UNIVERSITÄT OSNABRÜCK

Rechenzentrum

FORTRAN IV

MIT EINEM ANHANG ÜBER SPEZIELLE
MÖGLICHKEITEN AUF DEM RECHNER TR 440

Universität Osnabrück

Rechenzentrum

Universität Osnabrück · 45 Osnabrück · Postfach 4469

PROGRAMMIERANLEITUNG

FORTRAN IV

von Dipl.-Ing. M. Brühl, Dipl.-Math. S. Schwenkler,
Dipl.-Phys. K.-J. Stüken (Rechenzentrum der TU Berlin)

für den Bereich Osnabrück (Universität und Fachhochschule)
überarbeitet von Dipl.-Math. K. Brauer

1. Auflage, Oktober 1975

INHALTSVERZEICHNIS

1.	Vorwort.....	1
2.	Einige Grundlagen	3
2.1	Aufbau einer Rechenanlage	3
2.1.1	Eingabegeräte	3
2.1.2	Ausgabegeräte	4
2.1.3	Die Zentraleinheit.....	4
2.1.3.1	Das Leitwerk	4
2.1.3.2	Das Rechenwerk	5
2.1.3.3	Der interne Speicher	6
2.2	Vom Quellenprogramm zum Objektprogramm	6
3.	Aufbau eines FORTRAN-Programms.....	8
3.1	Anweisungen (Statements)	8
3.2	Codierung von FORTRAN-Anweisungen	8
3.3	Grundelemente von FORTRAN	10
3.3.1	Schriftzeichen	10
3.3.2	Darstellung von Größen.....	11
3.3.3	Ganzzahlige Konstanten	12
3.3.4	Reelle Konstanten	12
3.3.5	Variable	13
4.	Arithmetische Ausdrücke und Anweisungen	15
4.1	Arithmetische Ausdrücke.....	15
4.2	Regeln für die Bildung arithmetischer Ausdrücke	16
4.3	Auswertung von arithmetischen Ausdrücken	17
4.3.1	Natürliche Rangordnung	17
4.3.2	Auswertung von Klammern	18
4.4	Arithmetische Anweisungen (Ergibtanweisungen).....	19
5.	Programmende	21
5.1	Die STOP-Anweisung	21
5.2	Die END-Anweisung	21

6.	Ein- und Ausgabeanweisungen (Einführung)	21
6.1	Vorbemerkungen.....	21
6.2	FORMAT-gesteuerte READ- und WRITE-Anweisung	22
6.3	Die FORMAT-Anweisung	22
6.4	Die I-Spezifikation	23
6.5	Die F-Spezifikation	25
6.6	Die H-Spezifikation (Ausgabe von Text).....	27
7.	Steuerungsanweisungen	29
7.1	Vorbemerkungen	29
7.2	Unbedingtes GOTO	29
7.3	Die arithmetische IF-Anweisung	30
8.	Typdeklarationen.....	33
9.	Standardfunktionen	35
10.	Benutzung indizierter Variablen	37
10.1	Indizierte Variablen	37
10.1.1	Felder (Arrays)	37
10.1.2	Indizes.....	37
10.1.3	Anordnung der Feldelemente im Speicher	38
10.2	Die DIMENSION-Anweisung	39
10.3	Die DO-Anweisung	41
10.4	Regeln für die Programmierung von DO-Schleifen	45
10.5	Die CONTINUE-Anweisung.l.....	48
11.	Erweiterung der Ein- Ausgabeanweisungen	49
11.1	Die E-Spezifikation	49
11.2	Die D-Spezifikation	51
11.3	Die G-Spezifikation	51
11.4	Ein-/Ausgabe von COMPLEX-Variablen.....	52
11.5	Die A-Spezifikation	52
11.6	Die X-Spezifikation	54
11.7	Die L-Spezifikation	55
11.8	Die Vorschubsteuerzeichen beim Drucken	55

11.9	Ergänzungen zur FORMAT-Anweisung	56
11.10	Ergänzungen zur Ein-/Ausgabeliste	61
11.11	Der Sc-alenfaktor P	64
11.12	Das variable FORMAT	66
12.	Logische Ausdrücke und Anweisungen	69
12.1	Logische Ausdrücke.....	69
12.1.1	Vergleichsausdrücke	69
12.1.2	Logische Operatoren	70
12.1.3	Auswertung von logischen Ausdrücken	72
12.2	Logische Anweisungen	73
13.	Weitere Steuerungsanweisungen.....	75
13.1	Bedingte GOTO's	75
13.1.1	Computed GOTO	75
13.1.2	Assigned GOTO	76
13.2	Die logische IF-Anweisung	78
14.	Unterprogramme.....	79
14.1	Statementfunktion	80
14.2	FUNCTION-Unterprogramme	81
14.3	SUBROUTINE-Unterprogramme	84
15.	Die COMMON-Anweisung	89
15.1	Die einfache COMMON-Anweisung.....	89
15.2	Der benannte COMMON-Block	92
16.	Variable DIMENSION in Unterprogrammen	95
17.	Die EXTERNAL-Anweisung	99
18.	Die DATA-Anweisung	101
19.	Die EQUIVALENCE-Anweisung.....	103
20.	Das BLOCK-DATA-Unterprogramm	107
21.	Formatfreies Lesen und Schreiben	109
22.	File-Handling	111
22.1	Die BACKSPACE-Anweisung	111

22.2	Die ENDFILE-Anweisung.....	112
22.3	Die REWIND-Anweisung	112
23.	Die DEFINEFILE-Anweisung	113
24.	FORTRAN-Literatur	115
	Sachverzeichnis.....	117
	Anhang 1 (Zusätzliche Möglichkeiten auf dem TR 440)	125
	Anhang 2 (Einige wesentliche Kommandos zum Bearbeiten von FORTRAN)	132
	Anahng 3 (Verweis auf spezielle TR-440 Literatur)	142

1. Vorwort

FORTRAN IV ist eine problemorientierte Programmiersprache, die insbesondere für die Bearbeitung mathematischer und technischer Probleme konzipiert wurde. FORTRAN ist eine Abkürzung für FORmula TRANslator (Formelübersetzer). Die Sprache ist genormt nach ANS X3.9 vom März 1966 (ANS=American National Standards).

Ziel dieser Programmieranleitung ist, dem Leser eine Einführung in den vollen Umfang der Sprache nach der oben zitierten Norm zu geben. In FORTRAN geschriebene Programme laufen auf den Rechenanlagen fast aller Hersteller. Fast immer jedoch werden von den einzelnen Herstellern zusätzliche Sprachelemente hinzugefügt, werden diese benutzt, so geht die Maschinenunabhänigkeit meist verloren. Bewußt wurden daher in dieser Anleitung nur die Standardelemente aufgenommen.

Im Rechenzentrum der Universität Osnabrück ist ein Rechner des Typs TR 440 (Hersteller: Computer Gesellschaft Konstanz) installiert.
Die Abteilung Vechta sowie der Schloßbereich (Raum EW 132) sind über Remote-Batch-Terminals an diesen Rechner angeschlossen. Das System des TR 440 enthält einen ausgereiften FORTRAN-Compiler, außerdem werden umfangreiche Testhilfen durch Systemkomponenten geboten.

Spezielle Kenntnisse der Mathematik werden nicht vorausgesetzt. Bei allen Programmierbeispielen wird der mathematische Hintergrund der Aufgabe, soweit notwendig, erläutert. Absichtlich wurden auch Beispiele auf der nichtnumerischen Datenverarbeitung aufgenommen, um zu zeigen, daß FORTRAN sich nicht nur für mathematische Programme eignet.

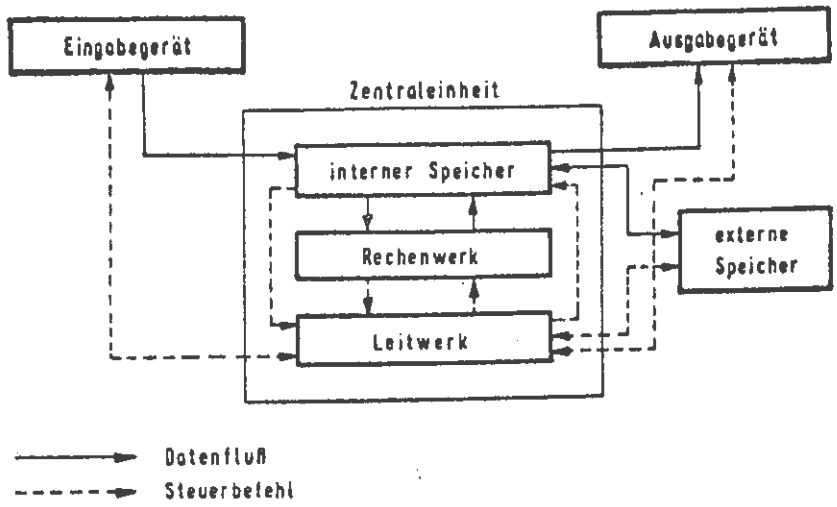
Ich danke der Zentraleinrichtung Rechenzentrum der Technischen Universität Berlin für die freundliche Überlassung des Manuskriptes, das sich bereits als begleitendes Kursmaterial bei Lehrveranstaltungen bewährt hat.

Für den Bereich Osnabrück (Universität und Fachhochschule) wurde diese 3. Auflage gegenüber der ersten an einigen Stellen verbessert.
Einige kleinere Änderungen betreffend die spezielle Osnabrücker Situation wurden von mir vorgenommen.
Insbesondere wurden in dieser 3. Auflage neu einige Anhänge aufgenommen, die die wesentlichen Zusatzinformationen für das Bearbeiten von FORTRAN-Programmen auf dem TR 440 enthalten.

Osnabrück, Oktober 1977 K. Brauer

2. Einige Grundlagen

2.1 Aufbau einer Rechenanlage



Wie das vorstehende Bild zeigt, besteht eine Rechenanlage aus 5 Grundeinheiten:

1.) Eingabegeräte
2.) Ausgabegeräte
3.) Leitwerk oder Steuerwerk
4.) Rechenwerk
5.) Interner Speicher

Die Einheiten 3.,4. und 5. werden zumeist räumlich zusammengefaßt und bilden dann die Zentraleinheit. Die im Bild sichtbaren externen Speicher wurden nur der Vollständigkeit halber eingezeichnet. Sie können vorhanden sein, müssen es aber nicht. Eine Rechenanlage ist auch ohne externe Speicher voll arbeitsfähig. Die Zentraleinheit und die externen Speicher befinden sich im Rechenzentrum der Universität Osnabrück, , da sie den eigentlichen TR 440 bilden.

Geräte des Typs 1 (Lochkartenleser) und des Typs 2 (Zeilendrucker) stehen zusätzlich in der Abteilung Vechta und im EW der Universität Osnabrück. Sie sind über eine Datenleitung an die Zentraleinheit angeschlossen. Ferner befinden sich an den Standorten Westerberg und Haste Geräte der Typen 1 und 2.

2.1.1 Eingabegeräte

Die Informationen (Programme und Daten), die in den Rechner eingegeben werden sollen, müssen ohne Ausnahme durch eines der Eingabegeräte gehen. Zwar besitzen die meisten Rechner eine Eingabetastatur, mit der Daten und Befehle direkt eingegeben werden können, dies erfolgt jedoch nur im Notfall. Die eigentliche Menge der zu verarbeitenden Daten muß mit wesentlich höherer Geschwindigkeit in den Rechner hineingehen. Dazu werden sie vorab in maschinenlesbarer Form auf einen Datenträger (Lochkarte, Lochstreifen, Beleg, Magnetband) gebracht. Beim Lesen des Datenträgers durch das Eingabegerät werden die Informationen in elektrische Signale umgewandelt und dem interen Speicher der Zentraleinheit über Kanäle zugeleitet. Typische Eingabegeräte sind:

Lochkartenleser, Lochstreifenleser, Belegleser, Terminaltastaturen, Magnetbandgeräte.

2.1.2 Ausgabegeräte

Die zur Ausgabe kommenden Informationen werden ebenfalls in Form von elektrischen Signalen zu den Ausgabeeinheiten geleitet und dort in Lochungen (Lochkarte, Lochstreifen), Druckzeichen (Schnelldrucker) oder Bewegungen (Zeichengeräte) auf einem Datenträger umgewandelt. Sie sind für den Benutzer entweder direkt lesbar oder müssen mit Hilfe eines weiteren Arbeitsganges entschlüsselt werden (z. B. Lochkartenbeschrifter). Teilweise können die Datenträger zur neuerlichen Eingabe in den Rechner verwendet werden.

2.1.3 Die Zentraleinheit

Dieser Abschnitt behandelt die wesentlichen Bauteile der eigentlichen Verarbeitungseinheit eines Rechners. Wie bereits unter 2.1 gezeigt, besteht eine Zentraleinheit aus drei Grundelementen:

1.) Leitwerk oder Steuerwerk
2.) Rechenwerk
3.) Interner Speicher (Arbeitsspeicher)

2.1.3.1 Das Leitwerk

Das Leitwerk, auch Steuerwerk oder Control Unit genannt, bildet den kompliziertesten Bestandteil der Zentraleinheit. Seine Aufgabe besteht darin, die Arbeitsgänge, die im restlichen Teil des Rechners vor sich gehen, zu überwachen. Hierzu gehört die Steuerung der zeitlichen Reihenfolge, in welcher die Befehle des Programms ausgeführt werden sollen, die Entschlüsselung und Interpretation der Befehle, sowie die Erzeugung von Signalen, welche im Rechenwerk, im internen Speicher oder in den Peripheriegeräten die Ausführung der Befehle bewirken. Zu diesem Zweck verfügt es über eine Anzahl von Spezialregistern, z. B. zum Speichern des Operations- und Adressenteils eines Befehls.

Die Reihenfolge, in der die Operationen ausgeführt werden sollen, ist dem Leitwerk mitzuteilen. Die einzige Möglichkeit, Instruktionen und Daten in den Rechner hineinzubekommen, ist über die Eingabegeräte. Soll das Leitwerk die Instruktionsfolge kennen, so muß es unmittelbaren Zugriff zum Speicher haben.

Aus diesem Grunde werden sowohl die zu Steuerung erforderlichen Angaben als auch die zu verarbeitenden Daten im Speicher untergebracht. Das Leitwerk "liest" in der Liste der gespeicherten Instruktionen, interpretiert jede einzelne Instruktion und weist eine der anderen Einheiten an, den Befehl auszuführen.
Das Leitwerk überwacht:

1.) die Eingabe der Daten
2.) das Speichern und Wiederauffinden der Daten
3.) die Verarbeitung der Daten im Rechner
4.) die Ausgabe der Daten
5.) seine eigene Tätigkeit

Die Tatsache, daß das Leitwerk Einfluß auf die Verarbeitung seiner eigenen Instruktionen besitzt, stellt eine der wesentlichsten Eigenschaften der heutigen Rechner dar, nämlich die Fähigkeit, sein eigenes Verhalten zu ändern.

2.1.3.2 Das Rechenwerk

Das Rechenwerk bildet denjenigen Teil eines Rechners, in dem die eigentliche Informationsverarbeitung stattfindet. Dazu werden die zu verarbeitenden Daten aus dem internen Speicher in die Spezialregister des Rechenwerks transportiert, dort miteinander verknüpft und danach evtl. wieder in den internen Speicher zurückgebracht.

Das Rechenwerk ist in der Lage, zwei Arten von Operationen durchzuführen, nämlich arithmetische und logische Operationen.

Die erste Gruppe umfaßt die vier Grundrechenarten: Addition, Subtraktion, Multiplikation und Division. Dabei wird eigentlich nur die Addition vom Rechenwerk beherrscht. Die übrigen Operationen werden auf die Addition zurückgeführt.

Die vom Rechenwerk ausführbaren logischen Operationen dienen dem Vergleich zweier Daten, wobei das dabei ermittelte Ergebnis für die Auswahl der zu durchlaufenden Programmzweige von ausschlaggebender Bedeutung ist.

Das Rechenwerk setzt sich aus folgenden drei Baugruppen zusammen:

1.) Addierwerk
2.) einige Register
3.) Rechensteuerung

Ad 1.) Da sämtliche arithmetischen Operationen auf eine Addition zurückzuführen sind, genügt die Ausrüstung mit einem Addierwerk.

Ad 2.) Die Register dienen zur Aufnahme der Operanden und der Zwischen- und Endergebnisse während des Rechenvorgangs.

Ad 3.) Die Rechensteuerung dient, wie der Name bereits erkennen läßt, der Steuerung und Überwachung der Rechenvorgänge innerhalb des Rechenwerks.

2.1.3.3 Der interne Speicher

Der interne Speicher, oft auch als Hauptspeicher eines Rechners bezeichnet, dient sowohl als Programm- als auch als Arbeitsspeicher. Ersteres insofern, als er das Programm aufnimmt, welches den Ablauf der Verarbeitung steuert.
Soweit jedoch der interne Speicher nicht mit Programminstruktionen belegt ist, steht er als Arbeitsspeicher zur Verfügung. Das bedeutet, daß er die Ergebnisse der Operationen aus dem Rechenwerk speichert, die von den Eingabeeinheiten und den externen Speichern kommenden Daten übernimmt und die Ergebnisse an die Ausgabeeinheiten und die externen Speicher abgibt. Jede Zelle des internen Speichers hat einen nur für sie selbst gültigen Namen: ihre Adresse. Soll ein Wert in den Speicher eingegeben werden, so gibt man diese Adresse an. Um den Wert wiederzuerhalten, gibt man die Adresse an und fordert, daß der Inhalt der Adresse geholt wird.

Technisch ist der interne Speicher meist als Magnetkernspeicher konzipiert, was dazu führt, daß er häufig in der Umgangssprache von vorneherein als Kernspeicher bezeichnet wird.

2.2 Vom Quellenprogramm zum Objektprogramm

Das vom Benutzer in FORTRAN-Anweisungen geschriebene Programm bezeichnet man als Quellenprogramm (Source Program). Um nun ein FORTRAN-Programm in eine für den Rechner "verständliche" Form zu bringen, ist ein Übersetzungsprogramm notwendig, ein sog. Compiler (=Übersetzer). Der Compiler übersetzt das FORTRAN-Programm in eine Folge von Instruktionen in der Maschinensprache. Dieses vom Compiler erzeugte Programm bezeichnet man als Objektprogramm (Object Program). Der schematische Ablauf bei der Bearbeitung eines FORTRAN-Programms ist folgender:

1.) Ablochen des FORTRAN-Programms, der Daten und der Jobsteuerkarten. Karten in der für einen Job vorgesehenen Folge anordnen.

2.) Einlesen des aus Steuerkarten, FORTRAN-Programm und evtl. Datenkarten bestehenden Jobs in den Rechner. Im allgemeinen sofortiges Verdrängen des Jobs auf externen Speicher, da andere Jobs höhere Priorität haben. Nach gewisser Wartezeit Beginn der Bearbeitung.

3.) Laden des FORTRAN-Compilers in den Kernspeicher und starten desselben. Der Compiler liest das FORTRAN-Programm und übersetzt es in die Sprache, die die Maschine verarbeiten kann. Gleichzeitig wird eine Liste des Quellenprogramms erzeugt, in der gegebenenfalls syntaktische und semantische Fehler gemeldet werden. War das Programm formal richtig, so liegt jetzt das sog. Montageobjekt vor.

4.) Jetzt tritt der Montierer in Aktion. Er befriedigt etwa noch vorhandene Aufrufe von Bibliotheksroutinen und erzeugt einen startfähigen Operator.

5.) Anschließend Starten des Objektprogramms (Operators). Dieser liest noch etwa notwendige Daten und produziert Ergebnisse. Treten hierbei noch Fehler auf, so werden diese gemeldet und das Programm gegebenenfalls abgebrochen.

6.) Ausdrucken der erzeugten Listen und der Ergebnisse über den Schnelldrucker.

Die unter 2.), 3.) und 4.) erwähnten Vorgänge, wie Laden, Montieren und Starten von Programmen, erfolgen durch Teile des Betriebssystems. Das Betriebssystem ist ein Paket von Programmen, die den Ablauf der einzelnen Jobs steuern und regeln. Anweisungen an das Betriebssystem werden mit Hilfe von Kommandos angegeben. Diese unterscheiden sich wesentlich von einer Rechenanlage zur anderen.

Bzgl. der Kommandos des TR 440 und der speziellen Erweiterungen der Sprache FORTRAN gegenüber dem standardmäßigen FORTRAN IV wird auf die Anhänge dieser Schrift verwiesen.

3. Aufbau eines FORTRAN-Programms

3.1 Anweisungen (statements)

Ein FORTRAN-Quellenprogramm setzt sich aus einer Folge von Anweisungen zusammen. Man unterscheidet ausführbare und nichtausführbare Anweisungen.

A) Die ausführbaren Anweisungen umfassen

a) arithmetische und boolesche (logische) Ergibtanweisungen.
Bei der Ausführung einer solchen Anweisung wird der jeweilige Wert einer einfachen oder indizierten Variablen durch das Ergebnis einer Rechnung oder den Wahrheitswert eines booleschen (logischen) Ausdrucks ersetzt.

b) Steueranweisungen.
Steueranweisungen ändern den Programmablauf (Sprungbefehle, Schleifen etc.) oder beenden die Ausführung des Programms.

c) Ein- und Ausgabeanweisungen.
Diese Anweisungen bewirken die Übertragung von Daten zwischen dem internen Speicher und einem externen Speichermedium.

B) Die nichtausführbaren Anweisungen enthalten

a) Spezifikationsanweisungen.
Spezifikationsanweisungen beschreiben Eigenschaften und Anordnung von Variablen, Feldern und Funktionsunterprogrammen.

b) Formatanweisungen.
Formatanweisungen dienen zur Beschreibung der Übertragung von Daten zwischen verschiedenen Speichereinheiten.

c) Unterprogramm-Anweisungen und Anweisungen zur Definition von Anweisungsfunktionen (statement functions).
Unterprogramm-Anweisungen definieren Namen von Subroutinen und Funktionsunterprogrammen und geben deren Anfang an.

3.2 Codierung von FORTRAN-Anweisungen

FORTRAN-Programme werden im allgemeinen auf 80-spaltigen Lochkarten abgelocht. Dabei gelten folgende Regeln:

1) Anweisungen müssen in den Spalten 7 bis 72 einschließlich stehen.
Auf einer Karte darf nur eine Anweisung stehen.

Rechenzentrum der Universität Osnabrück

2) Reicht der Platz einer Karte für eine Anweisung nicht aus, so können bis zu 19 Fortsetzungskarten verwandt werden. Eine Fortsetzungskarte enthält, im Gegensatz zur ersten Karte einer Anweisung, in Spalte 6 weder eine Leerstelle (blank) noch eine Null.

3) In Spalte 1 bis 5 einschließlich der Anfangskarte einer Anweisung kann eine Anweisungsnummer (label) stehen, die aus 1 bis 5 Dezimalziffern besteht. Führende Nullen werden dabei ignoriert. Anweisungsnummern können in beliebiger Reihenfolge angegeben werden. Ihr Zahlenwert hat keinen Einfluß auf die Reihenfolge der Abarbeitung der Anweisungen. Es darf in einem Haupt- oder Unterprogramm die gleiche Anweisungsnummer zur Kennzeichnung nur einer Anweisung benutzt werden.

4) Die Spalten 73 bis 80 haben für FORTRAN-Compiler keine Bedeutung und können daher beliebig für Kennzeichnungszwecke verwandt werden.

Um bessere Lesbarkeit zu erreichen, können in FORTRAN-Programme beliebig Leerstellen (blanks) eingeführt werden. In Zeichenketten jedoch sind Leerstellen signifikant.
Zur übersichtlichen Gestaltung eines Programmes können sich an beliebiger Stelle zwischen den Anweisungen, außer unmittelbar vor Fortsetzungskarten, Kommentarkarten befinden.
Kommentarkarten sind durch ein C in Spalte 1 gekennzeichnet, der Kommantartext steht in den Spalten 2 bis 80.
In den Spalten 2 bis 80 sind nur FORTRAN-Zeichen (s. 3.3.1) erlaubt.

Beim Auflisten des Programms entspricht eine Lochkarte genau einer Zeile des mit dem Zeilendrucker erzeugten Listings.

3.3 Grundelemente von FORTRAN

3.3.1 Schriftzeichen

Die in FORTRAN-Programmen zulässigen Zeichen (characters) sind

a) alphabetische Zeichen (Buchstaben):

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

b) numerische Zeichen (Ziffern):

1 2 3 4 5 6 7 8 9 0

c) Sonderzeichen:

+ - / = ( ) . , $ *

Zu den Sonderzeichen zählt noch die Leerstelle (blank).

Buchstaben und Ziffern werden zur Unterscheidung von den Sonderzeichen auch alphanumerische Zeichen genannt.

3.3.2 Darstellung von Größen

In FORTRAN unterscheidet man 5 Typen von Größen:
ganzzahlige, reelle, doppeltgenaue, komplexe und logische (boolesche) Größen.
Durch den Typ sind die mathematischen bzw. logischen Eigenschaften einer Größe und ihre interne Darstellung festgelegt.
Größen können als Konstanten oder Variablen vorliegen.

a) Konstante:

Eine Konstante ist ein fester, unveränderlicher Wert, der in FORTRAN durch eine Zeichenreihe dargestellt wird. Numerische Konstanten (ganzzahlig, reell, doppeltgenau und komplex) sind Zahlen.

b) Variable:

Eine Variable ist eine Größe, die an verschiedenen Stellen innerhalb eines Programms oder bei verschiedenen Ausführungen eines Programms ihren Wert ändern kann. Sie wird durch einen Variablennamen (identifier) gekennzeichnet. Ein Variablenname besteht aus 1 bis 6 alphanumerischen Zeichen, von denen das erste ein Buchstabe sein muß. Variablennamen dürfen keine Sonderzeichen enthalten.

Beispiele
(Im folgenden wollen wir das O in FORTRAN-Beispielen zur Unterscheidung von der Null mit / durchstreichen):

A
S72
ØTTØ3
RITUB
R01NEU
FØRMEL
J12345

Folgende Variablennamen sind falsch:

LEXIKØN	(länger als 6 Zeichen)
BI-SEX	(Sonderzeichen unzulässig)
6A	(beginnt nicht mit einem Buchstaben)

3.3.3 Ganzzahlige Konstanten

Eine ganzzahlige Konstante (INTEGER-Konstante) ist eine ganze Zahl, die als Folge von Dezimalziffern ohne Dezimalpunkt geschrieben wird, vor der ein Vorzeichen (+ oder -) stehen kann. Fehlt das Vorzeichen, so wird die Zahl als positiv angenommen. Der Wertebereich der INTEGER-Größen hängt von der jeweiligen Rechenanlage ab.

Beispiele:

```
11
123456
919 888
-125
0
+583
-000003
```

3.3.4 Reelle Konstanten

Eine reelle Konstante (REAL-Konstante) ist eine Dezimalpunktkonstante oder eine von einem Exponent gefolgte Dezimalpunkt- oder INTEGER-Konstante. Dabei versteht man unter einer Dezimalpunktkonstanten eine Folge von Dezimalziffern, die von einem Dezimalpunkt angeführt, unterbrochen oder gefolgt wird. Ein Dezimalexponent wird geschrieben als Buchstabe E, dem eine INTEGER-Konstante folgt. Der Dezimalexponent bewirkt eine Multiplikation der vor ihm stehenden Konstanten mit derjenigen Potenz von 10, die durch die INTEGER-Konstante des Dezimalexponenten gegeben ist.

Eine REAL-Konstante kann mit einem Vorzeichen versehen werden. Fehlt das Vorzeichen, so wird die Konstante als positiv gewertet.

Die Anzahl der signifikanten Stellen der REAL-Konstanten und der Bereich des Exponenten hängen von der jeweiligen Rechenanlage ab.

Beispiele für REAL-Konstanten:

```
1.
-3.00
0.7831
+.82
3158.279
.0
+84.199E 3      (bedeutet: 84199.)
.396 E+21
123E-5          (bedeutet: 0.00123)
.000 0817
```

Folgende REAL-Konstanten sind falsch:

323	(Dezimalpunkt oder Dezimalexponent fehlt)
-7.14E	(nach E fehlt die INTEGER-Konstante)
.531 E -10.	(im Exponenten darf nur eine ganze Zahl ohne Dezimalpunkt stehen)

3.3.5 Variable

In FORTRAN unterscheidet man zwei Arten von Variablen, nämlich einfache Variable und indizierte Variable.

Der Typ (ganzzahlig, reell, doppeltgenau, komplex oder logisch) der jeweiligen Variablen muß im allgemeinen durch eine Typ-Spezifikationsanweisung festgelegt werden. Geschieht dieses nicht, so ist die Variable durch ihren Variablennamen in folgender Weise spezifiziert:

> Variable, deren Namen mit einem I, J, K, L, M oder N beginnen, werden als INTEGER-Größen aufgefaßt.
> Alle anderen Variablen gelten als vom Typ REAL.

Eine einfache Variable gibt einen einzelnen Wert wieder, der auf einem Speicherplatz gespeichert ist, auf den sich der Name der Variablen bezieht. Der Wert der Variablen wird intern in der gleichen Weise behandelt wie eine Konstante desselben Typs.

Mit indizierten Variablen werden wir uns später befassen.

4. Arithmetische Ausdrücke und Anweisungen

4.1 Arithmetische Ausdrücke

Oft wird ein Programm erstellt, um mathematische Formeln auszuwerten. Die Programmiersprache FORTRAN erleichert die Darstellung der Formeln, da die mathematische Schreibweise so weit wie möglich nachempfunden wird.
Seien A und B Konstanten oder Namen für Variable (siehe 3.3.5), dann stellt

A o B

einen arithmetischen Ausdruck dar. Das Zeichen o steht für einen der fünf verschiedenen arithmetischen Operatoren:

Arithm. Operator	Bedeutung
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
**	Exponentiation

Beispiele:

```
ALPHA + BETA
IX + IY - IZ
3.*Q3P4/Z17
16.34-Q3**5
STV+DRUCK**2-ANT*13.9254
```

Es sei darauf hingewiesen, daß die obige Definition des arithmetischen Ausdruckes rekursiv zu verstehen ist, d. h. sind A und B arithmetische Ausdrücke, so ist A o B ebenfalls ein arithmetischer Ausdruck. Die Operatoren + und - kann man als einstellige Operatoren benutzen, z. B. + B oder - B. Diese Ausdrücke stellen einen Spezialfall von A-B mit A=0.0 dar. Der Operator + kann hierbei weggelassen werden. Die primitivste Form eines arithmetischen Ausdruckes ist also die Konstante und die Variable.

Welche Gestalt ein arithmetischer Ausdruck im einzelnen annehmen kann, soll nun angegeben werden.

1.	Eine Konstante	3.141592
2.	Eine einfache Variable	X
3.	Eine indizierte Variable (siehe Kap.10)	Y(9)
4.	Ein Funktionsaufruf (siehe Kap.9)	COS(Z)
5.	Arithmetische Operanden, die durch einen arithmetischen Operator verknüpft werden	X-Y+A
6.	Ein geklammerter arithm. Ausdruck	(BETA)

4.2 Regeln für die Bildung arithmetischer Ausdrücke

Für die Bildung arithmetischer Ausdrücke gelten folgende Regeln:

1.) Wird ein arithmetischer Ausdruck aus mehr als einem Operanden gebildet, so müssen die verwendeten Operanden durch einen arithmetischen Operator verknüpft werden. Der mathematische Ausdruck $X \cdot Y$ müßte geschrieben werden als

X * Y

Die Ausdrücke X · Y oder XY sind unzulässig. XY würde z. B. als Variablenname interpretiert.

2.) Bei der Bildung eines arithmetischen Ausdruckes dürfen zwei Operatoren nicht unmittelbar aufeinanderfolgen.

falsch	richtig
X**/Y	X*1./Y
X*-Y	X*(-Y) oder -Y*X
X***-1.	X***(-1.)

3.) In einem arithemtischen Ausdruck dürfen mit Hilfe der arithmetischen Operatoren Operanden unterschiedlichen Typs miteinander verbunden werden. Die zulässigen Kombinationen gibt folgende Tabelle wieder:

Operator +,-,*,/		Typ des rechten Operanden			
		INTEGER	REAL	DOUBLE a)	COMPLEX b)
Typ des linken Operanden	INTEGER	INTEGER	REAL	N	N
	REAL	REAL	REAL	DOUBLE	COMPLEX
	DOUBLE a)	N	DOUBLE	DOUBLE	N
	COMPLEX b)	N	COMPLEX	N	COMPLEX

a) Operand vom Typ doppelte Genauigkeit
b) Operand vom Typ Komplex
N = diese Kombination ist unzulässig

Steht kein N in der Tafel, so zeigt der angegebene Typ den Typ des Ergebnisses an.

Die Tabelle ist nach folgender Hierarchie aufgebaut:

1. COMPLEX
2. DOUBLE PRECISION
3. REAL
4. INTEGER

Werden zwei Operanden verschiedenen Typs miteinander verknüpft, so ist der entstehende Ausdruck vom Typ des in der Hierarchie höher stehenden Operanden, z.B. ergibt REAL + COMPLEX wieder COMPLEX.

