SIEMENS

SIEMENS SYSTEM 4004

FORTRAN IV
(Band-Betriebs-
system)

Beschreibung

Vorläufige Ausgabe

SIEMENS AKTIENGESELLSCHAFT

Bestell-Nr. 2-2600-302

SIEMENS

SIEMENS SYSTEM 4004

FORTRAN IV
(Band-Betriebs-
system)

Beschreibung

Vorläufige Ausgabe
Okt. 1966

SIEMENS AKTIENGESELLSCHAFT

Inhaltsverzeichnis

Inhaltsverzeichnis (Fortsetzung)

Inhaltsverzeichnis (Fortsetzung)

1. FORTRAN IV für das Siemens-System 4004

1.1. Einführung

FORTRAN IV ist das neueste und vollständigste Glied einer Familie von symbolischen Programmiersprachen, die auf der mathematischen Schreibweise aufbauen. Sein Gebrauch ist besonders zweckmäßig zum Schreiben von Programmen für statistische, wissenschaftliche und technische Anwendungen.

Der FORTRAN IV-Compiler ist ein Programm, das die in FORTRAN geschriebenen Anweisungen eines Primärprogramms analysiert und sie in ein Maschinenprogramm übersetzt. Das vom Compiler erzeugte Maschinenprogramm wird zusammen mit geeigneten Unterprogrammen aus der Programmbibliothek auf der Anlage ausgeführt und veranlaßt den Ablauf der Vorgänge, die in dem Primärprogramm beschrieben sind.

Das Manual beschreibt die FORTRAN IV-Sprache für das Siemens-System 4004. Sie enthält alle die Bestandteile, die in dem von der American Standard Association (ASA) beschlossenen FORTRAN vorgesehen sind. Zusätzlich bietet sie zahlreiche weitere wirkungsvolle Möglichkeiten wie die allgemeinen Typenanweisungen, gemischte Arithmetik, Literalangaben ohne Zeichenzählung, FORMAT-Spaltensteuerung sowie die Anweisung NAMELIST. Es sind ferner Vorkehrungen getroffen, um die in früheren FORTRAN-Versionen geschriebenen Programme ohne Änderung und ohne erneutes Ablochen übersetzen zu können. Außerdem kann der Programmierer bestimmen, ob das Primärprogramm aufgelistet werden soll, welche Testhilfen er wünscht usw. Diese Möglichkeiten sind in Anhang 3 "Übersetzung und Ausführung von FORTRAN-Programmen" zusammen mit dem Band-Betriebssystem (BBS) und dem Monitor beschrieben.

Die allgemeinen Richtlinien für das Schreiben von FORTRAN-Programmen sind in Abschnitt 2 enthalten. Die FORTRAN-Sprache baut sich aus Elementen, d.h. Konstanten und Variablen, auf, die in Abschnitt 3 beschrieben

sind. Diese werden durch Operatoren miteinander verbunden und bilden so Ausdrücke, die ihrerseits Bestandteile von Anweisungen darstellen. Die Erläuterung der FORTRAN-Anweisungen ist ab Abschnitt 4 enthalten.

1.2. Charakteristische Merkmale

1.2.1. FORTRAN-Anweisungen

Es folgt eine Aufstellung der Anweisungen, aus denen sich die FORTRAN IV-Sprache zusammensetzt. Einige dieser Anweisungen bewirken, daß der Compiler Befehle für das Maschinenprogramm erzeugt. Andere Anweisungen beschreiben dem Compiler die Struktur und Anordnung der Daten, die von dem Maschinenprogramm verarbeitet werden sollen, oder die Organisation des Primärprogramms selbst.

1.2.2. Minimalausstattung

Die Minimalausstattung, die für Anwendung des FORTRAN IV im Band-Betriebssystem benötigt wird, ist folgende:

1 Zentraleinheit (65K Bytes)
5 Magnetbandgeräte
1 Bedienungsblattschreiber
1 Eingabegerät - Magnetbandgerät, Lochkartenleser oder optischer Belegleser
1 Protokollierungsgerät - Magnetband gerät oder Schnelldrucker

Ein Lochkartenstanzer oder ein Magnetbandgerät kommt hinzu, wenn gestanzte Ausgabe gewünscht wird.

2. Programmvorbereitung

Das Primärprogramm wird in das unten gezeigte FORTRAN-Programmformular eingetragen. Überschreitet eine Anweisung die Länge einer Zeile, so kann sie auf bis zu 19 zusätzlichen Zeilen fortgesetzt werden. Es ist jedoch nicht mehr als eine Anweisung pro Zeile zugelassen. Die Zeilen können auch zur Eintragung von Bemerkungen verwendet werden. Derartige Zeilen werden vom Compiler ignoriert.

Jede Zeile des FORTRAN-Programmformulars enthält eine Folge von max. 72 Zeichen, die dem folgenden Vorrat entnommen sind:

Zeichen	Zeichen	Zeichen	Zeichen	Name
0	C	Ø	=	Gleichheitszeichen
1	D	P	+	Plus
2	E	Q	-	Minus
3	F	R	*	Stern
4	G	S	/	Schrägstrich
5	H	T	.	Dezimalpunkt
6	I	U	,	Komma
7	J	V	(	Klammer auf
8	K	W	)	Klammer zu
9	L	X	$	Dollarzeichen
A	M	Y	&	Und
B	N	Z	'	Apostroph

Zusätzlich zu diesen Zeichen ist noch das Zeichen Zwischenraum - im Text durch "_" (Unterstrich) gekennzeichnet - erlaubt.

Alle anderen Zeichen des aus 256 Zeichen bestehenden EBCDIC können nur in Literalen verwandt werden.

Der Lochkartencode ist dem Anhang 1 zu entnehmen.

Jede FORTRAN-Anweisung kann durch eine Nummer gekennzeichnet werden, um innerhalb eines Programms die Bezugnahme auf die Anweisung zu ermöglichen. Die einer Anweisung zugeordnete Nummer kann jede ganze Zahl zwischen 1 und 99999 sein, die beiden Grenzen eingeschlossen. Führende Nullen tragen zum Wert dieser Nummer nicht bei. Die Anweisungsnummern 0001 und 1 sind also gleich. Zwei Anweisungen dürfen nicht die gleichen Anweisungsnummern haben. Die Reihenfolge der Anweisungsnummern ist beliebig.

Die Zeichenpositionen (Spalten) auf dem Programmformular sind von links nach rechts, von 1 anfangend, fortlaufend numeriert. Bei Benutzung des Formulars müssen folgende Regeln beachtet werden:

(1) Eine Zeile mit Bemerkungen muß in der ersten Spalte ein "C" enthalten.

(2) Die Spalten 1 bis 5 der ersten Zeile einer Anweisung enthalten gegebenenfalls die Nummer der Anweisung.

(3) Spalte 6 der ersten Zeile einer Anweisung muß leer sein oder eine Null enthalten. Eventuell folgende Zeilen derselben Anweisung müssen in Spalte 6 ein Zeichen enthalten, das ungleich Null und Zwischenraum sein muß.

(4) Die Anweisung selbst wird in die Spalten 7 bis 72 eingetragen. Jede Anweisung darf aus höchstens 1320 Zeichen, Zwischenräume eingeschlossen, bestehen.

(5) Das Zeichen Zwischenraum hat nur in Spalte 6 des Programmformulars sowie innerhalb von Literalen Bedeutung. Es wird im übrigen vom Compiler ignoriert. Zwischenräume können daher nach Belieben zur Verbesserung der Lesbarkeit eines Programms verwendet werden.

(6) Die Spalten 73 bis 80 werden vom Compiler nicht verarbeitet. Sie werden gewöhnlich für die Eintragung einer Programmidentifikation und von Folgenummern verwendet. Sie werden zusammen mit dem Primärprogramm ausgedruckt.

Jede Zeile des Programmformulars wird in eine Lochkarte gelocht. Der Aufbau der Programmkarten ist mit dem Aufbau der Zeilen auf dem Programmformular identisch.

3. Sprachelemente

Die Elemente der FORTRAN-Anweisungen sind Konstanten, Variable und Felder von Variablen. Eine Konstante im Primärprogramm wird vom Compiler als Konstante im Maschinenprogramm generiert. Einer Variablen oder einem Feld ordnet der Compiler einen Speicherbereich innerhalb des Maschinenprogramms zu. Diese Sprachelemente werden im folgenden einzeln besprochen.

3.1. Konstanten

Eine Konstante ist eine feste, unveränderliche Größe. Es gibt drei Klassen von Konstanten - solche, die als Zahlen (numerische Konstante), solche, die als Wahrheitswerte (Boolesche Konstanten) und solche, die als Literale auftreten.

Numerische Konstanten sind ganze, reelle oder komplexe Zahlen. Aufgrund des Ergebnisses der Auswertung konstanter Ausdrücke durch den Compiler unter Berücksichtigung ihres Verwendungszweckes besteht jedoch die Möglichkeit, daß Konstanten in anderer Form in das Maschinenprogramm eingesetzt werden als vom Programmierer vorgesehen worden war, wenn dadurch die Leistungsfähigkeit des Maschinenprogramms erhöht wird.

Boolesche Konstanten sind .TRUE. (wahr) oder .FALSE. (falsch). Literale sind Zeichenreihen von alphanumerischen und/oder Sonderzeichen. Die verschiedenen Konstanten werden im folgenden ausführlicher besprochen.

3.1.1. Ganzzahlige Konstanten

Definition: Ganzzahlige Konstante - eine ganze Zahl, die ohne Dezimalpunkt angegeben ist. Sie belegt je nach Größe entweder 2 oder 4 Speicherplätze.

Höchstwert ist 2147483647, d.h. $2^{31} - 1$. Liegt der Wert der Konstante im Bereich 0 bis 32767, so belegt sie zwei, andernfalls vier Speicherplätze.

Eine ganzzahlige Konstante kann positiv, Null oder negativ sein. Fehlt das Vorzeichen, wird sie als positiv angenommen. Sie darf nicht größer als der angegebene Höchstwert sein und keine Kommas enthalten.

Beispiele

Als ganzzahlige Konstante zulässig:

0
91
173
-2147483647
-12

Als ganzzahlige Konstante unzulässig:

0.0	-	enthält einen Dezimalpunkt
27.	-	enthält einen Dezimalpunkt
3145903612	-	überschreitet den zugelassenen Bereich
5,396	-	enthält ein Komma.

Ganzzahlige Konstanten werden in den Anweisungen des Primärprogramms für Indizes, Exponenten, Parameter - z.B. in der Schleifenanweisung - sowie als arithmetische Operanden verwendet. In dem folgenden Beispiel sind 3, 5 und 2 ganzzahlige Konstanten:

```
A = Y(3)/5 + X**2
```

3.1.2. Reelle Konstanten

Definition: Reelle Konstante - eine Zahl mit einem Dezimalpunkt, der bei Bedarf ein Dezimalexponent folgt, oder eine Zahl ohne Dezimalpunkt, der ein Dezimalexponent folgt. Der Dezimalexponent wird durch den Buchstaben E oder D, gefolgt von einer ein- oder zweistelligen ganzen Zahl mit oder ohne Vorzeichen dargestellt. Eine reelle Konstante belegt 4 oder 8 Speicherplätze gemäß folgenden Regeln:

1. Wenn der Zahl ein durch E gekennzeichneter Dezimalexponent folgt oder wenn sie 1 bis 7 gültige Dezimalziffern enthält und ihr kein Exponent folgt, dann belegt sie 4 Speicherplätze.

2. Wenn der Zahl ein durch D gekennzeichneter Dezimalexponent folgt oder wenn sie 8 bis 16 gültige Dezimalziffern enthält und ihr kein Exponent folgt, belegt sie 8 Speicherplätze. Eine solche Konstante wird als Konstante mit doppelter Genauigkeit bezeichnet.
   Größenbereich ist 0 und 16^{-65} bis 16^{63} (d.h. etwa 10^{75}).

Die von der FORTRAN-Sprache des Siemens-Systems 4004 benutzten Gleitkommazahlen werden intern in dem Format dargestellt, das von den verdrahteten Gleitkommabefehlen vorausgesetzt wird. Der Exponent belegt 7 Bits und gibt die Potenz an, zu der 16 erhoben werden soll. Der Bereich des Exponenten ist -64 bis +63, entsprechend den 2^7 Binärwerten. Der angenommene Radixpunkt der Mantisse liegt unmittelbar neben der höchsten Stelle. Ist die Mantisse Null, ist der Wert der Gleitkommazahl Null. Ist die Mantisse von Null verschieden, liegt die Gleitkommazahl in folgendem

Größenbereich:

$$16^{-65} \leq X < 16^{63}$$

Eine reelle Konstante kann positiv, Null oder negativ sein. Fehlt das Vorzeichen, so wird sie als positiv angenommen. Sie muß im zulässigen Größenbereich liegen und darf keine Kommas enthalten. Der Dezimalexponent E oder D erlaubt die Darstellung einer reellen Konstante als Produkt einer reellen Konstante mit einer Potenz von 10.

Beispiele

Als reelle Konstanten (4 Speicherplätze) zulässig:

```
+0.
-999.9999
0.0
5764.1
7.0E+0          (d.h. 7.0 x 10^0 = 7.0)
19761.253E+1    (d.h. 19761.253 x 10^1 = 197612.53)
7.E3
7.0E3           (d.h. 7.0 x 10^3 = 7000.0)
7.0E03
7.0E+03
7E-03           (d.h. 7 x 10^-3 = .007)
```

Als reelle Konstanten (8 Speicherplätze) zulässig:

```
21.98753829457168
1.0000000
7.9D3
7.9D03          } (d.h. 7.9 x 10^3 = 7900.0)
7.9D+03
7.9D+3
7.9D-03         (d.h. 7.9 x 10^-3 = .0079)
79D-1           (d.h. 79 x 10^-1 = 7.9)
0D0             (d.h. 0 x 10^0 = 0.0)
```

Als reelle Konstanten unzulässig:

0	(Dezimalpunkt fehlt)
3,471.1	(enthält ein Komma)
1.E	(dem E folgt keine ein- oder zweistellige ganzzahlige Konstante. Es ist zu beachten, daß dies nicht als 1.0×10^0 interpretiert wird)
7.9D	(dem D folgt keine ein- oder zweistellige ganzzahlige Konstante)
1.2E+113	(E folgt eine dreistellige ganzzahlige Konstante)
21.3D90	(Wert überschreitet die zulässige Größe, d.h. $21.3 \times 10^{90} > 16^{63}$)
23.5E+97	(Wert überschreitet die zulässige Größe, d.h. $23.5 \times 10^{97} > 16^{63}$)

3.1.3. Komplexe Konstanten

Definition: Komplexe Konstante - ein geordnetes Paar von mit oder ohne Vorzeichen versehenen Konstanten. Beide können reell oder ganzzahlig sein. Sie sind durch ein Komma voneinander getrennt und in Klammern eingeschlossen. Jede der beiden Konstanten wird intern als reelle Konstante dargestellt. Die Regeln für die Speicherbelegung für jede dieser beiden Konstanten entsprechen den für reelle Konstanten gegebenen. Wenn nur eine der beiden Konstanten 8 Speicherplätze belegt, tut es auch die andere. Die komplexe Konstante belegt in diesem Fall 16 Speicherplätze, andernfalls 8 Speicherplätze.

Größenbereich ist 0 und 16^{-65} bis 16^{63} (d.h. etwa 10^{75}) für jede der beiden Konstanten des Paares .

Die Konstanten innerhalb einer komplexen Konstante können positiv, Null oder negativ sein. (Fehlt das Vorzeichen einer Konstanten, wird angenommen, daß sie positiv sei. Sie müssen im zulässigen Größenbereich liegen. Die erste reelle Konstante in einer komplexen Konstante stellt den reellen Teil, die zweite den imaginären Teil der komplexen Zahl dar.

Beispiele

Als komplexe Konstanten zulässig:

(3,2)	(hat den Wert 3.0+2.0i)	alle belegen 8 Speicherplätze
(3,-1.86)	(hat den Wert 3.0-1.86i)	
(-5.0E+03,.0212E+02)	(hat den Wert -5000.+2.12i)	
(4.0E+03,.16E+02	(hat den Wert 4000.+16.0i)	
(2.1,0)	(hat den Wert 2.1+0.0i)	
(4.7E+2,1.973)	(hat den Wert 470.+1.973i)	alle belegen 16 Speicherplätze
(4.7E+2,1.9736148)	(hat den Wert 470.+1.9736148i)	
(0,0D0)	(hat den Wert 0.0+0.0i)	
(0D0,0)		

wo i = $\sqrt{-1}$

Als komplexe Konstanten unzulässig:

(1.2E113,279.3)	(der reelle Teil enthält einen ungültigen Dezimalexponent)
1.2E13,279.3	(die Klammern fehlen)
(12.5 - 1D0)	(das Komma fehlt)

3.1.4. Boolesche Konstanten

Definition: Boolesche Konstante - Es gibt zwei Boolesche Konstanten:

.TRUE. (wahr)
.FALSE. (falsch)

Einer Booleschen Konstanten muß ein Punkt vorausgehen und ein Punkt folgen. Die Booleschen Konstanten .TRUE. und .FALSE. geben an, daß der Wert einer Booleschen Variablen, die sie ersetzen oder eines Ausdruckes, dem sie zugeordnet sind, wahr oder falsch ist (siehe Abschnitt 4.3. "Boolesche Ausdrücke").

3.1.5. Literale

Definition: Literal - eine Reihe beliebiger Zeichen, einschließlich dem Zeichen Zwischenraum, die in einer der beiden folgenden Formen geschrieben ist:

1. Die Zeichen sind in Apostrophe eingeschlossen. Soll in einer solchen Reihe ein Apostroph dargestellt werden, so müssen zwei Apostrophe geschrieben werden.

2. Die Form nH; n ist eine ganzzahlige Konstante, die größer als Null ist. Dem Ausdruck nH folgen genau n Zeichen. Um in dieser Zeichenreihe einen Apostroph darzustellen, muß nur ein einziger Apostroph geschrieben werden.

Die Anzahl der Zeichen in der Reihe, einschließlich Zwischenraum, darf 255 nicht übersteigen.

Beispiele

```
'DATEN'
'EINGABE-/AUSGABE-BEREICH NR. 2'
'X-KOORDINATE  Y-COORDINATE  Z-KOORDINATE'
'GEHT"S GUT'      } (Diese beiden Darstellungen
10HGEHT'S GUT     }  sind gleichwertig.)
'3.14'
```

3.1.6. Variablen

Eine Variable ist in FORTRAN die symbolische Darstellung einer Größe, die verschiedene Werte annehmen kann. Der Wert einer Variablen kann sich während der Ausführung des Programms ändern. Dies geschieht durch Ausführung einer Anweisung, die der Variablen einen Wert zuweist, durch Verwendung als Argument eines Unterprogramms, durch ihren Gebrauch als Parameter einer Schleifenanweisung (einschließlich implizierter Schleifenanweisungen bei Ein-Ausgabe-Listen) oder durch Einlesen eines neuen Wertes von einem Eingabemedium.

Beispielsweise sind in der Anweisung

```
A = 5.0 + B
```

sowohl A als auch B Variablen. Der Wert B kann durch eine vorhergehende Anweisung von einer Lochkarte gelesen worden sein. Ein neuer Wert für A wird immer dann errechnet und gespeichert, wenn die vorstehende Anweisung ausgeführt wird.

3.1.6.1. Variablennamen

Definition: Variablenname - 1 bis 6 alphanumerische Zeichen (d.h. die numerischen Zeichen 0 - 9 oder die alphabetischen Zeichen A - Z und $), von denen das erste alphabetisch sein muß.

Ein Variablenname darf kein Sonderzeichen (siehe Anhang 1) enthalten. Variablennamen sind Symbole, die der Unterscheidung der Variablen voneinander dienen. Ein Name kann innerhalb eines Programms auf eine und nur eine Art verwendet werden (d.h. der Name einer Variablen und der eines Unterprogramms dürfen innerhalb eines Primärprogramms nicht gleich sein).

Die Verwendung sinnvoller Variablennamen kann als Hilfe bei der Dokumentation eines Programms dienen. Das hat den Vorteil, daß ein Programmierer, der das betreffende Programm nicht geschrieben hat, sich das Programm ansehen und seine Funktion verstehen kann. Wenn z.B. die Entfernung, die ein Kraftfahrzeug in einer gegebenen Zeit mit einer gegebenen Geschwindigkeit durchfahren hat, berechnet werden soll, dann könnte hierzu folgende Anweisung geschrieben werden:

```
X = Y * Z
```

wo * Multiplikation bedeutet. Es wäre jedoch beim Lesen des Programms aussagefähiger, wenn der Programmierer geschrieben hätte

```
ENTF = GESCHW*ZEIT
```

Beispiele

Als Variablennamen zulässig:

JOHN
B292
VAN
RATE
L17NOY
SQ704

Als Variablennamen unzulässig:

B292704	(enthält mehr als sechs Zeichen)
4ARRAY	(erstes Zeichen ist nicht alphabetisch)
SI.X	(enthält ein Sonderzeichen)

3.1.6.2. Variablentypen und Längenangaben

Der Typ einer Variablen entspricht dem Typ der Daten, die der Variablen zugewiesen werden. Eine ganzzahlige Variable stellt ganzzahlige Daten, eine reelle Variable reelle Daten dar usw.

Jedem Variablentyp ist eine standardmäßige sowie eine wählbare Länge, d.h. die Anzahl der für die Variable bereitzustellenden Speicherplätze, zugeordnet. Die folgende Aufstellung enthält jeden Variablentyp mit der standardmäßigen und wählbaren Länge:

Variablentyp	Standardlänge	Wählbare Länge
Ganzzahlig	4	2
Reell	4	8
Komplex	8	16
Boolesch	4	1

Der Programmierer kann den Variablentyp auf dreierlei Weise festlegen:

1. durch die Konvention der FORTRAN-Sprache,
2. durch die Spezifikationsanweisung IMPLICIT,
3. durch explicite Spezifikationsanweisungen.

Die wählbare Länge einer Variablen kann nur durch die Spezifikationsanweisung IMPLICIT oder durch explicite Spezifikationsanweisungen vereinbart werden. Wenn in diesen Anweisungen keine Längenangabe erfolgt, wird die Standardlänge angenommen (siehe Abschnitt 7 "Spezifikationsanweisungen").

3.1.6.3. Typenvereinbarung nach der FORTRAN-Konvention

Die FORTRAN-Konvention vereinbart Variable als ganzzahlig oder reell in folgender Weise:

1. Wenn das erste Zeichen des Variablennamens I, J, K, L, M oder N ist, ist die Variable ganzzahlig und hat die Standardlänge.
2. Ist das erste Zeichen des Variablennamens irgendein anderes alphabetisches Zeichen, ist die Variable reell und hat die Standardlänge.

Diese Vereinbarung ist die traditionelle FORTRAN-Methode zur Festlegung des Types von Variablen als ganzzahlig oder reell. Alle Beispiele, die in dieser Veröffentlichung folgen, gehen, wenn nicht ausdrücklich anders vermerkt, von dieser Konvention aus.

3.1.6.4. Typenvereinbarung durch die Spezifikationsanweisung IMPLICIT

Mit Hilfe dieser Anweisung kann der Programmierer den Typ von Variablen auf dieselbe Weise definieren, auf die es gemäß der FORTRAN-Konvention geschieht. Das bedeutet, daß in beiden Fällen der Typ durch das erste Zeichen des Variablennamen festgelegt ist. Mit der Anweisung IMPLICIT hat der Programmierer jedoch die Möglichkeit zu bestimmen, welche Anfangsbuchstaben welchen Variablentypen zugeordnet sein sollen. Darüber hinaus ist die Anweisung IMPLICIT anwendbar auf alle Typen von Variablen - auf ganzzahlige, reelle, komplexe und Boolesche Variable.

Die Anweisung IMPLICIT setzt die durch die FORTRAN-Konvention getroffene Festlegung des Variablentyps außer Kraft. Wenn z.B. die Anweisung IMPLICIT spezifiziert, daß die Variablen, die mit den Buchstaben A bis M beginnen, reell und die Variablen, die mit den Buchstaben N bis Y beginnen, ganzzahlig sind, dann wird die Variable INDEX - die nach der FORTRAN-Konvention ganzzahlig wäre, als reelle Variable behandelt. Es ist zu beachten, daß die Variablen, die mit dem Buchstaben Z oder mit $ beginnen, durch die FORTRAN-Konvention als reelle Variable vereinbart werden.

Die Anweisung IMPLICIT wird in dem Abschnitt 7 "Spezifikationsanweisungen" näher beschrieben.

3.1.6.5. Typenvereinbarung durch explicite Spezifikationsanweisungen

Explicite Spezifikationsanweisungen (INEGER, REAL, COMPLEX und LOGICAL) unterscheiden sich von den ersten beiden Arten zur Vereinbarung von Typen einer Variablen dadurch, daß sie den Typ einer bestimmten Variablen durch ihren Namen erklären und nicht für eine ganze Gruppe von Variablen den Typ durch das erste Zeichen des Namens festlegen.

Beispielsweise sei angenommen:

1. Eine Spezifikationsanweisung IMPLICIT habe die FORTRAN-Konvention für Variablen, die mit dem Buchstaben I beginnen, außer Kraft gesetzt und diese als reell erklärt.
2. Eine folgende explicite Spezifikationsanweisung habe vereinbart, daß die Variable mit dem Namen INDEX komplex ist.

In diesem Fall ist die Variable INDEX komplex und alle anderen Variablen, die mit dem Buchstaben I beginnen, sind reell. Es ist zu beachten, daß Variablen, die mit den Buchstaben J bis N beginnen, durch die FORTRAN-Konvention als ganzzahlig vereinbart sind.

Die expliciten Spezifikationsanweisungen werden in größerer Ausführlichkeit im Abschnitt 7 "Spezifikationsanweisungen" beschrieben.

3.1.7. Felder

In FORTRAN ist ein Feld ein Satz von Variablen, die durch einen einzigen Variablennamen gekennzeichnet sind. Eine bestimmte dieser Variablen kann durch ihre Position innerhalb des Feldes angesprochen werden (beispielsweise die erste Variable, die dritte Variable, die siebente Variable usw.). Betrachtet sei das Feld mit dem Namen NORM, das aus fünf Variablen besteht, die die folgenden Werte darstellen:

273, 41, 8976, 59 und 2.

NORM (1) ist die Darstellung von 273,

NORM (2) ist die Darstellung von 41,

NORM (3) ist die Darstellung von 8976,

NORM (4) ist die Darstellung von 59,

NORM (5) ist die Darstellung von 2.

Jede Variable in diesem Feld besteht aus dem Namen des Feldes (d.h. NORM), dem eine in Klammern eingeschlossene Zahl folgt, die Index genannt wird. Die Variablen, aus denen das Feld besteht, werden indizierte Variable genannt. Infolgedessen hat die indizierte Variable NORM (1) den Wert 273, die indizierte Variable NORM (2) hat den Wert 41 usw. In einer Anweisung kann ein bestimmtes Element innerhalb des Feldes wie z.B. NORM (4) oder ein beliebiges Element wie NORM(J) angesprochen werden. Im letzteren Fall bezieht sich NORM (J) auf das "J-te" Element des Feldes; J ist eine ganzzahlige Variable, die einen der Werte 1, 2, 3, 4 oder 5 annehmen kann. Vor Ausführung der Anweisung, die NORM (J) enthält, muß der Variablen J einer der zulässigen Werte zugewiesen worden sein.

Allgemeine Form	Elemente
Indizes - in einem der sieben Formate: v c v+c v-c d*v d*v+c d*v-c	v stellt eine vorzeichenlose, nicht indizierte, ganzzahlige Variable dar. c und d stellen vorzeichenlose, ganzzahlige Konstanten dar.

Unabhängig davon, welche Indexform benutzt wird, sollte der ausgerechnete Wert immer größer als Null sein. Wenn z.B. eine indizierte Variable V (I-2) angesprochen wird, sollte der Wert von I größer als 2 sein.

Beispiele

```
FELD (INDEX)
NORM (19)
MATRIX (1-5)
A (5*L)
W (4*M+3)
```

Es können Felder mit bis zu 7 Dimensionen verwendet werden. Infolgedessen kann eine Variable bis zu 7 Indizes haben, die voneinander durch Kommas getrennt sind. In den entsprechenden Feldern sind daher folgende indizierte Variablen zulässig:

Feldname	Anzahl der Dimensionen	Indizierte Variable
A	4	A(5,100,J,K+2)
TAB	7	TAB(1,1,1,1,1,1,1)
B	6	B(I,J,K,L,M,N)
MATRIX	3	MATRIX(I+2,6*JOB-3,KFRAN)

Betrachtet sei das folgende zweidimensionale Feld LISTE, dessen erster Index von 1 bis 5, der zweite von 1 bis 3 läuft:

	Spalte 1	Spalte 2	Spalte 3
Zeile 1	82	4	7
Zeile 2	12	13	14
Zeile 3	91	1	31
Zeile 4	24	16	10
Zeile 5	2	8	2

Angenommen, es soll zu der Zahl in Zeile 2, Spalte 3 gegriffen werden. Dies würde wie folgt ausgedrückt:

```
LISTE (2,3)
```

LISTE (2,3) hat also den Wert 14 und LISTE (4,1) hat den Wert 24.

Nach der üblichen mathematischen Notation wäre $LISTE_{i,j}$ zu schreiben, um ein beliebiges Element des Feldes LISTE darzustellen. In FORTRAN wird geschrieben LISTE (I,J) mit I=1,2,3,4 oder 5 und J=1,2 oder 3.

Als ein weiteres Beispiel sei das vierdimensionale Feld mit dem Namen KOSTEN betrachtet. Dies Feld werde dazu benutzt, die Prämien für Antragsteller auf eine Lebensversicherung in Abhängigkeit von (1) Alter, (2) Geschlecht, (3) Gesundheit und (4) der Höhe der Versicherungssumme zu speichern. Für jede Angabe könnte eine Schlüsselzahl entwickelt werden, wobei IALT das Alter, ISEX das Geschlecht,

IGES den Gesundheitszustand und IWERT die gewünschte Versicherungssumme darstellen (s. Tabelle 1).

Tabelle 1 Schlüssel für die Ermittlung von Versicherungsprämien

Alter		Geschlecht	
Alter in Jahren	Schlüssel	Geschlecht	Schlüssel
1-5	IALT=1	männlich	ISEX=1
6-10	IALT=2	weiblich	ISEX=2
-	-		
-	-		
-	-		
96-100	ALT=20		
Gesundheit		Versicherungssumme	
Zustand	Schlüssel	Betrag (DM)	Schlüssel
schlecht	IGES=1	1.000	IWERT=1
mäßig	IGES=2	2.000	IWERT=2
gut	IGES=3	3.000	IWERT=3
ausgezeichnet	IGES=4	5.000	IWERT=4
		10.000	IWERT=5
		25.000	IWERT=6
		50.000	IWERT=7
		100.000	IWERT=8

Angenommen, ein Antragsteller ist 14 Jahre alt, männlich, bei guter Gesundheit und wünscht eine Versicherung über DM 25.000,-- abzuschließen. Gemäß Tabelle 1 könnten diese Angaben durch folgende Schlüsselzahlen dargestellt werden:

IALT=3 (11-15 Jahre alt)
ISEX=1 (männlich)
IGES=3 (Gesundheitszustand gut)
IWERT=6 (Versicherungssumme DM 25.000,--)

KOSTEN(3, 1, 3, 6) ist dann die zugehörige Prämie. Es ist zu beachten, daß IALT von 1 bis 20, ISEX von 1 bis 2, IGES von 1 bis 4 und IWERT von 1 bis 8 laufen kann.

(Die Anzahl indizierter Variablen in dem Feld KOSTEN ist gleich der Anzahl von Kombinationen, die aus den verschiedenen Alters-, Geschlechts-, Gesundheits- und Betragsangaben gebildet werden können - insgesamt also 20 x 2 x 4 x 8 = 1280. Folglich können bis zu 1280 verschiedene Prämien in dem Feld mit dem Namen KOSTEN gespeichert werden.)

3.7.1.1. Größe eines Feldes

Die Größe eines Feldes wird durch die Anzahl der Dimensionen des Feldes und dem größten Wert, den jeder Index annehmen kann, bestimmt. Diese Informationen müssen für ein Feld vor bereitgestellt werden, bevor sich eine Anweisung auf das Feld bezieht, damit ausreichend Speicherplatz vorgesehen werden kann. Diese Informationen werden dem Compiler durch eine Feldanweisung (DIMENSION), eine Speicherblockanweisung (COMMON) oder durch eine explicite Spezifikationsanweisung (INTEGER, REAL, COMPLEX und LOGICAL) übermittelt. Diese Anweisungen werden ausführlich in dem Abschnitt "Spezifikationsanweisungen" besprochen.

3.7.1.2. Anordnung von Feldern im Speicher

Ein Feld wird in aufsteigenden Speicherplätzen gespeichert, wobei der Wert seines ersten Index am schnellsten und der seines letzten Index am langsamsten zunimmt.

Beispiele

Das eindimensionale Feld mit dem Namen A, dessen Index von 1 bis 5 läuft, wird im Speicher wie folgt aufgebaut:

A(1) A(2) A(3) A(4) A(5)

Das zweidimensionale Feld mit dem Namen B, dessen erster Index von 1 bis 5 und dessen zweiter Index von 1 bis 3 läuft, wird in aufsteigenden Speicherplätzen in folgender Reihenfolge aufgebaut:

```
B(1,1)  B(2,1)  B(3,1)  B(4,1)  B(5,1)
B(1,2)  B(2,2)  B(3,2)  B(4,2)  B(5,2)
B(1,3)  B(2,3)  B(3,3)  B(4,3)  B(5,3)
```

Es ist zu beachten, daß im Speicher die Variable B(1,2) bzw. B(1,3) der Variablen B(5,1) bzw. B(5,2) folgt.

Die folgende Aufstellung gibt die Anordnung eines dreidimensionalen Feldes mit dem Namen C an, dessen erster Index von 1 bis 3, dessen zweiter Index von 1 bis 2 und dessen dritter Index von 1 bis 3 läuft:

```
C(1,1,1)  C(2,1,1)  C(3,1,1)  C(1,2,1)  C(2,2,1)  C(3,2,1)
C(1,1,2)  C(2,1,2)  C(3,1,2)  C(1,2,2)  C(2,2,2)  C(3,2,2)
C(1,1,3)  C(2,1,3)  C(3,1,3)  C(1,2,3)  C(2,2,3)  C(3,2,3)
```

Es ist zu beachten, daß im Speicher die Variable C(1,1,2) bzw. C(1,1,3) der Variablen C(3,2,1) bzw. C(3,2,2) folgt.

4. Ausdrücke und Ergibtanweisungen

4.1. Ausdrücke

In ihrer einfachsten Form bestehen Ausdrücke aus einer einzigen Konstanten oder Variablen. Sie können auch eine Rechenvorschrift darstellen oder die Beziehung zwischen zwei oder mehreren Konstanten und/oder Variablen wiedergeben. Ausdrücke treten in arithmetischen und Booleschen Ergibtanweisungen und in bestimmten Steueranweisungen auf.

FORTRAN enthält zwei Arten von Ausdrücken, nämlich arithmetische und Boolesche. Der Wert eines arithmetischen Ausdruckes ist immer eine Zahl vom Typ ganzzahlig, reell oder komplex. Die Auswertung eines Booleschen Ausdrucks liefert dagegen immer einen Wahrheitswert, nämlich entweder .TRUE. (=wahr) oder .FALSE. (=falsch).

