robotron

Programmtechnische
Beschreibung

FORTRAN77

Sprachbeschreibung

C 1016-0200-2 M 3030

Arbeitsplatzcomputer A 7150 Betriebssystem SCP/DCP

Anwender-Dokumentation	FORTRAN77 Sprachbeschreibung	MOS
1/87		SCP/DCP

Programmtechnische Beschreibung

MOS

FORTRAN77

Sprachbeschreibung

VEB Robotron-Projekt Dresden

C 1016-0200-2 M 3030

Kurzreferat

Diese Dokumentation beschreibt den vom FORTRAN77-Compiler des SCP/DCP realisierten Sprachumfang. Realisiert werden alle Elemente der Sprache FORTRAN77 (gemäß dem Standard ANSI X3.9 - 1978) sowie eine geringe Anzahl von Sprachelementen früherer FORTRAN-Versionen, die mit diesem Standard verträglich sind und besondere Bedeutung für Kleinrechner haben. Implementierungsbedingte Einschränkungen betreffen nur Größen, die bei Übersetzung oder Ausführung vom verfügbaren Speicherplatz abhängen. Solche Einschränkungen werden nicht hier, sondern in [1] behandelt. Dort werden auch das Zusammenwirken des FORTRAN77-Compilers mit dem Betriebssystem und alle Fragen der Übersetzung und Ausführung von FORTRAN77-Programmen behandelt.

Diese Dokumentation ist ein Handbuch und Nachschlagewerk, aber kein Lehrbuch. Sie wendet sich an alle FORTRAN77-Nutzer, sowohl an Anfänger als auch an fortgeschrittene Programmierer. FORTRAN-Kenntnisse werden nicht unbedingt vorausgesetzt, jedoch sollte der Leser mit wenigstens einer höheren Programmiersprache vertraut sein.

Ag 706/542/88

Die vorliegende Systemunterlagen-Dokumentation,
FORTRAN77-Sprachbeschreibung,
entspricht dem Stand 10/85.

Die Ausarbeitung erfolgte durch ein Kollektiv
des VEB Robotron-Projekt Dresden.

Im Interesse einer ständigen Weiterentwicklung
werden alle Leser gebeten, Hinweise zur
Verbesserung dem Herausgeber mitzuteilen.

Herausgeber:

VEB Robotron-Projekt Dresden
Leningrader Straße 9
8010 Dresden

Manuskriptdruck

Inhaltsverzeichnis

Tabellenverzeichnis

Bildverzeichnis

Einführung

Der in der vorliegenden Dokumentation beschriebene Sprachumfang basiert auf dem FORTRAN77-Standard ANSI X3.9-1978 und enthält einige zusätzliche Sprachelemente. Die Beschreibung zusätzlicher Elemente, deren Realisierung vom verwendeten Betriebssystem abhängt, erfolgt in [1].
Diese Dokumentation beschreibt die Form von FORTRAN77-Programmen und gibt an, wie solche Programme und ihre Daten vom Rechner interpretiert werden. Für die Beschreibung der Form von FORTRAN77-Sprachelementen gelten folgende Konventionen:

- Sonderzeichen des FORTRAN77-Zeichensatzes, Großbuchstaben und Worte aus Großbuchstaben muß der Programmierer wie angegeben notieren.
- Kleinbuchstaben und kleingeschriebene Worte kennzeichnen allgemein Objekte, die in den einzelnen Anweisungen durch spezielle Objekte ersetzt werden müssen.
- Eckige Klammern, [ ], werden benutzt, um wahlweise angebbare Elemente zu kennzeichnen.
- Abkürzungspunkte, ..., zeigen an, daß das vorangehende wahlweise Element mehrmals hintereinander angegeben werden kann.
- Leerzeichen dienen, wenn nicht extra angeführt, nur der Verbesserung der Lesbarkeit.

Eine exakte, geschlossene Syntax von FORTRAN77 ist in Anlage 1 enthalten. Die verwendete, grafisch orientierte Syntaxnotation wird am Anfang dieser Anlage erläutert. Verbale Aussagen ergänzen die formale Darstellung.

1. Programmstruktur und Codierungsvorschriften

Ein FORTRAN77-Programm besteht aus genau einem Hauptprogramm und einer beliebigen Anzahl von Unterprogrammen und externen Prozeduren. Das Hauptprogramm und jedes Unterprogramm bildet eine Programmeinheit. Der FORTRAN77-Compiler übersetzt jeweils eine Programmeinheit und überprüft keine Beziehungen zwischen den einzelnen Programmeinheiten eines Programms. Externe Prozeduren sind entweder spezielle Unterprogramme (FUNCTION- bzw. SUBROUTINE-Unterprogramme) oder Programmteile, die nicht mit FORTRAN77-Mitteln erzeugt wurden. Details zum Aufbau eines Hauptprogramms und der verschiedenenen Arten von Unterprogrammen sind den Abschnitten 1.4. und 5. zu entnehmen.
Jede Programmeinheit besteht aus einer Folge von Anweisungen, die stets mit der END-Anweisung abgeschlossen wird. Zum Zwecke der Dokumentation können vor der END-Anweisung an beliebigen Stellen der Programmeinheit Kommentarzeilen eingefügt werden. Kommentarzeilen haben keine Auswirkung auf die Arbeitsweise des Programms.

1.1. FORTRAN77-Zeichensatz

Grundlage für die Codierung des Quellprogramms ist der aus 49 Zeichen bestehende FORTRAN77-Zeichensatz. Er kann, wie in Tabelle 1 angegeben, in drei Gruppen von Zeichen eingeteilt werden. In Kommentarzeilen, Zeichenkettenkonstanten und in den bei der FORMAT-Anweisung im Abschnitt 7. beschriebenen Hollerith-Konstanten können alle Zeichen des Datenzeichensatzes des benutzten Rechners auftreten.

Tabelle 1 FORTRAN77-Zeichensatz

Gruppe	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
Ziffern	0 1 2 3 4 5 6 7 8 9
Sonder-zeichen	Leerzeichen = + - * / ( ) , . ¤ ' ;

FORTRAN77 verlangt, daß die Sortierfolge des verwendeten Datenzeichensatzes auch bestimmte Größer-Kleiner-Beziehungen zwischen den Elementen des FORTRAN77-Zeichensatzes gewährleistet, siehe hierzu Abschnitt 2.7.3.

1.2. Syntaktische Elemente

Syntaktische Elemente sind die Grundbausteine aller Anweisungen. Zur Bildung syntaktischer Elemente sind in der Regel nur die Zeichen des FORTRAN77-Zeichensatzes zugelassen. Die grundlegenden syntaktischen Elemente von FORTRAN77 sind:

- Konstanten (siehe Abschnitt 2.4.)
- symbolische Namen (siehe Abschnitt 2.2.)
- Marken (siehe Abschnitt 1.3.2.)
- Schlüsselwörter
- Operatoren
- Sonderzeichen (siehe Abschnitt 1.1.)

Operatoren sind teils durch Sonderzeichen oder Folgen davon, teils durch je von einem Punkt begrenzte Buchstabenfolgen realisiert (siehe Abschnitt 2.7.).
Schlüsselwörter sind spezielle Zeichenfolgen, die der Benennung von Anweisungen oder von in Anweisungen zu codierenden Detailangaben dienen. Für viele Schlüsselwörter oder Teile davon sind die gleichen Bildungsregeln wie für symbolische Namen zutreffend. Die Entscheidung darüber, ob eine Zeichenfolge als Schlüsselwort oder als symbolischer Name gedeutet wird, beruht immer auf ihrer Stellung im Kontext. Schlüsselwörter sind deshalb in FORTRAN77 keine reservierten Zeichenfolgen. Je nach dem Kontext kann eine Zeichenfolge als symbolischer Name und auch als Schlüsselwort benutzt werden.
FORTRAN77 verlangt keine Trennung syntaktischer Elemente durch Leerzeichen. Das Ende eines syntaktischen Elements wird stets aus dem Kontext abgeleitet. Um die Lesbarkeit eines Programms zu erhöhen, dürfen aber zwischen zwei syntaktischen Elementen beliebig viele Leerzeichen eingefügt werden. Leerzeichen dürfen (außer in Zeichenkettenkonstanten) sogar beliebig in syntaktische Einheiten eingefügt werden, ohne deren Bedeutung zu verändern.

1.3. Codierungsvorschriften, Anweisungen und Kommentare

FORTRAN77 verlangt den Programmtext in einem festen Format. Jede Anweisung beginnt auf einer neuen Zeile (eine Ausnahme hierzu bildet die logische IF-Anweisung, siehe Abschnitt 4.3.2.). Für Marken und die Kennzeichnung von Kommentar- und Fortsetzungszeilen sind bestimmte Spalten vorgeschrieben.

1.3.1. Codierungsvorschriften

Programmzeilen

Tabelle 2 Aufbau einer Programmzeile

Feldname	Position
Feld für Kommentarkennzeichnung oder	1
Markenfeld	1-5
Feld für Fortsetzungskennzeichnung	6
Anweisungsfeld	7-72
Numerierungsfeld	73-80

Für den Programmtext müssen die Spalten 1 bis 72 verwendet werden. Im Numerierungsfeld können eine Folgenummer oder andere Informationen angegeben werden, die im Compilerprotokoll erscheinen, aber vom Compiler ignoriert werden.

Kommentarzeilen

Kommentarzeilen enthalten in Spalte 1 das Zeichen C oder *. In solch einer Zeile können alle Zeichen des verwendeten Datenzeichensatzes angegeben werden.
Eine Kommentarzeile hat keinen Einfluß auf die Programmausführung und kann an beliebiger Stelle eines Programms stehen. Eine Zeile, die in den Spalten 1 bis 72 nur Leerzeichen enthält, gilt auch als Kommentarzeile.

Anfangszeilen

Eine Anfangszeile ist eine Zeile, die keine Kommentarzeile ist und im Feld für Fortsetzungskennzeichen das Leerzeichen oder die Ziffer 0 enthält. Das Markenfeld kann eine Marke oder nur Leerzeichen enthalten.
Eine Anfangszeile, die in den Spalten 7-72 nur Leerzeichen und END enthält, wird stets als END-Anweisung gedeutet.

Fortsetzungszeilen

In einer Fortsetzungszeile:
- darf das Markenfeld nur aus Leerzeichen bestehen,
- muß im Feld für Fortsetzungskennzeichen ein Zeichen angegeben werden, das verschieden vom Leerzeichen und der Ziffer 0 ist.

Eine Anweisung darf nicht mehr als 19 Fortsetzungszeilen besitzen.

1.3.2. Anweisungen und Marken

Anweisungen

Anweisungen werden benutzt, um Programmeinheiten aufzubauen. Dabei werden ausführbare und nicht-ausführbare Anweisungen unterschieden. Während ausführbare Anweisungen Aktionen darstellen, umfassen die nicht-ausführbaren Anweisungen Typvereinbarungen, Anfangswertzuweisungen, Datenanordnungen, Formatbeschreibungen und Definitionen von Anweisungsfunktionen. Jede Anweisung wird im Anweisungsfeld einer Anfangszeile und in maximal 19 Fortsetzungszeilen geschrieben, d.h. die maximale Länge beträgt 1320 Zeichen.
Eine END-Anweisung darf keine Fortsetzungszeilen haben. Außer als Teil einer logischen IF-Anweisung, darf keine Anweisung auf einer Zeile beginnen, die einen Teil der voranstehenden Anweisung enthält.
Leerzeichen, die einer Anweisung vorangehen, innerhalb einer Anweisung auftreten oder ihr folgen, verändern die Bedeutung der Anweisung nicht, gehen aber in ihre Gesamtlänge ein.

Marken

Marken dienen der Kennzeichnung von Anweisungen und können im Markenfeld angegeben werden.
Obwohl jede Anweisung mit einer Marke versehen werden kann, ist eine Bezugnahme nur auf FORMAT-Anweisungen und markierte ausführbare Anweisungen zulässig. Eine Marke ist eine Folge von ein bis fünf Ziffern, von denen mindestens eine verschieden von Null sein muß. Sie kann irgendwo im Markenfeld einer Anfangszeile angeordnet werden. Führende Nullen und Leerzeichen werden ignoriert.
Jede Marke ist nur innerhalb einer Programmeinheit gültig und darf in einer Programmeinheit nur einmal im Markenfeld auftreten.

1.4. Programmeinheiten

FORTRAN77-Programmeinheiten müssen so aufgebaut sein, daß der Compiler zur Übersetzung einer beliebigen Anweisung keine Information aus nachfolgenden Anweisungen benötigt. Deshalb bestehen Regeln für die Aufeinanderfolge von Anweisungen innerhalb einer Programmeinheit Die im Bild 1 untereinander stehenden Programmelemente müssen stets nacheinander programmiert werden, während nebeneinander stehende keiner solchen Forderung bei der Programmierung unterliegen.

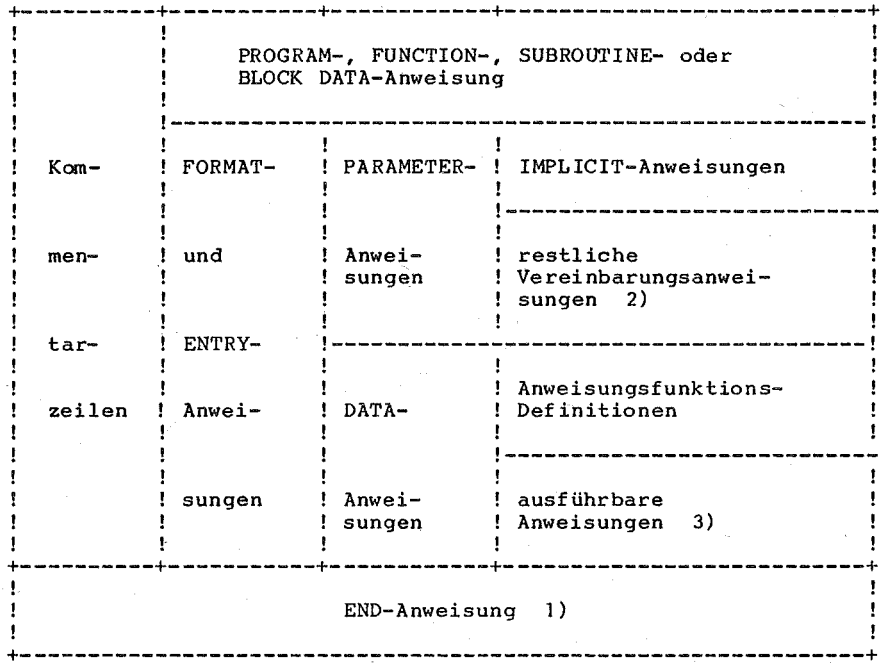


Bild 1 Aufbau einer Programmeinheit

1) Eine END-Anweisung muß stets angegeben werden.

2) Die Vereinbarungsanweisungen umfassen:

DIMENSION-Anweisung
COMMON-Anweisung
EQUIVALENCE-Anweisung
EXTERNAL-Anweisung
INTRINSIC-Anweisung
SAVE-Anweisung
Typvereinbarungs-Anweisung

3) Zu den ausführbaren Anweisungen gehören:

Ergibtanweisung
ASSIGN-Anweisung
unbedingte, berechnete, gesetzte GOTO-Anweisung
arithmetische, logische, Block-IF-Anweisung
ELSE-Anweisung
ENDIF-Anweisung
CONTINUE-Anweisung
STOP-Anweisung
PAUSE-Anweisung
DO-Anweisung
READ-Anweisung
WRITE-Anweisung
PRINT-Anweisung
BACKSPACE-Anweisung
ENDFILE-Anweisung
REWIND-Anweisung
OPEN-Anweisung
CLOSE-Anweisung
INQUIRE-Anweisung
CALL-Anweisung
RETURN-Anweisung

Zusätzlich zu der durch Bild 1 vorgegebenen Reihenfolge der Anweisungen müssen folgende Regeln eingehalten werden:

- Eine PROGRAM-Anweisung darf nur als erste Anweisung eines Hauptprogramms auftreten. Die erste Anweisung eines Unterprogrammes muß eine FUNCTION-, SUBROUTINE- oder BLOCK DATA-Anweisung sein. Entsprechend der ersten Anweisung werden Funktions-, SUBROUTINE- und BLOCK DATA-Unterprogramme unterschieden.
- Ein Hauptprogramm muß nicht mit einer PROGRAM-Anweisung beginnen. Jede Programmeinheit, die nicht mit einer FUNCTION-, SUBROUTINE- oder BLOCK DATA-Anweisung beginnt, ist ein Hauptprogramm.
- ENTRY-Anweisungen dürfen nicht innerhalb von IF-, ELSEIF- oder ELSE-Blöcken und nicht in DO-Schleifen auftreten, siehe Abschnitte 4.3.3. und 4.4.
- Bei der Verwendung von symbolischen Namen für Konstanten muß eine Vereinbarungsanweisung, die den Typ des symbolischen Namens der Konstanten festlegt, vor der PARAMETER-Anweisung stehen, welche den symbolschen Namen der Konstanten definiert. Die PARAMETER-Anweisung wiederum muß vor allen anderen Anweisungen stehen, die den symbolischen Namen dieser Konstanten enthalten.