4.) Werden zwei Operanden durch den Exponentiationsoperator miteinander verknüpft, so muß der Exponent vom Typ INTEGER, REAL oder DOUBLE PRECISION sein. Die zulässigen und unzulässigen Kombinationen gibt folgende Tabelle wieder:

Operator **		Typ des Exponenten			
		INTEGER	REAL	DOUBLE	COMPLEX
Typ der Basis	INTEGER	INTEGER	N	N	N
	REAL	REAL	REAL	DOUBLE	N
	DOUBLE	DOUBLE	DOUBLE	DOUBLE	N
	COMPLEX	COMPLEX	N	N	N

N = diese Kombination ist unzulässig

In einem Ausdruck A**B, wobei B vom Typ REAL ist, muß A positiv sein, da derartige Exponentiation mittels logarithmischer Rechnung vorgenommen werden. Es gilt nämlich

$$a^b = e^{b*\ln(a)}$$

Der ln(a) ist aber nur für a>0 erklärt.
Ist jedoch B vom Typ INTEGER, so darf A auch negative Werte annehmen, da derartige Exponentiationen über eine mehrfache Multiplikation vorgenommen werden.

4.3 Auswertung von arithmetischen Ausdrücken

4.3.1 Natürliche Rangordnung

Die Auswertung von arithmetischen Ausdrücken erfolgt in FORTRAN nach einer bestimmten Rangordnung, die wie folgt festgelegt ist:

1. Stufe Funktionsaufrufe und Ausdrücke in Klammern (siehe Kap. 4.3.2)
2. Stufe Exponentiation
3. Stufe Multiplikation und Division
4. Stufe Addition und Subtraktion

Operationen in gleicher Stufe werden grundsätzlich von links nach rechts ausgewertet.

Beispiele: (die Ri bzw. Li stehen abkürzend für Zwischenresultate, →Ri bzw. →Li steht für "Ergebnis wird in Ri bzw. Li gespeichert"):

a) Auswertung von A**B**C

1.) A**B→R1
2.) R1**C → R2 (Ergebnis)

b) Auswertung von I*J/K

1.) I*J → L1
2.) I1/K←L2 (Ergebnis)

Beim Schreiben arithmetischer Ausdrücke vom Typ INTEGER ist besondere Vorsicht geboten. Der Rechner rundet weder auf noch ab, sondern ein eventuell auftretender Dezimalbruch wird abgehackt, z.B. 11/3=3.

c) Auswertung von A**B/C+D*E*F-G

1.) A**B →R1
2.) R1/C →R2
3.) D*E →R3
4.) R3*F →R4
5.) R4+R2→R5
6.) R5-G →R6 (Ergebnis)

4.3.2 Auswertung von Klammern

Die Rangordnung wird durch das Setzen von Klammern aufgehoben; denn wie in der Algebra müssen in arithmetischen Ausdrücken Klammern gesetzt werden, um bei der Auswertung eine besondere Reihenfolge zu erzwingen. Eine Schachtelung der Klammern ist erlaubt. Treten in einem arithmetischen Ausdruck Klammern auf, so werden zuerst diese ausgewertet. Dabei ist zu beachten, daß die Auswertung der geschachtelten Klammern von innen nach außen durchgeführt wird. Die Berechnung des in der Klammer stehenden Teilausdruckes wird unter Beachtung der Rangordnung durchgeführt. Dieser Klammerausdruck nimmt dann die Stellung eines Operanden ein.

Beispiele

a) Auswertung von ((X + Y) / (X - Y) + A3)* B

1.) X+Y ⟶ R1
2.) X-Y ⟶ R2
3.) R1/R2 ⟶ R3
4.) R3+A3 ⟶ R4
5.) R4*B ⟶ R5 (Ergebnis)

Bemerkung: Überflüssige Klammerpaare beeinflussen nicht das Ergebnis, deshalb sollte man im Zweifelsfall ein Klammerpaar setzen.

Weitere Beispiele

mathematische Schreibweise	FORTRAN - Schreibweise
$\frac{(a-b)\varepsilon}{c-q}$	(A - B) * EPS) / (C - Q)
$\frac{x+y}{x-y}$	(X + Y) / (X - Y)
$x+\frac{y}{x-y}$	X + Y / (X - Y)
$\frac{x+y}{x}-y$	(X + Y) /X - Y
$x+\frac{Y}{x}-y$	X + Y/X - Y
$(a+b)^{3n}$	(A + B)**(3*N)
$(a+b)^{3}n$	(A + B)**3*N
$a+b^{3}n$	A + B**3*N

4.4 Arithmetische Anweisungen (Ergibtanweisungen)

Die allgemeine Form der arithmetischen Anweisung lautet:

V = A

Dabei ist V eine einfache oder indizierte Variable und A ein arithmetischer Ausdruck. Die arithmetische Anweisung besagt, daß der Variablen V der Wert zuzuweisen ist, der sich aus der Berechnung des arithmetischen Ausdrucks A ergibt. Die arithmetische Anweisung ist keine algebraische Gleichung, obwohl sie mit dem Gleichheitszeichen geschrieben wird. Tritt eine Variable auf beiden Seiten des Gleichheitszeichens auf, so bedeutet dies lediglich, daß der

Variablen ein neuer Wert zuzuweisen ist, der sich unter Verwendung ihres alten Wertes errechnet.

```
X = X + 2.
```

bedeutet: Nimm den Inhalt von X (genauer, den Inhalt des Speicherwortes mit der symbolischen Adresse X), vermehre ihn um 2. und speichere das Ergebnis wieder nach X. Der alte Wert von X geht also verloren!

Beispiele:

```
PI = 3.141592
A  = R
I  = I + 1
Q  = A*B/(C + D)
Y  = YM + RADIUS
```

Für den Fall, daß die Variable auf der linken Seite der arithmetischen Anweisung vom gleichen Typ ist wie der arithmetische Ausdruck auf der rechten Seite, erübrigt sich eine nähere Erläuterung. Ist dies jedoch nicht der Fall, so wird der arithmetische Ausdruck seinem Typ entsprechend berechnet und nach der Umwandlung in den Typ der Variablen auf deren Speicherplatz gespeichert.

Beispiel:

```
A = 4.0
I = A + 1.9
```

Wirkung: Zunächst wird der REAL-Zahl A der Wert 4.0 zugewiesen. Dann wird der Ausdruck A + 1.9 berechnet. Das Ergebnis ist 5.9, der Typ ist REAL. Die Variable I ist jedoch vom Typ INTEGER. Es wird nun der Dezimalbruch abgehackt, der ganzzahlige Anteil in eine INTEGER-Zahl umgewandelt und der Variablen I zugewiesen, d.h. I erhält den Wert 5.

In der folgenden Tabelle sind die in FORTRAN zulässigen Typkombinationen zwischen Variable und arithmetischem Ausdruck zusammengestellt:

Typ der Variablen V	Typ des arithmetischen Ausdrucks A			
	INTEGER	REAL	DOUBLE	COMPLEX
INTEGER	INTEGER	INTEGER	INTEGER	N
REAL	REAL	REAL	REAL	N
DOUBLE	DOUBLE	DOUBLE	DOUBLE	N
COMPLEX	N	N	N	COMPLEX

N = diese Kombination ist unzulässig. DOUBLE ≙ DOUBLE PRECISION

5. Programmende

5.1 Die STOP-Anweisung

Die STOP-Anweisung ist eine ausführbare Anweisung. Sie kennzeichnet das logische Ende eines Programms. Sie hat die Form:

```
        STOP
oder    STOP n
```

Hierbei ist n eine Ziffernkette, bestehend aus 1 bis 5 Ziffern. Bei der Ausführung der STOP-Anweisung wird das Programm an dieser Stelle beendet und kann nicht wieder neu gestartet werden. Die STOP-Anweisung ist die letzte ausführbare Anweisung einer Programmeinheit. Verschiedene n können dazu benutzt werden, verschiedene STOP's eines Programms zu unterscheiden.

5.2 Die END-Anweisung

Die END-Anweisung ist eine nicht ausführbare Anweisung. Sie hat die Form:

```
END
```

Die END-Anweisung kennzeichnet für den Compiler das physikalische Ende eines Programms. Jedes Haupt- oder Unterprogramm muß mit dieser Anweisung schließen. Sie darf jeweils in einer Programmeinheit nur einmal auftreten.

6. Ein- und Ausgabeanweisung (Einführung)

6.1 Vorbemerkungen

In einem arithmetischen Ausdruck können bekanntlich Konstanten und Variablen auftreten. Ehe man einen arithmetischen Ausdruck auswerten kann, müssen den Variablen Werte zugewiesen worden sein. Das kann z.B. mittels einer arithmetischen Anweisung oder mittels Einlesen des aktuellen Wertes geschehen. Es sei bemerkt, daß - liest man den Wert ein - ein flexibleres Programm geschaffen worden ist; denn abhängig von den eingelesenen Werten werden die Werte der anderen Variablen errechnet. Um Informationen über die Werte von Variablen zu erhalten, muß man diese vom Programm her ausgeben. Zuerst werden wir die Datenübertragung von Lochkarten in den Kernspeicher und vom Kernspeicher zum Zeilendrucker kennenlernen (Batchbetrieb). Im Time-Sharing-Betrieb entspricht dies der Datenübertragung vom Terminal zum Kernspeicher bzw. umgekehrt.

6.2 FORMAT-gesteuerte READ- und WRITE-Anweisung

Die READ- und die WRITE-Anweisungen gehören zu der Gruppe der ausführbaren Anweisungen. Sie haben folgende allgemeine Form

über Terminal : Eingabe ⇒ Kanal 8
Ausgabe ⇒ " " 9

Spalte 7
↓
READ (Kanal, Num) Liste
bzw.
WRITE (Kanal,Num) Liste

Dabei gilt:

Kanal ist eine INTEGER-Konstante oder -Variable größer als Null die die periphere Eingabeeinheit bzw. Ausgabeeinheit angibt.

Num ist die Statementnummer der zugehörigen FORMAT-Anweisung (siehe 6.3).

Liste steht stellvertretend für eine Liste von Variablen, die eingelesen bzw. eingegeben werden sollen. Die einzelnen Listenelemente werden durch Kommata voneinander getrennt.

Beispiele:

a) READ (1, 2) MAX, MORITZ, ADAM, EVA
über den Eingabekanal 1 sollen die Werte für MAX, MORITZ, ADAM und EVA eingelesen werden. 2 ist die Statementnummer der zugehörigen FORMAT-Anweisung. (siehe 6.3)

b) WRITE (10, 15) MAX, MORITZ, ADAM, EVA
über den Ausgabekanal 10 sollen die Werte für MAX, MORITZ, ADAM, EVA ausgegeben werden. Hierbei stellt 15 die (Statement-) Nummer der zugehörigen FORMAT-Anweisung (siehe 6.3) dar.

6.3 Die FORMAT-Anweisung

Die FORMAT-Anweisung ist eine nichtausführbare Anweisung. Sie darf an beliebiger Stelle zwischen den Anweisungen eines Programms stehen. Mit Hilfe dieser Anweisung legt man fest, in welchem Druckformat auf dem Zeilendrucker ausgegeben werden soll, ebenfalls bestimmt man, in welchem Format die Daten von den Lochkarten gelesen werden sollen. Der allgemeine Aufbau der FORMAT-Anweisung ist wie folgt:

Spalte 5 7
↓ ↓
NUM FORMAT (. . . . .)

NUM ist die Statementnummer des FORMAT's.
Jede FORMAT-Anweisung muß mit einer Statementnummer versehen sein.

Innerhalb der runden Klammer findet man die Spezifikationen, die die Übertragung beim Lesen bzw. Schreiben festlegen. Werden für das Lesen einer Lochkarte bzw. das Drucken einer Zeile mehrere Spezifikationen in der FORMAT-Anweisung angegeben, müssen sie durch Komma voneinander getrennt werden.

Es ist möglich, daß sich mehrere Lese- und Schreibanweisungen auf eine FORMAT-Anweisung beziehen.

6.4 Die I-Spezifikation

Diese Spezifikation gibt die Möglichkeit, INTEGER-Variable zu übertragen.
Der allgemeine Aufbau lautet:

r I w

r stellt den Wiederholungsfaktor dar und muß eine positive ganzzahlige vorzeichenlose Konstante sein. Hat r den Wert 1, so kann r auch fortgelassen werden

w ist ebenfalls eine positive ganzzahlige vorzeichenlose Konstante.
Sie gibt die Anzahl der Stellen des Datenbereichs an, aus dem der jeweilige INTEGER-Wert gelesen bzw. in den dieser Wert gedruckt werden soll.

Eingabe

Die einzulesene Dezimalzahl muß ohne Dezimalpunkt abgelocht werden. Sie kann mit einem vor der Zahl stehenden Vorzeichen versehen sein. Ist kein Vorzeichen angegeben, so wird der Wert als positiv angesehen. Zwischenräume (blanks) werden als Nullen interpretiert. Die Zahl wird innerhalb des w Stellen langen Bereichs rechtsbündig erwartet.

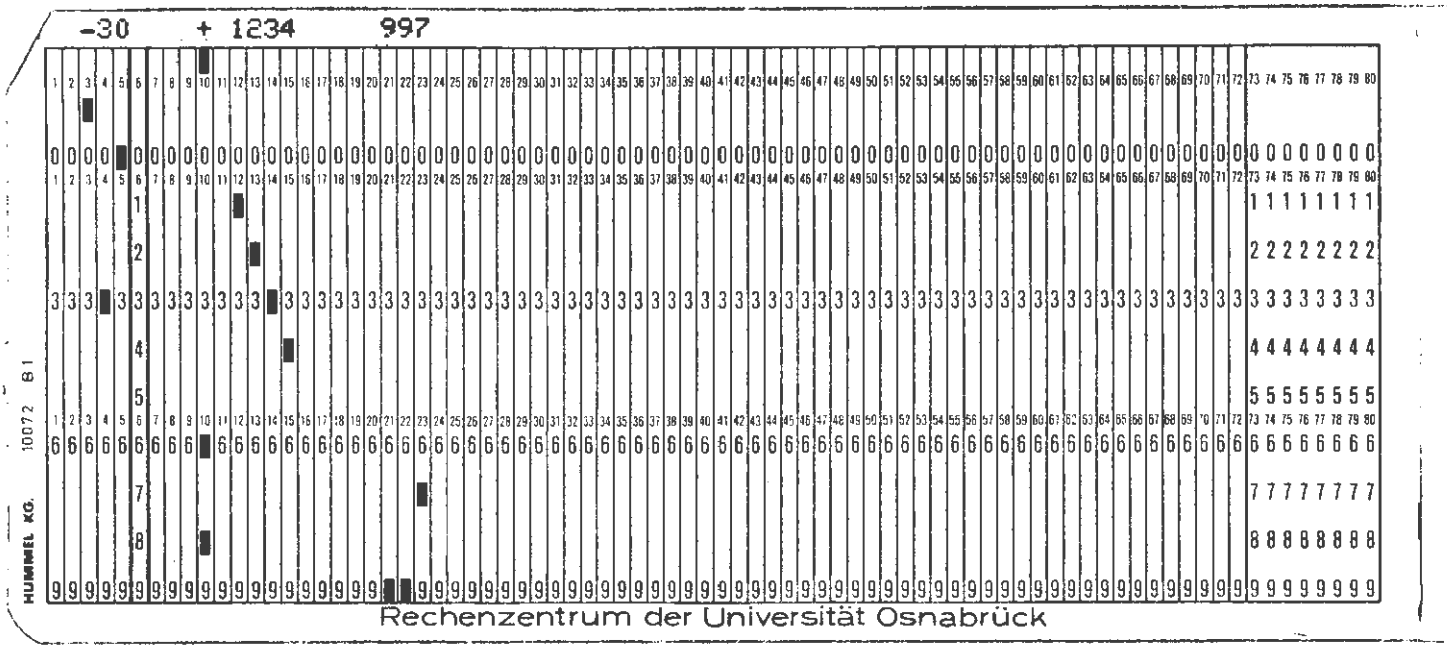
Beispiel 1

Vom Programm sollen die Werte für MAX, MORITZ und IDA eingelesen werden.

```
C
    READ,(1,10) MAX,MORITZ,IDA
 10 FORMAT(I5,2I10)
C
```

Wirkung: Lese von der nächsten Datenkarte ab Spalte 1 die Werte für MAX, MORITZ, IDA im angegebenen Format.

Zugehörige Datenkarte



Innerhalb der Spalte 1 bis 5 steht der Wert von MAX
" " " 6 ! 15 " " " " MORITZ
" " " 16 ! 25 " " " " IDA

MAX erhält den Wert - 30, MORITZ den Wert 1234
und IDA den Wert 9870.

Ausgabe

Der Wert der INTEGER-Variablen wird in den w Stellen langen Bereich geschrieben. Dabei wird eine Stelle für das Vorzeichen benötigt. Ist die Zahl positiv, wird ein Zwischenraum ausgegeben. Analog zur Eingabe wird der Wert der Variablen rechtsbündig in den Bereich gedruckt. Vorlaufende Nullen werden unterdrückt.

Beispiel 2

Die im Beispiel 1 angegebenen Werte sollen ausgedruckt werden.

```
C
      WRITE (2,15) MAX,MORITZ,IDA
 15   FORMAT (10X,I5,2I10)
C
```

Wirkung: In der nächsten Zeile werden zuerst 10 Leerzeichen - 10 X besagt, gib 10 Leerzeichen aus - danach der Wert von MAX im Format I 5 und die Werte von MORITZ und IDA im Format I 10 ausgegeben.

Ergebnisausdruck

-30 1234 98700

Für die Ausgabe sei vorerst empfohlen, die FORMAT-Anweisung mit 10X beginnen zu lassen.

6.5 Die F-Spezifikation

Mittels dieser Spezifikation kann man REAL-Variable übertragen.
Der allgemeine AUfbau lautet

r F w.d

r stellt den Wiederholungsfaktor dar (siehe I-Spezifikation)
w gibt die Stellenanzahl des Bereichs an (siehe I-Spezifikation).
d ist ebenfalls eine positive ganzzahlige vorzeichenlose Konstante. Sie gibt die Anzahl der Stellen nach dem Dezimalpunkt an.

Achtung: Es sollte stets gelten $w \geq d + 3$

Eingabe

Die einzulesende Dezimalzahl muß als Dezimalpunktkonstante rechtsbündig im angegebenen Bereich abgelocht sein. Ihr kann ein Vorzeichen voranstehen. Dem Dezimalpunkt müssen d Stellen, die zu diesem Bereich gehören, folgen. (Bezüglich Vorzeichen und Zwischenräume siehe auch I-Spezifikation-Eingabe) (Eine Erweiterung hierzu wird in Kapitel 11. angegeben). Steht ein Punkt im Eingabefeld, so hat seine Stellung Vorrang. Enthält das Eingabefeld nur w Ziffern (ggf. ein Vorzeichen und w-1 Ziffern), so werden die letzten d Ziffern als Dezimalstellen gedeutet.

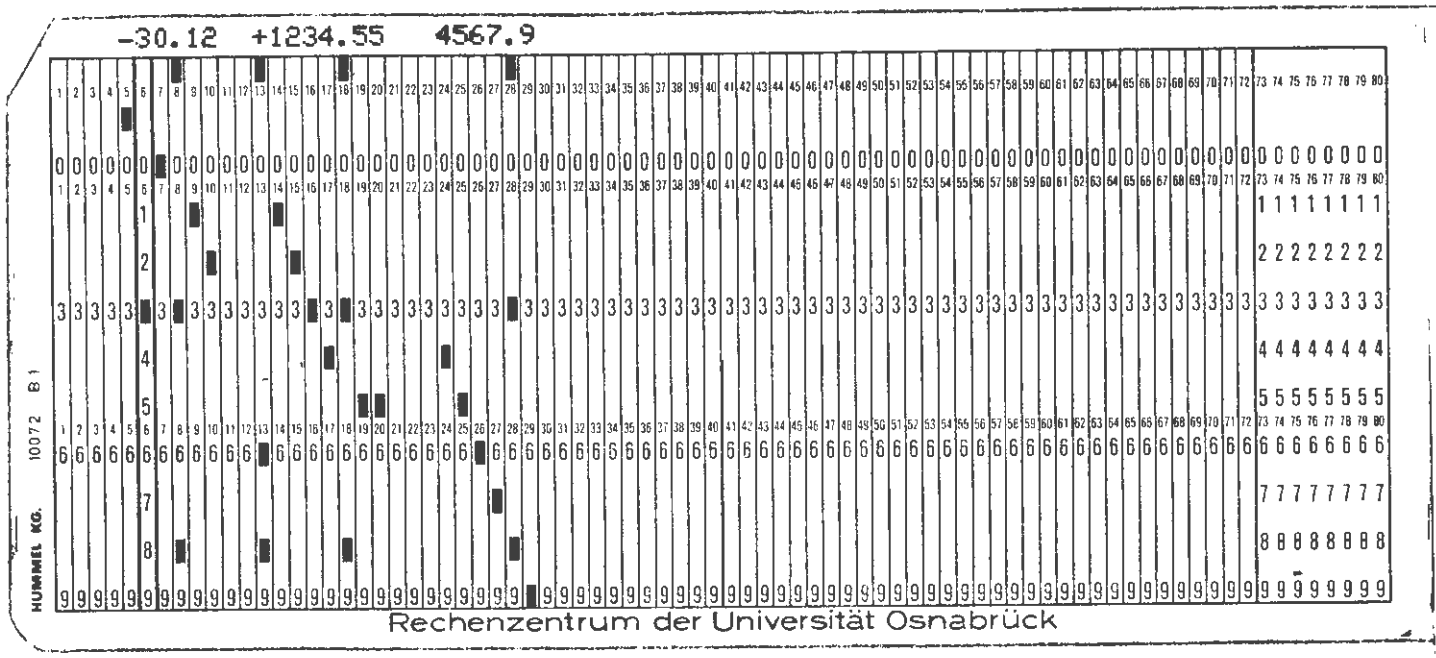
Beispiel 1

Vom Programm sollen die Werte für ADAM, EVA, ASTOR eingelesen werden

```
      READ (1,20) ADAM, EVA, ASTOR
20    FORMAT (3F10.2)
```

Wirkung: Lese von der nächsten Datenkarte ab Spalte 1 die Werte für ADAM, EVA, ASTOR jeweils im FORMAT F 10.2.

zugehörige Datenkarte



Innerhalb der Spalten 1 bis 10 steht der Wert von ADAM
Innerhalb der Spalten 11 bis 20 steht der Wert von EVA
Innerhalb der Spalten 21 bis 30 steht der Wert von ASTOR.

ADAM erhält den Wert - 30.12, EVA den Wert 1234.56
und ASTOR den Wert 4567.90

Ausgabe:

In den Bereich, der die Länge von w Druckstellen besitzt, wird der Wert der REAL-Variablen in Form einer Dezimalpunktkonstanten rechtsbündig ausgegeben. Vorlaufende Nullen werden dabei unterdrückt. Nur bei negativen Werten wird das Vorzeichen mit angegeben. Man beachte, daß stets d Stellen nach dem Dezimalpunkt ausgedruckt werden und somit maximal w-d-2 für die Ausgabe der Ziffern vor dem Dezimalpunkt zur Verfügung stehen.

Beispiel 2

Die im Beispiel 1 eingelesenen Werte sollen ausgedruckt werden

```
C
      WRITE (2,25) ADAM,EVA,ASTOR
   25 FORMAT (10X,3F10.2)
C
```

Wirkung: In der nächsten Zeile werden zuerst 10 Leerzeichen und danach die Werte von ADAM, EVA, ASTOR jeweils im Format F 10.2 ausgegeben.

Ergebnisausdruck

```
-30.12   1234.56   4567.90
```

Für die Ein- bzw. Ausgabe wird vorerst folgendes empfohlen:

Für INTEGER-Variable die Spezifikation I 10,
für REAL-Variable die Spezifikation F 10.2,
der Klammerausdruck der FORMAT-Anweisung für die Ausgabe beginnt mit 10 X.

6.6 Die H-Spezifikation (Ausgabe von Text)

Es besteht die Möglichkeit, Klartext im Ergebnisausdruck einzustreuen und dadurch den Ausdruck übersichtlicher zu gestalten. Die H-Spezifikation stellt einen Weg dar, diese Möglichkeit zu verwirklichen.
Der allgemeine Aufbau lautet

n H...

n gibt die Anzahl der Zeichen an, die dem Buchstaben H folgen und ausgegeben werden sollen. Hier sind alle Zeichen des Zentralcodes des benutzten Rechners erlaubt.

Beispiel:

Es sind die Koordinaten des Punktes P berechnet worden und diese sollen mit Text versehen ausgedruckt werden.

```
C
      WRITE (2,100)X,Y
  100 FORMAT (10X,30HKOORDINATEN DES PUNKTES P X = ,
     1F5.2,5H Y = ,F5.2)
C
```

Ergebnisausdruck

```
KOORDINATEN DES PUNKTES P X =  5.40 Y = -4.60
```

7. Steuerungsanweisungen

7.1 Vorbemerkungen

Im allgemeinen werden in einem FORTRAN-Programm aufeinanderfolgende Anweisungen auch nacheinander ausgeführt. Diesen linearen Ablauf eines Programms oder Programmteiles kann man durch die Steuerungsanweisungen durchbrechen. Steuerungsanweisungen sind bedingte und unbedingte Sprunganweisungen. Dabei darf der Sprung nur zu einer ausführbaren Anweisung erfolgen.

7.2 Unbedingtes GOTO

Das unbedingte GOTO ist eine ausführbare Anweisung.
Unter einem unbedingten GOTO versteht man eine Sprunganweisung, die keine Bedingung bezüglich des Sprungzieles an irgendwelche Größen innerhalb des Programms stellt. Die Anweisung lautet:

GOTO n

Dabei gibt n die Anweisungsnummer (Label, Statementnummer) des Sprungzieles an. Es wird zu dieser Anweisungsnummer gesprungen und die Verarbeitung mit der dort stehenden Anweisung fortgesetzt.

Beispiel:

```
      .
      .
      .
    GOTO 47
13  Y = X + R * 10.0
      .
      .
      .
47  Z = X + V ** 3
      .
      .
      .
```

Die Anweisung GOTO 47 bewirkt, daß ein Sprung zu der Anweisungsnummer 47 durchgeführt und als nächstes die dort stehende Anweisung ausgeführt wird.
Die Anweisung mit dem Label 13 wird nur dann ausgeführt, wenn sie durch eine andere Steueranweisung angesprungen wird.

Im vorigen Beispiel wurde mit Hilfe der GOTO-Anweisung ein bestimmter Programmteil in Verarbeitungsrichtung übersprungen. Im folgenden Beispiel wird aufgezeigt, daß auch ein Rücksprung im Programm mittels der GOTO-Anweisung möglich ist:

Beispiel:

```
          .
          .
          .
504 IZEHN = KODE / 10
    IEINS = KODE - (KODE / 10) * 10
    RZEIT = RZEIT + TIME
    GOTO 100
505 MZEHN = 6
    MEINS = 5
    GOTO 504
100       .
          .
          .
```

7.3 Die arithmetische IF-Anweisung

Die arithmetische IF-Anweisung ist eine ausführbare Anweisung. Sie stellt die erste Möglichkeit einer bedingten Verzweigung dar. Ihre allgemeine Form lautet:

IF (a)n1, n2, n3

Hierbei symbolisiert a einen beliebigen arithmetischen Ausdruck des Typs INTEGER, REAL oder DOUBLEPRECISION, nicht jedoch vom Typ COMPLEX. n1, n2 und n3 sind die Anweisungsnummern der drei möglichen Sprungziele. Die Anweisung wird so ausgeführt, daß zunächst der arithmetische Ausdruck a berechnet und dann in Abhängigkeit vom Ergebnis das Sprungziel ausgewählt wird, wobei gilt:

a < 0	Sprung zur Anweisungsnummer	n1
a = 0	Sprung " "	n2
a > 0	Sprung " "	n3

Beispiel:

```
          .
          .
          .
    IF(X - 2.)10,20,30
20  Y = A + B ** 2
          .
          .
          .
    GOTO 50
30  Z = B ** 2
          .
          .
          .
    GOTO 50
10  ETA = (X * Y) / D + 3.141592
50.       .
          .
          .
```

Für X < 2. erfolgt ein Sprung zur Anweisungsnummer 10.
" X = 2. " " " " " " 20.
" X > 2. " " " " " " 30.

8. Typdeklarationen

Die Typdeklarationen informieren den Compiler über die Art der im Quellenprogramm verwendeten Variablen und Variablengruppen (Felder, Arrays). Zusätzlich können sie Informationen über die Anzahl der zu reservierenden Speicherplätze enthalten.

In Kapitel 3.3.5 hatten wir festgelegt, daß "Variable, deren Namen mit einem I, J, K, L, M oder N beginnen, als INTEGER-Größen aufgefaßt werden. Alle anderen Variablen gelten als vom Typ REAL". In diesem Falle wird der Typ der Variablen durch den ersten Buchstaben des gewählten Namens festgelegt. Diese Art der Typdeklaration bezeichnet man als implizite Typdeklaration.

Bei der expliziten Typdeklaration wird der Typ der Variablen oder Variablengruppe durch eine besondere Anweisung festgelegt.
Diese Anweisungen sind:

INTEGER liste	(für ganzzahlige Variable)
REAL liste	(für Gleitkomma-Variable)
DOUBLE PRECISION liste	(für doppeltgenaue Variable)
COMPLEX liste	(für komplexe Variable)
LOGICAL liste	(für logische Variable)

Unter liste werden die Namen der einzelnen Variblen, durch Komma voneinander getrennt, einfach aufgezählt. Diese Typdeklarationen sind nicht ausführbare Anweisungen und sie müssen vor der ersten ausführbaren Anweisung auftreten. In liste dürfen nicht nur Namen einfacher Variabler sondern auch die Namen indizierter Variabler aufgeführt werden. Dadurch wird festgelegt, daß alle Elemente des Feldes mit diesem Namen vom entsprechenden Typ sind. Dem Namen kann eine Angabe über die Dimension folgen. Werden keine Angaben zur Dimensionierung in der expliziten Typendeklaration gemacht, so müssen diese Spezifikationen in der DIMENSION-Anweisung (siehe 10.2) angegeben werden. In FORTRAN besteht die indizierte Variable aus einem Namen, dem eine in runde Klammer eingeschlossene Indexliste folgt. Die Indizes werden durch Kommata voneinander getrennt.

Beispiele:

mathematische Schreibweise	Schreibweise in FORTRAN
$a_{i,j}$	A (I, J)
b_1	B (1)
$y_{5,7,i}$	Y (5, 7, I)
x_o	in FORTRAN aufgrund des Indexes nicht darstellbar

In der expliziten Typendeklaration ist die Schreibweise für die Feldspezifikation mit Dimensionsangabe formal identisch mit der Schreibweise einer indizierten Variablen. Bei der Feldspezifikation wird der Maximalwert jedes Index, der stets eine positive INTEGER-Konstante (größer Null) sein muß, für das Feld festgelegt.

Beispiele:

```
REAL IDA,JOT(20,5,2),LOGIC,MPH
INTEGER GAR(60),BETA,ALPHA,DATUM
DOUBLE PRECISION JUNO,E605,PLATTE(20)
COMPLEX A412,DATA,EXPORT,IMPORT
LOGICAL DISC,BOOLE,MODUS,SWITCH(10)
```

Man beachte, daß alle Variablen vom Typ DOUBLE PRECISION, COMPLEX und LOGICAL explizit deklariert werden müssen. Überflüssig wären folgende Deklarationen:

```
INTEGER LOT,MIX,MAX
REAL ETA,GAMMA,EPSI
```

da in allen Fällen der Typ bereits implizit durch den ersten Buchstaben des Namens festgelegt ist. Bei der Arbeit mit indizierten Variablen muß darauf geachtet werden, daß der Maximalwert des Index nicht überschritten wird und der Index selber nicht kleiner als 1 ist. Der Name des Feldes darf nicht mit dem Namen einer einfachen Variable übereinstimmen. Bis auf wenige Ausnahmen darf der Name eines Feldes innerhalb des betreffenden Programms nur indiziert verwendet werden.

Durch die Angabe des Arraynamens ohne Klammern und Indizes wird keinesfalls das erste Arrayelement definiert. Solange einem Arrayelement kein Wert zugewiesen worden ist, gilt der Inhaltder Speicherstelle als undefiniert.

9. Standardfunktionen

FORTRAN stellt dem Programmierer eine Reihe von allgemein gebrauchten Funktionen zur Verfügung, die fast durchweg wie Konstante oder Variable entsprechenden Typs benutzt werden können. Als Argumente sind beliebige Ausdrücke des jeweils erlaubten Typs zulässig. Der Name von Funktionen, die komplexe oder doppeltgenaue Werte liefern, brauchen nicht in einer Spezifikationsanweisung als COMPLEX oder DOUBLE PRECISION erklärt zu werden.

Es folgt eine Liste der verfügbaren Funktionen. Darin sind die Typen der Argumente und der Funktionswerte in folgender Weise abgekürzt: I bedeutet: INTEGER, R: REAL, C: COMPLEX und D: DOUBLE PRECISION.