4.2. Arithmetische Ausdrücke

Der einfachste arithmetische Ausdruck ist eine einzige Konstante, Variable, indizierte Variable oder ein Funktionsaufruf (s. Abschnitt 8.4.2. "Funktionen"). Er ist vom Typ ganzzahlig, reell oder komplex. Ist die Konstante, Variable, indizierte Variable oder die aufgerufene Funktion vom Typ ganzzahlig, so ist auch der Ausdruck vom Typ ganzzahlig. Ist sie vom Typ reell, ist auch der Ausdruck vom Typ reell usw.

Beispiele für einfache arithmetische Ausdrücke:

Ausdruck	Art der Größe	Typ
3	Ganzzahlige Konstante	Ganzzahlig
I	Ganzzahlige Variable	Ganzzahlig
3.0	Reelle Konstante	Reell
A	Reelle Variable	Reell
3.14D3	Reelle Konstante	Reell, belegt 8 Speicherplätze

Ausdruck	Art der Größe	Typ
B(2*I)	Reelle Variable (laut Definition durch eine Typanweisung)	Reell, belegt 4 Speicherplätze
(2.0,5.7)	Komplexe Konstante	Komplex
C	Komplexe Variable (laut Definition durch eine Typanweisung)	Komplex

In dem Ausdruck B(2*I) wirkt sich der Index (2*I), der immer eine ganze Zahl darstellt, nicht auf den Typ des Ausdrucks aus, d.h., daß der Typ des Ausdrucks einzig und allein durch den Typ der Konstanten, Variablen, indizierten Variablen oder der aufgerufenen Funktion bestimmt wird, die in dem Ausdruck auftreten.

Kompliziertere arithmetische Ausdrücke, die zwei oder mehrere Konstanten und/oder Variablen enthalten, können unter Verwendung von arithmetischen Operatoren gebildet werden, die die auszuführenden Berechnungen angeben. Die arithmetischen Operatoren sind folgende:

Arithmetischer Operator	Erklärung
**	Potenzierung
*	Multiplikation
/	Division
+	Addition
-	Subtraktion

4.2.1. Regeln für den Aufbau von arithmetischen Ausdrücken

Es folgen die Regeln für den Aufbau arithmetischer Ausdrücke, die arithmetische Operatoren enthalten:

1. Alle gewünschten Berechnungen müssen explicit angegeben werden. Tritt in einem Ausdruck mehr als eine Konstante, Variable, indizierte Variable oder ein Funktionsname (s. Abschnitt 8.4.2. "Funktionen") auf, so müssen diese durch arithmetische Operatoren voneinander getrennt werden. Beispielsweise wird die Multiplikation der Variablen A und B nicht ausgeführt, wenn sie wie folgt geschrieben wird:

   AxB oder AB oder A.B

2. Es dürfen nicht zwei arithmetische Operatoren unmittelbar aufeinander folgen. Beispielsweise sind die folgenden Ausdrücke ungültig:

   A*/B oder A*-B

   Wenn jedoch in dem Ausdruck A*-B das Zeichen "-" als Minuszeichen und nicht als arithmetischer Operator für Subtraktion gedacht ist, kann der Ausdruck wie folgt geschrieben werden:

   A*(-B)

   Es wird dann zuerst -B ermittelt und A mit diesem Wert multipliziert. (Für den weiteren Gebrauch von Klammern s. Regel 5.)

3. Typ und Länge eines arithmetischen Ausdruckes werden durch die Typen- und Längenvereinbarung der Variablen des Ausdruckes bestimmt. Tabelle 2 zeigt, wie sie aus den Typen und Längen der durch +, -, *, /, ** verknüpften Variablen bestimmt werden.

   Der Tabelle 2 ist zu entnehmen, daß es eine Rangordnung von Typen- und Längenvereinbarungen gibt (s. hierzu auch den Abschnitt 7 "Spezifikationsanweisungen"), die Typ und Länge eines Ausdruckes

festlegt. Werden z.B. komplexe Daten mit einer Länge von 16 Speicherplätzen mit Konstanten und Variablen eines beliebigen anderen Typs verknüpft, so ergeben sich komplexe Daten mit einer Länge von 16 Speicherplätzen. Ganzzahlige Ausdrücke haben immer die Länge 4, die bei Zuweisung zu Variablen der Länge 2 oder zu Unterprogrammen, die Argumente der Länge 2 benötigen, abgebrochen wird.

Tabelle 2 Bestimmung des Typs und der Länge eines arithmetischen Ausdrucks aus dem Typ und der Länge der in ihm enthaltenen Variablen

** +-*/	GANZZAHLIG (2)	GANZZAHLIG (4)	REELL (4)	REELL (8)	KOMPLEX (8)	KOMPLEX (16)
GANZZAHLIG (2)	Ganzzahlig (2)	Ganzzahlig (4)	Reell (4)	Reell (8)	Komplex (8)	Komplex (16)
GANZZAHLIG (4)	Ganzzahlig (4)	Ganzzahlig (4)	Reell (4)	Reell (8)	Komplex (8)	Komplex (16)
REELL (4)	Reell (4)	Reell (4)	Reell (4)	Reell (8)	Komplex (8)	Komplex (16)
REELL (8)	Reell (8)	Reell (8)	Reell (8)	Reell (8)	Komplex (16)	Komplex (16)
KOMPLEX (8)	Komplex (8)	Komplex (8)	Komplex (8)	Komplex (16)	Komplex (16)	Komplex (16)
KOMPLEX (16)	Komplex (16)	Komplex (16)	Komplex (16)	Komplex (16)	Komplex (16)	Komplex (16)

Angenommen, es seien Variablen hinsichtlich ihres Typs und ihrer Länge in folgender Weise festgelegt:

Variablenname	Typ	Länge
FORM, E	Reelle Variable	4,8
A, I, F	Ganzzahlige Variable	4,2,2
C, D	Komplexe Variable	16,8

Dann zeigen die folgenden Beispiele, wie Konstanten und Variablen verschiedenen Typs mit Hilfe der arithmetischen Operatoren +,-,/,*,** miteinander verknüpft werden können:

Ausdruck	Typ	Länge
FORM*5	Reell	4
A+3	Ganzzahlig	4
C+2.9D10	Komplex	16
E/F+19	Reell	8
C-18.7E05	Komplex	16
A/I-D	Komplex	8
FORM**C	Komplex	16
FORM**(A+2)	Reell	4
C**A	Komplex	16
I**F	Ganzzahlig	4
A**E	Reell	8
A**D	Komplex	8

4. Reihenfolge der Berechnung: Fehlen Klammern oder ist der gesamte arithmetische Ausdruck in einem einzigen Klammernpaar eingeschlossen, so werden die Operationen in folgender Reihenfolge ausgeführt:

Rangordnung	Operation
1	Auswertung von Funktionen (siehe Abschnitt 8.4.2. "Funktionen")
2	Potenzierung (**)
3	Multiplikation und Division (* und /)
4	Addition und Subtraktion (+ und -)

So wird der Wert des Ausdruckes (A*B/C**I+D) wie folgt ermittelt:

a.	C**I	Das Ergebnis heiße X (Potenzierung)
b.	A*B	Das Ergebnis heiße Y (Multiplikation)
c.	Y/X	Das Ergebnis heiße Z (Division)
d.	Z+D	Letzte Operation (Addition)

Stehen Operatoren derselben Rangordnung in einem Ausdruck (mit Ausnahme der Potenzierung), so werden die Einzeloperationen innerhalb des Ausdrucks von links nach rechts ausgeführt. Der arithmetische Ausdruck A/B*C wird z.B. wie folgt ausgewertet: Zunächst wird A durch B dividiert und dann der sich ergebende Quotient mit C multipliziert.

Im Falle der Potenzierung verläuft dagegen die Auswertung von rechts nach links. Folglich wird der Wert des Ausdrucks A**B**C wie folgt ermittelt:

a.	B**C	Das Ergebnis heiße Z
b.	A**Z	Letzte Operation.

5. Gebrauch von Klammern: Wie in der Algebra können Klammern in arithmetischen Ausdrücken benutzt werden, um die Reihenfolge anzugeben, in der arithmetische Operationen auszuführen sind. Werden Klammern benutzt, wird der Wert des Ausdrucks innerhalb der Klammern ermittelt, bevor die außerhalb der Klammern liegenden Operationen ausgeführt werden.

Der Wert des Ausdrucks

(B+((A+B)*C)+A**2)

wird also in folgender Reihenfolge ermittelt:

a.	(A+B)	Das Ergebnis heiße X
b.	(X*C)	Das Ergebnis heiße Y
c.	A**2	Das Ergebnis heiße Z
d.	B+Y+Z	Abschließende Operationen.

6. Abbrechen nach Division ganzer Zahlen: Werden zwei ganzzahlige Operanden durch den Operator / miteinander verknüpft, oder werden zwei ganzzahlige Operanden, von denen der zweite negativ ist, durch den Operator ** miteinander verknüpft, so ist das Ergebnis der angegebenen oder implizierten Division die ganze Zahl, die sich nach Abschneiden des Bruchteils des Quotienten ergibt.

Beispiele

Operation	Ergebnis
5/2	2
-5/2	-2
5/(-2)	-2
1**(-5)	1
5**(-5)	0

4.3. Boolesche Ausdrücke

Die einfachste Form eines Booleschen Ausdruckes ist eine Boolesche Konstante, eine Boolesche Variable, eine indizierte Boolesche Variable oder eine Boolesche Funktion, deren Wert immer ein Wahrheitswert ist (d.h. entweder .TRUE. (=wahr) oder .FALSE. (=falsch)).

Kompliziertere Boolesche Ausdrücke können unter Verwendung von Booleschen Operatoren oder Vergleichsoperatoren gebildet werden. Diese Ausdrücke können in einer der drei folgenden Formen auftreten:

1. Vergleichsoperatoren, die ganzzahlige oder reelle arithmetische Ausdrücke miteinander verbinden.
2. Boolesche Operatoren, die Boolesche Konstanten, (.TRUE. und .FALSE.), Boolesche Variablen, indizierte Boolesche Variablen oder Boolesche Funktionen miteinander verbinden.
3. Boolesche Operatoren, die Boolesche Ausdrücke der unter 1 und 2 beschriebenen Form miteinander verbinden.

Punkt 1 wird in dem nachfolgenden Abschnitt "Vergleichsoperatoren" besprochen. Die Punkte 2 und 3 werden in dem sich weiter anschließenden Abschnitt "Boolesche Operatoren" erläutert.

4.3.1. Vergleichsoperatoren

Die sechs Vergleichsoperatoren, denen jeweils ein Punkt vorausgehen und nachfolgen muß, sind folgende:

Vergleichsoperator	Bedeutung
.GT.	Größer als (>)
.GE.	Größer oder gleich (≥)
.LT.	Kleiner als (<)
.LE.	Kleiner oder gleich (≤)
.EQ.	Gleich (=)
.NE.	Ungleich (≠)

Die Vergleichsoperatoren drücken eine Größenbeziehung aus, die entweder wahr oder falsch sein kann. Nur ganzzahlige oder reelle arithmetische Ausdrücke können durch Vergleichsoperatoren miteinander verknüpft werden. Angenommen,

es seien die folgenden Variablen gegeben:

Variablenname	Typ
FORM, E	Reelle Variable
A, I, F	Ganzzahlige Variable
L	Boolesche Variable
C	Komplexe Variable,

dann zeigen die folgenden Beispiele zulässige und unzulässige Boolesche Ausdrücke, gebildet unter Verwendung von Vergleichsoperatoren.

Beispiele

Als Boolesche Ausdrücke zulässig:

```
(FORM*A).GT.E
A.LT.I
E**2.7.EQ.(5*FORM+4)
57.9.LE.(4.7+F)
.5.GE..9*FORM
E.EQ.27.3D+05
```

Als Boolesche Ausdrücke unzulässig:

C.LT.FORM	Komplexe Größen dürfen in Booleschen Ausdrücken nicht auftreten
C.GE.(2.7,5.9E3)	Komplexe Größen dürfen in Booleschen Ausdrücken nicht auftreten
L.EQ.(A+F)	Boolesche Größen und Vergleichsoperatoren dürfen nicht aneinander grenzen
E**2.EQ97.1E9	Der dem Vergleichsoperator folgende Punkt fehlt
.GT.9	Dem Vergleichsoperator geht kein arithmetischer Ausdruck voraus

4.3.2. Boolesche Operatoren

Es gibt drei Boolesche Operatoren, denen jeweils ein Punkt vorausgehen und nachfolgen muß. Dies sind folgende (A und B stellen Boolesche Konstanten oder Variablen dar oder Ausdrücke, die Vergleichsoperatoren enthalten):

Boolescher Operator	Bedeutung
.NOT.	.NOT.A - Hat A den Wert .TRUE., so hat .NOT.A den Wert .FALSE.; hat A den Wert .FALSE., so hat .NOT.A den Wert .TRUE..
.AND.	A.AND.B - Haben sowohl A als auch B den Wert .TRUE., so hat auch A.AND.B den Wert .TRUE.; haben entweder A oder B oder beide den Wert .FALSE., so hat A.AND.B den Wert .FALSE..
.OR.	A.OR.B - Haben entweder A oder B oder beide den Wert .TRUE., so hat A.OR.B den Wert .TRUE.. Haben sowohl A als auch B den Wert .FALSE., so hat A.OR.B den Wert .FALSE..

Zwei Boolesche Operatoren können unmittelbar hintereinander nur in der Form .OR..NOT. und .AND..NOT. auftreten.

Zur Bildung von Booleschen Ausdrücken durch Boolesche Operatoren können nur Ausdrücke, deren Wert entweder .TRUE. oder .FALSE. ist, miteinander verknüpft werden.

Es seien z.B. folgende Variablen gegeben:

Variablenname	Typ
FORM, E	Reelle Variable
A, I, F	Ganzzahlige Variable
C	Komplexe Variable
L, W	Boolesche Variable,

dann zeigen die folgenden Beispiele gültige und ungültige Boolesche Ausdrücke, die unter Verwendung sowohl von Booleschen Operatoren als auch von Vergleichsoperatoren gebildet sind:

Beispiele

Als Boolesche Ausdrücke zulässig:

```
(FORM*A.GT.A).AND.W
L.AND..NOT.(I.GT.F)
(E+5.9D2.GT.2*E).OR.L
.NOT.W.AND..NOT.L
L.AND..NOT.W.OR.I.GT.F
(A**F.GT.FORM).AND..NOT.(I.EQ.E)
```

Als Boolesche Ausdrücke unzulässig:

A.AND.L	A ist kein Boolescher Ausdruck
.OR.W	Dem Operator .OR. muß ein Boolescher Ausdruck vorausgehen
.NOT.(A.GT.F)	Der dem Operator .NOT. vorauszugehende Punkt fehlt.
(C.EQ.I).AND.L	Komplexe Größen dürfen in Booleschen Ausdrücken nicht auftreten.
L.AND..OR.W	Zwischen den Operatoren .AND. und .OR. muß immer ein Boolescher Ausdruck stehen.
.AND.L	Dem Operator .AND. muß ein Boolescher Ausdruck vorausgehen.

Reihenfolge der Berechnungen in Booleschen Ausdrücken:
Werden keine Klammern verwendet oder ist der ganze Boolesche Ausdruck in einem einzigen Klammerpaar eingeschlossen, so ist die Reihenfolge, in der die Operationen ausgeführt werden, folgende:

Rangordnung	Operation
1	Auswertung von Funktionen (s. Abschnitt 8.4.2. "Funktionen")
2	Potenzierung (**)
3	Multiplikation und Division (*und/)
4	Addition und Subtraktion (+und -)
5	.LT.,.LE.,.EQ.,.NE.,.GT.,.GE.
6	.NOT.
7	.AND.
8	.OR.

Der Wert des Ausdrucks (A.GT.D**B.AND..NOT.L.OR.N) wird z.B. in folgender Reihenfolge ermittelt:

1.	D**B	Das Ergebnis heiße W (Potenzierung)
2.	A.GT.W	Das Ergebnis heiße X (Vergleichsoperator)
3.	.NOT.L	Das Ergebnis heiße Y (höchster Boolescher Operator)
4.	X.AND.Y	Das Ergebnis heiße Z (zweithöchster Boolescher Operator)
5.	Z.OR.N	Letzte Operation.

Gebrauch von Klammern in Booleschen Ausdrücken:
Klammern können in Ausdrücken verwendet werden, um die Reihenfolge anzugeben, in der die Operationen auszuführen sind. Werden Klammern benutzt, so wird der Wert des Ausdrucks in der innersten Klammer zuerst ermittelt. Der Wert des Booleschen Ausdrucks

L.AND.(M.OR.I.GT.B+D)

wird also in folgender Reihenfolge bestimmt:

1.	B+D	Das Ergebnis heiße X
2.	I.GT.X	Das Ergebnis heiße Y
3.	M.OR.Y	Das Ergebnis heiße Z
4.	L.AND.Z	Letzte Operation.

Enthält ein Boolescher Ausdruck, dem der Operator .NOT. vorausgeht, zwei oder mehrere Größen, muß er in Klammern eingeschlossen werden. Es sei z.B. angenommen, daß der Wert der Booleschen Variablen A und B .FALSE. bzw. .TRUE. ist. Die beiden folgenden Ausdrücke sind dann nicht gleichwertig:

.NOT.(A.OR.B)
.NOT.A.OR.B

Im ersten Ausdruck wird A.OR.B zuerst ausgewertet. Das Ergebnis ist .TRUE.. Jedoch ist .NOT.(.TRUE.) gleichbedeutend .FALSE.. Der Wert des ersten Ausdrucks ist infolgedessen .FALSE..

Im zweiten Ausdruck wird zuerst .NOT.A ausgewertet. Das Ergebnis ist .TRUE.. Jedoch ist .TRUE..OR.B gleichbedeutend .TRUE.. Der Wert des zweiten Ausdrucks ist daher .TRUE..

4.4. Arithmetische und Boolesche Ergibtanweisungen

Form	Elemente
v = e	v ist eine beliebige indizierte oder nichtindizierte Variable
	e ist ein beliebiger arithmetischer oder Boolescher Ausdruck

Form	Elemente
	Regel: v ist dann und nur dann eine Boolesche Variable, wenn e ein Boolescher Ausdruck ist.

Die arithmetischen und Booleschen Ergibtanweisungen in FORTRAN spiegeln in etwa eine gewöhnliche algebraische Gleichung wieder. Das Gleichheitszeichen der arithmetischen Anweisung in FORTRAN bedeutet jedoch Ersetzung und nicht Gleichsetzung. Das heißt, daß der Wert des Ausdrucks auf der rechten Seite des Gleichheitszeichens ermittelt wird und der sich so ergebende Wert den vorhandenen Wert der Variablen auf der linken Seite des Gleichheitszeichens ersetzt.

Angenommen, es seien die folgenden Variablen gegeben:

Variablenname	Typ	Länge
I, J, W	Ganzzahlige Variable	4, 4, 2
A, B, C, D	Reelle Variable	4, 4, 8, 8
E	Komplexe Variable	8
G, H	Boolesche Variable	4,4

Dann zeigen die folgenden Beispiele zulässige arithmetische Anweisungen, die unter Verwendung von Konstanten und Variablen verschiedenen Typs gebildet sind:

Anweisung	Beschreibung
A = B	Der Wert von A wird durch den Wert B ersetzt.
W = B	Der Wert von B wird durch Abbrechen ganzzahlig und ersetzt den Wert von W.
A = I	Der Wert von I wird in einen reellen Wert umgewandelt. Das Ergebnis ersetzt den Wert von A.

Anweisung	Beschreibung
I = I+1	Der Wert von I wird durch den Wert I+1 ersetzt.
E = I**J+D	I wird zur J-ten Potenz erhoben, das Ergebnis in reelle Darstellung umgewandelt und der Wert von D hinzuaddiert. Das Ergebnis ersetzt den reellen Teil der komplexen Variablen E. Der imaginäre Teil der komplexen Variablen E wird Null gesetzt.
A = C*D	Der höchste, d.h. linke Teil des Produktes von C und D ersetzt den Wert von A.
G = .TRUE.	Der Wert von G wird durch die Boolesche Konstante .TRUE. ersetzt.
H = .NOT.G	Wenn G den Wert .TRUE. besitzt, wird der Wert von H durch die Boolesche Konstante .FALSE. ersetzt. Hat G den Wert .FALSE., so wird der Wert von H durch die Boolesche Konstante .TRUE. ersetzt.
G = 3..GT.I	Der Wert von I wird in die reelle Darstellung umgewandelt. Ist die reelle Konstante 3. größer als das Ergebnis, so ersetzt die Boolesche Konstante .TRUE. den Wert von G. Ist 3. nicht größer als I, wird der Wert von G durch die Boolesche Konstante .FALSE. ersetzt.
E = (1.0,2.0)	Der Wert der komplexen Variablen E wird durch die komplexe Konstante (1.0,2.0) ersetzt. Es ist zu beachten, daß die Anweisung E = (A,B) im Falle, daß A und B reelle Variablen sind, ungültig ist.

Anweisung	Beschreibung
A = E	Der Wert der reellen Variablen A wird durch den reellen Teil der komplexen Variablen E ersetzt.
I = E	Der Wert der ganzzahligen Variablen I wird durch den ganzzahligen Anteil des Wertes des reellen Teils der komplexen Variablen E ersetzt.

4.5. Ergibtanweisung

Form	Elemente
ASSIGN n TO i	n ist eine Anweisungsnummer, die in dem Klammerausdruck einer Anweisung für gesetzten Sprung auftritt.
	i ist eine nichtindizierte ganzzahlige Variable, die 4 Speicherplätze belegt.

Die Ergibtanweisung bewirkt, daß eine nachfolgende Anweisung für gesetzten Sprung zu der Anweisung mit der Nummer n verzweigt.

5. Steueranweisungen

Normalerweise wird in FORTRAN eine Anweisung nach der anderen ausgeführt. Dieser Abschnitt behandelt die Anweisungen, die verwendet werden können, um diese normale Reihenfolge der Ausführung von Anweisungen innerhalb eines Programms zu verändern und zu steuern.

Sprunganweisungen

Diese Anweisungen bewirken, daß zu der Anweisung verzweigt wird, die durch die angegebene Anweisungsnummer gekennzeichnet ist. Es gibt drei Arten von Sprunganweisungen: den unbedingten Sprung, den berechneten Sprung und den gesetzten Sprung. Bei jeder Ausführung des gleichen unbedingten Sprunges wird zur gleichen Anweisung verzweigt. Die Anweisungen für berechneten und gesetzten Sprung bewirken dagegen Verzweigung zu einer von mehreren möglichen Anweisungen, in Abhängigkeit von dem augenblicklichen Wert einer bestimmten Variablen.

5.1. Unbedingte Sprunganweisung

Form	Elemente
GO TO n	n ist die Anweisungsnummer einer beliebigen ausführbaren Anweisung innerhalb desselben Programms.

Die Sprunganweisung verzweigt zur Anweisung mit der Nummer n. Diese wird als nächste ausgeführt. Die Anweisungen ab Anweisung n werden aufeinanderfolgend ausgeführt, bis eine andere Steuerungsanweisung erneut die Reihenfolge ändert.

Beispiel

```
50    GO TO 25
10    A = B + C
        :
        :
25    C = E**2
        :
        :
```

Erklärung

Bei jeder Ausführung der Anweisung Nummer 50 wird zu der Anweisung Nummer 25 verzweigt.

5.2. Anweisung für berechneten Sprung

Form	Elemente
GO TO $(n_1,n_2,\ldots,n_m),i$	$n_1,n_2,\ldots,n_m$ sind Nummern von ausführbaren Anweisungen desselben Programms.
	i ist eine nichtindizierte ganzzahlige Variable, deren Wert zwischen 1 und m einschließlich liegt.

Die Anweisung für berechneten Sprung gibt die Möglichkeit, zu einer von m verschiedenen Anweisung zu verzweigen. Dies erfolgt in Abhängigkeit von dem Wert von i im Zeitpunkt der Ausführung der Anweisung. Die nächste ausgeführte Anweisung ist die i-te der in dem Klammerausdruck aufgeführten m Anweisungen, wenn dieser von links nach rechts gelesen wird. Liegt der Wert von i außerhalb des zulässigen Bereiches, wird die nächste Anweisung ausgeführt, d.h. die Anweisung, die der Anweisung für gesetzten Sprung unmittelbar folgt.

Beispiel

```
    GO TO (25,10,50,7),FELD
          :
50 A = B+C
          :
 7 C = E**2+A
          :
25 L = C.GT.D.AND.F.LE.G
          :
10 B = 21.3E02
```

Erklärung

Ist in diesem Beispiel der Wert der ganzzahligen Variablen FELD gleich 1, wird als nächste die Anweisung Nummer 25 ausgeführt. Ist der Wert von FELD gleich 2, wird die Anweisung Nummer 10 als nächste ausgeführt usw..

5.3. Anweisung für gesetzten Sprung

Form	Elemente
GO TO i,(n_1,n_2,...,n_m)	i ist eine nichtindizierte, ganzzahlige Variable. n_1,n_2,...,n_m sind Nummern von ausführbaren Anweisungen desselben Programms.

Die Anweisung für gesetzten Sprung bewirkt eine Verzweigung zu der Anweisung, deren Nummer i zuletzt durch eine Ergibtanweisung zugewiesen worden ist. i kann jeden der Werte n_1,n_2,...,n_m annehmen.

Beispiel

```
GO TO IN, (10,25,8)
```

Ist der Wert der Variablen IN gleich 8, wird als nächste die Anweisung Nummer 8 ausgeführt. Ist der Wert der Variablen IN gleich 10, wird als nächste die Anweisung Nummer 10 ausgeführt. Ist der Wert der Variablen IN gleich 25, wird als nächste die Anweisung Nummer 25 ausgeführt.

Eine nichtindizierte ganzzahlige Variable n kann in mehreren Anweisungen für gesetzten Sprung verwendet werden. Eine Ergibtanweisung kann dieser Variablen eine beliebige Anweisungsnummer aus der Liste einer jeden folgenden Anweisung für gesetzten Sprung, die sich auf diese Variable bezieht, zuordnen.

Wird eine Anweisung für gesetzten Sprung ausgeführt, ohne daß der ganzzahligen Variablen i eine der Anweisungsnummern aus der Liste dieser Anweisung zugeordnet ist, können die Folgen nicht vorhergesagt werden.

Der jeweilige Wert der ganzzahligen Variablen i ist durch die zuletzt ausgeführte Ergibtanweisung festgelegt. Die Variable in einer Anweisung für gesetzten Sprung kann in demselben Programm auch in anderem Zusammenhang dort verwendet werden, wo ganzzahlige Variable zugelassen sind. Folgende Einschränkung ist hierbei zu beachten: Der Gebrauch einer Variablen in einer Anweisung für gesetzten Sprung ist nur dann zulässig, wenn die letzte Anweisung, die ihr einen Wert zugeordnet hat, eine Ergibtanweisung war. In ähnlicher Weise ist der Gebrauch einer Variablen in einer Anweisung, in der ihr augenblicklicher Wert ganzzahlig ist und nicht die Nummer einer Anweisung sein darf, nur dann zulässig, wenn die letzte Anweisung, die der Variablen einen Wert zugewiesen hat, keine Ergibtanweisung war.

Beispiel 1

```
        :
   ASSIGN 50 TO NUMMER
10 GO TO NUMMER, (35,50,25,12,18)
        :
50 A = B +C
        :
```

Erklärung

Im vorstehenden Beispiel wird Anweisung 50 unmittelbar im Anschluß an Anweisung 10 ausgeführt.

Beispiel 2

```
          :
    ASSIGN  10 TO FELD
          :
13  GO TO FELD,  (8,12,25,50,10)
          :
 8  A = B+C
          :
10  B = C+D
    FELD = 25
    X = B**FELD
    ASSIGN 25 TO FELD
    GO TO 13
          :
25  C = E**2
```

Erklärung

In obigem Beispiel wird bei der ersten Ausführung der Anweisung 13 nach Anweisung 10 verzweigt. Bei der zweiten Ausführung der Anweisung 13 wird nach Anweisung 25 verzweigt. Der Gebrauch von FELD in dem arithmetischen Ausdruck ist nur wegen der vorausgehenden arithmetischen Ergibtanweisung zulässig. Die arithmetische Ergibtanweisung zu FELD schließt den Gebrauch von FELD in einem

gesetzten Sprung aus. Eine neue Ergibtanweisung für gesetzten Sprung wird ausgeführt, um FELD für diesen Zweck wieder verfügbar zu machen.

5.4. Arithmetische Wennanweisung

Form	Elemente
IF (e) n_1,n_2,n_3	e ist ein arithmetischer Ausdruck, der keine komplexen Größen enthält.
	n_1,n_2 und n_3 sind Nummern von ausführbaren Anweisungen desselben Programms.

Die Wennanweisung verzweigt nach Anweisung n_1, n_2 oder n_3 in Abhängigkeit von dem Wert des arithmetischen Ausdruckes. Es wird nach Anweisung n_1 verzweigt, wenn der Wert von e negativ ist, nach n_2, wenn er Null und nach n_3, wenn er positiv ist.

Beispiel

```
      .
      .
      .
   IF (A(J,K)**3-B)10,4,30
      .
      .
      .
 4 D = B+C
      .
      .
      .
30 C = D**2
      .
      .
      .
10 E = (F*B)/D+1
      .
      .
      .
```

Erklärung

Wenn in obigem Beispiel der Wert des Ausdrucks A(J,K)**3-B negativ ist, wird als nächste die Anweisung Nummer 10 ausgeführt. Wenn der Wert des Ausdrucks positiv ist, wird als nächste die Anweisung mit der Nummer 30 ausgeführt.

5.5. Boolesche Wennanweisung

Form	Elemente
IF (e) S	e ist ein Boolescher Ausdruck. S ist eine ausführbare Anweisung (ausgenommen Schleifenanweisungen und weitere Boolesche Wennanweisungen).

Wenn der Wert des Booleschen Ausdrucks e .TRUE. ist, wird Anweisung S ausgeführt. Der Programmablauf geht anschließend zur nächsten Anweisung über, es sei denn, S ist eine Wenn- oder Sprunganweisung. In diesem Fall wird entsprechend verzweigt. Wenn S eine Aufrufanweisung ist, wird nach dem Rücksprung aus dem Unterprogramm der Ablauf mit der nächsten Anweisung, die der Booleschen Wennanweisung folgt, fortgesetzt. Ist der Wert von e .FALSE., wird unmittelbar zur nächsten Anweisung nach der Booleschen Wennanweisung übergegangen.

Beispiel 1

```
      :
 5  IF (A.LE.0.0) GO TO 25
10   C = D+E
15  IF (A.EQ.B) FELD = 2.0*A/C
20  F  = G/H
      :
25  W  = X**Z
```

Erklärung

Wenn der Wert des Ausdrucks in Anweisung 5 .TRUE. ist (d.h. wenn A kleiner oder gleich 0.0 ist), wird die Anweisung GO TO 25 als nächste ausgeführt und infolgedessen zur Anweisung mit der Nummer 25 verzweigt. Wenn der Wert des Ausdrucks .FALSE. ist (d.h., wenn A größer als 0.0 ist), wird die Anweisung GO TO 25 ignoriert und zur Anweisung mit der Nummer 10 weitergegangen.

Wenn der Wert des Ausdrucks in Anweisung 15 .TRUE. ist (d.h. wenn A gleich B ist), wird der Wert von FELD durch den Wert des Ausdrucks (2.0*A/C) ersetzt und anschließend die Anweisung mit der Nummer 20 ausgeführt. Ist der Wert des Ausdrucks in Anweisung 15 .FALSE. (d.h. A ist nicht gleich B), bleibt der Wert von FELD unverändert und die Anweisung mit der Nummer 20 wird als nächste ausgeführt.

Beispiel 2

Angenommen, P und Q seien Boolesche Variablen

```
          :
 5 IF (P.OR..NOT.Q) A=B
10 C = B**2
          :
```

Erklärung

Wenn der Wert des Ausdrucks in Anweisung 5 .TRUE. ist, wird der Wert von A durch den Wert von B ersetzt und Anweisung 10 als nächste ausgeführt. Ist der Wert des Ausdrucks .FALSE., wird die Anweisung A=B übersprungen und Anweisung 10 als nächste ausgeführt.

5.6. Schleifenanweisung

Form	Elemente
DO n i = m_1,m_2,m_3 oder	n ist die Nummer einer nachfolgenden ausführbaren Anweisung.
DO n i = m_1,m_2	i ist eine nichtindizierte ganzzahlige Variable und wird "Index" genannt.

Form

Elemente

m_1,m_2 und m_3 sind vorzeichenlose, ganzzahlige Konstanten größer Null oder vorzeichenlose, nichtindizierte, ganzzahlige Variablen, deren Wert größer als Null ist.

Ist m_3 nicht angegeben, wird sein Wert zu 1 angenommen.

Die Schleifenanweisung bewirkt, daß die folgenden Anweisungen bis zu der Anweisung mit der Nummer n, diese selbst mit eingeschlossen, wiederholt ausgeführt werden. Bei der ersten Ausführung der Anweisungen im Schleifenbereich besitzt i den Anfangswert m_1. Bei jeder folgenden Ausführung ist i um den Wert von m_3 erhöht. Hat i den größten, m_2 nicht übersteigenden Wert angenommen, so verzweigt das Programm am Ende der Iteration zu der Anweisung, die der Anweisung n folgt.

Die Anzahl der Durchläufe der Anweisungen im Schleifenbereich ist somit gegeben durch den Ausdruck

$$\left[\frac{m_2 - m_1}{m_3}\right] + 1$$

wobei die eckigen Klammern den größten ganzzahligen Wert kennzeichnen, der den Wert des in der Klammer stehenden Ausdrucks nicht übersteigt. Ist m_2 kleiner als m_1, werden die Anweisungen in dem Bereich der Schleifenanweisung einmal ausgeführt. Nach Ausführung der Schleifenanweisung ist die Schleifenvariable i nicht definiert.

Es gibt in FORTRAN verschiedene Arten, Schleifen (d.h. mehrfach durchlaufene Anweisungen) zu programmieren. Angenommen, ein Unternehmen habe 1000 verschiedene Maschinenteile auf Lager. Es kann dort erwünscht sein, in regelmäßigen Abständen für jedes Teil die auf Lager befindliche Menge zu berechnen. Diese Menge kann als Differenz zwischen dem vorhandenen Lagerbestand LAGER(I) und dem Abgang AUS(I) errechnet werden.

Beispiel

```
    :
 5 I=0
10 I=I+1
25 LAGER(I)=LAGER(I)-AUS(I)
15 IF (I-1000) 10,30,30
30 A=B+C
    :
```

Erklärung

Die drei Anweisungen 5, 10 und 15, die zur Steuerung des Schleifendurchlaufes benötigt werden, könnten durch eine einzige Schleifenanweisung, wie in Beispiel 1 gezeigt, ersetzt werden.

Beispiel 1

```
        :
   DO 25 I=1,1000
25 LAGER(I)=LAGER(I)-AUS(I)
30 A=B+C
        :
```

Erklärung

In Beispiel 1 wird der Schleifenvariablen I zunächst der Anfangswert 1 zugewiesen. Vor der zweiten Ausführung der Anweisung 25 wird I um 1 erhöht und Anweisung 25 erneut ausgeführt. Nach 1000 Ausführungen dieser Anweisung ist I gleich 1000. Da I jetzt gleich dem höchsten Wert ist, der nicht den Vergleichswert 1000 überschreitet, verläßt das Programm die Schleife und führt Anweisung 30 als nächste aus. Es ist zu beachten, daß die Schleifenvariable I jetzt keinen definierten Wert mehr besitzt. Ihr Wert ist nicht notwendigerweise gleich 1000 oder 1001.

