarteninhalten), für die Ausgabe das Übertragen von n leeren Sätzen (zum Beispiel Ausgabe von ungelochten Karten, Ausgabe von Leerzeilen auf dem Drucker). Ein Schrägstrich im Inneren der Formatliste beendet die Ein- oder Ausgabe des jeweiligen Satzes, er veranlaßt also beispielsweise die Eingabe der nächsten Lochkarte oder die Ausgabe des Datensatzes in eine Druckzeile mit einem Papiervorschub vor dem Druckvorgang gemäß des ersten Zeichens des Datensatzes. Folgen diesem Schrägstrich weitere k Schrägstriche, definieren diese analog zu oben k leere Sätze. Bei einer Ausgabe über den Drucker ist die Papiervorschubsteuerung für den nachfolgenden Druckvorgang davon unberührt. Ein nachfolgender Zeilendruck mit einem + (kein Papiervorschub) als erstem Zeichen im Datensatz würde in die letzte Leerzeile erfolgen.

4.4.5. Zusammenfassende Darstellung eines vollständigen formatgebundenen Ein- und Ausgabevorganges

An sechs Beispielen hatten wir in 4.2.4. die wichtigsten Grundregeln für das Zusammenwirken von Übertragungs- und Formatliste kennengelernt (Formatsteuerung). Wir wollen nun den gesamten Ablauf noch einmal zusammenfassend unter Einbeziehung aller am Ein- und Ausgabevorgang beteiligten Größen betrachten. Im Anhang (A 4/9) finden Sie den Ablauf der sequentiellen formatgebundenen Ein- und Ausgabeanweisungen in Form von Diagrammen dargestellt.
Der Vorgang der Ein- oder Ausgabe überträgt Daten von einem externen Speichermedium in die den Variablen der Übertragungsliste zugeordneten Speichereinheiten bzw. in umgekehrter Richtung. Zu jeder Variablen der aufgelösten Übertragungsliste (vgl. 4.4.2.1.) gehört beim formatgebundenen Datentransport ein I-, F-, E- oder A-Format aus der zugeordneten Formatliste in der gleichen Reihenfolge von links nach rechts. Nur zu den H- und X-Formaten gibt es keine Entsprechungen in der Übertragungsliste. Wenn in der aufgelösten Übertragungsliste mehr Variablennamen stehen als I-, F-, A- und E-Formate in der Formatliste unter Berücksichtigung eventueller Wiederholungsfaktoren (vgl. Beispiel 3 in 4.2.4.), wird die Reihenfolge der Formate mit der letzten, das heißt der am weitesten rechts stehenden Formatgruppe fortgesetzt. Wenn es eine derartige Formatgruppe nicht gibt, wird die Formatliste von ihrem Anfang an wiederholt abgearbeitet. Der durch eine READ- oder WRITE-Anweisung ausgelöste Ein- bzw. Ausgabevorgang läuft so lange ab, bis entweder

a) das Ende der Formatliste erreicht und kein Element der Übertragungsliste mehr vorhanden ist
   oder, wenn das nicht der Fall ist,
b) für ein I-, F-, A- oder E-Format (eventuell bei der wiederholten Abarbeitung der Formatliste bzw. ihrer letzten Formatgruppe) kein entsprechendes Element der Übertragungsliste mehr vorhanden ist.

Damit ist zugleich festgelegt, inwieweit nach dem letzten berücksichtigten I-, F-, A- oder E-Format noch H- und X-Formate ausgeführt werden.
Mit Kenntnis dieses prinzipiellen Ablaufs können wir den bereits benutzten Begriff des Datensatzes bei der formatgebundenen Ein- und Ausgabe präzisieren. Bei Formatanweisungen ohne Satztrennzeichen stellt die mit dem einmaligen Abarbeiten der Formatliste erfaßte Datenmenge einen Datensatz dar. Sind Satztrennzeichen vorhanden, unterteilt sich die übertragene Datenmenge in mehrere Sätze entsprechend der Satztrennzeichen. Dabei definieren zwei unmittelbar aufeinanderfolgende Satztrennzeichen, ein unmittelbar nach der öffnenden Klammer der Formatliste folgendes und ein unmittelbar vor der schließenden Klammer der Formatliste stehendes Satztrennzeichen jeweils einen leeren Satz (vgl. dazu 4.4.4.3.).
Wenn nach vollständiger Abarbeitung der Formatliste zur letzten Formatgruppe oder bei deren Fehlen zum Anfang der Formatliste zurückgegangen wird, beginnt in jedem Falle ein neuer Satz. So bewirkt die Formatanweisung

```
FORMAT(1H1,11HERGEBNISSE://2(11X,F10.3)/)
```

die folgende Satzstrukturierung für auszudruckende Daten:

Satz 1 – ERGEBNISSE: (mit vorherigem Papiervorschub zur nächsten Seite)
Satz 2 – leerer Satz (Leerzeile)
Satz 3 – 2 Zahlen im Format F1Ø.3 mit 10 bzw. 11 Leerzeichen davor
(mit vorherigem Papiervorschub auf nächste Zeile)
Satz 4 – leerer Satz (Leerzeile)
Satz 5 – 2 Zahlen im Format F1Ø.3 mit Leerzeichen und Vorschub wie Satz 3
Satz 6 – leerer Satz (Leerzeile)
usw.

Zwei Beispiele sollen den Sachverhalt noch weiter erläutern.

Beispiel 1

```
    READ(1,1Ø)N,(X(I),I=1,N)
1Ø  FORMAT(/I4/ 8(F8.3,2X))
```

Die erste Karte des einzugebenden Lochkartenstapels wird wegen des Schrägstriches zu Beginn der Formatliste nicht in den Speicher übertragen. In der folgenden Lochkarte steht N als ganze Zahl in den Spalten 1 bis 4. Ein Satztrennzeichen bewirkt den Beginn eines neuen Satzes, d. h. das Einlesen der nächsten Lochkarte. Auf den folgenden Lochkarten stehen jeweils 8 Komponenten des Vektors X, gelocht in den Spalten 1 bis 8, 11 bis 18, 21 bis 28 usw. Wenn die Formatgruppe (F8.3,2X) achtmal abgearbeitet wurde, wird mit Erreichen der Endklammer der Formatliste ein Satz beendet, also zur nächsten Lochkarte übergegangen und die Formatgruppe erneut verwendet. Der Eingabevorgang ist nach dem Einlesen der N-ten Komponente des Vektors X abgeschlossen. Auf der letzten Karte können weniger als 8 Elemente stehen.

Beispiel 2

```
    WRITE(3,2Ø)A,B,E,C,D,F,X,Y
2Ø  FORMAT(1H1,'KONTROLLDRUCK␣EINGANGSDATEN:'/11X,3
   1(1ØX,F1Ø.2)/11X,3(1ØX,F1Ø.2)/1X,'LOESUNGEN:',2(
   2F1Ø.2,1ØX))
```

Die drei Ausgabeanweisungen des Beispiels 2 in 4.2.7. sind hier zu einer zusammengefaßt worden. Die Satztrennzeichen zwischen den Formaten beenden jeweils einen Satz, also im Normalfall eine Druckzeile. Das erste Zeichen eines jeden Satzes wird unverändert zur Steuerung des Papiervorschubs vor dem Druck des Satzes genutzt. Insgesamt bewirkt obiges Anweisungspaar das Ausdrucken von vier Datensätzen in der in 4.2.7. skizzierten Anordnung.
Abschließend sei noch bemerkt, daß natürlich die von den Formaten der zur E/A-Anweisung gehörenden Formatliste geforderten Formen der Eingangsdaten und Typen der Variablen eingehalten werden müssen. Bei jedem Ausgabevorgang ist darauf zu achten, daß die Summe der Feldweiten der zu einem Satz gehörenden Formate bestimmte maschinenbedingte Größen nicht überschreiten darf (80 Spalten bei Lochkarten, 121 oder 157 Druckpositionen bei ESER-Druckern).

4.4.6. Hilfsanweisungen für die Ein- und Ausgabe

Z

Wenn mit sequentiellen Dateien auf Magnetband oder -platte gearbeitet wird, bedarf es, bedingt durch die Besonderheiten der sequentiellen Arbeitsweise, einiger zusätzlicher Anweisungen:

REWIND d	Rückspulanweisung
BACKSPACE d	Rücksetzanweisung
ENDFILE d	Anweisung zum Dateiabschluß.

Alle drei Anweisungen sind ausführbare Anweisungen. Die *REWIND-Anweisung* setzt die Datei d auf ihren Anfangspunkt, spult also das entsprechende Magnetband auf seinen Startpunkt zurück. Mit Hilfe der *BACKSPACE-Anweisung* kann die Datei d um einen Satz zurückgesetzt werden, kann also der Anfang des vorhergehenden Satzes unter den Lese-/Schreibmechanismus des Gerätes gebracht werden. Stand die Datei schon auf ihrem Anfangspunkt, wird die BACKSPACE-Anweisung ignoriert. Mit der *ENDFILE-Anweisung* schließlich ist es möglich, einen Schlußsatz zur Kennzeichnung des Dateiendes zu schreiben.
Die prinzipielle Vorgehensweise bei der sequentiellen Arbeit mit einem Magnetband als Zwischenspeicher soll durch das folgende Beispiel demonstriert werden. Das zweidimensionale Feld MATRIX, bestehend aus 20 Zeilen zu je 100 Zahlen, soll auf einem Magnetband in interner Darstellung mit der Dateinummer 13 zwischengespeichert werden. Eine Zeile bildet dabei einen Datensatz. Der Schreibvorgang kann mit den Anweisungen

```
  DO 1 I=1,2Ø
1 WRITE(13) (MATRIX(I,J),J=1,1ØØ)
  ENDFILE 13
```

realisiert werden. Die formatfreie Anweisung 1 wird zwanzigmal hintereinander ausgeführt und schreibt dabei jedesmal einen Satz auf Magnetband. Soll nun eine bestimmte Zeile wieder eingelesen werden, bedarf es entsprechender Magnetbandbewegungen, um das Band vom augenblicklichen Stand in die zum Lesen des gesuchten Satzes notwendige Position zu bringen. So würde das Einlesen der Zeile 15 und ihr Zuordnen zum Vektor MZ15 im Anschluß an den obigen Schreibvorgang die folgenden Anweisungen erfordern:

```
  DO 2 I=1,7
2 BACKSPACE 13
  READ(13) (MZ15(I),I=1,1ØØ)
```

Soll nach erfolgtem Schreiben zu einem späteren Zeitpunkt die gesamte Datei wieder eingelesen werden, kann dies durch

```
  REWIND 13
  DO 3 I=1,2Ø
3 READ(13) (MATRIX(I,J),J=1,1ØØ)
```

geschehen.

Ö

4.4.7. Beispiele

Beispiel 1

Nehmen wir an, daß in einem Betrieb die Lagerbestände in Form einer Lochkartendatei registriert sind. Für jeden im Lager geführten Artikel enthält eine Lochkarte die erforderlichen Informationen:

Spalten 1 bis 5 Sachnummer
Spalten 7 bis 30 Bezeichnung
Spalten 58 bis 59 Mengeneinheit
Spalten 61 bis 67 Preis pro Mengeneinheit
Spalten 69 bis 75 Menge.

Das nachfolgende FORTRAN-Programm druckt eine Liste der Lagerbestände aus, die für jeden Artikel die Informationen Sachnummer, Bezeichnung, gelagerte Menge mit Mengeneinheit und Wert dieser Lagerposition in der Form

```
 SACHNUMMER          BEZEICHNUNG                    MENGE              WERT

    xxxxx      xxxxxxxxxxxxxxxxxxxxxxxxx      xxxxxxx xx      x.xxxxxxExxx MARK
    xxxxx      xxxxxxxxxxxxxxxxxxxxxxxxx      xxxxxxx xx      x.xxxxxxExxx MARK
      :                  :                          :                :
      :                  :                          :                :
GESAMTWERT DES BESTANDES = x.xxxxxxExxx MARK
```

enthält. Nach dem Ausdrucken der gesamten Lochkartendatei, deren Ende durch eine leere Karte angezeigt sein soll, wird abschließend der Gesamtwert des Bestandes ausgedruckt.

```
C AUSDRUCKEN EINER BESTANDSKARTEI
      INTEGER SACHNR,BEZEI(6)
C DAS FELD BEZEI NIMMT IN SEINEN 6 KOMPONENTEN DIE 24 ZEICHEN
C DER BEZEICHNUNG AUF (TYP KANN BELIEBIG GEWAEHLT WERDEN).
C
      WRITE(3,1Ø)
   1Ø FORMAT('1SACHNUMMER',1ØX,'BEZEICHNUNG',15X,'MENGE',
     115X,'WERT'//)
      GESAMT = Ø.Ø
    1 READ(1,11) SACHNR,(BEZEI(I),I=1,6),EINH,PREIS,MENGE
   11 FORMAT(I5,1X,6A4,27X,A2,1X,F7.2,1X,I7)
      IF(SACHNR)3,3,2
C BEI ERREICHEN DER SCHLUSSKARTE WIRD MIT ANWEISUNG 3
C FORTGESETZT.
    2 WERT = MENGE*PREIS
      GESAMT = GESAMT+WERT
      WRITE(3,12) SACHNR,BEZEI,MENGE,EINH,WERT
   12 FORMAT(4X,I5,6X,6A4,6X,I7,1X,A2,5X,E12.6,'␣MARK')
      GOTO 1
```

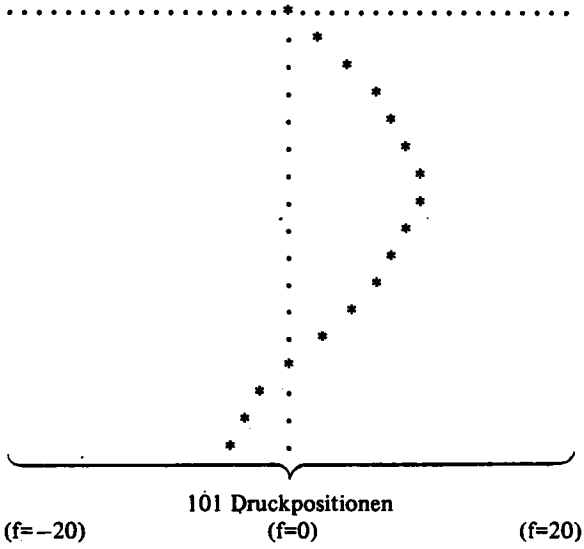
```
    3 WRITE(3,13) GESAMT
   13 FORMAT('ØGESAMTWERTUDESUBESTANDESU=U',E12.6,'UMARK')
      STOP
      END
```

Das Austesten der Schlußkarte beruht auf der Tatsache, daß die Leerzeichen (keine Lochung) in den Spalten 1 bis 5 bei der Eingabe im I-Format für SACHNR den Wert Null ergeben, der durch eine IF-Anweisung abgefragt werden kann.

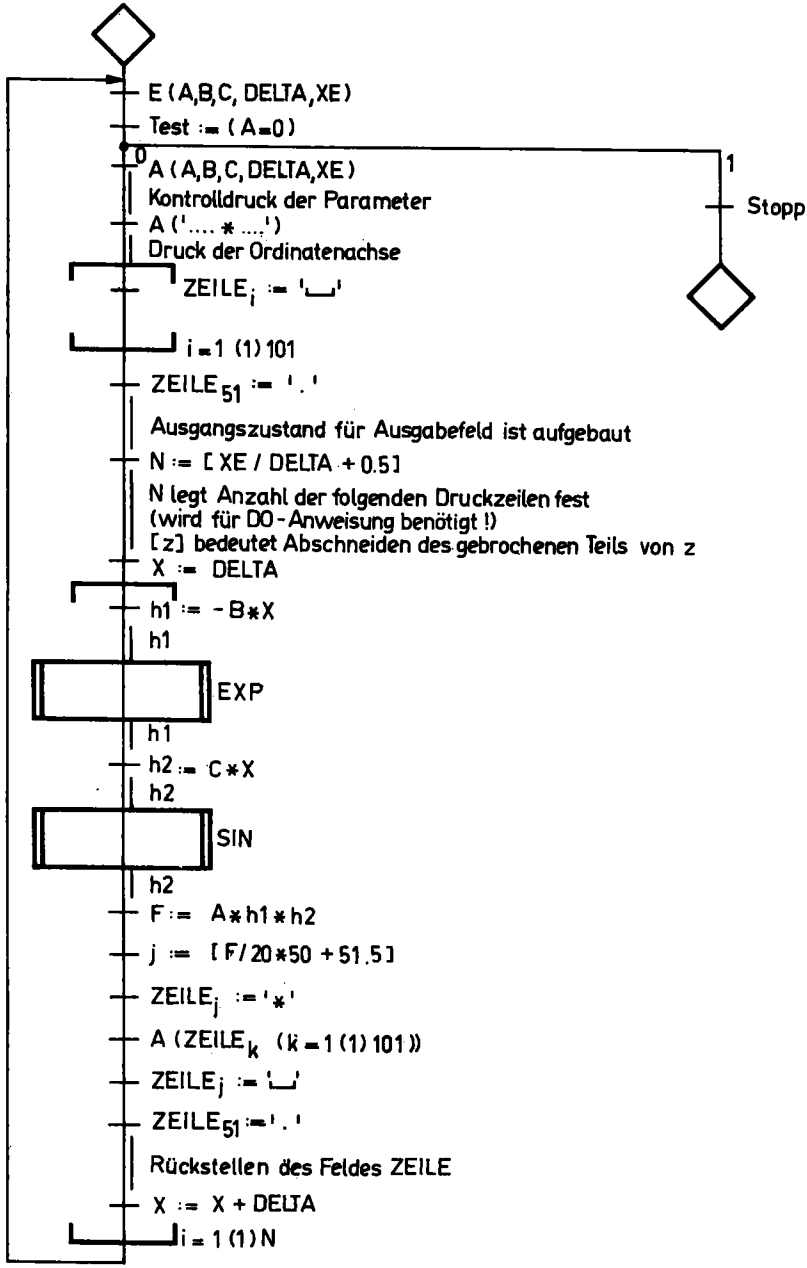
Beispiel 2

T

Die Funktion $f(x) = a \cdot e^{-bx} \sin cx$ soll im Bereich $0 \leq x \leq x_E$ mit der Schrittweite Δx für mehrere Parameterfälle ($|a| \leq 20, a \neq 0, b > 0$) in der Form



graphisch dargestellt werden. Die Parameter werden als A, B, C, DELTA und XE von einer Lochkarte eingelesen, den Abschluß der Parameterkarten bildet eine Leerkarte. Während die Ordinatenachse gesondert gedruckt wird, ist zum Druck der folgenden Zeilen, die jeweils einem Abszissenwert entsprechen, der Aufbau eines eindimensionalen Ausgabefeldes ZEILE sinnvoll. Nach der Berechnung eines Funktionswertes F bestimmt sich die Position des zugehörigen Zeichens * im Ausgabefeld nach J = F/2Ø*5Ø+51.5. Ist der Druck erfolgt, wird die Belegung dieses Ausgabefeldes auf den Anfangszustand (100 Leerzeichen und ein Punkt) zurückgestellt.
Verdeutlichen wir uns den Algorithmus zunächst in einem Programmablaufplan (s. S. 72).


E (A,B,C, DELTA,XE)
Test := (A=0)
0
A (A,B,C, DELTA,XE)
Kontrolldruck der Parameter
A ('.... * ....')
Druck der Ordinatenachse
ZEILE$_i$:= '␣'
i = 1 (1) 101
ZEILE$_{51}$:= '.'
Ausgangszustand für Ausgabefeld ist aufgebaut
N := [XE / DELTA + 0.5]
N legt Anzahl der folgenden Druckzeilen fest
(wird für DO-Anweisung benötigt !)
[z] bedeutet Abschneiden des gebrochenen Teils von z
X := DELTA
h1 := -B*X
h1
EXP
h1
h2 := C*X
h2
SIN
h2
F := A*h1*h2
j := [F/20*50 + 51.5]
ZEILE$_j$:= '*'
A (ZEILE$_k$ (k = 1 (1) 101))
ZEILE$_j$:= '␣'
ZEILE$_{51}$:= '.'
Rückstellen des Feldes ZEILE
X := X + DELTA
i = 1 (1) N
1
Stopp

Der Programmlauf endet, wenn nach den zu bearbeitenden Parameterfällen die Leerkarte eingelesen worden ist. Dies wird durch Prüfung von A auf den Wert Null realisiert, da die Leerzeichen bei der Eingabe Nullen ergeben und der Fall A=0 unter den vorgegebenen Parameterwerten sinnvollerweise nicht enthalten ist.

FORTRAN-Programm

```
C GRAPHISCHE DARSTELLUNG VON FUNKTIONEN
      DIMENSION ZEILE(101)
      WRITE(3,10)
   10 FORMAT(1H1,'GRAPHISCHE␣DARSTELLUNG␣VON␣F(X)=A*EXP
     2(-B*X)*SIN(C*X)')
C DRUCK DER UEBERSCHRIFT
    1 READ(1,11)A,B,C,DELTA,XE,STERN,PUNKT,LEER
   11 FORMAT(5E10.2,27X,3A1)
C STERN,PUNKT,LEER NEHMEN DIE ZUM AUFBAU DES AUSGABEFELDES
C ERFORDERLICHEN FORTRAN-ZEICHEN AUF, DIE MIT DEN PARAMETERN
C EINGEGEBEN WERDEN.
      IF(A)5,5,2
    2 WRITE(3,12) A,B,C,DELTA,XE
   12 FORMAT('0A␣=␣',E10.2,6X,'B␣=␣',E10.2,6X,'C␣=␣',E10.2,6X,
     1'DELTA-X␣=␣',E10.2,6X,'XE␣=␣',E10.2//)
C
      WRITE(3,13)(PUNKT,I=1,100)
   13 FORMAT(11X,50A1,1H*,50A1)
      DO 3 I=1,101
    3 ZEILE(I) = LEER
      ZEILE(51) = PUNKT
   50 N = XE/DELTA+0.5
      X = DELTA
C
      DO 4 I=1,N
      F = A*EXP(-B*X)*SIN(C*X)
   51 J = F/20.0*50.0+51.5
      ZEILE(J) = STERN
      WRITE(3,14) ZEILE
   14 FORMAT(11X,101A1)
      ZEILE(J) = LEER
      ZEILE(51) = PUNKT
    4 X = X+DELTA
C
      GOTO 1
    5 STOP
      END
```

Bei den Anweisungen 50 und 51 ist zu beachten, daß die berechneten Ausdrücke vom Typ REAL sind, ihre Werte aber Variablen vom Typ INTEGER zugeordnet werden, wobei jeweils nur der ganze Teil der reellen Größen berücksichtigt wird

(intern Einsatz der Standardfunktion IFIX, s. Anhang A 4/4). Um ein Runden auf die nächstgelegene ganze Zahl zu erzielen, muß deshalb vorher eine Korrekturgröße 0.5 hinzugefügt werden.

4.4.8. Aufgaben

4.4.8./1

Welche Werte werden bei der Eingabe der angegebenen Zeichenfolgen unter Verwendung der numerischen Formate erzeugt?

Zeichenfolge	Format	Zeichenfolge	Format
␣␣1Ø28␣4	I8	␣␣␣1.2␣3	F8.1
	F8.2		E8.1
	E8.4		I8
␣-86.3E2	E8.1	-.33E-2Ø	E8.2
␣+912-12	E8.2	+178␣D+3	E8.3

4.4.8./2

Geben Sie das zu den Elementen der Übertragungsliste jeweils gehörende Format und die Satzgliederung der übertragenen Datenfelder an.

a)
```
      READ(1,1Ø) A,B,(X(I),I=1,5),F(5),TEXT
   1Ø FORMAT(2E1Ø.Ø /F5.2,4(2X,F5.2)/I6,1ØX,A2)
```

b)
```
      WRITE(3,2Ø) U,V,((MATRIX(I,J),J=1,6),I=1,1Ø)
   2Ø FORMAT(1H1,'ERGEBNISSE:'//5X,3HU=␣,E1Ø.Ø,5X,3HV=␣,E1Ø.Ø/5X,
     1'MATRIX␣=␣'/6(6X,I8))
```

4.4.8./3

a) Man gebe die zum Druck des angegebenen Bildes notwendigen FORTRAN-Anweisungen an.