Definition	Name der Funktion	Anzahl der Argumente	Typ der Argumente	Typ des Funktionswertes
exp(x)	EXP	1	R	R
	DEXP	1	D	D
	CEXP	1	C	C
ln(x)	ALØG	1	R	R
natürlicher Logarithmus	DLØG	1	D	D
	CLØG	1	C	C
lg(x)	ALØG10	1	R	R
Logarithmus zur Basis 10	DLØG10	1	D	D
arctan(x)	ATAN	1	R	R
	DATAN	1	D	D
arctan(x1/x2)	ATAN2	2	R	R
	DATAN2	2	D	D
sin(x)	SIN	1	R	R
(Argument im Bogenmaß)	DSIN	1	D	D
	CSIN	1	C	C
cos(x)	CØS	1	R	R
(Argument im Bogenmaß)	DCØS	1	D	D
	CCØS	1	C	C
$\sqrt{x}$	SQRT	1	R	R
	DSQRT	1	D	D
	CSQRT	1	C	C
tanh(x)	TANH	1	R	R
Divisionsrest	MØD	2	I	I
$x1 - [x1/x2] * x2$	AMØD	2	R	R
$[a]$ ist dabei die größte ganze Zahl $\leq a$	DMØD	2	D	D

Definition	Name der Funktion	Anzahl der Argumente	Typ der Argumente	Typ des Funktionswertes
|x|	IABS	1	I	I
Betrag von x	ABS	1	R	R
	DABS	1	D	D
	CABS	1	C	C
Runden auf ganzzahligen	INT	1	R	I
Teil:	AINT	1	R	R
Vorzeichen von x mal	IDINT	1	D	I
größte ganze Zahl ≤ | x|				
Max(x1,x2,...)	AMAX0	≥ 2	I	R
Größter Wert	AMAX1	≥ 2	R	R
	MAX0	≥ 2	I	I
	MAX1	≥ 2	R	I
	DMAX1	≥ 2	D	D
Min(x1,x2,...)	AMIN0	≥ 2	I	R
kleinster Wert	AMIN1	≥ 2	R	R
	MIN0	≥ 2	I	I
	MIN1	≥ 2	R	I
	DMIN1	≥ 2	D	D
Umwandlung INTEGER→REAL	FLØAT	1	I	R
Umwandlung REAL→INTEGER	IFIX	1	R	I
Vorzeichen von x2, damit versehen	SIGN	2	R	R
Betrag von x1	ISIGN	2	I	I
	DSIGN	2	D	D
x1 - MIN(x1,x2)	DIM	2	R	R
	IDIM	2	I	I
Verkürzen auf REAL-Form	SNGL	1	D	R
Verlängern auf DOUBLE PRECISION-Form	DBLE	1	R	D
Realteil von x	REAL	1	C	R
Imaginärteil von x	AIMAG	1	C	R
Bildung der komplexen Zahl x1 + i*x2	CMPLX	2	R	C
Konjugiert komplexe Zahl	CØNJG	1	C	C

Achtung: Bei allen Winkelfunktionen ist das Argument im Bogenmaß anzugeben

10. Benutzung indizierter Variablen

10.1 Indizierte Variablen

10.1.1 Felder (Arrays)

Statt mehrere einfache Variablen des gleichen Typs kann man in FORTRAN auch indizierte Variable benutzen. Eine geordnete Menge solcher Variablen bildet ein Feld (Array), das durch einen allen Feldvariablen gemeinsamen Namen gekennzeichnet ist, durch den auch der Typ der das Feld bildenden Variablen festgelegt ist. Die einzelnen Feldelemente lassen sich aufgrund ihrer Position im Feld unterscheiden, die durch die den betreffenden Elementen jeweils zugehörige Indexliste angegeben wird.

Die einzelne indizierte Variable wird in der Weise geschrieben, daß an den jeweiligen Feldnamen die in Klammern eingeschlossene Indexliste angehängt wird.

Ein Feld kann entweder 1, 2 oder 3 Dimensionen haben, je nachdem wieviel Indizes die Indexliste der Feldelemente umfaßt. Die Dimension eines Feldes und maximale Größe der Indizes seiner Elemente muß in einer entsprechenden Spezifikationsanweisung angegeben werden, die vor der ersten ausführbaren Anweisung der jeweiligen Programmeinheit steht, in der auf eine Variable des betreffenden Feldes Bezug genommen wird.

10.1.2 Indizes

Indizes müssen eine der folgenden Formen haben, die sich alle aus dem Ausdruck $c * v \pm k$ ableiten lassen:

$$c*v+k,\ c*v-k,\ c*v,\ v+k,\ v-k,\ v \text{ oder } k$$

wobei c und k vorzeichenlose INTEGER-Konstanten sind und v eine INTEGER-Variable darstellt.

Der Wert jedes Indexes muß immer größer als Null sein und darf seinen vorgegebenen Maximalwert nie überschreiten.

Eine Indexliste besteht aus einer Folge durch Komma getrennter Indexausdrücke, deren Anzahl mit der Dimension des entspechenden Feldes übereinstimmen muß.

Beispiele:
Sei VEKTOR ein eindimensionales, MATRIX ein zweidimensionales und RAUM ein dreidimensionales Feld.

a) Folgende Variable sind dann richtig indiziert:

```
VEKTOR(2)
MATRIX(3*I,J-1)
RAUM(I+121,2*J-3,J)
VEKTOR(5*K+1)
MATRIX(4,8*I-7)
RAUM(3,3,3)
```

b) Folgende Kombinationen sind falsch:

VEKTOR (2, 3)	(Anzahl der Indizes stimmt nicht mit der Anzahl der Dimensionen überein)
VEKTOR (-2)	(Negativer Indes nicht erlaubt)
MATRIX (0, 1. 34)	Null oder reelle Zahl als Index nicht möglich)
MATRIX(VEKTOR(2), 1)	(nur einfache Variable in Indexausdrücken erlaubt)
RAUM(5-I, J+K, J*3-2)	(Indexausdrücke entsprechen nicht der erlaubten Form c * v + k)

10. 1. 3 Anordnung der Feldelemente im Speicher

Die Variablen eines Feldes werden in aufeinanderfolgenden Speicherplätzen entsprechend der Werte ihrer Indizes in der Weise abgespeichert, daß vordere Indizes schneller laufen als die nachfolgenden. Die folgenden Beispiele sollen diesen Sachverhalt verdeutlichen:

Sei A ein eindimensionales Feld der Länge 7, so wird es in folgender Weise abgespeichert:

A(1) A(2) A(3) A(4) A(5) A(6) A(7)

Das zweidimensionale Feld M habe als Maximalgrößen für seine beiden Indizes 3 und 5. Dann steht es in folgender Reihenfolge im Speicher:

M(1, 1) M(2, 1) M(3, 1) M(1, 2) M(2, 2) M(3, 2) M(1, 3)
M(2, 3) M(3, 3) M(1, 4) M(2, 4) M(3, 4) M(1, 5) M(2, 5)
M(3, 5) (zeilenweise gelesen)

Betrachtet man M als Matrix, in der der erste Index wie üblich die Zeilen- und der zweite die Spaltennummer angibt, so wird M spaltenweise abgespeichert.

Sei R ein dreidimensionales Feld mit den Maximalindizes 4, 2 und 3, so wird es in folgender Weise gespeichert:

```
R(1, 1, 1)  R(2, 1, 1) R(3, 1, 1)  R(4, 1, 1)  R(1, 2, 1)
R(2, 2, 1)  R(3, 2, 1) R(4, 2, 1)  R(1, 1, 2)  R(2, 1, 2)
R(3, 1, 2)  R(4, 1, 2) R(1, 2, 2)  R(2, 2, 2)  R(3, 2, 2)
R(4, 2, 2)  R(1, 1, 3) R(2, 1, 3)  R(3, 1, 3)  R(4, 1, 3)
R(1, 2, 3)  R(2, 2, 3) R(3, 2, 3)  R(4, 2, 3)
                                   (zeilenweise gelesen)
```

10.2 Die DIMENSION - Anweisung

Die DIMENSION-Anweisung ist eine nichtausführbare Anweisung, die die Anzahl und die Größe der Felddimensionen definiert und damit dem Compiler die für die Speicherzuweisung notwendigen Informationen gibt. Sie hat die allgemeine Form:

```
DIMENSION a1(k1), a2(k2), . . . . . aj(kj)
```

Dabei stellt jedes a1, a2, ... , aj den Namen eines Feldes dar. In der DIMENSION-Anweisung dürfen die zu spezifizierenden Felder vom unterschiedlichsten Typ sein. Die k1, k2. . . . . kj sind Indexlisten, bestehend aus 1 bis 3 positiven INTEGER-Konstanten (größer als Null), welche durch Komma voneinander getrennt sind und die den Maximalwert jedes Index des Feldes definieren.

Jede im Quellenprogramm verwendete indizierte Variable muß vor ihrer erstmaligen Verwendung entweder in einer expliziten Typdeklaration oder in einer DIMENSION-Anweisung dimensioniert worden sein. Deshalb muß auch die DIMENSION-Anweisung stets am Anfang eines Programms vor der ersten ausführbaren Anweisung stehen. Die Dimensionierung eines Feldes in mehr als einer DIMENSION-Anweisung ist nicht gestattet. Eine Feldangabe kann außerdem auch über die Benutzung von COMMON (siehe 15) erfolgen.

Beispiel:

```
DIMENSION  AFELD(10), BFELD(20, 3)
```

Durch diese Anweisung werden für AFELD 10 Speicherplätze und für BFELD 20*3 = 60 Speicherplätze reserviert.

Für den Fall, daß eine Vielzahl von indizierten Variablen zu Beginn eines Programmes dimensioniert werden müssen, können bei der DIMENSION-Anweisung Folgekarten verwendet werden oder mehrere DIMENSION-Anweisungen hintereinander folgen. Die beiden folgenden Darstellungen sind daher völlig gleichwertig:

```
      DIMENSION ARRAY(10, 20), MATRIX(10, 10, 10), VEKTOR(100),
     *  NEXT(10, 15, 30), LISTE(136), IPUNCH(80)
```

oder

```
DIMENSION ARRAY(10, 20), MATRIX(10, 10, 10), VEKTOR(100)
DIMENSION NEXT(10, 15, 30), LISTE(136), IPUNCH(80)
```

Es ist streng darauf zu achten, daß die in der DIMENSION-Anweisung definierten Grenzen während des Programmablaufs nicht überschritten werden. Der in der DIMENSION-Anweisung deklarierte Name eines Feldes darf bis auf wenige Ausnahmen in dem betreffenden Programm nur indiziert verwendet werden.

Bei der Verwendung von indizierten Variablen muß man sich klar darüber sein, daß die Zahlen in der DIMENSION-Anweisung eine andere Bedeutung haben als die Indizes der ausführbaren Anweisungen. Im ersten Falle geben sie nur die zulässigen Maximalwerte der Indizes an, während sie im zweiten Falle einen bestimmten Speicherplatz innerhalb des Feldes ansprechen.

Durch die Angabe des Arraynamens ohne Klammern und Indizes wird keinesfalls das erste Arrayelement definiert. Solange einem Arrayelement kein Wert zugewiesen worden ist, gilt der Inhalt der Speicherstelle als undefiniert.

Da die Dimension eines Feldes entweder in der expliziten Typdeklaration oder in der DIMENSION-Anweisung aber nicht gleichzeitig in beiden anzugeben ist, muß man sich für eine der beiden Darstellungen entscheiden, z.B. entweder

```
DIMENSION JOT(20, 5, 2)
REAL IDA, JOT, LOGIC, MPH
```

oder

```
REAL IDA, JOT(20, 5, 2), LOGIC, MPH
```

Dabei ist es gleichgültig, ob die DIMENSION-Anweisung der REAL-Spezifikation voransteht oder ihr folgt.

10.3 Die DO-Anweisung

Oftmals will man einen Rechenvorgang mehrfach wiederholen. Erreichen kann man das mit Hilfe einer Kontrollvariablen (einem Zähler), die bei jeder Wiederholung erhöht und darauf abgefragt wird, ob die Anzahl der gewünschten Wiederholungen schon erreicht ist. Eine derartige Konstruktion wird als Schleife bezeichnet. Dazu ein kleines Beispiel:
Es sollen die Potenzen von IEXPO = 1..... 10 zur Basis IBAS = 2 gebildet und ausgedruckt werden. Ausgedruckt werden soll das Ergebnis der Exponentiation (IWERT) und der Exponent (IEXPO) selbst.

```
      IBAS          = 2
      IEXPO         = 1
1     IWERT         = IBAS**IEXPO
      WRITE(2, 2)IWERT, IEXPO
2     FORMAT (10X, I10, I10)
      IEXPO         =IEXPO  +1
      IF  (IEXPO    -10) 1, 1, 3
3     STOP
      END
```

Dieser Vorgang läßt sich mit Hilfe der DO-Anweisung einfacher programmieren. Die DO-Anweisung hat die allgemeine Form:

```
DO  n  i=m1, m2, m3
```

Die einzelnen Parameter haben folgende Bedeutung:

n Die Anweisungsnummer der letzten zur DO-Schleife gehörenden Anweisung.

i Die Laufvariable der DO-Anweisung. Sie ist eine einfache INTEGER-Variable, die nicht indiziert und nicht mit einem Vorzeichen versehen sein darf.

m1 Der Anfangsparameter der DO-Anweisung. Er ist eine vorzeichenlose INTEGER-Konstante oder INTEGER-Variable, deren Wert positiv sein muß.

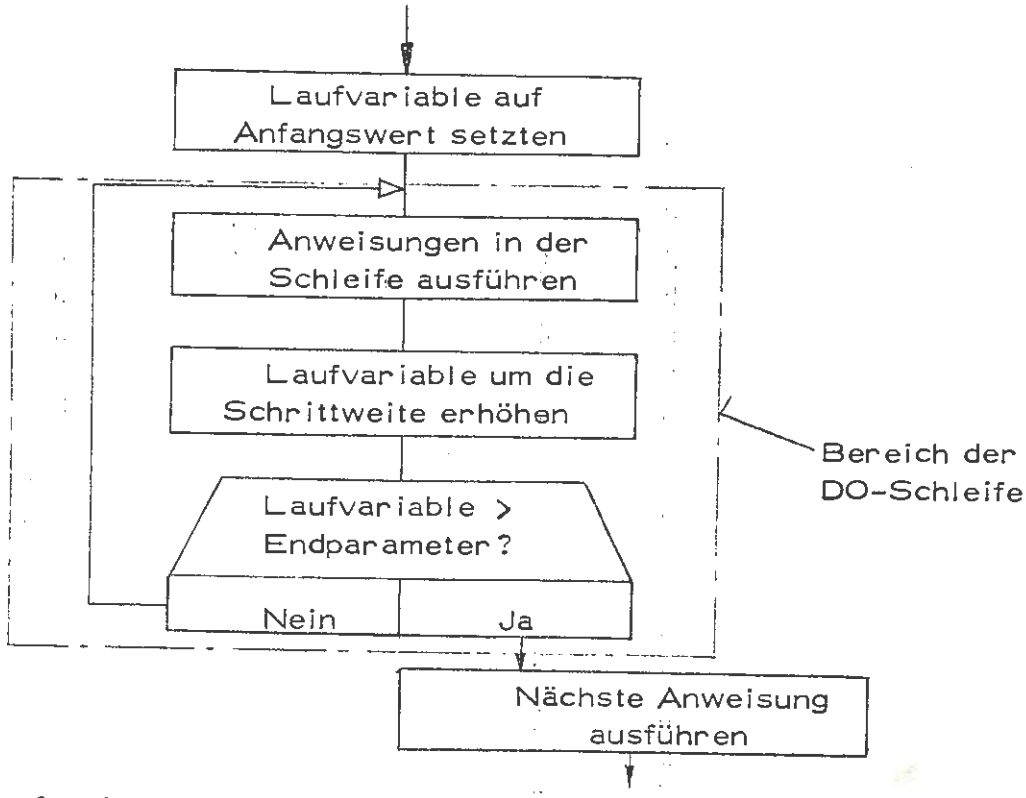
m2 Der Endparameter der DO-Anweisung. Er ist eine vorzeichenlose INTEGER-Konstante oder INTEGER Variable, deren Wert positiv sein muß.

m3 Der Schrittweitenparameter (oder Inkrement) der DO-Anweisung. Er ist eine vorzeichenlose INTEGER-Konstante oder INTEGER-Variable, deren Wert positiv sein muß. Bei jedem Durchlauf durch die Schleife wird i um m3 erhöht. Soll die Schrittweite gleich 1 sein, so kann die Angabe von m3 entfallen, dann entfällt auch das Komma hinter m2.

Die DO-Anweisung gehört zu den ausführbaren Anweisungen. Sie bildet mit den auf sie folgenden Anweisungen bis einschließlich derjenigen mit der Anweisungs-

nummer n eine DO-Schleife. Diese wird entsprechend den Laufparametern m1, m2 und m3 zyklisch wiederholt. Bei der ersten Ausführung der DO-Anweisung wird der Laufvariablen i der Wert m1 zugewiesen. Vor jeder weiteren Ausführung wird i um den Wert von m3 erhöht. Die Schleife wird zum letzten Mal durchlaufen, wenn i den höchsten seiner Werte angenommen hat, der m2 noch nicht übersteigt. Anschließend wird das Programm mit der auf die Anweisung n folgenden Anweisung fortgesetzt.

Der folgende Flußdiagramm-Ausschnitt gibt die Struktur einer DO-Schleife wieder:



Da die Laufvariable i erst am Ende der DO-Schleife auf ihren Grenzwert geprüft wird, wird die Schleife mindestens einmal durchlaufen. Wird eine DO-Schleife vollständig abgearbeitet, d.h. die Laufvariable i durchläuft vollständig ihren Wertebereich, dann ist nach Verlassen der Schleife der Wert von i nicht mehr definiert.

Bei den folgenden Beispielen ist jeweils links die DO-Schleife und rechts ihre Wirkung in Einzelanweisungen angegeben.

1. Beispiel

```
      DO 7  I=1, 3
    7 A(I)    = B(I)
```

```
A(1)  = B(1)
A(2)  = B(2)
A(3)  = B(3)
```

Bei der Abarbeitung dieser Anweisung wird die Schleife zunächst mit I = 1 durchlaufen. Dann wird I um die Schrittweite 1 erhöht, es ist also I = 2.

Dieser Wert ist nicht größer als der Endparameter, die Schleife wird nochmals durchlaufen. I erhält nun den Wert 3. Auch dieser Wert ist nicht größer als der Endparameter, die Schleife wird abermals durch laufen. Jetzt erhält I den Wert 4. Dieser Wert ist größer als der zulässige Endparameter: Die Schleife wird verlassen. Von jetzt an ist I nicht mehr definiert. Es wurde jedes Element des Feldes B umgespeichert in das Feld A.

2. Beispiel:

```
   DO 8  I = 1, 9, 2
8  C(I)     = D(I)
```

```
C(1)  = D(1)
C(3)  = D(3)
C(5)  = D(5)
C(7)  = D(7)
C(9)  = D(9)
```

Die DO-Schleife wird insgesamt fünfmal mit der Schrittweite 2 durchlaufen und zwar für I = 1, 3, 5, 7, 9. Nach dem Durchlauf für I = 9 wird I um 2 erhöht, es gilt also I = 11. Dieser Wert ist größer als der Endparameter: Die Schleife wird verlassen. Es sind jeweils die Feldelemente mit ungeradzahligem Index von Feld D nach Feld C gespeichert worden. I ist nun undefiniert.

3. Beispiel:

```
   SUMME  = 0.0
   DO 9   I = 1, 10
9  SUMME  =  SUMME + X(I)
```

```
SUMME  = SUMME+X(1)
SUMME  = SUMME+X(2)
   .
   .
   .
SUMME  = SUMME+X(10)
```

Die DO-Schleife summiert die ersten zehn Elemente des Feldes X.

4. Beispiel:

```
   DO 5  J = 4, 20, 5
5  ARRAY(J) = BARRAY (J)
```

```
ARRAY (4)  = BARRAY (4)
ARRAY (9)  = BARRAY (9)
ARRAY (14) = BARRAY (14)
ARRAY (19) = BARRAY (19)
```

Man sieht also: Die obere Grenze muß nicht unbedingt genau erreicht werden. Hätte man statt der 20 eine 19, 21, 22 oder 23 geschrieben, so wäre der gleiche Effekt erzielt worden.

5. Beispiel:

```
      A = 0.0
      DØ 4 I =1,6
      X(I) = SIN (A)
      Y(I) = COS (A)
      Z(I) = EXP(A)
4     A    = A + 0.2
```

```
X(1) = SIN (0.0)
Y(1) = COS(0.0)
Z(1) = EXP(0.0)
A    = A + 0.2
X(2) = SIN (0.2)
       .
       .
       .
X(6) = SIN (1.0)
Y(6) = COS(1.0)
Z(6) = EXP(1.0)
```

Es werden die Funktionswerte für A = 0.0, 0,2, 0.4, 0.6, 0.8 und 1.0 berechnet und in die Felder X, Y und Z gespeichert.
Nach dem Verlassen der Schleife hat A den Wert 1.2, I ist nicht mehr definiert.

6. Beispiel:

```
      N = 4
      X = 2.0
      Y = 1.0
      DO 3 L =1,N
3     Y = Y * X
```

```
Y = 1. *2.
Y = 2. *2.
Y = 4. *2.
Y = 8. *2.
```

Die Schleife berechnet den Wert der Funktion $y = x^n$.

7. Beispiel:

Das gleiche Programm, wie zu Beginn dieses Kapitels, diesmal jedoch mit einer DO-Anweisung programmiert.

```
      IBAS = 2
      DO 1 IEXPO = 1,10
      IWERT = IBAS ** IEPO
1     WRITE(2,2) IWERT, IEXPO
2     FORMAT (10X,I10, I10)
      STOP
      END
```

Innerhalb der DO-Schleife wird die Potenz zur Basis IBAS =2 berechnet und die jeweiligen Werte des Ergebnisses IWERT und des Exponenten IEXPO gedruckt. Der Exponent wird bei jedem Durchgang durch die DO-Schleife um 1 erhöht. Für IEXPO = 10 wird die letzte Zeile gedruckt und die Schleife verlassen.

10.4 Regeln für die Programmierung von DO - Schleifen

1.) Der Laufvariablen i und den INTEGER-Größen m1, m2 und m3 dürfen innerhalb der DO-Schleife keine neuen Werte zugewiesen werden. Die folgende DO-Schleife ist unzulässig:

```
      :
      :
     DØ 3 K=1,10
     IWERT = 2**K
     K = K + 1
   3 IDIFF = 2**K
      :
      :
```

Die Schleifenparameter dürfen jedoch innerhalb der Schleife für Berechnungen und Verzweigungen benutzt werden.

2.) Die letzte Anweisung einer DO-Schleife muß ausführbar sein und darf keine der folgenden Anweisungen sein:

GOTO-Anweisung in jeder Form
DO-Anweisung
Arithmetisches IF
RETURN-Anweisung (siehe Kap. 14.).
PAUSE-Anweisung (siehe Kap. 23.).
STOP-Anweisung

Das logische IF ist am Ende einer Schleife zulässig, wenn es keine der vorstehend genannten Anweisungen enthält.

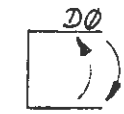
3.) Wird eine DO-Schleife durch einen Sprungbefehl vor der vollständigen Abarbeitung verlassen, so kann die Laufvariable i im weiteren Programmablauf verwendet werden. Sie hat dann den Wert, der ihr vor Verlassen der Schleife zugewiesen wurde. Dies zeigt das folgende Beispiel:

```
      .
      .
      .
5  DO 6 INDEX=1,19
   IF(ISTYP - MSTYP(INDEX))6,7,6
6  MERKE = INDEX
   GOTO 11
7  JUMP = 2 * INDEX
      .
      .
      .
```

4.) Bei der Programmierung von DO-Schleifen sind weitere Regeln zu beachten, die Aussagen über erlaubte und verbotene Sprünge betreffen:

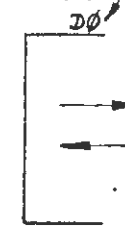
1.) Erlaubte Sprünge

a.) Innerhalb des Wiederholungsbereiches vorwärts und rückwärts



b.) Aus der Schleife heraus in und entgegen Programmablaufrichtung.

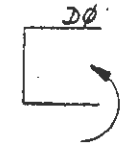
c.) Sprung aus der Schleife heraus, um z.B. spezielle Rechnungen durchführen zu können und Rücksprung in die gleiche Schleife. Dabei dürfen die Werte der Laufvariablen i und der Parameter m1, m2 und m3 nicht verändert werden.



2.) Verbotene Sprünge:

a.) In Programmablaufrichtung auf eine Anweisung innerhalb des Wiederholungsbereichs der DO-Schleife.

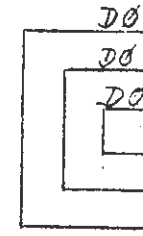
b.) Entgegen Programmablaufrichtung auf eine Anweisung innerhalb des Wiederholungsbereichs der DO-Schleife.



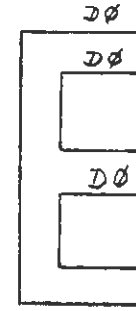
5.) Der Wiederholungsbereich einer DO-Schleife darf weitere DO-Schleifen beinhalten, so daß eine Schachtelung von DO-Schleifen entsteht (sog. DO-Nest). Die Abarbeitung derartig geschachtelter DO-Schleifen erfolgt stets von innen nach außen.

1.) Erlaubte Schachtelungen:

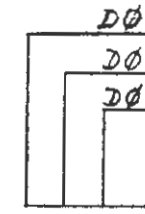
a.) Schachtelung aller DO-Schleifen ineinander.



b.) Schachtelung von nacheinander auftretenden DO-Schleifen in eine umfassende Schleife.

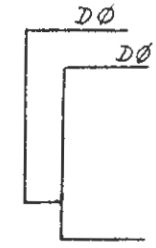


c.) Schachtelung der DO-Schleifen ineinander, jedoch mit gemeinsamer Schlußanweisung.
Achtung: Eine Anweisung, die Endanweisung für mehrere DO-Schleifen zugleich ist, zählt als zur innersten DO-Schleife gehörig!



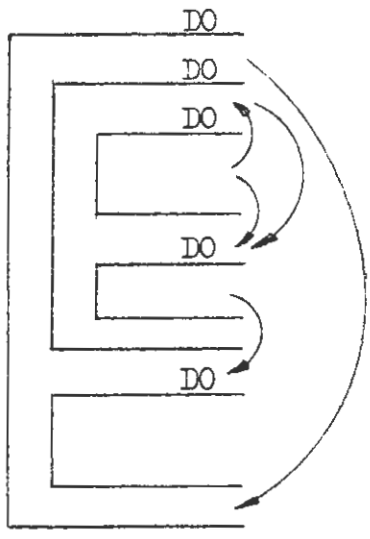
2.) Verbotene Schachtelung:

a.) Überschneidungen geschachtelter DO-Schleifen.

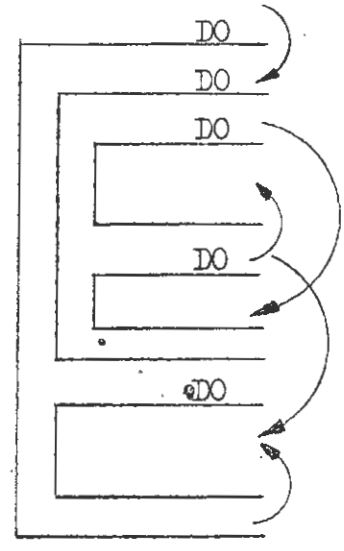


Bei Beachtung der Regeln für Programmsprünge und strikter Anwendung auf geschachtelte DO-Schleifen ergeben sich folgende Möglichkeiten:

a.) Erlaubte Sprünge:



b.) Verbotene Sprünge:



Die erlaubte Tiefe der Schachtelung ist vom jeweiligen FORTRAN-Compiler abhängig.

10.5 Die CONTINUE - Anweisung

Bei der Programmierung von DO-Schleifen - aber nicht nur hier - kann es notwendig werden, eine FORTRAN-Anweisung zu verwenden, die keinen Maschinenbefehl erzeugt. Eine solche Anweisung ist

CONTINUE

Die CONTINUE-Anweisung ist eine leere Anweisung. Sie kann an jeder beliebigen Stelle eines Programmes stehen und sie zählt zu den ausführbaren Anweisungen. Sie wird zumeist als letzte Anweisung einer DO-Schleife verwendet, wenn der logische Ablauf der Schleife mit einer unzulässigen Anweisung enden würde (z.B. Sprunganweisung oder arithmetisches IF).

1.) Beispiel:

```
    DØ 10 L=1,18
  3 IF(X(L) - Y(L))4,10,10
  4 X(L) = X(L) + 4.0
    Y(L) = Y(L) - 3.0
    GØTØ 3
 10 CØNTINUE
 12 ZET = X(2) + Y(6)
          :
          :
```

Die CONTINUE-Anweisung ist an dieser Stelle erforderlich, da der unbedingte Sprung GOTO 3 nicht als letzte Anweisung einer DO-Schleife erlaubt ist.

2.) Beispiel:

```
    DØ 9 L=1,12
    IF(X(L) - Y(L))4,15,15
  4 X(L) = Z(L)
    GØTØ 9
 15 X(L) = 0.0
  9 CØNTINUE
          :
          :
```

Die CONTINUE-Anweisung ist erforderlich, um aus beiden Verzweigungen das Ende der DO-Schleife erreichen zu können.

11. Erweiterung der Ein- und Ausgabeanweisungen

11.1 Die E-Spezifikation

Mittels der E-Spezifikation kann man ebenfalls wie mittels der F-Spezifikation REAL-Variable übertragen.
Der allgemeine Aufbau lautet

r E w. d

r stellt den Wiederholungsfaktor dar (siehe auch I - Spezifikation)
w gibt die Anzahl der Stellen des Bereichs an (siehe auch I - Spezifikation)
d ist gleich der Anzahl der Stellen nach dem Dezimalpunkt (siehe auch F - Spezifikation).

Achtung: Es sollte stets gelten w≥d + 7

Eingabe
Eine reelle Konstante kann bei der Eingabe an beliebiger Stelle innerhalb des Datenbereiches stehen, sofern sie einen Dezimalpunkt enthält. (In diesem Fall ist d ohne jede Bedeutung - es muß für d jedoch aus formalen Gründen ein Wert angegeben werden). Wird die Zahl mit Exponent dargestellt, so kann bei der Eingabe auch statt E der Buchstabe D oder nur eine mit Vorzeichen versehene INTEGER - Konstante, die den Wert des Exponenten angibt, stehen. Der Exponent muß stets rechtsbündig gelocht sein. Wird die reelle Konstante als INTEGER - Größe (also ohne Dezimalpunkt) angegeben, so wird zwischen der d-ten und der (d + 1)-ten Stelle - gerechnet vom rechten Rand des Datenbereichs (Darstellung ohne Exponent) bzw. gerechnet E, D oder Vorzeichen (Darstellung mit Exponent) - ein Dezimalpunkt angenommen.
Die einzulesende Dezimalzahl kann demnach folgendermaßen dargestellt werden, (dabei ist angegeben, ob rechtsbündig abgelocht werden muß oder nicht. Bezüglich Vorzeichen und Zwischenräume siehe auch I - Spezifikation):

Aufbau der Zahl	Beispiel	rechtsbündig abzulochen
Sie besteht aus		
ganzem Teil	123	ja
ganzem Teil, Dezimalpunkt	1234.	nein
ganzem Teil, Dezimalpunkt gebrochenem Teil	- 39.79	nein
Dezimalpunkt, gebrochenem Teil	.35	nein
ganzem Teil, Exponent	139 E 17	ja
ganzem Teil, Dezimalpunkt, gebrochenem Teil. Exponent	-39.56 E -13	ja (zumindest der Exponent)
ganzem Teil , Dezimalpunkt, Exponent	14. E +03	ja (zumindest der Exponent)
Dezimalpunkt, gebrochenem Teil, Exponent	.575 E ± 25	ja (zumindest der Exponent)

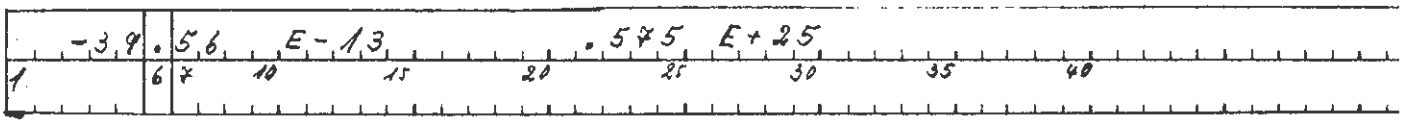
Wenn der Exponent statt durch E durch D oder nur einem Vorzeichen eingeleitet wird, so gilt entsprechendes.

Achtung: Gibt man nur den Exponenten an, so wird diese Zahl als Null interpretiert.

Beispiel 1:

```
      READ (1,200) ASTOR,PLUTO
200   FORMAT (2E15.5)
```

Eingabedaten:



Nach dem Einlesen hat ASTOR den Wert -0.3956E-128 und PLUTO den Wert 0.575E 25. Es hängt von der internen Darstellung der Zahl in der Anlage ab, ob der Wert von Astor mit diesem Exponenten noch intern darstellbar ist.

Eine Ergänzung sei an dieser Stelle gegeben. Liest man unter der F-Spezifikation Daten ein, so gelten für das Ablochen der Daten die gleichen Bedingungen wie bei der Eingabe unter der E-Spezifikation.

Ausgabe

Bei der Ausgabe wird der Wert der Zahl in folgender Form dargestellt

$$V.X_1X_2X_3\ldots\ldots\ldots\ldots X_d\ Z.$$

V	steht für das Vorzeichen - bzw. anstelle von Zwischenraum
$X_1X_2X_3\ldots X_d$	stellen die ersten d signifikanten Ziffern dar.
Z	steht als Abkürzung für den Exponenten. Er hat folgende allgemeine Darstellung $E\pm Y_1Y_2$ oder $\pm Y_1Y_2Y_3$. Dabei steht Y_1 an Stelle einer Dezimalziffer (die Anzahl der Stellen vom Exponenten ist anlagenabhängig).