Beispiel 2

```
          :
   DO 25 I=1,10,2
15 J=I+K
25 AFELD(J)=BEFELD(J)
30 A=B+C
          :
```

Erklärung

In Beispiel 2 ist Anweisung 25 das Ende des Schleifenbereiches. Der Schleifenvariablen I wird zunächst 1 als Anfangswert zugewiesen. Vor dem zweiten Durchlaufen der Anweisungen des Schleifenbereiches wird I um 2 erhöht. Nach dem fünften Durchlauf der Schleife ist I gleich 9. Da I jetzt gleich dem höchsten Wert ist, der nicht den Vergleichswert 10 überschreitet, verzweigt das Programm aus der Schleife heraus und Anweisung 30 wird als nächste ausgeführt. Es ist zu beachten, daß der Wert der Schleifenvariablen I jetzt nicht mehr definiert ist. Ihr Wert ist nicht notwendigerweise gleich 9 oder 11.

Hinweise

1. Die Parameter einer Schleifenanweisung (i,m_1,m_2,m_3) dürfen nicht durch eine Anweisung innerhalb des Schleifenbereiches geändert werden.

2. Innerhalb des Schleifenbereiches können wieder Schleifenanweisungen auftreten. Alle Anweisungen im Bereich der inneren Schleife müssen im Bereich der äußeren Schleife enthalten sein. Eine Folge von Schleifenanweisungen, die dieser Regel genügen, wird eine Schleifenschachtelung genannt.

Beispiel 1

```
   DO 50 I=1,4
   A(I)=B(I)**2
   DO 50 J=1,5          } innerer Schleifenbereich
50 C(J+1)=A(I)          } äußerer Schleifenbereich
```

Beispiel 2

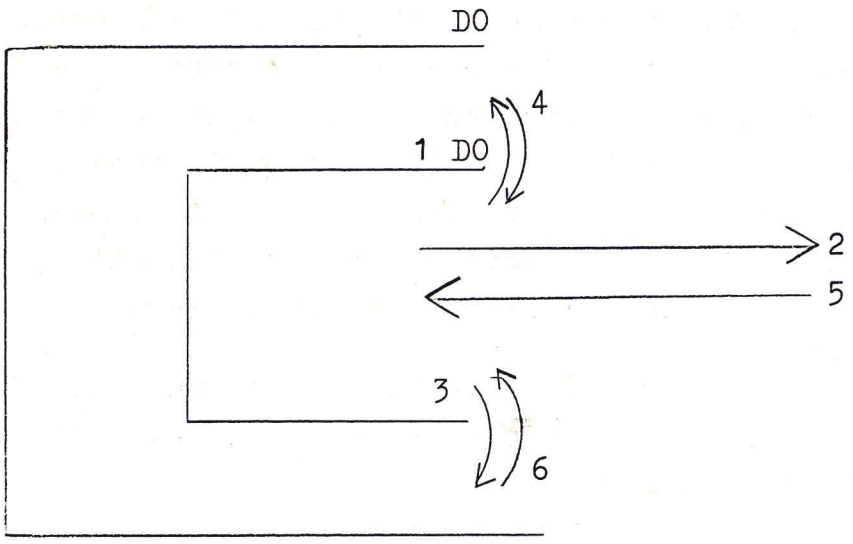
```
   DO 10 INDEX=L,M
   N=INDEX+K
   DO 15 J=1,100,2      } innerer Schleifenbereich
15 TAB(J)=SUM(J,N)-1    } äußerer Schleifenbereich
10 B(N)=A(N)
```

3. Ein Sprung aus dem Bereich einer Schleife ist jederzeit zugelassen. Tritt dieser Fall ein, so bleibt die Schleifenvariable mit dem zuletzt angenommenen Wert definiert. Sie wird also nicht undefiniert wie bei normaler Beendigung der Schleife.

4. Ein Sprung in den Bereich der innersten Schleife hinein ist dann und nur dann erlaubt, wenn ein Sprung aus dem innersten Schleifenbereich heraus erfolgte und keiner der Schleifenparameter (i, m_1,m_2 oder m_3)

außerhalb des Schleifenbereiches verändert wurde. Ein Sprung in den Bereich einer beliebigen anderen Schleife der Schleifenschachtelung ist nicht zugelassen.

Beispiel



In dem vorhergehenden Beispiel sind die Sprünge mit den Nummern 1, 2 und 3 zugelassen, diejenigen mit den Nummern 4 und 6 aber nicht. Der Sprung Nummer 5 ist dann und nur dann zulässig, wenn der Sprung Nummer 2 zuerst auftritt, zur Ausführung des Sprunges 5 führt und die Bedingungen des Punktes 4 dieser Hinweise außerhalb des Bereiches der Schleife nicht verletzt werden.

5. Die Parameter i, m_1,m_2 oder m_3 können außerhalb des Schleifenbereiches verändert werden, wenn nicht wieder in den Bereich der Schleifenanweisung, die diese Parameter benutzt, zurückgesprungen wird.

6. Die letzte Anweisung innerhalb eines Schleifenbereiches (Anweisung n) darf nicht eine Sprunganweisung, eine arithmetische Wennanweisung, eine weitere Schleifenanweisung und auch keine Rücksprung-, Stop- oder Pauseanweisung sein. Boolesche Wennanweisungen als letzte Anweisung innerhalb des Bereiches einer Schleife sind erlaubt. Wenn jedoch die Boolesche Wennanweisung eine arithmetische Wennanweisung oder eine beliebige Sprunganweisung enthält, erfolgt Erhöhung und Testen des Schleifenparameters nur dann, wenn der Wert des Booleschen Ausdruckes innerhalb der Booleschen Wennanweisung .FALSE. ist, oder wenn die Variable einer berechneten Sprunganweisung außerhalb des für sie zulässigen Bereichs liegt.

7. Der Gebrauch von Unterprogrammen und der Rücksprung aus ihnen ist innerhalb jeder Schleife einer Schleifenschachtelung zugelassen.

5.7. Leeranweisung

Form	Elemente
CONTINUE	keine

CONTINUE ist eine Leeranweisung, die an beliebiger Stelle in das Primärprogramm eingefügt werden kann, ohne daß die Reihenfolge der Ausführung der anderen Anweisungen beeinflußt wird. Sie kann dazu verwendet werden, als letzte Anweisung im Bereich einer Schleife die Beendigung der Schleife mit einer Sprunganweisung, einer arithmetischen Wennanweisung oder einer weiteren Schleifenanweisung zu verhindern.

Beispiel 1

```
        :
   DO 30 I=1,20
 7 IF (A(I)-B(I))5,30,30
 5 A(I)=A(I)+1.0
   B(I)=B(I)-2.0
   GO TO 7
30 CONTINUE
40 C=A(3)+B(7)
        :
```

Erklärung

In Beispiel 1 wird die Leeranweisung benutzt, um als letzte Anweisung im Schleifenbereich die Beendigung der Schleife mit der Anweisung GO TO 7 zu vermeiden.

Beispiel 2

```
        :
   DO 30 I=1,20
   IF (A(I)-B(I))5,40,40
 5 A(I)=C(I)
   GO TO 30
40 A(I)=0.0
30 CONTINUE
        :
```

Erklärung

In Beispiel 2 liefert die Leeranweisung einen Bezugspunkt, durch den es dem Programmierer ermöglicht wird, die Ausführung der Anweisung 40 zu übergehen.

5.8. Pauseanweisung

Form	Elemente
PAUSE oder PAUSE n oder PAUSE 'Nachricht'	n ist eine vorzeichenlose, 1- bis 5stellige, ganzzahlige Konstante. 'Nachricht' ist ein Literal.

Die Pauseanweisung veranlaßt das Programm, PAUSE auszugeben. Ist n angegeben, so wird PAUSE n ausgegeben. Ist ein Literal: 'Nachricht' angegeben, wird PAUSE Nachricht ausgegeben. Das Programm wartet, bis ein Eingriff des Operateurs seine Fortsetzung veranlaßt, woraufhin die Anweisung, die der Pauseanweisung folgt, ausgeführt wird.

5.9. Stopanweisung

Form	Elemente
STOP oder STOP n	n ist eine vorzeichenlose, 1-bis 5stellige, ganzzahlige Konstante.

Diese Anweisung beendet die Abarbeitung des Maschinenprogramms und gibt n aus, wenn dies angegeben wurde.

5.10. Aufrufanweisung

Die Aufrufanweisung wird nur dazu verwendet, ein Subroutinenunterprogramm aufzurufen.

Form	Elemente
CALL Name (P1,P2,...,Pn)	Name ist der Name des Subroutinenunterprogramms. P1,P2,...,Pn sind die Aktualparameter, die an das Subroutinenunterprogramm weitergegeben werden.

Beispiele

```
CALL AUS
CALL MATMPY(X,5,40,Y,7,2)
CALL QDRTIC(X,Y,Z,ROOT1,ROOT2)
CALL SUB1(X+Y*5, 'ABDF',SINUS)
```

Die Aufrufanweisung verzweigt zu dem Subroutinenunterprogramm und ersetzt die Formalparameter durch den Wert der Aktualparameter, die in der Aufrufanweisung erscheinen. Als Argumente einer Aufrufanweisung können auftreten:

Jeder Typ von Konstanten, jeder Typ von indizierten oder nichtindizierten Variablen, ein Ausdruck, der Name eines Unterprogramms, die Nummer einer ausführbaren Anweisung (siehe Abschnitt 5.11. "Rücksprunganweisung"), die Nummer einer Formatanweisung oder der Name einer Namenliste.

Anweisungsnummern werden in der Form &n geschrieben, wobei n die Anweisungsnummer ist. Im allgemeinen müssen die Parameter einer Aufrufanweisung hinsichtlich Anzahl, Reihenfolge und Typ mit den entsprechenden Parametern des Unterprogramms übereinstimmen. Ist der Aktualparameter die Nummer einer Formatanweisung, so muß der entsprechende Formalparameter als ein Feld innerhalb des Unterprogramms definiert sein. Ist der Aktualparameter der Name einer Namenliste, so darf der Formalparameter kein Feld sein. Die Feldgrößen müssen im Unterprogramm und den aufrufenden Programmen gleich sein, es sei denn, veränderliche Indexgrenzen werden benutzt (s.Abschnitt 7.5.1. "Veränderliche Indexgrenzen"). Wenn ein Aktualparameter einem Formalparameter entspricht, der in dem aufgerufenen Unterprogramm definiert oder neu definiert wird, muß der Aktualparameter ein Variablenname, ein Name einer indizierten Variablen oder ein Feldname sein.

5.11. Rücksprunganweisung

Form	Elemente
RETURN oder RETURN i	i ist eine ganzzahlige Konstante oder Variable, deren Wert (etwa n) die n-te Anweisungsnummer in der Argumentenliste einer Subroutinenanweisung bezeichnet.

Normalerweise wird nach Ausführung der Rücksprunganweisung eines Subroutinenunterprogramms die Anweisung ausgeführt, die der Aufrufanweisung des aufrufenden Programms als nächste folgt. Es ist aber auch möglich, zu einer beliebigen numerierten Anweisung des aufrufenden Programms zurückzukehren, indem die Rücksprunganweisung in der Form RETURN i geschrieben wird, wobei i eine ganzzahlige Konstante oder Variable ist. Rücksprünge der Art RETURN können sowohl in einem Subroutinen- als auch in einem Funktionsunterprogramm verwendet werden (s. "Rücksprung- und Endanweisung in einem Funktionsunterprogramm"). Dagegen können Rücksprünge der Art RETURN i nur in einem Subroutinenunterprogramm auftreten. Innerhalb eines Hauptprogramms kommt der Rücksprunganweisung dieselbe Funktion zu wie einer Stopanweisung (s. Abschnitt 8. "Organisatorische Anweisungen").

Beispiel

Aufrufendes Programm

```
      .
      .
      .
10 CALL SUB(A,B,C,&30,&40)
20 Y=A+B
      .
      .
30 Y=A+C
      .
      .
    END
```

Unterprogramm

```
SUBROUTINE SUB(X,Y,Z,*,*)
      .
      .
      .
100 IF (R) 200,300,400
200 RETURN
300 RETURN 1
400 RETURN 2
    END
```

Erklärung

Die Ausführung der Anweisung 10 in dem aufrufenden Programm bewirkt einen Sprung in das Unterprogramm SUB. Wenn Anweisung 100 ausgeführt wird, wird zu dem aufrufenden Programm zurückgesprungen, und zwar zu Anweisung 20, 30 oder 40, je nachdem, ob R kleiner, gleich oder größer als Null ist.

Eine Aufrufanweisung, die einen Rücksprung der Form RETURN i benutzt, ist am besten als eine Aufrufanweisung, gefolgt von einer berechneten Sprunganweisung zu verstehen. Beispielsweise ist die Aufrufanweisung

```
CALL SUB (P,&20, Q,&35,R,&22)
```

gleichwertig mit

```
CALL SUB (P,Q,R,I)
GO TO (20,35,22),I
```

falls dem Index I in dem aufgerufenen Unterprogramm einer der Werte 1, 2 oder 3 zugewiesen wird.

6. Ein-Ausgabe-Anweisungen

6.1. Allgemeines

Die Ein-Ausgabe-Anweisungen versetzen den Benutzer in die Lage, Daten, die einer benannten Datenmenge angehören, zwischen Ein-Ausgabeeinheiten - wie Plattenspeicher-Einheiten, Kartenlese- und Magnetband-Einheiten - und dem Speicher zu übertragen. Die benannte Datenmenge wird Dateiengruppe genannt und ist nicht an einen bestimmten peripheren Gerätetyp gebunden.

Gruppen von Dateien werden durch eine vorzeichenlose ganzzahlige Konstante oder durch eine ganzzahlige Variable - die Gruppennummer - gekennzeichnet.

In FORTRAN wird eine Gruppe von Dateien als eine kontinuierliche Folge von Daten angesehen, die in Sätze unterteilt sein kann. Diese Unterteilung kann auf einer der folgenden Weisen beschrieben werden:

1. durch eine Formatanweisung, auf die sich eine Ein-Ausgabe-Anweisung bezieht.
2. durch eine Ein-Ausgabe-Liste, die in einer Ein-Ausgabe-Anweisung enthalten ist.
3. durch einen Namen einer Namenliste, der in einer Ein-Ausgabe-Anweisung aufgeführt ist.

Zusätzlich zur Unterteilung der Dateiengruppen in Sätze kann eine Formatanweisung benutzt werden, um die Form, in der die Daten übermittelt werden sollen, zu beschreiben.

Es gibt fünf Ein-Ausgabe-Anweisungen:

Lesen	(READ)
Schreiben	(WRITE)
Dateiabschluß	(END FILE)
Rückspulen	(REWIND)
Rücksetzen	(BACKSPACE)

Die Lese- und Schreibanweisung wird für die Übertragung von Daten in oder aus dem Speicher verwendet. Die Dateiabschlußanweisung definiert das Ende einer Dateiengruppe. Die Rückspul- und Rücksetzanweisung steuert die Positionierung der Dateiengruppen.

Obwohl die im folgenden beschriebenen Ein-Ausgabe-Anweisungen unabhängig von den verwendeten Geräten derart sind, daß eine gegebene Ein-Ausgabe-Anweisung auf eine beliebige Anzahl von Geräten oder Gerätetypen angewendet werden kann, ist es oft zweckmäßig, den Ursprung oder das Ziel der zu übertragenden Daten zu betrachten. Aus Gründen der Anschaulichkeit beziehen sich die folgenden Beispiele auf Lochkarteneingabe und Druckerausgabe.

6.2. Grundlegende Ein-Ausgabe-Anweisungen

Die grundlegenden Ein-Ausgabe-Anweisungen sind READ und WRITE. Die Format- und Namenlistenanweisungen können, obwohl sie keine Ein-Ausgabe-Anweisungen sind, in Verbindung mit bestimmten Formen der Lese- bzw. Schreibanweisung verwendet werden.

Alle vier Anweisungen werden in den folgenden Abschnitten ausführlich besprochen.

6.2.1. Leseanweisung

Form	Elemente
READ (a,b,END=c, ERR=d)Liste	a ist eine vorzeichenlose, ganzzahlige Konstante oder eine ganzzahlige Variable und stellt die Nummer einer Gruppe von Dateien dar.
	b ist entweder die Anweisungsnummer oder der Feldname einer Formatanweisung, die die zu lesenden Daten beschreibt, oder der Name einer Namenliste.
	c ist die Nummer der Anweisung, zu der beim Erkennen des Endes der Dateiengruppe verzweigt werden soll.
	d ist die Nummer der Anweisung, zu der beim Auftreten einer Fehlerbedingung bei der Datenübertragung verzweigt werden soll.
	Liste ist eine Folge von durch Kommas getrennten Variablen- oder Feldnamen, die indiziert und inkrementiert sein können. Sie gibt die Anzahl der zu lesenden Posten und die Speicherbereiche, in die die Daten zu übertragen sind, an.

6.2.1.1. Die Parameter END und ERR

Die Parameter END=c und ERR=d können wahlweise verwendet werden. Sie müssen also nicht auftreten. Werden beide benutzt, so können sie in beliebiger Reihenfolge hinter

dem Parameter b geschrieben werden. Die folgenden Anweisungen sind also zulässig:

```
READ(5,36)A,B,C
READ(5,36,ERR=92)A,B,C
READ(5,36,END=99)A,B,C
READ(5,36,ERR=92,END=99)A,B,C
```

Beim Auftreten einer Fehlerbedingung wird die Leseanweisung beendet und dann zu der Anweisung verzweigt, die durch den ERR-Parameter angegeben ist. Es wird in diesem Fall kein Hinweis geliefert, welcher oder welche Posten fehlerhafte Daten enthalten. Das Programm erhält bei der Verzweigung zu der durch den END-Parameter bezeichneten Anweisung keine Information darüber, wieviele Posten bis zum Erreichen des Endes der Dateiengruppe gelesen worden sind. Diese Anzahl kann im übrigen auch gleich Null sein. Wenn in Ausführung einer Leseanweisung sowohl die Ende- als auch eine Fehlerbedingung auftritt, wird zu der durch den ERR-Parameter bestimmten Anweisung verzweigt. Der Übergang zu der im END-Parameter angegebenen Anweisung wird bei Ausführung der nächsten Leseanweisung bewirkt, die sich auf dieselbe Dateiengruppe bezieht, es sei denn, eine inzwischen ausgeführt Ein-Ausgabe-Anweisung hat die Bedingung aufgehoben.

Wenn bei Ausführung einer Leseanweisung, die keinen ERR-oder END-Parameter enthält, eine Fehler- oder Endebedingung auftritt, wird die Ausführung des Maschinenprogramms beendet.

Folgt eine Leseanweisung einer Schreibanweisung oder einer Dateiabschlußanweisung, ohne daß eine Rückspulanweisung oder eine Rücksetzanweisung für dieselbe Dateiengruppe dazwischen geschaltet war, so bewirkt dies eine Endebedingung.

Die drei grundlegenden Formen der Leseanweisung sind:

READ(a,x)
READ(a,n)Liste
READ(a)Liste

Die Parameter END=c und ERR=d können bei jeder dieser drei Formen in der zuvor beschriebenen Anordnung verwendet werden.

6.2.1.2. Die Form READ(a,x)

Diese Form wird verwendet, um Daten von der Dateiengruppe a in einen Speicherbereich einzulesen, der durch den Namen x einer Namenliste gekennzeichnet ist. Eine spezielle Liste von Variablen- oder Feldnamen wird in einer Namenlistenanweisung niedergelegt. Der Programmierer braucht lediglich den Namen der Namenliste in die Leseanweisung READ (a,x) einzusetzen, um sich später innerhalb des Programms auf diese Liste zu beziehen.

Aufbau und Regeln für die Bildung und den Gebrauch der Namenlistenanweisung werden in Abschnitt 8. "Organisatorische Anweisungen" beschrieben.

6.2.1.3. Die Form READ(a,n)Liste

Diese Form wird verwendet, um Daten von der Dateiengruppe a in Speicherbereiche zu lesen, die durch die Variablennamen in der Liste gekennzeichnet sind. Die Liste, die im Zusammenhang mit der Formatanweisung n (s.Abschnitt 8.9. "Formatanweisung") benutzt wird, bestimmt die Anzahl der zu lesenden Posten, die diesen zugeordneten Speicherbereiche und die Form, die die Daten im Speicher annehmen.

Beispiel 1

Es sei angenommen, daß A,B und C als ganzzahlige Variablen erklärt sind.

```
      :
75 FORMAT (G10,G8,G9)
      :
READ(J,75)A,B,C
      :
```

Erklärung

Die Leseanweisung im Beispiel 1 bewirkt, daß Eingabedaten von der Dateiengruppe J in die Speicherbereiche A, B und C entsprechend der angegebenen Formatanweisung 75 gelesen werden. Es werden also die ersten 10 Stellen eines Satzes gelesen, in eine ganzzahlige Binärzahl umgewandelt und in den Speicherbereich A übertragen. Es werden dann die nächsten acht Stellen gelesen, ebenfalls in eine ganzzahlige Binärzahl umgewandelt und in den Speicherbereich B übertragen. Darauf werden die folgenden 9 Stellen gelesen und umgewandelt in eine ganzzahlige Binärzahl in C gespeichert.

Die Liste kann aus einer Leseanweisung READ(a,n)Liste weggelassen werden. In diesem Fall wird ein Satz überlesen, oder es werden Daten von der Dateiengruppe a in den Speicherbereich übertragen, der durch die Formatanweisung mit der Nummer n belegt wird.

Beispiel 2

```
      :
98 FORMAT('EINGABE')
      :
READ(5,98)
      :
```

Erklärung

Die Anweisungen des Beispiels 2 bewirken, daß die Zeichen E, I,N,G,A,B und E im Speicher durch die nächsten sieben Zeichen aus der Dateiengruppe 5 ersetzt werden.

Beispiel 3

```
      :
98 FORMAT(G10,'EINGABE')
      :
   READ(5,98)
      :
```

Erklärung

Die Anweisungen des Beispiels 3 bewirken, daß der nächste Satz in der Dateiengruppe 5 überlesen wird. Es werden keine Daten in den Speicher übertragen, da die Leseanweisung keine Listeneintragung enthält, die dem Formatschlüssel G10 entspricht.

6.2.1.4. Die Form READ(a)Liste

Diese Form der Leseanweisung bewirkt, daß binäre Daten (interne Darstellung) von der Dateiengruppe a in die durch die Variablennamen der Liste bezeichneten Speicherbereiche übertragen werden. Da Eingabedaten immer in interner Darstellung angegeben sind, wird eine Formatanweisung nicht benötigt. Diese Anweisung wird verwendet, um Daten wieder zurückzuübertragen, die durch die Anweisung WRITE (a)Liste geschrieben wurden.

Beispiel

```
READ (5), A, B, C
```

Erklärung

Diese Anweisung bewirkt, daß binäre Daten von der Dateiengruppe 5 in die Speicherbereiche A,B und C gelesen werden.

Die Liste kann aus der Anweisung READ(a)Liste weggelassen werden. In diesem Fall wird ein Satz überlesen.

Beispiel

READ (5)

Erklärung

Vorstehende Anweisung bewirkt, daß der nächste Satz in der Dateiengruppe 5 überlesen wird. Es werden keine Daten in den Speicher übertragen.

6.2.1.5. Indizierung von Ein-Ausgabe-Listen

Variablen innerhalb einer Ein-Ausgabe-Liste können in derselben Art wie in einer Schleifenanweisung indiziert und inkrementiert werden. Diese Variablen und ihre Indizes müssen in Klammern eingeschlossen sein. Wenn z.B. Daten in die ersten fünf Positionen des Feldes A übertragen werden sollen, dann kann dies durch den Gebrauch einer indizierten Liste in folgender Weise erreicht werden:

```
15 FORMAT(G10.3)
    :
READ(2,15)(A(I),I=1,5)
```

Diese Form ist der folgenden gleichwertig:

```
15 FORMAT(G10.3)
      :
   DO 12 I=1,5
12 READ(2,15)A(I)
```

Bei der ersten Schreibweise wird die Leseanweisung nur ein einziges Mal ausgeführt, sie überträgt aber fünf Größen. Die Formatanweisung beschreibt nur eine Größe, so daß jedesmal, wenn eine Größe übertragen wird, ein neuer Satz benötigt wird. Bei der zweiten Schreibweise wird die Leseanweisung fünf Mal ausgeführt. Diese ruft bei jeder Ausführung einen neuen Satz ab, unabhängig von der Formatanweisung.

Wie bei Schleifenanweisungen kann ein dritter Parameter zur Angabe der Schrittweite verwendet werden, um den der Index bei jeder Iteration erhöht werden soll.

```
READ(2,15)(A(I),I=1,10,2)
```

bewirkt also die Übertragung von Werten nach A(1), A(3), A(5), A(7) und A(9).

Diese Notation kann überdies geschachtelt werden.
Z.B. würde die Anweisung

```
READ(2,15)((C(I,J),D(I,J),J=1,5),I=1,4)
```

Daten in folgender Reihenfolge übertragen:

```
C(1,1),D(1,1),C(1,2),D(1,2)...C(1,5),D(1,5),
C(2,1),D(2,1),C(2,2),D(2,2)...C(2,5),D(2,5),
C(3,1),D(3,1),C(3,2),D(3,2)...C(3,5),D(3,5),
C(4,1),D(4,1),C(4,2),D(4,2)...C(4,5),D(4,5)
```

Da J der innere Index ist, ändert er sich schneller als I.

Ein weiteres Beispiel:

```
READ(2,25)I,(C(J),J=1,I)
```

Die Variable I wird zuerst gelesen. Ihr Wert dient anschließend als Index für die Angabe der Anzahl von Posten, die in das Feld C gelesen werden sollen.

Wenn Daten in ein gesamtes Feld gelesen werden sollen, ist es nicht notwendig, dieses Feld in der Ein-Ausgabe-Liste zu indizieren. Es sei beispielsweise angenommen, daß das Feld A eindimensional ist. Der entsprechende Index laufe von 1 bis 10. Dann würde die Leseanweisung

```
READ(2,5)A
```

die sich auf die Formatanweisung 5 bezieht, bewirken, daß Daten nach A(1,),A(2),...,A(10) übertragen werden.

Die Indizierung von Ein-Ausgabe-Listen gilt für die Listen von Schreibanweisungen in gleicher Weise wie für die von Leseanweisungen.

6.3. Schreibanweisung

Form	Elemente
WRITE(a,b)Liste	a ist eine vorzeichenlose, ganzzahlige Konstante oder eine gannzahlige Variable und stellt die Nummer einer Dateiengruppe dar.
	b ist entweder die Anweisungsnummer oder der Feldname einer Formatanweisung, die die zu schreibenden Daten beschreibt, oder der Name einer Namenliste.

Form	Elemente
	Liste ist eine Folge von durch Kommas getrennten Variablen- oder Feldnamen, die indiziert und inkrementiert sein können. Sie gibt die Anzahl der zu schreibenden Posten und die Speicherbereiche, aus denen sie zu entnehmen sind, an.

Die Schreibanweisung kann in vielen verschiedenen Formen auftreten. Beispielsweise kann der Parameter Liste oder b weggelassen werden.

Die drei grundlegenden Formen der Schreibanweisung sind:

WRITE(a,x)
WRITE(a,n)Liste
WRITE(a)Liste

6.3.1. Die Form WRITE(a,x)

Diese Form wird verwendet, um Daten aus einem Speicherbereich, der durch den Namen x einer Namenliste gekennzeichnet ist, in die Dateiengruppe a zu schreiben (s.Abschnitt "Die Form READ(a,x)").

Beispiel

```
WRITE (6,NAM1)
```

Erklärung

Diese Anweisung bewirkt, daß alle Variablen- und Feldnamen (d.h. deren Werte), die zu der Namenliste NAM1 gehören, in die Dateiengruppe 6 geschrieben werden (s. Abschnitt 8.).

6.3.2. Die Form WRITE(a,n)Liste

Diese Form wird verwendet, um Daten auf die Dateiengruppe a zu schreiben, und zwar aus den Speicherbereichen, die durch die Variablennamen in der Liste gekennzeichnet sind. Die Liste, die im Zusammenhang mit der angegebenen Formatanweisung n benutzt wird, bestimmt die Anzahl der zu schreibenden Posten, die Speicherbereiche und die Form, die die Daten der Dateiengruppe annehmen.

Beispiel 1

Es sei angenommen, daß A,B und C als ganzzahlige Variablen erklärt sind.

```
75 FORMAT(G10,G8,G9)
    :
WRITE(J,75)A,B,C
```

Erklärung

Die Schreibanweisung des Beispiels 1 bewirkt, daß Ausgabedaten in die Dateiengruppe J aus den Speicherbereichen A, B und C entsprechend der angegebenen Formatanweisung 75 geschrieben werden. Das bedeutet, daß die binären ganzen Zahlen A, B und C in ganze dezimale Zahlen umgewandelt und rechtsbündig in Felder der Länge 10, 8 bzw. 9 in die Dateiengruppe J geschrieben werden.

Die Liste kann aus der Anweisung WRITE (a,n)Liste weggelassen werden. In diesem Fall wird ein Leersatz eingefügt, oder es werden Daten auf die Dateiengruppe a aus dem Speicherbereich geschrieben, der von der Formatanweisung n belegt wird.

Beispiel 2

```
98 FORMAT ('EINGABE')
      :
   WRITE  (5,98)
```

Erklärung

Die Anweisungen des Beispiels 2 bewirken, daß ein Zeichen Zwischenraum und die Zeichen E,I,N,G,A,B und E auf die Dateiengruppe 5 geschrieben werden.

Beispiel 3

```
98 FORMAT (G10,'EINGABE')
      :
   WRITE  (5,98)
```

Erklärung

Die Anweisungen des Beispiels 3 bewirken, daß ein Leersatz auf die Dateiengruppe 5 übertragen wird. Daten werden nicht in die Dateiengruppe übertragen, weil die Schreibanweisung keine Listeneintragung enthält, die dem Formatschlüssel G10 entspricht.

6.3.3. Die Form WRITE(a)Liste

Diese Form bewirkt, daß binäre Daten in interner Darstellungsweise aus den Speicherbereichen, die durch die Variablennamen in der Liste gekennzeichnet sind, in die Dateiengruppe a geschrieben werden. Da Ausgabedaten immer in interner Form vorliegen, wird eine Formatanweisung nicht benötigt. Die Anweisung READ(a)Liste wird benutzt, um Daten, die durch eine Anweisung WRITE(a)Liste geschrieben wurden, wieder zurückzuübertragen.

Beispiel

```
WRITE (5) A, B, C
```

Erklärung

Diese Anweisung bewirkt, daß binäre Daten aus den Speicherbereichen A, B und C in die Dateiengruppe 5 übertragen werden.

6.4. Zusätzliche Ein-Ausgabe-Anweisungen

Die Dateiabschlußanweisung END FILE, die Rückspulanweisung REWIND und die Rücksetzanweisung BACKSPACE werden entsprechend der folgenden Beschreibung zur Dateisteuerung verwendet.

6.4.1. Dateiabschlußanweisung

Form	Elemente
END FILE a	a ist eine vorzeichenlose, ganzzahlige Konstante oder ganzzahlige Variable und stellt die Nummer einer Dateiengruppe dar.

Die Dateiabschlußanweisung schreibt einen Satz, der das Ende der Dateiengruppe a kennzeichnet. Eine nachfolgende Schreibanweisung, die sich auf a bezieht, definiert den Anfang einer neuen Dateiengruppe.

6.4.2. Rückspulanweisung

Form	Elemente
REWIND a	a ist eine vorzeichenlose, ganzzahlige Konstante oder eine ganzzahlige Variable und stellt die Nummer einer Dateiengruppe dar.

Die Rückspulanweisung bewirkt, daß eine nachfolgende Lese- oder Schreibanweisung, die sich auf a bezieht, Daten vom oder auf den ersten Satz der ersten Datei mit der Gruppennummer a überträgt.

6.4.3. Rücksetzanweisung

Form	Elemente
BACKSPACE a	a ist eine vorzeichenlose, ganzzahlige Konstante oder eine ganzzahlige Variable und stellt die Nummer einer Dateiengruppe dar.

Die Rücksetzanweisung bewirkt, daß die Dateiengruppe a um einen Satz zurückgesetzt wird. Die Anweisung hat keine Wirkung, wenn der Datenträger so gesetzt ist, daß der erste Satz der ersten Datei der Gruppe a gelesen oder geschrieben würde.

Der folgende Abschnitt befaßt sich mit den Eigenschaften früherer FORTRAN IV-Sprachen, die in die FORTRAN IV-Sprache für das Siemens-System 4004 aufgenommen worden sind. Die Übernahme dieser zusätzlichen Eigenschaften ermöglicht es, daß bestehende FORTRAN-Programme für die Verwendung auf dem Siemens-System 4004 neu übersetzt werden können, und zwar mit wenig oder überhaupt keiner Neuprogrammierung.

6.4.4. Leseanweisung

Form	Elemente
READ n,Liste	n ist die Anweisungsnummer oder der Feldname der Formatanweisung, die die Daten beschreibt.

Form	Elemente
	Liste ist eine Folge von durch Kommas getrennten Variablen- oder Feldnamen, die indiziert und inkrementiert sein können. Sie gibt die Anzahl der zu lesenden Posten und die Speicherbereiche, in die sie zu übertragen sind, an.

Diese Anweisung wird übersetzt, als wäre READ (97,n)Liste geschrieben worden.

6.4.5. Stanzanweisung

Form	Elemente
PUNCH n,Liste	n ist die Anweisungsnummer oder der Feldname der Formatanweisung, die die Daten beschreibt.
	Liste ist eine Folge von durch Kommas getrennten Variablen- oder Feldnamen, die indiziert und inkrementiert sein können. Sie gibt die Anzahl der zu schreibenden Posten und die Speicherbereiche, aus denen sie zu entnehmen sind, an.

Diese Anweisung wird übersetzt, als wäre WRITE (98,n)Liste geschrieben worden.

6.4.6. Druckanweisung

Form	Elemente
PRINT n,Liste	n ist die Anweisungsnummer oder der Feldname der Formatanweisung, die die Daten beschreibt.

Form	Elemente
	Liste ist eine Folge von durch Kommas getrennten Variablen- oder Feldnamen, die indiziert und inkrementiert sein können. Sie gibt die Anzahl der zu schreibenden Posten und die Speicherbereiche, aus denen sie zu entnehmen sind, an.

Diese Anweisung wird übersetzt, als wäre WRITE (99,n)Liste geschrieben worden.

7. Spezifikationsanweisungen

Die Spezifikationsanweisungen versehen den Compiler mit Informationen über den Typ der Namen, die im Primärprogramm verwendet werden. Außerdem liefern sie die Informationen, die benötigt werden, um Variablen und Feldern Speicherplätze zuzuweisen. Spezifikationsanweisungen können an beliebiger Stelle im Programm auftreten. Zur Erzielung möglichst günstiger Übersetzungszeiten sollten sie der ersten Formelfunktion oder ausführbaren Anweisung vorausgehen und in der Reihenfolge des Inhaltsverzeichnisses dieses Manuals geschrieben werden.

Typanweisungen

Es gibt zwei Arten von Typanweisungen, nämlich die Spezifikationsanweisung IMPLICIT und die expliciten Spezifikationsanweisungen (INTEGER - ganzzahlig, REAL - reell, COMPLEX - komplex, LOGICAL - Boolesch).

Die Spezifikationsanweisung IMPLICIT bietet dem Benutzer folgende Möglichkeiten:

1. Festlegung des Typs einer Gruppe von Variablen, von Feldern oder Funktionen durch den Anfangsbuchstaben ihres Namens.
2. Festlegung der Größe des jeder Variablen entsprechend ihrem Typ zuzuordnenden Speicherbereichs.