*	*	*				*	*						*
*			*		*			*				*	*
*			*				*				*		*
*	*	*				*							*
*	*				*								*
*		*			*								*
*			*		*	*	*	*					*

b) Welches Druckbild entsteht bei dem folgenden Ausgabevorgang?

```
   WRITE(3,3Ø) (X(I),Y(I),I=1,N)
3Ø FORMAT(1H1,14X,8HVEKTOR␣X,15X,'VEKTOR␣Y'///2(13X,E11.4))
```

4.4.8./4

Ö

Stellen Sie ein FORTRAN-Programm für die stark vereinfachte Form einer Bruttolohnrechnung auf. Für jeden Beschäftigten werden auf einer Lochkarte Name (20 Zeichen), Gehaltsnummer (4 Ziffern), Anzahl der geleisteten Normalstunden (3 Ziffern), Anzahl der Überstunden (2 Ziffern) und Lohngruppe (1 Ziffer) eingegeben. Zu erstellen ist eine Liste mit den Informationen Name, Gehaltsnummer und Bruttolohn. Für die Überstunden soll ein Zuschlag von 25 Prozent gezahlt werden. Die Tabelle der Stundenlöhne in den Lohngruppen 1 bis 9 ist auf einer gesonderten Karte dem Stapel der Personendaten voranzulegen. Den Abschluß des gesamten Kartenstapels bildet eine Leerkarte.

Anleitung:

Die Einteilung der Lochkarten ist geeignet festzulegen. Zur Aufnahme der 20 Zeichen des Namens wird ein eindimensionales Feld der Länge 5 vereinbart. Das Ende des Programmlaufes kann durch einen Test auf Lohngruppe 0, die nur in der abschließenden Leerkarte vorkommt, festgestellt werden.

4.4.8./5

Ö

Für die zwei Meßreihen x_i, y_i $(i = 1(1)n)$ $(n \leqslant 1000)$ soll der Korrelationskoeffizient ρ nach

$$\rho = \frac{s_{xy}}{s_x s_y}$$

mit

$$s_x = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2 - \frac{1}{n^2}\left(\sum_{i=1}^{n} x_i\right)^2}$$

$$s_y = \sqrt{\frac{1}{n}\sum_{i=1}^{n} y_i^2 - \frac{1}{n^2}\left(\sum_{i=1}^{n} y_i\right)^2}$$

$$s_{xy} = \frac{1}{n}\sum_{i=1}^{n} x_i y_i - \frac{1}{n^2}\sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i$$

berechnet werden. Zur Eingabe ist n als ganze Zahl in die Spalten 1 bis 4 der ersten Lochkarte zu lochen, die Wertepaare (x_i, y_i) stehen jeweils auf einer Karte in den Spalten 1 bis 10 und 11 bis 20 (Format F10.3).

4.4.8./6

Ö

Z

Auf einem Magnetband sei eine sequentielle Personaldatei in folgender Form gespeichert:

Jeder Satz besteht aus einer Personenkennzahl (9 Ziffern) und weiteren zur Person gehörenden nicht näher spezifizierten Angaben (60 beliebige Zeichen). Der letzte Satz vor dem durch die ENDFILE-Anweisung erzeugten Dateiendekennzeichen ist durch die Kennzahl 999999999 ausgezeichnet.
Schreiben Sie ein FORTRAN-Programm, mit dem diese Personaldatei auf Grund von über Lochkarten eingegebenen Änderungsvorschriften aktualisiert werden kann. Die Lochkarten enthalten als Informationen

Spalte 1 – Kennzeichen über die Art der Korrektur
1 Der Satz mit der angegebenen Personenkennzahl soll gelöscht werden.
2 Die in der Lochkarte enthaltenen Angaben sind als neuer Satz in die Personaldatei einzufügen.

Spalten 2 bis 10 – Personenkennzahl
Spalten 21 bis 80 – 60 beliebige Zeichen als Angabe zur Person bei Kennzeichen 2

Enthält beim Kennzeichen 2 die ursprüngliche Datei bereits einen Satz mit der angegebenen Personenkennzahl, so ist dieser durch den neuen Satz aus der Lochkarte zu ersetzen, anderenfalls wird er als zusätzlicher Satz eingefügt. Den Abschluß des eingegebenen Kartenstapels bildet eine Leerkarte. Sowohl auf Band als auch im Kartenstapel seien die Datensätze nach steigenden Personenkennzahlen geordnet. Die geänderte Datei ist als neue Datei anzulegen.

4.5. Unterprogrammtechnik

4.5.1. Einführung in die Unterprogrammtechnik

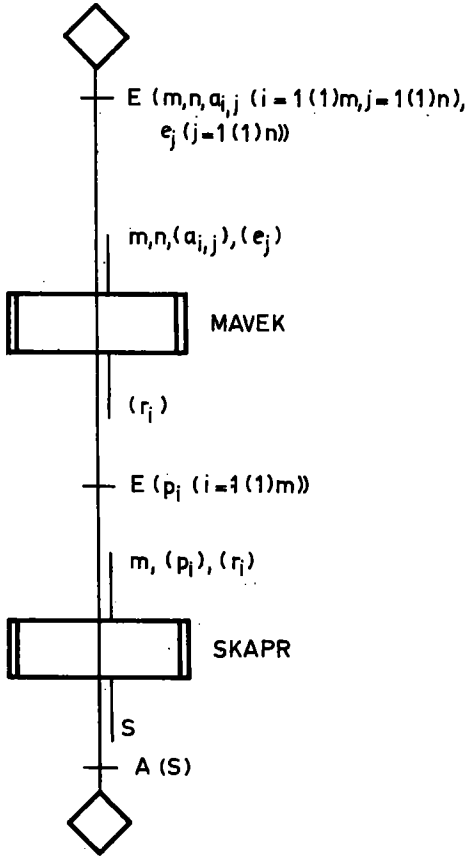
Bei der Lösung größerer Probleme tritt häufig ein Teilproblem mehrfach mit unterschiedlichen Parametern auf. Denken wir zum Beispiel an die Teilaufgabe Matrix * Vektor im Zusammenhang mit einem mehrstufigen linearen Produktionsmodell (vgl. Aufgabe 4.3.2.5./4). Wesentlich rationeller als die Niederschrift desselben zugehörigen Programmstücks an den entsprechenden Stellen ist seine einmalige Notation als Unterprogramm (UP) und dessen Aktivierung von diesen Stellen aus jeweils mit den erforderlichen Parametern. Man spart so Speicherplatz, Programmierzeit und schränkt Fehlerquellen ein, wogegen lediglich eine geringe Laufzeiterhöhung wegen Organisationsarbeit bei der Parametervermittlung steht. Das gilt gleichfalls für ein Teilproblem, von dem bekannt oder zu erwarten ist, daß es auch bei anderen Aufgaben vorkommt und erst recht, wenn man auf ein in einer Programmbibliothek vorhandenes UP zurückgreifen kann.
Schließlich ermöglicht die Unterprogrammtechnik auf einfache Weise die Arbeitsteilung in der Programmierung.
Diese Rationalisierungsaspekte sollten Sie stets im Auge haben. Im übrigen entspricht die Unterprogrammtechnik der Verwendung von Operatoren bei Programmablaufplänen (vgl. Abschnitt „Beschreibung eines Programmablaufs" dieses Lehrbuchs).
Wir wollen uns einen Überblick über die Begriffe der FORTRAN-Unterprogrammtechnik an Hand eines Demonstrationsbeispiels verschaffen. Zu diesem Zweck

greifen wir auf Beispiel 2 aus 4.3.2.4. zurück: Es sind Materialkosten S nach der Formel

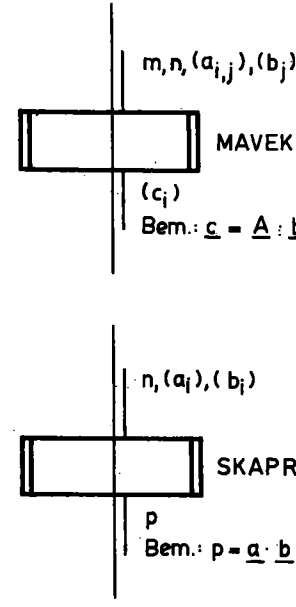
$$S = \underline{p}^T \cdot \underline{A} \cdot \underline{e}$$

zu berechnen, wobei $\underline{e} = (e_j)$ der Produktionsmengenvektor, $\underline{A} = (a_{i,j})$ die Matrix der Materialverbrauchsnormen und $\underline{p} = (p_i)$ der Vektor der Materialpreise ist ($i = 1(1)m, j = 1(1)n$).
Diese Aufgabe besteht aus den zwei Teilaufgaben Matrix * Vektor ($\underline{r} = \underline{A} \cdot \underline{e}$) und Skalarprodukt zweier Vektoren ($S = \underline{p}^T \cdot \underline{r}$). Für diese Teilaufgaben sollen Unterprogramme in der Programmbibliothek zur Verfügung stehen. Dem Programmierer obliegt dann nur die Niederschrift des sogenannten Haupt- oder Rahmenprogramms, welches bei unserem Beispiel lediglich die Dateneingabe, die Aktivierungen der Unterprogramme sowie die Datenausgabe zu besorgen hat.

Programmablaufplan



Verkürzte Form der Operatordefinitionen:



Wir geben die Darstellung des vollständigen Programms (Hauptprogramm mit Unterprogrammen), auch zusammengesetztes Programm genannt, als PAP (s. S. 77) und in FORTRAN an. Nehmen Sie das Programm zu Beispiel 2 in 4.3.2.4. als Vergleich.

FORTRAN-Programm

```
C HAUPTPROGRAMM MATERIALKOSTEN
      DIMENSION A(5Ø,5Ø), E(5Ø), P(5Ø), R(5Ø)
      EQUIVALENCE (A(1),R(1)), (E(1),P(1))
      READ (1,2Ø) M, N, ((A(I,J), J=1,N), I=1,M), (E(J), J=1,N)
   2Ø FORMAT (2I2 / (F8.2))
      CALL MAVEK(M,N,A,E,R)
      READ (1,3Ø) (P(I), I=1,M)
   3Ø FORMAT (F8.2)
      S = SKAPR(M,P,R)
      WRITE (3,4Ø) S
   4Ø FORMAT ('ØMATERIALKOSTEN␣:␣', F12.2, '␣MARK')
      STOP
      END
C UNTERPROGRAMM MATRIX*VEKTOR
      SUBROUTINE MAVEK(M,N,A,B,C)
        .
        .
        .
      RETURN
      END
C UNTERPROGRAMM SKALARPRODUKT
      FUNCTION SKAPR(N,A,B)
        .
        .
        .
      RETURN
      END
```

Einzelheiten der UP interessieren hier nicht. Diese UP werden in 4.5.3.2. und 4.5.6.2. behandelt.
Wir wollen nun den gegenüber der Darstellung in 4.3.2.4. veränderten Programmaufbau unter Herausarbeitung allgemeiner Gesichtspunkte diskutieren.
Dem Hauptprogramm (HP) folgen die Unterprogramme, deren Reihenfolge beliebig wählbar ist. Man unterscheidet

Funktions(FUNCTION)-Unterprogramme und
Subroutine(SUBROUTINE)-Unterprogramme.

Diese UP werden auch als externe UP bezeichnet, im Unterschied zu den

Anweisungsfunktionen,

die innerhalb einer Programmeinheit (HP oder externes UP) definiert werden und nur eine spezielle Möglichkeit der Berechnung eines Funktionswertes (Auswertung einer Formel) bieten. Wir wollen unter dem Wort UP stets nur ein Funktions- oder Subroutine-UP verstehen.
Ein Funktions-UP dient der Berechnung eines (einzigen) Wertes, des Funktions-

wertes, aus einem oder mehreren Argumenten. Im Beispiel wurde das Skalarprodukt als Funktions-UP geschrieben, da sein Resultat in einem einzigen Wert besteht. Der Aufruf eines Funktions-UP entspricht völlig dem einer Standardfunktion. Er erfolgt bei der Berechnung eines Ausdrucks, in dem als ein Operand der Funktionsname gefolgt von der (aktuellen) Parameterliste steht. Bei unserem Beispiel besteht dieser Ausdruck nur aus dem Funktionsaufruf SKAPR(M,P,R). Die Vektoren P, R sowie ihre Dimension M fungieren als Eingangsparameter. Der Resultatwert wird unter dem Funktionsnamen vom UP nach Abarbeitung an das HP übermittelt.
Ein Subroutine-UP unterscheidet sich von einem Funktions-UP nur dadurch, daß sein Ergebnis nicht aus einem einzigen Wert bestehen muß und der Aufruf durch eine spezielle Anweisung, die Rufanweisung (CALL-Anweisung), erfolgt. An CALL schließen sich dabei der Subroutinename und die (aktuelle) Parameterliste an. Da die Teilaufgabe Matrix * Vektor unseres Beispiels einen Vektor (Wertesatz) als Resultat hat, mußte sie als Subroutine-UP geschrieben werden. In dem Aufruf CALL MAVEK(M,N,A,E,R) fungieren M, N, A, E als Eingangsparameter und R als Resultatparameter.
Beim Aufruf eines UP erfolgt zunächst die Parametervermittlung und anschließend seine Abarbeitung. Die Programmsteuerung ist dem UP während seiner Laufzeit übertragen. Nach Abarbeitung des UP erfolgt die Rückkehr in die aufrufende Programmeinheit (PE) unmittelbar „hinter" die Aufrufstelle. Die Resultate des UP stehen anschließend unter den gewählten Namen (Resultatparameter, Funktionsname) in der aufrufenden PE zur Verfügung. In unserem Beispiel folgt also auf die Abarbeitung des Subroutine-UP MAVEK die Abarbeitung der Anweisung READ (1,3Ø) (P(I), I = 1,M), während nach Abarbeitung des Funktions-UP SKAPR der Funktionswert der Variablen S zugewiesen wird.
Die Parametervermittlung geschieht so, daß jeder aktuelle Parameter (AP) des UP-Aufrufs die Rolle des an gleicher Listenposition stehenden formalen Parameters (FP) der UP-Definition übernimmt. Ein solcher Mechanismus der Parametervermittlung ist im Interesse der Flexibilität eines UP – dieses darf nicht an feste Namen seiner Operanden gebunden sein – erforderlich. Die FP haben damit eine reine Platzhalterfunktion, ihre Namen stehen in keinem Bezug zu denen der AP. Eine weitere Form der Parametervermittlung (über den sogenannten COMMON-Bereich) werden wir in 4.5.5. kennenlernen.
Schließlich sei erwähnt, daß durch geeignete Festlegungen über die Parameterspezifizierung erreicht wird, daß sich ein UP unabhängig von anderen PE übersetzen läßt.

4.5.2. Funktionen

Aufgabe eines Funktions-UP wie auch einer Anweisungsfunktion ist die Berechnung eines (einzigen) Funktionswertes aus einem oder mehreren Argumenten (Parametern). Der Aufruf erfolgt beim Auswerten eines arithmetischen Ausdrucks durch das Auftreten des Funktionsaufrufes als Operand. Bei einem Funktions-UP können als Nebeneffekt weitere Resultatwerte (wertemäßig veränderte Eingangsparameter) auftreten.

4.5.2.1. Funktions-Unterprogramme (FUNCTION-UP)

4.5.2.1.1. Definition

Ein Funktions-UP wird eingeleitet durch eine *Funktionsanweisung*

$$\left.\begin{array}{l}\text{FUNCTION } f(p_1,p_2,\ldots,p_n)\\ \text{oder}\\ \text{t FUNCTION } f(p_1,p_2,\ldots,p_n)\end{array}\right\} (n \geqslant 1)$$

Bei der ersten Form ist der Typ des Funktionsnamens bzw. -wertes f implizit definiert, bei der zweiten Form explizit gemäß t, welches für INTEGER, REAL oder DOUBLE PRECISION steht.
FUNCTION ist ein FORTRAN-Schlüsselwort, f der *Funktionsname*, die *formalen Parameter* p_i (Anzahl $\geqslant 1$!) sind Namen. Ein FP wird in der UP-Definition entweder als

a) eine einfache Variable oder als
b) ein Feldname oder als
c) Name eines Funktions-UP oder als
d) Name eines Subroutine-UP

verwendet.
Der Funktionsanweisung folgt der UP-Körper, der prinzipiell den gleichen Aufbau aufweist wie die bisher besprochenen Programme, die aus der jetzigen Sicht Hauptprogramme ohne Unterprogramme sind.
Am Anfang eines UP-Körpers können Vereinbarungsanweisungen (s. Anhang A 4/2) stehen, die auch FP enthalten können. Dann können Anweisungsfunktionsdefinitionen folgen und schließlich folgt der operative Programmteil der ausführbaren Anweisungen. Den Schluß bildet die END-Zeile.
Hinzu kommen einige spezifische Festlegungen:
Ein FP, außer einem solchen, der für den Namen eines Subroutine-UP steht, hat einen Typ, der entweder implizit oder durch eine Typanweisung innerhalb des Funktions-UP festzulegen ist. Ein FP, der für einen Feldnamen steht, muß in einer Feldvereinbarung mit festen Grenzen spezifiziert werden.
Aus dem Aufbau eines zusammengesetzten Programms (s. 4.5.1.) geht schon hervor, daß innerhalb eines Funktions-UP als selbständiger Programmeinheit kein anderes UP vereinbart werden kann. Ebenso kann auf Größen anderer Programmeinheiten nur über Parameter beim Aufruf oder mit Hilfe der Globalanweisung (s. 4.5.5.) bei einem gemeinsamen Speicherbereich für alle Programmeinheiten Bezug genommen werden.
FP dürfen nicht in einer Äquivalenz- oder Globalanweisung (s. 4.5.5.) des UP stehen.
Der Name des Funktions-UP muß im operativen Teil als Name einer einfachen Variablen erscheinen. Ihm muß bei jeder möglichen Abarbeitung des UP mindestens einmal ein Wert zugewiesen werden. Das erfolgt im allgemeinen durch eine Ergibtanweisung; andere Möglichkeiten sind das Auftreten des Funktionsnamens in der Übertragungsliste einer READ-Anweisung oder als Resultatparameter in einer aktuellen Parameterliste eines erneuten UP-Aufrufs.

Die Abarbeitung eines Funktions-UP beginnt mit der ersten ausführbaren Anweisung. Mit Hilfe der Rücksprunganweisung

RETURN

wird die Abarbeitung des UP beendet und zur aufrufenden Programmeinheit in die Auswertung des den Funktionsaufruf enthaltenden Ausdrucks zurückgekehrt. Dabei wird der Funktionswert übermittelt.
Auf folgende Weise können als „Nebeneffekt" über den ausgezeichneten Funktionswert hinaus weitere Resultatwerte erzeugt werden: Eine Variable, die AP ist, und ein Feld, dessen Name AP ist, können im UP wertemäßig verändert werden (Namensaufruf, s. 4.5.2.1.2.) und haben nach der Rückkehr aus dem UP die Werte, die sie im UP zuletzt hatten. Ein Beispiel folgt in 4.5.2.1.2.
Ein Funktions-UP kann mehrere Rücksprunganweisungen enthalten. Der dem Funktionsnamen jeweils zuletzt zugewiesene Wert ist der Funktionswert.
Ein Funktions-UP darf andere UP aufrufen, aber weder direkt noch indirekt sich selbst. Rekursivität ist also nicht gestattet.

Beispiele

Funktionsanweisung	Typ des Funktions-UP	Typ der FP
`FUNCTION COSH(X)`	reell	reell
`REAL FUNCTION FKT(N,A,B)`	reell	N ganzzahlig A,B reell
`DOUBLE PRECISION FUNCTION WERT (S,T)` `DOUBLE PRECISION T`	doppeltgenau	S reell T doppeltgenau

Beispiel für ein Funktions-UP

Das UP soll die Berechnung des arithmetischen Mittels

$$\frac{1}{n} \cdot \sum_{i=1}^{n} a_i \qquad (n \leqslant 50)$$

durchführen. Als Parameter fungieren n und der Feldname a. Wir geben der Funktion den Namen MITTEL.

```
  REAL FUNCTION MITTEL(N,A)        } Funktionsanweisung
  DIMENSION A(5Ø)                  |
  S = Ø.Ø                          |
  DO 1 I = 1,N                     |
1 S = S + A(I)                     } UP-Körper
  MITTEL = S/N                     |
  RETURN                           |
  END                              |
```

4.5.2.1.2. Aufruf

Der Aufruf (innerhalb eines Ausdrucks!) hat die Gestalt

$f(a_1, a_2, \ldots, a_n)$

wobei die a_i jetzt *aktuelle Parameter* darstellen. Sie treten an die Stelle der FP und müssen mit diesen in Typ, Anzahl und Reihenfolge übereinstimmen. Die folgenden Arten von AP sind zulässig und können an die Stelle der angegebenen Arten von FP treten:

AP	FP
Ausdrücke: Konstante einfache Variable indizierte Variable „echter" Ausdruck	einfache Variable
Feldname	Feldname
Funktionsname Name einer OUT-OF-LINE-Standardfunktion	Funktionsname
Subroutinename Name einer Standard-Service-Prozedur	Subroutinename

Diese jeweils für die Dauer der Abarbeitung eines UP gültigen Zuordnungen treten beim UP-Aufruf in Kraft und werden nachfolgend im einzelnen erläutert. Vorweg noch zwei Bemerkungen:

a) Weicht der Typ eines Funktions-UP vom implizit definierten Typ ab, muß in jeder PE, die dieses Funktions-UP aufruft, der Funktionsname in einer entsprechenden Typanweisung stehen.
b) Bei Subroutine-UP, für die die folgenden Erläuterungen auch voll zutreffen, werden Ergebnisse mit Hilfe von Resultatparametern übertragen.

1. *AP Konstante, FP einfache Variable.* Der FP wird innerhalb des UP durch die Konstante ersetzt. Es handelt sich um die einfachste Form des Wertaufrufs. Damit ist klar, daß der Parameter nur Eingangsparameter sein kann. Er kann zum Beispiel nicht auf der linken Seite einer Ergibtanweisung stehen.

2. *AP einfache Variable, FP einfache Variable.* Hier wird innerhalb des UP der Name des FP durch den des AP ersetzt (Namensaufruf). Damit ist eine Verbindung zum aufrufenden Programm aufgebaut, über die der Wert des AP empfangen (Eingangsparameter) oder sein im UP berechneter Wert übermittelt (Resultatparameter) oder der empfangene Wert verändert und übermittelt (Eingangs- und Resultatparameter) werden kann.

Beispiel

```
FUNCTION G(X,Y,Z)
A = X*(Y+2)
Z = A
Y = EXP(A)
G = A**2
RETURN
END
```

In diesem Beispiel sind X und Y Eingangsparameter, Y und Z Resultatparameter. Bei einem Aufruf

```
W=B-G(A,B,C)/2.5
```

müssen demnach A und B bereits einen Wert besitzen. Da die Werte von B und C im UP verändert werden (Nebeneffekt!), darf dieses UP nur in solchen Ausdrükken zur Anwendung kommen, in denen B und C nicht nochmals benutzt werden. Anderenfalls würde die Berechnung eines derartigen Ausdruckes unterschiedliche Resultate liefern, je nachdem, in welcher Reihenfolge der Ausdruck abgearbeitet wird.

3. *AP indizierte Variable, FP einfache Variable.* Die Verhältnisse sind wie bei 2. Hinzu kommt die einmalige Berechnung der Indexausdrücke der indizierten Variablen zu Beginn des Aufrufs. Das entsprechende Feld muß im aufrufenden Programm definiert sein.

4. *AP „echter" Ausdruck, FP einfache Variable.* "Echter" Ausdruck soll die Fälle 1–3 ausschließen. Zunächst wird der Wert des Ausdrucks berechnet und im UP für die Variable eingetragen (Wertaufruf). Der Parameter fungiert in diesem Fall nur als Eingangsparameter. Ist ein FP Resultatparameter, so ist also beim UP-Aufruf die Zuordnung eines „echten" Ausdrucks oder einer Konstanten nicht möglich.

5. *AP Feldname, FP Feldname.* Hier ist zunächst erforderlich, daß beide Felder in Dimension und Größe übereinstimmen mit Ausnahme des Laufbereiches des letzten Index', der im aufrufenden Programm größer sein darf. Außerdem ist gestattet, daß ein mehrdimensionales Feld als AP im UP unter Beachtung der Speicherung von Feldern als eindimensionales Feld behandelt wird. Der Parameteraufruf erfolgt als Namensaufruf. Die ersten Elemente beider Felder werden einander zugeordnet, die restliche Zuordnung ergibt sich aus der Speicherreihenfolge. Der Parameter kann Eingangs- oder Resultatparameter oder beides sein.