2. Grundelemente der Sprache

Buchstaben, Ziffern und Sonderzeichen des FORTRAN77-Zeichensatzes (siehe Abschnitt 1.1.) werden benutzt, um die Grundelemente der Sprache FORTRAN77 darzustellen. Die Grundelemente werden in Konstanten, Objekte und Ausdrücke unterteilt. Zu den Objekten gehören Variablen und Felder.

2.1. Datentypen

Jedes Grundelement der Sprache repräsentiert einen Datentyp, der durch explizite Vereinbarung, textabhängige Verwendung oder interne Regeln festgelegt wird. Dabei werden 6 Grundtypen unterschieden, die mit folgenden Schlüsselworten verbunden sind:

INTEGER	-	Festkommadaten
REAL	-	Gleitkommadaten
DOUBLE PRECISION	-	Gleitkommadaten doppelter Genauigkeit
COMPLEX	-	Komplexe Daten
LOGICAL	-	Logische Daten
CHARACTER	-	Zeichenkettendaten

INTEGER, REAL, DOUBLE PRECISION und COMPLEX werden unter dem Begriff numerische Datentypen zusammengefaßt. Dabei gilt in arithmetischen Ausdrücken (siehe Abschnitt 2.7.1.) die oben angegebene Reihenfolge, d.h. INTEGER ist der niedrigste numerische Datentyp und COMPLEX der höchste.
Ein wichtiges Merkmal jedes Datentyps ist der Speicherplatz, der erforderlich ist, um einen Wert dieses Datentyps darzustellen. Tabelle 3 enthält alle möglichen Datentypen, die benötigten Speicherplätze und die Wertebereiche.

2.2. Symbolische Namen

Symbolische Namen (im weiteren auch als Name bezeichnet) dienen der Identifizierung verschiedener Elemente einer Programmeinheit, z.B. Variable, Felder, Programmeinheiten.
Ein symbolischer Name ist eine Folge von maximal 6 alphanumerischen Zeichen, wobei das erste Zeichen ein Buchstabe sein muß.
Aus Tabelle 4 ist zu entnehmen, welche Elemente einer Programmeinheit über symbolische Namen identifiziert werden, ob sie einen Datentyp besitzen und ihr symbolischer Name im gesamten FORTRAN77-Programm, also in allen dazugehörigen Programmeinheiten, oder nur innerhalb einer Programmeinheit eindeutig sein muß.

Tabelle 3 Datentypangaben

Datentyp	Speicherplatz (in Bytes)	Wertebereich
INTEGER	2 oder 4 2)	
INTEGER*2 1)	2	-32768 bis 32767
INTEGER*4 1)	4	-2147483648 bis 214748367
LOGICAL	1,2 oder 4 2)	.TRUE. oder .FALSE.
LOGICAL*1 1)	1	
LOGICAL*2 1)	2	
LOGICAL*4 1)	4	
REAL	4 oder 8 2)	betragsmäßig:
REAL*4 1)	4	0.29E-38 bis 1.7E38
REAL*8 1)	8	0.56D-308 bis 0.9D308
DOUBLE PRECISION	8	
COMPLEX	8	für REAL- und Imaginärteil wie bei REAL*4 oder INTEGER*4
CHARACTER	1	alle auf dem Rechner darstellbaren Zeichen
CHARACTER*länge	Länge 3)	
CHARACTER*(*)	4)	

1) Alle Datentypen mit einer Typlängenangabe der Form *n wurden zusätzlich zum FORTRAN77-Standard in die Sprache aufgenommen

2) In Abhängigkeit von Auswahlbedingungen (siehe [1]) werden ein, zwei oder vier Bytes zugeordnet. Die Standardzuordnung beträgt vier Bytes.
   Beachte: Werten der Standardfunktionen vom Datentyp INTEGER, REAL oder LOGICAL werden immer vier Bytes zugeordnet.

3) Länge ist entweder eine vorzeichenlose ganzzahlige Konstante oder ein in Klammern eingeschlossener ganzzahliger konstanter Ausdruck mit einem Wert 0. Der Wert von Länge ist die Anzahl von Zeichen und kann im Bereich von 1 bis 32767 liegen. Die Länge von Zeichenkettenkonstanten ist durch die maximale Anweisungslänge von 1320 Zeichen beschränkt.

4) Diese Form der Längenangabe ist für formale Parameter und FUNCTIONUnterprogramm möglich, und bedeutet, daß die Länge des aktuellen Parameters oder der Funktionsbezugnahme (siehe Abschnitt 5.2.2.) verwendet wird.

Tabelle 4 Datentyp und Eindeutigkeit von symbolischen Namen

Elemente, die über symbolische Namen identifiziert werden	besitzt Datentyp	eindeutig Programm	innerhalb Programmeinheit
Variablen	x		x
Felder	x		x
Anweisungsfunktionen	x		x
Standardfunktionen	x	x	
FUNCTION-Unterprogramme	x	x	
SUBROUTINE-Unterprogramm		x	
COMMON-Blöcke		x	
Hauptprogramm		x	
BLOCKDATA-Unterprogramme		x	
FUNCTION-Eingangspunkte	x	x	
SUBROUTINE-Eingangspunkte		x	
PARAMETER-Konstanten	x		x
Externe Prozeduren	x	x	

Der Datentyp eines symbolischen Namens kann (außer bei Standardfunktionen)

explizit durch eine Typvereinbarungsanweisung oder eine FUNCTION-Anweisung
implizit durch eine IMPLICIT-Anweisung oder durch die Standardtypvereinbarung

festgelegt werden.

Die Standardtypvereinbarung besagt, daß die symbolischen Namen, die mit den Buchstaben I, J, K, L, M oder N vom Datentyp INTEGER mit Standardtyplänge (siehe [1]) und alle übrigen vom Datentyp REAL sind. Sie ist wirkungslos für alle symbolischen Namen, welche durch IMPLICIT-Anweisungen oder explizite Typvereinbarungen einen anderen Datentyp erhalten.

Die IMPLICIT-Anweisung (siehe Abschnitt 3.2.) erlaubt es, den Datentyp einschließlich der Typlänge von symbolischen Namen analog der Standardtypvereinbarung global für alle symbolischen Namen mit gleichem Anfangsbuchstaben festzulegen. Alle Datentypen außer CHARACTER*(*) können so vereinbart werden. Die IMPLICIT-Anweisung überschreibt für die angegebenen Buchstaben den Standardtyp und, falls angegeben, die Standardtyplänge.

Für alle symbolischen Namen, die einen anderen als den durch die Standardtypvereinbarung oder eine IMPLICIT-Anweisung festgelegten Datentyp erhalten sollen, muß der Datentyp explizit vereinbart werden. Dazu dienen die Typvereinbarungsanweisungen und für Namen von Funktionsunterprogrammen auch die FUNCTION-Anweisung. Für jeden symbolischen Namen darf der Datentyp in einer Programmeinheit nur einmal explizit vereinbart werden.

2.3. Variablen

Als Variablen werden symbolische Namen bezeichnet, die mit einem Speicherplatz verbunden sind, dessen Inhalt sich während der Programmausführung ändern kann. Den aktuellen Inhalt dieses Speicherplatzes, der dem vereinbarten Typ entsprechen muß, nennt man den Wert der Variablen.
Eine Variable gilt als definiert, wenn der mit ihr verbundene Speicherplatz mit einem Wert des zugehörigen Datentyps belegt wurde. Zur Definition von Variablen gibt es folgende Möglichkeiten:

- vor der Programmausführung mittels der DATA-Anweisung (siehe Abschnitt 3.10.)
- mittels einer Definitionsanweisung (siehe Abschnitt 4.)
- durch eine Eingabe-Anweisung (siehe Abschnitt 6.)

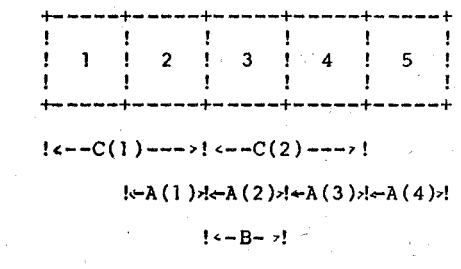
Wird einer Variablen mit numerischem Datentyp ein Wert zugewiesen,

der einen anderen numerischen Datentyp hat, so wird er in den Datentyp der Variablen konvertiert.
Mehrere Variablen heißen vollständig assoziiert, wenn sie mit ein und demselben Speicherplatz verbunden sind. Teilweise assoziiert heißen sie, wenn ein Teil des Speicherplatzes, der mit einer Variablen verbunden ist, mit einem Teil oder der Gesamtheit des Speicherplatzes, der mit einer anderen Variablen verbunden ist, identisch ist.
Vollständige oder teilweise Assoziation von Variablen können mit
- der COMMON-Anweisung oder
- der EQUIVALENCE-Anweisung

vereinbart oder durch Zuordnung von aktuellen zu formalen Parametern erreicht werden.
Die Anweisungsfolge

```
REAL A(4),B
COMPLEX C(2)
DOUBLE PRECISION D
EQUIVALENCE (C(2),A(2),B), (A,D)
```

bewirkt nach Ausführung der EQUIVALENCE-Anweisung (siehe Abschnitt 3.6.) folgende Speicherplatzbelegung



Daraus ist ersichtlich, daß A(2) und B vollständig assoziiert sind. Als teilweise assoziiert ergeben sich:

```
A(1) und C(1)
A(2) und C(2)
A(3) und C(2)
B    und C(2)
A(1) und D
A(2) und D
B    und D
C(1) und D
C(2) und D
```

Sind mehrere Variablen verschiedenen Datentyps assoziiert, so führt die Definition, d.h. die Wertzuweisung zu einer Variablen dazu, daß die anderen dieser Variablen als undefiniert gelten.

2.4. Konstanten

Als Konstanten werden Elemente bezeichnet, die ihren Wert während der Ausführung des Programms nicht ändern. In FORTRAN77 sind arithmetische, logische und Zeichenkettenkonstanten zugelassen. Ihr Wertebereich kann Tabelle 3 entnommem werden.

2.4.1. Ganzzahlige Konstanten

Eine ganzzahlige Konstante ist vom Typ INTEGER und hat folgende Form:

```
[v]zzz
```

Dabei gilt:

v - Vorzeichen
zzz - nichtleere Folge von Ziffern, die als Dezimalzahl interpretiert wird.

Führende Nullen werden ignoriert. Während negative ganzzahlige Konstanten mit einem Minuszeichen versehen sein müssen, ist das Pluszeichen für positive Konstanten wahlweise. Eine vorzeichenlose ganzzahlige Konstante wird als positiv angenommen.

2.4.2. Reelle Konstanten

Eine reelle Konstante ist vom Typ REAL oder DOUBLE PRECISION und muß eine der folgenden Formen haben:

```
[v]zzz.zzz[exp]
[v]zzz.[exp]
[v].zzz[exp]
[v]zzzexp
```

Dabei gilt:

v - Vorzeichen
zzz - nichtleere Folge von Ziffern
exp - Exponent zur Basis 10 in der Form E[v]zzz oder D[v]zzz

Die Mantisse der reellen Konstanten kann mehr Ziffern enthalten als im Rechner darstellbar sind.
Die Genauigkeit einer reellen Konstanten wird durch die Form des Exponenten festgelegt. Das Exponentensymbol E oder ein nicht angegebener Exponent legt einfache Genauigkeit fest, während D doppelte Genauigkeit erzwingt.

2.4.3. Komplexe Konstanten

Komplexe Konstanten sind vom Typ COMPLEX und haben folgende Form:

(x,y)

Dabei gilt:

- x - Realteil, reelle Konstante einfacher Genauigkeit oder ganzzahlige Konstante
- y - Imaginärteil, reelle Konstante einfacher Genauigkeit oder ganzzahlige Konstante

2.4.4. Logische Konstanten

Die logischen Konstanten "wahr" und "falsch" werden durch die Zeichenfolgen .TRUE. und .FALSE. dargestellt.

2.4.5. Zeichenkettenkonstanten

Eine Zeichenkettenkonstante hat den Typ CHARACTER und ist eine nicht leere auf dem Rechner darstellbare Kette von Zeichen, die in Hochkommas eingeschlossen ist. Die begrenzenden Hochkommas sind nicht Bestandteil des Wertes, den die Konstante verkörpert.
Ein Hochkomma innerhalb der Zeichenkettenkonstanten wird durch zwei aufeinanderfolgende Hochkommas dargestellt.
Die Länge einer Zeichenkettenkonstanten ist die Anzahl der Zeichen zwischen den begrenzenden Hochkommas, wobei ein Paar von aufeinanderfolgenden Hochkommas als ein Zeichen gezählt wird.

2.5. Felder

Ein Feld ist eine geordnete Menge von Daten, die in aufeinanderfolgenden Speicherplätzen untergebracht sind und auf die über einen symbolischen Namen, den Feldnamen, zugegriffen werden kann. Die Elemente dieser Datenmenge sind die Feldelemente. Die Bezugnahme auf die Feldelemente erfolgt über den Feldnamen, der mit einem Index versehen wird.

Ein Feld kann maximal 7 Dimensionen haben.
FORTRAN77 bietet folgende drei Anweisungstypen, die Feldvereinbarungen enthalten können:

- Typvereinbarungsanweisungen
- DIMENSION-Anweisung
- COMMON-Anweisung

Eine Feldvereinbarung legt den Feldnamen, die Anzahl der Dimensionen des Feldes und die Anzahl der Elemente in jeder Dimension fest.
Ein Feldelement gilt als definiert, wenn der dazugehörige Speicherplatz Daten enthält, die dem Datentyp des Feldnamens entsprechen. Bereits vor der Programmausführung kann ein Feldelement oder ein vollständiges Feld mittels einer DATA-Anweisung definiert werden. Durch eine Ergibtanweisung oder eine Anweisung für die Dateneingabe kann einem Feldelement während der Programmausführung ein Wert zugewiesen werden. Eingabeanweisungen bieten darüber hinaus die Möglichkeit, ein vollständiges Feld zu definieren.

2.5.1. Feldvereinbarungen

Eine Feldvereinbarung vereinbart den symbolischen Namen, der innerhalb einer Programmeinheit ein Feld bezeichnen soll, und beschreibt die Eigenschaften dieses Feldes.
Eine Feldvereinbarung hat die Form:

fn(dv[,dv]...)

Dabei ist

fn - der symbolische Name des Feldes, d.h. der Feldname
dv - eine Dimensionsvereinbarung der Form [ug:]og mit
ug - untere Grenze der Dimension und
og - obere Grenze der Dimension

Die Anzahl von Dimensionsvereinbarungen in der Liste bestimmt die Anzahl der Dimensionen des Feldes. Bis zu sieben Dimensionen sind erlaubt.
Jede Dimensionsgrenze ist ein ganzzahliger arithmetischer Ausdruck (siehe Abschnitt 2.7.1.), dessen Operanden Konstanten, formale Parameter oder Variable, die in einem COMMON-Block vereinbart wurden, sein können. Bezugnahmen auf Feldelemente oder Funktionen sind als Operanden verboten. Variable müssen vom Typ INTEGER sein. Sie dürfen in keiner nachfolgenden Typvereinbarung auftreten.
In einem Hauptprogramm sind nur Konstantenausdrücke als Dimensionsgrenzen gestattet.
Dimensionsgrenzen, die keine Konstantenausdrücke sind, können in

einem Unterprogramm benutzt werden, um dynamische Felder zu vereinbaren. Damit kann ein Unterprogramm Felder mit unterschiedlichen Dimensionsgrenzen verarbeiten. Dazu wird der Feldname dem Unterprogramm als Argument übergeben. Die Dimensionsgrenzen werden entweder als Argumente an das Unterprogramm übergeben oder sie befinden sich als Variable in einem COMMON-Block (siehe Abschnitt 5.1.1.). Der Wert des Ausdrucks für die untere Grenze kann negativ, Null oder positiv sein. Der Wert der unteren Grenze in der Dimensionsvereinbarung muß kleiner oder gleich dem Wert der entsprechenden oberen Grenze sein. Die Anzahl der Elemente in einer Dimension errechnet sich aus

og - ug + 1

Fehlt der Wert für die untere Grenze, so wird standardmäßig 1 angenommen und die Anzahl der Elemente in dieser Dimension ist identisch mit dem angegebenen Wert für die obere Grenze.
Die Anzahl der Elemente in einem Feld ist gleich dem Produkt der Anzahl der Elemente in jeder Dimension. Als obere Grenze der letzten Dimensionsvereinbarung kann ein Stern (*) angegegebn werden. Dann handelt es sich um die Vereinbarung eines Feldes angenommener Größe (siehe Abschnitt 5.1.2.).