Beispiel 2:

```
C
      WRITE (2,205) ASTOR,PLUTO
 205  FORMAT (10X,2E15.5)
C
```

Die im Beispiel 1 eingelesenen Werte sollen ausgedruckt werden.

Ergebnisausdruck:

```
-.3956E-128      .57500E+25
```

11.2 Die D - Spezifikation

Die D-Spezifikation wird für REAL-Variable, die vom Typ DOUBLEPRECISION - also doppelter Genauigkeit - sind, zur Übertragung verwandt. Der allgemeine Aufbau ist dem der E-Spezifikation identisch, nur steht statt E ein D, also

r D w. d

Alles, was über Ein- und Ausgabe unter E-Spezifikation gesagt wurde, gilt auch hier. (Außer, daß bei der Ausgabe der Buchstabe E im Exponenten durch D ersetzt wird).

Beispiel:

```
C
      DOUBLE PRECISION ADAM
      ADAM=3.12345678901234567890    D1
      WRITE (2,210) ADAM
 210  FORMAT (10X,D30.21)
C
```

Ergebnisausdruck:

.3123456789012345678900+02

11.3 Die G-Spezifikation

Der allgemeine Aufbau der G-Spezifikation lautet

r G w.d

r — stellt den Wiederholungsfaktor dar
w — gibt die Anzahl der Stellen des Bereichs an.
d — ist gleich der Anzahl der Stellen nach dem Dezimalpunkt (bei der Eingabe) bzw. gleich der Anzahl der signifikanten Stellen (bei der Ausgabe)

Achtung: Es sollte stets gelten $w \geq d + 7$

Eingabe:

Die Bedingungen und Wirkungen bei der Eingabe sind mit denen bei der Eingabe unter E- oder F-Spezifikation identisch.

Ausgabe:

Ist der Betrag des auszugebenden Wertes B kleiner als 1 oder größer oder gleich 10^d, so wird unter Ew.d ausgegeben. Liegt der Betrag von B nicht in diesen Bereichen, so ist für $10^{i-1} \leq |B| < 10^i$ mit $i = 0, 1, \ldots, d$ die Ausgabe wie unter F (w - 4). (d - i), $4H_{\nabla\nabla\nabla\nabla}$.

Beispiel:

Programmteil:

```
C       WERT, DER AUSGEGEBEN WERDEN SOLL, WIRD ZUGEWIESEN.
        EVA=-1234,56
        WRITE (2,120) EVA,EVA,EVA
 120    FORMAT (10X,G15.6,G15.4,G15.2)
C
```

Ergebnisausdruck:

```
      -1234,56          -1235,          -,12E+04
```

11.4 Ein-/Ausgabe von COMPLEX - Variablen

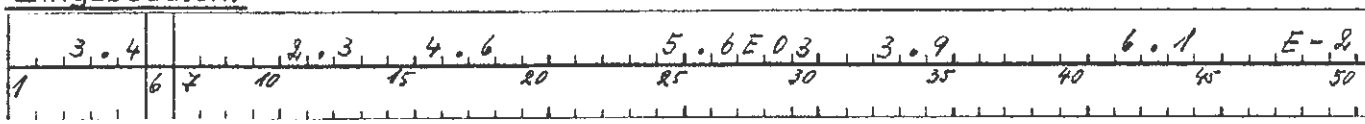
Die komplexe Zahl besteht aus einem Zahlenpaar vom Typ REAL. Deshalb muß man bei der Ein- und Ausgabe komplexer Variabler in der zugehörigen FORMAT-Anweisung die Spezifikation für zwei REAL-Variable pro COMPLEX-Variable angeben. Die erste Angabe wird für den Realteil und die zweite für den Imaginärteil benötigt.

Beispiel:

Programmteil:

```
        COMPLEX A,B,C
        READ (1,10) A,B,C
10      FORMAT (E10.2,2F5.2,3E10.2)
        WRITE (2,20) A,B,C
20      FORMAT (10X,6E10.2)
```

Eingabedaten:



Ergebnisausdruck:

```
      .34E+01   .23E+01   .46E+01   .56E+04   .39E+01   .61E-01
```

11.5 Die A-Spezifikation (Ein-/Ausgabe von Zeichenketten)

Mit Hilfe der A-Spezifikation können beliebige Zeichenketten ein- und ausgegeben werden.
Der allgemeine Aufbau lautet

r A w

r stellt den Wiederholungsfaktor dar
w ist wiederum eine ganzzahlige Konstante ohne Vorzeichen und gibt die Anzahl der Stellen des Bereichs an.

Eingabe

Mittels der A-Spezifikation kann in einer einfachen oder indizierten Variablen ein Zeichen oder eine Zeichenkette abgespeichert werden. Ist I die maximale Anzahl an Zeichen, die pro Kernspeicherwort (Variable) abgespeichert werden können (an der TR 440 ist I=4), so ist wie folgt zu verfahren. Besteht die Zeichenkette aus mehr als I Zeichen, so muß man die Zeichenkette auf mehrere Worte (Variable) aufteilen, daß je I aufeinanderfolgende Zeichen pro Wort abgespeichert werden, w gibt die Anzahl der einzulesenden Zeichen pro Wort an. Ist w kleiner als I, so werden die w Zeichen linksbündig eingelesen und der Rest des Wortes mit Zwischenräumen (blanks) aufgefüllt. Ist w größer als I, so werden nur die letzten I Zeichen im Wort abgespeichert. Die ersten (w-I) Zeichen werden überlesen.

Ausgabe

Die unter A-Spezifikation eingelesenen Zeichen können auch unter A-Spezifikation ausgegeben werden. Ist w kleiner als I, so werden die ersten w Zeichen des Wortes (der angegebenen Variablen) ausgedruckt. Ist w größer als I, so werden (w-I) Zwischenräume und danach die I im Wort abgespeicherten Zeichen ausgegeben.
Beispiel:

Programmteil:

```
000010          INTEGER ADAM,EVA(2)
000020          READ(5,15) ADAM, EVA
000030    15    FORMAT(A4,A9,A4)
000040          WRITE(6,25) ADAM, EVA
000050    25    FORMAT(10X,2A4,A11)
000060          END
```

Eingabedaten:

```
EMIL IST EIN KERL
```

Inhalt der Variablen nach dem Einlesen:

ADAM	EMIL
EVA(1)	EIN␣
EVA(2)	KERL

Ergebnisausdruck:

```
          EMILEIN    KERL
```

Bemerkungen zur H-Spezifikation (Eingabe)

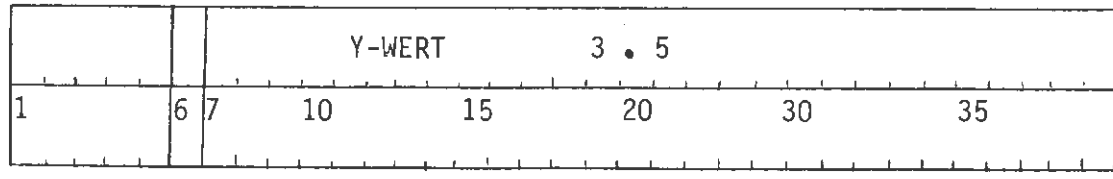
Nicht nur unter der A-Spezifikation sondern auch unter der H-Spezifikation können Zeichenketten eingelesen werden. Ist nH in der FORMAT-Anweisung angegeben, so werden die entsprechenden n Zeichen auf der Lochkarte anstelle der n Zeichen hinter dem H in der FORMAT-Anweisung abgespeichert. Es ist daraufhinzuweisen, daß so übertragene Zeichen nicht in das einer Variablen zugeordnete Wort abgespeichert werden.

Beispiel:

Programmteil:

```
C
      READ (5,35)Y
   35 FORMAT (10X,7HX-WERT␣,F10.2)
      WRITE (6,35) Y
C
```

Eingabedaten:



Ergebnisausdruck:

```
      Y-WERT        3.50
```

11.6 Die X-Spezifikation

Mittels der X-Spezifikation können Spalten übersprungen bzw. Leerstellen ausgegeben werden.

Der allgemeine Aufbau lautet

n X

n Spalten sollen bei der Eingabe übersprungen bzw. n Leerstellen sollen ausgegeben werden.

Beispiel:

Das zur A-Spezifikation angegebene Beispiel soll variiert werden

```
      000010          INTEGER ADAM,EVA(2)
      000020          READ(5,15) ADAM,EVA
      000030          WRITE(6,25) ADAM,EVA
      000040    15    FORMAT(2A4,5X,A4)
      000050    25    FORMAT(10X,2A4,7X,A4)
      000060          END
```

Die zugehörige Datenkarte ist identisch mit der des Beispiels zur A-Spezifikation.

Ergebnisausdruck:

```
      EMIL  IST     KERL
```

11.7 Die L-Spezifikation

Die L-Spezifikation ermöglicht es, den aktuellen Wert von logischen Variablen ein- und auszugeben (Einzelheiten siehe Kapitel 12).
Der allgemeine Aufbau lautet:

r L w

r stellt den Wiederholungsfaktor dar.

w ist eine vorzeichenlose ganzzahlige Konstante und gibt die Anzahl der Stellen des Bereichs an.

Eingabe

Ist in dem aus w Stellen bestehenden Bereich das erste vom Zwischenraum verschiedene Zeichen ein T bzw. ein F, (einer dieser beiden Buchstaben muß innerhalb des Bereiches nach beliebig vielen Zwischenräumen vorkommen) so wird der zugehörigen logischen Variablen der Wert .TRUE. bzw. .FALSE. zugewiesen. Welche Zeichen dem T bzw. F innerhalb des Bereichs folgen, sind unwesentlich.

Ausgabe

Es werden (w-1) Leerstellen und danach der Buchstabe T für .TRUE. und F für .FALSE. ausgegeben.

Beispiel:

```
      LOGICAL LOGI,LOG(2)
      READ (2,50) LOGI,LOG
      WRITE (6,55) LOGI,LOG
50    FORMAT (3L5)
55    FORMAT (10X,2L1,L5)
```

Eingabedaten:

```
 T R U E   F A F A L S
1         6 7  .  10        15        20        25        30        35
```

Ergebnisausdruck:

```
          TF    F
```

11.8 Die Vorschubsteuerzeichen beim Drucken

Will man Infomationen über den Drucker ausgeben, so muß man beachten, daß in jeder Zeile das erste Zeichen nicht gedruckt wird, sondern als Vorschubsteuerzeichen zur Steuerung des Zeilendruckers verwandt wird (jeder Druckbefehl fängt in der ersten Spalte der Zeile an). Die Druckzeile wird auch als FORTRAN-Satz bezeichnet.

Es gibt folgende Zuordnung:

1. Zeichen des FORTRAN-Satzes	Zeilenvorschub vor dem Drucken
Zwischenraum, blank	eine Zeile, es wird auf die nächste Zeile gedruckt
0	zwei Zeilen, es entsteht eine Zeile Abstand
1	1. Zeile der nächsten Seite
+	kein Vorschub, von vorn beginnend wird auf der gleichen Zeile gedruckt

Beispiel:
Programmteil:

```
          I=0
      190 I=I+1
          WRITE (6,50)I
          WRITE (6,55)
      50  FORMAT (11H1    SEITE ,I2)
      55  FORMAT (1HO,6X,18HX     Y     F(X,Y))
```

Ergebnisausdruck:

```
     SEITE  1
       X     Y     F(X,Y)
```

11.9 Ergänzungen zur FORMAT-Anweisung

Bis jetzt gibt es allein die Möglichkeit, aufgrund der Angaben in der FORMAT-Anweisung mit einem READ bzw. WRITE nur von einer Karte zu lesen bzw. nur eine Zeile auszudrucken. In diesem Abschnitt wird eine Erweiterung angegeben.

Unter einem FORTRAN-Satz versteht man die Menge von Daten, die auf eine Lochkarte (80 Zeichen) bzw. Druckerzeile (z. B. TR 440, CD 136 Zeichen) paßt. In der FORMAT-Anweisung ist diese Beschränkung zu berücksichtigen. Die öffnende Klammer bei der FORMAT-Anweisung besagt, ein FORTRAN-Satz beginnt, die schließende, ein Satz ist zu Ende. Um die in einer FORMAT-Anweisung auftretenden Spezifikation voneinander zu trennen, ist das Komma eingeführt worden; alle Angaben gehören jedoch zu einem FORTRAN-Satz. Ein weiteres Trennzeichen ist der Schrägstrich (slash)/. Es dürfen mehrere Schrägstriche unmittelbar aufeinander folgen. Anders als das Komma trennt der Schrägstrich nicht nur Spezifikationen, sondern beendet auch einen FORTRAN-Satz und eröffnet einen neuen.

Beispiel:

```
XXX FORMAT (----,----,----)
           <--------------->
           ein Datensatz
```

Die gesamte FORMAT-Anweisung definiert einen FORTRAN-Satz, die rechte Klammer sagt also, daß ein Satz zu Ende ist.

XXX FORMAT(-------/---, -----, ---/-----, -------)

1. Satz 2. Satz 3. Satz

Infolge der Schrägstriche (als Trennzeichen) liegen drei FORTRAN-Sätze vor.

Schrägstrich im Eingabeformat

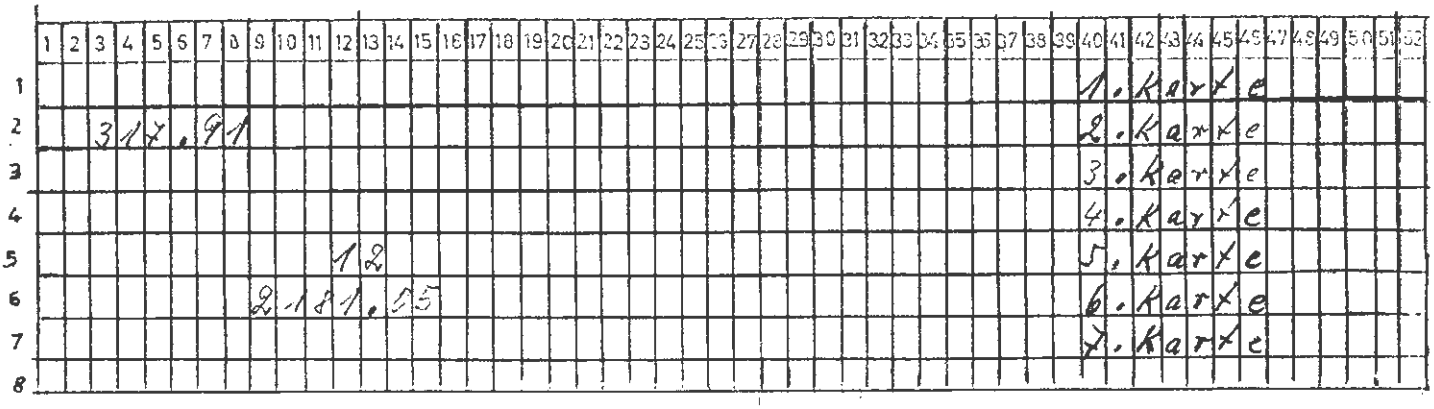
Wird bei der Eingabe in der zugehörigen FORMAT-Anweisung ein Schrägstrich angetroffen, so ist der Satz für die Eingabe beendet. Das bedeutet, daß mögliche weitere Informationen auf der Karte überlesen werden und die nächste Karte als neuer FORTRAN-Satz gilt. Folgen n Schrägstriche unmittelbar aufeinander, so werden (n-1) Karten überlesen, wenn die Schrägstriche Spezifikationen trennen. Stehen sie jedoch unmittelbar nach der öffnenden oder vor der schließenden Klammer, so werden n Datenkarten überlesen.

Beispiel 1:

Programmteil:

```
      C
            READ (1,200) ASTOR,LUX,PLUTO
       200  FORMAT (/F10.3///10X,I4/E15.5/)
      C
```

Eingabedaten:



Wirkung: Überlies die erste Datenkarte, von der zweiten lies mit F10.3 den Wert von ASTOR (317.91), danach überlies zwei Karten, dann wird der Wert für LUX (120) unter I4 (in der 11. bis 14. Spalte abgelocht) und von der folgenden Karte unter E15.5 der Wert für PLUTO (2181.05) eingelesen. Die folgende Karte wird auch noch überlesen.

Schrägstrich im Ausgabeformat

Mit WRITE wird unter Beachtung der zugehörigen FORMAT-Anweisung mindestens ein neuer FORTRAN-Satz ausgegeben. Die linke Klammer in der FORMAT-Anweisung zeigt den Beginn eines neuen FORTRAN-Satzes an. Wird ein Schrägstrich angetroffen, so wird dieser Satz abgeschlossen und ein neuer eröffnet. Das erste Zeichen in einem Satz wird stets als Vorschubsteuerzeichen interpretiert. Analog zur Eingabe gilt: Folgen n Schrägstriche unmittelbar aufeinander, so werden (n-1) Leerzeichen ausgegeben, falls die Schrägstriche Spezifikationen trennen. Stehen sie jedoch unmittelbar nach der öffnenden oder vor der schließenden Klammer, so werden n Leerzeilen ausgegeben.

Beispiel 2:

Die im Beispiel 1 eingelesenen Werte von ASTOR, PLUTO und LUX sollen ausgegeben werden.

```
      WRITE (2,205) ASTOR,LUX,PLUTO
205   FORMAT (1H1//9H ASTOR = ,F10.3/7HOLUX = ,I5//9H PLUTO = ,F10.3//)
```

Wirkung:

Aufgrund der angegebenen FORMAT-Anweisung wird beim Drucken auf eine neue Seite gegangen, danach eine Leerzeile erzeugt. Das erste Zeichen (hier Leerzeichen) vom Text (∇ ASTOR∇=∇) wird als Vorschubsteuerzeichen interpretiert; hinter diesem Text wird der Wert von ASTOR ausgedruckt. Dieser FORTRAN-Satz ist aufgrund des Schrägstriches beendet und ein neuer wird eröffnet. Wiederum wird das erste Zeichen vom Text (hier Null) als Vorschubsteuerzeichen interpretiert. Infolge dieser Angabe wird eine Zeile übersprungen. Der Text LUX∇=∇ und der Wert von LUX werden nach dieser Zeile ausgedruckt. Nach einer Leerzeile wird der Text PLUTO∇=∇ und der Wert von PLUTO ausgegeben. Auch hier wird das erste Zeichen des Textes (Leerzeichen) als Vorschubsteuerzeichen behandelt. Es werden noch zwei weitere Leerzeilen über den Drucker ausgegeben, da die beiden Schrägstriche zusammen mit der schließenden Klammer zwei FORTRAN-Sätze eröffnen und gleich wieder abschließen.

Ergebnisausdruck:

```
 ASTOR =     317.910

 LUX =    120

 PLUTO =    2181.050
```

Es ist bis jetzt nur möglich gewesen, eine einzige Spezifikation mittels des zugehörigen Wiederholungsfaktors entsprechend oft wiederholen zu lassen. Die Wiederholung einer Gruppe von Spezifikationen wird dadurch erreicht, daß diese Gruppe in runde Klammern - man bezeichnet sie als Klammern 1. Ordnung - eingeschlossen und der Wiederholungsfaktor eine vorzeichenlose INTEGER-Konstante sein muß. Wird kein Wiederholungsfaktor angegeben, so wird sein Wert als 1 angenommen. Diese Form der Darstellung wird einfache Gruppe genannt. Innerhalb der Klammern 1. Ordnung darf wiederum eine Gruppe von Spezifikationen von Klammern eingeschlossen werden. Der öffnenden Klammer darf

wiederum eine Gruppe von Spezifikationen von Klammern eingeschlossen werden. Der öffnenden Klammer darf ein Wiederholungsfaktor voranstehen. Diese Klammern werden Klammern 2. Ordnung genannt. Tiefer als bis zur Ordnung zwei darf nicht geschachtelt werden.

Beispiel 3:

Programmteil:

```
C
      AJAX=5.6
      PLUTO=-7.5
      LUX=3
      LEO=-9
      WRITE (2,210) AJAX,LUX,PLUTO,LEO
 210  FORMAT (1H0,2(F10.2,5X,I5/1X)/)
C
```

Wirkung:

Nach einer Leerzeile werden die Werte von AJAX und LUX in der einen und von PLUTO und LEO in der folgenden Zeile ausgedruckt. Danach werden noch zwei Leerzeilen über den Drucker ausgegeben.

Ergebnisausdruck:

```
  5.60          3
 -7.50         -9
```

Bei der Übertragung der Variablen wird überprüft, ob die Anzahl der Spezifikationen in der FORMAT-Anweisung mit der der Variablen in der Ein-/Ausgabeliste übereinstimmt. Dabei sind drei Fälle zu unterscheiden.

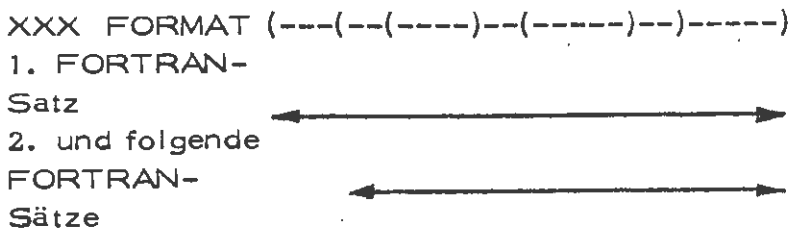
A) Anzahl der Variablen ist kleiner als die der zugehörigen Spezifikationen. Die Variablen werden der Reihe nach unter den zugeordneten Spezifikationen übertragen. Ist die letzte Variable der Liste eingelesen bzw. ausgegeben, wird die Übertragung beendet, sobald in der FORMAT-Anweisung auf eine Spezifikation zum Übertragen von Variablen gestoßen wird.

B) Anzahl der Variablen ist gleich der der zugehörigen Spezifikationen. Es erfolgt eine eindeutige Zuordnung zwischen den Spezifikationen und den Variablen. Die FORMAT-Anweisung wird vollständig abgearbeitet.

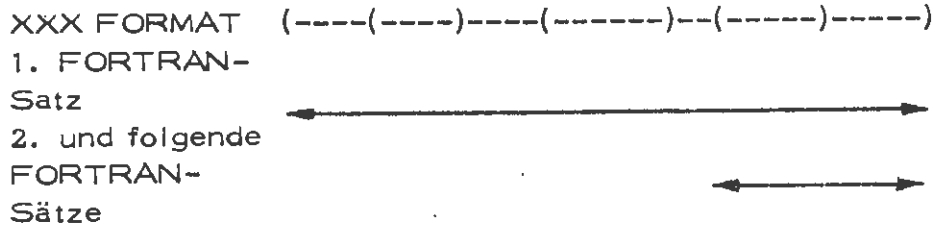
C) Anzahl der Variablen ist größer als die der zugehörigen Spezifikationen. Bei der Übertragung wird die letzte schließende Klammer der FORMAT-Anweisung erreicht, ehe alle Variablen eingelesen bzw. ausgedruckt worden sind. Der FORMAT-Satz wird abgeschlosssen und ein neuer eröffnet. Wie weiter verfahren wird, ist abhängig, ob die FORMAT-Anweisung Klammern 1. Ordnung enthält. Sind keine vorhanden, wird die gesamte Formatangabe von vorn beginnend - von neuem - abgearbeitet. Das wird so oft wiederholt,

bis alle Variablen übertragen sind.
Sind Klammern 1. Ordnung in der FORMAT-Anweisung zu finden, so wird mit der Klammer 1. Ordnung beginnend, die am weitesten rechts in der FORMAT-Anweisung steht, das Format wiederholt. (Der zugehörige Wiederholungsfaktor wird dabei berücksichtigt). Dies geschieht so oft, bis alle Variablen übertragen sind.

Beispiel 4:



Beispiel 5:



Beispiel 6:

```
C
      X=27.3954
      WRITE (2,230)X
 230  FORMAT (10X,E17.9//10X,10HWERT VON X//10X,E17.9//10X,10HWERT VON Y
     1//)
C
```

Wirkung:

In der ersten Zeile wird der Wert von X und nach einer Leerzeile der Text WERT VON X ausgedruckt. Es folgen zwei Leerzeilen. Da kein weiterer Wert ausgegeben werden soll, werden die weiteren Spezifikationen nicht beachtet (siehe A)).

Ergebnisausdruck:

```
          .273954000E+02

     WERT VON X
```

Beispiel 7:

Programmteil:

```
      I=1
      J=2
      K=3
      A=4.5
      B=7.3
      C=-12.34
      D=789.1
      E=-963.2
      F=85.2
      G=74.25
      WRITE (2,235) I,J,K,A,B,C,D,E,F,G
235   FORMAT (3(5X,I5)/2(5X,2F10.3))
```

Wirkung:

In die erste Zeile werden die Werte von I, I, K, in die zweite Zeile die Werte von A, B, C, D ausgedruckt. Da sich noch weitere Variable in der Liste der WRITE-Anweisung befinden, werden die Spezifikationen, die in der Klammer 1. Ordnung stehen, die weitere Ausgabe bestimmen. Der Wiederholungsfaktor wird mitberücksichtigt. Ein neuer FORTRAN-Satz beginnt und deshalb werden aufgrund der Spezifikationen in der nächsten Zeile die Werte von E, F, und G ausgedruckt (siehe C)).

Ergebnisausdruck:

```
    1         2         3
  4.500     7.300          -12.340    789.100
-963.200   85.200           74.250
```

Bei der Übertragung muß stets darauf geachtet werden, daß der Typ der Variablen und die Spezifikation einander entsprechen.
Die READ- bzw. WRITE-Anweisung kann,aber muß nicht, eine Variablenliste haben. Existiert eine Liste, so muß mindestens eine von F- und X-Spezifikation verschiedene Spezifikation vorhanden sein.
Es sei noch bemerkt, daß bei der Eingabe noch unverarbeitete Daten eines Satzes übersprungen werden, sobald die Liste der Variablen endet oder ein Schrägstrich erscheint.

11.10 Ergänzungen zur Ein-/Ausgabeliste

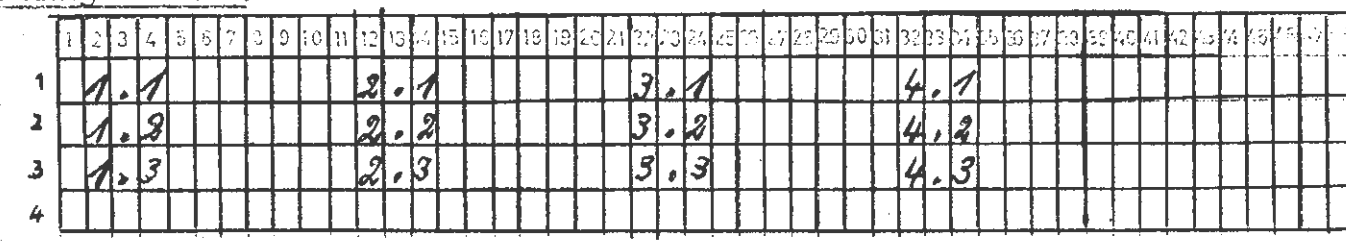
Als Listenelemente durften in einer Ein-bzw. Ausgabeliste bis jetzt nur einfache oder indizierte Variable auftreten. Will man ein gesamtes Feld übertragen, so braucht man nur den Namen des Feldes als Listenelement anzugeben. Dadurch wird erreicht, daß die Werte für alle Feldelemente eingelesen oder ausgegeben werden. Dabei ist die Reihenfolge der Feldelemente zu beachten. Ist das zu übertragene Feld A eindimensional und besteht es aus m Elementen, so wird zuerst A(1), danach A(2), usw. und zuletzt A(m) übertragen. Liegt ein mehrdimensionales Feld vor, so läuft der am weitesten links stehende Index zuerst. Bei einem zweidimensionalen Feld spricht man davon, daß "spaltenweise" eingelesen bzw. ausgegeben wird.

Beispiel 1:

Programmteil:

```
C
      DIMENSION A(4,3)
      WRITE (2,500)
500   FORMAT (1H0)
      READ (1,10)A
10    FORMAT (4F10.2)
      WRITE (2,15) A
15    FORMAT (5X,4F10.2)
C
```

Die Eingabedaten:



Wirkung:

Die Feldelemente werden in folgender Reihenfolge übertragen

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

Ergebnisausdruck:

1.10	2.10	3.10	4.10
1.20	2.20	3.20	4.20
1.30	2.30	3.30	4.30

Will man nur Teile des Feldes übertragen, so ist so ist das in der oben beschriebenen Weise nicht möglich. In diesem Falle kann man die implizite DO-Schleife verwenden. Die implizite DO-Schleife besteht aus einer Variablenliste, gefolgt von einem Komma und danach steht die implizite DO-Vereinbarung. Der gesamte Ausdruck wird in Klammern eingeschlossen. Die allgemeine Form lautet

$$(\text{Liste}, i = m_1, m_2, m_3)$$

Liste	enthält einfache oder indizierte Variable.
i	Laufvariable
m_1	Anfangswert (INTEGER – Variable oder + Konstante größer als Null)
m_2	Endwert (INTEGER – Variable oder – Konstante größer als Null)
m_3	Schrittweite (INTEGER – Variable oder – Konstante größer als Null)

Diese Laufvorschrift ist wie die Laufvorschrift in der DO-Anweisung zu interpretieren. Ist die Schrittweite gleich 1 , so braucht m_3 also nicht aufgeführt zu werden. Die allgemeine Form lautet dann

$$(\text{Liste}, i = m_1, m_2)$$

Innerhalb des Wirkungsbereichs der Schleife dürfen i, m_1, m_2 und m_3 bei der Eingabe nur im Indexausdruck enthalten sein.

Beispiel 2:

```
    DIMENSION B(2, 3, 4)
    .
    .
    WRITE (2, 75) B
75  FORMAT (E15.5)
    .
    .
    .
    hat die gleiche Wirkung wie
    .
    .
    .
    WRITE (2, 75) ((( B(I, J, K), I = 1, 2), J = 1, 3), K = 1, 4)
    .
    .
    .
    und wie
    .
    .
    .
    DO 10 K= 1, 4
    DO 10J = 1, 3
    DO 10 I = 1, 2
10  WRITE (2, 75) B (I, J, K)
    .
    .
    .
```

Bei einer anderen FORMAT-Anweisung muß das letzte Beispiel nicht die gleiche Wirkung wie die beiden vorhergehenden haben!

Beispiel 3:

Die im Beispiel 1 eingelesene Matrix soll "zeilenweise" ausgedruckt werden.

Programmteil:

```
      C
            WRITE (2,20) ((A(I,J),J=1,3),I=1,4)
       20   FORMAT (5X,3F10.2)
      C
```

Ergebnisausdruck:

```
      1.10        1.20        1.30
      2.10        2.20        2.30
      3.10        3.20        3.30
      4.10        4.20        4.30
```

Beispiel 4:

In der Eingabeliste dürfen auch für die implizite DO-Schleife Indexgrenzen als Listenelemente auftreten, allerdings müssen sie vor Eintritt in die implizite DO-Schleife Werte erhalten haben. Es ist also folgende READ-Anweisung möglich

```
      :
      :
READ (1,80) K, L,((A(I,J), J = 1, K), I = 1,L)
      :
      :
      :
```

11.11 Der Skalenfaktor P

Mittels des Skalenfaktors kann bei der Übertragung von reellen Variablen unter F-, E-, D- und G-Spezifikation die Stellung des Dezimalpunktes verschoben werden. Die allgemeine Form lautet

n P c

n ist eine ganzzahlige Konstante mit oder ohne Vorzeichen. n stellt die Zehnerpotenz dar, mit der der interne Wert der zu übertragenen Variablen für seine externe Darstellung modifiziert wird und umgekehrt.

c steht als Abkürzung für die Spezifikationen rFw.d, rEw.d, rDw.d bzw. rGw.d

Ist in einer FORMAT-Anweisung ein Skalenfaktor angegeben, so gilt er solange für alle folgende F-, E-, D- und G- Spezifikationen des gleichen Formats, bis ein anderer Skalenfaktor angetroffen wird.

Eingabe:

Bei der Eingabe ist zu unterscheiden, ob die Daten mit oder ohne Exponent abgelocht wurden. Dabei ist es unwesentlich, ob unter F- E-, G- oder D-Spezifikation eingelesen werden soll. Sind die Daten ohne Exponent, so bewirkt der Skalenfaktor, daß folgende Beziehung gilt:

$$\text{interne Größe} = \text{externe Größe} \cdot 10^{-n}$$

Bei der Eingabe von Daten mit Exponenten hat der Skalenfaktor keine Wirkung.

Ausgabe:

Soll ein Wert unter F-Spezifikation ausgegeben werden, so verändert der Skalenfaktor den Wert wie folgt:

$$\text{externe Größe} = \text{interne Größe} \cdot 10^{n}$$

Bei der Ausgabe unter E- und D-Spezifikation wird die Mantisse des auszugebenden Wertes mit 10^n multipliziert und vom Exponenten n subtrahiert. Der Wert verändert sich also nicht. Somit hat man die Möglichkeit, in nichtnormalisierter Form auszugeben.
Für die Mantisse gilt dann:

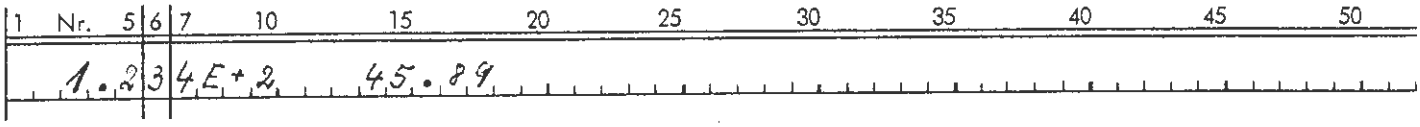
1) Bei $n \leq 0$ gibt es $-n$ führende Nullen und d^*n signifikante Stellen nach dem Dezimalpunkt.