Die expliciten Spezifikationsanweisungen bieten dem Benutzer folgende Möglichkeiten:

1. Festlegung des Typs einer Variablen, eines Feldes und einer Funktion durch ihren jeweiligen Namen.
2. Festlegung der Größe des jeder Variablen entsprechend ihrem Typ zuzuordnenden Speicherbereichs.
3. Angabe der Dimensionen eines Feldes.
4. Zuordnung von Anfangswerten zu Variablen und Feldern.

7.1. Anweisung IMPLICIT

Form	Elemente
IMPLICIT Typ*s(a_1,a_2,...),..., Typ*s(a_1,a_2,...)	Typ stellt einen der folgenden Begriffe dar: INTEGER (ganzzahlig), REAL (reell), COMPLEX (komplex) oder LOGICAL (Boolesch). *s kann wahlweise geschrieben werden, s stellt eine der zulässigen Längenangaben für den betreffenden Typ dar. a_1,a_2,... stellen entweder einzelne alphabetische Zeichen dar, die voneinander durch Kommas getrennt sind, oder einen Bereich von Zeichen (in alphabetischer Reihenfolge). Dieser wird durch das erste und letzte Zeichen gekennzeichnet, die durch ein Minus voneinander getrennt sind, z.B. A-D.

Die Typanweisung IMPLICIT versetzt den Benutzer in die Lage, den Typ von Variablen, die in seinem Programm auftreten, derart zu erklären (und zwar als ganzzahlig, reell, komplex oder Boolesch), daß Variablen, die mit einem bestimmten Buchstaben beginnen, von einem bestimmten Typ sind. Weiterhin ermöglicht es die Anweisung IMPLICIT dem Programmierer, die Anzahl von Speicherplätzen festzulegen, die jeder Variablen einer Gruppe

zugeordnet werden soll. Der Typ, den eine Variable annehmen kann, zusammen mit den zulässigen Längenangaben, kann einer der folgenden sein:

Typ		Länge
INTEGER	(ganzzahlig)	2 oder 4 (Standardlänge ist 4)
REAL	(reell)	4 oder 8 (Standardlänge ist 4)
COMPLEX	(komplex)	8 oder 16(Standardlänge ist 8)
LOGICAL	(Boolesch)	1 oder 4 (Standardlänge ist 4)

Jedem Typ ist eine standardmäßige Längenangabe zugeordnet. Wird diese Standardlänge eines Typs gewünscht, so kann *s aus der IMPLICIT-Anweisung weggelassen werden. Es kann aber für jeden Typ die andere zulässige Länge gewählt werden. Wird diese Länge gewünscht, so muß innerhalb der IMPLICIT-Anweisung *s angegeben werden.

Beispiel 1

```
IMPLICIT REAL (A-M, O-Z, $), INTEGER (I-M)
```

Erklärung

Alle Variablen, deren Namen mit den Buchstaben I bis N beginnen, werden durch die Anweisung des Beispiels 1 als ganzzahlig erklärt. Da die Anweisung keine explicite Längenangabe enthält, *s also weggelassen ist, werden jeder Variablen vier Speicherplätze - die Standardlänge für ganze Zahlen - zugeordnet.

Alle anderen Variablen, d.h. diejenigen, deren Namen mit den Buchstaben A bis H, O bis Z oder mit $ beginnen, werden als reell mit vier Speicherplätzen je Variable erklärt.

Zu beachten ist, daß die Anweisung in Beispiel 1 dieselbe Wirkung hat wie die FORTRAN-Konvention für die Typenvereinbarung von Variablen (s. Abschnitt 3.1.6.3. "Typenvereinbarung nach der FORTRAN-Konvention").

Beispiel 2

```
IMPLICIT INTEGER*2(A-H), REAL*8(I-K),LOGICAL(L,M,N)
```

Erklärung

Alle Variablen, deren Namen mit den Buchstaben A bis H beginnen, sind als ganzzahlig mit zwei Speicherplätzen je Variable erklärt. Alle Variablen, deren Namen mit den Buchstaben I bis K beginnen, sind als reell mit acht Speicherplätzen je Variable erklärt. Alle Variablen, deren Namen mit den Buchstaben M, L oder N beginnen, sind als Boolesch mit vier Speicherplätzen je Variable erklärt.

Die übrigen Buchstaben des Alphabetes, nämlich O bis Z, und das $-Zeichen wurden in der obigen IMPLICIT-Anweisung nicht aufgeführt.Hierdurch kommt für die Zeichen die übliche FORTRAN-Konvention zur Wirkung. Alle Variablen, deren Namen mit den Buchstaben O bis Z oder dem $-Zeichen beginnen, sind also als reell erklärt, und zwar jede mit der Standardlänge von vier Speicherplätzen.

Beispiel 3

```
IMPLICIT COMPLEX*16(C-F)
```

Erklärung

Alle Variablen, deren Namen mit den Buchstaben C bis F beginnen, sind als komplex erklärt, und zwar jede mit je acht Speicherplätzen für den reellen und acht Speicherplätzen für den imaginären Teil. Die Typen der Variablen, deren Namen mit den Buchstaben A, B, G bis Z oder dem $-Zeichen beginnen, sind durch die FORTRAN-Konvention festgelegt.

7.2. Abnormalanweisung

Form	Elemente
ABNORMAL e,f,g...	e,f,g... sind Namen von Funktionen, die als abnormal erklärt werden.

Enthält das Primärprogramm keine Abnormalanweisung, so wird angenommen, daß alle Funktionen mit Ausnahme der in FORTRAN enthaltenen mathematischen Unterprogramme, die im Anhang 3 aufgeführt sind, abnormal sind. Wenn eine Abnormalanweisung auftritt, wird angenommen, daß nur die Funktionen, die innerhalb dieser Anweisung aufgeführt sind, abnormal sind, und daß alle anderen Funktionen normal sind. Wenn insbesondere nur eine einzige Abnormalanweisung auftritt und diese keine Funktionsnamen enthält, werden alle Funktionen als normal angesehen.

7.2.1. Die Verwendung der Abnormalanweisung

Die Abnormalanweisung kann vom Programmierer verwendet werden, damit der Compiler ein besseres Objektprogramm erzeugt, da er sich bei normalen Funktionen nicht mit der Suche nach und der Ersetzung von doppelten Teilausdrücken befaßt.

7.2.2. Normale und abnormale Funktionen

Eine Funktion ist unter folgenden Voraussetzungem normal:

a) Keines ihrer Argumente wird durch das Funktionsunterprogramm verändert. Ihre einzige Ausgabe ist der jeweils errechnete Wert der Funktion.

b) Wird die Funktion mehrfach angesprochen, so liefert sie genau das gleiche Ergebnis, wenn der Wert der Aktualparameter in jedem Fall genau gleich ist. Dies schließt Funktionen aus, die beispielsweise Variablen enthalten, deren Werte von einem Aufruf der Funktion zum nächsten aufbewahrt werden.

c) Das Funktionsunterprogramm führt keine Ein-Ausgabe-Operation aus.

Eine Funktion ist abnormal, sobald sie auch nur eine der oben aufgeführten Eigenschaften nicht besitzt.

7.3. Externanweisung

Form	Elemente
EXTERNAL a,b,c...	a,b,c,... sind Namen von Unterprogrammen, die in diesem (Unter-)Programm beim Aufruf von anderen Unterprogrammen als Aktualparameter verwendet werden.

Der Name eines jeden Unterprogramms, der als Argument eines anderen Unterprogramms verwendet wird, muß in einer Externanweisung erscheinen. Es sei beispielsweise angenommen, daß SUB und MULT in den folgenden Anweisungen Unterprogrammnamen sind:

Beispiel 1

Aufrufendes Programm

```
      :
      :
EXTERNAL MULT
      :
      :
CALL SUB(A,MULT,C)
```

Unterprogramm

```
      SUBROUTINE SUB(X,Y,Z)
      IF (X)4,6,6
    4 D=Y(X,Z**2)
      :
      :
    6 RETURN
      END
```

In diesem Beispiel wird der Unterprogrammname MULT als Argument im Unterprogramm SUB benutzt. Der Unterprogrammname MULT wird dem Formalparameter Y in der gleichen Weise wie A und C den Formalparametern X und Z zugewiesen. Das Unterprogramm MULT wird im Unterprogramm SUB nur dann aufgerufen und ausgeführt, wenn der Wert von A negativ ist.

Beispiel 2

```
      :
CALL SUB(A,B,MULT(C,D),37)
      :
```

Erklärung

In diesem Fall wird eine Externanweisung nicht benötigt, da das Unterprogramm mit dem Namen MULT nicht als Argument auftritt; es wird ausgeführt und erst sein Ergebnis wird zum Argument.

Die Namen aller in der FORTRAN-Bibliothek enthaltenen Unterprogramme, die in Anhang B aufgeführt sind, können als Argumente für ein Unterprogramm benutzt werden und können daher in einer Externanweisung auftreten.

7.4. Explicite Spezifikationsanweisungen

Form	Elemente
Typ *sa*s_1(k_1)/x_1/, b*s_2(k_2)/x_2/,..., z*s_n(k_n)/x_n/	Typ ist INTEGER, REAL, COMPLEX oder LOGICAL. *s,s_1,*s_2,...,*s_n können wahlweise angegeben werden; s,si sind zulässige Längen des entsprechenden Typs.

Form	Elemente
	$\underline{a},\underline{b},\ldots,\underline{z}$ stellen Variablen-, Feld- oder Funktionsnamen dar (s.Abschnitt 8.4.4. "Funktionsunterprogramme"). $(\underline{k}_1)$, $\underline{k}_2),\ldots,(\underline{k}_n)$ können wahlweise angegeben werden. Jedes $\underline{k}_i$ besteht aus 1 bis 7 vorzeichenlosen, ganzzahligen Konstanten, die durch Kommas voneinander getrennt sind und die Maximalwerte der Indices des Feldes darstellen. Jedes $\underline{k}_i$ kann nur dann eine vorzeichenlose, ganzzahlige Variable sein, wenn die Typenvereinbarung in einem Unterprogramm enthalten ist.
	$/\underline{x}_1/,/\underline{x}_2/,\ldots,/\underline{x}_n/$ können wahlweise angegeben werden. Sie stellen Anfangswerte dar.

Die expliciten Spezifikationsanweisungen erklären den Typ (ganzzahlig, reell, komplex oder Boolesch) einer bestimmten Variablen oder eines Feldes durch ihren Namen statt durch ihren Anfangsbuchstaben. Dies unterscheidet sie von den anderen Verfahren zur Festlegung des Type einer Variablen oder eines Feldes (d.h. von der üblichen FORTRAN-Konvention und der IMPLICIT-Anweisung). Zusätzlich kann die Information über die Abmessungen von Feldern (Angabe der Dimension), die für die Speicherzuordnung notwendig ist, in die Anweisung mit aufgenommen werden. Erscheint diese Information nicht innerhalb einer expliciten Spezifikationsanweisung, so muß sie in eine Feldanweisung oder eine Speicherblockanweisung aufgenommen werden (s. Abschnitt 7.6. "Feldanweisung" und 7.7. "Speicherblockanweisung").

Variablen oder Feldern können Anfangswerte durch Verwendung von /$\underline{x}_n$/ zugeordnet werden, wobei $\underline{x}_n$ eine Konstante oder eine Liste von durch Kommas getrennten Konstanten ist. Diese Folge von Konstanten kann in der Form "r* Konstante" geschrieben werden, wobei r eine wahlweise anzugebende, vorzeichenlose ganze Zahl ist, die Wiederholungskonstante genannt wird. Für die $\underline{x}_n$ kann jede beliebige Art von Konstanten geschrieben werden, die in einer Ergibtanweisung von der Form $v_n = x_n$ zugelassen ist, wobei v_n die mit einem Anfangswert zu versehende Variable ist. Außerdem kann jeder Variablen als Anfangswert ein Literal zugeordnet werden.

In derselben Weise, wie die Anweisung IMPLICIT die normale FORTRAN-Konvention unterdrückt, setzen explicite Spezifikationsanweisungen die Wirkung von IMPLICIT-Anweisungen und der normalen FORTRAN-Konvention außer Kraft. Wird die Angabe einer Länge, d.h. *$\underline{s}$, weggelassen, so wird je Typ die Standardlänge angenommen.

Beispiel 1

```
INTEGER*2 FELD/76/,WERT
```

Erklärung

Diese Anweisung legt fest, daß die Variablen FELD und WERT ganzzahlig sind, und daß für jede zwei Speicherplätze reserviert werden. Außerdem erhält die Variable FELD den Anfangswert 76.

Beispiel 2

```
COMPLEX C,D/(2.1,4.7)/,E*16
```

Erklärung

Diese Anweisung legt fest, daß die Variablen C, D und E komplex sind. Da keine explicite Längenangabe erfolgt ist, wird die Standardlänge angenommen. Folglich werden sowohl für C als auch für D acht Speicherplätze reserviert (je 4 für den reellen und den imaginären Teil). D erhält als Anfangswert (2.1,4.7). Außerdem werden 16 Speicherplätze für die Variable E reserviert. Wenn also eine Länge explicit angegeben ist, ersetzt sie die Standardlänge.

Beispiel 3

```
REAL*8 AUS,EIN,WERT*4,FELD(5,5)
```

Erklärung

Diese Anweisung legt fest, daß die Variablen AUS, EIN, WERT und das Feld mit dem Namen FELD reell sind. Außerdem definiert sie die Größe des Feldes FELD. Den Variablen AUS und EIN sind je acht Speicherplätze zugeordnet. Für die Variable WERT werden vier Speicherplätze reserviert. Für das Feld mit dem Namen FELD sind 200 Speicherplätze vorgesehen (acht für jede Variable des Feldes). Es ist zu beachten, daß sich eine dem Typ beigefügte Längenangabe - beispielsweise REAL*8 - auf jede Variable in der Anweisung bezieht, es sei denn, daß sie explicit außer Kraft gesetzt wird (wie im Fall von WERT*4). Der Typ der Elemente eines Feldes ist der Typ, der für den Feldnamen festgelegt ist.

Beispiel 4

```
REAL A(5,5)/20*6.9E2,5*1.0/,B(100)/100*0.0/
```

Erklärung

Diese Anweisung erklärt die Größe der Felder A und B sowie ihren Typ (reell). Für das Feld A werden 100 Speicherplätze (vier für jede Variable des Feldes), für das Feld B werden 400 Speicherplätze reserviert (vier für jede Variable des Feldes). Außerdem wird den ersten 20 Variablen des Feldes A der Anfangswert 6.9E2 zugewiesen. Die letzten fünf Variablen erhalten den Anfangswert 1.0. Alle 100 Variablen des Feldes B erhalten anfänglich den Wert 0.0.

7.5. Anweisung für doppelte Genauigkeit

Form	Elemente
DOUBLE PRECISION a,b,c...	a,b,c... sind Namen von Funktionen, Variablen oder Feldern. Für Felder können die Dimensionen angegeben werden.

Die Anweisung für doppelte Genauigkeit gibt explicit an, daß die Variablen a,b,c,... vom Typ REAL*8 sind. Sie setzt die Wirkung sowohl der üblichen FORTRAN-Konvention als auch der IMPLICIT-Anweisung außer Kraft.

7.5.1. Veränderliche Indexgrenzen

Wie aus den vorausgehenden Beispielen ersichtlich ist, wird der Höchstwert für jeden Feldindex durch eine Zahl festgelegt. Diese Höchstwerte sind die absoluten Dimensionen eines Feldes und können niemals geändert werden. Wenn jedoch ein Feld in einem Unterprogramm benutzt wird (s. Abschnitt "Prozeduren"), braucht innerhalb des Unterprogramms die Größe des Feldes nicht durch einen numerischen Wert explicit erklärt zu werden. Die explicite

Spezifikationsanweisung, die in einem Unterprogramm erscheint, kann vielmehr ganzzahlige Variablen enthalten, die die Größe des Feldes festlegen. Wird das Unterprogramm aufgerufen, so werden diesen ganzzahligen Variablen ihre Werte von dem aufrufenden Programm zugewiesen. Auf diese Weise ist die Größe eines Feldes, das in einem Unterprogramm verwendet wird, veränderlich und kann bei jedem Aufruf des Unterprogramms wechseln.

Die absoluten Grenzen eines Feldes müssen in einem aufrufenden Programm festgelegt werden. Die veränderlichen Indexgrenzen des Feldes, die in einem Unteprogramm verwendet werden, sollten kleiner oder gleich seinen absoluten Indexgrenzen gemäß der Festlegung in dem aufrufenden Programm sein. Veränderliche Indexgrenzen dürfen nicht benutzt werden, wenn das Feld in einem gemeinsamen Speicherblock liegt (s. Abschnitt 7.7. "Speicherblockanweisung").

Das folgende Beispiel zeigt den Gebrauch von veränderlichen Indexgrenzen.

Beispiel

Aufrufendes Programm

```
   :
REAL*8 A(5,5)
   :
CALL MAPMY(...,A,2,3,...)
   :
```

Unterprogramm

```
SUBROUTINE MAPMY(...,R,L,M,...)
        :
REAL*8...,R(L,M),...
        :
DO 100 I=1,L
        :
```

Erklärung

Die Anweisung REAL*8 A(5,5), die in dem aufrufenden Programm enthalten ist, erklärt die absoluten Grenzen des Feldes A. Wird das Unterprogramm mit dem Namen MAPMY aufgerufen, so nimmt der Formalparameter R den Feldnamen A und die Formalparameter L und M die Werte 2 und 3 an. Die indizierten Variablen des Feldes A, die in dem aufrufenden Programm auftreten, belegen feste Speicherbereiche in folgender Reihenfolge:

A(1,1)	A(2,1)	A(3,1)	A(4,1)	A(5,1)
A(1,2)	A(2,2)	A(3,2)	A(4,2)	A(5,2)
A(1,3)	A(2,3)	A(3,3)	A(4,3)	A(5,3)
A(1,4)	A(2,4)	A(3,4)	A(4,4)	A(5,4)
A(1,5)	A(2,5)	A(3,5)	A(4,5)	A(5,5)

Die indizierte Variable A(1,2) bezieht sich in dem aufrufenden Programm folglich auf die sechste Variable des Feldes A. In dem Unterprogramm MAPMY dagegen bezieht sich die indizierte Variable A(1,2) auf die dritte Variable des Feldes A, nämlich A(3,1). Dies ist eine Folge der Tatsache, daß die Grenzen des Feldes A im Unterprogramm anders erklärt sind als im aufrufenden Programm.

Wären die absoluten Grenzen im aufrufenden Programm dieselben wie die veränderlichen Grenzen im aufgerufenen Unterprogramm, so würden sich die indizierten Variablen A(1,1) bis A(5,5) in dem Unterprogramm immer auf dieselben Speicherbereiche beziehen, die in dem aufrufenden Programm durch die indizierten Variablen A(1,1) bis A(5,5) gekennzeichnet sind.

Anstelle der Zahlen 2 und 3, die zu neuen Grenzen des formalen Feldes R geworden sind, hätten auch Variablen in die Argumentenliste des aufrufenden Programms aufgenommen werden können. Es sei beispielsweise angenommen, daß die folgende Anweisung in dem aufrufenden Programm enthalten ist:

```
CALL MAPMY(...,A,I,J,...)
```

Solange die Werte von I und J vorher festgelegt wurden, können die Argumente Variable sein. Außerdem können sich variable Indexgrenzen über mehr als eine Stufe von Unterprogrammen auswirken. Wenn beispielsweise innerhalb des Unterprogramms MAPMY ein weiteres Unterprogramm aufgerufen würde, kann Information über die Grenzen von A auch an dieses Unterprogramm weitergegeben werden.

Formale Parameter - z.B. L und M - können als Feldgrenzen nur für ein Funktions- oder ein Subroutinenunterprogramm verwendet werden. Für jedes derartige Feld müssen der Feldname und alle für die Festlegung der Grenzen des Feldes verwendeten Variablen als Argumente derselben Funktions-, Subroutinen- oder Einsprunganweisung erscheinen.

7.6. Feldanweisung

Form	Elemente
DIMENSION $\underline{a}_1(\underline{k}_1),\underline{a}_2(\underline{k}_2),\ldots,\underline{a}_n(\underline{k}_n)$	$\underline{a}_1,\underline{a}_2,\ldots,\underline{a}_n$ sind Feldnamen. $\underline{k}_1,\underline{k}_2,\ldots,\underline{k}_n$ bestehen aus je 1 bis 7 vorzeichenlose, ganzzahligen Konstanten, die durch Kommas voneinander getrennt sind, und den Höchstwert für jeden Index des Feldes darstellen. $\underline{k}_1$ bis $\underline{k}_n$

Form	Elemente
	können nur dann ganzzahlige Variablen sein, wenn sie in einer Feldanweisung innerhalb eines Unterprogramms auftreten.

Die Information, die für Zuordnung von Speicherbereichen für in einem Primärprogramm benutzte Felder notwendig ist, kann durch eine Feldanweisung gegeben werden. Die folgenden beiden Beispiele zeigen, wie diese Information zu formulieren ist.

Beispiele

```
DIMENSION A(10),FELD(5,5,5,5,5),LISTE(10,100)
DIMENSION B(25,50),TAB(25,25,25)
```

7.7. Speicherblockanweisung

Form	Elemente
COMMON $\underline{a}(\underline{k}_1),\underline{b}(\underline{k}_2),$ $\ldots/\underline{r}/\underline{c}(\underline{k}_3),\underline{d}(\underline{k}_4)\ldots$	$\underline{a},\underline{b},\underline{c},\underline{d},\ldots$ sind Variablen- oder Feldnamen.
	$\underline{k}_1,\underline{k}_2,\underline{k}_3,\underline{k}_4,\ldots$ können wahlweise angegeben werden. Sie sind Listen von maximal 7 vorzeichenlosen, ganzzahligen Konstanten, die durch Kommas voneinander getrennt sind und den Höchstwert des entsprechenden Index des Feldes darstellen.
	$/\underline{r}/$ stellt den Namen des gemeinsamen Speicherblockes dar und besteht aus 1 bis 6

Form	Elemente
	alphanumerischen Zeichen, von denen das erste ein alphabetisches Zeichen sein muß. Diese Namen müssen immer in Schrägstriche (/) eingeschlossen sein.

Die Speicherblockanweisung kann benutzt werden, um Informationen über Indexgrenzen zu liefern. Veränderliche Indexgrenzen sind nicht zulässig.

Variablen oder Felder, die in einem aufrufenden Programm oder einem Unterprogramm auftreten, können durch Verwendung der Speicherblockanweisung dieselben Speicherbereiche wie die Variablen oder Felder eines anderen Unterprogramms belegen. Enthält beispielsweise ein Programm die Anweisung

```
COMMON TAB
```

und ein zweites Programm die Anweisung

```
COMMON LISTE
```

so belegen die Variablen TAB und LISTE dieselben Speicherbereiche. Enthält das Hauptprogramm die Anweisungen

```
REAL A,B,C
COMMON A,B,C
```

und ein Unterprogramm die Anweisungen

```
REAL X,Y,Z
COMMON X,Y,Z
```

so belegt A denselben Speicherbereich wie X, B denselben Speicherbereich wie Y und C denselben Speicherbereich wie Z.

Das folgende Beispiel sei betrachtet:

Beispiel 1

Aufrufendes Programm

```
        :
COMMON A,B,C,R(100)
REAL A,B,C
INTEGER R
        :
CALL MAPMY(...)
```

Unterprogramm

```
SUBROUTINE MAPMY(...)
        :
COMMON X,Y,Z,S(100)
REAL X,Y,Z
INTEGER S
        :
```

Erklärung

Die Anweisung COMMON A,B,C,R(100) im aufrufenden Programm bewirkt, daß 412 Speicherplätze (vier Speicherplätze je Variable) in folgender Reihenfolge reserviert werden:

Anfang des gemeinsamen Bereichs

A 4 Plätze	B 4 Plätze	C 4 Plätze
R(1) 4 Plätze	...	R(100) 4 Plätze

Speichereinteilung

Die Anweisung COMMON X,Y,Z,S(100) bewirkt dann, daß die Variablen X,Y,Z und S(1)...S(100) denselben Speicherbereich belegen wie die Variablen A,B,C und R(1)...R(100).

Aus obigem Beispiel ist zu ersehen, daß die Speicherblockanweisung einem wichtigen Zweck dient, nämlich der impliciten Übertragung von Daten vom aufrufenden Programm in das Unterprogramm. Die Werte für X,Y,Z und S(1)...S(100) müssen also aufgrund der Tatsache, daß diese Variable denselben Speicherbereich wie die Variablen A,B,C und R(1) ...R(100) belegen, nicht in die Argumentliste der Aufrufanweisung aufgenommen werden.

Beispiel 2

Es sei angenommen, daß in einem Hauptprogramm und drei Unterprogrammen Speicherblockanweisungen wie folgt geschrieben wurden:

```
Hauptprogramm:      COMMON A,B,C
Unterprogramm 1:    COMMON D,E,F
Unterprogramm 2:    COMMON Q,R,S,T,U
Unterprogramm 3:    COMMON V,W,X,Y,Z
```

Es sei ferner angenommen, daß die Längen der obigen Variablen so festgelegt wurden, daß der gemeinsame Bereich wie folgt belegt ist:

A 8 Plätze		B 4 Plätze		C 2 Plätze
D 8 Plätze		E 4 Plätze		F 2 Plätze
Q 4 Plätze	R 4 Plätze	S 2 Plätze	T 2 Plätze	U 2 Plätze
V 4 Plätze	W 4 Plätze	X 2 Plätze	Y 2 Plätze	Z 2 Plätze

In diesem Fall können sowohl die Variablen A, B und C als auch die Variablen D,E und F in ihren Programmen ebenso angesprochen werden wie die Variablen Q,R,S,T und U und die Variablen V,W,X,Y und Z. Außerdem können alle Programme C,F,U oder Z ansprechen. Es ist ferner möglich, sich wechselseitig auf D im Unterprogramm 1 und auf Q und R im Unterprogramm 2 zu beziehen. Solche Beziehungen sind in hohem Maße von den Daten abhängig und können in bestimmten Fällen vorteilhaft genutzt werden. Wenn beispielsweise D eine komplexe Variable und Q und R reelle Variablen sind, entsprechen Q und R dem reellen und dem imaginären Teil von D. Solche Bezugnahmen durch den Programmierer müssen jedoch immer für sich alleine betrachtet werden.

7.7.1. Gemeinsame Speicherblöcke mit und ohne Namen

In den vorausgehenden beiden Beispielen war der gemeinsame Speicherblock nicht durch Namen gekennzeichnet. Den in den Speicherblockanweisungen auftretenden Variablen wurden Speicherplätze relativ zum Anfang des unbenannten gemeinsamen Blockes zugeordnet. Variablen und Felder können jedoch in getrennte, gemeinsame Bereiche übertragen werden. Jedem dieser Bereiche oder Blöcke wird ein Name gegeben, der aus 1 bis 6 alphanumerischen Zeichen besteht. Das erste dieser Zeichen muß ein alphabetisches Zeichen sein. Blöcke mit gleichem Namen belegen auch den gleichen Speicherbereich.

Denjenigen Variablen, die in einen benannten, gemeinsamen Speicherblock übertragen werden sollen, geht der Name für den Block, eingeschlossen in Schrägstriche (/), voraus. Beispielsweise werden die Variablen A,B und C durch folgende Anweisung in dem gemeinsamen Speicherblock EIN abgesetzt:

```
COMMON/EIN/A,B,C
```

In einer Speicherblockanweisung können benannte Speicherblöcke vom unbenannten Speicherblock dadurch unterschieden werden, daß den Variablen im unbenannten Speicherblock zwei unmittelbar aufeinanderfolgende Schrägstriche (/) vorausgehen oder, wenn die Variablen am Anfang einer Speicherblockanweisung stehen, durch das Weglassen eines Blocknamens. Beispielsweise bewirkt die Anweisung

```
COMMON A,B,C/FELD/X,Y,Z//D,E,F
```

daß die Variablen A,B,C,D,E und F in dieser Reihenfolge in dem unbenannten Speicherblock gespeichert werden. Die Variablen X,Y,Z dagegen werden in dem Speicherblock FELD abgesetzt.

Die Eintragungen in den Speicherblockanweisungen eines Programms akkumulieren sich über ein Programm. Die beiden Speicherblockanweisungen

```
COMMON A,B,C/R/D,E/S/F
COMMON G,H/S/I,J/R/P//W
```

haben beispielsweise dieselbe Wirkung wie die eine Anweisung

```
COMMON A,B,C,G,H,W/R/D,E,P/S/F,I,J
```

Es sei angenommen, daß A,B,C,K,X und Y je vier Speicherplätze, daß H und G je acht und D und E je zwei Speicherplätze belegen.

Aufrufendes Programm

```
        :
COMMON H,A/R/X,D//B
        :
CALL MAPMY(...)
        :
```

Unterprogramm

```
SUBROUTINE MAPMY(...)
        :
COMMON G,Y,C/R/K,E
        :
```

Erklärung

In dem aufrufenden Programm bewirkt die Anweisung

```
COMMON H,A/R/X,D//B
```

daß 16 Speicherplätze (je vier für A und B sowie acht für H) in dem unbenannten Block in der folgenden Reihenfolge reserviert werden:

Anfang des unbenannten Blocks

H 8 Plätze	A 4 Plätze	B 4 Plätze
Fortsetzung des unbenannten Blocks		

Außerdem bewirkt sie, daß sechs Speicherplätze (vier für X und zwei für D) in dem Block mit dem Namen R in folgender Reihenfolge vorgesehen werden:

Anfang des Blocks mit dem Namen R

X 4 Plätze	D 2 Plätze
Fortsetzung des benannten Blocks	

Die Anweisung

```
COMMON G,Y,C/R/K,E
```

in dem Unterprogramm mit dem Namen MAPMY bewirkt, daß die Variablen G,Y und C denselben Bereich im unbenannten Speicherblock belegen wie H, A und B. Sie bewirkt außerdem, daß die Variablen K und E denselben Bereich im Block mit dem Namen R belegen wie X und D.

Die Länge eines gemeinsamen Speicherbereiches kann sich durch eine Äquivalenzanweisung erhöhen (s. Abschnitt 7.8. "Äquivalenzanweisung").

Der Benutzer sollte die Variablennamen in einem gemeinsamen Speicherblock in folgender Reihenfolge anordnen:

Komplex	(Länge 16)
Komplex	(Länge 8)
Reell	(Länge 8)
Reell	(Länge 4)
Ganzzahlig	(Länge 4)
Boolesch	(Länge 4)
Ganzzahlig	(Länge 2)
Boolesch	(Länge 1)

7.8. Äquivalenzanweisung

Form	Elemente
EQUIVALENCE (a,b,c,...), (d,e,f,...),...	a,b,c,d,e,f... sind Variablen, die indiziert werden können. Die Indizes können zwei Formen haben: Ist die Variable einfach indiziert, so bezieht sich ihr Index auf ihre Position innerhalb des Feldes (1.Variable, 25.Variable usw.). Ist die Variable mehrfach indiziert, so legen die Indexangaben die Position der Variablen in dem Feld in der gleichen Weise fest, wie dies bei Verwendung indizierter Variabler in einer arithmetischen Anweisung der Fall ist.

Die Äquivalenzanweisung ermöglicht die Steuerung der Zuordnung von Bereichen für die Datenspeicherung innerhalb eines einzelnen Programms oder Unterprogramms. Sie entspeicht der Speicherblockanweisung für die Steuerung der

Zuordnung von Bereichen für die Datenspeicherung zwischen verschiedenen Programmen. Insbesondere kann, sofern die Logik des Programms es zuläßt, der Speicherbedarf dadurch reduziert werden, daß zwei oder mehrere Variablen desselben oder verschiedenen Typs und derselben oder verschiedener Länge sich in dieselben Speicherbereiche teilen.

Beispiel 1

```
DIMENSION B(5),C(10,10),D(5,10,15)
EQUIVALENCE (A,B(1),C(5,3)),(D(5,20,2),E)
```

Erklärung

Diese Äquivalenzanweisung ordnet den Variablen A,B(1) und C(5,3) denselben Speicherbereich zu. Außerdem legt sie fest, daß den Variablen D(5,10,2) und E derselbe Speicherbereich zugewiesen wird. In diesem Fall beziehen sich die indizierten Variablen in der gleichen Weise auf die Position innerhalb eines Feldes, als hätte eine arithmetische Anweisung die betreffende Position angesprochen. Es ist zu beachten, daß Variablen oder Felder, die in einer Äquivalenzanweisung nicht erwähnt werden, Speicherbereiche für sich zugewiesen erhalten. Eine Äquivalenzanweisung darf sich nicht selbst und auch keiner vorher festgelegten Äquivalenz widersprechen. Beispielsweise ist eine weitere Äquivalenz von B(2) mit irgendeinem anderen Element des Feldes C, das nicht gleich C(6,3) ist, ungültig.

Beispiel 2

```
DIMENSION B(5),C(10,10),D(5,10,15)
EQUIVALENCE (A,B(1),C(25)),(D(100),E)
```

Erklärung

Diese Äquivalenzanweisung legt fest, daß der Variablen A, der ersten Variablen des Feldes B, nämlich B(1) und der

25. Variablen des Feldes C, nämlich C(5,3), dieselben Speicherbereiche zugeordnet werden sollen. Außerdem legt sie fest, daß D(100) (d.h. D(5,10,2)) und E der gleiche Speicherbereich zugeteilt wird. Zu beachten ist, daß die Wirkung der Äquivalenzanweisung in Beispiel 1 und 2 dieselbe ist.

Variablen, die durch eine Äquivalenzanweisung in einen gemeinsamen Speicherblock gebracht werden, können die Größe des Blocks, wie durch folgende Anweisungen gezeigt, erhöhen:

```
COMMON A,B,C
DIMENSION D(3)
EQUIVALENCE (B,D(1))
```

Diese Anweisungen bewirken, daß ein gemeinsamer Speicherblock aufgebaut wird, der die Variablen A,B und C aufnimmt. Die Äquivalenzanweisung hat zur Folge, daß die Variable D(1) denselben Speicherbereich belegt wie die Variable B, daß ferner D(2) denselben Speicherbereich belegt wie C und daß D(3) die Größe des gemeinsamen Speicherbereichs in folgender Weise erweitern würde:

```
A       (Anfang des gemeinsamen Speicherblocks)
B D(1)
C D(2)
  D(3)  (Ende des gemeinsamen Speicherblocks)
```

Da Felder in aufeinanderfolgenden, aufsteigenden Speicherplätzen gespeichert werden müssen, kann eine Variable einer anderen Variablen eines beliebigen Feldes nicht derart äquivalent gesetzt werden, daß das Feld vor den Beginn des gemeinsamen Blocks ausgedehnt wird.

Beispielsweise ist die folgende Äquivalenzanweisung unzulässig:

```
COMMON A, B, C
DIMENSION D(3)
EQUIVALENCE (B,D(3)),
```

weil sie erzwingen würde, daß D(1) der Variablen A in folgender Weise vorausgeht:

```
  D(1)   (Anfang des gemeinsamen Speicherblocks)
A D(2)
B D(3)
C        (Ende des gemeinsamen Speicherblocks)
```

Zwei Variablen in einem gemeinsamen Speicherblock oder in zwei verschiedenen Speicherblöcken können nicht äquivalent sein.

8. Organisatorische Anweisungen

8.1. Hauptprogramm

Ein Hauptprogramm ist eine Folge von Anweisungen und Kommentaren, die durch eine Endanweisung abgeschlossen wird und keine Funktions-, Subroutinen-, Einsprung- oder Blockdatenanweisung enthält, die alle in diesem Abschnitt besprochen werden.

Ein ausführbares Maschinenprogramm besteht aus genau einem Hauptprogramm und möglicherweise einem oder mehreren Unterprogrammen.