2.5.2. Indizes

Die Bezugnahme auf ein einzelnes Feldelement erfolgt über den Feldelementnamen. Der Feldelementname ist der Feldname gefolgt von einem Index.
Ein Index hat die Form:

(ia [,ia] ...)

Dabei ist

ia - Indexausdruck

Ein Indexausdruck ist ein ganzzahliger arithmetischer Ausdruck. Eine indizierte Feldbezugnahme muß für jede Dimensionsvereinbarung einen Indexausdruck enthalten, wobei die Werte im Bereich von ug und og (siehe Abschnitt 2.5.1.) liegen müssen.

2.5.3. Speicherung von Feldern

In FORTRAN77 wird ein Feld als lineare Folge von Werten abgespeichert. Ein eindimensionales Feld wird beginnend mit dem ersten Element bis zu seinem letzten Element in fortlaufenden Speicherplätzen gespeichert. Die Elemente eines mehrdimensionalen

Feldes werden in aufeinanderfolgenden Speicherplätzen in solcher Reihenfolge gespeichert, daß sich die vorderen Indizes schneller ändern als hintere.

1. Vereinbarung des Feldes FE(2,3)

FE(1,1)	FE(2,1)	FE(1,2)	FE(2,2)	FE(1,3)	FE(2,3)

2. Vereinbarung des Feldes FE(2,3,3)

FE(1,1,1)	FE(2,1,1)	FE(1,2,1)	FE(2,2,1)	FE(1,3,1)	FE(2,3,1)
FE(1,1,2)	FE(2,1,2)	FE(1,2,2)	FE(2,2,2)	FE(1,3,2)	FE(2,3,2)
FE(1,1,3)	FE(2,1,3)	FE(1,2,3)	FE(2,2,3)	FE(1,3,3)	FE(2,3,3)

2.5.4. Der Datentyp eines Feldes

Der Datentyp eines Feldes wird auf die gleiche Weise festgelegt wie der Datentyp einer Variablen, d.h. der Datentyp eines Feldes wird entweder implizit durch den Anfangsbuchstaben des Feldnamens oder explizit durch eine Typvereinbarungsanweisung festgelegt.

2.5.5. Feldbezugnahmen ohne Index

Um anzuzeigen, daß das vollständige Feld benutzt oder vereinbart werden soll, können in folgenden Anweisungen Feldnamen ohne Index angegeben werden:

- COMMON-Anweisung
- DATA-Anweisung
- EQUIVALENCE-Anweisung
- SAVE-Anweisung
- E/A-Anweisungen
- Typvereinbarungsanweisungen

Nichtindizierte Feldnamen können ebenfalls als formale Parameter in FUNCTION-, SUBROUTINE- und ENTRY-Anweisungen oder als aktuelle Parameter in Bezugnahmen auf externe Prozeduren benutzt werden. In allen anderen Anweisungstypen ist die Angabe von nichtindizierten Feldnamen nicht gestattet.

2.6. Teilketten

Eine Teilkette ist ein zusammenhängender Teil einer Zeichenkette und hat den Datentyp
CHARACTER. Eine Teilkettenbezugnahme hat die Form:

```
vn([e1]:[e2])  oder
fn(ia[,ia]...)([e1]:[e2])
```

Dabei ist

- vn - der Name einer Zeichenkettenvariablen
- fn - der Name eines Feldes vom Typ CHARACTER
- ia - ein Indexausdruck
- e1 - ein ganzzahliger arithmetischer Ausdruck, der die linke Grenze der Teilkette angibt
- e2 - ein ganzzahliger arithmetischer Ausdruck, der die rechte Grenze der Teilkette angibt

e1 und e2 werden als Teilkettenausdrücke bezeichnet.
Die Positionen der Zeichen in der Speicherplatzfolge einer Zeichenkettenvariablen oder eines Feldelementes vom Typ CHARACTER werden, beginnend mit 1, von links nach rechts durchnumeriert. Die Grenzen, d.h. die Werte von e1 und e2, sind dann die Positionen des ersten bzw. letzten Zeichens der Teilkette innerhalb der Speicherplatzfolge der Zeichenkettenvariablen.
Sie müssen folgenden Bedingungen genügen:

```
1<=e1<=e2<=länge
```

wobei

- länge - die Länge der Zeichenkettenvariablen oder des Feldelementes vom Typ CHARACTER

ist.
e1 und e2 können beide weggelassen werden. Fehlt e1, so wird 1 angenommen. Wird e2 weggelassen, so bekommt e2 den Wert von länge.

Beispiel:

```
      CHARACTER*8 FELDCH(3,4)
      CHARACTER*5 KETTE
            .
            .
            .
      FELDCH(2,4) = 'BEISPIEL'
      KETTE = FELDCH(2,4)(4:)
```

Nach Ausführung dieser Anweisungen hat KETTE den Wert 'SPIEL'.

2.7. Ausdrücke

Ein Ausdruck repräsentiert einen Wert. Ausdrücke bestehen aus Operanden, Operatoren und Klammern. Die einfachste Form des Ausdrucks ist ein Operand. Die Operatoren geben an, welche Operationen mit den Werten der Operanden auszuführen sind, um den Wert des Ausdrucks zu erhalten. Durch Klammern wird festgelegt, in welcher Reihenfolge diese Operationen erfolgen sollen.

2.7.1. Arithmetische Ausdrücke

Arithmetische Ausdrücke dürfen nur arithmetische Operanden und Operatoren enthalten. Die Berechnung solch eines Ausdrucks liefert einen numerischen Wert. "Numerisch" soll hier einen der Typen INTEGER, REAL, DOUBLE PRECISION oder COMPLEX bedeuten.

Als arithmetische Operanden sind erlaubt:
- vorzeichenfreie numerische Konstanten
- symbolische Namen numerischer Konstanten
- Bezugnahmen auf numerische Variablen
- Bezugnahmen auf numerische Feldelemente
- Bezugnahmen auf numerische Funktionen
- arithmetische Ausdrücke, bei Bedarf in Klammern eingeschlossen

Die arithmetischen Operatoren und ihre Operationen sind:

Operator	Operation
**	Potenzierung
*	Multiplikation
/	Division
+	Addition oder Vorzeichen +
-	Subtraktion oder Vorzeichen -

Ein arithmetischer Ausdruck ist eine Folge arithmetischer Operanden und Operatoren, der mit einem Operanden endet. Zwei Operatoren oder zwei Operanden dürfen nicht unmittelbar aufeinander folgen. Der arithmetische Ausdruck beginnt mit +, - oder einem Operanden.

Beispiele:

```
-A ** Z
SQRT(B) + 25
A + B - C
A ** ( - (B + C))
```

Die Operatoren + und - sind unäre Operatoren (Vorzeichen), wenn unmittelbar vor ihnen kein arithmetischer Operand steht. Das gilt

für den Operator - im ersten und vierten Beispiel. In allen anderen Fällen sind die arithmetischen Operatoren binäre Operatoren und verlangen zwei Operanden.
Bei mehreren Operatoren im Ausdruck bestimmt der Rang der Operatoren die Reihenfolge der Operationen, sofern nicht durch Klammern eine andere Reihenfolge verlangt wird.

Operator	Rang
**	höchster Rang
* und /	mittlerer Rang
+ und -	niedrigster Rang

So hat im ersten Beispiel ** einen höheren Rang als - und wird zuerst ausgeführt. Das heißt, der Ausdruck wird als -(A**Z) interpretiert.
Mehrere Operatoren gleichen Ranges dürfen in beliebiger Reihenfolge ausgeführt werden, solange das Ergebnis algebraisch gleichwertig mit dem Ergebnis ist, das sich bei einer Links-Rechts-Folge der Operatoren ergibt. So darf das dritte Beispiel vom Rechner als (A+B)-C oder als A+(B-C) berechnet werden. Beide Ergebnisse sind zwar algebraisch gleichwertig, müssen aber bei Gleitkommazahlen nicht unbedingt numerisch identisch sein. Deshalb ist es sinnvoll, bei numerischen Berechnungen hoher Genauigkeit die Reihenfolge der Operationen durch Klammern festzulegen. Eine Ausnahme bildet die Potenzierung. Sie wird stets von rechts nach links ausgeführt, d.h. A**B**C wird immer als A**(B**C) berechnet.
Bei der Division von INTEGER-Operanden ist zu beachten, daß das Ergebnis nicht gerundet, sondern abgeschnitten wird. Zum Beispiel liefert -8/3 als Ergebnis den Wert -2. Das ist auch bei der Potenzierung von INTEGER-Operanden zu beachten. Beispielsweise hat 2**(-3) den Wert von 1/(2**3), also Null.
Die Potenzierung Z1**Z2 mit COMPLEX-Operanden liefert den Hauptwert von EXP(Z2*LOG(Z1)), wobei EXP und LOG Standardfunktionen sind (siehe Abschnitt 5.3.).

Datentyp eines arithmetischen Ausdrucks

In einem arithmetischen Ausdruck dürfen Operanden verschiedener numerischer Datentypen von INTEGER (niedrigster Typ) bis COMPLEX (höchster Typ) auftreten. Der Wert des Ergebnisses einer Operation erhält den höchsten Typ ihrer Operanden. Die letzte bei der Ausdrucksberechnung ausgeführte Operation bestimmt den Datentyp des Ausdrucks, der folglich dem höchsten Typ aller arithmetischen Operanden entspricht. Dabei ist zu beachten, daß der Datentyp von generischen Funktionsbezugnahmen von den Datentypen der Argumente abhängt (siehe Abschnitt 5.3.).
Treten in einer Operation Operanden verschiedener Typen auf, so

wird der Operand mit niedrigerem Typ vor Ausführung der Operation in den höheren Typ konvertiert. Solche impliziten Konvertierungen liefern dieselben Resultate wie die Standardfunktionen REAL, DBLE bzw. CMPLX. Diese Konvertierung entfällt bei der Potenzierung für Exponenten vom Typ INTEGER. Die Typen DOUBLE PRECISION und COMPLEX dürfen nicht in einer Operation auftreten.
Arithmetische Operationen, deren Ergebnis mathematisch nicht definiert ist, sind verboten. Beispiele dafür sind die Division durch Null, das Potenzieren mit Null als Basis und einem Wert = 0 als Exponent, sowie das Potenzieren einer Basis mit einem negativen Wert und einem Exponenten vom Typ REAL oder DOUBLE.

2.7.2. Zeichenkettenausdrücke

Ein Zeichenkettenausdruck hat die Form:

zeichenkettenoperand [// zeichenkettenoperand] ...

Dabei ist // der Verkettungsoperator. Der Wert des Zeichenkettenausdrucks ist die Zeichenkette, die durch schrittweise Verkettung ihrer Operanden entsteht. Die Länge des Zeichenkettenausdrucks ist die Summe der Längen seiner Operanden. Der Zeichenkettenausdruck 'ABC' // 'D' // 'EF' hat folglich den Wert 'ABCDEF' und die Länge 6.

Zeichenkettenoperanden sind:
- Zeichenkettenkonstanten
- symbolische Namen von Zeichenkettenkonstanten
- Bezugnahmen auf Zeichenkettenvariablen
- Bezugnahmen auf Zeichenkettenfeldelemente
- Bezugnahmen auf Teilketten
- Bezugnahmen auf Zeichenkettenfunktionen
- Zeichenkettenausdrücke, bei Bedarf in Klammern eingeschlossen

Wenn nicht durch Klammern eine andere Reihenfolge erzwungen wird, werden die Verkettungen von links nach rechts ausgeführt. D.h. 'AB' // 'CD' // 'EF' wird als ('AB' // 'CD') // 'EF' interpretiert.
Außer in Ergibtanweisungen darf ein Zeichenkettenausdruck keine Verkettungen mit Operanden enthalten, deren Länge mit (*) vereinbart wurde. Diese Einschränkung gilt nicht für symbolische Namen von Zeichenkettenkonstanten als Operanden. Der Rechner muß nur soviel von einem Zeichenkettenausdruck berechnen, wie es der Kontext verlangt. Zum Beispiel verlangen die Anweisungen

```
CHARACTER*2  C1,C2,C3,CF
C1 = C2 // CF(C3)
```

nicht, daß die Funktion CF aufgerufen wird, weil der Wert von C2 für die Zuweisung zu C1 schon ausreicht.

2.7.3. Vergleichsausdrücke

Ein Vergleichsausdruck besteht aus zwei arithmetischen Ausdrücken oder zwei Zeichenkettenausdrücken, die durch einen Vergleichsoperator getrennt sind. Die Berechnung des Vergleichsausdruckes liefert den logischen Wert .TRUE. oder .FALSE., je nach dem, ob die vom Vergleichsoperator angegebene Beziehung zwischen den Werten der arithmetischen bzw. Zeichenkettenausdrücke besteht oder nicht.
Die Vergleichsoperatoren und ihre Beziehungen sind:

Operator	Beziehung
.LT.	kleiner
.LE.	kleiner oder gleich
.EQ.	gleich
.NE.	ungleich
.GT.	größer
.GE.	größer oder gleich

Für den Vergleich zweier Ausdrücke vom Typ COMPLEX sind nur die Operatoren .EQ. und .NE. erlaubt. Enthält der Vergleichsausdruck arithmetische Ausdrücke unterschiedlicher Datentypen, so wird vor dem Vergleich der Ausdruck mit dem niedrigeren Typ in den höheren Typ konvertiert. Vergleiche eines Ausdrucks vom Typ DOUBLE PRECISION mit einem Ausdruck vom Typ COMPLEX sind verboten.

Beispiel:

```
1/3 .GT. 0.25
```

Dieser Vergleichsausdruck liefert den Wert .FALSE., weil das Ergebnis der ganzzahligen Division 0 ist. 0 wird vor dem Vergleich in den Typ REAL konvertiert.
Beim Vergleich von Zeichenketten wird vor dem Vergleich die kürzere der beiden Zeichenketten rechts mit Leerzeichen aufgefüllt, bis beide Ketten die gleiche Länge haben. Beim Vergleich entscheidet von links ausgehend das erste in beiden Ketten unterschiedliche Zeichen, welche Beziehung zwischen den Ketten besteht.

Beispiel:

```
'AB' .GE. 'AC' // '3'
```

Dieser Vergleichsausdruck hat den Wert .FALSE., da B kleiner als C ist und somit gilt:

'AB' < 'AC3'

Ein Zeichen ist kleiner, wenn es in der Sortierfolge des verwendeten Codes eher auftritt. Das bedeutet, daß die Ergebnisse von Zeichenkettenvergleichen von der Implementierung abhängen (siehe [1]).

FORTRAN77 verlangt eine der beiden Codeeigenschaften:

```
-----   -------------------------------   ----------------
! b ! < ! A < B < C   ...     X < Y < Z ! < ! 0 < 1  ...  9 ! oder
-----   -------------------------------   ----------------

-----   ----------------   -------------------------------
! b ! < ! 0 < 1  ...  9 ! < ! A < B < C     ...     X < Y < Z !
-----   ----------------   -------------------------------
```

2.7.4. Logische Ausdrücke

Logische Ausdrücke werden aus logischen Operanden und Operatoren gebildet. Ihre Berechnung liefert ein Ergebnis vom Typ LOGICAL mit dem Wert .TRUE. oder .FALSE..