Beispiele

a) Aktuelles Feld `DIMENSION A(2,4)`

Formales Feld `DIMENSION S(2,3)`

Feldelemente mit gleichen Indexpaaren sind einander zugeordnet, die letzte Spalte des Feldes A ist im UP nicht erreichbar.

b) Aktuelles Feld : `DIMENSION A(2,4)`

Formales Feld : `DIMENSION V(8)`

jetzt entsprechen einander A(1,1) und V(1), A(2,1) und V(2), ..., A(2,4) und V(8).
Die Vereinbarung DIMENSION V(9) im letzten Beispiel wäre fehlerhaft, da dann das formale Feld größer als das aktuelle wäre.

6. *AP Funktionsname oder Name einer OUT-OF-LINE-Standardfunktion (s. 4.5.4.), FP Funktionsname.* Der formale Funktionsname wird im UP durch den aktuellen Funktionsnamen ersetzt (Namensaufruf). Außerdem muß der Name des AP im aufrufenden Programm in einer *Externanweisung* aufgeführt werden. Diese hat die Form

EXTERNAL $n_1, n_2, \ldots, n_k$

EXTERNAL ist ein Schlüsselwort, die n_i sind Namen von Unterprogrammen, die hierdurch als extern gekennzeichnet werden. Die Externanweisung ist nicht ausführbar. Ist ein Ausdruck AP und enthält einen Funktionsaufruf, so muß der entsprechende Name nicht in einer Externanweisung stehen.

Beispiel

```
C AUFRUFENDES PROGRAMM
      .
      .
      .
      EXTERNAL FUN, COS
      .
      .
      .
      V = BL(A,EXP(A),FUN) + BL(A,B,COS)
C EXP(A) IST AUSDRUCK, FUN NAME EINES FUNKTIONS-UP
      .
      .
      .
      END
C
      FUNCTION BL(X1,X2,FKTNM)
C FKTNM SOLL FUER EINEN FUNKTIONSNAMEN STEHEN
      .
      .
      .
      BL = FKTNM(X1+X2)
      .
      .
      .
      END
C
      FUNCTION FUN(X)
      .
      .
      .
      FUN = X**2 - 3.5*X
      .
      .
      .
      END
```

7. *AP Subroutinename oder Name einer Standard-Service-Prozedur (s. 4.5.4.), FP Subroutinename.* Es gilt sinngemäß dasselbe wie bei 6.

4.5.2.1.3. Beispiele

Beispiel 1

Funktions-UP zur Berechnung der Funktion

$$y = \begin{cases} 4 \arctan x^2 + 4 - \pi & \text{für } 1 < x \\ 4x & \text{für } 0 \leqslant x \leqslant 1 \\ -y(-x) & \text{für } x < 0 \end{cases}$$

```
      FUNCTION Y(X)
      IF (ABS(X) - 1) 1,1,2
    1 Y = 4*X
      RETURN
    2 Y = 4*ATAN(X*X) + 4. - 3.1416
      IF (X) 3,4,4
    3 Y = -Y
    4 RETURN
      END
```

Wir haben im Programm ausgenutzt, daß der Ausdruck 4x ungerade (also $y(-x) = -y(x)$ bzw. $y(x) = -y(-x)$) und der Ausdruck $4\arctan x^2 + 4 - \pi$ gerade ($y(-x) = y(x)$) ist. Die Verwendung zweier Rücksprunganweisungen ist hier sinnvoll.
Ein Funktionsaufruf könnte zum Beispiel lauten:

```
A = B * Y(Z + X - 3) + 3.5
```

Beispiel 2

Die Berechnung von $n! = 1 \cdot 2 \cdot 3 \ldots \cdot n$ für $n \geqslant 1$ leistet folgendes Funktions-UP:

```
      INTEGER FUNCTION FAK(N)
      FAK = 1
      DO 1 I = 1,N
    1 FAK = FAK*I
      RETURN
      END
```

Gleichwertig wäre FUNCTION NFAK(N) als Funktionsanweisung und Ersetzen von FAK durch NFAK im UP-Körper.
Im Basis-FORTRAN vom DOS/ES kann dieses Funktions-UP für $n \leqslant 12$ verwendet werden. Für größere n ist eine reelle Funktion erforderlich.

Beispiel 3

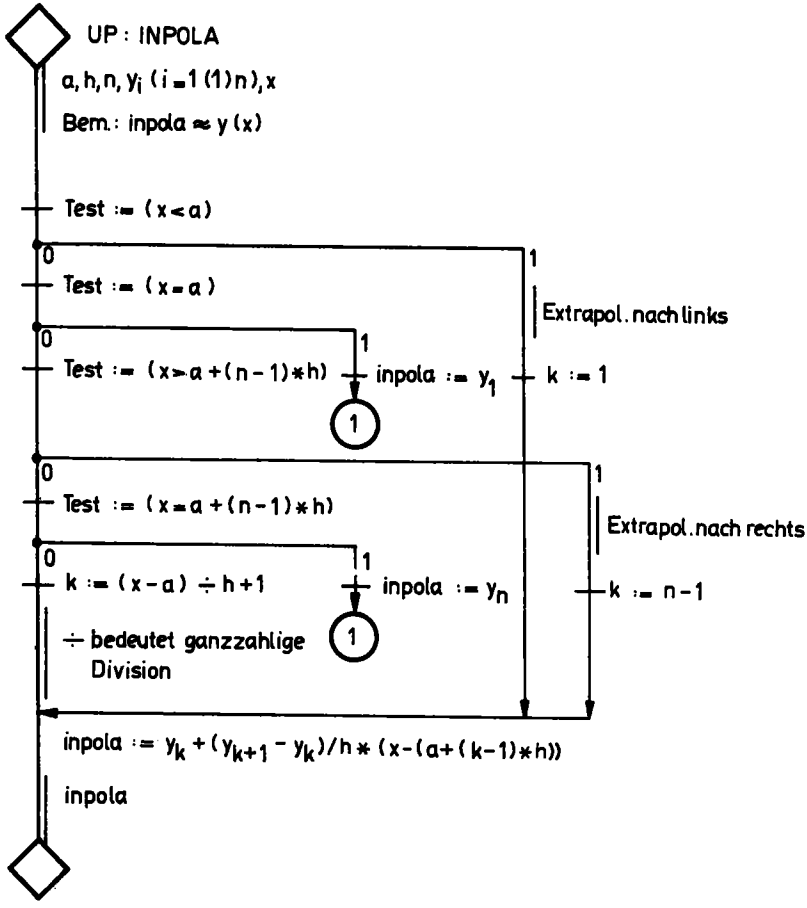
Eine Funktion y(x) sei tabellarisch gegeben durch n äquidistante Argumente $x_i = a + (i-1)\,h$ $(i = 1(1)n, h > 0)$ mit den zugehörigen Funktionswerten $y_i = y(x_i)$. Wir wollen ein Funktions-UP aufstellen, das zu einem vorgegebenen Argument x den Funktionswert y(x) angenähert durch lineare Interpolation bzw. Extrapolation berechnet, je nachdem, ob x innerhalb oder außerhalb des Intervalls $[x_1, x_n]$

liegt. Bekanntlich lautet der Interpolationswert, wenn x im Intervall $[x_k, x_{k+1}]$ liegt:

$$y = y_k + \frac{y_{k+1} - y_k}{h} (x - x_k)$$

Zur Extrapolation wird dieselbe Formel mit k = 1 (falls $x < x_1$) bzw. mit k = n − 1 (falls $x > x_n$) verwendet.

Programmablaufplan



FORTRAN-UP

```
      REAL FUNCTION INPOLA(A,H,N,Y,X)
      DIMENSION Y(1ØØ)
      IF (X - A) 3,2,4
    2 INPOLA = Y(1)
      RETURN
    3 K = 1
      GOTO 8
    4 IF (X- (A + (N - 1)*H)) 7,5,6
    5 INPOLA = Y(N)
      RETURN
    6 K = N - 1
      GOTO 8
    7 K = (X - A)/H + 1
    8 INPOLA = Y(K)+(Y(K+1)-Y(K))/H*(X-(A+(K-1)*H))
      RETURN
      END
```

Beispiel für einen Aufruf

```
C AUFRUFENDES PROGRAMM
      •
      •
      REAL INPOLA
      DIMENSION V(1ØØ)
      •
      •
      •
      Z = INPOLA(D,S,24,V,T)
      •
      •
      •
```

Wichtig ist, daß das eindimensionale Feld V im aufrufenden Programm von mindestens derselben Größe wie das entsprechende Feld Y im UP vereinbart wird. N muß ≤ 100 sein.

4.5.2.2. Anweisungsfunktionen

Z

Eine Anweisungsfunktion dient der Berechnung eines Funktionswertes durch Auswertung eines arithmetischen Ausdrucks für gewisse Parameter. Außerdem erfolgt ihre Definition innerhalb einer PE und ist auch nur in dieser PE gültig. Insofern ist die Anweisungsfunktion wesentlich spezieller als das Funktions-UP. Die Definition einer Anweisungsfunktion hat die Gestalt

$f(p_1, p_2, \ldots, p_n) = a$

Dabei sind f, $p_1, \ldots, p_n$ ($n \geq 1$) untereinander verschiedene Namen, f heißt *Anweisungsfunktionsname*, die p_i heißen formale Parameter. a ist ein arithmetischer Ausdruck, der nur formale Parameter, Konstante, einfache Variable und Aufrufe von Funktions-UP und Anweisungsfunktionen enthalten darf. Letzere müssen vorher in der gleichen Programmeinheit definiert worden sein. Also sind indizierte Variable und der Aufruf derselben Anweisungsfunktion ausgeschlossen. f darf in der Programmeinheit nicht Name einer anderen Größe sein. Die Definition muß

hinter den Vereinbarungsanweisungen und vor der ersten ausführbaren Anweisung der PE stehen (vgl. Anhang (A 4/2)).
Anweisungsfunktionsname und formale Parameter haben einen Typ, der entweder implizit oder durch eine Typanweisung explizit definiert ist. Der Typ des Wertes der Anweisungsfunktion ist gleich dem Typ des Anweisungsfunktionsnamens.
Da FP nur Platzhalter sind, die dazu dienen, Typ, Anzahl und Reihenfolge der AP festzulegen, können ihre Namen auch für einfache und indizierte Variable oder FP anderer Anweisungsfunktionen in derselben PE verwendet werden. Allerdings muß in diesem Fall Typgleichheit bestehen.

Beispiele für Anweisungsfunktionsdefinitionen

Definition	Typ der Anweisungsfunktion	Typ der FP
`COSH(X) = (EXP(X)+EXP(-X))*Ø.5`	reell	reell
`VHOHLK(RI,RA) = 4./3.*3.1416*(RA**3-RI**3)`	reell	reell
`WERT(A,B,C,X) = A*COSH(B*X)+1.5*C`	reell	reell
`FUN(U,V,N) = 2.4*U+V/N`	reell	U,V reell N ganzz.

Die Typangaben gelten bei impliziter Typvereinbarung.
Beachten Sie, daß WERT nur von COSH Gebrauch machen darf, wenn COSH in der gleichen Programmeinheit vor WERT definiert wurde.
Fehlerhaft wäre die Definition

```
SUB(4.Ø,A(1),B) = 4.Ø*X(3)/B**3
```

wegen der Konstanten und indizierten Variablen als formale Parameter und der indizierten Variablen im Ausdruck.
Beim Aufruf einer Anweisungsfunktion folgt dem Anweisungsfunktionsnamen in Klammern die Liste der AP, die mit den FP in Typ, Anzahl und Reihenfolge übereinstimmen müssen. Als aktueller Parameter kann ein Ausdruck (also auch die einfachen Formen Konstante, Variable, Funktionsaufruf) vom Typ des zugehörigen FP verwendet werden. Das Gleiche gilt für einen Feldnamen, der allerdings nur zur Vermittlung an ein in der Anweisungsfunktionsdefinition benutztes Funktions-UP in Frage kommt.

Beispiele für den Aufruf (Definitionen siehe oben)

a) `RMASSE = DICHTE * VHOHLK(DI/2,DA/2)`
ist äquivalent der Ergibtanweisung
`RMASSE = DICHTE*4./3.*3.1416*((DA/2)**3-(DI/2)**3)`

b) `PROD = FUN(A,B+T,1Ø)/WERT(FUN(3.2,V,M),U,S,2.Ø)`
Der Aufruf der Anweisungsfunktion WERT beginnt mit der Berechnung von FUN (3.2, V,M), was wiederum den Aufruf einer Anweisungsfunktion bedeutet.

4.5.3. Subroutine-Unterprogramme (SUBROUTINE-UP)

Ein Subroutine-UP hat keinen von vornherein ausgezeichneten Resultatwert wie das Funktions-UP. Es erzeugt im allgemeinen eine Reihe von Resultatwerten. Beispiele sind die Lösung eines linearen Gleichungssystems oder die Sortierung von Werten nach wachsender Größe. Der Aufruf eines Subroutine-UP erfolgt durch eine Rufanweisung.

4.5.3.1. Ergänzende Definitionen

Die Unterschiede zum Funktions-UP sind gering. Das Subroutine-UP beginnt mit der *Subroutineanweisung*

SUBROUTINE $s(p_1, p_2, \ldots, p_n)$ oder

SUBROUTINE s

SUBROUTINE ist ein Schlüsselwort, s der *Subroutinename*, die formalen Parameter p_i sind Namen. Die zweite Form ohne explizite Parameter kann durch COMMON-Größen (s. 4.5.5.) versorgt oder ohne jeden Parameter zum Beispiel zur Ausgabe eines festen Textes oder zum Auslösen von Steuerfunktionen verwendet werden. s hat keinen Typ.

Der Subroutineanweisung folgt der UP-Körper, wie beim Funktions-UP in 4.5.2.1.1. beschrieben. Gleichfalls gelten die dort angeführten Festlegungen sinngemäß auch für Subroutine-UP mit folgenden vier Modifikationen:

1. Der Name des Subroutine-UP kommt in keiner weiteren Anweisung des UP vor.
2. Der Rücksprung nach dem Erreichen einer Rücksprunganweisung erfolgt zur ersten ausführbaren Anweisung, die der Rufanweisung im aufrufenden Programm folgt.
3. Die Vermittlung der Resultate erfolgt im Gegensatz zum Funktions-UP über Resultatparameter (die zugleich Eingangsparameter sein können) bzw. über den COMMON-Bereich (s. 4.5.5.).
4. Ein Subroutine-UP muß keine Rücksprunganweisung enthalten bzw. eine solche muß bei der Abarbeitung nicht erreicht werden. In diesem Fall wird nach Abarbeitung des Subroutine-UP die Abarbeitung des zusammengesetzten Programms beendet.

Die *Rufanweisung* für ein Subroutine-UP hat die Form

CALL $s(a_1, a_2, \ldots, a_n)$

oder

CALL s

CALL ist ein Schlüsselwort; die a_i sind aktuelle Parameter.
Hinsichtlich der Art der AP und der möglichen Zuordnung zu den FP gilt wörtlich dasselbe wie in 4.5.2.1.2.

4.5.3.2. Beispiele

Beispiel 1

Da Basis-FORTRAN keine komplexen Variablen gestattet, müssen bei Verknüpfungen komplexer Zahlen die Berechnungen von Real- und Imaginärteil des Resultats, die auch separate Speicherplätze haben, getrennt als reelle Rechnungen durchgeführt werden.
Wir wollen ein Subroutine-UP zur Berechnung des Produkts

$(a + ib) \cdot (c + id) = x + iy = ac - bd + i\,(ad + bc)$

zweier komplexer Zahlen schreiben. Offensichtlich werden vier Eingangs- und zwei Resultatparameter benötigt:

```
SUBROUTINE KMULT(A,B,C,D,X,Y)
X = A*C - B*D
Y = A*D + B*C
RETURN
END
```

Soll damit zum Beispiel $\frac{(a + ib) \cdot (c + id)}{(p + iq) \cdot (r + is)} = x + iy$ berechnet werden, so könnte man programmieren:

```
•
•
•
CALL KMULT(A,B,C,D,X1,Y1)
CALL KMULT(P,Q,R,S,X2,Y2)
CALL KMULT(X1,Y1,X2,-Y2,X3,Y3)
CALL KMULT(X2,Y2,X2,-Y2,X4,Y4)
X = X3/X4
Y = Y3/X4
•
•
•
```

Die beiden letzten Aufrufe bewirken die Erweiterung des Bruches mit dem konjugiert komplexen Wert X2–iY2 des Nenners. Y4 muß sich zu Null ergeben. Falsch wäre beispielsweise ein Aufruf CALL KMULT(A,B,C,D,X1+Y1,1.4), da für einen Resultatparameter beim Aufruf weder ein „echter" Ausdruck noch eine Konstante stehen darf.

Beispiel 2

Die Berechnung des Produktes $\underline{c} = \underline{A} \cdot \underline{b}$ einer Matrix $\underline{A}$ vom Typ (m,n) mit einem Vektor $\underline{b}$ mit n Komponenten, die bereits in 4.3.2.3.2. programmiert wurde, soll jetzt als Subroutine-UP dargestellt werden. Für die c_i gilt bekanntlich die Formel

$$c_i = \sum_{j=1}^{n} a_{i,j} \cdot b_j \qquad (i = 1(1)m)$$

Subroutine-UP

```
      SUBROUTINE MAVEK(M,N,A,B,C)
      DIMENSION A(5Ø,5Ø), B(5Ø), C(5Ø)
C 5Ø IST OBERE GRENZE FUER M UND N
      DO 1 I = 1,M
      S = Ø.Ø
         DO 2 J = 1,N
    2    S = S + A(I,J)*B(J)
    1 C(I) = S
      RETURN
      END
```

Beispiel 3

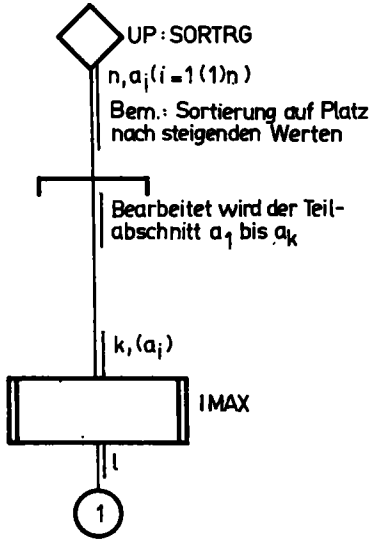
Mit Hilfe eines Subroutine-UP sollen n Zahlen $a_1, \ldots, a_n$ nach steigenden Werten sortiert werden. Ein möglicher Algorithmus ist der folgende:

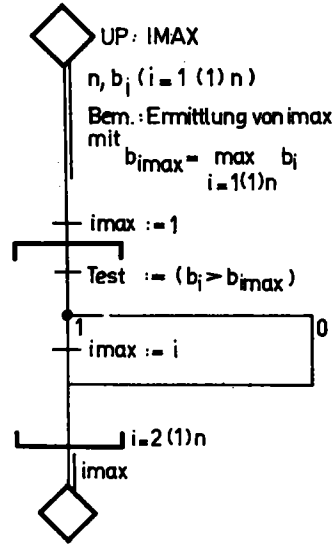
Erster Schritt: Es wird das größte Element einschließlich seines Index' ermittelt und anschließend mit dem letzten Element vertauscht.

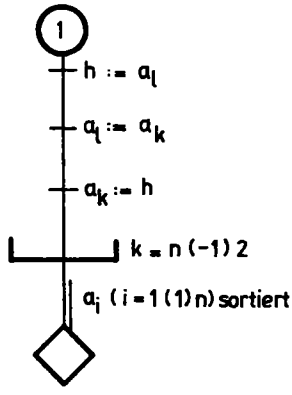
Zweiter Schritt: Wiederholung dieses Vorgehens auf dem Abschnitt $a_1, \ldots, a_{n-1}$. Das wird solange fortgesetzt, bis der Abschnitt a_1, a_2 bearbeitet ist.

Wir wollen die Teilaufgabe der Ermittlung des Index' IMAX des maximalen Elements selbst als UP schreiben, und zwar als Funktions-UP, und dieses vom Subroutine-UP zur Sortierung aus jeweils aktivieren.

Programmablaufplan







FORTRAN-Unterprogramme

```
      SUBROUTINE SORTRG(N,A)                FUNCTION IMAX(N,B)
      DIMENSION A(1ØØ)                      DIMENSION B(1ØØ)
      DO 1 I = 2,N                          IMAX = 1
      K = N + 2 - I                         DO 1 I = 2,N
      L = IMAX(A,K)                         IF (B(I)- B(IMAX)) 1,1,2
      H = A(L)                            2 IMAX = I
      A(L) = A(K)                         1 CONTINUE
    1 A(K) = H                              RETURN
      RETURN                                END
      END
```

4.5.4. Standard-Unterprogramme

Mit *Standardfunktionen* haben Sie bereits in 4.2.2. Bekanntschaft gemacht. Diese zeichnet aus, daß die zugehörigen Unterprogramme Bestandteil der Sprache sind und demzufolge die UP-Definition durch den Programmierer nicht erforderlich ist. Das gilt auch für Standardsubroutine-UP.
Es gibt zwei Arten von Standard-UP hinsichtlich ihrer Einbindung in das zusammengesetzte Programm: *OUT-OF-LINE* und *IN-LINE*. OUT-OF-LINE bedeutet, daß das entsprechende UP Bestandteil des zusammengesetzten Programms und von den Aufrufstellen aus aktiviert wird. IN-LINE hingegen bedeutet, daß an allen Aufrufstellen das UP in das Objektprogramm eingefügt wird und damit eine besondere Aktivierung entfällt. IN-LINE-Standard-UP sind sehr kurz, so daß die stärkere Speicherinanspruchnahme nicht weiter ins Gewicht fällt. Die bekannten mathematischen Standardfunktions-UP (außer Absolutbetrag) sind von der Art OUT-OF-LINE. Konvertierungsfunktionen und andere sind von der Art IN-LINE. Die Standardsubroutine-UP heißen auch *Standard-Service-UP*. Sie stellen Hilfsroutinen zur Feststellung von Fehlern und zum Einfahren von Programmen dar. Eine vollständige Übersicht der Standard-UP wird im Anhang A 4/4 und A 4/5

gegeben. Weitere Einzelheiten sind der entsprechenden ESER-Anwenderdokumentation zu entnehmen.

4.5.5. Globaler Speicherbereich (COMMON-Bereich)

Während mit Hilfe der Äquivalenzanweisung verschiedenen Variablen innerhalb einer PE derselbe Speicherplatz zugewiesen werden kann, erreicht man dies für Variable verschiedener PE mit Hilfe von Globalanweisungen in diesen PE. Grundlage ist ein gemeinsamer Speicherbereich (COMMON-Bereich). Dieser befindet sich im Hauptspeicher und hat eine vom jeweiligen zusammengesetzten Programm abhängige Länge. Jede Globalanweisung einer PE (von mehreren Globalanweisungen in einer PE sei zunächst abgesehen) bezieht sich auf diesen Speicherbereich von seinem Anfang an und legt in fortlaufender Reihenfolge die Namen fest, unter denen die einzelnen Positionen des Speicherbereichs in dieser PE angesprochen (gelesen oder beschrieben oder beides) werden sollen.

Der gemeinsame (globale) Speicherbereich bringt zwei Vorteile:

1. Er ermöglicht kurze Zugriffszeiten, was besonders bei der Parameterübergabe an ein UP vorteilhaft ist, da bei der Übergabe von Parametern über die Parameterliste zeitaufwendige Adressensubstitutionen erforderlich sind.
2. Er ermöglicht die Einsparung von Speicherplatz (Mehrfachnutzung) für mehrere PE. Wenn man zum Beispiel die lokalen Größen (Träger von Zwischenwerten) jedes UP ab derselben festen Position des Speicherbereiches speichert, so ist für den Gesamtspeicherbedarf für lokale Größen nur das UP mit den meisten lokalen Größen bestimmend. Von verschachtelten UP-Aufrufen wurde dabei abgesehen.