Um Flexibilität bei teilweiser, erneuter Übersetzung sicherzustellen, behandelt der FORTRAN-Compiler jede Programmeinheit (Hauptprogramm oder Unterprogramm) als eine unabhängige Einheit. Es ist daher möglich, mehrere Hauptprogramme und Unterprogramme als einen einzigen Satz von Primäranweisungen durch den Compiler übersetzen zu lassen. Diese Programmeinheiten können mit jedem anderen ausführbaren Programm in Beziehung gesetzt werden.

Die richtige Zusammenstellung von genau einem Hauptprogramm mit den gewünschten Unterprogrammen wird vom Binder vor dem Laden besorgt. Die Namen der gewünschten Programmeinheiten werden dem Binder vom Programmierer durch Steuerkarten mitgeteilt. Die Namen von Unterprogrammen werden in Funktions- oder Subroutinenanweisungen, die Namen von benannten, gemeinsamen Speicherblöcken in einem Blockdatenunterprogramm eingeführt.

Um einem Hauptprogramm durch eine Primäranweisung einen Namen zu geben, so daß es später durch die Steuerkarten für den Binder angesprochen werden kann, kann die Programmanweisung benutzt werden.

8.2. Programmanweisung

Form	Elemente
PROGRAM Name ⋮ END	Name ist eine beliebige Folge von 1 bis 6 alphanumerischen Zeichen. Das erste Zeichen muß ein alphabetisches Zeichen sein.

Enthält ein Hauptprogramm keine Programmanweisung, ist es in den Steuerkarten für den Binder durch den aus acht Zeichen bestehenden Namen MAINPROG anzusprechen.

8.3. Endanweisung

Form	Elemente
END	keine

Die Endanweisung ist eine nichtausführbare Anweisung, die für den Compiler das Ende eines Primärprogramms oder eines Primärunterprogramms definiert. Sie muß die letzte Anweisung eines jeden Programms oder Unterprogramms sein.

8.4. Prozeduren

Das Schreiben eines Programms, das an verschiedenen Punkten dieselbe Berechnung mit jeweils verschiedenen Daten notwendig macht, würde vereinfacht werden, wenn die erforderlichen Anweisungen nur einmal geschrieben zu werden bräuchten und dann beliebig auf sie verwiesen werden könnte. Jede spätere Bezugnahme auf diese Anweisungen sollte dieselbe Wirkung haben, als wären diese Instruktionen an der Stelle des Programms geschrieben, an der sie angesprochen wurden.

Ein allgemein gültiges Programm zur Berechnung der Quadratwurzel aus einer Zahl solle in die Programme übernommen

werden können, in denen Quadratwurzelberechnungen ausgeführt werden.

Die FORTRAN-Sprache trägt diesem Wunsch durch den Gebrauch von Prozeduren Rechnung. Es gibt vier Klassen von Prozeduren:

Formelfunktionen,
Funktionsunterprogramme,
Subroutinenunterprogramme und
eine Gruppe von Prozeduren,
die von FORTRAN zur Verfügung
gestellt werden (s. Anhang 3).

Die ersten beiden Klassen von Prozeduren werden Funktionen genannt. Funktionen unterscheiden sich von Subroutinenunterprogrammen dadurch, daß sie mindestens einen Wert an das aufrufende Programm zurückgeben. Im Gegensatz dazu brauchen Subroutinenunterprogramme keinen Wert an das aufrufende Programm zurückzuliefern.

8.4.1. Benennung von Prozeduren

Ein Prozedurname besteht aus 1 bis 6 alphanumerischen Zeichen, von denen das erste ein alphabetisches Zeichen sein muß. Sonderzeichen dürfen nicht benutzt werden. Der Typ einer Funktion kann genauso wie für Variablen festgelegt werden.

1. Typenvereinbarung einer Formelfunktion:
   Diese Erklärung kann auf eine von drei Weisen erreicht werden, nämlich durch die normale FORTRAN-Konvention, durch eine IMPLICIT-Anweisung oder durch eine explicite Spezifikationsanweisung. Es gelten also dieselben Regeln wie für die Typenvereinbarung von Variablen.

2. Typenvereinbarung von Funktionsunterprogrammen: Diese Vereinbarung kann in derselben Weise vorgenommen werden wie für Formelfunktionen. Außerdem kann der Typ (INTEGER, REAL, COMPLEX und LOGICAL) in der funktionsdefinierenden Anweisung festgelegt werden.

3. Typenvereinbarung eines Subroutinenunterprogramms: Der Typ eines Subroutinenunterprogramms kann nicht vereinbart werden, weil der Typ der Ergebnisse, die an das aufrufende Programm zurückgegeben werden, nur vom Typ der Variablennamen, die in der Argumentenliste des aufrufenden Programms auftreten und/oder dem Typ der impliciten Argumente in einem gemeinsamen Speicherblock abhängig ist.

8.4.2. Funktionen

Eine Funktion ist eine Anweisung für die Beziehung zwischen einer Reihe von Variablen. Für die Verwendung einer Funktion in FORTRAN sind folgende Bedingungen zu erfüllen:

1. Die Funktion muß definiert, d.h. die auszuführenden Rechnungen müssen festgelegt werden.

2. Die Funktion muß durch ihren Namen dort, wo sie im Programm benötigt wird, angesprochen werden.

8.4.2.1. Definition einer Funktion

In FORTRAN erfolgt die Definition einer Funktion in drei Schritten:

1. Der Funktion muß ein eindeutiger Name gegeben werden, mit dem sie aufgerufen werden kann (s.Abschnitt 8.4.1. "Benennung von Prozeduren").

2. Die Argumente der Funktionen müssen angegeben werden.

3. Das Verfahren zur Auswertung der Funktion muß angegeben werden.

Punkt 2 und 3 werden im einzelnen in den Abschnitten behandelt, die sich mit den spezifischen Unterprogrammen befassen (d.h. Formelfunktionen, Funktionsunterprogramme usw.).

8.4.2.2. Aufruf einer Funktion

Tritt der Name einer Funktion in einem arithmetischen oder Booleschen Ausdruck auf, so wird dadurch die Funktion aufgerufen. Auf diese Weise bewirkt das Auftreten eines Funktionsnamens mit den in Klammern angegeben Argumenten, daß die Rechnungen ausgeführt werden, wie sie bei der Definition der Funktion festgelegt wurden. Die sich ergebende Größe ersetzt in dem Ausdruck den Funktionsaufruf und nimmt den Typ der Funktion an.

8.4.3. Formelfunktionen

Formelfunktionen werden durch eine einzige arithmetische oder Boolesche Ergibtanweisung innerhalb des Programms definiert, in dem sie auftreten. Beispielsweise definiert die Anweisung

```
FUNC(A,B)=3.*A+B**2.+X+Y+Z
```

die Formelfunktion FUNC, wobei FUNC der Name der Funktion ist und A und B die Argumente der Funktion sind.

Der Ausdruck auf der rechten Seite definiert diejenigen Rechnungen, die ausgeführt werden müssen, wenn die Funktion in einer arithmetischen Anweisung benutzt wird. Diese Funktion könnte in einer Anweisung in folgender

Weise benutzt werden:

```
C=FUNC(D,E)
```

Dieser Schreibweise ist die folgende gleichwertig:

```
C=3.*D+E**2.+X+Y+Z
```

Zu beachten ist die Korrespondenz zwischen A und B in der funktionsdefinierenden Anweisung sowie D und E in der arithmetischen Anweisung. Die Größen A und B, die in Klammern eingeschlossen dem Funktionsamen folgen, sind die Argumente der Funktionsnamen. Sie sind Formalparameter, für die die Größen D und E eingesetzt werden, wenn die Funktion in der arithmetischen Anweisung benutzt wird.

Form	Elemente
Name(P_1,P_2...,P_n)=Ausdruck	Name ist der Name der Funktion (s.Abschnitt 8.4.1. "Benennung von Prozeduren").
	P_1,P_2,...P_n sind - innerhalb derselben Anweisung - einzelne, voneinander unterschiedene Variablen. Es können bis zu 15 derartige Variablen auftreten. Sie sind die Argumente der Funktion.
	Ausdruck ist ein beliebiger arithmetischer oder Boolescher Ausdruck, der keine indizierten Variablen enthält. Formelfunktionen, die in diesem Ausdruck auftreten, müssen zuvor definiert worden sein.

Die Variablen, die im Ausdruck auf der rechten Seite erscheinen, können Argumente der Funktion sein.
Hinweis: Alle Definitionen von Formelfunktionen müssen der ersten ausführbaren Anweisung eines Programms vorausgehen.

Beispiele

Als Definitionen von Formelfunktionen zulässig:

```
SUM(A,B,C,D)=A+B+C+D
FUNC(Z)=A+X*Y*Z
AVG(A,B,C,D)=(A+B+C+D)/4
WURZ(A,B,C)=SQRT(A**2+B**2+C**2)
WERT(A,B)=.NOT.A.OR.B
```

Hinweis: Dieselben Formalparameter können in mehr als einer Definition einer Formelfunktion verwendet werden und können auch als Variable desselben Typs außerhalb der Definition der Formelfunktion benutzt werden.

Als Definitionen von Formelfunktionen unzulässig:

`SUBPRG(3,J,K)=3*I+J**3`	(Argumente müssen Variable sein)
`SOMEF(A(I),B)=A(I)/B+3`	(Argumente dürfen nicht indiziert sein)
`SUBPROGRAM(A,B)=A**2+B**2`	(Funktionsname ist länger als sechs Zeichen)
`3FUNC(D)=3.14*E`	(Funktionsname muß mit einem alphabetischen Zeichen beginnen)
`ASF(A)=A+B(1)`	(Indizierte Variable im Ausdruck auf der rechten Seite)

Als Aufrufe von Formelfunktionen zulässig:

```
NET = GROS - SUM (TAC,FICA,HOSP,MISC)
ANS = FUNC (RESULT)
GRADE = AVG (LAB, LECTUR,SUM (TEST1, TEST2, TEST3,
        TEST4), FACTOR)
```

8.4.4. Funktionsunterprogramme

Das Funktionsunterprogramm ist ein FORTRAN-Unterprogramm, das aus einer beliebigen Anzahl von Anweisungen besteht. Es ist ein unabhängig geschriebenes Programm, das immer dann ausgeführt wird, wenn sein Name in einem anderen Programm auftritt.

Form

```
Typ FUNCTION Name*s(P1,P2,...,Pn)
        :
    RETURN
        :
    END
```

Elemente

Typ kann wahlweise angegeben werden und ist INTEGER (ganzzahlig), REAL (reell), COMPLEX (komplex) oder LOGICAL (Boolesch).

Name ist der Name des Unterprogramms (s.Abschnitt 8.4.1. "Benennung von Prozeduren").

*s kann wahlweise angegeben werden, s stellt eine der zulässigen Längen des betreffenden Typs dar.

P_1, P_2,...,P_n sind formale, nichtindizierte Variablen, Felder oder Prozedurenamen. (es muß wenigstens ein Argument in der Liste der Argumente enthalten sein.) Ist ein Argument in Schrägstriche (/) eingeschlossen, so wird es mit seinem Namen und

Form	Elemente
	nicht mit seinem Wert übermittelt; nur einfache (nicht indizierte) Variablen werden über den Wert aufgerufen.

Außerdem kann einem Funktionsunterprogramm der Typ REAL*8 in folgender Weise zugewiesen werden:

DOUBLE PRECISION FUNCTION Name (a_1,a_2,...,a_n)

Da FUNCTION ein eigenes Unterprogramm ist, besteht keine Beziehung zwischen seinen Variablen und Anweisungsnummern und irgendeinem anderen Programm.

Das Funktionsunterprogramm kann jede FORTRAN-Anweisung mit Ausnahme einer Subroutinenanweisung, einer anderen Funktionsanweisung oder einer Blockdatenanweisung enthalten.

Das Funktionsunterprogramm kann eines oder mehrere seiner Argumente für die Rückgabe von Werten an das aufrufende Programm verwenden. Jedes so benutzt Argument wird selbstverständlich innerhalb des Unterprogramms definiert oder neu definiert, d.h. es kann auf der linken Seite einer arithmetischen Ergibtanweisung erscheinen.

Die Argumente des Funktionsunterprogramms (d.h. P_1,P_2...P_n) können als Namen von formalen Variablen angesehen werden, die zum Zeitpunkt der Ausführung durch die in dem Funktionsaufruf enthaltenen Aktualparameter ersetzt sind. Aktualparameter können sein:

eine Konstante beliebigen Typs, mit Ausnahme eines Literals,
eine indizierte oder nichtindizierte Variable beliebigen Typs,

ein Ausdruck,
ein Namen einer Namensliste,
die Anweisungsnummer einer Formatanweisung
(geschrieben als &n, wobei n die Nummer
der Formatanweisung ist)
der Name eines anderen Unterprogramms.

Im allgemeinen müssen die Aktualparameter hinsichtlich Anzahl, Reihenfolge und Typ den Formalparametern entsprechen. Ist ein Aktualparameter die Anweisungsnummer einer Formatanweisung, so muß der entsprechende Formalparameter innerhalb des Unterprogramms als Feld definiert sein. Ist der Aktualparameter der Name einer Namensliste, darf der Formalparameter kein Feld sein. Die Feldgrößen sollten ebenfalls gleich sein, es sei denn, veränderliche Indexgrenzen werden benutzt.

Wird ein Argument seinem Wert nach übertragen, so wird beim Eintritt in das Unterprogramm der Wert der formalen Variablen gleich dem Wert des Aktualparameters gesetzt. Ist der Formalparameter durch das Unterprogramm verändert worden, so wird bei Ausführung der Rücksprunganweisung der Wert des Aktualparameters gleich dem augenblicklichen Wert des Formalparameters gesetzt. Wird ein Argument dem Namen nach übertragen, stellen Formalparameter und Aktualparameter dieselbe Größe dar, so daß jede Änderung des Formalparameters sofort auch den Aktualparameter verändert.

Die Beziehung zwischen den Variablennamen, die als Argumente in dem aufrufenden Programm auftreten, und den Formalparametern, die als Argumente in einem Funktionsunterprogramm erscheinen, wird durch folgendes Beispiel veranschaulicht:

Beispiel 1

Aufrufendes Programm

```
      :
      :
A=SOMEF(B,C)
      :
      :
```

Funktionsunterprogramm

```
FUNCTION SOMEF(X,Y)
SOMEF=X/Y
RETURN
END
```

Erklärung

In Beispiel 1 wird der Wert der Variablen B des aufrufenden Programms im Unterprogramm als Wert für die formale Variable X verwendet. Der Wert von C wird als Wert des Formalparameters Y verwendet. Ist also B=10.0 und C=5.0, so ist A=B/C, d.h. gleich 2.0.

Dem Namen der Funktion muß innerhalb des Unterprogramms mindestens einmal ein Wert zugeordnet werden, z.B. durch Ausführung einer arithmetischen Ergibtanweisung mit dem Namen der Funktion auf der linken Seite des Gleichheitszeichens. Ist der Typ des Funktionsnamens nicht explicit in der Funktionsanweisung angegeben, kann er auf drei andere Arten festgelegt werden:

1. durch die normale FORTRAN-Konvention
2. durch eine IMPLICIT-Anweisung, oder
3. durch eine explicite Spezifikationsanweisung.

Beispiel 2

Aufrufendes Programm

```
      :
      :
ANS=ROOT1*CALC(X,Y,I)
      :
      :
```

Funktionsunterprogramm

```
FUNCTION CALC(A,B,C)
      :
      :
I=J*2
CALC=A**I/B
      :
      :
RETURN
END
```

Erklärung

In diesem Beispiel werden die Werte von X, Y und I in dem Funktionsunterprogramm als Werte für A, B und J verwendet. Der Wert von CALC wird berechnet und an das aufrufende Programm zurückgegeben, wo der Wert von ANS ermittelt wird. Die Variable I in der Argumentenliste von CALC im aufrufenden Programm ist nicht die Variable I, die im Unterprogramm auftritt.

Die folgenden beiden Beispiele zeigen, wie der Typ in der Funktionsanweisung explicit vereinbart wird.

Beispiel 3

```
REAL FUNCTION SOMEF (A,B)
          :
SOMEF=A**2+B**2
          :
RETURN
END
```

Beispiel 4

```
INTEGER FUNCTION CALC*2(X,Y,Z)
          :
CALC=X+Y+Z**2
          :
RETURN
END
```

Erklärung

Die Funktionsunterprogramme SOMEF und CALC in Beispiel 3 und 4 sind als reell von der Länge 4 bzw. als ganzzahlig von der Länge 2 vereinbart.

Wenn ein Formalparameter ein Feldname ist, muß eine geeignete Feldanweisung oder explicite Spezifikationsanweisung innerhalb des Funktionsunterprogramms auftreten. Kein Formalparameter darf in einer Äquivalenz- oder Anfangswertanweisung innerhalb des Funktionsunterprogramms auftreten, noch darf ihm in einer expliciten Spezifikationsanweisung ein Anfangswert zugewiesen werden. Wenn in einer Speicherblockanweisung ein Formalparameter auftritt, ist die einzige Wirkung, daß ein Bereich von derselben Größe im gemeinsamen Speicherblock reserviert wird. Aber das Argument wird nicht dort gespeichert. In jeder anderen Hinsicht verhält sich dieser Formalparameter wie jeder andere. Ein nicht als Feld erklärter Formalparameter, der in einer Lese- oder Schreibanweisung als Bezugnahme auf eine Namensliste auftritt, ist ein formaler Name einer Namensliste und kann auf keine andere Weise verwendet werden.

Rücksprung- und Endanweisung in einem Funktionsunterprogramm

Es ist zu beachten, daß alle vorangehenden Beispiele von Funktionsunterprogrammen sowohl eine End- als auch mindestens eine Rücksprunganweisung enthalten. Die Endanweisung kennzeichnet dem Compiler das Ende des Unterprogramms. Die Rücksprunganweisung steht am logischen Abschluß der Berechnung und gibt alle berechneten Werte sowie die Steuerung des Programmablaufs an das aufrufende Programm zurück.

In einem FORTRAN-Unterprogramm können mehr als eine Rücksprunganweisung enthalten sein.

Beispiel

```
      FUNCTION DAV(D,E,F)
      IF(D-E)10,20,30
10    A=D+2.0*E
        :
 5    B=F+2.0*E
        :
20    DAV=A+B**2
        :
      RETURN
30    DAV=F**2
        :
      RETURN
      END
```

8.4.5. Subroutinenunterprogramme

Die Subroutinenunterprogramme sind in vielerlei Hinsicht den Funktionsunterprogrammen ähnlich. Die Regeln für die Benennung von Funktions- und Subroutinenunterprogrammen sind dieselben. Beide erfordern eine Endanweisung. Beide enthalten Formalparameter. Wie das Funktionsunterprogramm ist das Subroutinenunterprogramm eine Folge von mehrfach verwendeten Rechnungen, im Gegensatz zum Funktionsunterprogramm braucht jedoch das Subroutinenunterprogramm keine Resultate an das aufrufende Programm zurückzugeben.

Das Subroutinenunterprogramm wird durch eine Aufrufanweisung aufgerufen, die aus dem Wort CALL, gefolgt von dem Namen des Unterprogramms und aus seinen in Klammern aufgeführten Argumenten besteht.

Form	Elemente
SUBROUTINE Name(P_1,P_2...,P_n ⋮ RETURN ⋮ END	Name ist der Name des Unterprogramms (s.Abschnitt 8.4.1. "Benennung von Prozeduren"). P_1,P_2...,P_n sind Argumente (es braucht jedoch kein Argument vorhanden zu sein.) Jedes verwendete Argument muß der Name einer nichtindizierten Variablen oder eines Feldes, der formale Name eines anderen Subroutinen- oder Funktionsunterprogramms oder das Zeichen * sein, wobei "*" einen Rücksprungpunkt kennzeichnet, der durch die Nummer einer Anweisung im aufrufenden Programm gegeben ist. Ist ein Argument in Schrägstriche (/) eingeschlossen, so wird es mit seinem Namen und nicht mit seinem Wert aufgerufen; es werden jedoch nur einfache (nichtindizierte) Variablen dem Wert nach übermittelt.

Da SUBROUTINE ein abgeschlossenes Unterprogramm ist, stehen die in ihm verwendeten Variablen und Anweisungsnummern in keiner Beziehung zu irgend einem anderen Programm. Der Name der Subroutine ist keine Variable des Unterprogramms und darf in keiner anderen Anweisung innerhalb des Subroutinenunterprogramms auftreten.

Ein Subroutinenunterprogramm kann jede beliebige FORTRAN-Anweisung mit Ausnahme einer Funktionsanweisung, einer anderen Subroutinenanweisung oder einer Blockdatenanweisung enthalten.

Die Argumente P_1, P_2,... P_n können als die Namen von Formalparametern angesehen werden, die zur Zeit der Ausführung des Unterprogramms durch die Aktualparameter aus der Aufrufanweisung hinsichtlich Anzahl, Reihenfolge und Typ den Formalparametern entsprechen (s. Abschnitt 5.10. "Aufrufanweisung"). Die Feldgröße sollte ebenfalls gleich sein, es sei denn, es werden veränderliche Indexgrenzen benutzt. Formalparameter können nicht in einer Äquivalenz- oder Anfangswertanweisung innerhalb des Unterprogramms auftreten, noch können ihnen Anfangswerte durch eine explicite Spezifikationsanweisung zugewiesen werden. Wenn der Name eines Formalparameters in einer Speicherblockanweisung auftritt, hat dies nur die Wirkung, daß ein Bereich derselben Größe im Speicherblock reserviert wird, jedoch wird das Argument dort nicht gespeichert. In jeder anderen Hinsicht hat dieser Formalparameter dieselbe Wirkung wie jeder andere auch. Ein nicht dimensionierter Formalparameter, der zur Bezugnahme auf eine Namenliste benutzt wird, ist ein formaler Name einer Namenliste und kann nur als solcher verwendet werden.

Wird ein Argument seinem Wert nach übertragen, so wird beim Eintritt in das Unterprogramm der Wert des Formalparameters gleich dem Wert des Aktualparameters gesetzt. Wurde der Formalparameter durch das Unterprogramm geändert, wird bei Ausführung der Rücksprunganweisung der Wert des Aktualparameters gleich dem laufendem Wert des Formalparameters gesetzt. Wird ein Argument seinem Namen nach übertragen, stellen Formalparameter und Aktualparameter dieselbe Größe dar. Jede Änderung des Formalparameters ändert daher auch den Aktualparameter.

Das Subroutinenunterprogramm kann eines oder mehrere seiner Argumente verwenden, um Werte an das aufrufende Programm zurückzugeben. Jedes so verwendete Argument wird selbstverständlich innerhalb des Unterprogramms definiert oder neudefiniert, d.h. es kann auf der linken Seite einer arithmetischen Ergibtanweisung auftreten.

Beispiel

Die Beziehung zwischen Variablennamen, die als Argumente in dem aufrufenden Programm benutzt werden und formalen Variablen, die als Argumente in dem Subroutinenunterprogramm verwendet werden, wird durch das folgende Beispiel illustriert. Der Zweck des Unterprogramms ist es, ein Feld direkt in ein anderes zu "kopieren".

Hauptprogramm	Subroutinen-Unterprogramm
DIMENSION X(100),Y(100)	SUBROUTINE COPY(A,B,N)
	DIMENSION A(100),B(100)
	DO 10 I=1,N
CALL COPY(X,Y,K)	10 B(I)=A(I)
	RETURN
	END

In diesem Beispiel werden die Felder X und Y dem Namen nach und die Variable K dem Wert nach übertragen.

8.4.5.1. Namen- und Wertaufruf bei Argumenten

Jede Variable oder indizierte Variable, die als Aktualparameter in der Argumentliste des aufrufenden Programms erscheint, kann ihrem Namen oder ihrem Wert nach übermittelt werden.

Ein Aktualparameter wird seinem Namen nach verwendet, wenn der zugeordnete Formalparameter in dem Unterprogramm in Schrägstriche (/) eingeschlossen ist, andernfalls dem Wert nach. Normalerweise wirkt sich die Art des Aufrufes nicht auf die Ergebnisse der Berechnung aus. Das folgende Beispiel zeigt jedoch einen Fall, wo sie doch einen Unterschied bedeutet.

Beispiel

Aufrufendes Programm

```
      .
      .
      .
   I=1
   J=1
   CALL SUB(I,I,J,J)
```

Unterprogramm

```
SUBROUTINE SUB(/K/,/L/,M,N)
K=K+1
L=L+1
M=M+1
N=N+1
RETURN
END
```

Erklärung

Nach Ausführung des Unterprogramms ist der Wert von I gleich 3 und der Wert von J gleich 2. Dies wird klarer, wenn anstelle der Aufrufanweisung die gleichwertigen arithmetischen Ergibtanweisungen eingesetzt werden:

```
      .
      .
      .
   I=1
   J=1
   M=J
   N=J
   I=I+1
   I=I+1
   N=M+1
   N=N+1
   J=M
   J=N
      .
      .
      .
```

(Zu beachten ist mögliche Abhängigkeit von der Reihenfolge der abschließenden Substitution.)

Im allgemeinen entstehen Unterschiede nur über die Argumentenliste durch Definition oder Neu-Definition eines Formalparameters mit Hilfe eines anderen Formalparameters oder eines Elementes in einem gemeinsamen Speicherblock. (Zu beachten ist, daß eine solche Zuordnung im ASA-FORTRAN nicht zugelassen ist.)

Alle Feldnamen, Namen aus Namenlisten und Unterprogrammnamen, die als Aktualparameter in dem aufrufenden Programm auftreten, werden dem Namen nach übertragen, selbst dann, wenn die Formalparameter, denen sie zugeordnet sind, nicht in Schrägstriche eingeschlossen sind.

Alle arithmetischen oder Booleschen Ausdrücke, die als Aktualparameter in dem aufrufenden Programm auftreten, werden zuerst ausgewertet und dann in einem temporären Speicherbereich abgestellt. Die Formalparameter, die dem temporären Speicherbereich zugeordnet sind, werden dem Wert nach angesprochen, wenn sie lediglich durch Kommas voneinander getrennt sind. Die Formalparameter werden dem Namen nach angesprochen, wenn sie in Schrägstrichen eingeschlossen sind.

8.5. Mehrfache Einsprungstellen eines Unterprogramms

Der standardmäßige (normale) Einsprung in ein Subroutinenunterprogramm von dem aufrufenden Programm aus wird durch eine Aufrufanweisung bewirkt, die sich auf den Unterprogrammnamen bezieht. Der standardmäßig Einsprung in ein Funktionsunterprogramm wird durch einen Funktionsaufruf in einem arithmetischen Ausdruck bewirkt. Der Einsprung erfolgt bei der ersten ausführbaren Anweisung nach der Subroutinen- oder Funktionsanweisung.

Es ist ferner möglich, in ein Subroutinen-(oder Funktions)unterprogramm durch eine Aufrufanweisung (oder einen Funktionsaufruf) einzuspringen, die (der) sich auf eine Einsprunganweisung in dem Unterprogramm bezieht. Der Einsprung erfolgt bei der ersten ausführbaren Anweisung nach der Einsprunganweisung. Alle Einsprungstellen innerhalb des Subroutinenunterprogramms definieren Namen von Subroutinenunterprogrammen und alle Einsprungstellen innerhalb eines Funktionsunterprogramms definieren Namen von Funktionsunterprogrammen.

Form	Elemente
ENTRY Name(P_1,P_2,...,P_n)	Name ist der Name eines Einsprungpunktes (s.Abschnitt 8.4.1. "Benennung von Prozeduren").
	P_1,P_2,...,P_n sind Formalparameter, die den Aktualparametern in einer Aufrufanweisung oder einem Funktionsaufruf entsprechen. (Sie brauchen bei einem Einsprung in ein Subroutinenunterprogramm nicht aufzutreten.) Ist ein Formalparameter in Schrägstriche eingeschlossen, wird er dem Namen und nicht dem Wert nach übertragen.

Einsprunganweisungen können nur in Funktions- oder Subroutinenunterprogrammen auftreten. In einem Unterprogramm dürfen Formalparameter erst dann in einer Anweisung auftreten, wenn sie als Formalparameter in einer

Funktions-, Subroutinen- oder Einsprunganweisung aufgetreten sind. Das folgende Beispiel ist zulässig:

```
SUBROUTINE SUB(X,Y,Z,I)
        :
        :
ENTRY SUB1(A,B)
        :
        :
C=A+B
        :
```

Wenn die Einsprunganweisung in einem Subroutinen- bzw. Funktionsunterprogramm auftritt und dadurch einen Subroutinen- bzw. Funktionsnamen definiert, sind die Formalparameter und die ihnen zugeordneten Aktualparameter allen Einschränkungen und Bedingungen unterworfen, die für die Argumente von Subroutinen- bzw. Funktionsunterprogrammen gelten. Tritt ein Formalparameter, in einer Formalparameterliste in Schrägstrichen eingeschlossen, in anderen Formalparameterlisten nicht in Schrägstrichen eingeschlossen auf, so entscheidet die erste Liste der Formalparameter darüber, ob das Argument dem Namen oder dem Wert nach übermittelt wird. Einsprunganweisungen in einem Funktionsunterprogramm müssen immer Funktionen des Typs vereinbaren, der durch die Funktionsanweisung festgelegt ist. Unabhängig davon, welche Einsprunganweisung benutzt wird, ist der Wert der Funktion durch die letzte ausgeführte Zuweisung eines Wertes zu dem Funktionsnamen in dem Funktionsunterprogramm bestimmt. Die Namen von Einsprungpunkten werden niemals benutzt, um dem aufrufenden Programm Ergebnisse zurückzugeben.

Ein Einsprung in ein Unterprogramm weist allen Bezugsgrößen im gesamten Unterprogramm die Eintragungen der betreffenden Argumentenliste als Anfangswert zu. Wenn ein Argument dem Wert nach übertragen wird und in der Argumentenliste des letzten Einsprungs in das Unterprogramm

enthalten war, und wenn der entsprechende Formalparameter durch das Unterprogramm geändert wurde, dann bewirkt die Ausführung der Rücksprunganweisung, daß der entsprechende Aktualparameter gleich dem augenblicklichen Wert des Formalparameters gesetzt wird. Aktualparameter, die in der Argumentenliste des letzten Einsprungs nicht enthalten waren, werden von der Ausführung der Rücksprunganweisung nicht betroffen. In gleicher Weise werden die Werte von Formalparametern, die nicht in der Argumentenliste enthalten sind, durch den Einsprung in das Unterprogramm nicht betroffen. Wird ein Argument dem Namen nach übertragen, so wird zur Zeit der Ausführung der Aufrufanweisung oder des Funktionsaufrufes der Formalparameter mit dem Aktualparameter identifiziert. Er bleibt mit demselben Aktualparameter identifiziert, bis er durch eine weitere Aufrufanweisung oder einen weiteren Funktionsaufruf geändert wird. Jedesmal, wenn der Formalparameter durch eine Anweisung im Unterprogramm definiert oder neudefiniert wird, wird auch der Aktualparameter, mit dem er identifiziert ist, definiert oder neudefiniert.

Einsprunganweisungen beeinflussen die Ablaufsteuerung während der normalen Ausführung eines Unterprogramms nicht. Die Reihenfolge, der Typ und die Anzahl von Argumenten brauchen zwischen der Subroutinen- oder Funktionsanweisung und den Einsprunganweisungen nicht übereinzustimmen, und auch die Einsprunganweisungen untereinander brauchen in dieser Hinsicht nicht übereinzustimmen. Jede Aufrufanweisung oder jeder Funktionsaufruf muß jedoch hinsichtlich Reihenfolge, Typ und Anzahl der Argumente mit der Subroutinen-, Funktions- oder Einsprunganweisung übereinstimmen, auf die sie sich bezieht. Es kann nicht in den Bereich einer Schleifenanweisung eingesprungen werden. Außerdem kann sich ein

Unterprogramm nicht selbst direkt oder über irgendeinen seiner Einsprungspunkte ansprechen.

Beispiel 1

Aufrufendes Programm

```
        :
1 CALL SUB1(A,B,C,D,E,F)
        :
2 CALL SUB2(G,H,P)
        :
3 CALL SUB3
        :
```

Unterprogramm

```
SUBROUTINE SUB1(U,V,W,X,Y,Z)
        :
U=B
        :
ENTRY SUB2(T,U,V)
        :
ENTRY SUB3
        :
END
```

Erklärung

In Beispiel 1 bewirkt die Ausführung der Anweisung 1 einen Einsprung nach SUB1, so daß die Abarbeitung mit der ersten ausführbaren Anweisung des Unterprogramms beginnt. Die Ausführung der Anweisung 2 oder 3 bewirkt ebenfalls einen Einsprung in das aufgerufene Programm zu der ersten ausführbaren Anweisung, die auf die Anweisung ENTRY SUB2(T,U,V) bzw. ENTRY SUB3 folgt.

Beispiel 2

Aufrufendes Programm

```
      :
   CALL SUB1(A,B,C,D,E,F)
      :
   CALL SUB2(G,&10,&20)
      :
   CALL SUB3(510,&20)
 5 Y=A+B
      :
10 Y=C+D
20 Y=E+F
      :
```

Unterprogramm

```
   SUBROUTINE SUB1(U,V,W,X,Y,Z)
   RETURN
   ENTRY SUB2(T,*,*)
   U=V*W+T
   ENTRY SUB3(*,*)
   X=Y**Z
50 IF(U-X)100,200,300
100 RETURN 1
200 RETURN 2
300 RETURN
   END
```

Erklärung

In obigem Beispiel 2 bewirkt der Aufruf von SUB1 lediglich, daß bestimmte Anfangsbedingungen hergestellt werden, und zwar werden die Werte der Formalparameter U,V, W,X,Y und Z den Werten der Aktualparameter A,B,C,D,I und F gleichgesetzt. Die nachfolgenden Aufrufe von SUB2 und SUB3 haben die Ausführung von verschiedenen Abschnitten des Unterprogramms SUB1 zur Folge. Im Anschluß daran wird je nach Ergebnis der arithmetischen IF-Anweisung mit der Nummer 50 zur Anweisung 10, 20 oder 5 des aufrufenden Programms zurückgesprungen. Es ist zu beachten, daß die Aufrufe von SUB2 und SUB3 den Wert der Aktualparameter A und D nicht verändern, da diese Argumente dem Wert nach übertragen werden und nicht Argumente dieser beiden Einsprungpunkte sind. Die Formalparameter U und X werden dagegen durch die Ergibtanweisungen, die den Einsprungpunkten SUB2 und SUB3 folgen, neu definiert. Ein Einsprung am Punkt SUB3 ohne vorherigen Einsprung am Punkt SUB2 würde den Wert von U gleich dem Wert lassen, den A in dem aufrufenden Programm zur Zeit des Einsprungs nach SUB1 hatte.

8.5.1. Zusätzliche Regeln für die Benutzung der Einsprunganweisung

1. Eine Aufrufanweisung kann Aktualparameter, die dem Wert nach übermittelt werden, nur dann verändern, wenn sie explicite Argumente der Aufrufanweisung sind; sie kann nicht jene beeinflussen, die durch eine vorhergehende Aufrufanweisung einen Anfangswert zugewiesen bekommen haben. Eine Aufrufanweisung kann jedoch Aktualparameter verändern, die durch eine vorhergehende Aufrufanweisung über den Namen aufgerufen worden sind. Die Aktualparameter mußten

jedoch entweder Variable, indizierte Variable oder Feldnamen gewesen sein. Es können sich unvorhersehbare Resultate einstellen, wenn es Ausdrücke allgemeiner Art waren.