Logische Operanden sind:
- die Konstanten .TRUE. und .FALSE.,
- symbolische Namen von logischen Konstanten,
- Bezugnahmen auf logische Variablen,
- Bezugnahmen auf logische Feldelemente,
- Bezugnahmen auf logische Funktionen,
- Vergleichsausdrücke und
- logische Ausdrücke, bei Bedarf in Klammern eingeschlossen

Die logischen Operatoren und ihre Bedeutung sind:

Operator	Beispiel	Bedeutung
.NOT.	.NOT. A	Logische Negation, einziger unärer Operator; Das Ergebnis ist .TRUE., wenn A den Wert .FALSE. hat, sonst .FALSE.
.AND.	A .AND. B	Logische Konjunktion. Das Ergebnis ist .TRUE., wenn A und B den Wert .TRUE. haben, sonst .FALSE..
.OR.	A .OR. B	Logische Disjunktion (inklusives Oder). Das Ergebnis ist .FALSE., wenn A und B den Wert .FALSE. haben, sonst .TRUE..
.EQV.	A .EQV. B	Logische Äquivalenz. Das Ergebnis ist .TRUE., wenn A und B denselben Wert haben, sonst .FALSE..
.NEQV.	A .NEQV. B	Exclusives Oder. Das Ergebnis ist .TRUE., wenn A und B verschiedene Werte haben, sonst .FALSE..

Ein logischer Ausdruck ist eine Folge logischer Operanden und Operatoren, die mit einem Operanden endet. Der logische Ausdruck beginnt mit .NOT. oder einem Operanden. Zwei Operanden dürfen nicht unmittelbar aufeinander folgen. Zwei logische Operatoren dürfen nur aufeinander folgen, wenn der zweite .NOT. und der erste von .NOT. verschieden ist.

Beispiele:

```
(A .OR. B(D)) .AND. .NOT. C .EQV. .TRUE.
I*J+K*L   .EQ. J+L*I .AND. .NOT. K*L .LE. M
```

Da als logische Operanden auch Vergleichsausdrücke erlaubt sind, können logische Ausdrücke alle Ausdrucksoperatoren enthalten. Ihr Rang gibt die Reihenfolge an, in welcher die Operatoren ausgeführt werden.

Operator	Rang
**	1 (höchster Rang)
*,/	2
+,-,//	3
Vergleichsoperatoren	4
.NOT.	5
.AND.	6
.OR.	7
.EQV.,.NEQV.	8 (niedrigster Rang)

Operatoren von gleichem Rang werden mit Ausnahme der Potenzierung in Links-Rechts-Folge ausgeführt. Wie bei arithmetischen Ausdrücken kann diese normale Reihenfolge der Operationen durch Klammern unterdrückt werden. Für + und - sowie * und / kann der Rechner eine algebraisch gleichwertige Reihenfolge benutzen (siehe Abschnitt 2.7.1.). Analog dazu darf der Rechner auch logische Operatoren gleichen Ranges in einer logisch gleichwertigen Reihenfolge verarbeiten, d.h. A .AND. B .AND. C darf als A .AND. (B .AND. C) berechnet werden.
Die obigen Beispiele werden wie folgt berechnet:

```
((A .OR. B(D)) .AND. (.NOT. C)) .EQV. .TRUE.
(((I*J)+(K*L)) .EQ. (J+(L*I))) .AND. (.NOT.  ((K*L) .LE. M))
```

Der Rechner muß nur die Operanden eines logischen Ausdrucks berechnen, die Einfluß auf das Ergebnis haben. Im ersten Beispiel kann auf den Aufruf der Funktion B(D) verzichtet werden, wenn A den Wert .TRUE. hat. Das muß der Programmierer berücksichtigen, wenn der Funktionsaufruf Seiteneffekte wie beispielsweise die Veränderung von Variablen im COMMON-Block hat.

2.7.5. Auflösung von Ausdrücken

Alle Variablen, Feldelemente, Funktionen oder Teilketten, auf die in einem Ausdruck Bezug genommen wird, müssen zum Zeitpunkt der Bezugnahme definiert sein. Ein Operand vom Typ INTEGER darf keine Anweisungsmarke als Wert haben. In Zeichenketten oder Teilketten müssen alle Zeichen definiert sein, auf die der Ausdruck Bezug

nimmt.
Enthält eine Anweisung mehrere Funktionsbezugnahmen, so darf der Rechner die Funktionsaufrufe in beliebiger Reihenfolge ausführen. Deshalb müssen die Funktionswerte unabhängig von der Reihenfolge sein, die der Rechner für die Funktionsaufrufe wählt. Eine Ausnahme hiervon bilden die logische IF-Anweisung und Argumentlisten, die wiederum Funktionsbezugnahmen enthalten. Wenn in der Anweisung

```
Y = F(G(X))
```

F und G Funktionen sind, so muß G vor dem Aufruf von F berechnet werden.
Ein Funktionsaufruf in einer Anweisung darf den Wert irgendeiner anderen Größe, die in derselben Aweisung auftritt, nicht verändern. Ein Funktionsaufruf darf auch den Wert einer Größe im COMMON-Block nicht verändern, wenn diese Größe Einfluß auf einen anderen Funktionswert in derselben Anweisung hat. Jedoch darf ein Funktionsaufruf im Testausdruck der logischen IF-Anweisung solche Größen verändern, die in der Anweisung auftreten, welche ausgeführt wird, wenn der Testausdruck den Wert .TRUE. hat.
Wenn eine Funktionsbezugnahme bewirkt, daß ein aktuelles Argument dieser Funktion definiert wird, darf dieses Argument oder eine damit assoziierte Größe nicht noch einmal in derselben Anweisung auftreten. Zum Beispiel sind die Anweisungen

```
A(I) = F(I)        und
Y = G(X) + X
```

verboten, wenn die Bezugnahme auf F das Argument I und die Bezugnahme auf G das Argument X definiert.

2.7.6. Konstantenausdrücke

Konstantenausdrücke sind Ausdrücke, in denen nur Konstanten oder symbolische Namen von Konstanten als Operanden auftreten. Exponenten müssen in Konstantenausdrücken vom Typ INTEGER sein. Teilketten von Zeichenkettenkonstanten sind in Konstantenausdrücken nicht erlaubt.

3. Vereinbarungsanweisungen

Mit Hilfe der Vereinbarungsanweisungen besteht die Möglichkeit, benötigte Eigenschaften von Variablen und Feldern festzulegen. Solche Eigenschaften sind Datentyp, Speicherplatzbedarf, Geltungsbereich, Anfangswerte usw.
Vereinbarungsanweisungen müssen am Anfang einer Programmeinheit stehen. Ihre Reihenfolge untereinander ist in Abschnitt 1.4. beschrieben.

3.1. PROGRAM-Anweisung

Diese Anweisung kann wahlweise benutzt werden. Mit ihr kann das Hauptprogramm mit einem Namen versehen werden.
Die PROGRAM-Anweisung hat die Form:

```
PROGRAM name
```

wobei 'name' der symbolische Name des Hauptprogrammes ist, in dem die PROGRAM-Anweisung auftritt.
Der symbolische Name des Hauptprogramms ist global zum ausführbaren Programm und damit zu allen dazugehörigen Programmeinheiten. Er darf nicht der Name einer externen Prozedur, eines BLOCKDATA-Unterprogrammes oder eines COMMON-Bereiches sein.

3.2. IMPLICIT-Anweisung

Eine IMPLICIT-Anweisung wird benutzt, um die Standardtypvereinbarung innerhalb einer Programmeinheit zu verändern.
Die IMPLICIT-Anweisung hat die Form:

```
IMPLICIT  typ(b[,b]...)[,typ(b[,b]...)] ...
```

Dabei ist

typ - einer der in Tabelle 3 angegebenen Datentypen (außer CHARACTER *(*)) und

b - entweder ein einzelner Buchstabe oder ein Intervall von einzelnen Buchstaben in alphabetischer Reihenfolge. Ein Intervall wird wie folgt dargestellt:

```
bl - bn
```

bl - erster Buchstabe des Intervalls
bn - letzter Buchstabe des Intervalls
Die Codierung bl - bn ist identisch mit
bl, b2, b3,... bn.

Die Liste (b[,b]...) gibt eine Menge von Buchstaben an. Alle symbolischen Namen, die mit einem Buchstaben aus dieser Menge beginnen, erhalten den davor angegebenen Datentyp.

Beispiel:

```
IMPLICIT INTEGER (I,J,K,L,M,N), REAL (A-H, O-Z)
```

Diese Anweisung stellt die in Abschnitt 2.1. beschriebene Standardtypvereinbarung dar.
Eine IMPLICIT-Anweisung hat keinerlei Einfluß auf die Datentypen von Standardfunktionen.

3.3. Typvereinbarungen

Die Typvereinbarungsanweisungen legen explizit den Datentyp (siehe Abschnitt 2.1.) von angegebenen symbolischen Namen fest. Es gibt zwei Arten von Typvereinbarungen:

- numerische und logische Typvereinbarungen
- Zeichenketten-Typvereinbarungen

Die Stellung der Typvereinbarungsanweisungen innerhalb einer Programmeinheit ist in Abschnitt 1.4. beschrieben. Für beide Arten der Typvereinbarungen gilt:

- Die Typvereinbarungen sind nur innerhalb einer Programmeinheit gültig.
- Der Datentyp eines symbolischen Namens kann nur einmal innerhalb einer Programmeinheit explizit vereinbart werden .
- Eine Typvereinbarung kann nicht den Datentyp einer Standardfunktion ändern. Stimmt der Datentyp der Standardfunktion mit dem zu vereinbarenden Datentyp überein, so darf der symbolische Name einer Standardfunktion angegeben werden.
- Wird anstelle eines symbolischen Namens eine Feldvereinbarung (siehe Abschnitt 2.5.) angegeben, so wird nicht nur der Datentyp, sondern auch die Dimensions- und die Elementezahl des Feldes festgelegt.

3.3.1. Numerische und logische Typvereinbarungen

Numerische und logische Typvereinbarungen haben die Form:

```
typ  sn[,sn] ...
```

Dabei ist

typ - ein beliebiger, von CHARACTER verschiedener Datentyp (siehe Abschnitt 2.1., Tabelle 3)
sn - der symbolische Name einer Variablen, einer Konstanten, eines Feldes, einer Anweisungsfunktion, eines FUNCTION-Unterprogramms, eines FUNCTION-Eingangspunktes, einer externen Prozedur oder eine Feldvereinbarung.

Beispiele:

```
INTEGER*2 A, B, FELD(10)
REAL      R, O, MATRIX(10,20)
```

3.3.2. Zeichenketten-Typvereinbarungen

Die Anweisung für die Zeichenketten-Typvereinbarungen hat die Form:

```
CHARACTER [*länge[,]] sn[*lang][,sn[*lang]] ...
```

Dabei ist

sn - wie sn in der numerischen oder logischen Typvereinbarung
länge,
lang - eine vorzeichenlose, von Null verschiedene ganzzahlige Konstante, ein in Klammern eingeschlossener ganzzahliger Konstantenausdruck oder ein in Klammern eingeschlossener Stern (*).

Wird CHARACTER*länge benutzt, so ist 'länge' die Standardlängenangabe für die folgende Liste von symbolischen Namen. Besitzt ein Element dieser Liste keine Längenangabe, so ist 'länge' die Elementlänge. Die Form 'sn*lang' bewirkt, daß für dieses Listenelement der Standardwert aus CHARACTER*länge überschrieben und 'lang' als Länge zugeordnet wird.
Eine Längenangabe mit Stern (z.B. CHARACTER*(*)) legt fest, daß einem FUNCTION-Namen oder einem formalen Parameter die aktuelle Länge der entsprechenden Funktionsbezugnahme bzw. des aktuellen Parameters zugeordnet wird.
Eine (*)-Spezifikation für den symbolischen Namen einer Konstanten (siehe Abschnitt 3.11.) bedeutet, daß die symbolische Konstante

die aktuelle Länge der Konstanten annimmt, die sie darstellt. Ist keine Längenspezifikation angegeben, so wird standardmäßig 1 angenommen.

Beispiel:

```
SUBROUTINE CHAR(PARM)
PARAMETER (LAENGE = 8)
CHARACTER TEIL(15), PARM*(*)
CHARACTER*4 ANFANG, ENDE*(4+LAENGE)
CHARACTER*32 FELD1(100), FELD2(100)*2
 .
 .
 .
```

3.4. DIMENSION-Anweisung

Die DIMENSION-Anweisung definiert die Anzahl der Dimensionen von Feldern und die Anzahl der Elemente in jeder Dimension.
Die DIMENSION-Anweisung hat die Form:

```
DIMENSION sn(dim)[,sn(dim)] ...
```

Dabei ist

sn(dim) - eine Feldvereinbarung (siehe Abschnitt 2.5.),
sn - der symbolische Name eines Feldes,
dim - eine Dimensionsvereinbarung.

Die DIMENSION-Anweisung veranlaßt, daß für jedes Feldelement in jeder Dimension Speicherplatz reserviert wird. Wieviel Speicherplatz durch das Feld belegt wird, hängt vom Datentyp ab.
Weitere Informationen über Felder und über die Anordnung und den Speicherplatz von Feldelementen können dem Abschnitt 2.5. entnommen werden.
Außer in der DIMENSION-Anweisung konnen Feldvereinbarungen auch in Typvereinbarungen und in COMMON-Anweisungen benutzt werden. Dabei ist jedoch darauf zu achten, daß in jeder Programmeinheit ein Feldname in nur einer Feldvereinbarung benutzt werden darf.

3.5. COMMON-Anweisung

Die COMMON-Anweisung ermöglicht es, daß Objekte in unterschiedlichen Programmeinheiten gemeinsamen Speicherplatz benutzen. Somit können auch ohne Parameter (siehe Abschnitt 5.) Daten zwischen verschiedenen Programmeinheiten ausgetauscht werden. Durch COMMON-Anweisungen werden gemeinsame Speicherbereiche (COMMON-Bereiche) vereinbart und die Objekte, die darin gespeichert werden

sollen, festgelegt. Diese COMMON-Bereiche dürfen in allen Programmeinheiten eines ausführbaren Programms benutzt werden, in denen sie durch COMMON-Anweisungen vereinbart wurden. Ein und derselbe COMMON-Bereich kann in den einzelnen Programmeinheiten für unterschiedliche Objekte benutzt werden. Dadurch können Daten der einzelnen Programmeinheiten speicherplatzsparend überlagert werden.
Die COMMON-Anweisung hat die Form:

```
COMMON [ / [cbn] / ] nlist [[,] / [cbn] / nlist] ...
```

Dabei ist

cbn - der Name eines COMMON-Bereiches. Wenn cbn angegeben ist, wird der entsprechende COMMON-Bereich als benannter COMMON-Bereich bezeichnet. Wenn cbn weggelassen wird, wird der unbenannte COMMON-Bereich vereinbart. Wenn der unbenannte COMMON-Bereich am Anfang einer COMMON-Anweisung vereinbart wird, können die Zeichen "//" weggelassen werden.

nlist - eine Liste von Variablennamen, Feldnamen und Feldvereinbarungen. Jeder Variablen- und Feldname oder jede Feldvereinbarung darf nur einmal in einer solchen Liste innerhalb einer Programmeinheit auftreten. Die Namen von formalen Parametern sowie Variablennamen in FUNCTION-Unterprogrammen, die gleichzeitig auch FUNCTION- oder Eintrittspunktnamen sind, sind in der Liste nicht erlaubt.

Ein benannter oder unbenannter COMMON-Bereich kann beliebig oft in einer oder mehreren COMMON-Anweisungen einer Programmeinheit auftreten. Jedes weitere Auftreten eines COMMON-Bereichs bewirkt dessen Fortsetzung. Alle Objekte, die zu einem COMMON-Bereich gehören, werden in der Reihenfolge ihres Auftretens in diesem COMMON-Bereich angeordnet. Dabei ist darauf zu achten, daß Objekte des Datentyps CHARACTER nicht gemischt mit Objekten eines von CHARACTER verschiedenen Datentyps in einem COMMON-Bereich auftreten dürfen.
Die Länge eines COMMON-Bereiches in einer Programmeinheit ist gleich der Summe des Speicherbedarfs für alle Objekte, die durch COMMON- oder EQUIVALENCE-Anweisungen (siehe Abschnitt 3.6.) für diesen COMMON-Bereich vereinbart wurden.
In einer Programmeinheit können ein unbenannter und mehrere benannte COMMON-Bereiche, die alle ihren eigenen Namen haben, verwendet werden. Benannte COMMON-Bereiche mit dem gleichen Namen belegen in allen Programmeinheiten, in denen sie verwendet werden, denselben Speicherbereich und müssen mit der gleichen Länge vereinbart werden. Unbenannte COMMON-Bereiche innerhalb eines ausführbaren Programms brauchen nicht die gleiche Länge zu haben.