2) Bei $n > 0$ gibt es n signifikante Stellen vor und $d - n + 1$ Ziffern nach dem Dezimalpunkt.

Ist die G-Spezifikation angegeben, so wird immer der interne Wert ausgedruckt. Wird aufgrund der Kriterien das G-Format unter F-Spezifikation geschrieben, so hat also der Skalenfaktor keine Wirkung. Wird der Gebrauch der E-Spezifikation verlangt, so gilt oben Beschriebenes.
Wird kein Skalenfaktor angegeben, so wird n = 0 angenommen.

Beispiel:

In dem folgende Programmbeispiel haben die Datenkarten gleiches Format und enthalten die gleichen Werte. Die Datenkarten sind wie folgt abgelocht:



Programmteil:

```
C
      READ (1,45) A,B
 45   FORMAT (2F10.2)
      WRITE (2,50) A,B
 50   FORMAT (1H0,5X,2F10.4)
C
      READ (1,55) A,B
 55   FORMAT (2P2F10.2)
      WRITE (2,50) A,B
C
      READ (1,60) A,B
 60   FORMAT (-2P2F10.2)
      WRITE (2,50) A,B
C
      READ (1,45) A,B
C
      WRITE (2,65) A,B
 65   FORMAT (1H0,5X,2P2F10.2)
C
      WRITE (2,70) A,B
 70   FORMAT (1H0,5X,-2P2F10.2)
C
      WRITE (2,75) A,B
 75   FORMAT (1H0,5X,2P2E10.2)
C
```

Ergebnisausdruck:

```
      123.4000    45.8900

      123.4000      .4589

      123.4000 4589.0000

      12340.00    4589.00

          1.23        .46

      12.3E+01   45.9E+00
```

11.12 Das variable Format

In manchen Fällen ist es sinnvoll das FORMAT der zu übertragenden Daten erst während der Ausführung des Programms festzulegen. In FORTRAN ist das in der Weise möglich, daß die FORMAT-Spezifikationen einschließlich öffnender und schließender Klammer in eine Variable oder in ein Feld - mit dem ersten Feldelement beginnend - gespeichert werden. Dies kann durch Einlesen unter der A-Spezifikation, durch Textübergabe in einer SUBROUTINE-Parameterliste (siehe *14.3*) oder mittels einer DATA-Anweisung (siehe *18.*) geschehen. Anstelle der Statementnummer des FORMAT's in der READ- bzw. WRITE-Anweisung steht dann der Name der einfachen Variablen oder des Feldes. Es ist empfehlenswert zum Speichern von Text INTEGER-Größen zu verwenden.

Achtung:

Das variable Format darf keine H-Spezifikation enthalten.

Beispiel:

Dieses Beispiel wurde an einer Anlage gerechnet (TR 440), deren Worte 4 Zeichen enthalten können.

Programmteil:

```
000010          INTEGER IDA(3),FELD(6)
000020          WRITE(6,500)
000030   500    FORMAT(1H0)
000040          READ(5,100) IDA
000050   100    FORMAT(3A4)
000060          I=5678
000070          WRITE(6,10)
000080   10     FORMAT(10X,10HWERT VON I)
000090          WRITE(6,IDA) I
000100          READ(5,105) FELD
000110   105    FORMAT(6A4)
000120          R=3.4
000130          F=R*R*3.1415926536
000140          WRITE(6,15)
000150   15     FORMAT(10X,31HKREISRADIUS R    KREISFLAECHE F)
000160          WRITE(6,FELD) R,F
000170          END
```

Ergebnisausdruck:

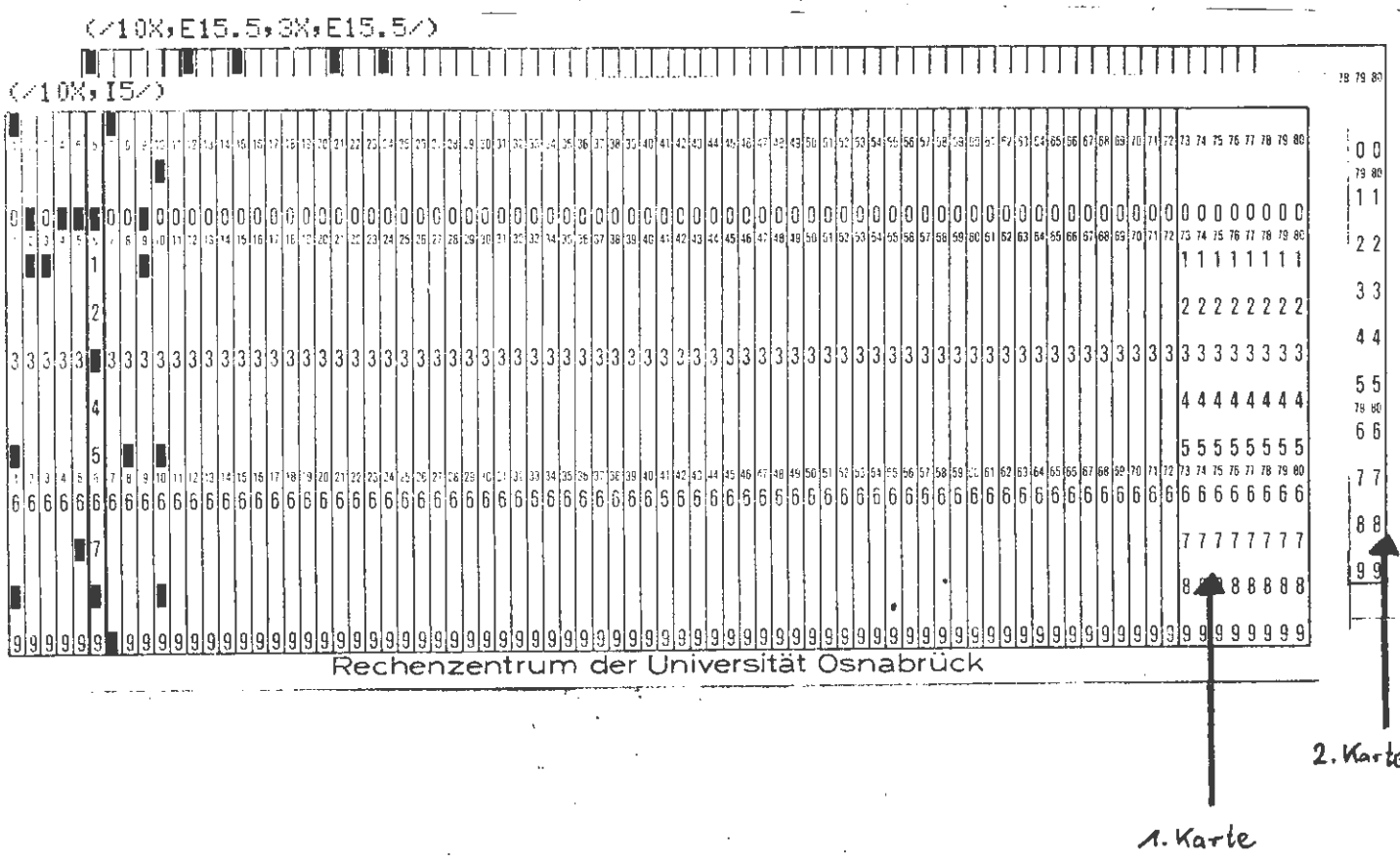
```
 WERT VON I

  5678

 KREISRADIUS R     KREISFLAECHE F

     0.34000E+01       0.36317E+02
```

Eingabedaten:



12. Logische Ausdrücke und Anweiseungen

12.1 Logische Variable und Logische Ausdrücke

Neben den Variablen, die zur Aufnahme einer Zahl dienen (REAL und INTEGER), gibt es Variable, die genau nur einen der beiden logischen Werte .TRUE. (wahr) oder .FALSE. (falsch) annehmen können. Variablen dieser Art werden deklariert durch:

LØGICAL Liste

Die Anweisung ist nicht ausführbar und muß stets vor den ausführbaren Anweisungen eines Programms stehen.

Beispiel: LØGICAL L1,L2,L3 (7)

Hierdurch werden zwei einfache logische Variable (L1 und L2) sowie ein logisches Feld L3 mit 7 Elementen deklariert.

Ein logischer Ausdruck definiert eine logische Verknüpfung, die den Wahrheitswert .TRUE. oder .FALSE. besitzt. Die einfachste Form eines logischen Ausdrucks -auch logischer Primärausdruck genannt- besteht aus:

1.) Einer logischen Konstanten.	.TRUE.
2.) Einer einfachen logischen Variablen.	LØGICAL BØØLE
3.) Einer indizierten logischen Variable.	LØGICAL LØGIK(10)
4.) Dem Aufruf einer logischen Function.	PRAWDA(MAØ)
5.) Einem Vergleichsausdruck.	A. GT. 16.
6.) Einem geklammerten logischen Ausdruck.	(B.LE.C)

Die unter Punkt 1...6 aufgeführten Möglichkeiten bilden die Elemente, aus denen mit Hilfe von logischen Operatoren beliebige logische Ausdrücke aufgebaut werden können.

12.1.1 Vergleichsausdrücke

Ein Vergleichsausdruck besteht aus zwei arithmetischen Ausdrücken, die durch einen Vergleichsoperator miteinander verknüpft sind. In FORTRAN unterscheidet man sechs Vergleichsoperatoren (die Punkte sind wesentlicher Bestandteil des Operators):

Vergleichsoperator	Bedeutung
.GT.	> ; Größer als (Greater Than).
.GE.	≥ ; Größer als oder gleich (Greater than or Equal to).
.LT.	< ; Kleiner als (Less Than).
.LE.	≤ ; Kleiner als oder gleich (Less than or Equal to).
.EQ.	=; Gleich (Equal to).
.NE.	≠; Ungleich (Not Equal).

Die Vergleichsoperatoren drücken eine logische Bedingung aus, die entweder .TRUE. oder .FALSE.sein kann. Von welchem Typ die beiden arithmentischen Ausdrücke sein dürfen, die durch den Vergleichsoperator miteinander verknüpft werden, gibt folgende Tabelle wieder:

Typ des arithmetischen Ausdrucks	INTEGER	REAL	DØUBLE	CØMPLEX
INTEGER	JA	NEIN	NEIN	NEIN
REAL	NEIN	JA	JA	NEIN
DØUBLE	NEIN	JA	JA	NEIN
CØMPLEX	NEIN	NEIN	NEIN	NEIN

Bzgl. zusätzlicher erlaubter Möglichkeiten auf dem TR 440 siehe Anhang 1.

Beispiele für Vergleichsausdrücke:

```
ALPHA .GT. 16.
REST - QUØT(I) *ZETA .LE. 3.141592
BETA - CAESAR .NE. DELTA + AESØP
RØUND(I) .GE. RØUND (I-1)
KØNST .LT. 16
IDA .EQ. JØSEF(K)
(IDA) .EQ. (JØSEF(K))
```

Es gelte folgende Typzuweisung:

```
REAL ERNIE
INTEGER ALEX, IDA, FRED
LØGICAL LET
CØMPLEX CØMPL
```

Dann sind folgende Vergleichsausdrücke unzulässig:

CØMPL .GE. (3.5, 7.25)	Komplexe Ausdrücke können nicht miteinander verglichen werden.
ERNIE .LT. IDA	REAL-Variable kann nicht mit einer INTEGER-Variablen verglichen werden.
.GT. 15	Der arithmetische Ausdruck vor dem Vergleichsoperator fehlt.
LET .EQ. (ALEX + FRED)	Logische Variable dürfen nicht bei Vergleichsausdrücken auftreten.
ERNIE **3 .LT 94.1E1	Beim Vergleichsoperator fehlt der abschließende Punkt.

12.1.2 Logische Operatoren

In FORTRAN sind drei logische Operatoren definiert:

1.	.NØT.	Logisches NICHT (Negation)
2.	.AND.	Logisches UND (Konjunktion)
3.	.ØR.	Logisches ODER (Disjunktion)

Die beiden Punkte zu beiden Seiten der Operatoren müssen stets angegeben werden.

X und Y mögen logische Konstanten, logische Variablen oder Ausdrücke mit Vergleichsoperatoren darstellen. Dann haben die logischen Operatoren folgende Bedeutung:

.NOT. X hat den Wert .TRUE., wenn X den Wert .FALSE. hat

.NOT. X hat den Wert .FALSE., wenn X den Wert .TRUE. hat.

X .AND. Y bedeutet, der Ausdruck X .AND. Y hat genau dann den Wert .TRUE., wenn sowohl X als auch Y den Wert .TRUE. haben. In allen anderen Fällen hat der Ausdruck den Wert .FALSE.. Diesen Sachverhalt gibt die folgende Wahrheitstafel für .AND. wieder:

Y = \ X =	.TRUE.	.FALSE.
.TRUE.	.TRUE.	.FALSE.
.FALSE.	.FALSE.	.FALSE.

X .OR. Y bedeutet, der Ausdruck X .OR. Y hat genau dann den Wert .TRUE., wenn entweder X oder Y oder beide den Wert .TRUE. haben. Sonst hat der Ausdruck den Wert .FALSE.. Diesen Sachverhalt gibt die folgende Wahrheitstafel für .OR. wieder:

Y = \ X =	.TRUE.	.FALSE.
.TRUE.	.TRUE.	.TRUE.
.FALSE.	.TRUE.	.FALSE.

Im allgemeinen dürfen nicht zwei logische Operatoren hintereinander stehen. Eine Ausnahme bildet nur der logische Operator .NOT.. Erlaubt sind also nur:

.AND. .NØT.
.ØR. .NØT.

nicht erlaubt ist .AND. . ØR. oder .NØT. .NØT.

Es gelte folgende Typzuweisung:

```
REAL RØBIN, ERNIE
INTEGER ALEX, IDA, FRED
LØGICAL LET, WHAT
CØMPLEX CØMPL
```

Folgende Ausdrücke sind dann zulässig:

```
LET .AND..NØT. (IDA .GT. FRED)
.NØT. WHAT .AND. .NØT. LET
LET .AND..NØT. WHAT .ØR. IDA .GT. FRED
```

(Bezüglich Auswertung siehe 12.1.3)

Folgende Ausdrücke sind jedoch unzulässig:

```
(RØBIN *ALEX .GT. ALEX) .AND. WHAT
```

Der Vergleichsausdruck enthält Variablen gemischten Typs.

```
ALEX .AND. LET
```

ALEX ist kein logischer Ausdruck

.ØR. WHAT	Vor .OR. muß ein logischer Ausdruck stehen.
NØT. (ALEX .GT. ERNIE)	Vor dem Operator fehlt der Punkt.
(CØMPL .EQ. IDA) .AND. LET	Eine komplexe Größe darf nicht Operand eines Vergleichsoperators sein.

12.1.3 Auswertung von logischen Ausdrücken

Wie bei den arithmetischen Ausdrücken gibt es auch bei den logischen Ausdrücken und daher auch bei der Kombination beider eine Hierarchie in der Auswertung, die wie folgt aussieht:

1. Stufe Auswertung von Funktionsaufrufen und Klammerungen
2. Stufe ** Exponentiation
3. Stufe *, / Multiplikation und Division
4. Stufe +, - Addition und Subtraktion
5. Stufe .GT., .GE., .LT., .LE., .EQ., .NE. Vergleichsoperatoren
6. Stufe .NOT.
7. Stufe .AND.
8. Stufe .OR.

} logische Operatoren

Beispiel: Auswertung von A.GT.D**B.AND..NØT.L.ØR.N
RX = Zwischenergebnis vom Typ REAL
LX = Zwischenergebnisse vom Typ LOGICAL

1. D**B → R1
2. A.GT.R1 → L1
3. .NØT.L → L2
4. L1.AND.L2 → L3
5. L3.ØR.N → L4 (Ergebnis)

Genau wie bei den arithmetischen Ausdrücken kann bei den logischen Ausdrücken durch das Setzen von Klammern eine besondere Reihenfolge der Auswertung erzwungen werden. Der in Klammern stehende Teilausdruck wird dann als Ganzes ausgewertet und als Operand weiterverwendet.

Beispiel: Auswertung von .NØT.((B.GT.C.ØR.K) .AND.L)

1. B.GT.C → L1
2. L1.ØR.K → L2
3. L2.AND.L → L3
4. .NØT.L3 → L4 (Ergebnis)

Man beachte: Bezieht sich der logische Operator .NOT. auf mehr als einen logischen Ausdruck, so müssen Klammern gesetzt werden. Der logische Ausdruck

.NØT.X.ØR.Y

bedeutet (.NØT.X).ØR.Y

und nicht .NØT.(X.ØR.Y)

12.2 Logische Anweisungen

Die allgemeine Form der logischen Anweisung lautet:

A = B

Dabei ist A eine einfache oder indizierte logische Variable und B ein logischer Ausdruck. Die logische Anweisung besagt, daß der Variablen A der Wert zuzuweisen ist, der sich aus der Auswertung des logischen Ausdrucks B ergibt.

Analog zur arithmetischen Anweisung ist auch die logische Anweisung keine algebraische Gleichung, obwohl sie mit dem Gleichheitszeichen geschrieben wird.

Schreibt man

```
REAL A,B
LOGICAL U
  .
  .
  .
U = A .LT. B
```

so erhält U genau dann den Wert .TRUE., wenn A kleiner als B ist und den Wert .FALSE., wenn A größer oder gleich B ist.

Beispiele:

```
LOGICAL A, B, C, D
```

`A = .FALSE.`	Der logischen Variablen A wird der Wahrheitswert .FALSE. zugewiesen.
`B = C .AND. .NOT. D`	Der logischen Variablen B wird der Wahrheitswert .TRUE. zugewiesen, wenn C den Wert .TRUE. und D den Wert .FALSE. haben.
`C = 2. .LT. X`	Die logische Variable C erhält den Wert .TRUE., wenn X größer als 2.0 ist, sonst erhält C den Wert .FALSE. .
`D = .NOT. (X .GT.Y)` `D = .NOT. X .GT. Y`	D wird .FALSE. für X >Y und wird .TRUE. für X ≤ Y.

13. Weitere Steuerungsanweisungen

13.1 Bedingte GOTO's

Unter einem bedingten GOTO versteht man eine Sprung- bzw. Steueranweisung, die bestimmte Bedingungen an irgendwelche Größen innerhalb des Programms stellt und daraufhin die Programmverarbeitung in bestimmte Programmzweige lenkt.

13.1.1 Computed GOTO

Das computed (errechnete) GOTO besteht aus der Anweisung:

GOTO(n1,n2,n3,.....,nm),I

Dabei sind n1,n2,n3,.....,nm die Anweisungsnummern der verschiedenen Sprungziele und I ist eine nichtindizierte INTEGER-Variable, für die gilt: $1 \leq I \leq m$.

Hat die INTEGER-Variable I den Wert K, so wird die K-te Anweisungsnummer der in den Klammern angegebenen Liste angesprungen. Liegt der Wert von I außerhalb des zulässigen Bereichs, so ist der Transfer undefiniert. Dieser Fehler wird erst zur Objektzeit entdeckt und kann dann zu einer Fehlermeldung führen.

Beispiel:

```
      .
      .
      .
   GØTØ (1,5,10,40), JUMP
      .
      .
      .
10 ALPHA = BETA + GAMMA
      .
      .
      .
40 CØNST = EMIL **2. + ALPHA
      .
      .
      .
 1 LETTER = .FALSE.
      .
      .
      .
 5 BETA = 47.4321
```

Hat JUMP den Wert 1, so wird als nächstes die Anweisung mit der Nummer 1 ausgeführt. Ist JUMP gleich 3, so wird die Anweisung mit der Nummer 10 ausgeführt.

Aufgabe:

In einem Programmausschnitt sollen die Größen x und y in Abhängigkeit von der Steuergröße J berechnet werden. An-

schließend soll das Programm für alle Fälle gemeinsam weitergeführt werden. Es soll gelten:

$x = ab$	und	$y = a + c - 2$	für J = 1
$x = \frac{ab}{2} + c$	und	$y = a + c - \sqrt{b}$	für J = 2
$x = a^2 + b^2 - \sqrt{c}$	und	$y = (a + b)\frac{c}{2}$	für J = 3
$x = \text{asin}(c) + b$	und	$y = \sqrt{a + b} - \sin(c)$	für J = 4

Lösung:

```
      .
      .
      .
      GØTØ (10,20,30,40), J
10    X = A * B
      Y = A + C - 2.
      GØTØ 50
20    X = A *B/2. + C
      Y = A + C - SQRT (B)
      GØTØ 50
30    X = A**2  + B**2  - SQRT(C)
      Y = (A + B) *C/2.
      GØTØ 50
40    X = A*SIN(C) + B
      Y = SQRT (A + B) - SIN(C)
50          .
            .
            .
```

13.1.2 Assigned GOTO

Das assigned (zugewiesene) GOTO besteht aus zwei Anweisungen:

```
ASSIGN n TO i
      .
      .
      .
GOTO i, (n1,n2,n3,....nm)
```

Hierbei ist n eine Anweisungsnummer, i ist eine nichtindizierte INTEGER-Variable und n1, n2,....nm sind die Anweisungsnummern der verschiedenen Sprungziele. Das assigned GOTO bewirkt, daß das Programm mit der Anweisung fortgesetzt wird, deren Anweisungsnummer durch den Wert der INTEGER-Variablen i gegeben ist.

Wird der INTEGER-Variablen i eine Anweisungsnummer zugewiesen, die in dem zugehörigen assigned GOTO nicht vorhanden ist, so ist der Transfer undefiniert. Dieser Fehler wird erst zur Objektzeit entdeckt und führt dann zu einer Fehlermeldung.

Beispiel:

```
      :
ASSIGN 10 TØ IZWEIG
      :
8 GØTØ IZWEIG, (5,10,15,20)
      :
10 Y = X**2 + ARRAY (12)
      :
```

Der INTEGER-Variablen IZWEIG wird der Wert 10 zugewiesen. Bei Erreichen der Anweisung mit der Nummer 8 wird zu der Anweisung mit der Nummer 10 gesprungen und diese ausgeführt. Die Wirkung ist also die gleiche, als ob man GOTO 10 programmiert hätte.

Folgende Regeln sind beim assigned GOTO zu beachten:

1. Der INTEGER-Variablen i darf nur durch eine ASSIGN-Anweisung ein Wert zugewiesen werden, nicht jedoch durch eine arithmetische Anweisung.

2. Mindestens eine ASSIGN-Anweisung muß vor Erreichen des assigned GOTO ausgeführt worden sein.

3. Die INTEGER-Variable i darf keine indizierte Variable sein.

4. Zwischen der ASSIGN-Anweisung und der zugehörigen GOTO-Anweisung dürfen andere Anweisungen stehen, in denen jedoch die INTEGER-Variable i nicht vorkommen darf.

Beispiel:

```
      :
   ASSIGN 25 TO IBRNCH
11 GØTØ IBRNCH, (5, 10, 15, 20, 25)
      :
 5 ALPHA = BETA + GAMMA
      :
25 XNEW  = XALT + TRANSX
   ASSIGN 15 TO IBRNCH
   GØTØ 11
15 YNEW  = YALT + TRANSY
      :
```

13.2 Die logische IF-Anweisung

Die logische IF-Anweisung hat folgende Form:

IF(a)s

Dabei stellt a einen logischen Ausdruck dar und s eine ausführbare Anweisung. s darf keine DO-Anweisung oder eine andere logische IF-Anweisung sein. Beim logischen IF wird der in den Klammern stehende logische Ausdruck ausgewertet. Hat er den Wert .TRUE., so wird die Anweisung s ausgeführt. Hat er den Wert .FALSE., so wird nicht die Anweisung s, sondern die dem logischen IF unmittelbar folgende Anweisung ausgeführt. Ist das IF die letzte Anweisung einer DO-Schleife, so bedeutet "unmittelbar folgende Anweisung" logisch, daß die Schleife mit dem nächsten Wert des Laufindex durchzuführen ist. Daher sollte hinter das IF eine CONTINNE-Anweisung gesetzt werden und diese mit dem Label des DO versehen werden.

1. Beispiel:

```
      .
      .
      .
      IF(X.LE.0.0)GOTO5
    2 WERT=WERT+25.4
    3 IF(X.GT.Y)POSIT=X/WERT+4.
      FAST=D*PI*N
      .
      .
      .
    5 ZETA=-X**W
      .
      .
      .
```

Ist beim ersten logischen IF die Variable X kleiner-gleich Null, dann hat der logische Ausdruck den Wert .TRUE. und es wird auf Anweisung 5 gesprungen. Ist X größer als Null, so hat der logische Ausdruck den Wert .FALSE. , die Sprunganweisung wird übergangen und als nächstes die Anweisung 2 ausgeführt. Bei Anweisung 3 wird dann geprüft, ob X größer als Y ist. Ist dies der Fall, der logische Ausdruck also .TRUE. , so wird POSIT der entsprechende Wert zugewiesen und anschließend bei der nächsten Anweisung weitergearbeitet. Lieferte der logische Ausdruck den Wert .FALSE. , so bleibt POSIT unverändert und es wird sofort die nächste Anweisung ausgeführt.

2. Beispiel:

```
      .
      .
      .
    4 IF (X .LT. 0.0) X = -X
    6 P = SQRT (X)
      .
      .
      .
```

Hat der logische Ausdruck in den Klammern den Wert .TRUE. , so wird der Variablen X ihr aktueller Wert mit umgekehrtem Vorzeichen zugewiesen und dann Anweisung 6 ausgeführt. Hat der logische Ausdruck den Wert .FALSE. , so bleibt X unverändert und es wird sofort Anweisung 6 ausgeführt.

14. Unterprogramme

In einem Programm ist es oft notwendig, den gleichen Rechnungsgang mit unterschiedlichen Werten an verschiedenen Stellen zu wiederholen. Eine wesentliche Vereinfachung beim Erstellen eines Programms wird erreicht, wenn solche gleichen Rechnungsgänge nur einmal programmiert werden müssen und an beliebigen Stellen im Programm mit den jeweiligen Werten ausgeführt werden können.

Es bietet sich für derartige Fälle der Gebrauch von Unterprogrammen an. In diesem Kapitel werden Anweisungen beschrieben, mittels derer Unterprogramme programmiert werden können, wobei die Standardunterprogramme, die vom FORTRAN-Kompiler bereitgestellt werden, an anderer Stelle behandelt wurden. (Siehe Kapitel 9.)

Es gibt die folgenden Möglichkeiten, Unterprogramme zu definieren:

1. Statementfunktionen
2. FUNCTION-Unterprogramme
3. SUBROUTINE-Unterprogramme

Statementfunktionen und FUNCTION-Unterprogramme liefern bei ihrem Aufruf einen Wert, der im rufenden Programmteil unmittelbar benutzt werden kann. SUBROUTINE-Unterprogramme liefern keinen Wert, der unmittelbar weiterverarbeitet werden kann. FUNCTION und SUBROUTINE-Unterprogramme haben gemeinsam, daß über die Parameterversorgung mehrere Werte von rufender zur gerufenen Programmeinheit oder umgekehrt übergeben werden können.

Zu den Unterprogrammen werden auch die BLOCK DATA-Unterprogramme gezählt. Sie werden zur Initialisierung von COMMON-Blöcken benutzt, können aber nicht im Programm aufgerufen werden.

Funktionen

Funktionen dienen zur Berechnung eines Funktionswertes aus einem oder mehreren Argumenten. Um eine Funktion zu benutzen, ist es erforderlich:

a) die Funktion zu definieren (d. h. anzugeben, wie und welche Rechenoperationen auszuführen sind).
b) an der entsprechenden Programmstelle die Funktion durch ihren Namen aufzurufen.

Definition einer Funktion

Die Definition einer Funktion besteht aus dreierlei:

a) Der Funktion wird ein eindeutiger Name zugeordnet, mit welchem sie aufgerufen werden kann. Der Name besteht aus 1 bis 6 alphanumerischen Zeichen, von denen das erste ein Buchstabe sein muß. Der Typ des gelieferten Funktionswertes wird durch die Typanweisung oder implizit durch den Funktionsnamen definiert.

b) Formalparameter der Funktion werden definiert.

c) Die Prozedur zur Berechnung des Funktionswertes wird definiert.

14.1 Statementfunktion

Eine Statementfunktion wird durch eine einzige arithmetische oder Boolesche Ergibtanweisung definiert, die in derjenigen Programmeinheit steht, die diese Funktion aufruft.

Die Form der Statementfunktion ist

$$\text{name}\,(a_1, a_2, \ldots\ldots, a_n) = \text{Ausdruck}$$

wobei name = Name der Statementfunktion

$a_1, a_2, \ldots a_n$ = Formalparameter: Name für nichtindizierte Variable, sie müssen innerhalb der gleichen Statementfunktion voneinander verschieden sein.

Ausdruck = arithmetischer oder Boolescher Ausdruck. Jede Statementfunktion, die in einem solchen Ausdruck benutzt wird, muß selbst vorher definiert sein.

Alle Statementfunktionsanweisungen einer Programmeinheit müssen der ersten ausführbaren Anweisung vorangehen. Die Statementfunktion muß mindestens einen Formalparameter enthalten. Anzahl, Reihenfolge und Typ der Aktualparameter im Aufruf müssen den Formalparametern entsprechen.
Im Ausdruck zur Definition einer Statementfunktion darf diese nicht selbst aufgerufen werden, jedoch können vorher definierte Statementfunktionen aufgerufen werden.

Beispiel:

```
      DIMENSIØN T2F (20, 5)
      ARSINU (X)=ALØG(X+SQRT(X*X+1.))
      WRITE(6, 1)
    1 FØRMAT (1H1)
      .
      .
      .
      DØ 60     I =1, 20
      DØ 60     K =1, 5
      .
      .
      .
      Z = 10.**(I-3)
      .
      .
      .
   60 T2F (I, K) =ARSINU (Z)
      .
      .
      .
```

Die Namen der Formalparameter können zugleich in mehreren Statementfunktionsanweisungen einer Programmeinheit und außerhalb dieser Funktionen als Variablennamen für Variablen des gleichen Typs benutzt werden.

14.2 FUNCTION- Unterprogramme

FUNCTION-Unterprogramme sind selbständige Programmeinheiten in FORTRAN, die aus einer beliebigen Anzahl von Anweisungen bestehen können.

Das Unterprogramm kommt zur Ausführung, wenn sein Name in einer anderen FORTRAN-Programmeinheit aufgerufen wird.

Die Form des FUNCTION-Unterprogrammes ist

```
typ FUNCTION name (a1, a2,.....,an)
.
.
.
RETURN
.
.
.
END
```

wobei
- name = Name der Funktion
- typ = Typ des Unterprogrammnamens: eine der Typangaben INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL. Ohne Typangabe gilt für den Funktionsnamen die implizite Typvereinbarung
- $a_1, \ldots\ldots a_n$ = Formalparameter. Name für nichtindizierte Variablen, Felder, FUNCTION- und SUBROUTINE-Unterprogramme.

Ein FUNCTION-Unterprogramm wird durch eine FUNCTION-Anweisung eingeleitet. In ihr werden der Name und die Formalparameter (mindestens einer) des Unterprogramms angegeben. In mindestens einer Anweisung des Unterprogramms muß dem Funktionsnamen ein Wert zugewiesen werden. Danach kann auf die Variable mit dem Funktionsnamen Bezug genommen oder ihr ein neuer Wert zugewiesen werden, d. h. in einer Ergibtanweisung geschieht dies durch Angabe des Funktionsnamens auf der linken Seite.
Der rekursive Aufruf ist nicht erlaubt, d. h. das FUNCTION-Unterprogramm darf sich nicht selbst aufrufen.
Jedes FUNCTION-Unterprogramm beginnt mit einer FUNCTION-Anweisung und endet mit einer END-Zeile. Von den dazwischenliegenden Anweisungen muß mindestens eine die Rücksprunganweisung RETURN sein. (s. nächste Seite)
Ein FUNCTION-Aufruf besteht aus dem Funktionsnamen, gefolgt von einer in runde Klammern eingeschlossenen Liste von aktuellen Paramtern. Diese Parameter müssen in Reihenfolge, Anzahl und Typ mit den entsprechenden formalen Parametern in der FUNCTION-Anweisung übereinstimmen.

Als aktuelle Parameter sind erlaubt:

1. ein Variablenname
2. ein Feldelementname
3. ein Feldname
4. irgendein arithmetischer oder Boolescher Ausdruck
5. der Name eines externen Unterprogramms

Wenn ein formaler Parameter ein Feldname ist, muß der entsprechende aktuelle Parameter ein Feldname oder ein Feldelement Name sein.

Beispiel:

FUNCTION-Unterprogramm

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

Aufrufende Programmeinheit

```
        .
        .
        .
A = DIV (B, C)
        .
        .
        .
```

Beim Aufruf der FUNCTION werden die Werte der aktuellen Parameter B und C für X und Y benutzt.
Sei z. B. B = 10.0 und C = 5.0, so wird der Funktion DIV der Wert B/C = 2.0 zugewiesen und mit dem RETURN wird in die aufrufende Programmeinheit zurückgekehrt.