2. Wenn der Name eines Feldes mit veränderlichen Grenzen oder einer seiner veränderlichen Indexgrenzen in der Argumentenliste einer Einsprunganweisung auftritt, müssen dieser Feldname und alle seine veränderlichen Indexgrenzen in der Argumentenliste enthalten sein.

3. Das Auftreten einer Einsprunganweisung ändert nicht die Regeln für die Anordnung von Formelfunktionen in Unterprogrammen. Formelfunktionen können einer Einsprunganweisung nur dann folgen, wenn sie der ersten ausführbaren Anweisung vorausgehen, die der Subroutinen- oder Funktionsanweisung folgt.

4. Der Name eines Einsprungpunktes kann in einer Externanweisung in derselben Weise auftreten wie ein Funktions- oder Subroutinenname.

8.5.2. FORTRAN-Unterprogramme

FORTRAN stellt dem Programmierer eine Bibliothek von allgemein benutzbaren Funktions- und Subroutinenunterprogrammen zur Verfügung. Sie besteht aus drei Arten von Prozeduren:

1. Mathematische Funktionen

   a) entsprechend den Funktionsunterprogrammen (abgeschlossene Funktionen),

   b) entsprechend den arithmetischen Operationen (offene Funktionen).

2. Unterprogramme, die den Zustand von Pseudo-Maschinenindikatoren testen - definiert als Subroutinenunterprogramme.

3. Die drei Unterprogramme EXIT, DUMP und PDUMP - ebenfalls als Subroutinenunterprogramme definiert.

Die Subroutine EXIT beendet die Ausführung eines Programms. DUMP bewirkt einen Speicherabzug und beendet im Anschluß daran die Ausführung des Programms. PDUMP bewirkt einen Speicherabzug und setzt darauf die Ausführung des Programms fort.

Numerische Konstanten oder Variablen, die als Argumente von mathematischen Funktionen benutzt werden, werden in den Typ umgewandelt, der durch die Funktion verlangt wird. Den Funktionen kann auch ein höherer Typ zugeordnet werden, ähnlich wie bei der Festlegung des Typs von Ausdrücken.

Die gesamte Bibliothek, zusammen mit den zugehörigen Definitionen für jedes Unterprogramm, zeigt Anhang 2.

8.6. Blockdatenunterprogramm

Um Daten in einen gemeinsamen Speicherblock einzusetzen, muß ein eigenes Unterprogramm geschrieben werden. Dieses Unterprogramm besteht nur aus Anfangswert-, Speicherblock-, Feld-, Äquivalenz- und Typanweisungen, die den zu definierenden Daten zugeordnet sind. Die Daten können durch das Blockdatenunterprogramm nur in benannte Speicherblöcke eingesetzt werden.

Form	Elemente
BLOCK DATA	keine
:	
END	

1. Das Blockdatenunterprogramm darf keine ausführbaren Anweisungen enthalten.

2. Die erste Anweisung des Blockdatenunterprogramms muß die Blockdatenanweisung sein.

3. Alle Elemente eines gemeinsamen Speicherblocks müssen in der Speicherblockanweisung aufgeführt sein, selbst wenn sie nicht alle in der Anfangswertanweisung auftreten. So tritt die Variable A in der Speicherblockanweisung des folgenden Beispiels in keiner anderen Anweisung auf:

```
BLOCK DATA
COMMON/ELN/C,A,B/RMG/Z,Y
REAL B(4)/1.0,1.2,2*1.3/,Z*8(3)/3*7.64980825D0/
COMPLEX C/(2.4,3.760)/
```

4. Daten können durch ein einziges Blockdatenunterprogramm in mehr als einen gemeinsamen Speicherblock eingesetzt werden. Jeder derartige gemeinsame Speicherblock wird vom Binder als getrennter Modul behandelt.

8.7. Anfangswertanweisung

Form	Elemente
DATA $\underline{k}_1/\underline{d}_1/,\underline{k}_2/\underline{d}_2/,\ldots,\underline{k}_n/\underline{d}_n/$	$\underline{k}_1,\ldots,\underline{k}_n$ sind Listen von Variablen, indizierten Variablen - wobei die Indizes ganzzahlige Konstanten sein müssen - oder Feldern. $\underline{d}_1,\ldots,\underline{d}_n$ sind Listen von Konstanten und/oder Wiederholungskonstanten der

Form	Elemente
	der Form j*c, wobei j eine ganzzahlige und c eine beliebige Konstante ist. Jede Konstante c kann wahlweise mit Vorzeichen versehen sein und ganzzahlig, reell oder komplex sein;bzw.ein Literal oder eine Boolesche Konstante sein. Die einzelnen Listeneintragungen werden durch Kommas voneinander getrennt.

Eine Anfangswertanweisung wird verwendet, um Variablen und Feldern Anfangswerte zuzuordnen. Es muß eine 1:1-Korrespondenz bestehen zwischen den Variablen, die durch die k's und den Konstanten, die durch d's dargestellt werden. Jeder Feldname repräsentiert alle Feldelemente in der Reihenfolge, in der das Feld gespeichert ist. Jede Wiederholungskonstante j*c stellt j Konstanten dar, die sämtlich gleich c sind. Wenn jedoch die letzte Eintragung in der Liste der k's ein Feldname ist, ist es zulässig, weniger Konstanten in dem entsprechenden d zu haben, als zur Auffüllung des Feldes notwendig wären. Wenn eine Konstante einer Variablen v entspricht, kann c jede Form annehmen, die in der Ergibtanweisung v=c zugelassen wäre, oder c kann ein Literal sein.

Beispiel 1

```
DIMENSION D(5,10)
DATA A,B,C/5.0,6.1,7.3/,D/25*1.0/
```

Erklärung

Die Anfangswertanweisung in Beispiel 1 legt fest, daß die Variablen A, B und C die Anfangswerte 5.0, 6.1 und

7.3 erhalten. Außerdem bewirkt sie, daß die ersten 25 Variablen des Feldes D den Anfangswert 1.0 erhalten.

Beispiel 2

```
DIMENSION A(5),B(3,3),L(4)
DATA A/5*1.0/,B/9*2.0/,L/4*.TRUE./,C/'VIER'/,E,F/2*10.0/
```

Erklärung

Die Anfangswertanweisung in Beispiel 2 legt fest, daß alle Variablen in den Feldern A und B den Anfangswert 1.0 bzw. 2.0 erhalten sollen. Alle Booleschen Variablen in dem Feld L erhalten als Anfangswert den Wert .TRUE.. (Die Buchstaben T bzw. F können als Abkürzung für .TRUE. bzw. .FALSE. verwendet werden.) Außerdem wird der Variablen C als Anfangswert das Literal VIER und den Variablen E und F der Wert 10.0 zugewiesen.

Variablen oder Feldern in einem unbenannten Speicherblock können keine Anfangswerte zugewiesen werden. Sind sie in einem benannten Speicherblock enthalten, so kann die Zuweisung der Anfangswerte nur in einem Blockdatenunterprogramm erfolgen.

8.8. Namenlistenanweisung

Form

NAMELIST/x/a,b,...,c/y/d,e,...,
f/z/g,h,...,i

Elemente

x,y, und z,... sind Namen von Namenlisten.

a,b,c,d,... sind Namen von Variablen oder Feldern.

Die folgenden Regeln gelten für die Definition und den Gebrauch des Namens einer Namenliste:

1. Der Name einer Namenliste besteht aus 1 bis 6 alphanumerischen Zeichen, von denen das erste ein alphabetisches Zeichen sein muß.

2. Der Name einer Namenliste ist in Schrägstrichen (/) eingeschlossen. Die Liste der Variablen- oder Feldnamen, die dem Namen einer Namenliste zugeordnet ist, endet mit einem neuen in Schrägstrichen eingeschlossenen Namen oder mit dem Ende der Namenlistenanweisung.

3. Ein Variablen- oder ein Feldname kann zu einer oder mehreren Namenlisten einer Namenlistenanweisung gehören.

4. Bevor auf den Namen einer Namenliste bezogen werden kann, muß er in einer Namenlistenanweisung aufgeführt und damit definiert werden. Ein Name einer Namenliste kann auf diese Weise nur einmal definiert werden, er kann in dem Programm oder Unterprogramm nur als Name einer Namenliste verwendet werden.

5. END kann nicht als Name einer Namenliste verwendet werden.

Beispiel

```
A, I und L seien Feldnamen:
  .
  .       .
  .       .
NAMELIST/NAM1/A,B,I,J,L/NAM2/A,C,J,K
  .       .
  .       .
READ(5,NAM1)
  .       .
  .       .
  .
```

Erklärung

Die Leseanweisung des Beispiels bewirkt, daß die Sätze, die die Eingabedaten für die Variablen und Felder der angesprochenen Namenliste, nämlich NAM1, enthalten, von der Dateiengruppe 5 gelesen werden.

Wenn eine Leseanweisung den Namen einer Namenliste anspricht, werden Eingabedaten in der im folgenden beschriebenen Form von der bezeichneten Eingabedateiengruppe gelesen.

Eingabedaten

Eine Datengruppe, die durch eine einzige Leseanweisung, die sich auf eine Namenliste bezieht, gelesen werden soll, muß drei oder mehr Sätze enthalten. Das erste Zeichen eines jeden Satzes wird immer ignoriert. Der erste Satz einer Gruppe muß das Zeichen & in der zweiten Zeichenposition enthalten, diesem wiederum muß der Name der Namenliste ohne eingeschobene Zwischenräume unmittelbar folgen. Der letzte Satz einer Gruppe muß die Zeichen &END, anfangend in der zweiten Zeichenposition, ohne eingeschobene Zwischenräume enthalten.

Der Satz bzw. die Sätze zwischen dem ersten und dem letzten Satz enthalten einen oder mehrere Datenposten, die in einer der beiden folgenden Formen vorliegen:

1. Variablenname = Konstante
2. Feldname = Folge von Konstanten

"Variablenname" ist der Name einer einfachen oder indizierten Variablen. Indizes müssen ganzzahlige Konstanten sein. "Feldname" ist nicht indiziert. "Konstante" ist eine einzelne Konstante beliebigen Typs. "Folge von Konstanten" ist eine oder mehrere einzelne Konstanten und/

oder Wiederholkonstanten der Form "k*Konstante", wobei "k" eine vorzeichenlose ganze Zahl und "Konstante" eine Konstante beliebigen Typs ist, die k-mal wiederholt werden soll. Datenposten werden durch Kommas voneinander getrennt. Konstanten und/oder Wiederholkonstanten in einem Datenposten der zweiten Form werden durch Kommas getrennt.

Jeder Typ von Konstanten (ganzzahlig, reell, komplex und Boolesch) kann benutzt werden, wobei nur die Einschränkungen hinsichtlich des Typs zu beachten sind, die früher für arithmetische und Boolesche Ergibtanweisungen genannt worden sind. Boolesche Konstanten können entweder als T oder .TRUE. und als F oder .FALSE. geschrieben werden. Außerdem können in Verbindung mit Variablen- und Feldnamen beliebigen Typs Literale eingesetzt werden.

Die Sätze zwischen dem ersten und letzten Satz müssen eine Konstante, gefolgt von einem Komma als ihre letzten, von Zwischenraum verschiedenen Zeichen enthalten. Die Verwendung eines Kommas hinter der letzten Konstante, die dem &END-Satz vorausgeht, ist jedoch freigestellt. Zwischenräume können in beliebigem Umfang an jeder Stelle in den Sätzen verwendet werden, die die Datenposten enthalten.

Eine Gruppe von Daten, die durch eine Leseanweisung unter Bezugnahme auf eine Namenliste gelesen wird, darf nur Namen enthalten, die tatsächlich in der angesprochenen Namenliste enthalten sind. Die Datenposten können in beliebiger Reihenfolge stehen und eine beliebige Untermenge der vollständigen Liste darstellen. Einzelne Elemente von Feldern können als indizierte variable Datenposten eingelesen werden, während andere Elemente desselben Feldes ignoriert werden können.

Ein einem Feld zugeordneter Datenposten kann das gesamte Feld oder jeden beliebigen Anfangsteil desselben füllen. Die Folge von Konstanten wird in das Feld eingesetzt, wie weiter oben unter "Anordnung von Feldern im Speicher" beschrieben, wobei eine geforderte Wiederholung von Konstanten beachtet wird.

Beispiel

Es sei angenommen, daß L ein eindimensionales Feld ist. Der Index laufe von 1 bis 10.

```
                                  Spalte 2
                                     ↓
1. Datenkarte:                    &NAM1
2. Datenkarte:                    I(2,3)=5,J=4,
3. Datenkarte:                    A(3)=4.0,L=2,3,8*4,
      :                                 :
Letzte Datenkarte:                &END
```

Erklärung

Wenn dies Eingabedaten sind, die mit den weiter oben angegebenen Namenlisten- und Lesenanweisung eingelesen werden, dann geschieht folgendes:
Die erste Datenkarte wird gelesen und geprüft, um sicherzustellen, daß ihr Name mit dem Namen der durch die Leseanweisung angesprochenen Namenliste übereinstimmt. Wird dieser Name nicht gefunden, so wird bezüglich der nächsten Gruppe in der Namenlistenanweisung eingelesen.

Wird die zweite Datenkarte gelesen, werden die ganzzahligen Konstanten 5 und 4 in I(2,3) bzw. J eingesetzt. Wird die dritte Datenkarte gelesen, wird die reelle Konstante 4.0 in A(3) eingesetzt. Da L ein Feldname ist, wird das gesamte Feld mit den nachfolgenden Konstanten gefüllt. Folglich werden die ganzzahligen Konstanten 2

und 3 in L(1) bzw. L(2), die ganzzahlige Konstante 4 in L(3), L(4),...,L(10) eingesetzt.

Ausgabedaten

Bezieht sich eine Schreibanweisung auf eine Namenliste, so ist folgendes zu beachten:

1. Alle zu dem Namen der Namenliste gehörigen Variablen und Felder mit ihren Werten werden geschrieben, und zwar jeweils entsprechend ihrem Typ. Ein vollständiges Feld wird spaltenweise geschrieben.

2. Die Ausgabedaten werden so geschrieben, daß
   a) die Felder für die Daten groß genug sind, um alle gültigen Stellen aufzunehmen, und
   b) die ausgegebenen Daten durch eine Eingabeanweisung gelesen werden können, die sich auf den Namen der Namenliste bezieht.

Beispiel

A sei ein Feld von der Größe 3 x 3.

```
          :
NAMELIST/NAM1/A,B,I,D
WRITE(8,NAM1)
          :
```

Werden die Daten auf Lochkarten ausgestanzt, so ist deren Aufbau ab Spalte 2 folgender:

Werden die Daten auf Lochkarten ausgestanzt, so ist deren Aufbau ab Spalte 2 folgender:

Erste Ausgabekarte:
&NAM1
Zweite Ausgabekarte:
_____A=+3.456789E_11,_______-9.876543E-21,_______+1.234567E_20,
Dritte Ausgabekarte:
_______-1.000000E_00,_______+8.000000-01,_______-2.500000E-01,
Vierte Ausgabekarte:
_____-6.250000E-02,_______+6.250000E-01,_______-5.000000E_01,
Fünfte Ausgabekarte:
_____B=+3.141591E_00,_______I=_____10,_____D=+2.123987E_09
Sechste Ausgabekarte:
&END

8.9. Formatanweisung

Form	Elemente
nnnnn FORMAT (Folge von Formatschlüsseln)	nnnnn ist eine Anweisungsnummer (1 bis 5 Ziffern). Folge von Formatschlüsseln ist einer oder mehrere der hier zu erläuternden Formatschlüssel. Sie werden durch Kommas und/oder Schrägstriche (/) voneinander getrennt und können durch Verwendung von runden Klammern in Gruppen zusammengefaßt werden.

Die Formatanweisung wird in Verbindung mit der Lese- und Schreibanweisung benutzt, um die gewünschte Form der zu übertragenden Daten festzulegen. Die Form der Daten wird durch den Gebrauch verschiedener Formatschlüssel bestimmt. Die 12 Formatschlüssel sind: G,T,X,P, Literal,A,I,F,E,D,H und L. Jede in einer Formatanweisung benutzte Zahl mit Ausnahme einer Anweisungsnummer oder eines Literals muß kleiner als 256 sein.

Gebrauch der Formatanweisung

Dieser Abschnitt enthält allgemeine Informationen über die Formatanweisung. Die unten besprochenen Punkte werden durch nachfolgende Beispiele veranschaulicht.

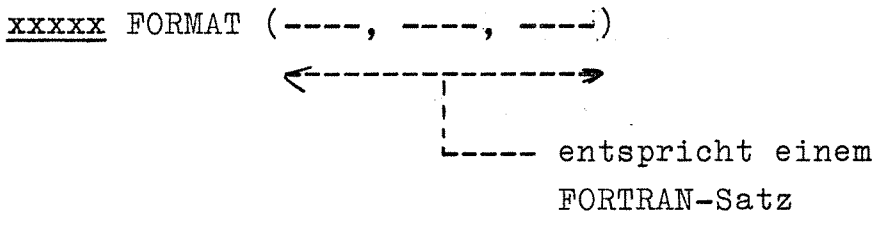
1. Formatanweisungen sind nicht ausführbar und können an beliebiger Stelle im Primärprogramm eingesetzt werden.

2. Eine Formatanweisung kann verwendet werden, um einen FORTRAN-Satz in folgender Weise zu definieren.

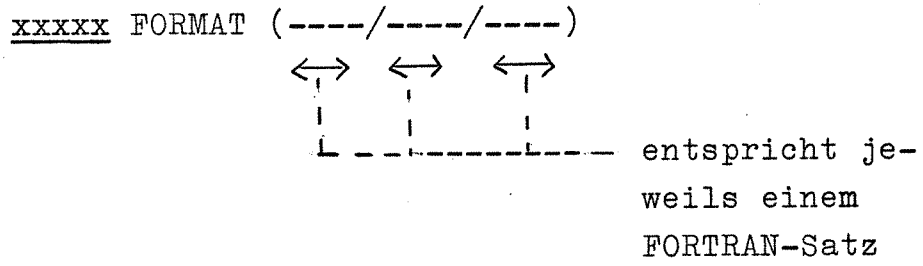
   a) Wenn keine Schrägstriche oder zusätzlichen Klammern innerhalb einer Formatanweisung auftreten, wird ein FORTRAN-Satz durch Beginn (Klammer auf) und Ende (Klammer zu) der Formatanweisung definiert.

      Es wird also ein neuer Satz bei Beginn der Formatsteuerung (Klammer auf) gelesen. Ein neuer Satz ist geschrieben, wenn die Formatsteuerung beendet ist (Klammer zu).

Beispiel

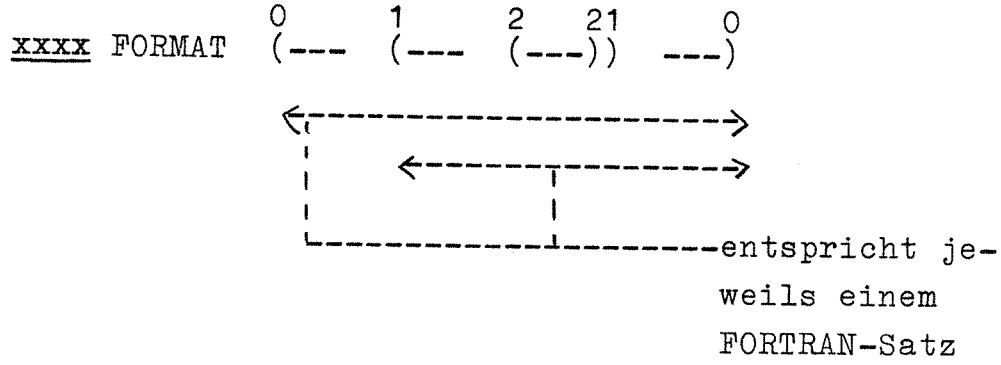


b) Treten Schrägstriche innerhalb einer Formatanweisung auf, so sind FORTRAN-Sätze durch Beginn der Formatanweisung bis zum ersten Schrägstrich, durch einen Schrägstrich zum nächsten Schrägstrich oder vom letzten Schrägstrich bis zum Ende der Formatanweisung definiert. Bei Beginn der Formatsteuerung wird also ein neuer Satz gelesen. Danach wird ein Satz jeweils beim Antreffen eines Schrägstrichs gelesen. Ein neuer Satz ist beim Antreffen eines Schrägstrichs oder sobald die Formatsteuerung beendet ist, geschrieben.

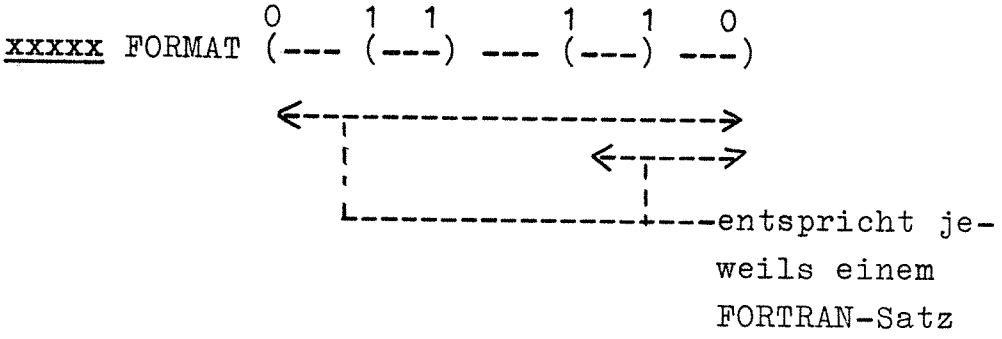


c) Tritt innerhalb einer Formatanweisung mehr als eine Klammerstufe auf, so ist ein FORTRAN-Satz vom Beginn bis zum Ende der Formatanweisung definiert; im weiteren von der auf der ersten Klammerstufe am weitesten rechts in der Formatanweisung stehenden Klammer auf bis zum Ende der Formatanweisung.

Beispiel 1



Beispiel 2



Bei der Definition eines FORTRAN-Satzes durch eine Formatanweisung ist es wichtig, die Quelle (Eingabe) oder das Ziel (Ausgabe) des Satzes zu beachten. Wenn z.B. ein FORTRAN-Satz auf Lochkarten ausgestanzt werden soll, sollte der Satz nicht länger als 80 Zeichen sein. Bei der Eingabe sollte eine Formatanweisung einen FORTRAN-Satz nicht länger definieren als der in der angesprochenen Datei enthaltene Satz ist.

3. Leere Ausgabesätze können eingeschoben oder Eingabesätze überlesen werden, indem in einer Formatanweisung aufeinanderfolgende Schrägstriche (/) geschrieben werden. Stehen am Anfang oder Ende einer Formatanweisung n aufeinanderfolgende Schrägstriche, so werden n Eingabesätze überlesen bzw. n leere Sätze in die Ausgabesätze eingeschoben. Treten n

aufeinanderfolgende Schrägstriche irgendwo sonst in einer Formatanweisung auf, so ist die Anzahl der überlesenen oder eingefügten leeren Sätze gleich n-1.

4. Ein Formatschlüssel kann beliebig oft wiederholt werden, wenn dem Formatschlüssel eine vorzeichenlose, ganzzahlige Konstante vorangesetzt wird.

5. Es ist ein begrenzter Klammerausdruck zugelassen, um die Wiederholung von Datenfeldern entsprechend bestimmten Formatschlüsseln innerhalb einer längeren Formatanweisung zu ermöglichen. Hierzu sind zwei Stufen von Klammern zusätzlich zu den Klammern, die für die Formatanweisung notwendig sind, zugelassen. Die Klammern können jede Zahl von einzelnen Formatschlüsseln oder wiederholten Formatschlüsseln mit der zugehörigen Interpunktion (Kommas oder Schrägstriche) zu einer Gruppe zusammenfassen. Die Gruppe kann beliebig oft wiederholt werden, indem der Klammer auf der Gruppe eine vorzeichenlose, ganzzahlige Konstante vorangesetzt wird.

6. Bei der Eingabe oder Ausgabe von Daten sollten sich im allgemeinen der Typ des verwendeten Formatschlüssels (A,G,I,F,E,D und L), der Typ der Variablen in der Ein-/Ausgabe-Liste und der Typ der Daten (nur bei Eingabe) entsprechen. Die Schlüssel G und A können für jede Art von Übertragung verwendet werden, wie noch gezeigt werden wird. Für Variablen vom Typ "reell" und "komplex" können auch die Schlüssel F, E und D verwendet werden. Für Variablen vom Typ "ganzzahlig" kann auch der Schlüssel I verwendet werden. Für Variablen vom Typ "Boolesch" kann auch der Schlüssel L verwendet werden. Entsprechung von Daten auf externen Sätzen ist nur bei Eingabe von Belang. Daten können sich aus beliebigen Zeichen ein-

schließlich Zwischenräumen zusammensetzen, und zwar für jeden Variablentyp, sofern der Formatschlüssel A vorliegt. Sonst gilt folgendes:

Für Variablen vom Typ "ganzzahlig" müssen die Daten ganzzahlige Konstanten sein. Für Variablen vom Typ "Boolesch" müssen die Daten T oder .TRUE. bzw. F oder .FALSE. sein. Für Variablen vom Typ "reell" müssen die Daten eine beliebige ganze oder reelle Konstante sein. Für Variablen vom Typ "komplex" müssen die Daten aus zwei getrennten Konstanten bestehen, von denen jede eine ganzzahlige oder reelle Konstante ist, und die durch zwei beliebige der Formatschlüssel G,F,E oder D übertragen werden.

7. In den folgenden Beispielen wird die Ausgabe als eine Druckzeile gezeigt. Ein Zeichen für die Wagensteuerung 'x' (s. Abschnitt 8.9. "Wagensteuerung") ist in der Formatanweisung enthalten, tritt aber nicht in der ersten Druckposition der Druckzeile auf. Dieses Steuerzeichen ist in jedem Ausgabesatz als erstes Zeichen enthalten, mit Ausnahme von Druckzeilen.

8. Die für die Trennung von Formatschlüsseln benötigte Interpunktion ist sehr flexibel. Ein oder mehrere Schrägstriche, die als Satzgrenzen eingefügt werden, wie unter 3 oben beschrieben, reichen aus, um zwei Formatschlüssel voneinander zu trennen. Werden keine Schrägstriche gewünscht, können auch Kommas eingesetzt werden. Verwendung von sowohl einem Komma als auch einem oder mehreren Schrägstrichen ist überflüssig, aber zulässig. In den Fällen, in denen das Ende des Formatschlüssels ohne Interpunktion erkannt werden kann (Ende eines Literals, das mit wH begonnen wurde, Ende des Ausdrucks wX nach einer inneren Klammer zu), ist die

Verwendung von Interpunktionszeichen freigestellt.

9. Der Gesamtablauf während einer formatgesteuerten Ein-/Ausgabe kann am besten in folgender Weise zusammengefaßt werden:

   Beim Lesen wird zunächst ein Satz gelesen, sodann beginnt die Formatsteuerung. Beim Schreiben wird ein Zeiger auf den Anfang des ersten zu schreibenden Satzes gesetzt, sodann beginnt die Formatsteuerung. Während der Formatsteuerung wird die Formatanweisung von links nach rechts abgesucht und faktisch "ausgeführt", wobei gleichzeitig alle Einzel- und Gruppenwiederholungen beachtet werden. Beim Antreffen eines jeden Schrägstrichs wird ein neuer Satz übertragen. Wird beim Absuchen ein Literal in Anführungsstrichen oder ein Literal, das mit wH beginnt, erreicht, so werden Daten zwischen dem Satz und der Formatanweisung selbst übertragen. Wenn beim Absuchen einer der Schlüssel A,G,I, F,E,D oder L erreicht wird, wird die Liste der Ein-/Ausgabe-Variablen geprüft. Ist ihr Ende erreicht, endet die Formatsteuerung. Andernfalls wird ein Posten zwischen Satz und Variablenliste unter gleichzeitiger Umwandlung und Positionierung entsprechend dem Formatschlüssel übertragen. Eine komplexe Variable erfordert die Interpretation von zwei aufeianderfolgenden Formatschlüsseln. Nach dieser Übertragung schreitet das Absuchen der Formatanweisung wie oben fort. Wenn das Absuchen die am weitesten rechts stehende Klammer der Formatanweisung (Stufe 0) erreicht hat, wird die Ein-/Ausgabe-Liste erneut geprüft. Ist ihr Ende jetzt erreicht, endet die Formatsteuerung. Ist das nicht der Fall, wird ein weiterer Satz übertragen und das Absuchen wird, gesteuert durch die Gruppenwiederholung, wieder aufgenommen, und zwar beginnend mit der am weitesten rechts stehenden Klammer der Stufe 1 in der

Formatanweisung (siehe Punkt 2,c oben). Wenn die Formatsteuerung beim Lesen beendet ist, dann werden evtl. verbleibende Daten des Satzes überlesen. Wird die Formatsteuerung beim Schreiben beendet, wird der letzte Satz geschrieben.

8.9.1. Formatschlüssel G

Form	Elemente
aGw.s	a kann wahlweise angegeben werden und stellt eine vorzeichenlose, ganzzahlige Konstante dar. Diese gibt an, wie oft der betreffende Formatschlüssel wiederholt angesprochen wird.
	w ist eine vorzeichenlose, ganzzahlige Konstante. Sie gibt die gesamte Feldlänge an.
	s ist eine vorzeichenlose, ganzzahlige Konstante. Sie gibt die Anzahl gültiger Stellen bei Ausgabe an und bei Eingabe die Anzahl der Dezimalstellen nach dem Dezimalpunkt.

Der Formatschlüssel G ist ein verallgemeinerter Schlüssel. Er kann verwendet werden, um die gewünschte Form der Daten festzulegen, sei ihr Typ ganzzahlig, reell, komplex oder Boolesch. Bei der Übertragung von ganzzahligen oder Booleschen Daten kann der Anteil .s weggelassen werden. Ist er dennoch vorhanden, so wird er ignoriert.

Die folgenden Regeln beziehen sich auf die Ausgabe. Werden reelle Daten (oder einer der beiden Teile einer komplexen Größe) übertragen, so enthält der Anteil w

des Formatschlüssels G vier Stellen für ein dezimales Exponentenfeld. Wenn eine reelle Größe n (oder einer der beiden Teile einer komplexen Größe) in dem Bereich $0.1 \leq n < 10^{**}s$ liegt, wo s der .s-Anteil des Formatschlüssels Gw.s ist, ist dieses Exponentenfeld leer. Andernfalls wird die reelle Größe mit einem dezimalen Exponenten E oder D übertragen, je nach Länge - entweder 4 oder 8 Speicherplätze - der reellen Größe.

Bei Eingabe einer reellen Größe (oder einem der beiden Teile einer komplexen Größe) gibt der Anteil .s die Anzahl von Dezimalstellen nach dem Dezimalpunkt der ganzen oder reellen Konstante auf dem externen Satz an. Jedoch setzt ein in die Eingabedaten gelochter Dezimalpunkt die Angabe .s, außer Kraft. Zur Vereinfachung behandeln die folgenden Beispiele die Druckzeile. Das dabei entwickelte Konzept ist jedoch auf alle Ein- und Ausgabemedien anwendbar.

Beispiel 1

Es sei angenommen, daß die Variablen A,B,C,D reell sind und die Werte 292.7041,82.43441,136.7632 und .8081945 haben:

```
1. FORMAT ('x',G12.4,G12.5,G12.4,G12.7)
2. FORMAT ('x',G13.4,G13.5,G13.4)
3. FORMAT ('x',G13.4)
       :
   WRITE (5,c)A,B,C,D
       :
```

Erklärung

a) Hat c den Wert 1, so hat die gedruckte Ausgabe folgendes Aussehen (␣ bedeutet Zwischenraum):

```
Druckposition 1                                   Druckposition 48
↓                                                                 ↓
___292.7_____82.434_______136.8____.8081945____
```

b) Hat c den Wert 2, hat die gedruckte Ausgabe folgendes Aussehen:

```
Druckposition 1          Druckposition 39
↓                                       ↓
____292.7_______82.434________136.8____      Zeile 1
___0.8082____                                Zeile 2
```

Aus Beispiel 1 ist zu ersehen, daß durch Ausdehnung der reservierten Feldlänge (w) Zwischenräume eingeschoben werden.

c) Hat c den Wert 3, hat die gedruckte Ausgabe folgendes Aussehen:

```
Druckposition 1
↓
____292.7____        Zeile 1
____82.43____        Zeile 2
____136.8____        Zeile 3
___0.8082____        Zeile 4
```

Dem Beispiel läßt sich entnehmen, daß derselbe Formatschlüssel für jede Variable in der Liste verwendet wurde. Jede Wiederholung desselben Formatschlüssels bewirkt den Druck einer neuen Zeile.

Beispiel 2

Es sei angenommen, daß die Variablen I,J,K und L ganzzahlig sind und die Werte 292, 443428, 4908081 und 40018 haben.

```
1 FORMAT ('x',G10,2G7,G5)
2 FORMAT ('x',G6)
3 FORMAT ('x',4G10)
     :
  WRITE (5,c)I,J,K,L
     :
```

Erklärung

a) Hat c den Wert 1, hat die gedruckte Ausgabe folgendes Aussehen:

```
Druckposition 1 Druckposition 29
↓              ↓
_______292_443428490808140018          Zeile 1
```

Dieselben Resultate wären erzielt worden, wenn die Formatanweisung 1 wie folgt geschrieben worden wäre:

```
FORMAT('x',G10,G7,G7,G5)
```

Es ist zu beachten, daß der Anteil .s des Formatschlüssels G weggelassen werden kann, wenn ganzzahlige Daten übertragen werden.

b) Hat c den Wert 2, hat die gedruckte Ausgabe folgendes Aussehen:

```
Druckposition 1
↓
___292       Zeile 1
443428       Zeile 2
908081       Zeile 3
_40018       Zeile 4
```

Es ist zu beachten, daß der zweite Formatschlüssel G6 hinsichtlich der dritten Variablen K, d.h. 4908081, falsch ist. Infolgedessen geht die am weitesten links stehende Stelle verloren. Ist allgemein die Längenspezifikation w nicht ausreichen, werden die am weitesten links stehenden Zeichen nicht gedruckt.

c) Wenn c den Wert 3 hat, hat die gedruckte Ausgabe folgendes Aussehen:

```
Druckposition 1          Druckposition 40
↓                                    ↓
_______292____443428___4908081_____40018    Zeile 1
```

Aus diesem Beispiel geht hervor, daß Vergrößerung der Feldlänge w die Lesbarkeit verbessert.