Objekten, die zum unbenannten COMMON-Bereich gehören, können keine Anfangswerte zugewiesen werden. Objekten eines benannten COMMON-Bereichs dürfen nur in einem BLOCK DATA-Unterprogramm Anfangswerte zugewiesen werden. Die Ausführung einer RETURN- oder END-Anweisung kann bewirken, daß Objekte in benannten COMMON-Bereichen als undefiniert zu betrachten sind (siehe Abschnitt 5.2.6.). Auf Objekte im unbenannten COMMON-Bereich haben diese beiden Anweisungen keinen Einfluß.

3.6. EQUIVALENCE-Anweisung

Die EQUIVALENCE-Anweisung wird zur Zuordnung gemeinsamen Speicherplatzes für zwei oder mehrere Objekte einer Programmeinheit benutzt.
Sie hat die Form:

```
EQUIVALENCE (nlist)[,(nlist)] ...
```

Dabei ist

nlist - eine Liste von Variablen-, Feld-, Feldelement- und Teilkettennamen. Außer beim Datentyp CHARACTER können in dieser Liste unterschiedliche Datentypen auftreten. Jede Liste muß aus mindestens zwei Namen bestehen. Jeder Index- oder Teilkettenausdruck in einer Liste nlist muß ein ganzzahliger Konstantenausdruck sein. Die Namen von formalen Parametern sowie Variablennamen in FUNCTION-Unterprogrammen, die gleichzeitig auch FUNCTION- oder Eintrittspunktnamen sind, sind nicht erlaubt.

Die EQUIVALENCE-Anweisung bewirkt, daß die Speicherplatzfolgen aller in einer Liste nlist angegebenen Objekte an einer gemeinsamen Speicherstelle beginnen. Durch diese gemeinsame Speicherplatzbenutzung gelten die Objekte als assoziiert innerhalb der Programmeinheit.
Eine EQUIVALENCE-Anweisung darf sich nicht selbst, keiner beliebigen vorher festgelegten Äquivalenz und keiner anderen Festlegung der Sprache widersprechen.

Beispiel:

```
DIMENSION X(10)
REAL A(2)
DOUBLE PRECISION D(2)
EQUIVALENCE (X(1),Y), (Y,Z), (Z,X(3))
EQUIVALENCE (A(1),D(1)), (A(2),D(2))
```

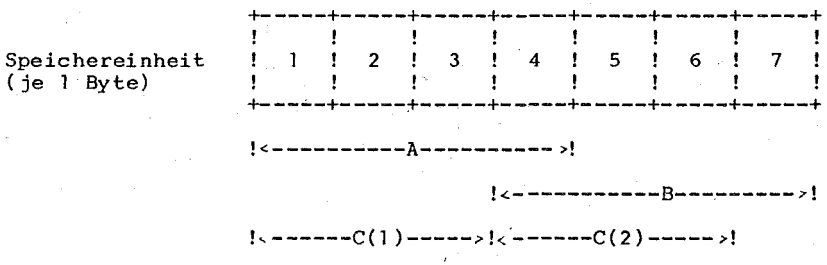
Bei der ersten EQUIVALENCE-Anweisung wird der Speicherplatz für die zwei ersten Listen so zugewiesen, daß die Variablen Y und Z sowie das Feldelement X(1) denselben Speicherplatz erhalten. Die dritte EQUIVALENCE-Liste allerdings enthält die indirekte Äquivalenz des dritten Feldelementes mit dem ersten. Eine solche Äquivalenz ist nicht möglich, da dem ersten und dritten Element eines Feldes nicht der gleiche Speicherplatz zugewiesen werden kann.
Die zweite EQUIVALENCE-ANWEISUNG ist ebenfalls nicht erlaubt, denn sie verstößt gegen das Prinzip, daß Feldelemente lückenlos aufeinander folgen müssen.

EQUIVALENCE-Anweisung für Zeichenkettenvariable

Ein Objekt vom Datentyp CHARACTER kann nur mit Objekten vom Datentyp CHARACTER assoziiert sein. Die Längen der Objekte müssen dabei nicht notwendigerweise gleich sein.
Die Anweisungsfolge

```
CHARACTER A*4, B*4, C(2)*3
EQUIVALENCE (A, C(1)), (B, C(2))
```

führt zu folgender Assoziation von A, B und C



EQUIVALENCE-Anweisung für Felder

In einer EQUIVALENCE-Anweisung braucht für das erste Element eines Feldes statt des Feldelementnamens nur der Feldname angegeben zu werden. Die obige EQUIVALENCE-Anweisung kann also auch als

```
EQUIVALENCE (A,C), (B,C(2))
```

geschrieben werden.
Das ist der einzige Fall, in welchem in einer EQUIVALENCE-Anweisung ein Feldname benutzt werden darf.

EQUIVALENCE-Anweisung und COMMON-Bereiche

In der Liste nlist einer EQUIVALENCE-Anweisung darf ein Name der Name eines Objektes aus einem COMMON-Bereich sein. Alle weiteren Elemente aus nlist werden in diesem Falle im selben COMMON-Bereich angeordnet.
Eine direkte oder indirekte Assoziierung von zwei Elementen aus verschiedenen COMMON-Bereichen ist nicht gestattet. Es ist möglich, einen COMMON-Bereich nach rechts zu erweitern. Eine Erweiterung nach links ist nicht möglich.

Beispiele:

```
DIMENSION G(4)
COMMON Q, B, F
EQUIVALENCE (B,G)
```

Die COMMON-Anweisung vereinbart einen COMMON-Bereich, der die Variablen Q, B und F enthält. Danach wird die Äquivalenz der Variablen B mit dem ersten Feldelement von G, also G(1) festgelegt. Das führt zur Assoziation von F und G(2) und zur Erweiterung des COMMON-Bereiches um die Feldelemente G(3) und G(4).
Dagegen ist die Anweisungsfolge

```
COMMON /X/A
REAL B(2)
EQUIVALENCE (A, B(2))
```

nicht gestattet, denn sie würde den COMMON-Bereich X nach links um das Element B(1) erweitern.

3.7. SAVE-Anweisung

Die SAVE-Anweisung wird benutzt, um zu sichern, daß ein Objekt nach der Ausführung einer RETURN- oder END-Anweisung in einem Unterprogramm definiert bleibt. Ist ein solches Objekt in einem COMMON-Bereich enthalten, so kann es jedoch durch eine andere Programmeinheit neu definiert oder erstmals definiert werden.
Die SAVE-Anweisung hat folgende Form:

```
SAVE [a [,a] ...]
```

Dabei ist

a - der Name eines benannten COMMON-Bereiches (eingeschlossen in Schrägstriche), ein Variablenname oder ein Feldname. Die Namen von formalen Parametern, von Prozeduren und von Objekten in einem COMMON-Bereich dürfen nicht angegeben werden.

Eine SAVE-Anweisung ohne eine Liste ist identisch mit einer SAVE-Anweisung, in der alle erlaubten Objekte innerhalb der Programm-

einheit angegeben sind. In einem Hauptprogramm ist die SAVE-Anweisung wirkungslos.
Wird in einem Unterprogramm eines ausführbaren Programmes der Name eines COMMON-Bereiches in einer SAVE-Anweisung angegeben, so muß für ihn in jedem anderen Unterprogramm, in dem dieser COMMON-Bereich benutzt wird, eine SAVE-Anweisung angegeben werden. Ist dagegen ein benannter COMMON-Bereich im Hauptprogramm vereinbart, so sind die gegenwärtigen Werte der Objekte des COMMON-Bereiches in jedem Unterprogramm verfügbar, in welchem dieser benannte Bereich angegeben ist. In diesem Falle ist die SAVE-Anweisung im Unterprogramm wirkungslos.

3.8. EXTERNAL-Anweisung

Die EXTERNAL-Anweisung wird verwendet, um symbolische Namen als Namen externer Prozeduren oder BLOCK DATA-Unterprogramme zu vereinbaren.
Die EXTERNAL-Anweisung hat die Form:

```
EXTERNAL  sn[,sn] ...
```

Dabei ist

sn - der Name einer externen Prozedur, eines BLOCK DATA-Unterprogrammes oder eines formalen Parameters, der mit dem Namen eines Unterprogrammes verbunden ist.

Der Name einer externen Prozedur, eines BLOCK DATA-Unterprogrammes oder eines formalen Parameters, der mit dem Namen eines Unterprogrammes verbunden ist, darf nur dann als aktueller Parameter verwendet werden, wenn er zuvor in derselben Programmeinheit in einer EXTERNAL-Anweisung aufgetreten ist.
Der Name einer Anweisungsfunktion darf nicht in einer EXTERNAL-Anweisung auftreten. Wird der Name einer Standardfunktion angegeben, so werden alle Bezugnahmen auf diese Standardfunktion in Bezugnahmen auf ein externes Unterprogramm mit diesem Namen umgewandelt, und die Standardfunktion ist in dieser Programmeinheit nicht verfügbar. Der Programmierer ist somit dafür verantwortlich, daß das entsprechende Unterprogramm mit diesem Namen existiert. Jeder symbolische Name darf nur einmal in einer EXTERNAL-Anweisung der betreffenden Programmeinheit auftreten.

3.9. INTRINSIC-Anweisung

Die INTRINSIC-Anweisung wird benutzt, um einen symbolischen Namen als Namen einer Standardfunktion zu vereinbaren. Eine INTRINSIC-Anweisung ist nur dann notwendig, wenn spezielle Namen von Stan-

dardfunktionen als aktuelle Parameter benutzt werden sollen.
Die INTRINSIC-Anweisung hat die Form:

INTRINSIC sn [,sn] ...

Dabei ist

sn - der Name einer Standardfunktion.

Wird ein spezieller Name einer Standardfunktion (siehe Anlage 2) in einer Programmeinheit als aktueller Parameter benutzt, so muß er in dieser Programmeinheit in einer INTRINSIC-Anweisung angegeben sein. Alle generischen Namen und die speziellen Namen der Standardfunktionen für die Datentypumwandlung, die Bestimmung des Maximums und Minimums von numerischen Werten sowie für den lexikalischen Vergleich dürfen nicht als aktuelle Parameter benutzt werden.
Das Auftreten eines generischen Namens einer Standardfunktion in einer INTRINSIC-Anweisung berührt in keiner Weise die generischen Eigenschaften dieses Namens.
Jeder symbolische Name darf nur einmal in den INTRINSIC-Anweisungen einer Programmeinheit auftreten. Die Angabe eines symbolischen Namens in einer EXTERNAL- und in einer INTRINSIC-Anweisung ist nicht gestattet.

Beispiel für EXTERNAL-/INTRINSIC-Anweisungen

Hauptprogramm:

```
      .
      .
      .
      EXTERNAL COTAN
      INTRINSIC SIN, COS
      .
      .
      .
      CALL TRIGON (ARGU, SIN, SINUS)
      .
      .
      .
      CALL TRIGON (ARGU, COS, CSINUS)
      .
      .
      .
      CALL TRIGON (ARGU, COTAN, COTANG)
      .
      .
      .
      END
```

Unterprogramme:

```
SUBROUTINE TRIGON (ARG, FUNC, ERG)
ERG = FUNC(ARG)
RETURN
END

FUNCTION COTAN (ARG)
COTAN = COS(ARG)/SIN(ARG)
RETURN
END
```

Wird TRIGON mit SIN oder COS als zweitem aktuellen Parameter aufgerufen, so bezieht sich FUNC(ARG) auf die Standardfunktion SIN bzw. COS. Der Aufruf mit COTAN als zweitem aktuellen Parameter bewirkt, daß ERG = FUNC(ARG) sich auf das FUNCTION-Unterprogramm COTAN bezieht.

3.10. DATA-Anweisung

Die DATA-Anweisung wird benutzt, um Variable, Felder, Feldelemente oder Teilketten durch Zuweisung von Anfangswerten zu definieren. Die DATA-Anweisung zählt zu den nichtausführbaren Anweisungen, kann aber nach den Vereinbarungsanweisungen an beliebiger Stelle in einer Programmeinheit auftreten. Während alle in DATA-Anweisungen auftretenden Objekte zu Beginn der Ausführung eines FORTRAN-Programms als definiert gelten, sind alle Objekte, die keinen Anfangswert haben oder nicht mit einem Anfangswert assoziiert sind, zu diesem Zeitpunkt undefiniert.
Die DATA-Anweisung hat die Form:

```
DATA snlist/kliste/[[,]snlist/kliste/] ...
```

Dabei ist

snlist - eine Liste von Variablen-, Feld-, Feldelement- und Teilkettennamen und impliziten DO-Listen. Die Namen werden durch Komma getrennt. Jeder Indexausdruck muß, außer für implizite DO-Variablen, ebenso wie jeder Teilkettenausdruck, ein ganzzahliger Konstantenausdruck sein.
Formale Parameter, Objekte im unbenannten COMMON-Bereich oder Objekte, die mit Objekten im unbenannten COMMON-Bereich assoziiert sind, und Variablen in einem FUNCTION-Unterprogramm , die den gleichen Namen wie das FUNCTION-Unterprogramm oder eines seiner Eintrittspunkte haben, sind verboten. Namen von Objekten in benannten COMMON-Bereichen dürfen nur in-

nerhalb eines BLOCK DATA-Unterprogramms auftreten.

kliste - eine Liste von Konstanten, wobei jeder Konstanten j* vorangestellt werden kann, was bedeutet, daß diese Konstante j-mal wiederholt werden soll. Der Wiederholungsfaktor j muß eine von Null verschiedene, vorzeichenlose ganzzahlige Konstante oder der symbolische Name einer solchen Konstanten sein (siehe Abschnitt 3.11.).

Die Abarbeitung einer DATA-Anweisung erfolgt elementweise von links nach rechts. Dabei ist darauf zu achten, daß zwischen der Gesamtzahl der Elemente in snlist und der Gesamtzahl der in der entsprechenden Konstantenliste kliste angegebenen Konstanten (unter Berücksichtigung der Wiederholungsfaktoren j) eine eindeutige Beziehung besteht. Befindet sich in snlist ein Feldname ohne Index, so muß in kliste für jedes Feldelement eine Konstante angegeben werden. Die Reihenfolge der Feldelemente ist in Abschnitt 2.5.3. beschrieben.
Die Anfangswertzuweisung erfolgt nach den Regeln der Ergibtanweisungen. Dabei ist zu beachten, daß beim Datentyp CHARACTER und LOGICAL die entsprechenden Konstanten vom gleichen Datentyp sein müssen. Sind die Objekte in snlist vom Typ INTEGER, REAL, DOUBLE PRECISION oder COMPLEX, so müssen die zugehörigen Konstanten ebenfalls einen arithmetischen Datentyp besitzen. Wenn es erforderlich ist, wird die Konstante in den entsprechenden Zieltyp umgewandelt.
Einer Variablen, einem Feldelement oder einer Teilkette darf nur einmal in einem ausführbaren Programm ein Anfangswert zugewiesen werden.
Sind zwei oder mehrere Objekte miteinander assoziiert, so darf nur einem davon mittels der DATA-Anweisung ein Anfangswert zugewiesen werden.

Implizite DO-Listen in DATA-Anweisungen

Die implizite DO-Liste einer DATA-Anweisung hat die Form:

(fdlist, iv = ikaus1, ikaus2[,ikaus3])

Dabei ist

fdlist - eine Liste von Feldelementnamen und impliziten DO-Listen

iv - der symbolische Name einer ganzzahligen Variablen, die als implizite DO-Variable bezeichnet wird.

ikausj - ein ganzzahliger Konstantenausdruck, der auch implizite DO-Variablen von solchen impliziten DO-Listen enthalten kann, die die betreffende DO-Liste umfassen.