Allerdings eignen sich häufig nicht alle Parameter für diese Form der Übergabe, da sie starr hinsichtlich der Speicheradressen ist (vgl. 4.5.6.2. Programmversion 2). Häufig wird daher ein Teil der Parameter über die Parameterliste und der andere Teil über den gemeinsamen Speicherbereich vermittelt.

Eine *Globalanweisung* (COMMON-Anweisung) hat die Form

COMMON $v_1, v_2, \ldots, v_n$

und ist eine nichtausführbare Anweisung. COMMON ist ein Schlüsselwort, v_i ist

eine einfache Variable oder
ein Feldname oder
ein Feldname mit Indexliste.

Beispiel

```
COMMON A, B, X(3)
```

Damit sind die ersten fünf Speichereinheiten des Speicherbereichs den Variablen A, B, X(1), X(2), X(3) zugeordnet. Mit dem Auftreten von X(3) in der COMMON-Anweisung wird gleichzeitig das Feld X definiert. Erfolgt die Definition des Feldes X in einer DIMENSION- oder Typanweisung, so darf in der COMMON-

Liste nur der Feldname erscheinen. Äquivalent wären also die Formulierungen

```
DIMENSION X(3)    und    COMMON A, B, X
COMMON A, B, X           DIMENSION X(3)
```

falsch wäre

```
DIMENSION X(3)
COMMON A, B, X(3)
```

Weitere Festlegungen:

1. Formale Parameter dürfen nicht in einer COMMON-Liste stehen.
2. Die Anordnung der Elemente im Speicherbereich erfolgt gemäß der Reihenfolge der Aufzählung in der Globalanweisung (Felder entsprechend ihrer festen Speicherfolge). Da der gemeinsame Speicherbereich an einer Doppelwortgrenze beginnt (ein Doppelwort entspricht zwei Speichereinheiten), führt man eventuelle DOUBLE PRECISION-Größen in einer Globalanweisung zweckmäßig zuerst an.
3. Stehen in einer PE mehrere COMMON-Anweisungen, so sind diese einer COMMON-Anweisung äquivalent, deren Liste durch Aneinanderfügen der einzelnen Listen in der Notationsreihenfolge entsteht.
4. Die Länge des gemeinsamen Speicherbereichs für das zusammengesetzte Programm bestimmt sich durch die COMMON-Anweisung aller PE mit dem größten Speicherplatzbedarf für ihre Elemente.
5. Eine COMMON-Anweisung beschreibt stets die Belegung des gemeinsamen Speicherbereichs vom Anfang an in ununterbrochener Reihenfolge, wobei nicht der volle Speicherbereich ausgeschöpft werden muß.

Beispiele

a)
```
C HP MAX                          FUNCTION MORITZ(P,Q)
      COMMON A, C(4), X           COMMON A, B, T(3), X
      .                           .
      .                           .
      .                           .
      END                         END
```

Gemeinsamer Speicherbereich (Beginn mit Speichereinheit i):

Speichereinheit	i	i+1	i+2	i+3	i+4	i+5
HP	A	C(1)	C(2)	C(3)	C(4)	X
Funktions-UP	A	B	T(1)	T(2)	T(3)	X

Da untereinanderstehende Variable also jeweils dieselbe Speichereinheit benutzen, kann ein im HP z. B. für C(1) eingespeicherter Wert im UP unter dem Namen B benutzt werden und umgekehrt.

b) Greifen wir auf das letzte Beispiel von 4.3.2.2. zurück. Dort sollte die achte Spalte einer Matrix (Tabelle) A(2Ø,12) als Vektor V(2Ø) in derselben PE (HP) benutzt werden. Jetzt soll diese Benutzung als Vektor in einer anderen PE (UP) erfolgen. Das läßt sich bequem über den COMMON-Bereich wie folgt erreichen:

```
C HP                              C UP
      COMMON A(2Ø,12)                   COMMON A(2Ø,7), V(2Ø)
      •                                 •
      •                                 •
      END                               END
```

Wird im UP das Feld A(2Ø,7) nicht benötigt, so ist diese Angabe eine (notwendige) Pseudoangabe.

c)
```
C HP                              FUNCTION F1(X)
      DOUBLE PRECISION C          COMMON V(2), A, B
      COMMON C, A(4)              •
      •                           •
      •                           END
      END                         FUNCTION F2
                                  COMMON R, S, T(6)
                                  •
                                  •
                                  END
```

Gemeinsamer Speicherbereich (Beginn mit Speichereinheit i):

Speicher-einheit	i	i+1	i+2	i+3	i+4	i+5	i+6	i+7
HP	C		A(1)	A(2)	A(3)	A(4)		
UP F1	V(1)	V(2)	A	B				
UP F2	R	S	T(1)	T(2)	T(3)	T(4)	T(5)	T(6)

Interessant ist nun, daß eine einfache oder indizierte Variable einer COMMON-Anweisung (gemeint ist eine indizierte Variable eines dort vorkommenden Feldes) auch in einer Äquivalenzanweisung auftreten kann. Dabei dürfen keine Widersprüche erzeugt werden. Insbesondere können nicht zwei verschiedene Variable einer COMMON-Anweisung durch eine Äquivalenzanweisung einander entsprechen. Das widerspräche der Organisation des COMMON-Bereichs. Eine Äquivalenzanweisung kann aber mittels einer indizierten Variablen zu einer Erweiterung des gemeinsamen Speicherbereichs (Speichern des ganzen zugehörigen Feldes im Speicherbereich) führen. Eine solche Erweiterung darf nur nach „rechts" über das letzte Element hinaus erfolgen.

Beispiel

In einer PE soll stehen

```
DIMENSION F(3), A(5)
COMMON F, WERT
EQUIVALENCE (F(3), A(1))
```

Der durch COMMON F, WERT zunächst definierte Speicherbereich von vier Speichereinheiten wird durch die Äquivalenzanweisung auf sieben Speichereinheiten nach „rechts" erweitert:

Speichereinheit	i	i+1	i+2	i+3	i+4	i+5	i+6
ursprünglicher Bereich	F(1)	F(2)	F(3)	WERT			
erweiterter Bereich	F(1)	F(2)	A(1)	A(2)	A(3)	A(4)	A(5)

Die Speichereinheit i + 2 beispielsweise ist damit in dieser PE sowohl mit F(3) als auch mit A(1) ansprechbar.
Fehlerhaft wäre

```
EQUIVALENCE (F(1), A(3))
```

da dann der ursprüngliche Speicherbereich um zwei Einheiten nach „links" zu erweitern wäre.
Ein ausführliches Anwendungsbeispiel mit COMMON-Größen folgt in 4.5.6.2.

4.5.6. Zusammengesetzte Programme

4.5.6.1. Allgemeines

Die wichtigsten Gesichtspunkte seien hier wiederholt und zusammengestellt.
Ein *zusammengesetztes Programm* besteht aus einem HP, dem in beliebiger Reihenfolge Funktions- und Subroutine-UP folgen können. Eine Verschachtelung oder Überlappung von PE ist nicht möglich. Der Programmkörper aller PE baut sich in der Reihenfolge

Vereinbarungsanweisungen
Anweisungsfunktionsdefinitionen
ausführbare Anweisungen
Endezeile

auf, falls entsprechende Anweisungen überhaupt vorkommen. Lediglich Formatanweisungen können beliebig eingefügt werden. Vergleichen Sie zur Klassifizierung der Anweisungen den Anhang A 4/2.
Die Abarbeitung einer PE beginnt mit der ersten ausführbaren Anweisung. Der erste Aufruf eines UP erfolgt vom HP aus. Ein UP kann ein anderes UP aufrufen. Der Rücksprung aus einem UP erfolgt stets in die aufrufende PE. Ein Subroutine-UP muß keinen Rücksprung bewirken. In der Regel wird aber ins HP zurückgekehrt und mit Erreichen der Stoppanweisung des HP ist das zusammengesetzte Programm abgearbeitet.
Bild 7 zeigt ein Prinzipbeispiel für ein zusammengesetzes Programm und seine Abarbeitung. Die Nummern an den vertikalen Pfeilen geben eindeutig die Abarbeitungsfolge an. Das Funktions-UP F2 wird zweimal aufgerufen.
Namen von einfachen Variablen, Feldern und Anweisungsfunktionen sind nur innerhalb der jeweiligen PE gültig. Über eine aktuelle Parameterliste oder eine Globalanweisung ist jedoch ihre Weitergabe (einschließlich ihres Wertes) an eine andere PE möglich.

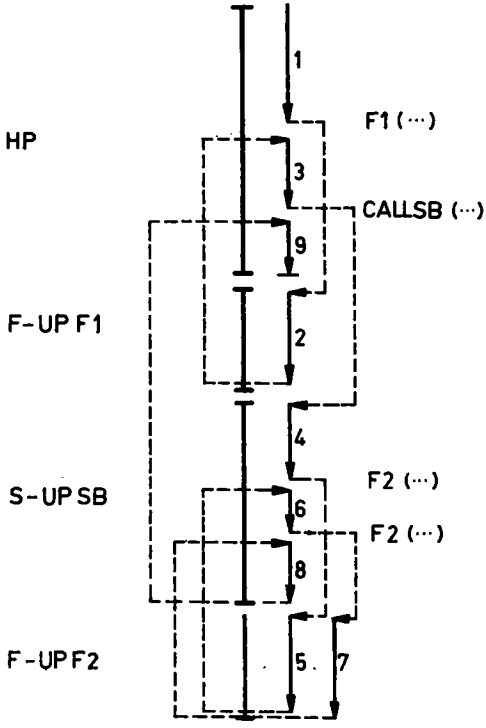


Bild 7

Namen von Funktions- und Subroutine-UP gelten im gesamten zusammengesetzten Programm. Letzteres bedeutet, daß sie in jeder PE, in der sie als UP-Name benutzt werden, dasselbe UP kennzeichnen.
Schließlich sind Marken ebenfalls lokal bezüglich der PE, der sie angehören.

4.5.6.2. Beispiel

Die Berechnung des Wertes des Vektorausdrucks

$(\underline{p} + \lambda \underline{q}) \cdot (\underline{v} + \mu \underline{w})$

soll mit Hilfe zweier Unterprogramme (Linearkombination zweier Vektoren; Skalarprodukt) erfolgen. Die Vektoren $\underline{p}$, $\underline{q}$, $\underline{v}$, $\underline{w}$ sollen je n Komponenten haben, λ und μ seien skalare Faktoren.
Wir wollen das zusammengesetzte Programm in zwei Versionen aufstellen: ohne und mit Globalanweisungen. Um Speicherplatz zu sparen, werden $\underline{v}$ und $\underline{w}$ nach der Berechnung des ersten Faktors auf die Plätze von $\underline{p}$ bzw. $\underline{q}$ eingelesen.

Das erste der genannten Unterprogramme ist ein Subroutine-UP und wird zweimal aufgerufen, das zweite ist ein Funktions-UP und hat nur einen Aufruf.

Programmversion 1

```
C VEKTORAUSDRUCK
      REAL LAMBDA, MUE
      DIMENSION P(5Ø),Q(5Ø),V(5Ø),W(5Ø),S1(5Ø),S2(5Ø)
      EQUIVALENCE (P(1), V(1)), (Q(1), W(1))
      READ (1,1Ø) N, LAMBDA, MUE, (P(I), I=1,N)
   1Ø FORMAT (I2 / 2F6.2 / 8(F1Ø.3))
      READ (1,2Ø) (Q(I), I=1,N)
   2Ø FORMAT (8F1Ø.3)
C Q BEGINNT MIT NEUER LOCHKARTE, DESHALB SEPARATE READ-ANWEISU
      CALL VADD(N,P,Q,1.,LAMBDA,S1)
      READ (1,2Ø) (V(I), I=1,N)
      READ (1,2Ø) (W(I), I=1,N)
      CALL VADD(N,V,W,1.,MUE,S2)
      WERT = SKAPR(N,S1,S2)
      WRITE (3,3Ø) WERT
   3Ø FORMAT (//'␣WERT␣SPEZ.␣SKALARPRODUKT␣=␣', F15.5)
      STOP
      END
C
      SUBROUTINE VADD(N,A,B,F1,F2,S)
C VADD BERECHNET DIE LINEARKOMBINATION F1*A + F2*B
      DIMENSION A(5Ø), B(5Ø), S(5Ø)
      DO 2 I = 1,N
    2 S(I) = F1*A(I) + F2*B(I)
      RETURN
      END
C
      FUNCTION SKAPR(N,A,B)
      DIMENSION A(5Ø), B(5Ø)
      SKAPR = Ø.Ø
      DO 2 I = 1,N
    2 SKAPR = SKAPR + A(I)*B(I)
      RETURN
      END
```

Achten Sie beim Studium dieses Beispiels besonders auf die Arten des UP-Aufrufs, die Zuordnungen zwischen formalen und aktuellen Parametern und die Rolle von Eingangs- und Resultatparametern.
Machen Sie sich einen Speicherbelegungsplan.

Programmversion 2 (Wir geben nur geänderte Passagen an.)

```
C VEKTORAUSDRUCK
      REAL LAMBDA, MUE
      COMMON P(5Ø),Q(5Ø),S1(5Ø),S2(5Ø)
      DIMENSION V(5Ø),W(5Ø)
      EQUIVALENCE (P(1), V(1)), (Q(1), W(1))
      .
      .
      .
      CALL VADD2(N,1.,LAMBDA,S1)
      .
      .
      .
      CALL VADD2(N,1.,MUE,S2)
      WERT = SKAPR2(N)
      .
      .
      .
      END
C
      SUBROUTINE VADD2(N,F1,F2,S)
      COMMON A(5Ø), B(5Ø)
      DIMENSION S(5Ø)
      .
      .
      .
      END
C
      FUNCTION SKAPR2(N)
      COMMON C(1ØØ), A(5Ø), B(5Ø)
      SKAPR2 = Ø.Ø
      DO 2 I = 1,N
    2 SKAPR2 = SKAPR2 + A(I)*B(I)
      RETURN
      END
```

Bemerkungen

Die Übergabe eines Teils der Versorgungsparameter an die UP erfolgt über den COMMON-Bereich, dessen Interpretation zu Beginn jedes UP mit einer Globalanweisung erfolgen muß. Der Ergebnisparameter S konnte nicht über den COMMON-Bereich übergeben werden, da VADD2 zweimal hintereinander aufgerufen wird und das Ergebnis des ersten Aufrufs zerstört (überschrieben) würde, falls man im HP kein Umspeichern zwischenschaltet. Beachten Sie ferner die Pseudoangabe C(1ØØ) in SKAPR2, da der COMMON-Bereich stets ab erste Stelle aufgeführt werden muß.

4.5.7. Aufgaben

4.5.7./1

Die Berechnung des Flächeninhalts eines Dreiecks bei drei gegebenen Seiten ist nach der HERONschen Formel

$$F = \sqrt{s(s-a)(s-b)(s-c)} \qquad \text{mit } s = \frac{a+b+c}{2}$$

Z

a) als Funktions-UP und
b) als Anweisungsfunktion zu programmieren.

Z

4.5.7./2

Mit Hilfe eines FORTRAN-Programms sollen die Funktionen $y_1 = \sinh x$ und $y_2 = \cosh x$ für die $x_i = a + (i - 1)h$ tabelliert werden (i = 1(1)n). Die Funktionswerte sind mit Anweisungsfunktionen zu berechnen. Zur Kontrolle soll jeweils $\cosh^2 x - \sinh^2 x$ mit ausgegeben werden.

T

4.5.7./3

Die Berechnung des Funktionswertes eines Polynoms

$$y = a_{n+1}x^n + a_n x^{n-1} + \ldots + a_2 x + a_1 \qquad (n \geqslant 1)$$

ist in Form eines Funktions-UP zu programmieren. Vergleichen Sie hierzu das letzte Beispiel in 4.3.2.3.1.
Wie lautet der UP-Aufruf zur Berechnung von

$$y = \frac{\sum_{i=1}^{n} c_i x^{i-1}}{\sum_{j=1}^{m} b_j (x-t)^{j-1}} \quad ?$$

Z

4.5.7./4

Die Berechnung des Binomialkoeffizienten

$$\binom{n}{k} = \frac{n}{k} \cdot \frac{n-1}{k-1} \cdot \ldots \cdot \frac{n-(k-1)}{1}$$

mit ganzzahligen n, k soll in einem Subroutine-UP erfolgen, das über einen zusätzlichen Ergebnisparameter L mitteilt, ob $\binom{n}{k}$ definiert ist (L = 1) oder nicht (L = 0 bei $k < 0$ und bei $n = k = 0$).

4.5.7./5

Für die Multiplikation zweier Matrizen $\underline{A} = (a_{i,k})$ und $\underline{B} = (b_{k,j})$ mit i = 1(1)m, k = 1(1)n, j = 1(1)p ist ein Subroutine-UP zu schreiben. Eine Darstellung des Algorithmus' als PAP findet sich im Abschnitt „Beschreibung eines Programmablaufs" dieses Lehrbuchs (vgl. Lösung zur Aufgabe 3., Abschnitt 1, S. 65).

4.5.7./6

Z

T

Die näherungsweise Berechnung eines Integrals nach der SIMPSONschen Formel

$$I = \int_a^b f(x)dx \approx \frac{h}{3}(f(a) + 4 \cdot \sum_{i=1}^{n} f(a + (2i - 1)h) + 2 \cdot \sum_{i=1}^{n-1} f(a + 2ih) + f(b))$$

mit $h = \frac{b-a}{2n}$ bei ganzzahligem $n \geqslant 2$ ist

a) als Funktions-UP zu schreiben und
b) dieses in einem zusammengesetzten Programm zur Berechnung des Integrals

$$\frac{1}{\sqrt{2\pi}} \int_0^x e^{-\frac{t^2}{2}} dt$$

für ein gegebenes x anzuwenden.

4.5.7./7

Ö

Es ist das Programm zu Beispiel 2 aus 4.3.2.4. unter Einbeziehung der beiden bekannten UP Matrix * Vektor (4.5.3.2.) und Skalarprodukt (4.5.6.2.) neu zu schreiben. Die Speicherausnutzung soll wie in 4.3.2.4. erfolgen, die Parametervermittlung (außer m und n) mit den UP entsprechend über einen COMMON-Bereich. Die UP sind demgemäß zu modifizieren.

4.5.7./8

T

Vorgegeben sei das Polynom wie in Aufgabe 4.5.7./3 durch seine Ordnung n und seine Koeffizienten a_i (i = 1(1)n + 1). Ferner seien ein Näherungswert x_0 für eine einfache Nullstelle und eine Genauigkeitsschranke ϵ gegeben. Schreiben Sie unter Benutzung des UP aus Aufgabe 4.5.7./3 ein Programm zur Berechnung der Nullstelle nach dem NEWTONschen Näherungsverfahren:

$$x_{i+1} = x_i - \frac{y(x_i)}{y'(x_i)}$$

Das Verfahren ist abzubrechen, wenn $|y(x_i)| < \epsilon$ erreicht ist.
Anleitung: Die Koeffizienten des Polynoms y' sind als ein Feld b abzuspeichern.

4.6. Hinweise zur praktischen Arbeit mit Basis-FORTRAN

4.6.1. Allgemeine Bemerkungen

Bei der rechentechnischen Bearbeitung eines Problems unterscheidet man drei Etappen: Planung, Programmierung und Programmerprobung. Wie die Erfahrung zeigt, werden insbesondere von Anfängern bei der Erarbeitung eines funktionstüchtigen Programms die Planungsetappe und die Erprobungsetappe in ihrer Bedeutung unterschätzt. Im allgemeinen benötigt die Programmerprobung den größten Teil der Gesamtzeit, die Programmierung den geringsten.
Zur Programmplanung gehören im allgemeinen: Auswahl eines geeigneten Lösungsverfahrens und dessen vollständige mathematische Beschreibung, Festlegung der ein- bzw. auszugebenden Größen sowie der Form der Ein- und Ausgabe (z. B. Druckbildgestaltung), zweckmäßige Strukturierung des Programms durch Abspaltung geeigneter Teile als Unterprogramme, algorithmische Aufbereitung des Lösungsprozesses durch Aufstellen eines Programmablaufplans, Wahl einer rationellen Speicherorganisation bei Problemen mit hohem Speicherbedarf, Vorbereiten der Programmerprobung durch zeitweiliges Einfügen von Kontrollen.
Allgemein ist eine einfache Programmversion stets einer komplizierten vorzuziehen. Fehler lassen sich dann leichter korrigieren und Veränderungen zu einem späteren Zeitpunkt leichter vornehmen. Eine wichtige diesbezügliche Möglichkeit besteht in der sinnvollen Strukturierung eines Programms mittels UP. UP sollte man nach Möglichkeit separat erproben. Voraussetzung hierfür ist allerdings, daß ein UP möglichst unabhängig von anderen UP ist.
Bei großen Problemen ist die Strukturierung mittels UP ohnehin eine technische Notwendigkeit infolge der von Gerätetechnik und Systemunterlagen gesetzten Kapazitätsgrenzen.
Der rationelle Umgang mit Speicherplatz macht sich wesentlich dadurch erforderlich, daß man die Ausführungszeit so niedrig wie möglich halten möchte. Prinzipielle Speicherplatzprobleme gibt es bei modernen Anlagen nicht mehr. Das Ziel ist die weitgehende Nutzung des Hauptspeichers wegen dessen kurzer Zugriffszeit. Speicherplatzökonomie erreicht man durch Mehrfachnutzung von Speicherplatz, worauf schon bei der Behandlung der EQUIVALENCE-Anweisung (s. 4.3.2.2.) und der COMMON-Anweisung (s. 4.5.5.) hingewiesen wurde, sowie durch Einsparung redundanter Daten. So kann man beispielsweise statt einer symmetrischen Matrix nur deren obere Dreicksmatrix speichern oder statt einer schwach besetzten Matrix deren von Null verschiedene Elemente zusammen mit ihren Indizes in drei eindimensionalen Feldern.

4.6.2. Programmerprobung

4.6.2.1. Fehlerarten

Erfahrungen der praktischen Programmierung besagen, daß ein neu geschriebenes Programm mehr oder minder fehlerbehaftet ist. Dabei müssen wir drei

Fehlerarten unterscheiden: *syntaktische Fehler, Laufzeitfehler* und *logische Fehler.*
Syntaktische Fehler resultieren aus dem falschen Gebrauch gewisser Sprachelemente und werden im allgemeinen in Verbindung mit dem Übersetzungsprozeß diagnostiziert.
Laufzeitfehler können erst während der Programmabarbeitung erkannt werden. Ein Beispiel dafür ist die Berechnung eines Index eines Feldelements, der die vereinbarte Indexgrenze übersteigt.
Beide Fehlerarten werden vom Datenverarbeitungssystem mit Hilfe sogenannter Fehlerdiagnoseprogramme spezifiziert ausgewiesen.
Logische Fehler sind Algorithmierungsfehler, also Fehler in der algorithmischen Aufbereitung der Aufgabenstellung. Sie müssen nicht mit einer der anderen Fehlerarten einhergehen. Damit sind solche Fehler häufig nicht automatisch diagnostizierbar und können recht tückisch sein.
Ein Fehler, der bei der Datenträgerherstellung entstanden ist, kann prinzipiell als ein Programmierfehler aufgefaßt werden und gehört damit einer der drei Arten an. Wurde beispielsweise I = J + 1 statt I = I + 1 gelocht, so ist das ein logischer Fehler und wird im allgemeinen zu keiner Meldung führen.