Beispiel 3

Es werde folgendes angenommen: Die Variable I sei ganzzahlig mit der Länge 2, die Variablen A und B seien reell je mit der Länge 4, D sei reell mit der Länge 8, C sei komplex mit der Länge 8 und L sei Boolesch mit der Länge 1. Diese Variablen hätten die Werte 292, 471.93, 81.91, 6.9310072, (2.1,3.7) bzw. .TRUE. .

```
1 FORMAT('x',G3,2G9.2,G13.7,2G8.2,G3)
2 FORMAT('x'G3/'x',2G10.2/'x',G9.1/'x',2G8.2,G3)
3 FORMAT(//'x',G3,2G9.2//'x',G13.7,2G8.2,G3///)
     :
  WRITE(5,c)I,A,B,D,C,L
     :
```

Erklärung

a) Hat c den Wert 1, hat die gedruckte Ausgabe folgendes Aussehen:

```
Druckposition 1                                    Druckposition 53
↓                                                                 ↓
292_0.47E_03__82._____6.931007_____2.1_____3.7______T
```

Werden komplexe Daten übertragen, sind zwei Formatschlüssel erforderlich. Der reelle und der imaginäre Teil wird je als eine eigene reelle Zahl behandelt, die Klammern und das Komma werden nicht mitgedruckt. Das gleiche gilt für die Eingabe. Es werden zwei Formatschlüssel benutzt, und zwei reelle oder ganzzahlige Konstanten des externen Satzes werden gelesen. Diese beiden Konstanten dürfen nicht durch ein Komma voneinander getrennt und nicht in Klammern eingeschlossen sein, wie es der Fall wäre, wenn sie als komplexe Konstanten in einer FORTRAN-Anweisung geschrieben worden wären, es sei denn, die Formatanweisung schließt diese Interpunktionszeichen von den numerischen Feldern, die gelesen und umgewandelt werden sollen, aus.

b) Hat c den Wert 2, hat die gedruckte Ausgabe folgendes Aussehen:

```
Druckposition 1
↑
292                                  Zeile 1
__0.47E_03___82.____                 Zeile 2
__0.7D_01                            Zeile 3
_2.1_____3.7______T                  Zeile 4
```

Aus obigem Beispiel ist zu entnehmen, daß der Gebrauch des Schrägstrichs (/) zur Trennung von zwei Formatschlüsseln bewirkt, daß die noch nicht gedruckten Daten auf eine neue Zeile gedruckt werden. Sollen die Ausgabedaten in Karten gestanzt werden, so gibt der Schrägstrich an, daß das Stanzen der verbleibenden Daten auf der nächsten Karte fortzusetzen ist.

c) Hat c den Wert 3, hat die gedruckte Ausgabe folgendes Aussehen:

```
Druckposition 1
↑
(Leerzeile)                              Zeile 1
(Leerzeile)                              Zeile 2
 292_0.47E_03__82.____                   Zeile 3
(Leerzeile)                              Zeile 4
 _6.931007_____2.1_____3.7______T        Zeile 5
(Leerzeile)                              Zeile 6
(Leerzeile)                              Zeile 7
(Leerzeile)                              Zeile 8
```

An obigem Beispiel ist zu beachten, daß zwei aufeinanderfolgende Schrägstriche am Anfang und drei Schrägstriche am Ende der Reihe von Formatschlüsseln bewirken, daß Leerzeilen wie gezeigt eingefügt werden. Zwei aufeinanderfolgende Schrägstriche an anderer Stelle in der Formatanweisung bewirken, daß nur eine Leerzeile eingeschoben wird, wie an Zeile 4 zu sehen ist.

Die an den vorhergehenden Ausgabebeispielen gezeigten Grundsätze gelten in gleicher Weise für den Gebrauch der Leseanweisung bei der Eingabe. Außerdem sind folgende weiteren Punkte beim Gebrauch der Formatanweisung für Ein- oder Ausgabe zu beachten:

1. Der Gebrauch von zusätzlichen Klammern (bis zu 2 Stufen) innerhalb einer Formatanweisung ist zulässig, um dem Benutzer die Wiederholung desselben Formatschlüssels für die Übertragung von Daten zu ermöglichen. Beispielsweise ist die Anweisung

```
10 FORMAT(2(G10.6,G7.1),G4)
```

gleichwertig der Anweisung

```
10 FORMAT(G10.6,G7.1,G10.6,G7.1,G4)
```

2. Wird eine Mehrzeilenliste derart gewünscht, daß die ersten beiden Zeilen nach einem speziellen Format zu drucken sind und alle übrigen Zeilen nach einem anderen Format, dann sollte der letzte Formatschlüssel in der Anweisung in einem zweiten Paar von Klammern eingeschlossen werden. Ein Beispiel ist die folgende Anweisung:

```
FORMAT('x',G2,2G3.1/'x',G10.8/('x',3G5.1))
```

Sind nach Verwendung aller Formatschlüssel weitere Datenposten zu übertragen, so wiederholt sich das Format ab der letzten Klammer auf. Die gedruckte Ausgabe würde so folgende Form erhalten:

```
G2,G3.1,G3.1
G10.8
G5.1,G5.1,G5.1
G5.1,G5.1,G5.1
   ⋮    ⋮  ⋮
```

Als weiteres Beispiel sei folgende Anweisung betrachtet:

```
FORMAT ('x',G2/2('x',G3,G6.1),G9.7)
```

Wären 13 Datenposten zu übertragen, so würde die gedruckte Ausgabe folgende Form haben:

```
G2
G3,G6.1,'x',G3,G6.1,G9.7
G3,G6.1,'x',G3,G6.1,G9.7
G3,G6.1
```

8.9.2. Numerische Formatschlüssel I,F,E,D

Es gibt vier Typen von Formatschlüsseln für die Übertragung von numerischen Daten. Diese sind in der folgenden Form festgelegt:

Form	Elemente
aIw aFw.d aEw.d aDw.d	a kann wahlweise geschrieben werden und ist eine vorzeichenlose, ganzzahlige Konstante, durch die angegeben wird, wie oft derselbe Formatschlüssel wiederholt angesprochen wird. I,F,E und D sind Formatschlüssel. w ist eine vorzeichenlose, ganzzahlige Konstante, die die gesamte Feldlänge der Daten angibt. d ist eine vorzeichenlose, ganzzahlige Konstante, die die Anzahl der Dezimalen angibt, die rechts auf den Dezimalpunkt folgen (d.h. des gebrochenen Anteils).

Zur Vereinfachung behandelt die folgende Besprechung der Formatschlüssel die Druckzeile. Das dabei entwickelte Konzept ist jedoch auf alle Ein-/Ausgabe-Medien anwendbar.

8.9.2.1. Formatschlüssel I

Der Formatschlüssel I wird zur Ausgabe von ganzzahligen Daten benutzt. Der Schlüssel I8 kann zum Drucken von ganzzahligen Daten verwendet werden. Acht Druckpositionen werden dann für die Zahl reserviert. Sie wird in diesem 8stelligen Feld rechtsbündig gedruckt (d.h. die Einerstelle erscheint am rechten Ende des Feldes).

Wird die zu übertragende Zahl in einen Dezimalwert umgewandelt, der größer als acht Stellen ist, so gehen die führenden Stellen verloren. Hat die Zahl weniger als acht Stellen, werden die linken Druckpositionen mit Zwischenräumen gefüllt. Ist die Größe negativ, enthält die Druckposition, die der am weitesten links stehenden Stelle vorausgeht, ein Minuszeichen. In diesem Fall sollte für das Minuszeichen eine zusätzliche Druckposition in der Feldlänge vorgesehen werden.

Die folgenden Beispiele zeigen, wie jede der Größen in der linken Spalte gemäß dem Formatschlüssel I3 gedruckt wird.

Interner Wert	Gedruckter Wert
721	721
-721	721 (falsch, da Länge des Druckfeldes nicht ausreicht)
-12	-12
568114	114 (falsch, da Länge des Druckfeldes nicht ausreicht)
0	--0
-5	--5
9	__9

8.9.2.2. Formatschlüssel F

Der Formatschlüssel F wird verwendet, um reelle Daten ohne Dezimalexponenten auszugeben. Beim Formatschlüssel F ist w die gesamte reservierte Feldlänge und d ist die Anzahl der Stellen rechts vom Dezimalpunkt, d.h. die Stellenzahl für den gebrochenen Anteil. Das ist anders als bei dem Formatschlüssel G, wo für Ausgabe die Anzahl von gültigen Stellen angegeben wird. Die insgesamt reservierte Feldlänge

muß eine ausreichende Anzahl von Positionen für ein möglicherweise auftretendes Vorzeichen und einen Dezimalpunkt vorsehen. Das Vorzeichen, wenn negativ, wird gedruckt.

Reicht die Anzahl der durch d reservierten Positionen nicht aus, werden die letzten Stellen des Bruchteiles abgeschnitten. Wurden durch d überschüssige Positionen reserviert, werden am rechten Ende Nullen aufgefüllt. Der ganzzahlige Anteil der Zahl wird in derselben Weise behandelt wie Zahlen, die durch den Formatschlüssel I übertragen werden.

Die folgenden Beispiele zeigen, wie jede der Größen in der linken Spalte gemäß dem Formatschlüssel F5.2 gedruckt wird:

Interner Wert	Gedruckter Wert	
12.17	12.17	
-41.16	41.16	(falsch, da Länge des Druckfeldes nicht ausreicht)
-.2	-0.20	
7.3542	_7.35	(Die letzten beiden Stellen gehen wegen nicht ausreichender Feldlänge verloren)
-1.	-1.00	
9.03	_9.03	
187.64	87.64	(falsch, da Länge des Druckfeldes nicht ausreicht)

8.9.2.3. Formatschlüssel D und E

Die Formatschlüssel D und E werden benutzt, um reelle Daten mit Dezimalexponent D oder E, d.h. Zahlen vom Typ REAL*4 oder REAL*8 auszugeben. Bei den Formatschlüsseln D und E wird die Stellenzahl für den ge-

brochenen Anteil ebenfalls wieder durch d angegeben. w schließt das Feld d ein, sowie Speicherpositionen für das Vorzeichen und den Dezimalpunkt und schließlich vier Speicherpositionen für den Exponenten. Es sollte mindestens eine Position vor dem Dezimalpunkt reserviert werden.

Der Exponent bestimmt die Potenz von 10, mit der die Zahl multipliziert werden muß, um ihren wahren Wert zu erhalten. Der Exponent wird in der Form D oder E gefolgt von einer Position für das Vorzeichen und zwei Positionen für den Exponenten (Maximalwert ist 75) ausgegeben.

Die folgenden Beispiele zeigen, wie jede der Größen auf der linken Seite entsprechend den Formatschlüsseln (D10.3/E10.3) gedruckt wird:

Interner Wert	Gedruckter Wert	
238.	_0.238D_03	
-.002	-0.200E-02	
.00000000004	_0.400D-10	
-21.0057	-0.210E_02	(die letzten drei Stellen gehen wegen nicht ausreichender Feldlänge verloren)

8.9.2.4. Lesen numerischer Eingabedaten

Es wurden bereits weiter oben einige Bemerkungen hinsichtlich der notwendigen Übereinstimmung des Variablentyps, des Formatschlüssels und der externen Daten bei Eingabeoperationen gemacht. Die folgenden Hinweise beziehen sich auf die Eingabe von Variablen vom Typ "ganzzahlig", "reell" oder "komplex" bei Verwendung von Formatschlüsseln, die von A verschieden sind:

1. Eingabeoperationen für ganzzahlige Variablen können den Formatschlüssel G oder I verwenden, wie es in den vorausgehenden Beispielen für die Ausgabe gezeigt wurde. Die Form der zu lesenden Daten ist die von gegebenenfalls mit Vorzeichen versehenen ganzzahligen Konstanten in einem Feld der Länge w, entsprechend der Angabe im Formatschlüssel, wobei führende Zwischenräume ignoriert und andere Zwischenräume als Nullen interpretiert werden.

2. Eingabeoperationen für reelle Variablen oder für beide Teile von komplexen Variablen können einen der Formatschlüssel G,F,E oder D in ähnlicher Weise verwenden, wie sie weiter oben für die Ausgabe gezeigt wurde. Die Form der zu lesenden Daten ist die einer beliebigen, gegebenenfalls mit Vorzeichen versehenen ganzen oder reellen Konstanten in einem Feld der Länge w, entsprechend der Angabe im Formatschlüssel. Führende Zwischenräume in der ganzen Zahl, der reellen Basis oder dem Exponententeil werden ignoriert. Andere, d.h. in dem numerischen Anteil von Konstanten oder Exponenten enthaltene Zwischenräume sowie Zwischenräume, die dem Anfang des Exponenten oder dem Ende des Feldes vorausgehen, werden als Nullen interpretiert. Der Anfang des möglicherweise auftretenden Exponentenfeldes ist durch eines der folgenden Kennzeichen, die alle gleichwertig sind, markiert: +,-,D oder E. Die Genauigkeit der Umwandlung hängt von der Länge des Speicherbereichs der Variablen in der Ein-Ausgabe-Liste ab. Ist ein Dezimalpunkt vorhanden, wird er wie angegeben verwendet. Liegt kein Dezimalpunkt vor, bestimmt bei dem Formatschlüssel F,E oder D der d-Teil, bei dem Formatschlüssel G der s-Teil die Anzahl von Dezimalen, die dem rechten Ende des Feldes oder dem Anfang des Exponenten vorausgehen.

8.9.3. Formatschlüssel L

Form	Elemente
aLw	a kann wahlweise angegeben werden und stellt eine vorzeichenlose, ganzzahlige Konstante dar. Diese gibt an, wie oft der betreffende Formatschlüssel wiederholt angesprochen wird.
	w ist eine vorzeichenlose, ganzzahlige Konstante und gibt die Zeichenzahl der Daten an.

Mit Hilfe des Formatschlüssels Lw können Boolesche Variablen geschrieben oder gelesen werden.

Bei der Eingabe bewirkt das erste T oder F, das in den nächsten w Zeichen des Eingabesatzes erkannt wird, daß der entsprechenden Booleschen Variablen der Wert .TRUE. oder .FALSE. zugewiesen wird. Wird kein T oder F angetroffen, wird .FALSE. zugewiesen.

Bei der Ausgabe wird in den Ausgabesatz ein T oder F entsprechend dem Wert der Booleschen Variablen in der Ein-Ausgabe-Liste eingesetzt. Diesem einzelnen Zeichen gehen w-1 Zwischenräume voraus.

8.9.4. Formatschlüssel A

Form	Elemente
aAw	a kann wahlweise angegeben werden und stellt eine vorzeichenlose, ganzzahlige Konstante dar. Diese gibt an, wie oft der betreffende Formatschlüssel wiederholt angesprochen wird.

Form	Elemente
	w ist eine vorzeichenlose, ganzzahlige Konstante, die die Zeichenzahl der Daten angibt.

Der Formatschlüssel Aw wird zum Lesen oder Schreiben von Daten verwendet. Wenn w gleich der Zeichenzahl entsprechend der Längenvereinbarung jedes Postens in der Ein-Ausgabe-Liste ist, werden w Zeichen gelesen oder geschrieben.

Ist bei einer Eingabe w kleiner als die Längenangabe des Postens in der Ein-Ausgabe-Liste, so werden w Zeichen gelesen, und die verbleibenden Zeichen am rechten Ende des Postens werden durch Zwischenräume besetzt. Ist w größer als die Längenangabe, so wird die Anzahl von Zeichen, die sich als Differenz zwischen w und der Längenangabe ergibt, überlesen und die restlichen Zeichen gelesen.

Ist bei einer Ausgabe w kleiner als die Längenangabe des Postens in der Ein-Ausgabe-Liste, besteht die Druckzeile aus den linken w Zeichen des Postens. Ist w größer als die Längenangabe, besteht die Druckzeile aus den Zeichen des Postens, die in dem Feld rechtsbündig angeordnet sind. Diesen gehen in der Druckzeile Zwischenräume voraus.

Beispiel 1

Es sei angenommen, daß das Feld ALPHA eindimensional ist. Der Index laufe von 1 bis 20. Folgende Anweisungen könnten geschrieben werden, um einen Satz von einer Dateiengruppe in eine andere zu "kopieren", die auf einem Lochkartenstanzer ausgegeben wird.

```
    :
10 FORMAT(20A4)
    :
   READ(5,10)(ALPHA(I),I=1,20)
    :
   WRITE(6,10)(ALPHA(I),I=1,20)
    :
```

Erklärung

In diesem Beispiel bewirkt die Leseanweisung, daß 20 Posten von Zeichen von der Dateiengruppe 5 gelesen werden. Jeder Posten besteht aus vier Zeichen und wird in die 20 Speicherbereiche ALPHA(1) bis ALPHA(20) übertragen. Die Schreibanweisung bewirkt, daß die 20 Posten zu je 4 Zeichen auf die Dateiengruppe 6 geschrieben werden.

Beispiel 2

Als weiteres Beispiel sei angenommen, daß alle Variablennamen in der Liste der folgenden Leseanweisung explicit als reell definiert wurden. Das Feld mit dem Namen CONST sei eindimensional, der Index laufe von 1 bis 10. Wird weiter angenommen, daß die Eingabedaten

ABCDE...XYZ$1234567890_

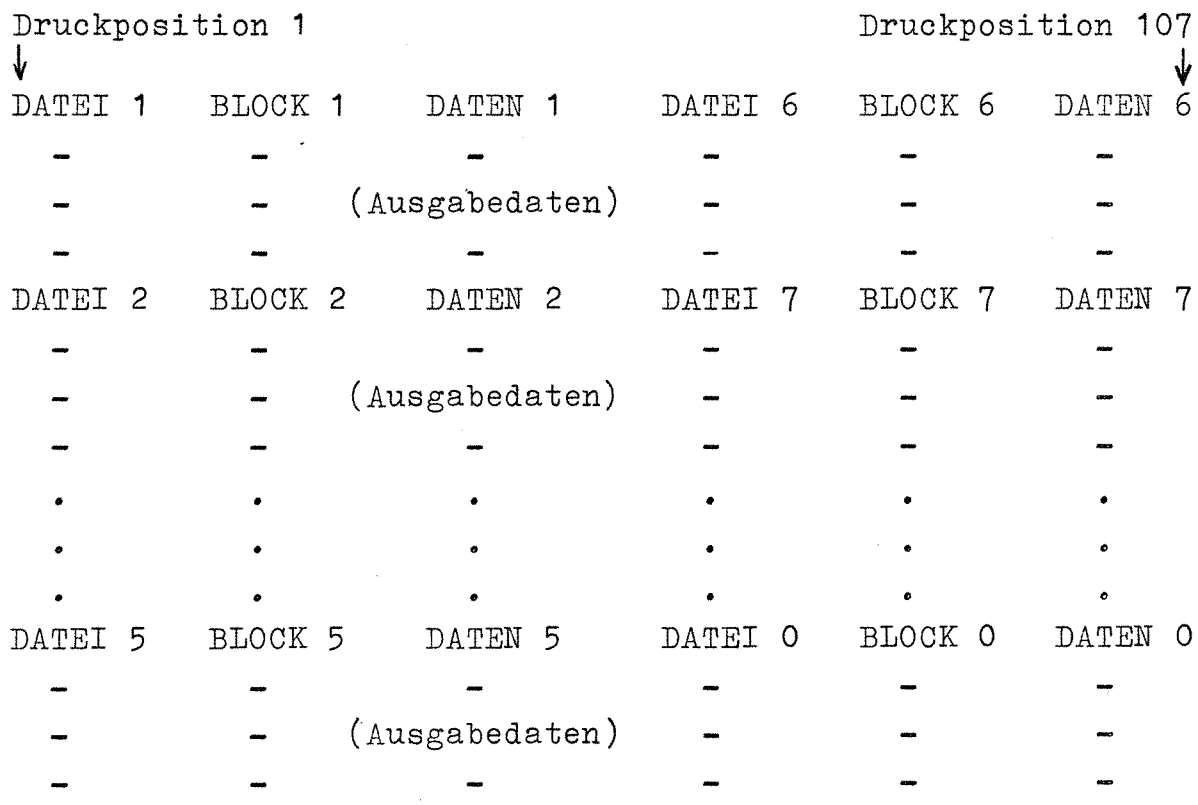
der Dateiengruppe 5 zugeordnet sind, wo ... die Buchstaben F bis W und _ einen Zwischenraum bedeutet, dann können die folgenden Anweisungen geschrieben werden:

```
    :
10 FORMAT(27A1,10A1,A1)
20 FORMAT('x',6(7A1,5X))
    :
```

```
   READ(5,10)A,B,C,D,E,F,G,H,I,
1            J,K,L,M,N,O,P,Q,R,
2            S,T,U,V,W,X,Y,Z,$,
3            (CONST(IND),IND=1,10),ZWRAUM
     :
   DO 50 INDEX = 1,5
     :
   WRITE(6,20)D,A,T,E,I,ZWRAUM,CONST(INDEX),
1             B,L,O,C,K,ZWRAUM,CONST(INDEX),
2             D,A,T,E,N,ZWRAUM,CONST(INDEX),
3             D,A,T,E,I,ZWRAUM,CONST(INDEX+5),
4             B,L,O,C,K,ZWRAUM,CONST(INDEX+5),
5             D,A,T,E,N,ZWRAUM,CONST(INDEX+5)
     :
50 CONTINUE
     :
```

Erklärung

Die Leseanweisung bewirkt, daß die 37 alphanumerischen Zeichen und das Zeichen Zwischenraum aus der Dateiengruppe 5 in die Speicherbereiche übertragen werden, die durch die Variablennamen in der Liste der Leseanweisung bezeichnet sind. Die Variablen A bis Z nehmen also als Wert A bis Z an. Die Variable $ erhält als Wert $. Die Zahlen 1 bis 9 und 0 werden in die 10 Speicherfelder, beginnend mit CONST(1) und endend mit CONST(10), übertragen. Der Variablen ZWRAUM wird ein Zwischenraum zugewiesen. Die Schreibanweisung innerhalb des Schleifenbereichs bewirkt, daß die folgende Überschriftszeile gedruckt wird. Innerhalb der Schleife könnte eine weitere Schreibanweisung geschrieben werden, um die entsprechenden Ausgabedaten zu drucken.



8.9.5. Literalangaben in einer Formatanweisung

Eine Literalangabe innerhalb einer Formatanweisung besteht aus einer Folge von alphanumerischen Zeichen und Sonderzeichen, die in Apostrophen eingeschlossen ist. Die Anzahl von Zeichen innerhalb einer solchen Folge muß kleiner als 256 sein.

Beispiel:

```
25  FORMAT('1966_LAGERLISTE')
```

Ein Apostroph selbst wird durch zwei aufeinanderfolgende Apostrophe dargestellt, sie zählen jedoch nur als ein einziges Zeichen. Beispielsweise werden die Zeichen MUELLER'S in folgender Weise dargestellt:

```
MUELLER''S
```

Die Wirkung einer Literalangabe in einer Formatanweisung hängt davon ab, ob das Format für eine Ein- oder eine Ausgabe-Anweisung benutzt wird.

Eingabe

Eine Anzahl von Zeichen, die gleich der Zahl von Zeichen zwischen den Apostrophen ist, wird von der angegebenen Datei gelesen. Diese Zeichen ersetzen im Speicher die Zeichen innerhalb der Apostrophe.

Beispielsweise würden die Anweisungen

```
      :
5 FORMAT(' EINGABEN')
      :
  READ(3,5)
      :
```

bewirken, daß die nächsten neun Zeichen von der Dateiengruppe 3 gelesen werden. Diese Zeichen ersetzen das Zwischenraumzeichen und die acht Zeichen E,I,N,G,A,B,E und N im Speicher.

Ausgabe

Alle Zeichen einschließlich der Zwischenräume innerhalb der Apostrophe sind Bestandteil der Ausgabedaten. Die Anweisungen

```
      :
5 FORMAT('DIES SIND ALPHANUMERISCHE DATEN')
  WRITE(2,5)
      :
```

würden infolgedessen bewirken, daß folgender Satz auf die Dateiengruppe 2 geschrieben wird:

DIES SIND ALPHANUMERISCHE DATEN

8.9.6. Formatschlüssel H

Form	Elemente
wH	w ist eine vorzeichenlose, ganzzahlige Konstante, die die Anzahl der Zeichen angibt, die auf den Buchstaben H folgen.

Der Formatschlüssel H wird als Alternative zu der Schreibweise von Literalangaben, wie sie gerade erläutert wurde, verwendet. Zur Darstellung eines Apostrophs muß hier nur ein Apostroph geschrieben werden.

Dem Formatschlüssel wH folgen in der Formatanweisung w Zeichen. Die Anzahl dieser Zeichen muß kleiner 256 sein.

Beispiel:

```
5 FORMAT(32H_DIES_SIND_ALPHANUMERISCHE_DATEN)
```

Zwischenräume sind gültig und müssen in w mitgezählt werden. Die Auswirkung von wH hängt davon ab, ob es sich um Eingabe oder Ausgabe handelt.

1. Bei Eingabe werden w Zeichen aus dem Eingabesatz entnommen und ersetzen die w Zeichen der Literalangabe in der Formatanweisung.
2. Bei Ausgabe werden die w Zeichen, die dem Formatschlüssel folgen, als Teil des Ausgabesatzes geschrieben.

8.9.7. Formatschlüssel X

Form	Elemente
wX	w ist eine vorzeichenlose, ganzzahlige Konstante, die die Anzahl der bei Ausgabe einzufügenden Zwischenraumzeichen oder der bei Eingabe zu überlesenden Zeichen angibt.

Wenn der Formatschlüssel wX in Verbindung mit einer Leseanweisung (d.h. bei Eingabe) verwendet wird, werden w Zeichen überlesen, bevor Daten übertragen werden. w muß kleiner 256 sein. Wenn z.B. eine Lochkarte aus sechs 10spaltigen Feldern besteht, die ganzzahlige Größen enthalten, und die zweite Größe nicht gelesen werden soll, kann die Anweisung

```
5 FORMAT(I10,10X,4I10)
```

zusammen mit der geeigneten Leseanweisung verwendet werden.

Wird der Formatschlüssel wX mit einer Schreibanweisung (d.h. bei Ausgabe) verwendet, werden w Zeichen leer gelassen. Es besteht so die Möglichkeit, in einer Druckzeile Leerstellen einzusetzen. Z.B. kann die Anweisung

```
10 FORMAT('x',3(F6.2,5X))
```

zusammen mit einer geeigneten Schreibanweisung verwendet werden, um eine Zeile wie folgt zu drucken:

```
123.45_____817.32_____524.67_____
```

8.9.8. Formatschlüssel T

Form	Elemente
Tw	w ist eine vorzeichenlose, ganzzahlige Konstante, die die Stelle eines FORTRAN-Satzes angibt, bei der die Datenübertragung beginnen soll.

Eingabe und Ausgabe können bei einer beliebigen Stelle beginnen, wenn der Formatschlüssel Tw verwendet wird. w muß hier wiederum kleiner als 256 sein. Zu beachten ist, daß im Falle von gedruckter Ausgabe w und die tatsächliche erste Druckposition unterschiedlich sind. Da das Zeichen für Formularvorschub das erste Zeichen des FORTRAN-Satzes ist, entspricht in diesem Fall die erste Druckposition dem Wert w-1, wie auch aus folgendem Beispiel hervorgeht:

```
5 FORMAT(T40,'LAGERLISTE_1966',T80,'DEZEMBER',T2,
        'TEIL_NR.10095')
```

Diese Formatanweisung würde folgende Druckzeilen liefern:

```
Druckposition 1     Druckposition 39     Druckposition 79
↓                   ↓                    ↓
TEIL NR.10095       LAGERLISTE 1966      DEZEMBER
```

Die Anweisungen

```
5 FORMAT(T40,'_EINGABEN')
      ⋮
  READ(3,5)
```

bewirken, daß die ersten 39 Zeichen der Eingabedaten überlesen werden und daß die nächsten neun Zeichen den Zwischenraum und die Zeichen E,I,N,G,A,B,E und N im Speicher ersetzen.

Der Formatschlüssel T kann innerhalb einer Formatanweisung zusammen mit beliebigen anderen Formatschlüsseln auftreten. Beispielsweise ist die folgende Anweisung zulässig:

```
5 FORMAT(T100,F10.3,T50,E9.3,T1,'ANTWORT IST')
```

8.9.9. Skalenfaktor P

Die interne oder externe Darstellung reeller Daten kann durch einen Skalenfaktor der Form nP modifiziert werden, wobei n eine, erforderlichenfalls mit Vorzeichen versehene ganzzahlige Konstante ist. Dieser Skalenfaktor geht einem Formatschlüssel voraus.

Bei Beginn der Formatsteuerung wird angenommen, daß der Skalenfaktor Ø ist. Wird die Formatanweisung "ausgeführt", indem sie in der weiter oben beschriebenen Weise abgesucht wird, kann ein Skalenfaktor angetroffen werden. Sobald ein Skalenfaktor auf diese Weise wirksam geworden ist, betrifft er alle folgenden Formatschlüssel G,F,E und D entsprechend den nachfolgenden Regeln. Wenn eine Formatanweisung selbst von Anfang an wiederholt wird, wird der Skalenfaktor nicht verändert. Er kann nur durch ein weiteres nP geändert und nur durch ØP beseitigt werden.

Der Skalenfaktor wirkt sich bei Verwendung des Formatschlüssels G nur auf reelle und komplexe Variablen aus. Variablen, die mit den Formatschlüsseln F,E und D benutzt werden, sind stets reell oder komplex.

Eingabe

Tritt in den externen Daten ein Exponent auf, so hat der Skalenfaktor keine Wirkung. Sonst gilt:

$$\text{Interne Größe} = \text{externe Größe}/10^{\underline{n}}$$

dabei ist n der Skalenfaktor.

Ausgabe

Liegt der Formatschlüssel E bzw. D vor oder der Formatscchlüssel G in einem Fall, in dem wegen der Größenordnung der Variablen ein Exponent aufgebaut wird (s."Formatschlüssel G"), so verändert der Skalenfaktor den erzeugten Wert nicht; er hat jedoch zur Folge, daß die Zahl mit $10^{\underline{n}}$ multipliziert und der Exponent um n vermindert wird. Ist im Falle des Formatschlüssels G die Größenordnung der Variablen so, daß kein Exponent erzeugt wird, so hat der Skalenfaktor keine Wirkung.
Im Falle des Formatschlüssels F gilt folgendes:

$$\text{Externe Größe} = \text{Interne Größe} * 10^{\underline{n}}$$

Eingabe

Es seien beispielsweise die folgenden Eingabedaten betrachtet:

27___-93.2094__-175.8041____55.3647,

wo _ ein Leerzeichen darstellt.

Die Anweisungen

```
5 FORMAT(I2,3F11.4)
      :
  READ(6,5)K,A,B,C
```

bewirken dann, daß die Variablen in der Liste die folgenden Werte annehmen:

K:27	B:-175.8041
A:-93.2094	C:55.3647

Die Anweisungen

```
5 FORMAT(I2,1P3F11.4)
      :
  READ(6,5)K,A,B,C
```

bewirken, daß die Variablen in der Liste die folgenden Werte annehmen:

K:27	B:-17.5804
A:-9.3209	C:5.5364

Die Anweisungen

```
5 FORMAT(I2,-1P3F11.4)
      :
  READ(6,5)K,A,B,C
```

bewirken, daß die Variablen in der Liste die folgenden Werte annehmen:

K:27	B:-1758.041
A:-932.094	C:553.647

Ausgabe

Es sei angenommen, daß die Variablen K,A,B und C die folgenden Werte haben:

K:27	B:-175.8041
A:-93.2094	C:55.3647

In diesem Fall bewirken die Anweisungen

```
5 FORMAT(I2,1P3F11.4)
      :
  WRITE(4,5)K,A,B,C
```

daß die Variablen in der Liste folgende Werte ausgeben:

```
K:27              B:-1758.0410
A:-932.0940       C:553.6470
```

Die Anweisungen

```
5 FORMAT(I2,-1P3F11.4)
    :
  WRITE(4,5)K,A,B,C
```

bewirken, daß die Variablen in der Liste folgende Werte ausgeben:

```
K:27              B:-17.4804
A:-9.3209         C:5.5364
```

Das folgende Beispiel zeigt einen Fall, in dem der Wert der ausgedruckten Ergebnisse unverändert bleibt, ihre Darstellung jedoch geändert ist. Hätte die Formatanweisung

```
5 FORMAT(1X,I2,3E13.3)
```

mit einer geeigneten Ausgabeanweisung den Druck der Zeile

```
27___-0.932E_02___-0.176E_03____0.554E_02
```

zur Folge, so würde der Gebrauch der Anweisung

```
5 FORMAT(1X,I2,1P3E13.3)
```

ohne Änderung der Variablen und der Ausgabeanweisung den Druck der folgenden Zeile bewirken:

```
27___-9.320E_01___-1.758E_02____5.536E_01
```

8.9.10. Formularvorschub

Falls Sätze, die mit Formatsteuerung geschrieben wurden, zum Ausdruck vorbereitet werden, muß für den Formularvorschub die folgende Vereinbarung beachtet werden:

Erstes Zeichen	Formularvorschub vor Druck
Zwischenraum	1 Zeile
0	2 Zeilen
1	Auf die 1.Zeile der nächsten Seite
+	kein Vorschub

Das erste Zeichen eines zu druckenden Ausgabesatzes steuert den Formularvorschub und wird nicht gedruckt. In allen anderen Speichermedien ist es Bestandteil der Daten.

8.9.11. Formatschlüssel in Feldern

FORTRAN bietet die Möglichkeit, eine Folge von Formatschlüsseln als Argumente an Unterprogramme weiterzugeben und während der Programmausführung zu verändern.

Um eine veränderliche Folge von Formatschlüsseln zu benutzen, hat der Programmierer die Möglichkeit, ein Feld zu beschreiben, dessen gesamte Länge (Produkt seiner Dimensionen und der Länge eines jeden Elements) ausreichend ist, die vollständige Folge der Formatschlüssel, eingeschlossen in Klammern, jedoch ohne das Wort FORMAT aufzunehmen. Er kann einem solchen Feld Anfangswerte in Form von Literalangaben zuweisen, und zwar mit Hilfe einer expliciten Spezifikationsanweisung oder einer Anfangswertanweisung. Er kann den Inhalt des Feldes verändern, indem er unter Verwendung des Formatschlüssels A Daten in das Feld einliest.

Um eine Folge von Formatschlüsseln als Argument für ein Unterprogramm weiterzugeben, kann der Name eines Feldes, das eine veränderliche Folge von Formatschlüsseln enthält, oder die Nummer einer Formatanweisung (geschrieben &n, wo n die Anweisungsnummer ist) als Aktualparameter des Unterprogramms benutzt werden. Innerhalb des Unterprogramms muß der zugehörige Formalparameter, der eine der beiden obigen Informationen erhält, ein Feldname sein, d.h. für ihn muß über eine explicite Spezifikationsanweisung, eine Feldanweisung oder eine Speicherblockanweisung einer Länge ($\leq$ 1) vereinbart sein.

Eine Folge von Formatschlüsseln, eingeschlossen in Klammern, kann als ein Argument an ein Subroutinenunterprogramm dadurch weitergegeben werden, daß ein Literal als Aktualparameter in eine Aufrufanweisung eingesetzt wird.

Beispiel

```
REAL*4FMT*8(9)/'(G4,G6)'/,C(15)
1 FORMAT(9A8)
      :
      :
  READ(5,FMT)M,N
      :
      :
2 READ(5,1)FMT
      :
      :
3 READ(5,FMT)A,B,(C(I),I=1,11,2)
      :
      :
  CALL SUB1(A,B,C,FMT,&1,'(G5)')
      :
      :
  END
```

```
SUBROUTINE SUB1(/X/,/Y/,Z,F1,F2,F3)
REAL*4F1*8(9),F2(1),F3(1),Z(15)
    :
READ(5,F2)F1
    :
READ(5,F1),X,Y,(Z(I),I=1,11,2)
    :
READ(5,F3)D
    :
END
```

Erklärung

In diesem Beispiel sind alle beschriebenen Möglichkeiten enthalten. Das Hauptprogramm weist über eine Typanweisung dem Formatfeld FMT als Anfangswerte die Formatschlüssel G4,G6 zu und verwendet es, um die beiden ganzen Zahlen M und N einzulesen. Anschließend wird ein festes Format mit dem Schlüssel A benutzt, um 72 Zeichen in das Feld FMT zu lesen. Dieses neue Format wird dann verwendet, um die Variablen A und B sowie das Feld C zu lesen. Danach wird eine Subroutine SUB1 aufgerufen, wobei als Argumente das Formatfeld (FMT), das feste Format (&1) und ein Literal (G5) auftreten. Im Unterprogramm werden die entsprechenden drei Formalparameter in einer expliciten Spezifikationsanweisung als Felder vom Typ REAL identifiziert und in Leseanweisungen verwendet. Die ersten beiden Leseanweisungen im Unterprogramm wirken ähnlich den Anweisungen 2 und 3 im Hauptprogramm. Sie verändern A und B sowie die Felder C und FMT des Hauptprogramms. Die dritte Leseanweisung liest die Variable D entsprechend dem Formatschlüssel G5.