Der Gültigkeitsbereich einer impliziten DO-Liste ist die Liste fdlist. Wie bei einer DO-Schleife (siehe Abschnitt 4.4.) werden ein Schleifenzähler und die Werte der impliziten DO-Variablen aus den Ausdrücken ikausj abgeleitet, wobei, abweichend von der DO-Anweisung, der Schleifenzähler stets positiv sein muß. Die implizite DO-Liste bewirkt eine Iteration ähnlich der DO-Anweisung. Ein Iterationsschritt bedeutet hier die Wiederholung aller Elemente von fdlist, wobei die implizite DO-Variable iv jedesmal durch ihren aktuellen Wert ersetzt wurde.

Beispielsweise ist damit

```
(A(i), i=3, -2, -2)
```

eine andere Schreibweise für

```
A(3), A(1), A(-1)
```

Der Gültigkeitsbereich einer impliziten DO-Variablen ist die DATA-Anweisung, in der sich die implizite DO-Liste befindet, d.h. ihr Auftreten hat keinerlei Einfluß auf eine Variable mit dem gleichen Namen in der Programmeinheit.

Beispiel:

```
DATA ((X(J,I), I=1,J), J=1,5) /15*0/
```

3.11. PARAMETER-Anweisung

Die PARAMETER-Anweisung wird benutzt, um innerhalb einer Programmeinheit einer Konstanten einen symbolischen Namen zu geben. Sie hat die Form:

```
PARAMETER (sn = ka[,sn = ka] ...)
```

Dabei ist

sn - ein symbolischer Name
ka - ein Konstantenausdruck (siehe Abschnitt 2.7.6.)

Jeder symbolische Name sn wird nach den Regeln der Ergibtanweisung mit dem Wert des Konstantenausdrucks ka definiert. Dabei wird der Datentyp entweder standardmäßig oder durch Typvereinbarungsanweisungen festgelegt. Hat der symbolische Name einen der Datentypen INTEGER, REAL, DOUBLE PRECISION oder COMPLEX, so muß der entsprechende Ausdruck ka ein arithmetischer Konstantenausdruck sein, der eventuell in den Zieltyp umgewandelt wird. Für den Fall CHARACTER oder LOGICAL müssen Quelle und Ziel datentypmäßig übereinstimmen. Soll sn der symbolische Name einer Konstanten vom Datentyp CHARACTER sein, so muß, wenn nicht standardmäßig die Länge 1 verwendet werden soll, vorher die Länge mittels einer IMPLICIT oder Typvereinbarungsanweisung (siehe Abschnitt 3.3.2.) festgelegt

werden. Eine (*)-Spezifikation für den symbolischen Namen einer Konstanten bedeutet dabei, daß die symbolische Konstante die aktuelle Länge der Konstanten annimmt, die sie darstellt.
Treten im Konstantenausdruck ka symbolische Namen von Konstanten auf, so müssen sie entweder vorher in der gleichen oder in einer vorangehenden PARAMETER-Anweisung definiert worden sein.
Der symbolische Name einer Konstanten darf weder als Teil einer Formatangabe noch als Teil einer anderen Konstanten (z.B. vom Datentyp COMPLEX) benutzt werden.

Beispiel:

```
      REAL PI, PIHALB
      DOUBLE PRECISION DPI, DPIHAL
      LOGICAL KENNZ
      CHARACTER*(*) LNAME
      CHARACTER*(10) NAME10
      PARAMETER (PI = 3.1415927,
                 DPI = 3.14159265358979323 8D0)
      PARAMETER (PIHALB = PI/2, DPIHAL = DPI/2,
                 NAME10 = ' ')
      PARAMETER (KENNZ = .FALSE.,
                 LNAME = '15 ZEICHEN LANG')
```

Dabei verkörpert NAME10 eine Konstante, die aus 10 Leerzeichen besteht.

4. Ausführbare Anweisungen

4.1. Definitionsanweisungen

Definitionsanweisungen definieren den Wert einer Variablen, eines Feldelementes oder einer Teilkette.

Es gibt folgende Definitionsanweisungen:
- arithmetische Ergibtanweisung
- logische Ergibtanweisung
- Zeichenketten-Ergibtanweisung
- ASSIGN-Anweisung

4.1.1. Arithmetische Ergibtanweisung

Die arithmetische Ergibtanweisung hat die Form:

v = e

Dabei ist

v - der Name einer numerischen Variablen oder eines numerischen Feldelementes
e - ein arithmetischer Ausdruck

Vor der eigentlichen Definition der Variablen oder des Feldelementes v, d.h. der Wertzuweisung, wird nach den in Abschnitt 2.7.1. beschriebenen Regeln der Wert des Ausdrucks e bestimmt und in den Datentyp von v umgewandelt.

Tabelle 5 Typumwandlung und Zuweisung

Datentyp von v	zugewiesener Wert
INTEGER	INT(e)
REAL	REAL(e)
DOUBLE PRECISION	DBLE(e)
COMPLEX	CMPLX(e)

Die Funktionen in der Spalte "zugewiesener Wert" sind generische Funktionen. Ihre Wirkungsweise wird in Anlage 2 beschrieben. Dabei ist zu beachten, daß die Anwendung von INT auf einen Gleitkommawert keine Rundung sondern ein Streichen des gebrochenen Anteils dieses Wertes bewirkt. So ist z.B. INT(-1.95)=-1.

4.1.2. Logische Ergibtanweisung

Die logische Ergibtanweisung hat die Form:

```
v = e
```

Dabei ist

- v - der Name einer logischen Variablen oder eines logischen Feldelementes
- e - ein logischer Ausdruck (siehe Abschnitt 2.7.4.)

Die Ausführung einer Ergibtanweisung beginnt mit der Auflösung des logischen Ausdrucks e, der nur den Wert .TRUE. oder .FALSE. haben kann. Anschließend wird dieser Wert der Variablen v zugewiesen.

4.1.3. Zeichenketten-Ergibtanweisung

Die Zeichenketten-Ergibtanweisung hat die Form:

```
v = e
```

Dabei ist

- v - der Name einer Variablen oder eines Feldelementes vom Typ CHARACTER oder einer Teilkette.
- e - ein Zeichenkettenausdruck (siehe Abschnitt 2.7.2.).

Nach Auflösung des Zeichenkettenausdrucks e erfolgt, von links nach rechts, die Definition von v mit dem Wert von e. Dabei ist zu beachten, daß es nicht gestattet ist, in e eine Bezugnahme auf v anzugeben.
Der Wert des Ausdrucks e und v können unterschiedliche Länge haben. In diesem Falle wird vor der Zuweisung der Wert von e auf die Länge von v gebracht. Ist v länger, so wird e rechtsbündig mit Leerzeichen aufgefüllt. Ist der Wert von e länger, so wird er von rechts auf die Länge von v gekürzt.
Ist v eine Teilkette, so definiert e nur diese Teilkette. Der Rest der Zeichenkettenvariablen bleibt unverändert.
Zum Zeitpunkt der Wertzuweisung an v muß vom Wert von e nur soviel definiert sein, wie zur Definition von v notwendig ist.

Beispiel:

```
CHARACTER, X*10, Y*4
X(2:3) = Y
```

Die Ergibtanweisung verlangt, daß die Teilkette Y(1:2) definiert sein muß. Die Teilketten X(1:1) und X(4:10) sind nach der Ausführung dieser Anweisung unverändert.

4.1.4. ASSIGN-Anweisung

Nur die ASSIGN-Anweisung erlaubt es, eine ganzzahlige Variable mit dem Wert einer Marke zu definieren. Diese Variable kann in einer folgenden gesetzten GOTO-Anweisung oder als FORMAT-Kennzeichnung in einer folgenden E/A-Anweisung benutzt werden.
Die Anweisung hat die Form:

```
ASSIGN m TO i
```

Dabei ist

m - eine Marke an einer ausführbaren oder einer FORMAT-Anweisung in der gleichen Programmeinheit, in welcher sich die ASSIGN-Anweisung befindet,
i - der Name einer ganzzahligen Variablen.

In ihrer Wirkungsweise entspricht sie einer arithmetischen Ergibtanweisung mit der Ausnahme, daß die Variable als Markenbezugnahme definiert wird und somit nicht in arithmetischen Ausdrücken benutzt werden darf.
Die ganzzahlige Variable, die mit dem Wert einer Marke definiert ist, kann mit dem gleichen oder einem anderen Markenwert oder einem ganzzahligen Wert neu definiert werden. Wird sie mit einem ganzzahligen Wert neu definiert, so kann sie in arithmetischen Ausdrücken verwendet werden.
So stellt zum Beispiel die Anweisung

```
ASSIGN 20 TO MARKE
```

eine Verbindung zwischen Marke und der Anweisungsmarke 20 her.
Eine nachfolgende Anweisung

```
MARKE = MARKE + 1
```

ist verboten, da Marke nicht als ganzzahlige Variable definiert ist. Die Ausführung der Anweisung

```
MARKE = 100
```

trennt Marke wieder von der Anweisungsmarke 20, versieht MARKE mit dem Status einer ganzzahligen Variablen und definiert diese mit dem Wert 100. Eine nachfolgende Anweisung

```
GOTO MARKE
```

ist somit nicht mehr erlaubt.

4.2. GOTO-Anweisungen

Die Anweisungen in einer Programmeinheit werden normalerweise in der Reihenfolge abgearbeitet, in der sie geschrieben sind. Die GOTO-Anweisungen ermöglichen es, die Steuerung an eine fest vorgegebene oder an eine vom Wert eines Ausdrucks abhängige ausführbare Anweisung in der gleichen Programmeinheit zu übergeben. Es gibt drei Arten der GOTO-Anweisung:

- die unbedingte GOTO-Anweisung
- die berechnete GOTO-Anweisung
- die gesetzte GOTO-Anweisung

4.2.1. Unbedingte GOTO-Anweisung

Um die Abarbeitung einer Programmeinheit mit einer bestimmten Anweisung fortzusetzen, muß eine unbedingte GOTO-Anweisung verwendet werden.
Sie hat die Form:

```
GOTO m
```

Dabei ist

m - die Marke einer ausführbaren Anweisung in der gleichen Programmeinheit, in welcher sich die GOTO-Anweisung befindet.

4.2.2. Berechnete GOTO-Anweisung

Bei diesem Typ der GOTO-Anweisung wird erst zur Ausführungszeit ermittelt, welcher Anweisung die Steuerung übergeben werden soll.
Die berechnete GOTO-Anweisung hat die Form:

```
GOTO (m[,m] ...) [,]i
```

Dabei ist

i - ein ganzzahliger Ausdruck
m - die Marke an einer ausführbaren Anweisung in der gleichen Programmeinheit wie die berechnete GOTO-Anweisung

Die Ausführung der berechneten GOTO-Anweisung beginnt mit der Auflösung des Ausdrucks i. Der Wert von i gibt an, daß der Anweisung die Steuerung übergeben wird, die mit der i-ten Marke in der Liste (m[,m]...) verbunden ist. Wenn n die Anzahl der Marken m in der Liste ist, so wird nur bei

```
1<=i<=n
```

die Steuerung an die entsprechende Anweisung übergeben. Gilt für den Wert von i

```
i<1 oder i n,
```

so wird die berechnete GOTO-Anweisung wie eine CONTINUE-Anweisung ausgeführt.

Beispiel:

```
GOTO (10,20,10,30), IAUSDR-6
```

4.2.3. Gesetzte GOTO-Anweisung

Die gesetzte GOTO-Anweisung hat die Form:

```
GOTO i[[,] (m[,m] ...)]
```

Dabei ist

- i - der Name einer ganzzahligen Variablen, die mittels einer ASSIGN-Anweisung mit dem Wert einer Marke definiert wurde (siehe Abschnitt 4.1.4.).
- m - die Marke an einer ausführbaren Anweisung in der gleichen Programmeinheit wie die gesetzte GOTO-Anweisung.
- (m[,m]...) - eine Liste von Marken, die anzeigt, mit welchen Markenwerten die ganzzahlige Variable i zur Ausführungszeit definiert sein darf.

Die Ausführung der gesetzten GOTO-Anweisung bewirkt, daß die Anweisung die Steuerung erhält, die den augenblicklichen Wert der Variablen i als Marke besitzt.

Beispiel:

```
GOTO MARKE,(10,20,30)
```

4.3. IF-Anweisungen

IF-Anweisungen erlauben bedingte Verzweigungen oder die bedingte Ausführung einer Anweisung bzw. eines Anweisungsblocks in Abhängigkeit vom Wert eines Ausdrucks.

4.3.1. Arithmetische IF-Anweisung

Die arithmetische IF-Anweisung hat die Form:

```
IF (e) s1,s2,s3
```

Hierbei ist

- e - ein arithmetischer Ausdruck vom Typ INTEGER, REAL oder DOUBLE PRECISION.
- s1,s2,s3 - die Marken von ausführbaren Anweisungen in derselben Programmeinheit.

Die Marken s1,s2 und s3 müssen sich nicht unbedingt auf verschiedene Anweisungen beziehen. Der Wert des Ausdrucks e wird berech-

net. Danach wird zu s1, s2 oder s3 verzweigt, jenachdem, ob der Wert von e kleiner, gleich oder größer als Null ist.

Beispiel:

```
IF (ZAHL/2*2-ZAHL) 1,2,1
```

Diese Anweisung verzweigt zu Anweisung 2, wenn die ganzzahlige Variable ZAHL gerade ist, zu Anweisung 1, wenn sie ungerade ist.

4.3.2. Logische IF-Anweisung

Die logische IF-Anweisung hat die Form:

```
IF (e) st
```

Hierbei ist

e - ein logischer Ausdruck.
st - eine ausführbare Anweisung. Nicht erlaubt sind an dieser Stelle DO-, Block-IF-, ELSEIF-, ELSE-, ENDIF- oder END-Anweisungen, sowie eine weitere logische IF-Anweisung.

Der Wert des Ausdrucks e wird berechnet. Ist der Wert .TRUE., wird die Anweisung st ausgeführt. Ist der Wert .FALSE., wird die Arbeit mit der ersten ausführbaren Anweisung hinter der logischen IF-Anweisung fortgesetzt. Enthält der Ausdruck e Funktionsbezugnahmen, so ist Abschnitt 2.7.5. zu beachten.

Beispiel:

```
IF (A .LT. 0) A = -A
```

Diese Anweisung berechnet den Absolutbetrag der Variablen A.

4.3.3. Block-IF-Anweisung

Die Block-IF-Anweisung wird zusammen mit der ELSEIF-, ELSE- und ENDIF-Anweisung benutzt, um die bedingte Ausführung von Anweisungsblöcken zu steuern.
Diese Anweisungen haben folgende Form:

```
IF (e) THEN
ELSE IF (e) THEN
ELSE
ENDIF
```

wobei e ein logischer Ausdruck ist.

Diese Anweisungen werden in Block-IF-Konstruktionen verwendet. Eine Block-IF-Konstruktion hat die Form:

```
IF (e) THEN
     IF-Block
[ELSE IF (e1) THEN ]
[     ELSEIF-Block1]
      .
      .
      .
[ELSE IF (en) THEN ]
[     ELSEIF-Blockn]
[ELSE              ]
[     ELSE-Block   ]
ENDIF
```

Die IF-, ELSEIF- und ELSE-Blöcke sind Folgen von Anweisungen, die durch die erste nachfolgende ELSEIF-, ELSE- oder ENDIF-Anweisung abgeschlossen werden. Diese Anweisung zählt nicht mehr zum vorangehenden Block. Ein ELSE-Block muß durch ENDIF abgeschlossen werden. Wenn ein Block eine Block-IF-Anweisung enthält, muß er die gesamte zugehörige Block-IF-Konstruktion enthalten. Enthält ein Block eine DO-Anweisung, so muß er auch die zugehörige Abschlußanweisung der DO-Schleife enthalten. Verzweigungen aus einem Block sind erlaubt, verboten sind aber alle Verzweigungen in einen Block oder von Block zu Block. Blöcke dürfen leer sein.
Die Verarbeitung einer Block-IF-Konstruktion beginnt mit der Berechnung des Ausdrucks e. Hat e den Wert .TRUE., werden die ausführbaren Anweisungen des IF-Blocks ausgeführt. Hat e den Wert .FALSE., wird der nächste logische Ausdruck (e1) berechnet. Sein Wert entscheidet, ob der zugehörige Block ausgeführt wird (.TRUE.), oder ob der nächste logische Ausdruck berechnet werden muß (.FALSE.). Wenn alle Ausdrücke den Wert .FALSE. haben, wird dieser Prozeß bis zur Berechnung des letzten Ausdrucks fortgesetzt. Bei .TRUE. wird dann der zugehörige Block ausgeführt. Bei .FALSE. wird der ELSE-Block ausgeführt, sofern die Block-IF-Konstruktion eine ELSE-Anweisung enthält. Das bedeutet, daß in einer Block-IF-Konstruktion höchstens ein Block ausgeführt wird. Genau ein Block wird ausgeführt, wenn ein ELSE-Block auftritt.
Nach Ausführung eines IF-, ELSEIF- oder ELSE-Blocks wird zur ersten ausführbaren Anweisung hinter der ENDIF-Anweisung verzweigt, sofern der Block nicht selbst als letzte Anweisung eine Verzweigung enthält. Der Sprung hinter die ENDIF-Anweisung erfolgt auch, wenn der Block leer ist oder keine ausführbare Anweisung enthält.
Die ELSEIF- und ELSE-Anweisungen können Marken haben. Bezugnahmen auf diese Marken sind jedoch verboten. Auch die ENDIF-Anweisung darf eine Marke haben. Sprünge zu dieser Marke sind jedoch nur aus Blöcken derselben Block-IF-Konstruktion erlaubt.