4.6.2.2. Vorbeugung gegen Fehler ohne Meldung

Der Programmierer muß von vornherein die Existenz von Fehlern einkalkulieren. Es genügt nicht, sich lediglich auf die vom DV-System zu liefernde Fehlerdiagnose zu verlassen, denn damit werden letztlich nur „Formfehler" (syntaktische und Laufzeitfehler) signalisiert, keine logischen Fehler, falls diese nicht zufällig mit „Formfehlern" einhergehen.
Eine zweckmäßige Prophylaxe ist das vorübergehende Einfügen geeigneter Kontrollen verbunden mit Kontrolldrucken. Über Art, Anzahl und Positionierung der Kontrollen ist in jedem Einzelfall gesondert zu entscheiden. Prinzipiell sollten möglichst alle Zweige jeder Programmeinheit erfaßt werden. Letzteres gilt auch für die Auswahl von Testbeispielen, die die Grundlage für die Auswertung der Kontrollen sind. Diese Testbeispiele sollen also möglichst repräsentativ sein. Häufig kann man auf bereits durchgerechnete Beispiele zurückgreifen. Ansonsten bleibt eine Durchrechnung von Hand nicht erspart. Eine überschlägige Einschätzung der Ergebnisse auf Grund der Erfahrung sollte man niemals als ausreichend betrachten.
Neben den zeitweiligen Erprobungskontrollen sollte ein Programm grundsätzlich den Kontrolldruck aller Eingangsdaten enthalten. Selbstverständlich müssen diese Werte mit den Originaldaten verglichen werden. Ebenfalls zum Schutz gegen Fehler beim Übertragen auf die Ablochbelege und bei der Datenträgerherstellung muß man den ersten Programmausdruck sorgfältig mit der handschriftlichen Version vergleichen.
Beispiel zur zeitweiligen Einführung eines Kontrolldrucks:
Zu berechnen sei der Ausdruck $r = \underline{a}^T \cdot (\underline{B} \cdot \underline{c})$ mit den Vektoren $\underline{a}$, $\underline{c}$ und der Matrix $\underline{B}$. Falls man sich nur den einzigen Resultatwert r ausgeben läßt, ist die

Lokalisierung eines eventuellen Fehlers sehr erschwert. Nehmen wir an, daß bei der Berechnung von $\underline{B} \cdot \underline{c}$ die Summationsvariable S für die einzelnen Skalarprodukte nur einmal auf Null gestellt wird, ein für Anfänger typischer Fehler. Dann dürfte die folgende Programmversion, die den Kontrolldruck des Vektors $\underline{B} \cdot \underline{c}$ enthält, sehr schnell zur Fehleraufklärung führen:

```
C BEISPIEL FUER KONTROLLDRUCK
      DIMENSION A(1Ø), B(1Ø,2Ø), C(2Ø)
      READ (1,1ØØ) A, B, C
  1ØØ FORMAT(1ØF8.3)
      WRITE (3,4ØØ) A, C, B
  4ØØ FORMAT (15H1EINGABE:␣A,C,B//1Ø(3X,F8.3)//1Ø(3X,F8.3)
     1/1Ø(3X,F8.3)//1Ø(3X,F8.3))
      R = Ø.Ø
      S = Ø.Ø
      DO 1 I = 1,1Ø
      DO 2 J = 1,2Ø
    2 S = S + B(I,J) * C(J)
      WRITE (3,2ØØ) S
  2ØØ FORMAT(1X,F16.6)
    1 R = R + A(I) * S
      WRITE (3,3ØØ) R
  3ØØ FORMAT(6HØ␣R␣=␣, F16.6)
      STOP
      END
```

4.6.2.3. Auswertung syntaktischer Fehlermeldungen

Die folgende Diskussion syntaktischer und Laufzeitfehlermeldungen erfaßt nicht alle Meldungen, da nur die typische Vorgehensweise demonstriert werden soll. Weitere Einzelheiten sind [7] zu entnehmen.
Syntaktische Fehler werden vom Übersetzerprogramm (Compiler) während des Übersetzungsprozesses erkannt und durch eine Fehlermeldung auf dem Zeilendrucker ausgewiesen. Diese Meldung gibt den Ort und die Klasse des Fehlers an. Man unterscheidet *lokale* und *globale* Fehler. Ein lokaler Fehler ist bei der Übersetzung einer Anweisung (unter Einbeziehung der Vereinbarungsanweisungen) erkennbar, während ein globaler Fehler nicht an eine einzelne Anweisung gebunden werden kann. Ein lokaler Fehler wird sofort ausgewiesen, indem im Protokolldruck des Quellprogramms auf die entsprechende Stelle ein festes Hinweiszeichen ($) in einer zusätzlich eingefügten Zeile verweist. Eine weitere zusätzliche Zeile enthält die Klassifizierung des Fehlers in Form eines charakteristischen Schlagwortes in englischer Sprache.

Beispiel 1:

```
              N = 8
              DIMENSION A(8,1Ø), B(N)
                        ⇵          ⇵
Ø1) ORDER               Ø2) SYNTAX
              COMMON A
                   ⇵
Ø1) ORDER
              .
              .
              .
```

Die erste Fehleranzeige weist darauf hin, daß die DIMENSION-Anweisung falsch plaziert (ORDER) ist, da sie nach der ausführbaren Anweisung N = 8 steht. Als nächster Fehler folgt eine unzulässige variable Indexgrenze in der DIMENSION-Anweisung (SYNTAX). Schließlich ist die COMMON-Anweisung ebenfalls falsch plaziert (ORDER).

Beispiel 2:

```
              .
              .
              .
              DO 1ØØ I = 1,3*M + 4
                               ⇵
Ø1). SYNTAX
              .
              .
              .
```

Fehlerhaft ist hier, daß in der DO-Anweisung als Testgröße keine Konstante oder einfache Variable verwendet wurde.
Wir geben im folgenden eine Auswahl der Klassifizierung lokaler Fehler des Basis-FORTRAN im DOS/ES verbunden mit Beispielen für das Auftreten dieser Fehler.

ALLOCATION	Mehrfaches bzw. widersprüchliches Auftreten von Größen, zum Beispiel mehrfache Vereinbarung desselben Namens oder Auftreten einer indizierten Variablen, wo nur eine einfache Variable zugelassen ist oder Auftreten eines formalen Parameters eines UP in einer COMMON- oder EQUIVALENCE-Anweisung
COMMA	Fehlendes Komma, zum Beispiel nach einer schließenden Klammer in einer Typ-, DIMENSION- oder EQUIVALENCE-Anweisung
DUP. LABEL	Doppelte Marke; Markierung mehrerer Anweisungen mit derselben Marke
ILLEGAL LBL	Fehlerhafte Marke in einer Steueranweisung, zum Beispiel FORMAT-Anweisung als Sprungziel oder Marke in einer Steueranweisung fehlt oder ist zu lang
LABEL	Fehlende Marke; eine FORMAT-Anweisung oder eine Anweisung, die einer IF-, GOTO-, RETURN- oder STOP-Anweisung folgt, trägt keine Marke
ID CONFLICT	Widersprüchliche Verwendung eines Namens, zum Beispiel Auftreten des Funktionsnamens gleichzeitig als formaler Parameter in einer FUNCTION-Anweisung oder eine indizierte

	Größe in einer EQUIVALENCE-Anweisung ist nicht als Feld vereinbart worden
NAME LENGTH	Name zu lang; kann durch Vergessen von Kommas oder Sonderzeichen verursacht sein
NO CORE	Speichermangel; die Übersetzung muß wegen Speichermangel abgebrochen werden (zur Behebung dieses Fehlers sei auf [7] verwiesen)
ORDER	Falsche Reihenfolge von Anweisungen
SIZE	Fehlerhafte Werte, zum Beispiel Zahlbereichsüberschreitung oder nichtpositive Indexgrenze in einer DIMENSION-Anweisung
SYNTAX	Allgemeiner syntaktischer Fehler; alle nicht durch die anderen Klassen erfaßten Fehler

Bei Auftreten eines lokalen Fehlers wird die Übersetzung nicht fortgeführt und damit kein Objektmodul erzeugt. Eine Ausnahme bildet hier lediglich der Fehler LABEL. Die Fehleranalyse wird jedoch fortgesetzt, wobei die fehlerhafte Anweisung nur insoweit ignoriert wird, wie sie fehlerhaft ist.
Bei lokalen und globalen Fehlermeldungen muß man grundsätzlich in Betracht ziehen, daß diese auch auf sogenannten Folgefehlern beruhen können. Es handelt sich hier um Fehler, die aus der Ignorierung fehlerhafter Anweisungen bzw. Teilen davon resultieren.

Beispiel:

```
                DIMENSION A(N)
                           Ø
Ø1) SYNTAX
                A(1) = SQRT(2 * 3.1415)
                 Ø
Ø1) SYNTAX
                :
```

Die Vereinbarung von A als Feld wird ignoriert. Damit ist gemäß impliziter Typfestlegung A eine einfache Variable vom Typ REAL. Die Charakterisierung der öffnenden Klammer bei A(1) als fehlerhaft ist daher lediglich Folge der falschen Feldvereinbarung.
Bei der Auswertung von Fehlermeldungen sollte man stets den ersten Fehler einer Anweisung genau überprüfen. Lassen sich weitere Fehlermeldungen zu dieser Anweisung nicht erklären, so übergeht man sie, da es sich höchstwahrscheinlich um Folgefehler handelt. Häufig erstrecken sich Folgefehler auch über mehrere Anweisungen. Der Ehrgeiz, Folgefehler aufklären zu wollen, ist fehl am Platz ohne genaue Kenntnisse über den Analysegang des Compilers.
Die Meldung globaler Fehler erfolgt im Anschluß an den Protokolldruck des Quellprogramms. Dabei werden mehrere Fehler derselben Klasse in einer Gruppe ausgewiesen. Diese beginnt mit dem entsprechenden Schlagwort, dem sich die Aufzählung der verursachenden Größen anschließt. Bei Fehlern, die Kapazitätsüberschreitungen betreffen, erfolgt lediglich die Ausgabe des Schlagwortes.

Ebenso wie bei einem lokalen Fehler wird auch bei einem globalen Fehler kein Objektprogramm erzeugt. Auf diese Tatsache verweist das Schlagwort COMPILATION TERMINATED als letzte Fehlermeldung.

Beispiel eines Programms mit Fehlermeldungen beider Arten:

```
        C BEISPIEL FUER FEHLERMELDUNGEN
              N = 1Ø
              DIMENSION A(8,1Ø), B(N), C(8)
                          $           $
Ø1) ORDER               Ø2) SYNTAX
              COMMON A
                    $
Ø1) ORDER
              EQUIVALENCE (A(1),C(2))
                          $
Ø1) ORDER
              READ (1,1ØØ) A, B
              DO 1 I = 1,8
              S = Ø.Ø
              DO 2 J = 1,1Ø
              S = S + A(I) * B(J)
                             $
Ø1) SYNTAX
            1 C(I) = S
              WRITE (3,2ØØ) C
          2ØØ FORMAT(5X, F1Ø.3)
              STOP
              END

                 COMMON ALLOCATION ERRORS
C
                          COMMON
SYMBOL  LOCATION     SYMBOL   LOCATION     SYMBOL  LOCATION...
A          ØØØØ      C           ØØØØ
                  UNCLOSED DO LOOP TARGETS
ØØØØ2      ØØØØ1
                    UNDEFINED LABELS
ØØ1ØØ      ØØØØ2
                      SCALARS
SYMBOL  LOCATION     SYMBOL   LOCATION     SYMBOL  LOCATION...
N          ØØ5Ø      B           ØØ54       I          ØØ58
                  COMPILATION TERMINATED
```

Die Deutung der Fehlermeldungen ist einfach und sei dem Leser in Verbindung mit den Fehlerübersichten überlassen. Das Beispiel zeigt noch, daß gewisse Fehlermeldungen durch die Ausgabe von Tabellen (zum Beispiel COMMON, SCALARS, ARRAYS) zur Unterstützung der Fehlersuche ergänzt werden. Beachten Sie auch, daß dieser erste Diagnoselauf nicht alle Fehler angezeigt hat.

Es folgt eine Auswahl von Meldungen globaler Fehler.

ARRAY ERRORS Zu groß definierte Felder; diese je einen Speicherbereich von 32768 Bytes überschreitenden Felder werden aufgeführt

COMMON ALLOCATION ERRORS
Widersprüche in der Vereinbarung des COMMON-Bereichs; die verursachenden Größen werden aufgelistet (zum Beispiel Widerspruch innerhalb des COMMON-Bereichs oder dessen Erweiterung nach links durch eine EQUIVALENCE-Anweisung)

COMPILATION TERMINATED, PROGRAM OVERFLOW
Abbruch der Übersetzung infolge zu großen Objektprogramms; der Speicherbedarf einschließlich der Variablen und Felder, die nicht COMMON-Größen sind, übersteigt 65532 Bytes

NON-COMMON EQUIVALENCE ERRORS
COMMON-unabhängige EQUIVALENCE-Fehler; die verursachenden Nicht-COMMON-Größen werden aufgelistet

UNCLOSED DO LOOP TARGETS
Nichtabgeschlossene DO-Schleifen; die anschließend aufgeführten Marken betreffen DO-Schleifen, die nicht abgeschlossen sind oder sich überlappen. (Im ersten Fall können korrekte umfassende Schleifen mit als nichtabgeschlossen ausgewiesen werden.)

UNDEFINED LABELS
Undefinierte Marken; diese Marken werden aufgelistet.

4.6.2.4. Auswertung von Laufzeitfehlermeldungen

Laufzeitfehler werden bei der Ausführung des Programms festgestellt und angezeigt. Die Programmausführung erfolgt erst, wenn das Programm frei von syntaktischen Fehlern ist. Die Meldung eines Laufzeitfehlers hat die Form IJTnnnI. Die Angabe nnn kennzeichnet die Fehlernummer.
Wir unterscheiden zwei Fehlertypen. Bei Typ 1 kann nach einer eventuellen automatischen Korrektur die Ausführung des Objektprogramms fortgesetzt werden. Ein Beispiel für einen solchen Fehler ist eine Programmunterbrechung bei der Ausführung einer arithmetischen Operation, etwa infolge einer Division durch Null. Hier erfolgt neben der Fehlermeldung IJT225I die Ausgabe des aktuellen Programmsteuerworts (PSW), das innerhalb der Fehlerklasse weiter zu differenzieren gestattet. Falls der Programmierer mit Hilfe des Standard-UP DVCHK keine spezielle Fehlermaßnahme vorgesehen hat, wird in diesem Fall automatisch mit dem Wert des Dividenden weitergerechnet. Weitere Einzelheiten dazu sind [7] zu entnehmen.
Nahezu alle Laufzeitfehler sind allerdings vom Typ 2 und führen zum Abbruch der Programmausführung. Beispiele für solche Fehler sind: Überschreiten der zulässigen Satzlänge bei Ausgabe mit Format (IJT212I); Ansprechen eines Gerätes

als Eingabegerät, das nur als Ausgabegerät verwendet werden kann (IJT216I); Aufruf der Standardfunktion SQRT mit negativem Argument (IJT251I); Überschreiten der Feldgrenze bei Aufruf eines Feldelements (IJT226I) usw.
Eine Übersicht aller Fehlermeldungen und weitere Einzelheiten müssen [7] entnommen werden.
Erwähnt sei noch, daß man sich bei einem Laufzeitfehler durch die Jobsteuerkarte // OPTION DUMP (vgl. auch Anhang A 4/10) die Inhalte der Register und einen Hauptspeicherauszug zur weiteren Unterstützung der Fehlersuche ausgeben lassen kann.

4.6.3. Hinweise zum Schreiben effektiver Programme

Von großer Bedeutung ist die Effektivität eines Programms. Eine allgemeine Definition dieses Begriffes ist kaum möglich, da je nach Programmierungsaufgabe mehr oder weniger Faktoren eine Rolle spielen. Bei Programmen, die nicht für die einmalige Nutzung – etwa im Rahmen einer Forschungsaufgabe – geschrieben werden, sondern für oft wiederkehrende Standardaufgaben, spielt zweifellos die Ausführungszeit eine hervorragende Rolle. Diese läßt sich mitunter durch die Auswahl eines „schnellen" Lösungsverfahrens, aber stets durch eine geschickte Programmierung reduzieren. Dazu muß man gewisse Hinweise zum Gebrauch von Feldern und einiger FORTRAN-Anweisungen beachten.

4.6.3.1. Verarbeiten und Ein- und Ausgabe von Feldern

Die Verarbeitungszeit einer indizierten Variablen steigt beträchtlich mit der Zahl ihrer Indizes. Der Grund liegt in der aufwendigeren Adressenrechnung, da auch zwei- und dreidimensionale Felder streng sequentiell wie eindimensionale Felder gespeichert werden. So erfordert das Aufsuchen einer Variablen B(I,J) bzw. B(I,J,K) die Adressenrechnung

$$A(B) + I + (J-1) \cdot M \quad \text{bzw.} \quad A(B) + I + (J-1) \cdot M + (K-1) \cdot M \cdot N$$

Dabei ist A(B) die Anfangsadresse des Feldes B minus 1, M die obere Grenze des ersten und N die obere Grenze des zweiten Index.
Legt man auf ein schnelles Programm Wert, so sollte man möglichst nur eindimensionale Felder benutzen. Man muß dann zwar die Adressenrechnung explizit programmieren, kann aber im allgemeinen effektiver vorgehen als bei obigen Formeln. Außerdem bietet sich über die EQUIVALENCE-Anweisung die Möglichkeit, zu einem Feldelement wahlweise auch mit der mehrfachen Indizierung zuzugreifen.
Beispiel: Von einer Matrix C vom Typ (30,30) interessieren die arithmetischen Mittel aller Spalten und das arithmetische Mittel der Hauptdiagonalelemente.

```
C HANDHABUNG EINER MATRIX ALS VEKTOR
      DIMENSION C(3Ø,3Ø), CC(9ØØ), SM(3Ø)
      EQUIVALENCE (C(1),CC(1))
      READ (1,2Ø) C
```

```
2Ø FORMAT(1ØF8.2)
   WRITE (1,4Ø) C
4Ø FORMAT(11H1EINGABE:␣C//1Ø(3X,F8.2))
   K = Ø
   DO 1 J = 1,3Ø
   S = Ø.Ø
   DO 2 I = 1,3Ø
   K = K + 1
 2 S = S + CC(K)
 1 SM(J) = S/3Ø.Ø
   S = Ø.Ø
 4 DO 3 I = 1,3Ø
 3 S = S + C(I,I)
   DM = S/3Ø.Ø
   WRITE (3,3Ø) SM, DM
3Ø FORMAT(/3Ø(7X,F8.2/)/7X,F8.2)
   STOP
   END
```

Erläuterung: CC ermöglicht den Zugriff zu C als Vektor. Die Adressenrechnung bei den Spaltenmitteln reduziert sich auf das jeweilige Weiterstellen von K um 1. In der Schleife zur Berechnung des Hauptdiagonalmittels wurde C benutzt. Da es sich hier um keine Schachtelung von DO-Schleifen handelt, fällt die höhere Ausführungszeit nicht sehr ins Gewicht. Natürlich könnte man auch hier CC benutzen und dann für die Anweisungen 4 und 3 schreiben:

```
4 DO 3 I = 1,9ØØ,31
3 S = S + CC(I)
```

Die Adressenrechnung läuft auf die Schrittweite 31 beim Zähler I hinaus. Bei dieser Version könnte die Benutzung von C ganz entfallen und dann statt CC gleich C geschrieben werden.

Es sei schließlich noch darauf hingewiesen, daß die Verarbeitung eines mehrdimensionalen Feldes einer aufrufenden Programmeinheit gegebenenfalls in einem UP vorteilhaft als eindimensionales Feld erfolgen kann. Dazu ist die Übermittlung des Feldnamens und der Anzahl der Feldelemente als Parameter an das UP ausreichend. Das UP kann auf diese Weise unabhängig von der Feldvereinbarung in der aufrufenden Programmeinheit geschrieben und aufgerufen werden.

Bei der Ein- und Ausgabe eines Feldes kann man drei prinzipiell verschiedene Programmierformen unterscheiden:

- Elementweise E/A gesteuert durch eine DO-Schleife, zum Beispiel

```
   DO 1 I = 1,5Ø
 1 WRITE (3,2Ø) L(I)
2Ø FORMAT(I8)
```

- E/A mittels implizitem DO, zum Beispiel

```
   WRITE (3,2Ø) (L(I), I=1,5Ø)
2Ø FORMAT(1ØI8)
```

- E/A lediglich mit dem Feldnamen, zum Beispiel

```
   WRITE (3,2Ø) L
2Ø FORMAT(1ØI8)
```

Es handelt sich jeweils um das gleiche ganzzahlige Feld L mit 50 Elementen. Während bei der ersten Variante prinzipiell nur ein Element pro Satz (Zeile) möglich ist, gibt es eine solche Schranke bei den anderen Varianten nicht. Die erste Variante ist durch den häufigen E/A-Aufruf und das kompliziertere Objektprogramm ungünstig. Die beiden anderen Varianten unterscheiden sich in der Ausführungszeit kaum, aber die dritte Variante ist einfacher in der Notation. Zu bedenken ist allerdings, daß man mit dieser Variante nur das vollständige Feld ausgeben kann.
Zu bemerken ist ferner, daß ein implizites DO mit einem „echten" Indexausdruck besser mit einer einfachen Variablen als Index dargestellt wird und dafür bedarfsweise Korrekturen an Anfangswert, Testwert und Schrittweite vorgenommen werden. Zum Beispiel ist

```
(A(I), I=2,78,4)    besser als    (A(4 + I - 2), I=1,2Ø)
```

4.6.3.2. Gebrauch spezieller Anweisungen und arithmetischer Ausdrücke

Einer IF-Anweisung sollte die Anweisung unmittelbar folgen, zu der voraussichtlich am häufigsten verzweigt wird.
Ganz besondere Aufmerksamkeit verdienen DO-Schleifen, da überflüssige Operationen innerhalb einer DO-Schleife so oft unnütz wiederholt werden, wie die Schleife durchlaufen wird. Das gilt in besonderem Maße für die innerste DO-Schleife einer Schachtelung. Es empfiehlt sich also eine „Optimierung" von DO-Schleifen, zumindest der innersten.
Grundsätzlich sollte man daher aus einer DO-Schleife alle Bestandteile herausnehmen, deren Berechnung einmalig erfolgen kann, das heißt nicht von der betreffenden Schleifenvariablen abhängt. In Verbindung mit der Vermeidung mehrfach indizierter Variabler in der innersten DO-Schleife läßt sich so die Ausführungszeit oft erheblich reduzieren. Der Faktor ist je nach Gestalt des Problems zwischen 0,7 und 0,2 zu erwarten.

Beispiel:

Aus einem Vektor $\underline{a}$ mit 30 Elementen sei ein Vektor $\underline{b}$ gemäß der Beziehung

$$\underline{b} = 3 \cdot \underline{a} - 4{,}2 \cdot (c + d_j)$$

zu berechnen. Die Form

```
      H = 4.2 * (C + D(J))
      DO 1 I = 1,3Ø
    1 B(I) = 3.Ø * A(I) - H
```

ist unbedingt der Form

```
      DO 1 I = 1,3Ø
    1 B(I) = 3.Ø * A(I) - 4.2 * (C + D(J))
```

vorzuziehen. Die Berechnung des Ausdrucks
`4.2 * (C + D(J))`
wird so 29mal eingespart.

Kommt die Laufvariable in der DO-Schleife nur in Form eines festen „echten" Ausdrucks vor, so gilt dasselbe wie beim impliziten DO (vgl. den letzten Absatz von 4.6.3.1.).
Neben der von uns besprochenen „Optimierung" innerhalb der Sprache FORTRAN besteht auch die Möglichkeit der Organisation einer innersten Schleife in Form eines UP, das in einer maschinennahen Sprache, der sogenannten Assemblersprache geschrieben wird.