RETURN- und END-Anweisungen im FUNCTION-Unterprogramm

Jedes FUNCTION-Unterprogramm enthält eine END- und mindestens eine RETURN-Anweisung.
Die RETURN-Anweisung bezeichnet den logischen Abschluß des Unterprogramms und bewirkt die Rückkehr in die rufende Programmeinheit. Sie kann für verschiedene Ausgänge des Unterprogramms mehrfach in diesem vorkommen.
Die END-Anweisung ist stets die letzte Anweisung des Unterprogramms; sie bezeichnet dem Kompiler das physikalische Ende des Unterprogramms.

Beispiel mit mehreren Rücksprunganweisungen

```
      FUNCTION ALTER(M,MAE,FRAU,K)
      REAL MAE
      IF(M)50,49,51
   50 FRAU = FRAU + 1.
      .
      .
      .
      ALTER = FRAU
      RETURN

C     DER PARAMETER K WIRD ALS FEHLER-
C     PARAMETER IN DIE RUFENDE PROGRAMM-
C     EINHEIT ZURUECKGEGEBEN.

   49 K = -99
      ALTER = 0.0
      RETURN
   51 MAE = MAE + 1.
      .
      .
      .
      ALTER = MAE * 10.
      .
      .
      .
      RETURN
      END
```

Das FUNCTION-Unterprogramm darf einem oder mehreren seiner Parameter einen Wert zuweisen und so effektiv neben dem Funktionswert weitere Ergebnisse in die rufende Programmeinheit zurückgeben.

14.3 SUBROUTINE-Unterprogramme

Die SUBROUTINE-Unterprogramme sind ebenfalls wie die FUNCTION-Unterprogramme selbständige Programmeinheiten.
Ein SUBROUTINE-Name besitzt bei der Rückkehr in die rufende Programmeinheit keinen Wert zum unmittelbaren Weiterrechnen. Die Wertübergabe der Ergebnisse geschieht über die Parameter. In Kapitel 15. wird noch eine weitere Möglichkeit zur Ergebnisübergabe angegeben werden.

Die allgemeine Form eines SUBROUTINE-Unterprogramms ist die folgende:

```
SUBROUTINE name(a1, a2, a3, ....., an)
:
:
:
RETURN
END
```

wobei name = Name der SUBROUTINE

$a_1, \ldots\ldots, a_n$ = Formalparameter
Namen für nicht indizierte Variable, Felder, SUBROUTINE- oder FUNCTION-Unterprogramme.

Ein SUBROUTINE-Unterprogramm wird durch eine SUBROUTINE-Anweisung eingeleitet. In ihr werden der Name und die Formalparameter des Unterprogramms angegeben. Das SUBROUTINE-Unterprogramm kann jede beliebige Fortran-Anweisung enthalten, ausgenommen sind folgende Anweisungen: weitere SUBROUTINE, FUNCTION und BLOCKDATA. Alle Variablennamen gelten nur in dem SUBROUTINE-Unterprogramm. Variable gleichen Namens in anderen Programmeinheiten haben keinerlei Beziehung zu den Variablen der SUBROUTINE. Tritt als Formalparameter ein Feldname auf, so müssen seine Dimensionen in einer DIMENSION-Anweisung definiert werden. Variable Dimensionierung siehe § 18. Der SUBROUTINE-Name darf im Unterprogramm in keiner anderen Anweisung als der SUBROUTINE-Anweisung auftreten.

Sollen die Befehle einer SUBROUTINE durchlaufen werden, so ist sie mit

CALL name (a_1, a_2, ..., a_n)

aufzurufen. Der Aufruf bewirkt, daß beim ersten ausführbaren Befehl des SUBROUTINE-Unterprogramms mit der Rechnung fortgefahren wird.

Durch den Aufruf CALL name (aktuelle Parameter) erhalten die formalen Parameter des SUBROUTINE-Unterprogramms die Werte, die die aktuellen Parameter zum Zeitpunkt des Aufrufs haben. Ein aktueller Parmeter in einem SUBROUTINE-Aufruf kann sein:

1. Eine Hollerith-Konstante
2. Ein Variablenname
3. Ein Feldelement
4. Ein Feldname
5. Irgendein anderer Ausdruck
6. Ein Name eines externen Unterprogramms

Aktuelle Parameter müssen in Typ, Reihenfolge und Anzahl mit den formalen Parametern der SUBROUTINE-Anweisung übereinstimmen.

Beispiel: (Matrizenaddition)

rufende Programmeinheit

```
      DIMENSION A(10,10),B(10,10),C(10,10)
      IDIM = 10
      JDIM = 10
           .
           .
           .
      CALL MATADD(A,B,C,IDIM,JDIM)
           .
           .
           .
      STOP
      END
```

Unterprogramm

```
      SUBROUTINE MATADD(X,Y,Z,M,N)
      DIMENSION X(10,10),Y(10,10),Z(10,10)
      DO 10 I=1,M
      DO 10 J=1,N
      Z(I,J) = X(I,J) + Y(I,J)
   10 CONTINUE
      RETURN
      END
```

Beispiel mit mehreren RETURN's

Je nach dem Wert von NSTEU wird für die Feldelemente von VOL (20, 2) das Prisma-, das Kreiskegel- oder das Kreiszylindervolumen berechnet.

Aufrufende Programmeinheit

```
      DIMENSION VOL(20,2),ERG(20)
      PI = 3,1415927
        .
        .
        .
      NSTEU = 1
        .
        .
        .
      CALL VOLUM(NSTEU,VOL,PI,ERG)
        .
        .
        .
      STOP
      END
```

SUBROUTINE-Unterprogramm

```
      SUBROUTINE VOLUM(I,FELD,PI,E)
      DIMENSION FELD(20,2),E(20)
      GOTO(100,200,300),I
  100 DO 101 IA=1,20
      E(IA) = FELD(IA,1) ** 2 * FELD(IA,2)
  101 CONTINUE
      RETURN
  200 DO 201 IA=1,20
      E(IA) = PI * FELD(IA,1) ** 2 * FELD(IA,2) / 3.
  201 CONTINUE
      RETURN
  300 DO 301 IA=1,20
      E(IA) = PI * FELD(IA,1) ** 2 * FELD(IA,2)
  301 CONTINUE
      RETURN
      END
```

Beispiel:

In einem Unterprogramm sollen die Elemente eines Feldes in absteigender Reihenfolge geordnet werden.

Rufende Programmeinheit

```
      DIMENSION A(100),B(50)
          .
          .
          .
      CALL ORDNEN(A,100)
          .
          .
          .
      CALL ORDNEN(B,50)
          .
          .
          .
      STOP
      END
```

SUBROUTINE - Unterprogramm

```
      SUBROUTINE ORDNEN(A,M)
      DIMENSION A(100)
      M1 = M - 1
      DO 10 I=1,M1
      I1 = I + 1
      DO 10 J=I1,M
      IF(A(I) .GE. A(J)) GOTO 10
      S = A(I)
      A(I) = A(J)
      A(J) = S
   10 CONTINUE
      RETURN
      END
```

15. Die COMMON-Anweisung

In umfangreichen Programmen werden bestimmte Variablen und Felder in allen oder in den meisten Programmsegmenten benötigt. In einfachen Fällen, d. h. bei wenigen Variablen oder Feldern, läßt sich das mittels der Parameterlisten zwischen dem Haupt- und den Unterprogrammen bewerkstelligen. Bei mehr als 10 Parametern wird diese Methode jedoch unübersichtlich und die Fehlerquote steigt erheblich.

Es gibt daher in FORTRAN eine andere Möglichkeit, Daten in Unterprogramme zu übertragen und Ergebnisse wieder zurückzuübergeben, nämlich mit Hilfe der COMMON-Anweisung.

15.1 Die einfache COMMON-Anweisung

Die in einer COMMON-Anweisung aufgeführten Variablen oder Felder werden im COMMON-Speicherbereich abgespeichert, zu dem alle Programme Zugriff haben, in denen dieser Bereich definiert wurde. Die allgemeine Form der COMMON-Anweisung lautet:

COMMON a1, a2, a3, ..., an

Hierbei sind a1, a2, a3, ... an Namen von Variablen und/oder Feldern, die nicht schon als Namen formaler Parameter verwendet wurden.

Beispiel:

```
CØMMØN ANTØN, BERTA, CAESAR, IDA
```

Durch diese Anweisung werden im COMMON-Bereich vier Speicherplätze für die Variablen ANTON, BERTA, CAESAR und IDA reserviert. Von allen Programmen aus, die diese Anweisung enthalten, kann auf die vier Variablen Bezug genommen werden.

Jedes Programm oder Unterprogramm darf mehrere COMMON-Anweisungen enthalten, wobei die Speicherplätze den Variablen derart zugeordnet werden, als ob es sich nur um eine Fortsetzung der ersten Anweisung handele. Daher haben die Anweisungen

```
CØMMØN ANTØN, BERTA, CAESAR, IDA
CØMMØN I, J, K, R, T
```

die gleiche Speicherplatzanordnung wie die Anweisung

```
CØMMØN ANTØN, BERTA, CAESAR, IDA, I, J, K, R, T
```

Es ist natürlich erlaubt, für nachfolgende COMMON-Anweisungen auch Folgekarten anstelle einer neu definierten COMMON-Anweisung zu ver-

wenden. Für den praktischen Gebrauch ist jedoch von einer zu großen Anzahl von Folgekarten abzuraten, da dadurch bei evtl. eintretenden Änderungen von Variablen ein größerer Arbeitsaufwand notwendig wird.

Die COMMON-Anweisung gehört zu den nichtausführbaren Anweisungen und steht stets vor der ersten ausführbaren Anweisung eines Programms. Jeder Variablen bzw. jedem Feld wird durch die COMMON-Anweisung ein fester Speicherplatz im COMMON-Speicherbereich zugewiesen. Die Gesamtzahl der Speicherplätze wird als "Länge der COMMON-Liste" bezeichnet.

Es ist nicht gestattet, innerhalb einer COMMON-Anweisung in einem Programmsegment mehrmals die gleiche Variable aufzuführen. Die Anweisung

```
COMMON A, B, RHO, C, A
```

ist also unzulässig. In jedem Haupt- und Unterprogramm müssen diejenigen Variablen, welche von einem Programmsegment an ein anderes übergeben werden sollen, durch eine COMMON-Anweisung festgelegt werden. Dabei ist darauf zu achten, daß zwar die Längen der COMMON-Listen unterschiedlich sein können, jedoch die Reihenfolge der abzuspeichernden Variablen und Felder in jedem Falle gleich sein muß. Andernfalls kommt es zu Überspeicherungen im COMMON-Bereich, da die Speicherplätze den Variablen stets in der Reihenfolge ihres Auftretens in der COMMON-Liste zugeordnet werden. Für die praktische Anwendung ist es daher empfehlenswert, jeweils alle Variablen bzw. Felder, die in den COMMON-Anweisungen zu vereinbaren sind, stets in sämtlichen Programmsegmenten anzugeben.

Die in einer COMMON-Liste aufgeführten Zahlenfelder können in dieser auch dimensioniert werden, so daß eine DIMENSION-Anweisung gespart wird. Somit lassen sich die Anweisungen

```
DIMENSION REDATA(10, 2), INDATA(20)
COMMON B, C, REDATA, R, INDATA
```

vereinfachen zu der Form

```
COMMON B, C, REDATA (10, 2), R, INDATA (20)
```

Es ist jedoch unzulässig, Felder in einer DIMENSION- und in einer COMMON-Anweisung zu dimensionieren. Den in einer einfachen COMMON-Anweisung spezifizierten Variablen und Feldern können keine Anfangswerte durch eine DATA-Anweisung zugeteilt werden.

Das Prinzip der Speicherplatzverteilung und der impliziten Parameterübergabe bei Verwendung der COMMON-Anweisung in Haupt- und Unterprogramm soll folgendes Beispiel verdeutlichen.

```
      SUBRØUTINE BETA
      CØMMØN C, D, E, JA, SA
      .
      .
      .
      E = ...
      JA = ...
      .
      .
      .
      RETURN
      END
C     HAUPTPRØGRAMM
      CØMMØN A, B, WERT, KA, S
      .
      .
      .
      CALL BETA
      .
      .
      .
      STØP
      END
```

Hierbei werden sämtliche Parameter, die das Unterprogramm BETA benötigt, im COMMON-Bereich übergeben. Die Rückgabe der berechneten Werte an das Hauptprogramm erfolgt ebenfalls über den COMMON. Folgende Variablen werden jeweils auf einem gemeinsamen Speicherplatz im COMMON-Bereich gespeichert:

A	und	C
B	und	D
WERT	und	E
KA	und	JA
S	und	SA

Selbst wenn die Variablen C und D im Unterprogramm nicht gebraucht würden, müssen sie dennoch in der COMMON-Anweisung aufgeführt werden, um eine ordnungsgemäße Speicherplatzverteilung zu gewährleisten. Würde man in der COMMON-Anweisung des Unterprogramms BETA die Variablen C und D weglassen, so erhielten folgende Variablen gemeinsame Speicherplätze:

A	und	E
B	und	JA
WERT	und	SA

An dieser Stelle sei noch erwähnt, daß natürlich, im Gegensatz zu oben angeführten Beispiel, in der COMMON-Anweisung des Haupt- und des Unterprogramms auch die gleichen Variablen stehen dürfen und in der Praxis auch meistens stehen.

Wie man aus dem obigen Beispiel weiterhin entnehmen kann, sollten Variablen, die den gleichen Speicherplatz belegen, vom gleichen Typ sein, da es sonst zu unerwünschten Nebeneffekten kommen kann. Bei Feldern ist darauf zu achten, daß für jedes Feldelement ein Speicherplatz reserviert wird. Die Reservierung erfolgt nach der FORTRAN-Konvention, wonach nach steigenden Indizes gespeichert wird und die links stehenden Indizes schneller wachsen als die rechts stehenden.

Bei der Übergabe von doppelt genauen und komplexen Variablen ist darauf zu achten, daß es sich hierbei um Zweiwortvariablen handelt, die demzufolge auch zwei aufeinanderfolgende Speicherplätze beanspruchen.

15.2 Der benannte COMMON-Block

Um die Handhabung des COMMON-Bereichs bei umfangreichen Programmen zu erleichtern, kann man diesen Bereich in Blöcke zerlegen, die dann auch gesondert verwendet werden können. Man kann damit in einem Unterprogramm nur diejenigen COMMON-Blöcke angeben, in denen die gerade benötigten Variablen und Felder zu finden sind. Diese Form der COMMON-Anweisung ist wie folgt definiert:

COMMON/name1/a1, a2, . . . , am/name2/b1, b2, . . . , bn/name3/. . . .

Die a1, . . . am und b1, . . . , bn sind Namen von Variablen und/oder Feldern, die nicht als Namen für formale Parameter verwendet werden. Diesen Namen können wahlweise die Indexlisten angehängt werden. Diese bestehen aus maximal drei vorzeichenlosen INTEGER-Konstanten, die voneinandern durch Kommata getrennt sind. Sie definieren die Maximalwerte der Feldindizes.

/name1/, . . . /namen/ sind wahlweise angebbare Namen von COMMON-Blöcken (benannte COMMON-Blöcke, labeled COMMON blocks). Der Name kann aus bis zu sechs Zeichen bestehen, von denen das erste ein Buchstabe sein muß. Die übrigen Zeichen dürfen Buchstaben oder Ziffern sein. Der Name muß immer von Schrägstrichen eingeschlossen sein.

Beispiele:

```
COMMON/BLOCKA/A1(5), B1, C1
COMMON/BLOCKD/DELT(5, 2), ECHO
COMMON/VECTOR/VECTOR(5), HECTOR, NECTOR
```

Da die Namen der benannten COMMON-Blöcke nur innerhalb des Compilers verwendet werden, dürfen sie gleichzeitig auch als Namen für Variablen und Felder verwendet werden (siehe 3. Zeile des obigen Beispiels), nicht jedoch als Unterprogrammnamen.

Die Angabe // (also ohne Zeichen zwischen den beiden Schrägstrichen, wobei Blanks jedoch zugelassen sind) bezeichnet einen unbenannten

COMMON-Bereich (blank COMMON). Bezeichnet name1 einen unbenannten COMMON, so können die beiden Schrägstriche auch weggelassen werden. Dies bedeutet, daß die im vorigen Abschnitt behandelte einfache COMMON-Anweisung nur ein Spezialfall, nämlich ein unbenannter COMMON-Block (unlabeled COMMON block) ist.

1.) Beispiel:

```
      CØMMØN A, B, C/BLØCK1/ETA(10), D, F, G//NØ, R, V
     *  /BLØCK2/H, J, S, T, WERT(2, 5)
```

Diese Anweisung definiert zuerst einen unbenannten COMMON, dem die Variablen A, B, C, NO, R und V angehören (die Variablen werden also vom Compiler miteinander verkettet). Dann folgt der benannte COMMON-Block /BLOCK1/ mit den Variablen ETA (10), D, F, G und der Block /BLOCK2/ mit den Variablen H, J, S, T und WERT(2, 5).

2.) Beispiel: Die beiden folgenden Anweisungen sind einander völlig gleichwertig.

```
CØMMØN//A, B, C, D, E, F
CØMMØN  A, B, C, D, E, F
```

COMMON-Anweisungen wirken kumulativ, d. h. alle Variablen- und Feldnamen, die in den verschiedenen Abschnitten einer COMMON-Anweisung und in verschiedenen COMMON-Anweisungen des gleichen Programms unter den gleichen Blocknamen geführt werden, bilden zusammen einen Block. Die Anweisungen

```
CØMMØN A, B, C/BLØCK1/ETA(10), D/BLØCK2/H, J, S
CØMMØN NØ, R, V/BLØCK1/F, G/BLØCK2/T, WERT(2, 5)
```

sind identisch mit der Anweisung

```
CØMMØN A, B, C, NØ, R, V/BLØCK1/ETA(10), D, F, G/BLØCK2/H, J, S, T,
*WERT(2, 5)
```

und auch identisch mit den Anweisungen

```
CØMMØN A, B, C, NØ, R, V
CØMMØN/BLØCK1/ETA(10), D, F, G
CØMMØN/BLØCK2/H, J, S, T, WERT(2, 5)
```

Die letztere Anordnung ist wegen ihrer besseren Übersichtlichkeit für die praktische Programmierung vorzuziehen.

Es ist auch zulässig, den gleichen Blocknamen mehrfach in einer COMMON-Anweisung zu verwenden. Dabei werden die Namen der Variablen und Felder der Blöcke mit gleichem Namen hintereinander gespeichert, als ob es nur ein Block wäre.

Die wesentlichen Unterschiede zwischen benanntem und unbenanntem COMMON sind:

1.) Es gibt nur einen einzigen unbenannten COMMON-Bereich in jedem Programm. Dagegen kann es beliebig viele benannte COMMON-Blöcke geben, von denen jeder einen bestimmten Namen hat.

2.) In allen Programmeinheiten, die einen bestimmten benannten COMMON-Block benutzen, muß dieser COMMON-Block mit der gleichen Länge definiert sein. Der unbenannte COMMON-Bereich kann in den verschiedenen Programmeinheiten unterschiedliche Länge haben.

3.) Den Variablen und Feldelementen eines unbenannten COMMON können keine Anfangswerte durch die DATA-Anweisung zugewiesen werden. Den Variablen und Feldelementen in einem benannten COMMON-Block dürfen durch die DATA-Anweisung Anfangswerte zugewiesen werden, jedoch nur in einem BLOCK-DATA-Subprogramm.

4.) Unbenannter COMMON und benannter COMMON werden völlig getrennt voneinander gespeichert. Der unbenannte COMMON am Ende des verfügbaren Kernspeicherplatzes, der benannte COMMON am Anfang der Programme.

16. Variable Dimension in Unterprogrammen

In vielen Fällen ist es erwünscht, die Dimensionen von Feldern variabel zu halten und erst beim aktuellen Aufruf des Unterprogramms zu bestimmen. Auf diese Weise können sich die Dimensionen formaler Felder von Aufruf zu Aufruf verändern.

Man erreicht dies dadurch, daß der Feldnamen und die Dimensionen des Feldes als formale Parameter in einer FUNCTION- oder SUBROUTINE-Anweisung spezifiziert werden. In der DIMENSION-Anweisung oder expliziten Typdeklaration des Unterprogramms müssen in den Klammern hinter dem Namen des Feldes anstelle der Maximalwerte für die Indizes Variable vom Typ INTEGER angegeben werden. Beim aktuellen Aufruf dürfen diese Werte natürlich höchstens so groß sein, wie im rufenden Programm bezüglich des aktuellen Feldes in einer DIMENSION-, COMMON- oder expliziten Typdeklaration festgelegt wurde und sie müssen vom Typ INTEGER sein.

Beispiel: Ein Unterprogramm zur Addition zweier reeller Rechtecksmatrizen.

```
      SUBROUTINE MATADD(X,Y,Z,M,N)
      DIMENSION X(M,N),Y(M,N),Z(M,N)
      DO 1 I=1,M
      DO 1 J=1,N
      Z(I,J) = X(I,J) + Y(I,J)
    1 CONTINUE
      RETURN
      END
```

Die Felder X, Y, Z und die variablen Dimensionen M und N erscheinen als formale Parameter in der SUBROUTINE- und in der DIMENSION-Anweisung. Der Aufruf dieses Unterprogramms in einem Hauptprogramm könnte folgendermaßen aussehen:

```
C     HAUPTPROGRAMM
      DIMENSION A(20,20),B(20,20),C(20,20)
      DIMENSION D(10,10),E(10,10),F(10,10)
        .
        .
        .
      CALL MATADD(A,B,C,20,20)
        .
        .
        .
      IP = 10
        .
        .
        .
      CALL MATADD(E,F,D,IP,IP)
        .
        .
        .
      STOP
      END
```

Die aktuellen Felder A, B, C, D, E und F müssen im Hauptprogramm mit festen Grenzen vereinbart sein. N kann irgendeine ganze Zahl zwischen 1 und 20 sein. Bei Benutzung des Feldes D darf sie natürlich nur bis 10 gehen. M gibt die Zahl der Zeilen der im Hauptprogramm definierten Matrix an.

Die INTEGER-Variablen, die in einem Unterprogramm die variable Dimension angeben, dürfen im Unterprogramm nicht verändert werden, d. h. sie dürfen im Unterprogramm keinesfalls auf der linken Seite eines Gleichheitszeichens auftreten.

Der Name eines Feldes mit variablen Dimensionen und die INTEGER-Variablen, die die variablen Dimensionen angeben, dürfen nicht in einer COMMON-Anweisung erscheinen.

Sind die Maximalwerte der Dimensionen im rufenden Programm und im gerufenen Unterprogramm gleich, so stimmt auch die Indizierung der Felder überein. Vorsicht ist jedoch geboten, wenn im Unterprogramm kleinere Maximalwerte verwendet werden. Zwar wird das Feld des rufenden Programms nicht verändert, aber die Speicherabbildungsfunktion des formalen Feldes im gerufenen Unterprogramm definiert eine andere Speicherlokalisierung, so daß die Indizes bei Formal- und Aktualfeld voneinander abweichen. In diesem Fall muß sich der Programmierer seine eigene Indexrechnung schaffen.

Beispiel: Ein Unterprogramm zu Nullsetzen eines rechteckigen Feldes vom Typ REAL.

```
      SUBRØUTINE ZERØ (X, L, M)
      DIMENSIØN X (L, M)
      DO  1  I=1, L
      DO  1  J=1, M
    1 X(I, J) = 0.0
      RETURN
      END
```

Aufruf:

```
C     HAUPTPRØGRAMM
      DIMENSION  Y(5, 5)
      .
      .
      .
      CALL ZERØ (Y, 2, 3)
      .
      .
      .
      STØP
      END
```

Die Anweisung DIMENSION Y (5, 5) definiert im rufenden Programm das zweidimensionale Feld Y mit absoluten Dimensionen. Beim Aufruf

der Subroutine ZERO entspricht dem formalen Feldnamen X der Feldname Y und die formalen Parameter L und M erhalten die aktuellen Werte 2 und 3. Die Zuordnung der Elemente des Feldes Y zu den Elementen des formalen Feldes X ist dann folgende:

```
Y(1,1) Y(2,1)  Y(3,1)  Y(4,1)  Y(5,1)  Y(1,2)  Y(2,2) ......
X(1,1) X(2,1)  X(1,2)  X(2,2)  X(1,3)  X(2,3)
```

Im rufenden Programm ist Y(1,2) das sechste Feldelement des Feldes Y. Beim Aufruf im Unterprogramm ZERO bezieht sich aber X(1,2) auf das dritte Feldelement des Feldes Y, d.h. X(1,2)=Y(3,1).

17. Die EXTERNAL-Anweisung

Die EXTERNAL-Anweisung ist eine nicht ausführbare Anweisung, sie hat die Form:

$$\text{EXTERNAL } a_1, a_2, \ldots\ldots, a_n$$

wobei jedes a_i der Name eines externen Unterprogrammes ist. Wenn der Name eines externen Unterprogrammes als aktueller Parameter für ein anderes externes Unterprogramm verwendet wird, muß er in der Programmeinheit, in der er so benutzt wird, in einer EXTERNAL-Anweisung erklärt werden.
Die EXTERNAL-Anweisung muß vor dem ersten ausführbaren Befehl in der rufenden Programmeinheit stehen.

Beispiel mit externen Standardfunktionen
rufende Programmeinheit

```
  .
  .
  .
EXTERNAL SIN, CØS
  .
  .
  .
CALL TRIGØN (A,SIN,Y)
  .
  .
CALL TRIGØN (A,CØS,Y)
  .
  .
END
```

Unterprogramm

```
SUBROUTINE TRIGØN (X,FUNKT,Y)
Y=FUNKT(X)
RETURN
END
```

Beide Funktionen, die für FUNKT eingesetzt werden, sind Bibliotheksfunktionen, die jedes FORTRAN-Programmiersystem zur Verfügung stellen muß (s.§11).

Beispiel mit selbstgeschriebenen externen Unterprogrammen

rufende Programmeinheit

```
  .
  .
  .
EXTERNAL MULT
CALL UNTER (MULT,X,Y,Z)
  .
  .
  .
END
```

Unterprogramm 1

```
SUBROUTINE MULT (A,B,C)
C=A*B
RETURN
END
```

Unterprogramm 2

```
SUBROUTINE UNTER (FUNK,Q,R,S)
CALL FUNK (Q,R,S)
RETURN
END
```

Das Unterprogramm namens MULT wird in der rufenden Programmeinheit als aktueller Parameter übergeben; daher muß MULT in der EXTERNAL-Anweisung stehen. Der Aufruf CALL FUNK (Q,R,S) im Unterprogramm 2 führt zu dem Ergebnis, daß Z der Wert X*Y zugewiesen wird.

18. Die DATA-Anweisung

Bei allen Programmen ist es notwendig, dem Programm beim Start die notwendigen Anfangswerte mitzuteilen. Dies kann auf drei verschiedene Arten geschehen:

1.) Man verwendet eine READ-Anweisung, wodurch die Werte, z. B. von einer oder mehreren Lochkarten, eingelesen und den entsprechenden Variablen zugewiesen werden. Diese Methode eignet sich besonders dann, wenn sich die einzulesenden Werte häufig ändern, wenn man also bei jedem Programmlauf Ausgangswerte benötigt.

2.) Indem man den Variablen zu Beginn des Programms direkt die Werte zuweist, z. B. durch eine Anweisung der Form PI=3.14152. Da man auf einer Lochkarte nur eine solche Anweisung unterbringen kann, erhält man, bei einer großen Anzahl von Variablen mit Anfangswerten rasch sehr große Lochkartenstapel.

3.) Die eleganteste Methode, Variablen zu Programmbeginn bestimmte Werte zuzuweisen -unabhängig davon, ob es sich um Konstante handelt oder nicht- bietet die DATA-Anweisung.

Die DATA-Anweisung gehört zu den nichtausführbaren Anweisungen und hat die allgemeine Form:

```
DATA k1/d1/,k2/d2/,.....,kn/dn/
```

Jedes k1,k2...,kn ist eine Liste von Variablen und / oder Feldelementen. Die Indizes der Feldelemente müssen vorzeichenlose INTEGER-Konstanten sein. Formale Parameter von Unterprogrammen dürfen in dieser Liste nicht aufgeführt werden. Die Variablen und / oder Feldelemente werden voneinander durch Komma getrennt.

Jedes d1,d2,...,dn ist eine Liste von Konstanten. Diese Konstanten können vom Typ INTEGER REAL, LOGICAL, DOUBLEPRECISION und COMPLEX sowie auch Hollerith-Konstanten sein. Diese Werte werden durch Schrägstriche von den zugehörigen Variablen getrennt. Jeder Konstanten kann ein Wiederholungsfaktor in der Form i* vorangestellt werden, der angibt, daß die Konstante i-mal wiederholt werden soll.

Die Zuweisung der Anfangswerte erfolgt während der Compilation, also nicht zur Laufzeit des Programms.

Beispiele:

```
DATALEDA, CASTØR, PØLLUX/15,16.0,84.0/
```

Der INTEGER-Variablen LEDA wird der Wert 15, den REAL-Variablen CASTOR und POLLUX werden die Werte 16.0 bzw. 84.0 zugewiesen.

```
DIMENSIØN GIB(5)
DATA GIB(1),GIB(2),GIB(3),GIB(4),GIB(5)/1.,2.,3*4.23/
```

Den ersten beiden Elementen des REAL-Feldes GIB werden die Werte 1. und 2. zugewiesen. Die restlichen drei Feldelemente erhalten mit Hilfe des Wiederholungsfaktors den Wert 4.23. Man beachte, daß alle Feldelemente einzeln aufgeführt werden müssen. Abkürzende Schreibweisen, wie z. B. nur Angabe des Feldnamens oder implizite DO-Notation sind nicht zulässig in Standard-FORTRAN. Bei fast allen FORTRAN-Compilern ist dies jedoch trotzdem möglich, z. B. auch beim TR 440. Selbstverständlich hätte man auch schreiben können:

```
DIMENSIØN GIB(5)
DATA GIB(1)/1.  /
DATA GIB(2)/2.  /
DATA GIB(3)/4.23/
DATA GIB(4)/4.23/
DATA GIB(5)/4.23/

DIMENSIØN A(3,3)
DATA A(1,3)/16.339/
```

Hierbei sollte man sich darüber im klaren sein, daß nur das Feldelement A(1,3) einen Anfangswert zugewiesen bekommt. Alle anderen Feldelemente sind undefiniert!

```
LØGICAL LØG(4)
CØMPLEXPRØTER(4)
DATAPRØTER(1),PRØTER(2),PRØTER(3),PRØTER(4)/
4*(1.,2)/
DATALØG(1),LØG(2),LØG(3),LØG(4)/2*.TRUE.,2*.FALSE./
```

Zu beachten ist hierbei, daß für das Feld PRØTER insgesamt 8 Werte angegeben werden müssen (Typ COMPLEX). Logische Größen können natürlich nur die Werte .TRUE. oder .FALSE. erhalten

```
DIMENSIØN MESSAG(6)
DATA MESSAG(1),MESSAG(2)/4HSTAT,4HEMEN/
DATA MESSAG(3),MESSAG(4)/4HT IS,4H INC/
DATA MESSAG(5),MESSAG(6)/4HOMPL,4HETE /
```

In dem Feld MESSAG wird eine Hollerithkonstante gespeichert, die zu irgendeinem Zeitpunkt mit A-FORMAT ausgedruckt werden könnte.

Zwischen den Variablen und den Feldelementen der Liste k und den Konstanten der Liste d muß eine 1:1 Korrespondenz bestehen. Die DATA-Anweisung kann im Programm an beliebiger Stelle zwischen der letzten Spezifikations- und Endanweisung auftrete Die Variablen und Feldelemente, denen durch die DATA-Anweisung Anfangswerte zugewiesen werden sollen, dürfen nicht in einer unbenannten COMMON-Anweisung aufgeführt sein. Variablen und Feldelemente, die in einem benannten COMMON-Block stehen, dürfe mittels einer DATA-Anweisung Anfangswerte nur in einem BLOCK-DATA-Subprogramm zugewiesen werden.

19. Die EQUIVALENCE-Anweisung

Beim Erstellen von sehr umfangreichen Programmen, die ohnehin einen hohen Speicherbedarf aufweisen, kommt es darauf an, diesen Speicherplatzbedarf auf ein Minimum zu reduzieren, um ein "Überlaufen" des Arbeitsspeichers zu vermeiden. Diese Reduzierung des Speicherbedarfs läßt sich z. B. dadurch erzielen, daß man Variablen und Feldern, die entweder gleiche Werte aufweisen oder nicht gleichzeitig innerhalb des Programms benötigt werden, gleiche Speicherplätze zuteilt.

Die EQUIVALENCE-Anweisung bietet diese Möglichkeit. Sie hat die allgemeine Form:

```
EQUIVALENCE (a1, a2, ....., am), (b1, b2, ...., bn), .....
```

Die a1,...,am und b1,...,bn sind Variablen und/oder Feldnamen, die nicht als formale Parameter verwendet werden. Die Folge dieser Anweisung ist, daß allen in einem Klammernpaar stehenden Variablen vom gleichen oder verschiedenen Typ derselbe Speicherplatz zugewiesen wird, d. h. die Variablen a1,...,am belegen gemeinsam einen Speicherplatz und die Variablen b1, ...,bn belegen gemeinsam einen Speicherplatz.

Die EQUIVALENCE-Anweisung ist eine nichtausführbare Anweisung. Sie muß vor allen ausführbaren Anweisungen, Anweisungsfunktionen und DATA-Anweisungen auftreten.