Beispiel:

```
          IF (X .EQ. 0E0) THEN
              Y = 0E0
              IF (I .LE. 0) CALL FEHLER
          ELSE
              IF (I .LT. 0) THEN
                  N = -I
              ELSE
                  N = I
              ENDIF
              Y = 1E0
              Z = X
        1     M = N
              N = N/2
              IF (2*N .NE. M) Y = Y*Z
              IF (N .NE. 0) THEN
                  Z = Z*Z
                  GOTO 1
              ELSE IF (I .LT. 0) THEN
                  Y = 1/Y
              ENDIF
          ENDIF
```

Dieses Beispiel berechnet Y = X**I für ganzzahlige Werte von I. Wenn X Null ist und der Exponent I nicht positiv ist, wird eine Fehlerroutine aufgerufen.

4.4. DO-Anweisung

Die DO-Anweisung steuert die wiederholte Abarbeitung einer Folge von Anweisungen, der sogenannten DO-Schleife. Sie legt Beginn und Ende der DO-Schleife fest und gibt an, wie oft diese Schleife zu durchlaufen ist.
Die DO-Anweisung hat die Form:

```
DO s[,] v = e1,e2[,e3]
```

Dabei ist

s - die Marke einer ausführbaren Anweisung. Diese Anweisung ist die Abschlußanweisung der DO-Schleife und muß physisch in derselben Programmeinheit folgen.

v - der Name der DO-Variablen. Diese Variable muß vom Typ INTEGER, REAL oder DOUBLE PRECISION sein.

e1,e2,e3 - arithmetische Ausdrücke vom Typ INTEGER, REAL oder DOUBLE PRECISION. Diese Ausdrücke geben den Anfangswert, den Endwert und die Schrittweite der DO-Variablen an. Fehlt e3, wird die DO-Variable zwischen zwei Schleifendurchläufen um 1 erhöht.

Die Abschlußanweisung der DO-Schleife darf keine der folgenden Anweisungen sein:

- unbedingte oder gesetzte GOTO-Anweisung
- arithmetische IF-Anweisung
- Block-IF-, ELSEIF-, ELSE- oder ENDIF-Anweisung
- RETURN-, STOP-, END- oder DO-Anweisung

Zur DO-Schleife gehören alle Anweisungen, die auf die DO-Anweisung folgen, bis einschließlich ihrer Abschlußanweisung. Eine Anweisung darf mehrere DO-Schleifen abschließen.

DO-Schleifensteuerung

Zuerst werden die Ausdrücke e1, e2 und e3 berechnet und wenn nötig in den Datentyp der DO-Variablen konvertiert. Die DO-Variable erhält den Wert von e1 als Anfangswert zugewiesen. Der Wert von e3 darf nicht Null sein. Die größte ganze Zahl, die kleiner oder gleich dem Wert von (e2-e1+e3)/e3 ist, gibt die Anzahl der Schleifendurchläufe an und wird dem Schleifenzähler zugewiesen. Der Schleifenzähler wird getestet. Ist er Null oder negativ, so wird die DO-Schleife beendet. Andernfalls werden die Anweisungen der DO-Schleife ausgeführt. Nach jedem Schleifendurchlauf wird die DO-Schleifensteuerung abgearbeitet. Diese Steuerung umfaßt folgende Schritte:

- Der Wert von e3 wird zur DO-Variablen addiert.
- Der Schleifenzähler wird um 1 erniedrigt.
- Wenn der Schleifenzähler größer als Null ist, wird wieder zur ersten ausführbaren Anweisung der DO-Schleife verzweigt.
- Wenn der Schleifenzähler Null ist, wird die DO-Schleife beendet.

Endet eine DO-Schleife, weil ihr Schleifenzähler Null oder negativ ist, wird mit der ersten ausführbaren Anweisung hinter der Abschlußanweisung fortgesetzt, sofern diese Abschlußanweisung nicht mehrere DO-Schleifen abschließt. In diesem Falle wird die Ausführung mit der Schleifensteuerung der nächsten umfassenderen DO-Schleife fortgesetzt. Die DO-Variable behält nach Beendigung der DO-Schleife ihren aktuellen Wert. Außer auf Grund ihres Schleifenzählerwertes kann die DO-Schleife noch wie folgt beendet werden:

- In der DO-Schleife wird eine RETURN-Anweisung ausgeführt.
- Eine Anweisung in derselben Programmeinheit, aber außerhalb der DO-Schleife, erhält die Steuerung.
- Eine STOP-Anweisung wird ausgeführt, oder das Programm wird aus irgendeinem Grunde abgebrochen.

Innerhalb der DO-Schleife darf der Wert der DO-Variablen nicht verändert werden. Die Werte der in e1, e2 und e3 auftretenden Variablen können in der DO-Schleife modifiziert werden. Das hat keinen Einfluß auf die Werte von Schleifenzähler und Schrittweite.

Beispiel:

```
          Y = A(N)
          DO 10 I = N-1, 0, -1
    10    Y = Y*X+A(I)
```

Diese DO-Schleife berechnet das Polynom N-ten Grades in X mit den Koeffizienten A(0), A(1) ... A(N):

```
Y = A(0) + A(1) * X + ... + A(N) * X ** N
```

Die DO-Schleife wird N-mal durchlaufen. I hat beim letzten Durchlauf den Wert 0, nach Beendigung der DO-Schleife den Wert -1.
Für den Schleifenzähler ist zu beachten, daß er beim Auftreten von REAL- oder DOUBLE PRECISION-Größen in der DO-Anweisung durch Rundungsfehler eventuell nicht den erwarteten Wert hat.

Geschachtelte DO-Schleifen

Eine DO-Schleife kann weitere DO-Schleifen enthalten. Eine innere DO-Schleife muß dabei völlig innerhalb der nächsten äußeren Schleife enthalten sein. Mehrere geschachtelte DO-Schleifen dürfen eine gemeinsame Abschlußanweisung haben.
Wenn eine DO-Schleife in einem IF-, ELSEIF- oder ELSE-Block auftritt, muß sie vollständig in diesem Block enthalten sein. Tritt in einer DO-Schleife eine Block-IF-Anweisung auf, so muß auch die zugehörige ENDIF-Anweisung noch innerhalb dieser DO-Schleife sein. Jede DO-Schleife darf von außen nur über ihre DO-Anweisung betreten werden. In geschachtelten DO-Schleifen sind Sprünge von inneren Schleifen in äußere erlaubt, jedoch nicht von äußeren Schleifen in innere. Besitzen mehrere DO-Schleifen dieselbe Abschlußanweisung, darf nur von der innersten DO-Schleife dorthin gesprungen werden. Jeder andere Sprung dorthin gilt als Sprung in eine innere Schleife und ist verboten.

Beispiel:

```
          DIMENSION A(N,N)
          X = 0
          K = 0
          L = 0
          DO 100 I = 1, N
          DO 100 J = 1, I - 1
          IF (ABS(A(I,J)) .LE. ABS(X)) GOTO 100
          X = A(I,J)
          K = I
          L = J
  100     CONTINUE
```

Dieses Beispiel sucht in einem zweidimensionalen quadratischen Feld das betragsmäßig größte Feldelement oberhalb der Diagonalen und gibt dessen Wert und Indizes in X, K und L zurück.

4.5. CONTINUE-Anweisung

Die CONTINUE-Anweisung entspricht einer Leeranweisung, d.h. die Steuerung wird an die nächste ausführbare Anweisung übergeben. Am häufigsten wird sie als letzte Anweisung einer DO-Gruppe benutzt. Die CONTINUE-Anweisung hat die Form:

```
CONTINUE
```

4.6. PAUSE-Anweisung

Diese Anweisung ermöglicht implementierungsabhängig (siehe [1]) eine Unterbrechung der Programmabarbeitung.
Die PAUSE-Anweisung hat die Form:

```
PAUSE [meld]
```

Dabei ist

meld - eine Zeichenkettenkonstante oder eine aus maximal fünf Ziffern bestehende Zeichenfolge.

4.7. STOP-Anweisung

Diese Anweisung beendet implementierungsabhängig (siehe [1]) die Ausführung eines Programms und gibt die Steuerung an das Betriebssystem zurück.

Die STOP-Anweisung hat die Form:

STOP [meld]

Dabei ist

meld - eine Zeichenkettenkonstante oder eine aus maximal fünf Ziffern bestehende Zeichenfolge.

4.8. END-Anweisung

Die letzte Zeile jeder Programmeinheit muß eine END-Anweisung sein. Sie zeigt das Ende der Folge von Anweisungen und Kommentarzeilen einer Programmeinheit an.
Die END-Anweisung hat die Form:

END

Wird eine END-Anweisung in einem FUNCTION- oder SUBROUTINE-Unterprogramm ausgeführt, so hat sie die Wirkung einer RETURN-Anweisung. Ihr Auftreten in einem Hauptprogramm bewirkt die Beendigung der Ausführung des Programms.
Eine END-Anweisung darf keine Fortsetzungszeilen haben.

4.9. CALL-Anweisung

Die CALL-Anweisung veranlaßt die Abarbeitung eines SUBROUTINE-Unterprogramms. Detaillierte Informationen sind Abschnitt 5.2.5. zu entnehmen.

4.10. RETURN-Anweisung

Die RETURN-Anweisung bewirkt, daß die Steuerung von einem Unterprogramm an die aufrufende Programmeinheit zurückgegeben wird. Sie darf nur in FUNCTION- und SUBROUTINE-Unterprogrammen angegeben werden. Detaillierte Informationen sind in Abschnitt 5.2.6. enthalten.

5. Unterprogramme

Unterprogramme sind Programmeinheiten, die von anderen Programmeinheiten aufgerufen werden können. Sie dienen im allgemeinen dazu, gemeinsam in verschiedenen Programmen oder Programmteilen benötigte Algorithmen nur einmal zu formulieren. In FORTRAN77 gibt es folgende Arten von Unterprogrammen:

- Anweisungsfunktionen
- FUNCTION-Unterprogramme
- SUBROUTINE-Unterprogramme
- Standardfunktionen
- BLOCK DATA-Unterprogramme

Während die ersten drei Unterprogrammarten vom Anwender selbst geschrieben werden können, sind die Standardfunktionen Bestandteil von FORTRAN77.
BLOCK DATA-Unterprogramme dienen dazu, den Objekten von benannten COMMON-Bereichen Anfangswerte zuzuweisen. Sie enthalten keine ausführbaren Anweisungen und können auch nicht aufgerufen werden.
Außer den Anweisungsfunktionen sind alle anderen Unterprogramme externe Unterprogramme, das heißt, sie werden als selbständige Programmeinheiten formuliert und übersetzt.
Parameter von Unterprogrammen dienen der Kommunikation zwischen der aufrufenden Programmeinheit und dem aufgerufenen Unterprogramm. Dabei wird zwischen aktuellen und formalen Parametern unterschieden. Die aktuellen Parameter werden beim Aufruf eines Unterprogrammes, die formalen Parameter bei der Vereinbarung des Unterprogrammes angegeben. Die Verbindung zwischen aktuellen Parametern und formalen Parametern erfolgt, wenn die Steuerung an das Unterprogramm übergeben wird.

5.1. Verbindung aktueller und formaler Parameter

Beim Aufruf eines Unterprogrammes wird eine Verbindung zwischen den korrespondierenden aktuellen und formalen Parametern hergestellt. Der erste aktuelle Parameter wird mit dem ersten formalen Parameter verbunden, der zweite aktuelle mit dem zweiten formalen Parameter und so weiter. Die Zahl der aktuellen und der formalen Parameter muß übereinstimmen. Auf diese Art wird jede Bezugnahme auf einen formalen Parameter im Unterprogramm zur Zeit der Ausführung des Unterprogrammes zu einer Bezugnahme auf einen aktuellen Parameter.

Aktueller Parameter kann sein:

- Name eines Feldes
- Name eines Unterprogrammes, außer Name einer Anweisungsfunktion
- Ausdruck
- alternativer Rücksprung (siehe Abschnitt 5.1.4.)

Der formale Parameter wird im Unterprogramm benutzt, um zu beschreiben, welcher Art der zu erwartende aktuelle Parameter ist, ob er ein einzelner Wert, ein Feld von Werten, ein Unterprogramm oder eine Anweisungsmarke ist. Mit dieser Information kann der Compiler den entsprechenden Code für den Zugriff auf den aktuellen Parameter herstellen.
Als formaler Parameter darf auftreten:

- Name einer Variablen
- Name eines Feldes
- Name eines Unterprogrammes
- ein Stern (*).

Jeder formale Parameter, außer dem Zeichen Stern, kann wieder als aktueller Parameter beim Aufruf weiterer Unterprogramme benutzt werden.
Eine Verbindung zwischen aktuellen und formalen Parametern ist nur dann gültig, wenn sie in ihrem Datentyp übereinstimmen. Das gilt nicht, wenn der aktuelle Parameter der Name eines SUBROUTINE-Unterprogrammes ist. In diesem Fall muß der formale Parameter ebenfalls ein SUBROUTINE-Unterprogramm sein, oder, wenn der aktuelle Parameter ein alternativer Rücksprung ist, muß der formale Parameter ein Stern (*) sein (siehe Abschnitt 5.1.4.).
Ist der aktuelle Parameter ein Ausdruck, erfolgt die Berechnung des Ausdrucks bevor die Verbindung zwischen aktuellem und formalem Parameter hergestellt wird. Die Verbindung zwischen aktuellem und formalem Parameter bleibt solange bestehen, bis im Unterprogramm eine RETURN- oder END-Anweisung ausgeführt wird.
Ist der aktuelle Parameter eine Konstante, der symbolische Name einer Konstanten, ein Ausdruck mit Operatoren, eine Funktionsbezugnahme oder ein Ausdruck in Klammern, dann darf dem formalen Parameter im Unterprogramm kein Wert zugewiesen werden.
Ein formaler Parameter, der als Feld vereinbart wurde, kann nur mit einem aktuellen Parameter verbunden werden, der ein Feld, ein Feldelelement oder eine Teilkette eines Feldelementes vom gleichen Datentyp ist.
Die Länge eines formalen Parameters darf nicht größer sein als die Länge des zugehörigen aktuellen Parametrs. Ist die Länge des formalen Parameters kleiner als die des aktuellen Parameters, so wird der aktuelle Parameter nur in der Länge des formalen Parameters verbunden. Hat der formale Parameter den Datentyp CHARACTER und die Längenangabe *(*), dann wird im Unterprogramm exakt mit der-

Länge des aktuellen Parameters gearbeitet. Ist der formale Parameter ein Unterprogrammname, dann muß er mit einem aktuellen Parameter verbunden sein, der Name einer Standardfunktion, eines FUNCTION- oder SUBROUTINE-Unterprogrammes ist. Ist der aktuelle Parameter der Name einer Standardfunktion, dann hat der formale Parameter nicht automatisch die Datentypeigenschaften wie sie sich nach Anlage 2 ergeben, auch dann nicht, wenn der formale Parameter der Name einer Standardfunktion ist. Tritt der Name eines formalen Parameters in einer Typvereinbarungs- und einer EXTERNAL-Anweisung auf, so darf der zugehörige aktuelle Parameter nur der Name einer Standardfunktion oder eines FUNCTION-Unterprogrammes sein.
Tritt der Name eines formalen Parameters in einer CALL-Anweisung als Bezugnahme auf ein SUBROUTINE-Unterprogramm auf, muß der aktuelle Parameter der Name eines SUBROUTINE-Unterprogrammes sein, und der Name darf nicht in einer Typvereinbarungsanweisung oder als Funktionsbezugnahme verwendet werden.