Schließlich sei noch darauf hingewiesen, daß die Organisation einer Parameter-Variablen als COMMON-Variable mit dem Vermittlungsaufwand zwischen aufrufender und aufgerufener Programmeinheit auch die Ausführungszeit reduziert.
Bei der Notation von arithmetischen Ausdrücken empfiehlt es sich, eventuelle Konstanten typgerecht anzugeben, um Konvertierungszeit einzusparen. Eine ganzzahlige REAL-Konstante sollte daher stets mit .Ø und eine INTEGER-Konstante stets ohne .Ø angegeben werden.
Die Berechnung des Quadrates einer Variablen oder Konstanten schreibt man effektiver als Multiplikation. Bei größeren Exponenten als 2 empfiehlt sich beim Basis-FORTRAN im DOS/ES keine Schreibweise mittels Faktoren.
Ebenfalls nicht vorteilhaft hinsichtlich Ausführungszeit und Genauigkeit ist die Schreibweise A**Ø.5 statt SQRT(A).
Schließlich sei bemerkt, daß Standard-UP so effektiv sind, daß man sie dort, wo sich ihre Verwendung anbietet, auch anwenden und nicht durch eigene in der Regel unvorteilhaftere Versionen ersetzen soll

Lösungen

4.1.7./1

Den syntaktischen Regeln zur Bildung von Namen widersprechen:

X-1	(keine Sonderzeichen zulässig)
ØA1	(erstes Zeichen kein Buchstabe)
RHO␣4	(Leerzeichen nicht erlaubt)
ERGEBNIS	(Anzahl der Zeichen größer als sechs)
SUMME	(Kleinbuchstaben nicht im Zeichenvorrat enthalten)
M.N	(keine Sonderzeichen zulässig)

4.1.7./2

a) Fehlerhaft sind:

1E-A	(Exponent muß ganzzahlige Konstante sein)
1Ø.1.2	(nur ein Dezimalpunkt möglich)
Ø,ØØ	(Komma nicht erlaubt)
D1	(Mantisse fehlt)
7.O4	(Buchstabe O nicht zulässig)

Werte der in fehlerfreier Exponentendarstellung angegebenen Konstanten:

0,07
0,07
$80{,}2 \cdot 10^8$
0
−0,21

b) .572EØ 88.23E2 36EØ
-33E-18 44E-6 -ØEØ
(Darstellungen sind nicht eindeutig)

4.2.8./1

```
A**2/(B*ABS(C))
(((E*X+D)*X+C)*X+B)*X+A          (Berechnung mit HORNERschema)
SQRT(COS(2*X)**2+SIN(5*X)**2)
TANH(3.14159*X)*EXP(-ABS(X-A))/SQRT(1.Ø+X**3)
ABS(AMAX1(Z1,Z2,Z3,Z4,Z5)-AMIN1(Y1,Y2,Y3,Y4))
EXP(-(X-1.Ø)**2)-ALPHA*(X-1.Ø)**2
(X**(I+2))**N
```

4.2.8./2

X = −2.00
Y = 4.00
Z = 1.41

4.2.8./3

Anweisung 2: Name PARAMETER zu lang
Anweisung 3: Format I2 und Typ von N stimmen nicht überein
Anweisung 4: Argument von SIN nicht in Klammern eingeschlossen
Anweisung 5: G−1 als Name unzulässig
Anweisung 6: AMAXØ verlangt Argumente vom Typ INTEGER, Name PARAMETER falsch (G−1 wird als Ausdruck interpretiert)
Anweisung 8: Komma im F-Format unzulässig
Abschlußzeile END fehlt

4.2.8./4

```
C LOESUNG ZUR AUFGABE KOORDINATENUMRECHNUNG
      READ(1,1Ø) R,PHI
   1Ø FORMAT(2F8.2)
C DIESES FORMAT IST NUR ALS BEISPIEL ANZUSEHEN
      X = R*COS(PHI)
      Y = R*SIN(PHI)
      WRITE(3,11) R,PHI,X,Y
   11 FORMAT(1X ,4HR␣=␣,F8.2,5X,6HPHI␣=␣,F8.2,2ØX,4HX␣=␣,F8.2,
     15X,4HY␣=␣,F8.2)
C ANWEISUNG 11 MIT FORTSETZUNGSZEILE
      STOP
      END
```

Alle im Programm vorkommenden Variablen sind implizit vom Typ REAL.

4.2.2./5

Die Lösungen zu $Ax^2 + Bx + C = 0$ können nach

$$x_{1,2} = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

ermittelt werden.

```
C LOESUNG ZUR AUFGABE QUADRATISCHE GLEICHUNG
      READ(1,1Ø) A,B,C
   1Ø FORMAT(3F8.2)
      D = SQRT(B*B-4.Ø*A*C)
      X1 = (-B+D)/(2.Ø*A)
      X2 = (-B-D)/(2.Ø*A)
      WRITE(3,11) A,B,C
   11 FORMAT(1X,46HLOESUNGEN␣DER␣QUADRATISCHEN␣GLEICHUNG␣MIT␣DEN␣,
     116HKOEFFIZIENTEN␣A=,F8.2,4H,␣B=,F8.2,4H,␣C=,F8.2,2H␣:)
      WRITE(3,12) X1,X2
   12 FORMAT(1X,3ØX,3HX1=,F8.2,5X,3HX2=,F8.2)
      STOP
      END
```

4.3.1.5./1

a)
```
      IF (A + 1.5) 1Ø,1Ø,1
    1 IF (A - 1.5) 2Ø,2,2
    2 IF (A - 3.Ø) 3Ø,3,3
    3 IF (A - 4.5) 4Ø,5Ø,5Ø
```

b)
```
      IF (A + 1.5) 1,1,2
    1 L = 1
      GOTO 1ØØ
    2 IF (A - 4.5) 4,3,3
    3 L = 5
      GOTO 1ØØ
    4 L = A / 1.5
      L = L + 2
  1ØØ GOTO (1Ø,2Ø,3Ø,4Ø,5Ø), L
```

Bemerkungen

1. Bei den Fallunterscheidungen muß man beachten, daß A und die jeweilige Konstante intern als duale „Maschinenzahlen" im allgemeinen einen Rundungsfehler haben. Bei zu geringem Unterschied zwischen A und der Konstanten kann die Konvertierung gleiche Maschinenzahlen, d.h. den Fall A = Konstante erzeugen.
2. Bei b) können die beiden letzten Ergibtanweisungen nicht zu L = A/1.5.+2 zusammengefaßt werden, da sonst die Typwandlung REAL → INTEGER erst zum Schluß erfolgen würde und für $-1.5 < A < 0$ den Wert L = 1 ergäbe.

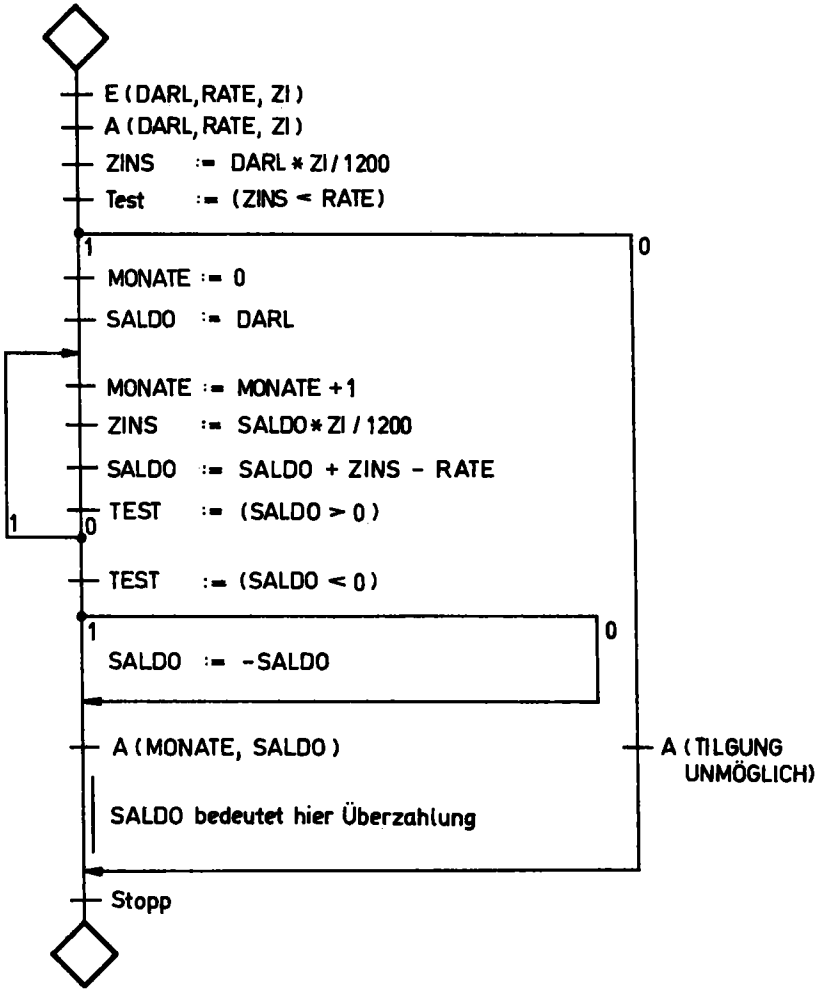
4.3.1.5./2

```
C LOESUNG ZUR AUFGABE GRUPPIERUNG
      N = Ø
      I = Ø
      J = Ø
      K = Ø
C N,I,J,K SIND INTEGER-VARIABLE (IMPLIZITE TYPVEREINBARUNG)
    8 READ (1,1Ø) A
   1Ø FORMAT (F 1Ø.4)
      IF (A) 2,9,3
    3 N = N + 1
      IF (A - 2.Ø) 4,4,5
    4 I = I + 1
      GOTO 8
    5 IF (A - 5.Ø) 6,6,7
    6 J = J + 1
      GOTO8
    7 K = K + 1
      GOTO 8
    9 P = I * 1ØØ / N
      Q = J * 1ØØ / N
      R = K * 1ØØ / N
      WRITE (3,2Ø) N
   2Ø FORMAT (1X,4HN␣=␣,I4)
      WRITE (3,21) P,Q,R
   21 FORMAT (1X,29HPROZENTANTEILE␣DER␣GRUPPEN␣:␣, 3F1Ø.2)
    2 STOP
      END
```

Bemerkung

Die Ausgabe am Schluß ließe sich auch mit einer WRITE-Anweisung mit zugehöriger FORMAT-Anweisung durchführen, indem man in letztere ein Satztrennzeichen aufnimmt (vgl. Beispiel 2 in 4.4.5.).

Ö


E (DARL, RATE, ZI)
A (DARL, RATE, ZI)
ZINS := DARL * ZI / 1200
Test := (ZINS < RATE)
1
0
MONATE := 0
SALDO := DARL
MONATE := MONATE + 1
ZINS := SALDO * ZI / 1200
SALDO := SALDO + ZINS - RATE
TEST := (SALDO > 0)
1
0
TEST := (SALDO < 0)
1
0
SALDO := -SALDO
A (MONATE, SALDO)
SALDO bedeutet hier Überzahlung
A (TILGUNG UNMÖGLICH)
Stopp

FORTRAN-Programm

```
C LOESUNG ZUR AUFGABE DARLEHNSTILGUNG
      READ (1,1Ø) DARL, RATE, ZI
   1Ø FORMAT (2F1Ø.2, F5.2)
      WRITE (3,2Ø) DARL, RATE, ZI
   2Ø FORMAT (1X,1ØHDARLEHN␣=␣, F1Ø.2, 3X, 7HRATE␣=␣, F1Ø.2,
     13X, 11HZINSFUSS␣=␣, F5.2)
      ZINS = DARL*ZI/12ØØ
      IF (ZINS - RATE) 3,2,2
    2 WRITE (3,21)
   21 FORMAT (1X,21H␣␣TILGUNG␣UNMOEGLICH.)
      STOP
    3 MONATE = Ø
      SALDO = DARL
    4 MONATE = MONATE + 1
      ZINS = SALDO*ZI/12ØØ
      SALDO = SALDO + ZINS - RATE
      IF (SALDO) 5,6,4
    5 SALDO = -SALDO
    6 WRITE (3,22) MONATE, SALDO
   22 FORMAT (1X,12H␣␣TILGUNG␣:␣, I3, 7H␣MONATE,
     28X, 15HUEBERZAHLUNG␣=␣, F1Ø.2, 5H␣MARK)
      STOP
      END
```

T

4.3.1.5./4

```
C LOESUNG ZUR AUFGABE SCHWINGKREIS
      REAL J,L
      READ (1,1Ø) R,L,C,T,A,B,PHI
   1Ø FORMAT (3F1Ø.4, 5X, 4F1Ø.4)
      F = EXP(-R * T / L * Ø.5)
      D = 1./(L * C) - R * R /(4. * L * L)
      IF (D) 3,2,1
    1 J = A * F * COS(SQRT(D) * T - PHI)
      GOTO 4
    2 J = F * (A + B * T)
      GOTO 4
    3 D = SQRT(-D)
      J = F * (A * EXP(-D * T) + B * EXP(D * T))
    4 WRITE (3,11) J,T
   11 FORMAT(1X,18HSCHWINGKREIS:␣J␣=␣, F1Ø.4, 15H␣␣ZUR␣ZEIT␣T␣=␣,
     1F1Ø.4)
      STOP
      END
```

Z

```
C LOESUNG ZUR AUFGABE ZWEIPERSONENSPIEL
      DIMENSION N(3)
      INTEGER ZEILE,ZUG,ZEI,ZU
      N(1) = 2
      N(2) = 3
      N(3) = 4
   6Ø READ (1,3Ø) ZEILE,ZUG
   3Ø FORMAT (I1, 3X, I1)
      J = ZEILE + 1 - (ZEILE / 3) * 3
      K = 6 - ZEILE - J
C ZEILE, J, K SIND EINE PERMUTATION VON 1, 2, 3
      N(ZEILE) = N(ZEILE) - ZUG
      IF (N(ZEILE)) 7Ø,1,3
    1 IF (N(J) * N(K)) 7Ø,2,7
C FALL ZWEIER LEERZEILEN
    2 WRITE (3,31)
   31 FORMAT (1X,36HENTFERNUNG␣RESTLICHER␣ZEILE␣UND␣SIEG
     111H␣FUER␣MICH.)
      STOP
    3 IF (N(J)) 7Ø,4,5
    4 J = ZEILE
      GOTO 7
    5 IF (N(K)) 7Ø,6,13
    6 K = ZEILE
C NUR ZEILEN J UND K NOCH BELEGT
    7 ZU = N(J) - N(K)
      IF (ZU) 8,1Ø,9
    8 ZEI = K
      ZU = -ZU
      GOTO 5Ø
    9 ZEI = J
   5Ø N(ZEI) = N(ZEI) - ZU
      WRITE (3,32) ZEI,ZU
   32 FORMAT (1X,6HZEILE␣, I1, 2X, 1ØHENTFERNEN␣, I1)
      PAUSE
      GOTO 6Ø
   1Ø IF (N(J))- 1) 7Ø,11,12
C FALL N(J) = N(K) = 1
   11 WRITE (3,33)
   33 FORMAT (1X,36HIHR␣SIEG␣IST␣SICHER,␣ICH␣GRATULIERE.)
      STOP
   12 ZEI = J
      ZU = 1
      GOTO 5Ø
C FALL ALLE N(I) NICHT NULL. SUCHE ZWEIER WERTGLEICHER :
   13 IF (N(1) - N(2)) 15,14,15
   14 ZEI = 3
      GOTO 19
```

```
   15 IF (N(1) - N(3)) 17,16,17
   16 ZEI = 2
      GOTO 19
   17 IF (N(2) - N(3)) 2Ø,18,2Ø
   18 ZEI = 1
   19 ZU = N(ZEI)
      GOTO 5Ø
   2Ø ZEI = 3
      M = N(1) * N(2) * N(3)
C ES GIBT NUR DIE N-TUPEL 1,3,4 ODER 1,2,4 ODER 1,2,3
      IF (M - 12) 22,21,7Ø
   21 ZU = 2
      GOTO 5Ø
   22 ZU = 1
      IF (M - 8) 23,5Ø,7Ø
   23 IF (N(2) - 3) 5Ø,24,7Ø
   24 ZEI = 2
      GOTO 5Ø
   7Ø STOP
      END
```

4.3.2.5./1

A = 0

4.3.2.5./2

Marke 2: Als Testwert ist nur eine positive Konstante oder eine einfache ganzzahlige Variable zulässig. Ausweg: Ersetzen von N + 1 − I durch K und Einfügen von K = N + 1 − I zwischen die Anweisungen 1 und 2.

Marke 3: 1. 1 + L ist kein Indexausdruck, richtig ist L + 1.
2. Die Sprünge nach 2 bewirken eine falsche Steuerung.
Erfolgt ein solcher Sprung einmal, wird die innere Schleife nie verlassen. Statt 2 müßte formal 7 stehen. Aber:

Marke 7: Eine DO-Schleife darf nicht mit einer Sprunganweisung enden. Richtig ist stattdessen:
7 CONTINUE

Die Anweisungen sortieren das Feld A auf Platz nach wachsenden Werten.

4.3.2.5./3

```
C MAX UND MIN VON N ZAHLEN
      DIMENSION A(999)
      REAL MAX, MIN
      READ (1,1Ø) N
   1Ø FORMAT(I3)
      READ (1,11) (A(I), I=1,N)
```

```
   11 FORMAT(5F15.3)
      MAX = A(1)
      MIN = A(1)
      DO 6 I = 2,N
      IF (A(I)-MAX) 3,6,5
    3 IF (A(I)-MIN) 4,6,6
    4 MIN = A(I)
      GOTO 6
    5 MAX = A(I)
    6 CONTINUE
      WRITE (3,2Ø) MAX, MIN
   2Ø FORMAT (1X, 6HMAX␣=␣, F15.3, 8X, 6HMIN␣=␣, F15.3)
      STOP
      END
```

4.3.2.5./4

Ö

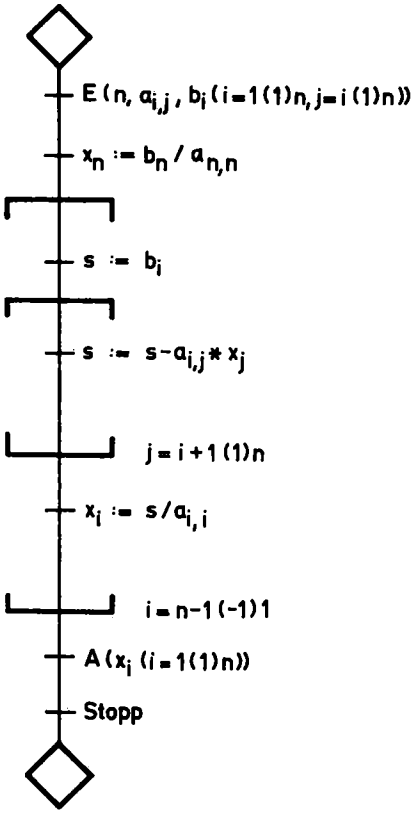
Mathematische Formulierung: $\underline{r} = \underline{A} \cdot \underline{B} \cdot \underline{e}$

Rechengang: $\underline{z} = \underline{B} \cdot \underline{e}, \underline{r} = \underline{A} \cdot \underline{z}$

Damit handelt es sich zweimal um die Aufgabe Matrix * Vektor.

```
C LOESUNG ZUR AUFGABE MATERIALBEDARF
      DIMENSION B(5Ø,5Ø), A(5Ø,5Ø), E(5Ø), Z(5Ø), R(5Ø)
      EQUIVALENCE (A(1), B(1)), (R(1), E(1))
C DAMIT WERDEN A AUF B UND R AUF E GESPEICHERT
      INTEGER P, S, T
      READ (1,1Ø) P,S,T
   1Ø FORMAT (3I2)
      READ (1,11) ((B(J,K), K=1,T), J=1,S)
C E/A MIT READ (1,11) B  WUERDE B(5Ø,5Ø) ERFASSEN
   11 FORMAT (5F1Ø.3)
      READ (1,11) (E(K), K=1,T)
      DO 1 J = 1,S
      SUM = Ø.Ø
         DO 2 K = 1,T
    2    SUM = SUM + B(J,K)*E(K)
    1 Z(J) = SUM
C NACH Z = B*E FOLGT R = A*Z
      READ (1,11) ((A(I,J), J=1,S), I=1,P)
      DO 3 I = 1,P
      SUM = Ø.Ø
         DO 4 J = 1,S
    4    SUM = SUM + A(I,J)*Z(J)
    3 R(I) = SUM
      WRITE (3,12)
   12 FORMAT (1X,16HMATERIALBEDARF␣:)
      WRITE (3,13) ((I,R(I)), I=1,P)
   13 FORMAT (11X, I2, 3X, F12.3)
      STOP
      END
```

T 4.3.2.5./5



Grundlage der Lösung sind die Formeln

$$x_n = \frac{b_n}{a_{n,n}}$$

und

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{i,j} \cdot x_j}{a_{i,i}}$$

für i = n−1(−1)1

FORTRAN-Programm

```
C LOESUNG ZUR AUFGABE GESTAFFELTES GLEICHUNGSSYSTEM
      DIMENSION A(5Ø,5Ø), B(5Ø), X(5Ø)
C DAMIT DARF N NICHT GROESSER ALS 5Ø SEIN
      READ (1,1Ø) N
   1Ø FORMAT (I2)
      READ (1,11) (((A(I,J), J=I,N), B(I)), I=1,N)
   11 FORMAT (5F15.3)
C DAS IST DIE GEWUENSCHTE EINGABE. SOLL MIT JEDER GLEICHUNG
C EINE NEUE LOCHKARTE BEGINNEN, IST DAS I MIT EINER SCHLEIFEN-
C ANWEISUNG ZU STEUERN.
      X(N) = B(N) / A(N,N)
      DO 2 K = 2,N
      I = N + 1 - K
      L = I + 1
```

```
      S = B(I)
         DO 3 J = L,N
C L IST NOETIG, DA HIER FUER L NICHT I + 1 STEHEN DARF
    3    S = S - A(I,J) *X(J)
    2 X(I) = S / A(I,I)
      WRITE (3,2Ø)
   2Ø FORMAT (1X,34HLOESUNG␣DES␣GESTAFFELTEN␣SYSTEMS␣:)
      WRITE (3,21) ((I, X(I)), I=1,N)
   21 FORMAT (15X, I2, 3X, F15.4)
      STOP
      END
```

Bemerkung

Zur Ausgabe vgl. Beispiel 2 in 4.3.2.4.

4.3.2.5./6

Z

Vergleichen Sie Ihren Programmablaufplan mit der Lösung der gleichen Aufgabe im Abschnitt „Beschreibung eines Programmablaufs" (Aufgabe 5) dieses Lehrbuches. Wir legen unserem Programm die zweite der dort angegebenen Lösungsformen (mit dem variablen Konnektor) zugrunde.

```
C LOESUNG ZUR AUFGABE MISCHFOLGE
      INTEGER Q
      DIMENSION A(3ØØ), B(3ØØ), C(6ØØ)
      READ (1,1Ø) M, N
   1Ø FORMAT (2I3)
      READ (1,11) (A(I), I=1,M)
      READ (1,11) (B(J), J=1,N)
   11 FORMAT (1ØF8.2)
      I = 1
      J = 1
      K = 1
      Q = 1
C Q WEGEN ANWEISUNG 5 UM 1 ERHOEHT GEGENUEBER PAP
      L = M + N
    7 IF (A(I) - B(J)) 2,2,3
    2 C(K) = A(I)
      I = I + 1
      K = K + 1
      IF (I - M) 5,5,4
    4 Q = Q + 1
      GOTO 5
    3 C(K) = B(J)
      J = J + 1
      K = K + 1
      IF (J - N) 5,5,6
    6 Q = Q + 2
```

```
    5 GOTO (7, 3, 2, 8), Q
    8 WRITE (3,12)
   12 FORMAT (1X,22HSORTIERTE␣MISCHFOLGE␣:)
      WRITE (3,13) (C(K), K=1,L)
   13 FORMAT (25X,F8.2)
      STOP
      END
```

Bemerkung

Zweckmäßiger wäre die Ausgabe von C in mehreren Spalten bei fortlaufender Anordnung jeweils innerhalb einer Spalte. Wegen der Vergrößerung des Programmieraufwandes wurde hier davon abgesehen.