Beispiel:

```
EQUIVALENCE(ALFA, BETA, GAMMA), (X, Y, Z), (B1, B2, B3)
```

Durch diese Anweisung erhalten jeweils ALFA, BETA und GAMMA, X, Y und Z sowie B1, B2 und B3 die gleichen Speicherplätze. Es werden also nur drei Speicherplätze anstelle von neun belegt. Die Anweisung hätte natürlich auch geschrieben werden können als

```
EQUIVALENCE (ALFA, BETA, GAMMA)
EQUIVALENCE (X, Y, Z)
EQUIVALENCE (B1, B2, B3)
```

Weitaus wirksamer wird diese Maßnahme der Speicherplatzreduzierung bei der Verwendung von Feldern.

Beispiel:

```
DIMENSIØN A (10, 10), B(10, 10)
EQUIVALENCE (A(1, 1), B(1, 1) )
```

Diese Anweisung bewirkt, daß den Feldelementen A (1, 1) und B (1, 1) die gleichen Speicherplätze zugewiesen werden. Da jedoch die einzelnen Feldelemente linear hintereinander im Speicher stehen, ergibt sich auf Grund der obigen Anweisung, daß auch die folgenden Feldelemente, also A(2, 1) und B(2, 1), A(3, 1) und B(3, 1) usw., auf gemeinsamen Speicherplätzen gespeichert werden. Die gleiche Wirkung wäre daher auch erzielt worden mit:

```
EQUIVALENCE(A(10, 10), B(10, 10))
```

Die Indizes von Feldelementen können auf zweierlei Weise angegeben werden: entweder durch Angabe einer normalen Indexliste mit allen Indizes oder durch eine Zählung vom Feldanfang (z. B. ARRAY(1) als erste Variable, ARRAY(23) als 23. Variable, obgleich ARRAY ein mehrdimensionales Feld ist). Man kann also in einer EQUIVALENCE-Anweisung bei einem mit

```
DIMENSIØN A (I, J, K)   erklärten Feld
```

das Element A(i, j, k) mit $1 \le i \le I$; $1 \le j \le J$; $1 \le k \le K$

auch schreiben A (L) wobei gilt:

```
L = (i+I*( (j-1 )+J* (k-1 ) )*E
```

E ist 1 oder 2, je nachdem, ob es sich um ein Feld der Typen INTEGER, REAL oder LOGICAL handelt, die ein Speicherwort pro Feldelement belegen, oder ob der Typ COMPLEX oder DOUBLE ist, die zwei Speicherworte pro Feldelement belegen. Man hätte bei dem vorherigen Beispiel auch schreiben können

```
EQUIVALENCE (A (1), B(1))
```

oder `EQUIVALENCE (A (100), B(100) )`

oder `EQUIVALENCE (A (11), B (1, 2) )`

Einige Vorsicht ist geboten, wenn man die EQUIVALENCE-Anweisung auf Felder unterschiedlicher Länge anwendet.

Beispiel:

```
DIMENSIØN A (6), B(4), C(2, 3)
EQUIVALENCE (A (3), B(2), C(1, 2))
```

Die Wirkung ist die, daß neben der Zuteilung eines gemeinsamen Speicherplatzes für die Feldelemente A (3), B (2) und C (1, 2) auch die übrigen Feldelemente bestimmte gemeinsame Speicherplätze zugewiesen bekommen und zwar in der Form, wie es die folgende Tabelle aufzeigt:

```
A (1)    A (2)   A(3)   A (4)   A (5)   A (6)
         B (1)   B(2)   B (3)   B (4)
C (1, 1) C (2, 1) C (1, 2) C (2, 2) C (1, 3) C (2, 3)
```

Zwei Variable, die im gleichen oder verschiedenen COMMON-Blöcken stehen, können nicht mittels EQUIVALENCE einander gleich gesetzt werden. Andererseits ist es erlaubt, einer nicht in einem COMMON-Block stehenden Größe und einer in einem COMMON-Block stehenden mittels EQUIVALENCE den gleichen Platz anzuweisen. Dadurch kann die Länge des COMMON-Blocks nach rechts vergrößert werden, dies ist erlaubt.

Beispiel:

```
DIMENSIØN B (5)
CØMMØN  A (4)
EQUIVALENCE (A (3), B(2) )
```

Ohne die EQUIVALENCE-Anweisung hätte dieser unbenannte COMMON die Länge 4. Die EQUIVALENCE-Anweisung bewirkt aber folgende Zuordnung:

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

d. h. die Länge des unbenannten COMMON erhöht sich auf 6 Speicherplätze. Eine Verlängerung des COMMON nach links, d. h. über seinen Anfang hinaus ist jedoch nicht erlaubt. Folgendes Beispiel ist unzulässig.

Beispiel:

```
DIMENSIØN B (5)
CØMMØN  A (4)
EQUIVALENCE (A(2), B(5) )
```

Dadurch wird versucht, folgende Anordnung zu erreichen:

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

mithin eine Verlängerung des unbenannten COMMON-Blocks nach vorne.

20. Das BLOCK-DATA -Unterprogramm

Um Variablen, die in einem benannten COMMON-Block stehen, Anfangswerte zuzuweisen, muß ein BLOCK DATA-Unterprogramm benutzt werden. Eine Initialisierung von Variablen im unbenannten COMMON (Blank-COMMON) ist nicht möglich.

Ein Block DATA-Unterprogramm beginnt mit der Anweisung

```
BLOCK DATA
```

und enthält, außer der abschließenden END-Anweisung, nur Typspezifikations-, DIMENSION-, EQUIVALENCE-, COMMON- und DATA-Anweisungen, d.h. es kommen nur nichtausführbare Anweisungen vor.

Wenn einem Teil eines (benannten) COMMON-Bereichs Anfangswerte zugewiesen werden, muß ein vollständiger Satz von Spezifikationsanweisungen für alle Variable dieses Bereichs im BLOCK DATA-Unterprogramm enthalten sein, obwohl nur einige Variable des betreffenden COMMON-Blocks in DATA-Anweisungen auftreten. Insbesondere müssen, wie allgemein bei benannten COMMON-Blöcken erforderlich, sämtliche Elemente eines solchen Blockes in der zugehörigen COMMON-Anweisung aufgeführt werden.

In einem BLOCK DATA-Unterprogramm können Variablen aus mehreren benannten COMMON-Bereichen Anfangswerte gegeben werden.

Beispiel für ein BLOCK DATA-Unterprogramm:

```
      BLØCK DATA
      DIMENSIØN A (21,3), B (20)
      CØMPLEX C1, C2
      CØMMØN /ROMEØ / A,B,C1,C2 /JULIA/ D,E,I (10)
      DATA C1,C2/(1.0,-2E-8),(0.0007,+.3)/
     *     D/-123.4/,I(2),I(3),I(6),I(4)/3*0,987/
      END
```

21. Formatfreies Lesen und Schreiben

Beim Lesen und Schreiben wird unterschieden zwischen formatierter Übertragung (also unter Angabe einer FORMAT-Anweisung) und unformatierter. Kartenleser, Schnelldrucker und Kartenstanzer lassen nur formatiertes Übertragen zu.
Dagegen sind bei Magnetband- und/oder Plattenbenutzung beide Übertragungsmodi möglich.

Die formatfreie Schreibanweisung

Die allgemeine Form lautet

WRITE (i)k

wobei k = Ausgabe-Liste

i = natürliche Zahl, die das Ausgabemedium bezeichnet

Bei der Ausführung dieser Anweisung wird ein neuer Satz (record) aus Werten, die der Ausgabe-Liste entsprechen, erzeugt.

Die formatfreie Leseanweisung

Die allgemeine Form lautet

READ (i)k oder READ (i)

wobei k = Eingabe-Liste

i = natürliche Zahl, die das Eingabemedium bezeichnet.

Die Ausführung dieser READ-Anweisung veranlaßt die Eingabe des nächsten Satzes vom Eingabemedium her. Bei vorhandener Liste darf die geforderte Folge von Werten nicht die Folge von Werten aus dem formatfreien Satz überschreiten.

Ein formatfreier Satz besteht aus einer Kette von Werten. Wird ein formatfreier oder formatgebundener Lesebefehl ausgeführt, müssen die geforderten Sätze auf der angegebenen Eingabeeinheit formatfrei bzw. formatgebunden vorhanden sein.

Das formatfreie Lesen und Schreiben findet vornehmlich Verwendung bei der Zwischenspeicherung großer Datenmengen auf Magnetband oder Platten.

Beispiel

```
:
DIMENSIØN KUR(50, 50), B(700)
:
WRITE (21) ((KUR (I, K), K=1, 50), I=1, 20), (B(IB), IB=1, 500)
:
REWIND 21
:
READ (21) ((KUR(J, L), L=1, 50), J=1, 20), (B(IR), IR=1, 300, 2)
:
```

Der Befehl REWIND 21 veranlaßt, daß die durch 21 definierte Einheit auf den Anfangspunkt zurückgesetzt wird. So kann danach von Anfang an gelesen werden.

Ein Abbruch des Programmlaufes würde durch folgenden Lesebefehl bewirkt werden:

```
READ (21) ((KUR(J, L), L=1, 50), J=1, 20), (B(IR), IR=1, 700)
```

Hier wird versucht, aus dem Feld B mehr zu lesen als geschrieben wurde.

22. File-Handling

Daten, die zu einer Datei zusammengefaßt sind, werden im Englischen als File bezeichnet. Solche Files werden vorübergehend oder für die Dauer auf externen Speichermedien (Magnetband, Platte oder Trommel) gespeichert und vom Programm verarbeitet. Dazu werden Anweisungen benötigt, die das Aufsuchen von gespeicherten Informationen erleichtern.

22.1 Die BACKSPACE-Anweisung

Die allgemeine Form dieser Anweisung lautet

```
BACKSPACE i
```

Hierbei ist i eine ganzzahlige Konstante ohne Vorzeichen oder eine einfache INTEGER-Variable, die die Datei (File, Kanalnummer) kennzeichnet. Die BACKSPACE-Anweisung verursacht die Rücksetzung der Datei um einen Satz. Möchte man die Datei um mehrere Sätze zurücksetzen, so muß man die Anweisung BACKSPACE mehrfach hintereinander angeben oder in einer DO-Schleife unterbringen.

Beispiel:

```
      .
      .
      .
   DØ 26 I=1,ISATZ
   BACKSPACE ICLTAP
26 CØNTINUE
      .
      .
      .
```

In der Schleife wird die Datei um soviele Sätze zurückgesetzt, wie die INTEGER-Variable ISATZ angibt.

Wird beim Backspacing der Anfang der Datei erreicht, so bleiben weitere BACKSPACE-Anweisungen wirkungslos.

22.2 Die ENDFILE-Anweisung

Die allgemeine Form dieser Anweisung lautet

```
ENDFILE i
```

Hierbei ist i eine ganzzahlige Konstante ohne Vorzeichen oder eine einfache INTEGER-Variable, die die Datei (File, Kanalnummer) kennzeichnet. Die Anweisung bewirkt, daß das Ende der Datei i mit einer besonderen Kennzeichnung versehen wird. Damit wird verhindert, daß über das Ende der Datei hinaus gelesen werden kann.

Beispiel:

```
      .
      .
      .
WRITE(ITAPE) INZAHL,(REDATA(L),L=1,INZAHL)
      .
      .
      .
ENDFILE ITAPE
REWIND  ITAPE
      .
      .
      .
READ(ITAPE)  INZAHL,(REDATA(L),L=1,INZAHL)
      .
      .
      .
```

22.3 Die REWIND-Anweisung

Die allgemeine Form dieser Anweisung lautet

```
REWIND i
```

Hierbei ist i eine ganzzahlige Konstante ohne Vorzeichen oder eine einfache INTEGER-Variable, die die Datei (File, Kanalnummer) kennzeichnet. Die REWIND-Anweisung bewirkt, das die durch i gekennzeichnete Datei auf ihren Anfang gesetzt wird, z.B. vollständiges Zurückspulen eines Magnetbandes.

Beispiele

```
      .
      .
      .
REWIND 2
      .
      .
      .
REWIND ITAPE
      .
      .
      .
```

23. Die DEFINE FILE-Anweisung

Die Anweisung DEFINE FILE beschreibt Anzahl, Größe und Struktur der Datensätze bei Ein- und Ausgabe von bzw. auf Dateien mit direktem Zugriff (siehe READ . und WRITE-Anweisung).
Die allgemeine Form der Anweisung lautet:

```
DEFINE FILE n (m,l,s,v)
```

Dabei bedeutet:

n: (INTEGER-Konstante oder INTEGER-Variable): Kanalnummer

m: (INTEGER-Konstante oder INTEGER-Variable): Satzparameter, Angabe der Sätze (1 Satz entspricht dem Inhalt einer Karte oder Zeile bei formatierte Übertragung), die maximal zur Datei gehören.

l: (INTEGER-Konstante oder INTEGER-Variable): Längenparameter; dieser Parameter legt die maximale Länge eines Satzes fest. Es gilt im Zusammenhang mit dem Parameter s für l: (siehe unten)

Form von s	der Wert von l legt fest
E	Zeichenzahl
U	Zahl der Speichereinheiten
W	1/4 x (Zahl der Speichereinheiten + 3)
L	Zahl der Speichereinheiten
M	1/4 x (Zahl der Speichereinheiten + 3)

S: (Einer der Buchstaben E,U,W,L oder M): Übertragungsscglüssel

Es gilt:	Übertragungsschlüssel	Bedeutung
	E	Zeichen (Oktade Byte)
	U	Viertelwort
	W	Ganzwort
	L	Viertelwort oder Zeichen
	M	Ganzwort oder Zeichen

V: (INTEGER-Variable): Assoziierte Variable

Diese Größe muß eine Variable sein. Ihr wird nach einer READ- oder WRITE-Anweisung mit bezug auf die Datei die um Eins erhöhte Nummer des zuletzt übertragenen Datensatzes zugewiesen.

Bemerkungen: 1.) Die Leistungen dieser Anweisung lassen sich auch durch ein Kommando an das Betriebssystem erbringen (beim TR 440 z. B. durch das DATEI-Kommando). Der Vorteil der Verwendung von DEFINE FILE liegt darin, daß Größe und Satzbau einer Datei von einem FORTRAN-Programm aus festgelegt werden.

2.) In READ- und WRITE-Anweisungen kann die assoziierte Variable ebenfalls auftreten, sogar in Form einer Konstanten:

z.B. READ (8'I,3) Liste
WRITE(9'6,5) Liste

3 Format (...)
5 Format (...)

Beim Lesen wird mit dem I-ten Satz der Datei Nr. 8 begonnen. Die assoziierte Variable der DEFINE FILE-Anweisung erhält dann den Wert I+1.

Beim Schreiben wird in den 6. Satz der Datei Nr. 9 geschrieben, die assoziierte Variable erhält den Wert 7.
Fehlt der Parameter so gilt:

bei READ: Es wird mit dem nächsten Datensatz zu lesen begonnen.

bei WRITE: Die Datei wird sequentiell bearbeitet.

3.) Mehrere Dateien können mit einer Anweisung erklärt werden

Beispiel: a) DEFINE FILE 25(100,80,E,IZAHEL) (maximal 100 Sätze a 80 Zeichen, d.h. Lochkartenbilder)

b) DEFINE FILE 30(50,133,E,I1),40(100,512,W,I2) (Beispiel zu Bemerkung 3.), 2 Dateien werden erklärt)

24. FORTRAN-Literatur

1.) American National Standards
Norm X3.9-1966, March 7.

2.) C. Berg
Programmieren mit FROTRAN
physica-verlag-Würzburg, Wien - 1972

3.) H. Breuer
FORTRAN - Fibel, BI-TB 204, 1969

4.) Colmann, H.L./ Smallwood, C.
FORTRAN - Problemorientierte Programmiersprache
Verlag: Kunst und Wissen, 1969

5.) Dörband, W.
Praxis der FORTRAN - Programmierung
Verlag: Technik, 1969

6.) Dreyfuss, M.
Anleitung zum praktischen Gebrauch von FORTRAN IV
Verlag: Oldenbourg, 1970

7.) F. Fischbach/K.-H. Thull
Problemorientiert mit FORTRAN
Verlagsgesellschaft Rudolf Müller
Köln - Braunsfeld -1973

8.) Gritsch, R.
Das Programmieren von Computern
(... unter Verwendung von FORTRAN)
Verlag: Hanser, 1972

9.) Guttropf, W./Stricker, U.
FORTRAN mit Pfiff
Verlag: Krausskopf, 1972

10.) Klein, G.
Einführung in die Programmiersprache FORTRAN IV
Verlag: AEG, 1969

11.) Lamprecht, G.
Einführung in die Programmiersprache FORTRAN IV
Verlag: Vieweg, 1970

12.) Lee, R.M.
Basic FORTRAN IV, eine Einführung in die Programmierung
Verlag: Kunst und Wissen, 1970

13.) Lehmann, F./Schmidt,W./Vollmer,K.
Programmierung in FORTRAN
Verlag: Exit, 1967

14.) Mc Cracken, D.D.
FORTRAN in der technischen Anwendung
Verlag: Hanser, 1970

15.) Müller, K.H./Streker, I.
FORTRAN IV, Programmierungsanleitung
Verlag: BI-Hochschulskripten 1970

16.) Paulin, G.
FORTRAN
Verlag: Vieweg, 1969

17.) Rehbein, H.
FORTRAN IV - leicht gemacht
Verlag: VDI, 1971

18.) Spiess, W. e./Rheingans, F. G.
Einführung in das Programmieren in FORTRAN
Verlag: de Gruyter, 1971

19.) Stempell, D.
Programmierte Einführung in FORTRAN
Verlag: Westdeutscher Verlag, 1970

S A C H V E R Z E I C H N I S

A-SPEZIFIKATION 52
ABLOCHSCHEMA 9
ABS-FUNKTION 36
ADDIERWERK 5
ADDITION 15
AIMAG-FUNKTION 36
AINT-FUNKTION 36
AKTUELLER PARAMETER 80,82,85
ALOG-FUNKTION 35
ALOG10-FUNKTION 35
ALPHANUMERICHE ZEICHEN 11
ALPHANUMERISCHE NAMEN 11
AMAX0-FUNKTION 36
AMAX1-FUNKTION 36
AMIN0-FUNKTION 36
AMIN1-FUNKTION 36
AMOD-FUNKTION 35
ANFANGSWERTE 101
-DURCH DATA ANWEISUNG 101
ANORDNUNG VON FELDELEMENTEN 38
ANS-NORM 1,115
ANWEISUNGEN 8
-AUSFUEHRBARE 8
-NICHT AUSFUEHRBARE 8
ANWEISUNGSFUNKTION 80
ANWEISUNGSNUMMER 10
ARITHMETISCHE ANWEISUNG 15,19
-TYPKOMBINATIONEN 20
-TYPENUMWANDLUNGEN 20
ARITHMETISCHER AUSDRUCK 15
-AUSWERTUNG VON 17
-FORMEN 15
-RANGORDNUNG DER AUSWERTUNG 17
-REGELN FUER DIE BILDUNG 16
-TYPKOMBINATIONEN 16
ARITHMETISCHE IF-ANWEISUNG 30
ARITHMETISCHE OPERATOREN 15
ARRAY 33,37
-SPEICHERPLATZRESERVIERUNG 37,38,39
-STRUKTUR 38
-UEBERTRAGUNG 61
-VARIABLE LAENGE 95
ASSIGN-ANWEISUNG 76
ASSIGNED GOTO 76
ATAN-FUNKTION 35
ATAN2-FUNKTION 35
AUFBAU EINER RECHENANLAGE 3
AUFBAU EINES FORTRAN-PROGRAMMS 8
AUFRUF EINER FUNCTION 81,82
-EINER SUBROUTINE 84
AUSGABEGERAETE 3,4

AUSDRUECKE 15,69
-ARITHMETISCHE 15
-AUSWERTUNG VON 17,72
-GEMISCHTEN TYPS 17,72
-LOGISCHE 69
-VERGLEICHS- 69
AUSGABEBEFEHLE 22,49,61,109
AUSWERTUNG
-VON ARITHMETISCHEN AUSDRUECKEN 17
-VON KLAMMERN 18,72
-VON LOGISCHEN AUSDRUECKEN 72

BACKSPACE-ANWEISUNG 111
BANDBENUTZUNG 111
BENANNTER COMMON 92
BETRIEBSSYSTEM 7
BIBLIOTHEKSROUTINEN 7,35
BLANK COMMON 89,93
BLOCK DATA SUBPROGRAMM 107

CABS-FUNKTION 36
CALL 84
CCOS-FUNKTION 35
CEXP-FUNKTION 35
CHARACTERS 10
CLOG-FUNKTION 35
CMPLX-FUNKTION 36
CODIERUNG VON FORTRAN-ANWEISUNGEN 8
COMMON 89
-BLANK 89,93
-BENANNTER 92
-LABELLED 92
-MIT DATA 94,107
-UNBENANNTER 89,93
COMPILER 7,10
COMPLEX 33
-EIN/AUSGABE 52
-KONSTANTE 11
-STANDARDFUNKTIONEN 35
-VARIABLE 11,33
COMPUTED GOTO 75
CONJG-FUNKTION 36
CONTINUE-ANWEISUNG 48
CONTROL-UNIT 4
COS-FUNKTION 35
CSIN-FUNKTION 35
CSQRT-FUNKTION 35

D-SPEZIFIKATION 51
DABS-FUNKTION 36
DARSTELLUNG VON GROESSEN 11
DATA-ANWEISUNG 101
-MIT COMMON 107
DATAN-FUNKTION 35
DATAN2-FUNKTION 35
DATEN, ABLOCHUNG 23,24,25,26
DATENTRAEGER 3

DEFINE FILE-Anweisung 113
DRLE-FUNKTION 36
DCOS-FUNKTION 35
DEXP-FUNKTION 35
DEZIMALEXPONENT 12
DEZIMALPUNKTKONSTANE 12
DIM-FUNKTION 36
DIMENSION-ANWEISUNG 39
-FOLGEKARTEN 39
-MIT DATA 101,102
DIMENSIONIERUNG 33,39
-VARIABLE 95
DIVISION 15
DLOG-FUNKTION 35
DLOG10-FUNKTION 35
DMAX1-FUNKTION 36
DMIN1-FUNKTION 36
DMOD-FUNKTION 35
DO-ANWEISUNG 41
-LAUFVARIABLE 41
-ANFANGSPARAMETER 41
-ENDPARAMETER 41
-SCHRITTWEITENPARAMETER 41
DO-SCHLEIFE 42
-AUSFUEHRUNG 42
-ENDE 45
-IMPLIZITE 62
-REGELN FUER DIE PROGRAMMIERUNG 45
-SCHACHTELUNG 46
-SPRUENGE 46
-WIEDERHOLUNGSBEREICH 45
DOUBLE PRECISION 33
-EIN/AUSGABE 51
-KONSTANTE 11
-STANDARDFUNKTIONEN 35
-VARIABLE 33
DRUCKEN AUF ZEILENDRUCKER 22
-VORSCHUBSTEUERZEICHEN 55
DSIGN-FUNKTION 36
DSQRT-FUNKTION 35

E-SPEZIFIKATION 49
EINGABEBEFEHLE 21,49,56,109
EINGABEGERAETE 3
EIN-/AUSGABELISTE 61
EIN/AUSGABEANWEISUNGEN 8,21,49
-EINFUEHRUNG 21
EIN/AUSGABEGERAETE 3,4
END-ANWEISUNG 21
ENDE EINES PROGRAMMS 21
-LOGISCHES 21
-PHYSIKALISCHES 21
ENDFILE-ANWEISUNG 111
EQUIVALENCE-ANWEISUNG 103
-MIT COMMON 105
ERGIBTANWEISUNGEN 8
-ARITHMETISCHE 8
-LOGISCHE 8
EXP-FUNKTION 35

EXPONENT 12
EXPONENTIATION 15
-TYP DES EXPONENTEN 17
EXTERNAL 99
EXTERNE SPEICHER 3

F-SPEZIFIKATION 25
FELD 37
-DIMENSIONEN 37
-ANORDNUNG DER FELDELEMENTE 38
FILE 111
-HANDLING 111
FLOAT-FUNKTION 36
FORMALER PARAMETER 80,81,84
FORMAT 22,49
-VARIABLES 66
FORMATFREIES
-LESEN 109
-SCHREIBEN 109
FORMAT-ANWEISUNG 8,22,49
-ALLGEMEINER AUFBAU 22
FORMATIERTE EIN/AUSGABE 21,22,49
FORTRAN 1
-AUFBAU EINES PROGRAMMS 8
-CODIERUNG 8
-GRUNDELEMENTE 10
-SATZ 55
FORMAT-SPEZIFIKATION
-A 52
-D 51
-E 49
-F 25
-G 51
-H 27
-I 23
-P 64
-X 54
FUNKTIONEN 35,36
-BIBLIOTHEKS- 35,36
-STANDARD 35,36
FORTRAN-ANWEISUNGEN 8
-CODIERUNG 8
FORTRAN-LITERATUR 115
FORTSETZUNGSKARTEN 10

G-SPEZIFIKATION 51
GOTO-ANWEISUNG 29,75,76
-ASSIGNED 76
-COMPUTED 75
-UNBEDINGTES 29
GROESSEN 11
-DARSTELLUNG VON 11
-DOPPELTGENAUE 11
-GANZZAHLIGE 11,12
-KOMPLEXE 11
-LOGISCHE 11
-REELLE 11,12
GRUNDELEMENTE VON FORTRAN 10

H-SPEZIFIKATION 27
HOLLERITHZEICHEN 27
-EIN/AUSGABE 27

I-SPEZIFIKATION 23
IABS-FUNKTION 36
IDENTIFIER 11
IDIM-FUNKTION 36
IDINT-FUNKTION 36
IF-ANWEISUNG 30
-ARITHMETISCHE 30
-LOGISCHE 78
IFIX-FUNKTION 36
IMPLIZITE DO-SCHLEIFE 62
INDIZES 37
-VON ARRAYELEMENTEN 37
INDIZIERTE VARIABLEN 37
-BENUTZUNG 37
INT-FUNKTION 36
INTEGER 12
-DIVISION 18
-EIN/AUSGABE 23
-KONSTANTE 12
-STANDARDFUNKTIONEN 35
-VARIABLE 13
INSTRUKTIONEN 6
INTERNER SPEICHER 3,6
ISIGN-FUNKTION 36

JOB 7

KLAMMERN 18
-AUSWERTUNG VON 18
-IM FORMAT 58
KOMMENTARE 10
KOMMENTAR-KARTEN 10
KOMPLEX SIEHE COMPLEX
KONSTANTE 11
-GANZZAHLIGE 12
-REELLE 12
KONTROLLVARIABLE 41,75
-BEI DO-LOOPS 41
-BEI COMPUTED GOTO 75

L-SPEZIFIKATIC 55
LABEL 10
LABELLED COMMON 92
-BLOCK DATA 107
LAUFVARIABLE 41
LEERSTELLEN 10
LEITWERK 3,4
LESEN VON FILES 109,111
LESEN VON KARTEN 22,49
LISTE 22,33,39,51,89
-EINES EIN/AUSGABEBEFEHLS 22,61
LOADER 7

LOGICAL 33
-EIN/AUSGABE 55
-KONSTANTE 11
-VARIABLE 11,13
LOGISCHE ANWEISUNG 73
LOGISCHES IF 78

MASCHINENSPRACHE 6
MAX0-FUNKTION 36
MAX1-FUNKTION 36
MIN0-FUNKTION 36
MIN1-FUNKTION 36
MOD-FUNKTION 35
MULTIPLIKATION 15

NAMEN 8
NEST VON DO-LOOPS 46

OBJEKTPROGRAMM 6
OKTALZAHL 21,113
OPERANDEN 25,69
-ARITHMETISCHE 15
-LOGISCHE 69
OPERATOREN 15,69
-ARITHMETISCHE 15
-LOGISCHE 69
-VERGLEICHS- 69

P-SPEZIFIKATION 64
PARAMETER
-AKTUELLE 80,82,85
-FORMALE 80,81,84

PROGRAMMABLAUF 29
PROGRAMMENDE 21
-LOGISCHES 21
-PHYSIKALISCHES 21

QUELLENPROGRAMM 6,8

READ-ANWEISUNG 22
-FORMAT-GESTEUERTE 22
REAL 12,13
-EIN/AUSGABE 25,49
-FUNKTION 35
-KONSTANTE 12
-STANDARDFUNKTIONEN 35
-VARIABLE 12
RECHENANLAGE 3
-AUFBAU EINER 3
RECHENSTEUERUNG 5
RECHENWERK 3,5
RECORD 56
-NEUER 56,58
REGISTER 5
RETURN-ANWEISUNG 83,84
REWIND-ANWEISUNG 112
RUECKSPRUNG-ANWEISUNG 82

SCHACHTELUNG 46
-VON DO-LOOPS 46,47
SCHLEIFEN 41
SCHNELLDRUCKER 7
SCHRAEGSTRICH 56
-IM AUSGABEFORMAT 57
-IM EINGABEFORMAT 56
SCHREIBBEFEHLE 21,49,109
SCHRIFTZEICHEN 10
SEITENVORSCHUB 56
SIGN-FUNKTION 36
SIGNIFIKANTE STELLEN 12
SIN-FUNKTION 35
SKALENFAKTOR P 64
SLASH 10,56
-IM EINGABEFORMAT 56
-IM AUSGABEFORMAT 57
SNGL-FUNKTION 36
SONDERZEICHEN 10
SOURCE-PROGRAMM 6
SPEICHER 3
-EXTERNER 3
-INTERNER 3
SPEZIFIKATIONSANWEISUNGEN 8
SPRUENGE 45
-BEI DO-LOOPS 46
SPRUNGBEFEHLE 29,75
-BEDINGTE 75
-UNBEDINGTE 29
SQRT-FUNKTION 35
STANDARD-FUNKTIONEN 35
STATEMENT 8
STATEMENT-FUNCTION 80
STATEMENT-NUMMER 10,23
STEUERUNGS-ANWEISUNGEN 8,29,75
STEUERKARTEN 7
STEUERWERK 3
STOP-ANWEISUNG 21
SUBROUTINE 84
-AUFRUF 84
-BLOCK DATA 107
-RUECKSPRUNG 85
SUBTRAKTION 15

TANH-FUNKTION 35
TYPE 33,61
TYPENUMWANDLUNG 17
TYP DES EXPONENTEN 17
TYPDEKLARATIONEN 33
-IMPLIZITE 13,33
-EXPLIZITE 33

UEBERTRAGEN VON ARRAYS 61
UNBEDINGTER SPRUNG 29
UNFORMATTIERTE EIN/AUSGABE 109
UNTERPROGRAMM 79FF

VARIABLE 11,13
-EINFACHE 13
-INDIZIERTE 13,37
VARIABLE DIMENSIONIERUNG 95
VARIABLES FORMAT 66
VERGLEICHSAUSDRUECKE 69
VORSCHUBSTEUERUNG 55
VORSCHUBSTEUERZEICHEN 55

WERTZUWEISUNG 19,73
-ARITHMETISCHE 19
-LOGISCHE 73
WIEDERHOLUNG EINES FORMATS 56
WIEDERHOLUNGSFAKTOR 23
WRITE-ANWEISUNG 22
-FORMAT-GESTEUERTE 22

X-SPEZIFIKATION 54

ZEICHEN 10
-ALPHABETISCHE 10
-ALPHANUMERISCHE 11
-NUMERISCHE 10
-SONDER- 10
ZEICHENSATZ FUER FORTRAN 10,11
ZENTRALEINHEIT 3,4

A N H A N G 1

;ätzliche Möglichkeiten in FORTRAN auf dem TR 440

) Bei der Zahldarstellung besteht eine enge Anlehnung an IBM 360-bzw. IBM 370- Fortran.
Implicit belegt eine INTEGER-Konstante 4 Speichereinheiten (1 Speichereinheit = 12 Bit, 4 Speichereinheiten = 1 Wort), ist also von der Art INTEGER * 4 (s. unten).
Eine INTEGER-Konstante muß betragsmäßig kleiner als 2^{46}=70368744177664 sein.

Implicit belegt eine REAL-Konstante 4 Speichereinheiten, ist also von der Art REAL * 4 (s.unten).REAL Konstanten r sind wie folgt beschränkt: $10^{-154} < |r| < 10^{+152}$ und r = 0.0

Implicit belegt eine COMPLEX-Konstante 8 Speichereinheiten (zwei aufeinanderfolgende Worte), ist also von der Art COMPLEX * 8 (s.unten)

Implicit belegt eine LOGICAL-Konstante 4 Speichereinheiten, ist also von der Form LOGICAL * 4.

Wie auch im IBM-Fortran sind explicit die Speichergrößen für eine Konstante angebbar. Wird nichts besonderes geschrieben (also implicite Typerklärung oder nur INTEGER, REAL, COMPLEX, LOGICAL etc.), dann gelten obige Aussagen.

Man kann aber auch wie folgt deklarieren:
REAL * 8 oder COMPLEX * 16 und hat dann REAL- bzw. COMPLEX-Größen doppelter Genauigkeit, die dann Jeweils 2 bzw. 4 Worte belegen (8 bzw. 16 Speichereinheiten). Man kann z.B. auch LOGICAL * 1 oder INTEGER * 2 schreiben. REAL * 8 ersetzt DOUBLE PRECISION, letzteres darf natürlich auch benutzt werden und entspricht dann REAL * 8.
Literalkonstante können auf zweierlei Art geschrieben werden:

Beispiel: a) 4HTEXT
b) 'TEXT'

Im Falle b) darf das Zeichen' in der Konstanten nicht auftreten. Jedes Zeichen belegt eine Speichereinheit á 12 Bit, d. h. 4 Zeichen passen in 1 Wort.
Daher ist bei Lesen im A-Format in eine Variable einfacher Länge höchstens das Format A4 erlaubt.