Anhang 1

Zeichen für Primärprogramme

Alphabetische Zeichen	Karten-lochung	Numerische Zeichen	Karten-lochung
A	12-1	0	0
B	12-2	1	1
C	12-3	2	2
D	12-4	3	3
E	12-5	4	4
F	12-6	5	5
G	12-7	6	6
H	12-8	7	7
I	12-9	8	8
J	11-1	9	9
K	11-2		
L	11-3		
M	11-4		
N	11-5		
O	11-6		
P	11-7		
Q	11-8		
R	11-9		
S	0-2		
T	0-3		
U	0-4		
V	0-5		
W	0-6		
X	0-7		
Y	0-8		
Z	0-9		
$	11-3-8		

Sonderzeichen	Kartenlochung
+	12-6-8
-	11
/	0-1
=	6-8
.	12-3-8
)	11-5-8
*	11-4-8
,	0-3-8
(	12-5-8
'	5-8
&	12

Diese 48 Zeichen zusammen mit dem Zeichen Zwischenraum sind die sämtlichen in FORTRAN zugelassenen Zeichen (eine Ausnahme für diese Einschränkung bilden Literale). Der Zeichenvorrat unterscheidet sich von demjenigen früherer FORTRAN-Versionen. Um sich dieser Änderung anpassen zu können, muß bei der Monitorinstruktion PARAM lediglich ein bestimmter Parameter angegeben werden. In diesem Falle ist es dann möglich, Programme in dem folgenden Code zu übersetzen:

Zeichen	Alte Lochung mit neuer Interpretation	
	7090 Zusatz	3301 Zusatz
$	Keine Änderung der Lochung; wird jedoch ebenso wie &(12) nicht als alphabetisches Zeichen angesehen	
+	12 (&)	12-2-8 (¢)
(	0-4-8 (%)	5-8 (')
)	12-4-8 (<)	6-8 (=)
'	4-8 (@)	12-6-8 (+)
=	3-8 (≠)	0-6-8 (>)

Funktion	Name	Bedeutung	offen (I) geschlossen (0)	Anzahl der Argumente	Typ u. Länge der Argumente	Typ u. Länge der Funktion
Exponential-funktion	EXP	e^{arg}	0	1	Reell, Länge 4	Reell, Länge 4
	DEXP	e^{arg}	0	1	Reell, Länge 8	Reell, Länge 8
	CEXP	e^{arg}	0	1	Komplex, Länge 8	Komplex, Länge 8
	CDEXP	e^{arg}	0	1	Komplex, Länge 16	Komplex, Länge 16
Natürlicher Logarithmus	ALOG	ln (Arg)	0	1	Reell, Länge 4	Reell, Länge 4
	DLOG	ln (Arg)	0	1	Reell, Länge 8	Reell, Länge 8
	CLOG	ln (Arg)	0	1	Komplex, Länge 8	Komplex, Länge 8
	CDLOG	ln (Arg)	0	1	Komplex, Länge 16	Komplex, Länge 16
Dekadischer Logarithmus	ALOG10	$\log_{10}$ (Arg)	0	1	Reell, Länge 4	Reell, Länge 4
	ALOG10	$\log_{10}$ (Arg)	0	1	Reell, Länge 8	Reell, Länge 8
Arcus-tangens	ATAN	arctan (Arg)	0	1	Reell, Länge 4	Reell, Länge 4
	ATAN2	arctan (Arg_1/Arg_2)	0	2	Reell, Länge 4	Reell, Länge 4
	DATAN	arctan (Arg)	0	1	Reell, Länge 8	Reell, Länge 8
	DATAN2	arctan (Arg_1/Arg_2)	0	2	Reell, Länge 8	Reell, Länge 8
Sinus (arg Bogenmass)	SIN	sin (Arg)	0	1	Reell, Länge 4	Reell, Länge 4
	DSIN	sin (Arg)	0	1	Reell, Länge 8	Reell, Länge 8
	CSIN	sin (Arg)	0	1	Komplex, Länge 8	Komplex, Länge 16
	CDSIN	sin (Arg)	0	1	Komplex, Länge 16	
Cosinus (arg Bogenmass)	COS	cos (Arg)	0	1	Reell, Länge 4	Reell, Länge 4
	DCOS	cos (Arg)	0	1	Reell, Länge 8	Reell, Länge 8
	CCOS	cos (Arg)	0	1	Komplex, Länge 8	Komplex, Länge 8
	CDCOS	cos (Arg)	0	1	Komplex, Länge 16	Komplex, Länge 16
Quadrat-wurzel	SQRT	$(Arg)^{1/2}$	0	1	Reell, Länge 4	Reell, Länge 4
	DSQRT	$(Arg)^{1/2}$	0	1	Reell, Länge 8	Reell, Länge 8
	CSQRT	$(Arg)^{1/2}$	0	1	Komplex, Länge 8	Komplex, Länge 8
	CDSQRT	$(Arg)^{1/2}$	0	1	Komplex, Länge 16	Komplex, Länge 16
Tangens hyperbolicus	TANH	tanh (Arg)	0	1	Reell, Länge 4	Reell, Länge 4
	DTANH	tanh (Arg)	0	1	Reell, Länge 8	Reell, Länge 8

Funktion	Name	Bedeutung	offen (1) geschlossen (0)	Anzahl der Argumente	Typ u. Länge der Argumente	Typ u. Länge der Funktion
Divisions-rest	MOD	Arg_1 (modArg_2)	1	2	Ganzzahlig, Länge 4	Ganzzahlig, Länge 4
	AMOD	$Arg_1 - \left(\frac{Arg_1}{Arg_2}\right) * Arg_2$	1	2	Reell, Länge 4	Reell, Länge 4
	DMOD	(X) ist der ganzzahlige Anteil von X.	1	2	Reell, Länge 8	Reell, Länge 8
Absolutwert	IABS	Arg	1	1	Ganzzahlig, Länge 4	Ganzzahlig, Länge 4
	ABS		1	1	Reell, Länge 4	Reell, Länge 4
	DABS		1	1	Reell, Länge 8	Reell, Länge 8
	CABS	$(a^2+b^2)^{1/2}$ für Arg=a+bi	0	1	Komplex, Länge 8	Reell, Länge 4
	CDABS		0	1	Komplex, Länge 16	Reell, Länge 8
Abbrechen	INT	$[Arg]$=	1	1	Reell, Länge 4	Ganzzahlig, Länge 4
	DINT	eine ganze Zahl, deren	1	1	Reell, Länge 8	Reell, Länge 8
	AINT	Vorzeichen gleich dem des Arg ist u. den	1	1	Reell, Länge 4	Reell, Länge 4
	IDINT	grössten Wert $\leq \lvert Arg \rvert$ hat	1	1	Reell, Länge 8	Reell, Länge 4
Höchstwert	AMAX0	Max (Arg_1,Arg_2,...)	0	≥ 2	Ganzzahlig, Länge 4	Reell, Länge 4
	AMAX1		0	≥ 2	Reell, Länge 4	Reell, Länge 4
	MAX0		0	≥ 2	Ganzzahlig, Länge 4	Ganzzahlig, Länge 4
	MAX1		0	≥ 2	Reell, Länge 4	Ganzzahlig, Länge 4
	DMAX1		0	≥ 2	Reell, Länge 8	Reell, Länge 8
Kleinstwert	AMIN0	Min (Arg_1,Arg_2,...)	0	≥ 2	Ganzzahlig, Länge 4	Reell, Länge 4
	AMIN1		0	≥ 2	Reell, Länge 4	Reell, Länge 4
	MIN0		0	≥ 2	Ganzzahlig, Länge 4	Ganzzahlig, Länge 4
	MIN1		0	≥ 2	Reell, Länge 4	Ganzzahlig, Länge 4
	DMIN1		0	≥ 2	Reell, Länge 8	Reell, Länge 8
Umwandlung ganzzahlig in reell	FLOAT		1	1	Ganzzahlig, Länge 4	Reell, Länge 4
	DFLOAT		1	1	Ganzzahlig, Länge 4	Reell, Länge 8

Funktion	Name	Bedeutung	offen(I) geschlossen(0)	Anzahl der Argumente	Typ u. Länge der Argumente	Typ u. Länge der Funktion
Umwandlung reell in ganzzahlig	IFIX HFIX		I I	1 1	Reell, Länge 4 Reell, Länge 4	Ganzzahlig, Länge 4 Ganzzahlig, Länge 2
Vorzeichenübertragung	SIGN ISIGN DSIGN	Vorzeichen von $Arg_2 \times \lvert Arg_1 \rvert$	I I	2 2	Reell, Länge 4 Ganzzahlig, Länge 4 Reell, Länge 8	Reell, Länge 4 Ganzzahlig, Länge 4 Reell, Länge 8
Positive Differenz	DIM IDIM	Max $(Arg_1-Arg_2, \emptyset)$	I I	2 2	Reell, Länge 4 Ganzzahlig, Länge 4	Reell, Länge 4 Ganzzahlig, Länge 4
Ermittlung der höchsten Stellen eines Arguments vom Typ REAL*8 oder COMPLEX*16	SNGL CSNGL		I I	1 1	Reell, Länge 8 Komplex, Länge 16	Reell, Länge 4 Komplex, Länge 8
Reeller Teil eines komplexen Arguments	REAL DREAL	a für Arg=a+bi	I I	1 1	Komplex, Länge 8 Komplex, Länge 16	Reell, Länge 4 Reell, Länge 8
Imaginärer Teil eines komplexen Arguments	AIMAG DIMAG	b für Arg=a+bi	I I	1 1	Komplex, Länge 8 Komplex, Länge 16	Reell, Länge 4 Reell, Länge 8
Zahl einfacher Genauigkeit als Zahl doppelter Genauigkeit darstellen	DBLE CDBLE		I I	1 1	Reell, Länge 4 Komplex, Länge 8	Reell, Länge 8 Komplex, Länge 16
Zwei reelle Argumente in komplexer Form darstellen	CMPLX DCMPLX	Arg_1+iArg_2	I I	2 2	Reell, Länge 4 Reell, Länge 8	Komplex, Länge 8 Komplex, Länge 16
Ermittlung der Konjugierten einer komplexen Zahl	CONJG DCONJG	$\lvert a-bi \rvert$ für Arg=a+bi	I I	1 1	Komplex, Länge 8 Komplex, Länge 16	Komplex, Länge 8 Komplex, Länge 16

Automatische Festlegung des Funktionstyps

Die Funktionen in Tabelle 5, die wie die ähnlich benannten Funktionen in Tabelle 4 definiert sind, können mit Argumenten verschiedenen Typs aufgerufen werden. Wenn sie mit einem Argument bzw. mit Argumenten, die in der entsprechenden Spalte mit * oder ** markiert sind, aufgerufen werden, wird eine Funktion dieses Typs automatisch geliefert. Werden sie ohne Argumente als ein Argument in einem Unterprogrammaufruf verwendet, so gilt der mit ** markierte Typ.

Tabelle 5 Funktionen, die in FORTRAN vorhanden sind

Funktion	Typ und Länge von Argument und Funktion					
Name	I*2	I*4	R*4	R*8	C*8	C*16
EXP			**	*	*	*
LOG			**	*	*	*
LOG 10			**	*	*	*
ATAN			**	*		
SIN			**	*	*	*
COS			**	*	*	*
SQRT			**	*	*	*
TANH			**	*		
MOD	*	*	*	*		
ABS	*	*	*	*		
CONJG					*	*
MAX	*	*	*	*		
MIN	*	*	*	*		
SIGN	*	*	*	*		
DIM	*	*	*			

Test von symbolischen Anzeigen

In der folgenden Liste von Testunterprogrammen für symbolische Anzeigen ist angenommen, daß i ein ganzzahliger Ausdruck und j eine ganzzahlige Variable ist. Diese Unterprogramme werden durch Aufrufanweisungen angesprochen.

SLITE(i): Wenn i = 0, dann werden alle symbolischen Schalter gelöscht. Ist i = 1,2,3 oder 4, dann wird der entsprechende symbolische Schalter eingeschaltet.

SLITET(i,j): Der symbolische Schalter i (gleich 1,2,3 oder 4) wird getestet und gelöscht. Die Variable j wird gleich 1 gesetzt, wenn i eingeschaltet war bzw. wird gleich 2 gesetzt, wenn i ausgeschaltet war.

Beispiel

Es sei angenommen, daß das Programm fortgesetzt werden soll, wenn der symbolische Schalter i eingeschaltet ist, und daß Ergebnisse ausgegeben werden sollen, wenn i ausgeschaltet ist. Dies kann durch Verwendung der Booleschen Wennanweisung oder einer Anweisung für berechneten Sprung erreicht werden:

```
      :
   CALL SLITET(3,KEN)
   GO TO(6,17)KEN
17 WRITE(3,26)(ANS(K),K=1,10)
 6 CONTINUE
      :
```

Erklärung

Wenn die Anweisung CALL SLITET(3,KEN) ausgeführt wird, wird der Variablen KEN der Wert 1 oder 2 zugewiesen, je nachdem, ob der symbolische Schalter 3 ein- oder ausgeschaltet war, wobei gleichzeitig der Schalter gelöscht wird. Ist KEN gleich 1, wird als nächstes die Anweisung mit der Nummer 6 ausgeführt. Ist KEN gleich 2, wird Anweisung 17 ausgeführt.

OVERFL(j): j wird gleich 1 gesetzt, wenn ein Gleitpunktüberlauf eintritt, d.h., wenn das Ergebnis einer arithmetischen Operation größer als 16^{63} ist. j wird gleich 2 gesetzt, wenn kein Überlauf eingetreten ist. j wird gleich 3 gesetzt, wenn ein Gleitpunktunterlauf auftritt, d.h., wenn das Ergebnis einer arithmetischen Operation kleiner ist als 16^{-64}. Die Maschine wird in den Zustand "Kein Überlauf" versetzt.

DVCHK(j): Wenn die Anzeige für Divisionsprüfung eingeschaltet ist, wird j gleich 1 gesetzt und die Anzeige für Divisionsprüfung gelöscht. Ist die Anzeige für Divisionsprüfung gelöscht, wird j gleich 2 gesetzt.

Die Unterprogramme EXIT, DUMP und PDUMP

Unterprogramm EXIT

Ein Aufruf des Unterprogramms EXIT beendet die Ausführung des Maschinenprogramms.

Unterprogramm DUMP

Ein Aufruf des Unterprogramms DUMP durch die Anweisung

$$\text{CALL DUMP}(A_1,B_1,F_1,\ldots,A_n,B_n,F_n)$$

bewirkt, daß ein Speicherauszug innerhalb der angegebenen

Grenzen angefertigt und die Ausführung des Programms beendet wird.

1. A und B sind Variablennamen, die die Grenzen des Speicherbereiches angeben, von dem der Speicherauszug vorgenommen werden soll. Sowohl A als auch B können die obere oder untere Grenze angeben.

2. Jedes F_i ist eine ganze Zahl, die das Format des gewünschten Speicherauszugs angibt:

   F_i = 0 Hexadezimal
   1 Boolesch*1
   2 Boolesch*4
   3 Ganzzahlig*2
   4 Ganzzahlig*4
   5 Reell*4
   6 Reell*8
   7 Komplex*8
   8 Komplex*16
   9 Literal

3. Wird das letzte Argument, F_n, weggelassen, so wird angenommen, daß es gleich 0 ist.

4. Die Argumente A und B sollten in demselben Programm (Hauptprogramm oder Unterprogramm) oder in demselben gemeinsamen Speicherblock enthalten sein.

Unterprogramm PDUMP

Ein Aufruf des Unterprogramms PDUMP durch die Anweisung

$$\text{CALL PDUMP}(A_1,B_1,F_1,\ldots,A_n,B_n,F_n)$$

bewirkt, daß ein Speicherauszug von dem innerhalb der angegebenen Grenzen liegenden Speicherbereich ausgeführt

und die Ausführung des Programms fortgesetzt wird. Die Argumente für das Unterprogramm PDUMP sind dieselben wie für das Unterprogramm DUMP.

Anhang 2

Anhang 3

Übersetzung und Ausführung von FORTRAN-Programmen

Übersetzung

Der FORTRAN-Compiler arbeitet unter Steuerung des Monitors des Band-Betriebssystems (BBS). Zwei Steuerinstruktionen des Monitors sind von besonderer Wichtigkeit für die Übersetzung, nämlich die Instruktion, die sowohl das Bereitstellen als auch den Ablauf des FORTRAN-Compilers veranlaßt und die Instruktion für die Spezifikation der Parameter, die den FORTRAN-Compiler modifizieren.

Der einmal bereitgestellt Übersetzer ist in der Lage, eine beliebige Anzahl von Primärprogrammeinheiten (Hauptprogramme und/oder Unterprogramme) zu übersetzen. Es ist daher vorteilhaft, diese in Gruppen zusammenzufassen. Die übliche Methode ist es, der Folge von Primärprogrammeinheiten, deren jede ihre eigene abschließende Endanweisung hat, in der Monitoreingabe die Instruktion für den Start des FORTRAN-Compilers vorausgehen zu lassen. Zu beachten ist, daß eine Übersetzung drei Magnetbandeinheiten zusätzlich zu den Eingabe- und Ausgabegeräten des Monitors benötigt. Die normale Vorgehensweise des Compilers ist es, das Primärprogramm aus der Monitoreingabe zu lesen, und zwar im Standardcode (EBCDIC). Das Primärprogramm wird nicht aufgelistet, und es werden keine Karten für die Maschinenprogramme gestanzt. Das Maschinenprogramm wird in Modulformat erzeugt und ist als Eingabe des Binders zu verwenden, bevor das Programm ausgeführt werden kann. Gewöhnlich werden nur allgemeine Speicherübersichten gedruckt. Sind in dem Primärprogramm Fehler enthalten, werden auch geeignete Fehlermeldungen gedruckt.

Der Programmierer hat möglicherweise den Wunsch, das Verhalten des Compilers für seine speziellen Zwecke zu verändern. Dies kann durch den Gebrauch der Parameter-Spezifikationsinstruktion für den Monitor mit geeigneten Eintragungen geschehen. Während der Fehlerbeseitigung kann der Programmierer z.B. von folgenden zusätzlichen Informationen und Möglichkeiten Gebrauch machen:

- eine vollständige Liste seines Primärprogramms auszudrucken
- detaillierte Speicherübersichten, geordnet nach Namen oder Nummern der Anweisungen und nach relativer Speicheradresse auszudrucken.

- die Compilierung von zusätzlichen Anweisungen, durch die das Testhilfe-Unterprogramm in die Lage versetzt wird, beim Auftreten eines den Weiterlauf unmöglich machenden Fehlers die Folgenummern der Karten auszudrucken, die die letzten - evtl. nur teilweise - ausgeführten Anweisungen des Hauptprogramms oder eines nicht durch Ausführung einer Rücksprunganweisung abgeschlossenen Unterprogramms enthalten.

Weitere Zusätze geben dem Programmierer folgende Möglichkeiten:

- Angabe eines von dem für die Monitoreingabe verschiedenen Eingabegerätes für das Primärprogramm im Falle, daß z.B. das Primärprogramm durch ein anderes Programm vorbereitet wurde.

- Ausstanzen seines Maschinenprogramms auf Lochkarten in Modulformat.

- Verhinderung der Speicherung des Maschinenprogramms auf Magnetband im Falle, daß die unmittelbar anschließende Ausführung nicht gewünscht wird.

- Mitteilung an den Compiler, daß ein Primärprogramm umzuwandeln ist, bei dem der frühere Lochkartencode (s. Anhang 1) verwendet wurde.

Die Parameter-Spezifikationsinstruktion muß der Instruktion zum Bereitstellen des Compilers vorausgehen. Die gewählten Angaben betreffen alle folgenden, für die Compilation in Gruppen zusammengefaßte Programmeinheiten.

Ausführung

Ein Objektprogramm kann im Band-Betriebssystem entweder vom Monitor gesteuert werden oder unter direkter Steuerung des Ablaufteils des Organisationsprogramms arbeiten. Die Methoden hierfür sind für alle Moduls gültig und werden hier nicht beschrieben. Der Programmierer sollte sich jedoch der Tatsache bewußt sein, daß jede Programmeinheit mit Ausnahme von Blockdatenunterprogrammen einen einzigen Modul erzeugt. Das Blockdatenunterprogramm erzeugt einen Modul für jeden benannten gemeinsamen Speicherblock. Externe Namen, d.h. Namen von Hauptprogrammen, Funktionsunterprogrammen, Subroutinenunterprogrammen einschließlich aller Einsprungnamen und Namen von gemeinsamen Speicherblöcken dürfen in einem ausführbaren Programm nicht doppelt auftreten. Im Fall, daß Namen von FORTRAN-Unterprogrammen und Namen, die vom Programmierer in seinem Programm festgelegt wurden, miteinander in Konflikt geraten, ist die Definition des Programmierers vorrangig.

Datendefinition

Die Eigenschaften einer Gruppe von Dateien, die in einem FORTRAN-Programm über die Gruppennummer angesprochen wird, werden in der Daten-Definitionstabelle festgelegt, die im

Speicher bei der Ausführung eines jeden FORTRAN-Programms vorhanden ist, das Ein-Ausgabe-Anweisungen enthält.

Für den normalen Betrieb von FORTRAN-Programmen während der Monitorabläufe bei einer gegebenen Anlagenausstattung wird eine Version dieser Tabelle - möglicherweise speziell auf diese Anlagenausstattung zugeschnitten - in die standardmäßige Modulbibliothek von FORTRAN eingeschlossen. Sie ordnet den Gruppen von Dateien, die Standardformat haben und auf Monitor-Geräten, möglicherweise aber auch auf anderen Geräten gespeichert sind, bestimmte Nummern zu.

Ein bei einer Anlagenausstattung tätiger Programmierer, der eine nicht-standardmäßige Zuordnung der Gruppen von Dateien und dieser Nummern benötigt, muß seine eigene Version der Tabelle dem Binder zusammen mit seinen übersetzten Programmeinheiten zuführen.

Der Makroaufruf Datendefinition (DD)

Besondere Versionen der Tabelle werden am einfachsten unter Verwendung des Makroaufrufs Datendefinition (DD) hergestellt, der durch den BBS-Assembler umgewandelt wird. Der Makroaufruf DD ist als ein Kennwort-Makroaufruf definiert. Die betreffende Makrodefinition ist in die standardmäßige Makrobibliothek eingeschlossen. Siehe Beschreibung "Assembler GBS/BBS", wo sich eine detaillierte Diskussion des Formats, der Definition und des Gebrauchs von Kennwort-Makroaufrufen findet.

Die Parameter des Makroaufrufs DD

DSREF - Gruppennummer der Dateien

Diese Nummer ist die ganze Zahl, die von FORTRAN-Programmen benutzt wird, um eine Gruppe von Dateien anzusprechen. Angabe dieses Parameters ist obligatorisch.

FILENO - Dateinummer

Im Falle, daß mehrere Dateien auf demselben Gerät gespeichert sind und diese Dateien durch dieselbe Gruppennummer angesprochen werden, muß jede dieser Dateien durch einen eigenen Makroaufruf DD definiert werden. Diese müssen mit FILENO fortlaufend numeriert werden. Sie sollten auch in dieser Reihenfolge in der Tabelle erscheinen. Fehlt dieser Parameter, so wird angenommen, daß er den Wert 1 hat.

DEVADDR - Symbolischer Gerätename

Dieser symbolische Gerätename (1 bis 8 Zeichen) wird verwendet, um dem Ablaufteil des BBS-Organisationsprogramms die Geräteidentifikation mitzuteilen, das wiederum die tatsächlichen Gerätezuordnungen vom Operateur empfängt. Dieser Name kann ein Name der Standard-Monitor-Geräte sein, so daß auf diese Weise eine Verbindung zwischen einer Gruppennummer und einem Monitor-Gerät hergestellt wird. Dieser Parameter ist obligatorisch.

DEVICE - Gerätetyp

Die genaue Bedeutung der verschiedenen zugelassenen Gerätetypen ist wie folgt:

TYPE:	9-Spur-Magnetbandgerät 432, 442, 4443 oder 4446
PRINTER:	Schnelldrucker 243 oder 4247
PUNCH 4:	Lochkartenstanzer 234
PUNCH 6:	Lochkartenstanzer 236
READER:	Lochkartenleser 4235 oder 237
TAPE:	Bedienungsblattschreiber

Dieser Parameter bewirkt, daß Ein-Ausgabe-Routinen, die für den betreffenden Gerätetyp benötigt werden, durch

den Binder in den Speicher geladen werden. Fehlt dieser Parameter, so wird TAPE angenommen.

DSNAME - Gruppenname der Dateien

Der aus acht Zeichen bestehende Name wird für das Schreiben von Standardetiketten und die Etikettprüfung verwendet. Fehlt dieser Parameter, so wird der Gruppenname der Datei in Etiketten nicht geprüft, bzw. ein Leername in zu schreibende Etikette eingesetzt.

Der Gebrauch von DUMMY als Gruppenname stellt ein bequemes Hilfsmittel dar, nicht gewünschte Ausgabe eines Programms zu unterdrücken. Es gibt an, daß die Ein-Ausgabe-Routinen zwar arbeiten, tatsächlich aber keine Ein-Ausgabe-Operationen ausführen sollen.

Insbesondere gilt folgendes:

a) Leseanweisungen führen Dateiendebedingungen herbei,

b) Schreibanweisungen haben keine Datenübertragung zur Folge,

c) Gerät-, Spulen- und Pufferinformationen haben keine Bedeutung und werden ignoriert.

VOLUME - Kennzeichnung für Datenträger

Dieser Parameter gibt die Kennzeichnung des Datenträgers an, auf dem die Datei gespeichert ist. Er wird von den Ein-Ausgabe-Steuerprogrammen verwendet, um die Quelle der Eingabe oder das Ziel der Ausgabe zu verifizieren; er kann z.B. eine Spulennummer sein. Er besteht normalerweise aus einer ganzen, bis zu 6stelligen Zahl. Fehlt dieser Parameter, so unterbleibt die Prüfung.

FILABL - Etikettvereinbarungen

Dieser Parameter beschreibt die Etikettvereinbarungen, für die eine der drei folgenden gewählt werden kann:

STD: Die Ein-Ausgabe-Routine prüft Standardanfangs- und -endetikette bzw. schreibt diese.

NSTD: Die Ein-Ausgabe-Routine prüft Etikette bei Eingabe nicht. Die Steuerung des Programms geht an die Etikettierungsroutine des Benutzers über, wenn LABADDR angegeben wurde. Andernfalls werden die Etikette überlesen. Bei Ausgabe geht die Steuerung an die Etikettierungsroutine des Benutzers, um nichtstandardmäßige Etikette zu bilden. Wurde LABADDR nicht angegeben, werden keine Etikette geschrieben.

NO: Die Ein-Ausgabe-Routine nimmt an, daß während der Eingabe keine Etikette auftreten. Bei Ausgabe erzeugt sie keine Etikette. Dieser Fall wird auch angenommen, wenn dieser Parameter überhaupt fehlt.

LABADDR - Routine zur Etikettierung

Durch diesen Parameter wird der symbolische Einsprungpunkt einer vom Benutzer vorgesehenen Etikettierungsroutine für die Verarbeitung von nichtstandardmäßigen Etiketten angegeben (siehe auch Parameter FILABL=NSTD).

LIFE - Anzahl der Tage

Die Ein-Ausgabe-Routine addiert diese Anzahl von Tagen zum gegenwärtigen Datum, um das Freigabedatum zu erhalten, das in das Etikett einer Ausgabedatei eingesetzt wird. Fehlt dieser Parameter, so wird Null angenommen.

BLKSIZE - Blocklänge in Bytes

Dieser Parameter gibt die Länge des größten Blocks an, der von der Dateiengruppe gelesen oder in die Dateiengruppe geschrieben werden soll. Aus ihm leitet sich die kleinstmögliche Größe des Pufferbereichs ab. Fehlt dieser Parameter, geschieht folgendes:

a) Bei Eingabe wird die entsprechende Angabe aus dem Standardanfangsetikett entnommen, wenn FILABL=STD ist. Andernfalls wird eine Standardgröße angenommen, wodurch eine Fehlerbedingung entstehen kann, wenn diese den Daten nicht entspricht.

b) Bei Ausgabe wird eine Standardgröße angenommen.

RECSIZE - Satzlänge in Bytes

Dieser Parameter gibt die Größe von Sätzen mit fester Länge sowie die maximale Größe von Sätzen mit variabler Länge und von undefinierten Sätzen an. Fehlt dieser Parameter, geschieht folgendes:

a) Bei Eingabe wird die entsprechende Angabe aus dem Standardanfangsetikett entnommen, wenn FILABL=STD ist. Andernfalls wird eine Standardgröße angenommen, wodurch eine Fehlerbedingung entstehen kann, wenn diese den Daten nicht entspricht.

b) Bei Ausgabe wird eine Standardgröße angenommen.

NUMBUF - Anzahl der Pufferbereiche

Es können ein oder zwei Pufferbereiche vorgesehen werden, wobei ein Bereich angenommen wird, wenn dieser Parameter fehlt.

RECFORM - Art der Sätze und Blockung

Die Art der Sätze und die Blockung wird durch einen der folgenden Schlüssel festgelegt:

FIXBLK: Sätze mit fester Länge, geblockt. FIXBLK wird angenommen, wenn dieser Parameter fehlt.

FIXUNB: Sätze mit fester Länge, ein Satz pro Block (ungeblockt).

VARUNB: Sätze mit variabler Länge; die Länge ist in den ersten vier Bytes eines Blockes enthalten, 1 Satz pro Block (ungeblockt).

VARBLK: Sätze mit variabler Länge; Länge ist in den ersten vier Bytes eines jeden Satzes enthalten, geblockt.

UNDEF: Logischer Satz ist identisch mit einem Block.

TYPEBUF - Anzeige

Wenn diese Anzeige gleich USURP gesetzt ist oder wenn dieser Parameter fehlt, kann ein teilweise gefüllter Pufferbereich geleert und für die Pufferung einer anderen Ein-Ausgabe-Übertragung verwendet werden. Ist sie gleich NOUSURP gesetzt, geschieht dies niemals. Auch geschieht dies nicht für Eingabedateien, die nicht zurückgesetzt werden können.

TYPEFLE - Art der Datei

Durch den Gebrauch dieses Parameters kann eine gewisse Dateisicherung erhalten werden. Die möglichen Werte für die Art der Datei haben folgende Bedeutung:

INPUT: Es werden keine Ausgabeoperationen zugelassen.

OUTPUT: Es werden keine Eingabeoperationen zugelassen.

INOUT: Es ist sowohl Ein- als auch Ausgabe zugelassen. Dieser Fall wird auch angenommen, wenn dieser Parameter überhaupt fehlt.

REWIND - Rückspulcode

Der Rückspulcode steuert die Interpretation einer Rückspulanweisung in einem FORTRAN-Programm in folgender Weise:

RWD: Einfaches Rückspulen eines Magnetbandes. Wird angenommen, wenn dieser Parameter fehlt.

UNLOAD: Rückspulen und Entladen eines Magnetbandes.

NORWD: Rückspulen nicht zugelassen.

CONTROL - Art der Formularvorschubsteuerung.

Das erste Zeichen von auszugebenden Drucksätzen wird entsprechend der angegebenen Art der Steuerung interpretiert:

ASA: Das erste Zeichen des Satzes steuert den Formularvorschub vor dem Druck entsprechend den Standardvorschriften der ASA, d.h.Zwischenraum, 0,1, + bedeuten einzeiligen bzw. zweizeiligen Vorschub, Vorschub auf erste Zeile des nächsten Blattes oder keinen Vorschub. Kein Vorschub wird angenommen, wenn dieser Parameter fehlt.

MACH: Das erste Zeichen wird direkt an den Drucker für die Steuerung des Formularvorschubs übertragen.

NO: Das erste Zeichen wird nicht für die Formularvorschubsteuerung verwendet, sondern wird gedruckt. Der Druck erfolgt einzeilig.

Das erste Zeichen eines Satzes, der auf Magnetband aufgezeichnet ist, wird, soweit nötig, übersetzt, um Umsetzerprogramme Band auf Drucker mit dem geeigneten Steuerzeichen zu versorgen.

Der Programmierer muß für jede Gruppe von Dateien Makroaufrufe DD schreiben, wenn er seine eigene Daten-Definitionstabelle verwendet. Das bedeutet, daß er die standardmäßigen Datendefinitionen nicht etwa nur teilweise verwenden oder diese ergänzen oder erweitern kann. Standard-Vorschriften, die übernommen werden sollen, müssen vielmehr ebenfalls für die spezielle Tabelle formuliert werden.

Beispiele für die Spezifikation von Makrobefehlen DD

Die folgenden drei Listen von Parametern werden verwendet, um die speziellen Lese-, Stanz- und Druckanweisungen zu übernehmen. Die zusätzlichen Parameter, die in jedem Fall den ersten beiden folgen, sind nicht erforderlich, wenn der Programmablauf unter der Steuerung des Monitors erfolgt; sie dienen jedoch dazu, die genannten Punkte hervorzuheben.

DD-Parameter für READ n,Liste

```
DSREF=97
DEVADDR=SYSINP
RECSIZE=80
TYPEBUF=NOUSURP
TYPEFLE=INPUT
REWIND=NORWD
```

DD-Parameter für PUNCH n,Liste

```
DSREF=98
DEVADDR=SYSOPT
RECSIZE=80
TYPEFLE=OUTPUT
REWIND=NORWD
```

DD-Parameter für PRINT n,Liste

```
DSREF=99
DEVADDR=SYSLST
TYPEFLE=OUTPUT
REWIND=NORWD
```

Die Satzlänge wurde im letzten Beispiel weggelassen, da sich möglicherweise manche Anlagenausstattungen der breiteren Druckzeile bedienen wollen.

Ein allgemeineres Beispiel

Ein allgemeineres Beispiel, und zwar das eines Programms, das ein Band liest, die geänderte Information auf ein anderes schreibt, sowie ein drittes Band als Zwischenspeicher benutzt, ist folgendes:

Parameter des ersten Makroaufrufs DD

```
DSREF=1
DEVADDR=ABC
DSNAME=MFL
FILABL=STD
VOLUME=935
NUMBUF=2
TYPEBUF=NOUSURP
TYPEFLE=INPUT
REWIND=UNLOAD
```

Parameter des zweiten Makroaufrufs DD

```
DSREF=2
DEVADDR=DEF
DSNAME=MFL
VOLUME=936
FILABL=STD
LIFE=7
BLKSIZE=1000
RECSIZE=100
NUMBUF=2
TYPEBUF=NOUSURP
TYPEFLE=OUTPUT
REWIND=UNLOAD
```

Parameter des dritten Makroaufrufs DD

```
DSREF=3
DEVADDR=TMPTP
BLKSIZE=2000
RECSIZE=2000
RECFORM=UNDEP
```

Erklärung

Diese Spezifikationen bewirken, daß die Ein-Ausgabespulen-etikette und die Eingabeetikette geprüft werden und eine neue Dateiengruppe mit demselben Namen und demselben Aufbau erzeugt wird. Letztere hat eine Aufbewahrungszeit von einer Woche. Doppelte Pufferung erhöht in vielen Fällen die Leistungsfähigkeit eines Programms. Der Zusatz UNLOAD hilft dem Operateur, die Bandspulen abzunehmen und die auf ihr gespeicherten Informationen zu sichern. Das Band, das nur für Zwischenspeicherung verwendet wird, erfordert außer der maximalen Satzlänge nur wenige Spezifikationen.