4.4.8./1

102804
1028, 04
10, 2804
$-86, 3 \cdot 10^{2}$
$9, 12 \cdot 10^{-12}$
1, 203
1, 203
Fehler (kein Punkt erlaubt)
$-0, 33 \cdot 10^{-20}$
$1, 780 \cdot 10^{3}$

4.4.8./2

a)

```
A       --  E1Ø.Ø  }
B       --  E1Ø.Ø  }   1. Satz
X(1)    --  F5.2   |
X(2)    --  F5.2   |
X(3)    --  F5.2   }   2. Satz
X(4)    --  F5.2   |
X(5)    --  F5.2   |
F(5)    --  I6     }
TEXT    --  A2     }   3. Satz
```

b)

```
U               --  E1Ø.Ø
V               --  E1Ø.Ø
MATRIX(1,1)     --  I8
    :               :
    :               :
MATRIX(1,6)     --  I8
MATRIX(2,1)     --  I8
    :               :
    :               :
MATRIX(2,6)     --  I8
    :               :
```

```
Satz  1: ERGEBNISSE:
Satz  2: Leerzeile
Satz  3:      U= xxxxxxxxxx      V= xxxxxxxxxx
Satz  4:      MATRIX =
Satz  5:       xxxxxxxx       xxxxxxxx        ... xxxxxxxx (Zeile 1
                                                    von MATRIX)
  :
  :
  :
Satz 14:       xxxxxxxx       xxxxxxxx        ... xxxxxxxx (Zeile 10
                                                    von MATRIX)
```

4.4.8./3

a)
```
10 FORMAT(1H1,11X,3H***,4X,2H**,5X,1H*)
11 FORMAT(12X,15H*␣␣*␣␣*␣␣*␣␣␣**)
12 FORMAT(12X,4H*␣␣*,4X,7H*␣␣␣*␣*)
13 FORMAT(12X,3H***,4X,1H*,6X,1H*)
14 FORMAT(12X,2H**,4X,1H*,7X,1H*)
15 FORMAT(12X,7H*␣*␣␣␣*,7X,1H*)
16 FORMAT(12X,10H*␣␣*␣␣****,4X,1H*)
   WRITE(3,10)
   WRITE(3,11)
   WRITE(3,12)
   WRITE(3,13)
   WRITE(3,14)
   WRITE(3,15)
   WRITE(3,16)
```

b)
```
          VEKTOR X                VEKTOR Y

        xx.xxxxExxx             xx.xxxxExxx
        xx.xxxxExxx             xx.xxxxExxx
            :                       :              (N Zeilen)
        xx.xxxxExxx             xx.xxxxExxx
```

4.4.8./4

Ö

Einteilung der Lochkarten:

Spalten 1 bis 20	NAME	Stundenlohntabelle auf erster Karte
Spalten 22 bis 25	GEHNR	mit jeweils 3 Spalten für einen Wert
Spalte 70	LOHNGR	Spalten 1 bis 3 Stundenlohn in Gruppe 1
Spalten 76 bis 78	NSTD	:
Spalten 79 bis 80	UESTD	Spalten 25 bis 27 Stundenlohn in Gruppe 9

```
C LOESUNG ZUR AUFGABE BRUTTOLOHNRECHNUNG
      DIMENSION NAME(5),TAB(9)
      INTEGER GEHNR,UESTD
C FUER ALLE ANDEREN VARIABLEN GELTEN IMPLIZITE TYPVEREINBARUNGEN
      READ(1,1Ø) TAB
   1Ø FORMAT(9F3.2)
C EINLESEN DER STUNDENLOHNTABELLE
      WRITE(3,11)
   11 FORMAT('1BRUTTOLOHNLISTE:'///)
    1 READ(1,12) NAME,GEHNR,LOHNGR,NSTD,UESTD
   12 FORMAT(5A4,1X,I4,44X,I1,5X,I3,I2)
C EINLESEN DER DATEN FUER EINE LISTENPOSITION
      IF(LOHNGR)3,3,2
C PROGRAMMFORTSETZUNG NUR BEI LOHNGR VERSCHIEDEN VON NULL
    2 BRUTTO = (NSTD+1.25*UESTD)*TAB(LOHNGR)
      WRITE(3,13) NAME,GEHNR,BRUTTO
   13 FORMAT(1X,5A4,5X,I4,1ØX,F7.2,2H␣M)
C AUSDRUCKEN EINER LISTENPOSITION
      GOTO 1
    3 STOP
      END
```

Ö

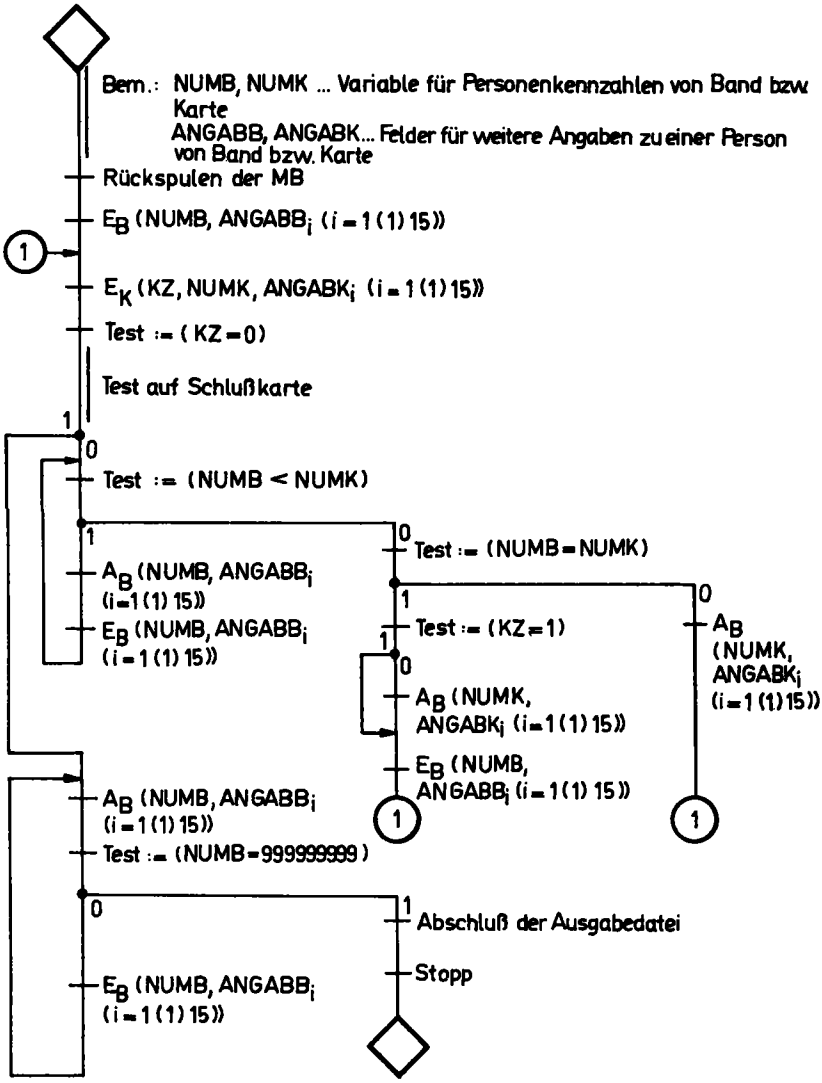
4.4.8./5

```
C LOESUNG ZUR AUFGABE KORRELATIONSKOEFFIZIENT
      DIMENSION X(1ØØØ),Y(1ØØØ),S(5)
      READ(1,1Ø) N,(X(I),Y(I),I=1,N)
   1Ø FORMAT(I4/(2F1Ø.3))
      DO 1 I=1,5
    1 S(I) = Ø.Ø
      DO 2 I=1,N
      S(1) = S(1)+X(I)
      S(2) = S(2)+Y(I)
      S(3) = S(3)+X(I)**2
      S(4) = S(4)+Y(I)**2
    2 S(5) = S(5)+X(I)*Y(I)
      RHO = (N*S(5)-S(1)*S(2))/SQRT((N*S(3)-S(1)**2)*
     1(N*S(4)-S(2)**2))
      WRITE(3,11) (X(I),Y(I),I=1,N)
   11 FORMAT(1H1,15X,'X',2ØX,'Y' //2(11X,F1Ø.3))
C KONTROLLDRUCK DER MESSREIHEN IN ZWEI SPALTEN
      WRITE(3,12) RHO
   12 FORMAT(/1ØX,'KORRELATIONSKOEFFIZIENT␣=␣',F1Ø.3)
      STOP
      END
```

Ö

Z

Die Anweisungen $E_B(\ldots)$ und $A_B(\ldots)$ charakterisieren das Lesen der Ursprungsdatei von einem Magnetband bzw. das Schreiben der geänderten Datei auf ein Magnetband, $E_K(\ldots)$ bezeichnet den Eingabevorgang über Lochkarten.



```
C LOESUNG ZUR AUFGABE AKTUALISIERUNG EINER PERSONALDATEI
      DIMENSION ANGABK(15),ANGABB(15)
      REWIND 12
      REWIND 13
C RUECKSPULEN DER MAGNETBAENDER FUER AUSGANGSDATEI
C UND GEAENDERTE DATEI
      READ(12) NUMB,(ANGABB(I),I=1,15)
    1 READ(1,5Ø) KZ,NUMK,(ANGABK(I),I=1,15)
   5Ø FORMAT(I1,I9,1ØX,15A4)
      IF(KZ)8,8,2
    2 IF(NUMB-NUMK)3,4,7
    3 WRITE(13) NUMB,(ANGABB(I),I=1,15)
      READ(12) NUMB,(ANGABB(I),I=1,15)
      GOTO 2
    4 IF(KZ-1)8,6,5
    5 WRITE(13) NUMK,(ANGABK(I),I=1,15)
    6 READ(12) NUMB,(ANGABB(I),I=1,15)
      GOTO 1
    7 WRITE(13) NUMK,(ANGABK(I),I=1,15)
      GOTO 1
C
    8 WRITE(13) NUMB,(ANGABB(I),I=1,15)
      IF(NUMB-999999999)9,1Ø,1Ø
    9 READ(12) NUMB,(ANGABB(I),I=1,15)
      GOTO 8
   1Ø ENDFILE 13
      STOP
      END
```

4.5.7./1

a)
```
FUNCTION F(A,B,C)
S = (A+B+C) * Ø.5
F = SQRT(S * (S-A) * (S-B) * (S-C))
RETURN
END
```

Z

b)
```
F(A,B,C) = Ø.25*SQRT((A+B+C)*(B+C-A)*(A+C-B)*(A+B-C))
```

4.5.7./2

Z

```
C LOESUNG ZUR AUFGABE TABELLIERUNG SINH(X) UND COSH(X)
      SINH(X) = (EXP(X) - EXP(-X)) * .5
      COSH(X) = (EXP(X) + EXP(-X)) * .5
      WRITE (3,1Ø)
   1Ø FORMAT ('1TABELLIERUNG␣:'// 8X, 'X', 8X, 'SINH(X)',
     18X, 'COSH(X)', 9X, 'PROBE' /)
      READ (1,2Ø) A, H, N
   2Ø FORMAT (2F5.2, 2X, I3)
      X = A
      DO 2 I = 1,N
      Y1 = SINH(X)
      Y2 = COSH(X)
      D = Y2 * Y2 - Y1 * Y1
      WRITE (3,3Ø) X, Y1, Y2, D
   3Ø FORMAT (6X, F5.2, 3(3X, E12.6))
    2 X = X + H
      STOP
      END
```

4.5.7./3

T

```
C LOESUNG ZUR AUFGABE FKT-UP POLYNOMWERT
      FUNCTION POLY(N,A,X)
      DIMENSION A(2Ø)
      S = A(N+1)
      DO 2 I = 1,N
      K = N + 1 - I
    2 S = S * X + A(K)
      POLY = S
      RETURN
      END
```

Aufruf: `Y = POLY(C,N-1,X) / POLY(B,M-1,X-T)`

4.5.7./4

Z

```
C LOESUNG ZUR AUFGABE SUBR-UP BINOMIALKOEFFIZIENT
      SUBROUTINE BIKOEF(N, K, BK, L)
      INTEGER BK
      IF (K) 11,2,4
    2 IF (N) 3,11,3
    4 IF (K-N) 6,3,1
    3 BK = 1
C DAMIT SIND DIE FAELLE K=N UND K=Ø BEARBEITET
      GOTO 1Ø
    1 IF (N) 8,5,5
```

```
 5 BK = Ø
   GOTO 1Ø
 6 IF (N - 2*K) 7,8,8
 7 M = N - K
   GOTO 12
 8 M = K
12 B = 1
   DO 9I = 1,M
 9 B = B * (N+1-I)/(M+1-I)
   BK = B+SIGN(0.5,B)
1Ø L = 1
   RETURN
11 L = Ø
   RETURN
   END
```

Es wird die Symmetrie $\left\lfloor \begin{matrix} n \\ n-k \end{matrix} \right\rfloor = \left\lfloor \begin{matrix} n \\ k \end{matrix} \right\rfloor$ benutzt, falls $n - k < k$ ist.

4.5.7./5

```
C LOESUNG ZUR AUFGABE SUBR-UP MATRIZENPRODUKT
      SUBROUTINE MAPROD (A,B,C,M,N,P)
      DIMENSION A(2Ø,2Ø), B(2Ø,2Ø), C(2Ø.2Ø)
      DO 2 I = 1,M
         DO 2 J = 1,P
         S = Ø.Ø
            DO 3 K = 1,N
    3       S = S + A(I,K) * B(K,J)
    2       C(I,J) = S
        RETURN
        END
```

Z

T

4.5.7./6

a) Aufbereitung: $I = \frac{2h}{3}\left(\frac{f(a) - f(b)}{2} + \sum_{i=1}^{n}(2f(a + (2i - 1)h) + f(a + 2ih))\right)$

```
C LOESUNG ZUR AUFGABE FKT-UP SIMPSON-INTEGRATION
      FUNCTION SIMPS (A,B,F,N)
      D = (B-A) / N
      H = D * Ø.5
      S = (F(A) - F(B)) * Ø.5
      X = A + H
    2 S = S + 2 * F(X) + F(X+H)
      X = X + D
      IF (X - B) 2,3,3
    3 SIMPS = S * D / 3.Ø
      RETURN
      END
```

b)
```
C LOESUNG ZUR AUFGABE PROGRAMM SIMPSON-INTEGRATION
      EXTERNAL FUN
      READ (1,1Ø) X, N
   1Ø FORMAT (F1Ø.5, 3X, I3)
      WERT = 1 / SQRT(2 * 3.1416) * SIMPS(Ø, X, FUN, N)
      WRITE (3,2Ø) X, WERT
   2Ø FORMAT ('ØWERT␣DES␣INTEGRALS␣VON␣Ø␣BIS␣', F1Ø.5,
     1'␣␣=␣␣', F7.5)
      STOP
      END
C
      FUNCTION SIMPS (A,B,F,N)
         :
         :
      END
C
      FUNCTION FUN(X)
      FUN = EXP(-X * X * Ø.5)
      RETURN
      END
```

4.5.7./7

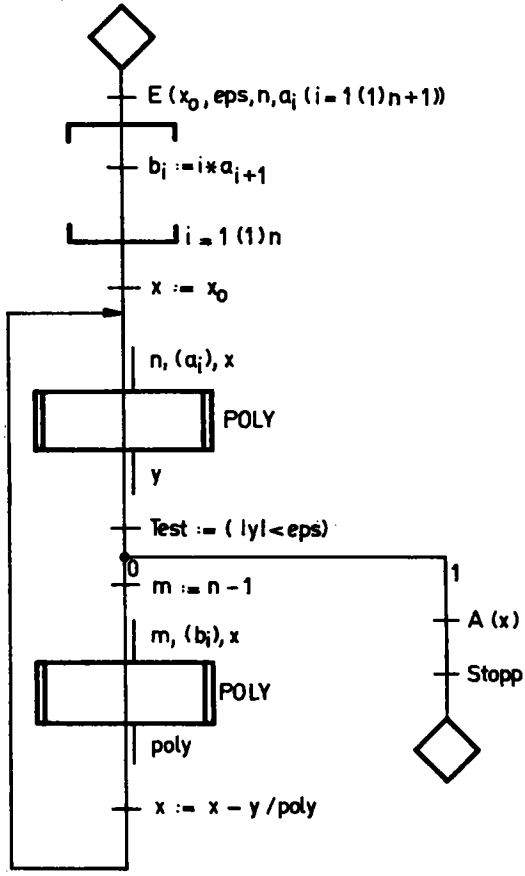
Ö

```
C LOESUNG ZUR AUFGABE MATERIALKOSTEN
      COMMON A(3Ø,3Ø), E(3Ø)
      DIMENSION P(3Ø)
      EQUIVALENCE (E(1),P(1))
        :   } Eingabe wie in 4.3.2.4.
        :
      CALL MAVEK2 (M,N)
      READ (1,3Ø) (P(I), I=1,M)
   3Ø FORMAT (1ØF8.2)
      S = SKAPR3 (M)
        :   } Ausgabe wie in 4.3.2.4.
        :
      END
C
      SUBROUTINE MAVEK2 (M,N)
      COMMON A(3Ø,3Ø), B(3Ø)
      DIMENSION R(3Ø)
      EQUIVALENCE (A(1),R(1))
      DO 3 I = 1,M
      S = Ø.Ø
         DO 2 J = 1,N
    2    S = S + A(I,J) * B(J)
    3 R(I) = S
      RETURN
      END
```

```
C
      FUNCTION SKAPR3 (N)
      COMMON R(3Ø), C(29,3Ø), B(3Ø)
C C(29.3Ø) STEHT, DA SONST B(3Ø) DIE PLAETZE 31 BIS 6Ø DES
C COMMON-BEREICHS ERFASSEN WUERDE
      S = Ø.Ø
      DO 2 I = 1,N
    2 S = S + R(I) * B(I)
      SKAPR3 = S
      RETURN
      END
```

T

4.5.7./8



Das UP POLY wurde als Lösung von Aufgabe 4.5.7./3 aufgestellt.

FORTRAN–Programm

```
C LOESUNG ZUR AUFGABE POLYNOMNULLSTELLE
      DIMENSION A(2Ø), B(2Ø)
      READ (1,1Ø) XØ, EPS, N, (A(I), I=1,N), A(N+1)
C DA I=1,N+1 UNZULAESSIG IST, WIRD A(N+1) GESONDERT AUFGEFUEHRT
   1Ø FORMAT (2E1Ø.5 / I2 / 8(F1Ø.5))
      DO 2 I = 1,N
    2 B(I) = I * A(I+1)
      X = XØ
    4 Y = POLY(N,A,X)
      IF (ABS(Y) - EPS) 5,3,3,
    3 X = X - Y / POLY(N-1,B,X)
      GOTO 4
    5 WRITE (3,2Ø) X
   2Ø FORMAT (// 16H␣NULLSTELLE␣X␣=␣, F1Ø.5)
      STOP
      END
C
      FUNCTION POLY(N,A,X)
        ⋮          } siehe Lösung zu Aufgabe 4.5.7./3
```

ANHANG

A 4/1 EBCDI-Lochkartenkode für Zeichen von Basis-FORTRAN im DOS/ES

Zeichen	Loch-kombination	interne Darstellung (hexadezimal)
A	12–1	C1
B	12–2	C2
C	12–3	C3
D	12–4	C4
E	12–5	C5
F	12–6	C6
G	12–7	C7
H	12–8	C8
I	12–9	C9
J	11–1	D1
K	11–2	D2
L	11–3	D3
M	11–4	D4
N	11–5	D5
O	11–6	D6
P	11–7	D7
Q	11–8	D8
R	11–9	D9
S	0–2	E2
T	0–3	E3
U	0–4	E4
V	0–5	E5
W	0–6	E6
X	0–7	E7
Y	0–8	E8
Z	0–9	E9

Zeichen	Loch-kombination	interne Darstellung (hexadezimal)
$	11–8–3	5B
0	0	F0
1	1	F1
2	2	F2
3	3	F3
4	4	F4
5	5	F5
6	6	F6
7	7	F7
8	8	F8
9	9	F9
,	0–8–3	6B
.	12–8–3	4B
=	8–6	7E
(	12–8–5	4D
)	11–8–5	5D
+	12–8–6	4E
–	11	60
*	11–8–4	5C
/	0–1	61
'	8–5	7D
Leer-zeichen	keine Lochung	40

A 4/2 Zusammenstellung und Klassifizierung aller Anweisungen von Basis-FORTRAN im DOS/ES

Klassifizierung				Schlüsselwort	Definition in
Anweisung	nichtausführbare Anweisung	Vereinbarungsanweisung	Typanweisung	INTEGER REAL DOUBLE PRECISION	4.2.5. 4.3.2.1.1.
			Feldanweisung	DIMENSION	4.3.2.1.1.
			Äquivalenzanweisung	EQUIVALENCE	4.3.2.2.
			Globalanweisung	COMMON	4.5.5.
			Externanweisung	EXTERNAL	4.5.2.1.2.
			Dateidefinitionsanweisung	DEFIE FILE	A 4/8
		Formatanweisung		FORMAT	4.2.4.
		Anweisungsfunktionsdefinition		entfällt	4.4.3. 4.5.2.2.
		Unterprogrammanweisung	Funktionsanweisung	FUNCTION typ FUNCTION	4.5.2.1.1.
			Subroutineanweisung	SUBROUTINE	4.5.3.1.
	ausführbare Anweisung	Wertanweisung	Ergibtanweisung	entfällt	4.2.3.
		Steueranweisung	unbedingte Sprunganweisung	GOTO	4.3.1.1.
			berechnete Sprunganweisung		4.3.1.3.
			Wennanweisung	IF	4.3.1.2.
			Schleifenanweisung	DO	4.3.2.3.
			Leeranweisung	CONTINUE	
			Pauseanweisung	PAUSE	A 4/6
			Stoppanweisung	STOP	
			Rufanweisung	CALL	4.5.3.1.
			Rücksprunganweisung	RETURN	4.5.2.1.1.
		sequentielle E/A-Anweisung	Eingabeanweisung	READ	4.2.4.
			Ausgabeanweisung	WRITE	4.4.2.
			Rückspulanweisung	REWIND	
			Rücksetzanweisung	BACKSPACE	4.4.6.
			Dateiendeanweisung	ENDFILE	
		E/A-Anweisung für Direktzugriff	Eingabeanweisung	READ	
			Ausgabeanweisung	WRITE	A 4/8
			Suchanweisung	FIND	

A 4/3 Konvertierungsregeln und Ergebnistyp bei arithmetischen Ausdrücken A op B

a) op: + – * /

Falls A und B von unterschiedlichem Typ sind, wird vor Ausführung der Operation der Operand mit dem niedrigeren Typ (Reihenfolge INTEGER – REAL – DOUBLE PRECISION) durch ein IN-LINE-Unterprogramm automatisch in den höheren Typ konvertiert. Der Ergebnistyp ist dann gleich dem höheren Operandentyp bzw. gleich dem gemeinsamen Typ der Operanden.

Verwendete IN-LINE-Unterprogramme:

FLOAT	Konvertierung von INTEGER in REAL
DFLOAT	Konvertierung von INTEGER in DOUBLE PRECISION
DBLE	Konvertierung von REAL in DOUBLE PRECISION

b) op: **

Typ von A \ B	INTEGER	REAL	DOUBLE PRECISION
INTEGER	keine Konvertierung INTEGER	Anwendung von FLOAT REAL	Anwendung von DFLOAT DOUBLE PRECISION
REAL	keine Konvertierung REAL	keine Konvertierung REAL	Anwendung von DBLE DOUBLE PRECISION
DOUBLE PRECISION	keine Konvertierung DOUBLE PRECISION	keine Konvertierung DOUBLE PRECISION	keine Konvertierung DOUBLE PRECISION

Bemerkung

Wenn der Exponent B vom Typ REAL oder DOUBLE PRECISION ist, darf die Basis A nicht negativ sein.