2.) Ein Variablenname darf außer mit Buchstaben auch mit dem Zeichen beginnen. (Es ist aber nicht zu empfehlen).

3.) Ist der Index bei Feldern nicht von der Art INTEGER * 2 oder INTEGER * 4, so wird er durch Abschneiden ganzzahlig gemacht. Konstanten vom Typ COMPLEX sind als Indices nicht erlaubt.

4.) Für die arithmetischen Operatoren gelten die üblichen Vorrangrege Bei der Verknüpfung von Größen verschiedenen Typs gelten die folgenden Tabellen. Es steht stets der Typ des Resultats oder ei für verbotene Kombination.

(1) Operator: **

Typ des linken Operanden \ Typ des rechten Operanden	INTEGER*2	INTEGER*4	REAL*4	REAL*8	COMPLEX*8	COMPLEX*16
INTEGER*2	INTEGER*4	INTEGER*4	REAL*4	REAL*8	V	V
INTEGER*4	INTEGER*4	INTEGER*4	REAL*4	REAL*8	V	V
REAL*4	REAL*4	REAL*4	REAL*4	REAL*8	V	V
REAL*8	REAL*8	REAL*8	REAL*8	REAL*8	V	V
COMPLEX*8	COMPLEX*8	COMPLEX*8	V	V	V	V
COMPLEX*16	COMPLEX*16	COMPLEX*16	V	V	V	V

(2) Operator: +, -, *, /

Typ des linken Operanden \ Typ des rechten Operanden	INTEGER*2	INTEGER*4	REAL*4	REAL*8	COMPLEX*8	COMPLEX*16
INTEGER*2	INTEGER*2	INTEGER*4	REAL*4	REAL*8	COMPLEX*8	COMPLEX*16
INTEGER*4	INTEGER*4	INTEGER*4	REAL*4	REAL*8	COMPLEX*8	COMPLEX*16
REAL*4	REAL*4	REAL*4	REAL*4	REAL*8	COMPLEX*8	COMPLEX*16
REAL*8	REAL*8	REAL*8	REAL*8	REAL*8	COMPLEX*16	COMPLEX*16
COMPLEX*8	COMPLEX*8	COMPLEX*8	COMPLEX*8	COMPLEX*16	COMPLEX*8	COMPLEX*16
COMPLEX*16	COMPLEX*16	COMPLEX*16	COMPLEX*16	COMPLEX*16	COMPLEX*16	COMPLEX*16

5.) Eine DO-Schleife darf höchstens $2^{22}-1=4194303$-mal durchlaufen werden. Die letzte Anweisung einer DO-Schleife darf keine GOTO-, STOP-, arithmetische IF-,RETURN-,DO-Anweisung sein.

6.) Bei SUBROUTINES sind Rücksprünge an unterschiedliche Label möglich.
z.B.: Deklaration: SUBROUTINE S(A,B,*,*)

```
      :
RETURN 1
      :
RETURN 2
      :
END
```

Aufruf:CALL S(2.5,X+Y,&120,&290)
Bei der Vereinbarung muß als formaler Parameter das Zeichen * stehen.
Im aktuellen Aufruf müssen an den Positionen, auf denen bei der Deklaration das Zeichen * stand, Labelnummern stehen, denen das Zeichen & voransteht.

Wirkung: Wird RETURN 1 erreicht, so erfolgt Rücksprung nach Label 120, wird RETURN 2 erreicht, so erfolgt Rücksprung nach Label 290.

7.)Es existiert eine ENTRY-Anweisung der Form

ENTRY name (Parameterliste)

Diese kann an beliebiger Stelle in einer SUBROUTINE stehen und gestattet es, von außen mitten hinein in eine Subroutine zu

springen, indem mittels CALL der Name verbunden mit einer Liste aktueller Parameter aufgerufen wird.

8.) Hinsichtlich der Art der Parameter sind außer den identischen Korrespondenzen zwischen formalen und aktuellen Parametern noch folgende Kombinationen bei SUBROUTINE und ENTRY zugelassen.

aktuelle Parameter	formale Parameter
Feld	Variable
Feldelement	Feld
Ausdruck	Variable
Literalkonstante	Variable
Literalkonstante	Feld

9.) Bei der Typfestlegung von Funktions kann auch die Länge der Ergebnisgröße festgelegt werden.

d.h. neben FUNCTION F(A,B)
und INTEGER FUNCTION BLA(X,Y,Z)
ist auch INTEGER FUNCTION BLA * 2(X,Y,Z)
erlaubt.

Die ENTRY-Anweisung ist in analoger Weise wie bei 3.)beschrieben auc bei FUNCTIONs möglich.

10.) Durch die Anweisung IMPLICIT läßt sich die Standardfestlegung (alle mit I bis N beginnenden Größen sind INTEGER, die übrigen sin REAL) umdefinieren.

z.B. IMPLICIT REAL * 8 (A-C,W-Z)

bedeutet: alle mit A,B,C,W,X,Y,Z beginnenden Namen bedeuten REAL-Größen doppelter Länge.

Folgende Längen sind möglich:

INTEGER * 2 ,	INTEGER * 4
REAL * 4 ,	REAL * 8
COMPLEX * 8,	COMPLEX * 16
LOGICAL * 1,	LOGICAL * 4

Bei DOUBLE PRECISION ist keine Längenangabe möglich. Die unterstrichenen Angaben sind Standard.

1.) **Felder werden so abgespeichert, daß der erste Index zuerst, der letzte zuletzt läuft (d.h. spaltenweise bei zweidimensionalen Feldern).**
Feldelemente dürfen auch mit weniger Indices benutzt werden als bei der Deklaration festgelegt wurde.

z.B. `DIMENSION A(10,10)`
`:`
`:`
`B = A(12)`

Wirkung: B = A(2,2)

2.) Es existieren folgende zusätzliche Format-Spezifikationen

a) T-Format, Tn, dient zur Tabulation
Eingabe: Nächstes Zeichen wird ab Spalte n gelesen
Ausgabe: Nächstes Zeichen wird in Spalte n gedruckt

b) B-FORMAT (siehe 14e)

3.) Computed GOTO/ASSIGNED GOTO
Ist bei GOTO $(l_1,\ldots,l_k)$,j
$j < 1$ oder $j > k$, so ist die Anweisung leer.
Ist bei ASSIGN n TO i und GOTO i, $(j_1,\ldots,j_k)$ keines der j_v=n, so ist die Anweisung leer.

4.) Ein-/Ausgabeanweisungen

a) Es gibt Möglichkeiten des Error- und End-Read.
z.B. `READ (5,90,ERR=100)A,B,C`
`READ (5,110,END=200) X,Y,Z`
Erster Fall:
Bei Fehler in der Eingabe wird zu Label 100 gesprungen.
Zweiter Fall:
Sind keine Daten mehr vorhanden, so wird zu Label 200 gesprungen.
Es ist auch folgende Kombination erlaubt:
`READ (5,85,ERR=150,END=250) A,B,X,Z`

b) Standardkanal für Eingabe von Kartenleser ist 5, für Druckausgabe 6.
Beim Terminal gilt für Eingabe Nr. 8, für Ausgabe Nr. 9.

c) Statt WRITE ist auch PRINT erlaubt. Ausgabe geht dann über Drucker.

`WRITE(6,100)A,B,C`
`PRINT 100,A,B,C`
sind gleichbedeutend.

Statt WRITE ist auch PUNCH erlaubt. Die Ausgabe geht dann in eine Datei deren Inhalt gestanztwerden kann.
Z.B. `PUNCH 200,X,Y`
Es ist jedoch in Starte-Kommando in der Spezifikation DATEI die Kanalnummer 7 vorzusehen (s.Anhang 2).

d) Es gibt die Anweisung DEFINEFILE.
Diese Anweisung beschreibt Anzahl, Größe und Struktur der Datensätze für E/A im direkten Zugriff.
Einzelheiten siehe FORTRAN-Handbuch der Fa. CGK.

e) Bei dem TR 440 besteht die Möglichkeit, formatfrei einzulesen.
Es existiert dazu das B-Format, schreibbar als B oder nB (n ist Wiederholungsfaktor)
Beispiel
```
      READ(5,10) A,B,C
   10 FORMAT (3B)
```
Die Daten können völlig frei abgelocht werden, Trennzeichen ist das Komma.
Werden einer Größe von Typ COMPLEX auf diese Weise Werte zugewiesen, so sind zwei Elemente vorzusehen, eines für Real- und eines für Imaginärteil.
Beispiel
```
      COMPLEX X,Y,Z
      READ(5,20)X,Y,Z
   20 FORMAT (6B)
```

15.) Es besteht die Möglichkeit, sich Bitmuster von Variablen mit Hilfe des Z-Formates ausgeben zu lassen.
Der Ausdruck ergibt sich als 12-stellige Hexadezimalzahl.
Beispiel
```
      WRITE (6,17) A,I
   17 FORMAT (1H1,2Z)
```

16.) In FORTRAN können ALGOL-Prozeduren angeschlossen werden. Diese sind mit

ALGOL EXTERNAL Liste der Prozeduren

zu erklären.
Beispiel
```
      .
      .
      .
      ALGOL EXTERNAL BLA,SBLA
      .
      .
      .
      CALL SBLA(X,Y)
      .
      .
      .
      Q=BLA(2,5,A,B,X)
```

17.) Bei Vergleichsausdrücken (s. Seite 70) sind auch Ausdrücke der Form

V_1 o V_2 mit o $\in$ {.EQ.,.NE.,.GE.,.GT.,.LE.,.LT.}

erlaubt, wenn eine der beiden Größen V_1 oder V_2 vom Typ INTEGER und die andere vom Typ REAL oder DOUBLE ist.

18.) Bei der Vorbesetzung von Feldern mittels DATA-Statement darf abweichend vom Standard auch nur der Feldname angegeben zu werden. Es werden dann die Elemente ab dem ersten besetzt.

Beispiel:

```
DIMENSION IA(10)
DATA IA/3*1,2,2*4/
 •
 •
 •
```

hat die Wirkung IA(1)=1
IA(2)=1
IA(3)=1
IA(4)=2
IA(5)=4
IA(6)=4

IA(7) bis IA(10) bleiben undefiniert.

A N H A N G 2

Einige wesentliche Kommandos zum Bearbeiten von FORTRAN-Programmen auf dem Rechner TR 440

1.) Allgemeine Bemerkungen

Es werden grundsätzlich die folgenden beiden Betriebsarten unterschieden:

1.) Abschnittsbetrieb (Batchbetrieb)
2.) Gesprächsbetrieb (Time-sharing-Betrieb)

Abschnittsbetrieb: Ein Abschnitt aus einem Stapel Lochkarten, der im Standardfall ein Programm in einer höheren Programmiersprache und ggf. dazugehörigen Problemdaten umfasst. Zusätzlich sind Anweisungen an das Betriebssystem vorhanden, z. B. der Aufruf des benötigten Compiler,ggf Angaben über gewünschten Traceback, Angaben über die benötigten Betriebsmittel usw.

Der Stapel Lochkarten wird als ganzes in den Kartenleser gegeben, vom Rechnersystem bearbeitet und abschließend werden Ergebnisse, ggf. Fehlermeldungen etc. nach vorheriger Zwischenspeicherung auf der Wechselplatte über den Zeilendrucker ausgegeben.

Gesprächsbetrieb: Der Benutzer hat hierbei die Möglichkeit, über ein Datensichtgerät mit dem Rechner in einen Dialog zu treten. Er kann beispielsweise Programme eintippen, diese starten, Ergebnisse auf dem Bildschirm empfangen, Änderungen durchführen, erneut starten usw.. Ermöglicht wird diese Betriebsart dadurch, daß kleinere Zeitscheiben der Rechnerzeit zyklisch jedem einzelnen Benutzer am Sichtgerät vom Rechner zugewiesen werden. Bei einem reibungslosen Betrieb entsteht für den Benutzer der Eindruck als arbeite der Rechner nur für ihn.

Im folgenden werden Einzelheiten über den Abschnittsbetrieb gebracht, wobei im wesentlichen an Benutzer gedacht ist, die Programme in einer höheren Programmiersprache bearbeiten lassen wollen. Der schematische Aufbau eines einfachen Abschnittes ist der folgende (jede mit Strich beginnende Zeile bezeichnet eine Lochkarte)

- Anmelden des Abschnitts, Bedarfsangaben
- Übersetzen des Quellenprogramms, das Resultat ist das Montageobjekt.

- Montieren der Montageobjekte zu einem lauffähigen Programm (genannt Operator)
- Starten des Operators
  Daten für das Programm
- Abmelden des Abschnitts

Alle Anweisungen an das Betriebssystem (hier z. B. Anmelden, Abmelden, Obersetzen, Montieren, Starten) beginnen mit einem ganz speziellen Zeichen,dem sog. Fluchtsymbol wobei zwischen einem codeabhängigen und einem codeunabhängigen Fluchtsymbol unterschieden wird.
Unter den Anweisungen ans Betriebssystem, den sog. Kommandos, sind die Vermittlerkommandos von besonderer Bedeutung. Sie beginnen alle mit dem Buchstaben X und bestehen aus insgesamt 3 Buchstaben. Sie finden Verwendung, um z. B. Anfang und Ende eines Abschnitts zu kennzeichnen oder um Codeumschaltung anzuzeigen.
Formal werden sie eingeleitet durch das codeunabhängige Fluchtsymbol, darauf folgt eine der Ziffern 1,2,3 oder 4, um anzuzeigen, in welchem der vier Kartencodes (KC1, KC2, KC3, KC4) der Rest der Karte gelocht ist. Außerdem sind die Vermittlerkommandos mit der Zeichenfolge Fluchtsymbol (codeabhängiges) und Punkt abzuschließen.
Alle anderen Kommandos beginnen mit dem codeabhängigen Fluchtsymbol.
Standardcode im Rechenzentrum der Universität Osnabrück ist der KC4, das codeabhängige Fluchtsymbol in diesem Code ist das Zeichen # (Lochung 3-8). Um Kompatibilität zum abschließenden Fluchtsymbol der Vermittlerkommandos zu haben, sollten diese als zweites Zeichen eine 4 erhalten. Dem ist bereits durch organisatorische Maßnahmen (s. Seite 134 unten) Rechnung getragen worden.

Anmelden eines Abschnittes

Einfachste Form (große Buchstaben sind festgelegt, kleine Buchstaben bezeichnen Variable)

𝔛4XBA,BEN= <benutzerstring> ,SBG=k,RZS=m#.

Es bedeuten: XBA : das eigentliche Vermittlerkommando

BEN : Benutzer, für <benutzerstring> ist die vom Rechenzentrum zugeteilte Kombination aus sechsstelliger Zahl und Benutzernamen (dem BKZ=Benutzerkennzeichen) zu verwenden.

RZS : Rechenzeitschranke in Minuten. Wird ein Abschnitt in der angegebenen Zeit nicht fertig, so wird er abgebrochen. Auch für die weiteren Angaben der RZS kann man dem Ergebnisausdruck entnehmen, wieviel Rechenzeit (Rechenkernzeit) tatsächlich verbraucht wurde.

SBG : Speicherbedarfsgruppen, k ist eine natürliche Zahl mit $1 \leq k \leq 12$.

Um dem Benutzer die lästigen Überlegungen, wieviel Kernspeicher, Trommelspeicher und Plattenspeicher bzw. wieviel zu druckende Seiten er anfordern soll, zu ersparen, wurden die SBGn eingeführt.

𝔛 : steht für das codeunabhängige Fluchtsymbol, es hat die Lochung 11-12-5-8 und entsteht durch Übereinanderlochen von (und) (mittels MULTIPUNCH).

Derzeit gilt folgende Festlegung

SBG	KSB	TSB	PSB	DRS
1	18	100	100	30
2	18	100	150	60
3	23	100	100	30
4	23	100	200	60
5	27	100	150	30
6	27	150	300	60
7	32	100	150	30
8	32	150	300	60
9	37	150	150	30
10	37	200	300	60
11	40	150	250	30
12	40	200	400	60

Diese Festlegungen können im Bedarfsfall geändert werden. Änderungen werden den Benutzern durch Aushang mitgeteilt.

Bei reinen Texthaltungsaufträgen reicht SBG=1 oder SBG=2

für die Bearbeitung von FORTRAN-Programmen ist SBG=3 oder SBG=4 nötig

für die Bearbeitung von COBOL-Programmen ist SBG=4 oder SBG=5 nötig

für die Bearbeitung von ALGOL 60- und BASIC-Programmen ist SBG=7 nötig

Wird keine Angabe zu SBG gemacht, so hat dies die Wirkung SBG=3.

Erläuterungen: Es bedeuten: KSP = Kernspeicherbedarf,
TSB = Trommelspeicherbedarf,
PSB = Plattenspeicherbedarf,
DRS = Druckseitenschranke.

Der Begriff TSB ist historisch zu sehen, hardwaremäßig gibt es nur Plattenspeicher, ein Teil dessen wird aber bei jedem Auftrag logisch anders als der PSB behandelt, dies ist der TSB.

Stets muß gelten TSB $\geq$ KSB.

Die Zahlenangaben der ersten drei Spalten bezeichnen K Worte, die der vierten die Anzahl der Druckseiten des Ergebnisausdruckes.

Die durch SBG=k gemäß obiger Tabelle festgelegten Größen können durch explizite Angabe von KSB, TSB, PSB und DRS überschrieben werden. Dabei ist zu bedenken, daß bei einer starken Erhöhung von DRS auch PSB erhöht werden muß, da der Output vor Ausgabe über den Drucker auf dem Plattenspeicher gesammelt wird. Sonst erfolgt die Fehlermeldung:

UNZULAESSIGE SPEZIFIKATION DRS/PSB.

Beispiele:

1.) X4XBA,BEN=254098MEYER,SBG=4,RZS=4#.

Zur Verfügung stehen dann: Kernspeicher: 23k
Trommelspeicher: 100k
Plattenspeicher: 200k
Druckseiten : 60 Seiten

2.) X4XBA,BEN=254098MEYER,SBG=4,KSB=30,DRS=40,RZS=4#.

wie 1.) nur mit Kernspeicher: 30k und DRS: 40 Seiten

3.) X4XBA,BEN=254098MEYER,KSB=28,TSB=100,PSB=150,DRS=40,RZS=4#.

Hier wird auf die Angabe von SBG völlig verzichtet, da alle 4 einzelnen Bedarfsgrößen explizit angegeben wurden.

Am Ende eines jeden Programmlaufs werden dem Benutzer die Werte ausgegeben, die er angefordert hat und zusätzlich erhält er die Maximalwerte, die er tatsächlich benötigt hat, so daß er bei einem weiteren Lauf desselben Programms eine Vorstellung über die zu wählende SBG hat. Es kann z. B. auch vorkommen, daß ein FORTRAN-Programm so groß ist, daß es zwar durch Angabe von SBG=2 noch übersetzt werden kann, daß aber ein Lauf des Programmes wegen Kernspeichermangel nicht erfolgen kann.

Weitere Angaben auf der XBA-Karte.
Hinter BEN= <benutzerstring> ist noch die Angabe
FKZ= <string> möglich. FKZ steht für freies Kennzeichen, von dem angegebenen string werden nur die ersten sechs Zeichen erkannt.
Ein FKZ ist immer anzugeben, da nach diesem die Ergebnissausdrucke vom Operateur sortiert werden. Auf dem Anfang eines jeden Ausdruckes befindet sich neben einer laufenden Nummer, die vom Betriebssystem vergeben wird, eine Angabe über BEN und FKZ.

Beispiel: X4XBA,BEN=991002BRAUER,FKZ=AUFG1,SBG=4,RZS=2#.

Abmelden eines Abschnitts

Diese Karte hat die konstante Form:

X4XEN#.

Achtung: Hinter diese Karte ist als allerletzte Karte eine Leerkarte zu legen, da der Abschnitt sonst mit der Fehlermeldung

FEHLER IM 2. VERMITTLERKOMMANDO

abgebrochen wird.

Organisatorische Regelungen zu XBA- und XEN-Karten

In den Locherräumen des RZ liegen vorgelochte rote Karten mit X4XBA,BEN= aus. Die weiteren Angaben muß der Benutzer selbst lochen. Es liegen ebenfalls vollständig vorgelochte XEN-Karten aus (Farbe blau).

2.) Das UEBERSETZE-Kommando

Der einfachste Fall zur Bearbeitung von FORTRAN-Programmen ist:

```
#UEBERSETZE,QUELLE=/

  Fortran-Programm
```

Der Compiler (Übersetzer) braucht eine Angabe über das zu übersetzende Programm. Dies kann entweder in einer Datei stehen (dann ist zu schreiben: QUELLE=dateiname) oder als sog. Fremdstring beizugeben. Im letzteren Fall wird wie oben verfahren.

Der Kommandoname UEBERSETZE und der sog. Spezifikationsname QUELLE können abgekürzt werden (unter Hinzufügung eines Abkürzungspunktes). D.H.: Man kann schreiben:

```
#UE.,Q.=/

Fortran-Programm
```

Die Abkürzung muß allerdings eindeutig sein. Da es noch ein weiteres TR 440-Kommando gibt, welches mit U beginnt, reicht es nicht, #U. zu schreiben.

Es gibt noch weitere Spezifikationen, von denen hier nur die wichtigsten erwähnt werden. Im übrigen wird auf das Benutzerhandbuch des Rechenzentrum der Universität Osnabrück verwiesen.

Weitere Spezifikationen:

SPRACHE =

Nur bei FORTRAN muß diese nicht angegeben werden, da bei Fehlen der Angabe automatisch angenommen wird, es handele sich um ein FORTRAN-Programm.

Sonst wäre zu schreiben: SPRACHE=FTN

DYNKON =

Einbau von dynamischen Kontrollen.

Beim Lauf des Programmes wird i.a. nicht geprüft, ob Feldgrenzen überschritten werden, es können dann die für den Anfänger merkwürdigsten Fehlermeldungen wie z. B. TYPENKENNUNGS-ALARM auftreten. Durch Angabe von DYNKON wird erreicht, daß bei Lauf des Programmes Feldgrenzenüberschreitungen als solche erkannt und gemeldet werden. Die Rechenzeit erhöht sich allerdings bei Verwendung dieser Spezifikation.

Möglichkeiten: DYNKON = -STD- : Kontrollen werden im ganzen Programm eingebaut

DYNKON =(a-e) : Kontrollen werden von Zeile a bis Zeile e eingebaut.
Im Standardfall werden die Zeilen des Programmlistings beginnend bei 10 in Schritten von 10 durchnumeriert.

DYNKON =(a) : Kontrollen werden für Zeile a eingebaut.

TRACE =

Einbau einer Ablaufüberwachung. Läßt sich ein Fehler überhaupt nicht finden, dann kann eine lückenlose historische Ablaufprotokollierung eines jeden Schrittes für das ganze Programm oder für Teile oder für spezielle Anweisungen erreicht werden.

Möglichkeiten: TRACE = -STD- : Überwachung für das ganze Programm

TRACE = (a-e) : Überwachung von Zeile a bis Zeile e

TRACE = GOTO : dto. bei allen Sprüngen

TRACE = GOTO (a-e): dto. bei allen Sprüngen von Zeile a bis Zeile e

TRACE = ASSIGN : dto. bei Wertzuweisungen

TRACE = ASSIGN(a-e): dto. bei Wertzuweisungen von Zeile a bis Zeile e

TRACE = CALL : dto. bei Unterprogrammaufrufen und Rücksprüngen

TRACE = CALL(a-e): wie oben, aber Zeile a bis Zeile e

Mehrere Werte können kombiniert werden, Trennzeichen ist Apostroph
z. B. TRACE = CALL'GOTO.
Mehrere Zeilenbereiche können kombiniert werden, Trennzeichen ist Komma.
z. B. 1.) TRACE = (10-100), (730-900)
2.) TRACE = (10-100)'CALL'ASSIGN(600-700),(900-1100)

Es ist zu beachten, daß bei Verwendung dieser Möglichkeit ggf. sehr viel Output entsteht (PSB und DRS entsprechend anfordern).
Vollständiges Beispiel:

```
# UE.,TRACE=CALL,DYNKON=(250-400),QUELLE=/
        Fortran-Programmkarten.
```

Anmerkung: Steht Quelle nicht als letzte Spezifikation, so ist hinter die gesamte Quelle noch die Zeichenfolge #/ als Abschluß zu setzen.

Beispiel: #UE.,Q.=/

```
      WRITE(6,1)
1     FORMAT(1H1,22H▽DIES▽IST▽EIN▽BEISPIEL)
      END
#/,TRACE=CALL'GOTO,DYN.=-STD-
```

3.) Das MONTIERE-Kommando

Dieses Kommando kommt hinter das UEBERSETZE-Kommando und die Quelle. Es lautet im einfachsten Fall:

MONTIERE bzw. #MO.

Weitere Möglichkeiten (Spezifikationen) finden sich im Benutzerhandbuch für den TR 440 des RZ der Universität Osnabrück.

4.) Das STARTE-Kommando

Mit diesem Kommando wird das durch UEBERSETZE und MONTIERE entstandene Programm gestartet.

Die wichtigsten Spezifikationen sind.

DATEN=/

Datenkarten

Daten sind gemäß den FORMAT-Angaben des Programms abzulochen.

Die Daten sind als Fremdstring beigefügt, folgen weitere Spezifikatione so ist am Ende noch die Zeichenfolge #/ anzufügen.

DATEI=

Liegen die Daten nicht auf Karten, sondern bereits in einer Plattendatei vor, so ist zu schreiben:

DATEI=x-name

x: Kanalnummer der Datei, sie taucht als erster Parameter in einer READ-Anweisung auf, $1 \leq x \leq 99$

name: Name der Datei

Es können mehrere Angaben -durch Apostroph voneinander getrennt- gemacht werden.

z.B.: DATEI=17-DATEIN '91-WERTE

DNUMMER= (abgekürzt:DN.=)

Umnumerierung von logischen Gerätenummern.

Form: DN.=xUy (Nummer x wird in y geändert)

bzw. DN.=x1Uy1'x2Uy2 (x1 wird in y1, x2 wird in y2 geändert)

Hat man z. B. seine Daten nicht mehr auf Karten, sondern in einer Datei BLABLA, so müßte man im ganzen Programm die Kanalnummer 5 durch eine andere ersetzen. Eleganter geht dies wie folgt:

#STARTE,DN.=5U29,DATEI=29-BLABLA

DUMP= (abgekürzt DU.)

Durch Angabe dieser Spezifikation läßt sich im Fehlerfall ein Dump erzeugen, d.h. eine Liste der Variablen und ihrer aktuellen Werte zum Zeitpunkt des durch den Fehler erzwungenen Programmabbruches.
Die für Fortran allgemeine Form lautet:

DUMP=F-d(e)

wobei (e)weggelassen werden kann.
Bedeutung von d(d kann sein):

ALLES (Gesamtdump)

NEST Dumps aller Variablen der an der aktuellen Aufrufverschachtelung beteiligten Programmeinheiten

TEIL Dump aller Variablen der betroffenen Programmeinheit

NICHTS kein Dump (s.jedoch unten)

Für e kann eine Liste von Variablennamen oder Programmeinheitsnamen (durch Kommata voneinander getrennt) angegeben werden.
Man hat dann die oben beschriebenen Möglichkeiten für alle Variablen bzw. Programmeinheiten mit Ausnahme der in der Liste e angegebenen.

Beispiel: 1.) DUMP=F-ALLES (Gesamtdump)

2.) DUMP=F-NICHTS(A,B,SUB1) (Dump nur der Variablen A und B im ganzen Programm sowie aller Größen der Subroutine SUB1)

3.) DUMP=F-TEIL (A1,A2,X,Y,Z) (Dump aller Größen der gerade durchlaufenen Programmeinheit mit Ausnahme von A1,A2,X,Y,Z).

5.) Vollständiges Beispiel eines FORTRAN-Jobs
(Eine Zeile entspricht einer Lochkarte)

```
X4XBA,BEN=991001BRAUER,FKZ=HUGO,SBG=3,RZS=1#.
#UE.,Q.=/
      DIMENSION A(100),B(100)
      READ(5,1) N
      IF(N .GT. 100 .OR. N .LE. 0) CALL ENDE(N)
      READ(5,2) (A(I), I=1,N)
      DO 7 I=1,N
    7 B(I) = A(I)
      N1 = N - 1
      DO 8 I=1,N1
      I1 = I+1
      DO 9 J = I1,N
      IF(A(I) .LE. A(J)) GOTO 9
      H = A(I)
      A(I) = A(J)
      A(J) = H
    9 CONTINUE
    8 CONTINUE
      WRITE(6,3) N
    3 FORMAT(1H1, 10HORDNEN VON,I3,7H ZAHLEN //
     1  11H UNGEORDNET, 20X, 8HGEORDNET)
      DO 6 I=1,N
    6 WRITE(6,4) B(I), A(I)
    1 FORMAT(I3)
    2 FORMAT(10B)
    4 FORMAT( F13.5,19X, F12.5)
      END
      SUBROUTINE ENDE(K)
      WRITE(6,25) K
   25 FORMAT(31H1FALSCHER WERT VON N EINGELESEN // 4H N = , I15)
      STOP
      END
#MO.

#STARTE,DATEN=/
 13
```

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80

12.[illegible]36[illegible],0.[illegible]25,-4.36,3.937,3.[illegible]6,-23406.123,0.000035,0.0,453214.1,0.0001

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80

1.0001,2.00[illegible],-32.45[illegible]

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80

```
X4XEN#.
```

Man erhält dann folgenden Ergebnisausdruck:

```
ORDNEN VON 13 ZAHLEN

UNGEORDNET                    GEORDNET
     12.35690                -23456.12300
      0.00230                   -32.45600
     -4.56000                    -4.56000
      3.98700                     0.0
      3.98600                     0.00003
-23456.12300                      0.00010
      0.00003                     0.00230
      0.0                         1.00010
453214.10000                      2.00100
      0.00010                     3.98600
      1.00010                     3.98700
      2.00100                    12.35690
    -32.45600                   453214.10000
```

A N H A N G 3

Verweis auf spezielle TR 440-Literatur

Der auf dem TR 440 implementierte Sprachumfang der Sprache FORTRAN ist in einem Handbuch der Fa. CGK beschrieben, Titel: FORTRAN-Sprachbeschreibung, Publikationsnummer ist: 440.D1.03.

1.) Eine Reihe spezieller Unterprogramme in FORTRAN auf dem TR 440 zur Manipulation von Zeichenketten, wobei konsequent die Möglichkeit des Rechners (1 Wort ≙ 6 Byte) ausgenutzt werden, ist in Kapitel 3 des o.a. Handbuches beschrieben. Die Unterprogramme können mittels CALL genauso aufgerufen werden, als hätte der Programmierer sie selbst geschrieben.
Publikationsnummer: 440.D1.03

2.) Es existieren einige Hilfsprogramme zum Anschluß in FORTRAN für folgende Aufgaben:

- Manipulation von TR 440-Worten
- Sortieren und Mischen
- Ein-/Ausgabe
- Fehler- und Alarmmeldungen
- Zeit- und Wahlschalterabfragen

Diese Unterprogramme können als FUNCTIONS oder SUBROUTINES ebenfalls so aufgerufen werden, als hätte der Programmierer sie selbst geschrieben.
Titel der Schrift: FORTRAN-Hilfsprogramme
Publikationsnummer: 440.E1.11

3.) Eine große Anzahl von Unterprogrammen zur Bearbeitung von Problemen der Statistik ist in einem weiteren Handbuch beschrieben. Es handelt sich um Unterprogramme für:

- Elementare statistische Probleme
- Zufallszahlen
- Verteilungen
- Bestimmung von Konfidenzintervallen
- Parametertests
- Anpassungstests
- Varianzanalysen
- Regressionsanalysen
- Korrelationsanalysen
- Faktorenanalyse
- Diskriminanzanalyse

Diese Unterprogramme können als FUNCTIONS oder SUBROUTINES so aufgerufen werden, als hätte der Programmierer sie selbst geschrieben.
Titel der Schrift: FORTRAN-Statistik
Publikationsnummer: 440.F4.12

4.) Eine weitere Schrift beschreibt die Anwendung von insgesamt 339

Unterprogrammen zur Lösung von Problemen der Linearen Algebra als da sind:

- Lösung linearer Gleichungssysteme
- Invertierung von Matrizen
- LU-Zerlegung von Matrizen
- Bestimmung von Eigenwerten und Eigenvektoren bei Matrizen

Die meisten dieser Unterprogramme existieren in 3 Versionen

- reell, einfach genau
- reell, doppelt genau
- komplex

Beim Aufruf dieser Routinen, der so erfolgen kann wie bei selbstgeschriebenen Programmen, ist vor dem MONTIERE-Kommando die Bibliothek anzumelden, in der sich diese vorübersetzten Unterprogramme befinden. Dies geschieht mit dem Kommando

#BIBANMELDE,FORMAT

Titel der Schrift: TR 440 Programmbiliothek, FORMAT-440
Publikationsnummer: 440.F3.13

Die vorstehend erwähnten Schriften sind bei Bedarf in beschränkter Anzahl im Rechenzentrum auszuleihen. In der Außenstelle des RZ in der Abt. Vechta können diese Schriften eingesehen werden.