A 4/4 Standardfunktionen von Basis-FORTRAN im DOS/ES

Mathematische Bezeichnung	Funktionsname	Parameter		Typ der Funktion	Erläuterung
		Anzahl	Typ		
OUT-OF-LINE:					
Quadratwurzel	SQRT DSQRT	1 1	REAL DOUBLE PRECISION	REAL DOUBLE PRECISION	$\sqrt{x}$ mit $x \geq 0$
Sinus	SIN DSIN	1 1	REAL DOUBLE PRECISION	REAL DOUBLE PRECISION	$\sin(x)$ mit $\lvert x\rvert \leq c \cdot \pi$ $(c = 2^{18})$ $(c = 2^{50})$
Kosinus	COS DCOS	1 1	REAL DOUBLE PRECISION	REAL DOUBLE PRECISION	$\cos(x)$ mit $\lvert x\rvert \leq c \cdot \pi$ $(c = 2^{18})$ $(c = 2^{50})$
Arcustangens	ATAN DATAN	1 1	REAL DOUBLE PRECISION	REAL DOUBLE PRECISION	arctan (x)
Tangens hyperbol.	TANH DTANH	1 1	REAL DOUBLE PRECISION	REAL DOUBLE PRECISION	tanh(x)
Exponentialfunktion	EXP DEXP	1 1	REAL DOUBLE PRECISION	REAL DOUBLE PRECISION	e^x mit $x < 174{,}673$
Natürlicher Logarithmus	ALOG DLOG	1 1	REAL DOUBLE PRECISION	REAL DOUBLE PRECISION	$\ln(x)$ mit $x > 0$
Dekadischer Logarithmus	ALOG10 DLOG10	1 1	REAL DOUBLE PRECISION	REAL DOUBLE PRECISION	$\lg(x)$ mit $x > 0$
Abschneiden des gebrochenen Teils	INT AINT IDINT	1 1 1	REAL REAL DOUBLE PRECISION	INTEGER REAL INTEGER	$\lvert x\rvert < 2^{24}$ [x] (s. Bemerkung) $\lvert x\rvert < 2^{31}$
Divisionsrest	MOD AMOD DMOD	2 2 2	INTEGER REAL DOUBLE PRECISION	INTEGER REAL DOUBLE PRECISION	$x_1 - [x_1/x_2] \cdot x_2$ mit $x_2 \neq 0$

Mathematische Bezeichnung	Funktionsname	Parameter		Typ der Funktion	Erläuterung
		Anzahl	Typ		
Maximum	MAXØ	n	INTEGER	INTEGER	
	MAX1	n	REAL	INTEGER	
	AMAXØ	n	INTEGER	REAL	$\max(x_1,x_2,\ldots,x_n)$ $(n \geq 1)$
	AMAX1	n	REAL	REAL	
	DMAX1	n	DOUBLE PRECISION	DOUBLE PRECISION	
Minimum	MINØ	n	INTEGER	INTEGER	
	MIN1	n	REAL	INTEGER	
	AMINØ	n	INTEGER	REAL	$\min(x_1,x_2,\ldots,x_n)$ $(n \geq 1)$
	AMIN1	n	REAL	REAL	
	DMIN1	n	DOUBLE PRECISION	DOUBLE PRECISION	
IN-LINE:					
Absoluter Wert	ABS	1	REAL	REAL	
	IABS	1	INTEGER	INTEGER	$\lvert x \rvert$
	DABS	1	DOUBLE PRECISION	DOUBLE PRECISION	
Vorzeichenübertragung	SIGN	2	REAL	REAL	
	ISIGN	2	INTEGER	INTEGER	$\text{sign}(x_2)\,\lvert x_1 \rvert$
	DSIGN	2	DOUBLE PRECISION	DOUBLE PRECISION	
Schwache Differenz	DIM	2	REAL	REAL	
	IDIM	2	INTEGER	INTEGER	$x_1 - \min(x_1,x_2)$
Konvertierung	FLOAT	1	INTEGER	REAL	Umwandlung von ganzzahlig in reell
	DFLOAT	1	INTEGER	DOUBLE PRECISION	Umwandlung von ganzzahlig in doppeltgenau
	IFIX	1	REAL	INTEGER	Umwandlung von [x] in ganzzahlig
	SNGL	1	DOUBLE PRECISION	REAL	Umwandlung von doppeltgenau in reell (Verkürzen der internen Darstellung)
	DBLE	1	REAL	DOUBLE PRECISION	Umwandlung von reell in doppeltgenau (internes Erweitern mit Nullen)

Bemerkung

[x] = sign(x) · Gx, wobei Gx die größte ganze Zahl kleiner oder gleich |x| ist.

A 4/5 Standard-Service-Unterprogramme von Basis-FORTRAN im DOS/ES

Zweck	Prozeduraufruf	Bemerkungen
Überlauf anzeigen	OVERFL(j)	j := 1 Exponentenüberlauf j := 3 Exponentenunterlauf j := 2 sonst
		j ganzzahlige Variable Fehler wird bis zur Abfrage gemerkt, danach wird die Kennzeichnung gelöscht
Division durch Null anzeigen	DVCHK(j)	j := 1 Division durch Null erfolgte j := 2 Division durch Null erfolgte nicht
Speicherausdruck mit Programmabbruch	DUMP(u_1,o_1,f_1,...,u_n,o_n,f_n)	u_i,o_i ganzzahlige Variable $i = 1(1)n,\ n \geq 1$ f_i ganzzahlige Konstante u_i,o_i untere und obere Grenze des auszudruckenden i-ten Hauptspeicherbereichs f_i Form des Druckes : $f_i = 4$ jedes Wort als ganze Zahl $f_i = 5$ jedes Wort als reelle Zahl mit E-Exponent $f_i = 6$ jedes Doppelwort als reelle Zahl doppelter Genauigkeit mit D-Exponent $f_i = \emptyset$ jedes Wort in hexadezimaler Form Wirkung: Bei Aufruf des UP erfolgt Druck obiger Hauptspeicherbereiche, anschließend Programmabbruch
Speicherausdruck ohne Programmabbruch	PDUMP(u_1,o_1,f_1,...,u_n,o_n,f_n)	Wirkung: Analog DUMP Nach Speicherausdruck Fortsetzung der Programmabarbeitung
Beenden eines Programms	EXIT	Wirkung: Der Aufruf des UP beendet die Abarbeitung des Programms

A 4/6 Syntax und Semantik der Anweisungen PAUSE, STOP und CONTINUE

Bezeichnung	Form	Semantik	Bemerkungen
Pausen-anweisung	PAUSE PAUSE n	Unterbrechung der Programm-ausführung Anzeige des Zustandes PAUSE und der eventuellen Mitteilung. Programmausführung kann vom Steuerpult der Anlage aus mit der folgenden Anweisung fortgesetzt werden.	
Stopp-anweisung	STOP STOP n	Programmausführung wird beendet Anzeige von STOP und der eventuellen Mitteilung.	Die der STOP-Anweisung im FORTRAN-Programm folgende Anweisung muß eine Marke tragen.
Leer-anweisung	CONTINUE	Leere Anweisung ohne Einfluß auf Objektprogramm.	Verwendung oft als letzte Anweisung einer DO-Schleife (vgl. 4.3.2.3.)

Bemerkung

n repräsentiert eine Mitteilung, die aus maximal fünf Ziffern bestehen kann.

A 4/7 Ergänzung der Formatlistenelemente

1. Während die durch aufeinanderfolgende Formate beschriebenen Datenfelder im Datensatz unmittelbar hintereinanderstehen, kann durch die *Positionierung*

   Tp $(1 \leqslant p \leqslant 254$, ganzzahlig)

   als Formatlistenelement explizit festgelegt werden, daß die nächsten Formate der Formatliste aufeinanderfolgende Datenfelder ab der Position p im Datensatz beschreiben (vgl. dazu auch A 4/9).

2. Vor den Formaten einer Formatliste (mit Ausnahme eines in Apostrophe eingeschlossenen Literals) darf ein *Skalenfaktor* n in der Form

   nP $(-127 \leqslant n \leqslant 127$, ganzzahlig)

   stehen. Seine Wirkung hängt vom jeweiligen Format ab. Nach der expliziten Angabe eines Skalenfaktors vor einem Format gilt dieser für alle folgenden Formate der Formatliste bis zu seiner expliziten Änderung. Durch ØP kann er insbesondere auf Null festgelegt werden. Solange in einer Formatliste kein Skalenfaktor explizit auftritt, hat er den Wert Null.

3. Wirkung des Skalenfaktors bei den einzelnen Formaten:

Format	Eingabe	Ausgabe
F	$Z_{intern} = Z_{extern} \cdot 10^{-n}$ (Änderung des Zahlenwertes um Zehnerpotenzen)	$Z_{extern} = Z_{intern} \cdot 10^{n}$
E, D	Zahlen ohne Exponent: wie bei F Zahlen mit Exponent: wirkungslos	Modifikation der Mantisse um 10^n mit entsprechender Exponentenänderung (keine Veränderung des Zahlenwertes)
sonstige	wirkungslos	wirkungslos

Beispiel

Ausgabe des internen Wertes 1,23456789

Format	ausgegebene Zeichenfolge
ØPF8.3	␣␣␣1.235
2PF8.4	123.4568
-2PF8.6	Ø.Ø12346
3PE1Ø.2	123.46E-Ø2
-2PE1Ø.4	Ø.ØØ12E␣Ø3

A 4/8 Anweisungen zur Verarbeitung von Dateien im direkten Zugriff

Dateien, deren Sätze im *direkten Zugriff* verarbeitet werden sollen (vornehmlich Plattendateien), sind vorher in einer Vereinbarungsanweisung gesondert zu definieren:

DEFINE FILE $d_1\,(n_1,s_1,k_1,z_1), d_2\,(n_2,s_2,k_2,z_2), \ldots, d_q\,(n_q,s_q,k_q,z_q)$ $(q \geqslant 1)$.

Die Größen d_i, n_i, s_i $(i = 1(1)q)$ sind natürliche Zahlen und kennzeichnen die Nummern der zu verarbeitenden Dateien, die Anzahlen der in ihnen enthaltenen Sätze und die maximalen Längen der zu übertragenden Sätze (bei Plattendateien Übereinstimmung mit der festen Länge von Plattendateien notwendig). Mit den Kennzeichen k_i $(i = 1(1)q)$ ist die vorgesehene Übertragungsart wie folgt festzulegen:

E – mit formatgebundenen E/A-Anweisungen (Satzlänge in Bytes)
U – mit formatfreien E/A-Anweisungen (Satzlänge in Worten)
L – mit oder ohne Format (Satzlänge in Bytes).

Jedem Satz der definierten Datei d_i ist eine Satznummer j mit $1 \leqslant j \leqslant n_i$ entsprechend seiner Stellung innerhalb der Datei zugeordnet. Um auch eine sequentielle Verarbeitung dieser Dateien zu ermöglichen, ist jeder Datei mit der Nummer d_i zur Kennzeichnung des dem zuletzt verarbeiteten Satz unmittelbar folgenden eine *assoziierte Variable* z_i zugeordnet.

Zur Übertragung eines beliebigen Satzes einer Datei für Direktzugriff oder mehrerer Sätze ab einem beliebigen Satz stehen die Anweisungen

READ (d'j, f)l

WRITE (d'j, f)l

zur Verfügung. Zu den bereits in den E/A-Anweisungen für sequentielle Verarbeitung vorkommenden Größen Dateinummer d, Formatanweisungsmarke f und Übertragungsliste l kommt jetzt noch die Nummer j des ersten zu übertragenden Satzes hinzu. Für j kann dabei ein ganzzahliger Ausdruck angegeben werden. Wenn der Übertragungsvorgang formatfrei in interner Form erfolgen soll, entfällt die Angabe von f. Analog zur sequentiellen Datenübertragung kann auch die Übertragungsliste l entfallen. Für die Konvertierung der zu übertragenden Datenelemente und den Ablauf der Datenübertragung gelten die für die sequentiellen E/A-Anweisungen angegebenen Regeln. So ist die Anzahl der übertragenen Datensätze bei der formatgebundenen Übertragung durch die Formatsteuerung festgelegt, die formatfreie Übertragung erfaßt jeweils nur einen einzelnen Satz.
Um den Suchprozeß beim Übertragen von Datensätzen im direkten Zugriff zu verkürzen, kann die Datei d durch die Anweisung

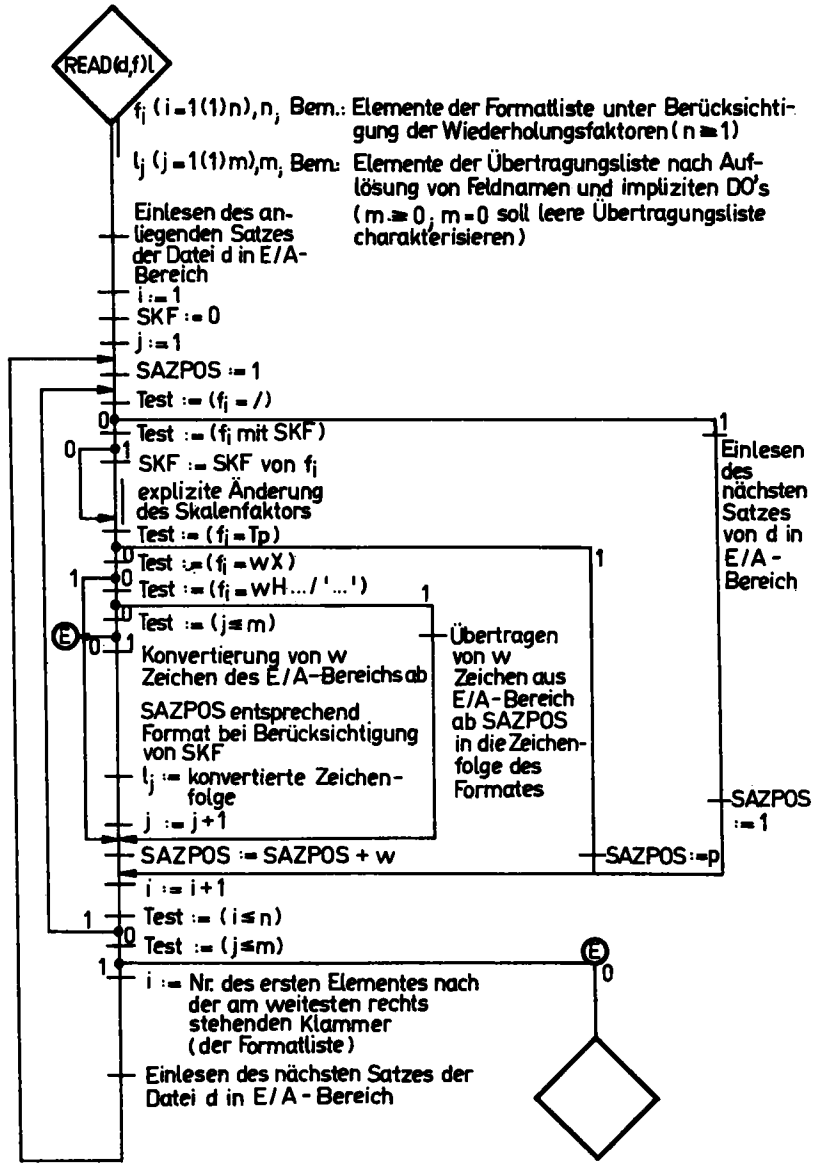
FIND (d'j)

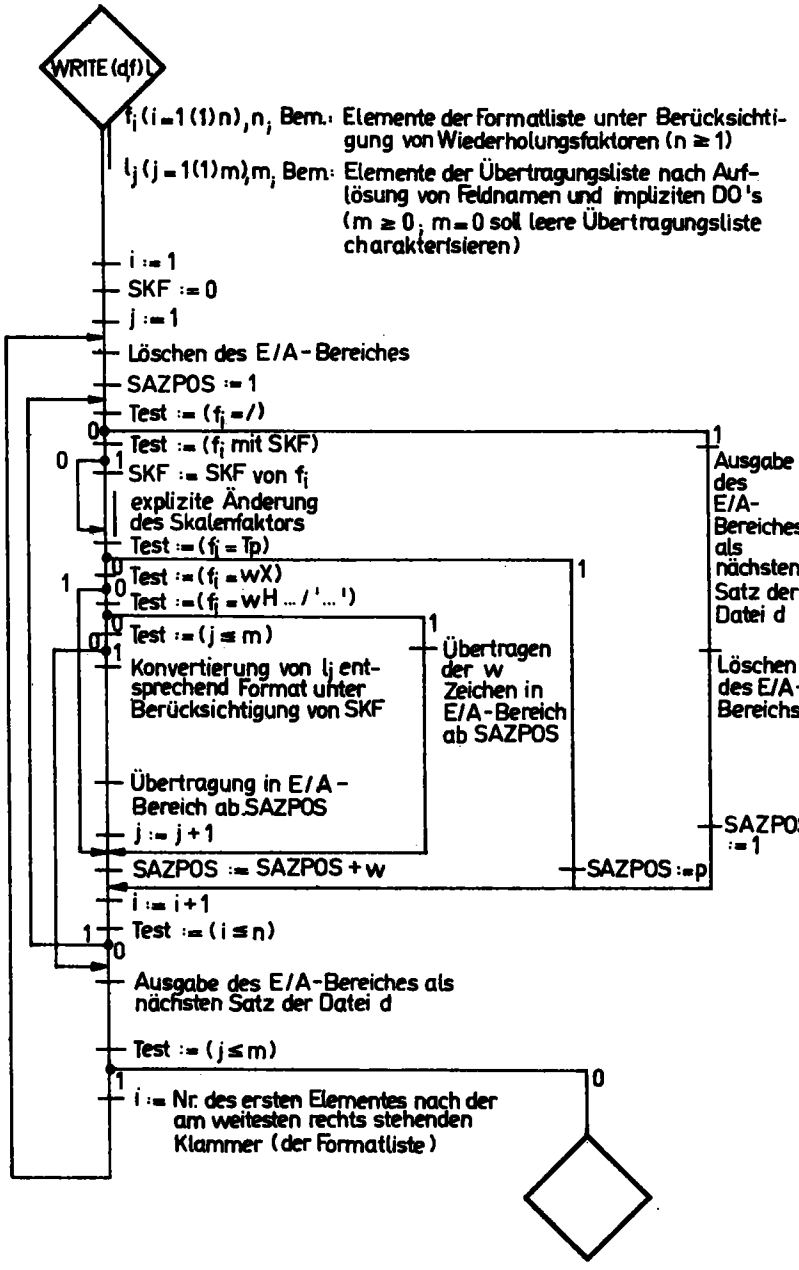
auf den durch j festgelegten Satz simultan zur Abarbeitung der nachfolgenden Anweisungen positioniert werden. Lese- oder Schreibkopf des Magnetspeichers befinden sich danach auf dem Anfangspunkt des Satzes mit der Nummer j. Sowohl die Anweisungen READ und WRITE als auch FIND belegen bei ihrer Abarbeitung die assoziierte Variable mit Werten:

$$z = \begin{cases} j & \text{nach vollständiger Abarbeitung einer FIND-Anweisung} \\ j + \text{Anzahl der übertragenen Sätze} & \text{nach vollständiger Abarbeitung einer E/A-Anweisung für Direktzugriff.} \end{cases}$$

z enthält also die Nummer des nächsten Satzes der Datei. Die Übermittlung des Wertes der assoziierten Variablen an eine andere Programmeinheit ist nur über einen COMMON-Bereich möglich. Die assoziierte Variable einer Datei kann nicht Bestandteil der Übertragungsliste einer diese Datei betreffenden E/A-Anweisung sein.

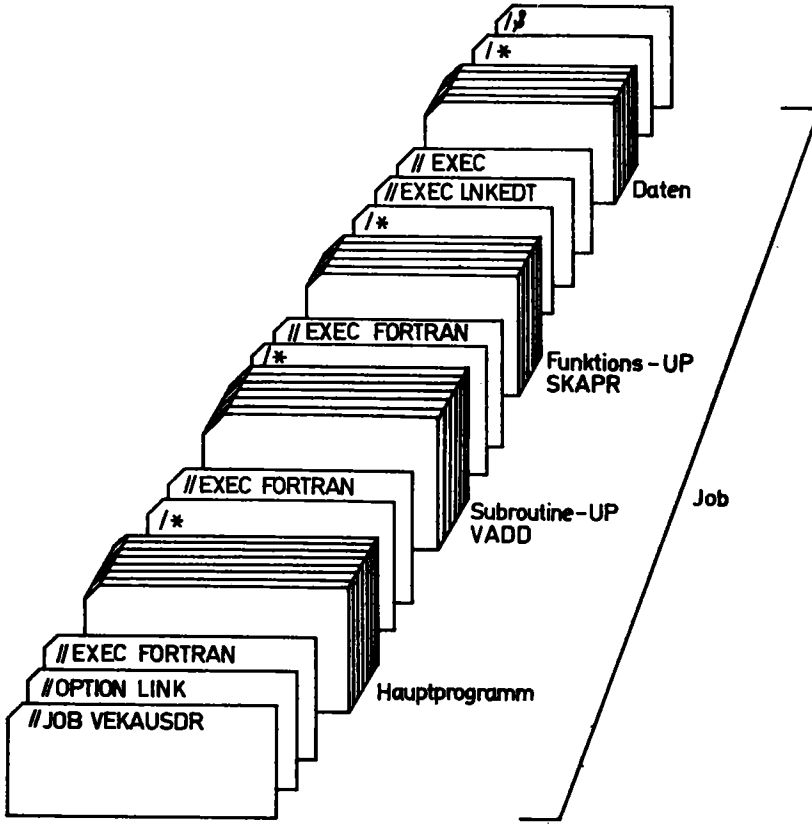
A 4/9 Ablaufdiagramme für die sequentiellen formatgebundenen E/A-Anweisungen




WRITE (d,f) L
f_i (i = 1 (1) n), n; Bem.: Elemente der Formatliste unter Berücksichtigung von Wiederholungsfaktoren (n ≥ 1)
l_j (j = 1 (1) m), m; Bem.: Elemente der Übertragungsliste nach Auflösung von Feldnamen und impliziten DO's (m ≥ 0; m = 0 soll leere Übertragungsliste charakterisieren)
i := 1
SKF := 0
j := 1
Löschen des E/A-Bereiches
SAZPOS := 1
Test := (f_i = /)
Test := (f_i mit SKF)
SKF := SKF von f_i
explizite Änderung des Skalenfaktors
Test := (f_i = Tp)
Test := (f_i = wX)
Test := (f_i = wH .../'...')
Test := (j ≤ m)
Konvertierung von l_j entsprechend Format unter Berücksichtigung von SKF
Übertragen der w Zeichen in E/A-Bereich ab SAZPOS
Übertragung in E/A-Bereich ab SAZPOS
j := j + 1
SAZPOS := SAZPOS + w
SAZPOS := p
Ausgabe des E/A-Bereiches als nächsten Satz der Datei d
Löschen des E/A-Bereichs
SAZPOS := 1
i := i + 1
Test := (i ≤ n)
Ausgabe des E/A-Bereiches als nächsten Satz der Datei d
Test := (j ≤ m)
i := Nr. des ersten Elementes nach der am weitesten rechts stehenden Klammer (der Formatliste)

A 4/10 Beispiel eines kompletten Jobs

Um ein in FORTRAN geschriebenes Programm auf einer EDVA abarbeiten zu können, bedarf es noch einer Aufbereitung. Das vollständige FORTRAN-Programm (Hauptprogramm und Unterprogramme) ist zusammen mit den vom Programm zu verarbeitenden Eingangsdaten und gewissen Steueranweisungen für das Betriebssystem der Anlage zu einem *Job* zusammenzustellen. Das folgende Beispiel zeigt eine Möglichkeit zur Ausführung des Programmbeispiels VEKTORAUSDRUCK (s. 4.5.6.2., Programmversion 1) im DOS/ES:



Soll das Programm mehrfach hintereinander mit verschiedenen Eingangsdaten ausgeführt werden, ist die Kombination

```
// EXEC - Datenkarten - /*
```

entsprechend oft einzugliedern.

Bedeutung der verwendeten Steuerkarten

`// JOB VEKAUSDR`	Jobkarte mit Jobnamen und eventuellem Kommentartext zur Kennzeichnung des Jobanfangs
`// OPTION LINK`	Festlegung einer bestimmten Variante der Programmausführung (Ausgabe des übersetzten Programms auf bestimmten Plattenspeicherbereich, von dem Programmverbindereingabe erfolgen wird)
`// EXEC FORTRAN`	Starten der Übersetzung eines FORTRAN-Programms
`// EXEC LNKEDT`	Verbinden der übersetzten Programme zu einem ausführbaren Maschinenprogramm
`// EXEC`	Ausführung des verbundenen Programms
`/*`	Abschluß der Programm- bzw. Datenkarten und Aufruf der Jobsteuerung
`/&`	Angabe des Jobendes

Literaturverzeichnis

[1] GRUND, F.:
FORTRAN IV – Programmierung (Berlin 1972)

[2] PAULIN, G.:
FORTRAN-Training – Aufgaben mit Lösungen (Berlin 1971)

[3] RANFT, J.:
FORTRAN-Programmierung und numerische Methoden für Naturwissenschaft und Technik (Leipzig 1972)

[4] SCHATZ/SCHOCH:
FORTRAN (Leipzig 1971)

[5] STEMPELL, D.:
FORTRAN – Programmierte Einführung (Berlin 1970)

[6] VEB Kombinat ROBOTRON:
Basis-FORTRAN – Beschreibung der Sprache
(Anwenderbeschreibung für ESER/ROBOTRON 21)

[7] VEB Kombinat ROBOTRON:
Basis-FORTRAN – Hinweise für die Arbeit mit dem Plattenbetriebssystem
(Anwenderbeschreibung für ESER/ROBOTRON 21)

[8] FORTRAN – Dokumente und Erläuterungen (3. Beiheft zur Zeitschrift „Rechentechnik/Datenverarbeitung" 1971)

EVP 4,50 Mark

LSV-Nr. 0394 / Bestell-Nr. 674 747 6