5.1.1. Dynamische Felder

Wird ein formaler Parameter als Feld vereinbart, dessen Dimensionsvereinbarungen Variablen enthalten (siehe Abschnitt 2.5.1.), handelt es sich um ein dynamisches Feld. Alle in einem Dimensionsgrenzenausdruck vorkommenden Variablen müssen vom Datentyp INTEGER sein. Außerdem müssen sie formaler Parameter oder Objekt eines COMMON-Bereichs sein. Weil die Dimensionsgrenzenausdrücke von dynamischen Feldern erst berechnet werden, wenn das Unterprogramm zur Ausführungszeit die Steuerung erhält, müssen die Variablen in den Dimensionsgrenzenausdrücken zu diesem Zeitpunkt definiert sein. Ist eine solche Variable ein formaler Parameter, so heißt das, er muß mit einem aktuellen Parameter verbunden sein, der definiert ist.
Ein dynamisches Feld darf, nachdem die Dimensionsgrenzen berechnet sind, nicht größer sein als das Feld, das der zugehörige aktuelle Parameter ist. Dabei ist zu beachten, daß der aktuelle Parameter auch ein Feldelement oder die Teilkette eines Feldelementes sein kann. In diesem Fall gilt als Länge des zugehörigen aktuellen Parameters die Restlänge des Feldes ab der Position des Feldelementes oder des Teilkettenbeginns des Feldelementes (siehe Abschnitt 5.1.2.).
Im folgenden Beispiel wird die Summe der Elemente eines zweidimensionalen Feldes berechnet.

```
      FUNCTION SUMME(A,M,N)
      DIMENSION A(M,N)
      SUMME = 0.0
      DO 10 J = 1,N
      DO 10 I = 1,M
10    SUMME = SUMME + A(I,J)
      RETURN
      END
```

Als Beispiele für den Aufruf von SUMME sollen folgende Anweisungen dienen:

```
      DIMENSION A1(10,35),A2(3,56)
      SUM1 = SUMME(A2,3,56)
      SUM2 = SUMME(A2,3,10)
      SUM3 = SUMME(A1,10,35)
```

Die Werte der Unter- und Obergrenzen einer Dimension werden einmal beim Aufruf des Unterprogrammes berechnet. Ändern sich bei der Abarbeitung des Unterprogrammes die Werte von Variablen, die in einer Feldvereinbarung verwendet wurden, so hat das keinen Einfluß auf die Größe des dynamischen Feldes.
Beispiel:

```
      DIMENSION FELD(9,5)
      L = 9
      M = 5
      CALL SUB(FELD,L,M)
      END
      SUBROUTINE SUB(X,I,J)
      DIMENSION X(-I/2:I/2,J)
      X(I/2,J) = 999
      J = 1
      I = 2
      END
```

In diesem Fall wird das dynamische Feld X mit der Dimension (-4:4,5) beim Betreten des Unterprogrammes SUB vereinbart. Die Zuweisung von Werten an I und J berühren diese Vereinbarung nicht mehr.
Eine Variable, die in einem Dimensionsgrenzenausdruck eines dynamischen Feldes verwendet wird, muß immer vorher durch eine Typvereinbarung den Datentyp INTEGER erhalten. Eine nachträgliche Vereinbarung ist nicht gestattet. Der folgende Programmabschnitt ist deshalb fehlerhaft:

```
      SUBROUTINE SUB1(A,X)
      DIMENSIOM A(X)
      INTEGER X
```

5.1.2. Felder mit angenommener Größe

Ist ein formaler Parameter ein Feld, dessen Obergrenze der letzten Dimension mit Stern (*) vereinbart wurde, so hat dieses Feld in der letzten Dimension eine variable Obergrenze. Ein solches Feld kann aber nur die Größe seines zugehörigen aktuellen Parameters annehmen. Deshalb ist ein solches Feld ein Feld mit angenommener Größe.
Im folgenden Beispiel ist A ein Feld mit angenommener Größe:

```
      SUBROUTINE ROUT(A,N)
      DIMENSION A(5:N,1:*)
        .
        .
        .
```

Wie in diesem Beispiel zu sehen ist, können Felder mit angenommener Länge auch dynamische Felder sein. Die Anzahl der Feldelemente, auf die man in einem Feld mit angenommener Länge zugreifen kann, ergibt sich nach folgenden Regeln:

- Jedes Feldelement erhält einen Wert (den Indexwert) zugeordnet, welcher angibt, die wievielte Stelle dieses Element bei der Speicherung des Feldes einnimmt.
  So haben z.B. für das mit
  ```
  DIMENSION B(-1:3, -2:2, 0:4)
  ```
  vereinbarte Feld die Elemente B(-1,-2,0), B(0,0,0) und B(3,2,1) die Indexwerte 1, 12 bzw. 50.
- Der Indexwert ist wesentlich zum Verständnis der Verbindung zwischen aktuellem und formalem Parameter bei Feldern. Aktuelle und formale Parameter müssen nicht in ihren Feldgrenzen, ja nicht einmal in der Anzahl ihrer Dimensionen übereinstimmen. Wichtig ist nur, daß der höchste für den formalen Parameter benutzte Indexwert nicht größer ist als der durch den aktuellen Parameter erlaubte maximale Indexwert.
- Ist der aktuelle Parameter ein Feld mit a Elementen, so kann der formale Parameter auf a Feldelemente zugreifen. Ist der zugehörige aktuelle Parameter ein Feldelement desselben Feldes mit dem Indexwert s, so darf der formale Parameter nur auf die restlichen a+1-s Feldelemente zugreifen.
- Für Felder vom Datentyp CHARACTER müssen die Längen der Feldelemente bei aktuellem und formalem Parameter nicht übereinstimmen. Analog zu oben legt das aktuelle Argument, das jetzt ein Feld, ein Feldelement oder die Teilkette eines Feldelements sein kann, das erste Zeichen des Feldes fest, das mit dem formalen Parameter verbunden wird. Hat der formale Parameter die Feldelementlänge y und ist der aktuelle Parameter ein Feld von insgesamt n Zeichen, so darf der formale Parameter auf INT(n/y) Feldelemente zugreifen. Beginnt der formale Parameter als Feldelement oder Teilkette eines Feldelements am b-ten Zeichen die-

ses Feldes, so darf der formale Parameter nur auf INT((n+1-b)/y) Feldelemente zugreifen.

Weil die aktuelle Größe eines Feldes mit angenommener Größe unbekannt ist, darf sein Name nicht verwendet werden als

- Feldname in der Datenliste einer E/A-Anweisung,
- Einheitennummer für eine interne Datei in einer E/A-Anweisung,
- FMT-Angabe in einer E/A-Anweisung (siehe Abschnitt 6.).

5.1.3. Zeichenketten mit übernommener Länge

Ist ein formaler Parameter eine Variable mit dem Datentyp CHARACTER und der Längenangabe (*), so übernimmt dieser formale Parameter exakt die Länge des zugehörigen aktuellen Parameters. Solche Variablen sind Zeichenketten mit übernommener Länge. Dynamische Felder und Felder mit angenommener Größe können Feldelemente haben, die Zeichenketten mit übernommener Länge sind. Im folgenden Beispiel verwendet ein FUNCTION-Unterprogramm eine Zeichenkette mit übernommener Länge. Die Funktion findet die Position des Zeichens mit der höchsten Code-Wertigkeit. Dabei wird die übernommene Länge zur Steuerung der Iteration verwendet. Die benutzte Standardfunktion LEN ist im Abschnitt 5.3.3. beschrieben.

```
      INTEGER FUNCTION ICMAX(CHARV)
      CHARACTER*(*) CHARV
      ICMAX = 1
      DO 10 I = 2,LEN(CHARV)
10    IF (CHARV(I:I) .GT. CHARV(ICMAX:ICMAX)) ICMAX = I
      RETURN
      END
```

Die Länge des formalen Parameters CHARV wird jedesmal bei Betreten der Funktion bestimmt. Die Länge des aktuellen Argumentes kann die Länge einer Zeichenkettenvariablen, eines Zeichenkettenfeldelelementes, einer Teilkette oder eines Zeichenkettenausdruckes sein. Jeder der folgenden Funktionsaufrufe führt zu einer anderen Länge des formalen Parameters:

```
      CHARACTER*10 VAR, CFELD(3,5)*20
       .
       .
       .
      I1 = ICMAX(VAR)
      I2 = ICMAX(CFELD(2,2))
      I3 = ICMAX(VAR(3:8))
      I4 = ICMAX(CFELD(1,3)(5:15))
      I5 = ICMAX(VAR(3:4)//CFELD(3,5))
```

5.1.4. Alternativer Rücksprung

Zur Kennzeichnung alternativer Rücksprünge in einer formalen Parameterliste wird das Zeichen Stern (*) verwendet.
Zum Beispiel:

```
      SUBROUTINE BSP1(A,*,*,B,C)
       .
       .
       .
      ENTRY BSP2(A,D,E,*)
       .
       .
       .
```

Alternative Rücksprünge sind nur in SUBROUTINE-Unterprogrammen erlaubt, und die aktuelle Parameterliste muß an der entsprechenden Position einen alternativen Rücksprungparameter enthalten. Für obiges Beispiel sind die nachfolgenden CALL-Anweisungen korrekt:

```
      CALL BSP2(F,G,H,*10)
      CALL BSP1(E,*10,*20,'MAX','MIN')
```

10 und 20 müssen dabei die Marken ausführbarer Anweisungen in der aufrufenden Programmeinheit sein.

5.2. Nutzergeschriebene Unterprogramme

In FORTRAN77 hat der Nutzer folgende drei Möglichkeiten, eigene Unterprogramme zu vereinbaren:

- die Anweisungsfunktion
- das FUNCTION-Unterprogramm
- das SUBROUTINE-Unterprogramm

Die Anweisungsfunktion und das FUNCTION-Unterprogramm werden durch eine Funktionsbezugnahme aufgerufen. Das SUBROUTINE-Unterprogramm kann nur durch eine CALL-Anweisung aufgerufen werden.
FUNCTION- und SUBROUTINE-Unterprogramme können die Werte der ihnen übergebenen Parameter verändern, und in der aufrufenden Programmeinheit sind die geänderten Werte verwendbar. Dabei ist aber zu beachten, daß es aktuelle Parameter gibt, deren Werte nicht verändert werden dürfen (siehe Abschnitt 5.1.).
Ein Unterprogramm kann ein anderes Unterprogramm aufrufen. Es ist aber nicht gestattet, daß sich ein Unterprogramm selbst aufruft, weder direkt noch indirekt.

5.2.1. Anweisungsfunktionen

Eine Anweisungsfunktion ist ein Unterprogramm, das nur einen Wert, den Funktionswert, ermitteln kann. Sie wird in Form einer einzelnen Anweisung vereinbart, die in ihrem Aufbau einer Ergibtanweisung entspricht. Der berechnete Funktionswert einer Anweisungsfunktion wird an der Stelle der Funktionsbezugnahme in einem Ausdruck bereitgestellt.
Eine Anweisungsfunktion wird folgendermaßen vereinbart:

```
fn([p[,p]...]) = e
```

wobei

fn - Name der Anweisungsfunktion,
p - Name eines formalen Parameters,
e - ein Ausdruck ist.

Der Datentyp einer Anweisungsfunktion wird entweder durch eine Typvereinbarung für den Anweisungsfunktionsnamen fn, oder anhand des Anfangsbuchstabens des Anweisungsfunktionsnamens fn implizit oder durch Standardtypvereinbarung festgelegt. Dabei sind alle Datentypen von FORTRAN77 möglich (siehe Abschnitt 2.1.), mit der Einschränkung, daß die Längenangabe beim Datentyp CHARACTER ein Konstantenausdruck vom Typ INTEGER sein muß.
Der Ausdruck e kann ein arithmetischer, logischer oder ein Zeichenkettenausdruck sein (siehe Abschnitt 2.7.), mit der Einschränkung, daß eine Bezugnahme auf eine andere Anweisungsfunktion nur erlaubt ist, wenn sie vor der Anweisungsfunktion fn vereinbart wurde. Die Beziehungen zwischen e und fn müssen den Regeln für eine Ergibtanweisung entsprechen (siehe Abschnitt 4.1.). Es ist zu beachten, daß sich entsprechend diesen Regeln die Datentypen von e und fn unterscheiden können.
Formaler Parameter einer Anweisungsfunktion kann nur ein nichtindizierter Name einer Variablen sein.
Die formalen Parameter p werden dazu verwendet, die Anzahl, Reihenfolge und den Datentyp der aktuellen Parameter anzuzeigen. Der Name eines formalen Parameter darf in der Liste der formalen Parameter einer Anweisungsfunktion nur einmal auftreten. Der Gültigkeitsbereich eines solchen Namens beschränkt sich nur auf die Anweisungsfunktion. Deshalb kann der Name eines formalen Parameters einer Anweisungsfunktion an anderer Stelle der Programmeinheit zur Bezeichnung einer anderen Variablen oder in einer anderen Anweisungsfunktion wieder als formaler Parameter verwendet werden. Hat der formale Parameter einer Anweisungsfunktion den Datentyp CHARACTER, dann ist eine Längenvereinbarung der Form (*) verboten.

Die Funktionsbezugnahme einer Anweisungsfunktion hat folgendes Aussehen:

```
fn([p[,p]...])
```

Dabei ist:

fn - Name der Anweisungsfunktion,
p - aktueller Parameter.

Die aktuellen Parameter müssen Ausdrücke sein. Ein Zeichenkettenausdruck darf keine Verkettungsoperation mit einer Zeichenkette mit übernommener Länge enthalten (siehe Abschnitt 5.1.3.). Tritt in einem Ausdruck die Funktionsbezugnahme einer Anweisungsfunktion auf, werden zuerst die Werte der aktuellen Parameter berechnet und diese mit den formalen Parametern der Anweisungsfunktion verbunden. Dann wird der Wert des Ausdrucks e, so wie in der Vereinbarung angegeben, ermittelt und eventuell in den Datentyp von fn konvertiert. Der berechnete Wert steht dann in dem Ausdruck zur Verfügung, der die Funktionsbezugnahme enthielt.

Gültige Vereinbarungen von Anweisungsfunktionen sind:

```
INHALT(RADIUS) = 4.189*RADIUS**3
SINH(X) = (EXP(X)-EXP(-X))*0.5
CHARACTER*10 CHARAF,A,B
CHARAF(A,B) = A(6:10)//B(1:5)
```

Die folgenden Beispiele sollen einige ungültige Vereinbarungen zeigen:

```
AVG(A,B,3.) = (A+B)*3.
BSP(A,B) = A+BSP(B,3)
EXTERNAL FAF
FAF(A,B) = A**(B+1)
```

Abschließend einige Beispiele von Bezugnahmen auf eine Anweisungsfunktion mit folgender Vereinbarung:

```
IMPLICIT REAL(A-C,T-Z),INTEGER(I)
 .
 .
 .
AVG(A,B,C) = (A+B+C)/3.
 .
 .
 .
GRAD = AVG(TEST1,TEST2,XLAB)
IF(AVG(A,B,C) .LT. AVG(X,Y,Z)) GO TO 30
ENDE = AVG(TEST3,TEST4,ILAB)
```

Die letzte der drei Bezugnahmen ist dabei fehlerhaft, weil die Datentypen zwischen aktuellem und formalem Parameter nicht übereinstimmen.

5.2.2. FUNCTION-Unterprogramme

Ein FUNCTION-Unterprogramm ist eine Programmeinheit, die mit einer FUNCTION-Anweisung beginnt und einen Wert, den Funktionswert, berechnet. Zu diesem Zwecke muß der Name des Funktionsunterprogramms (oder ein assoziierter ENTRY-Name vom gleichen Datentyp) auch als Variablenname im Funktionsunterprogramm auftreten. Diese Variable muß definiert sein, wenn das Funktionsunterprogramm durch eine RETURN oder END-Anweisung verlassen wird. Ihr Wert wird als Funktionswert in die aufrufende Programmeinheit zurückgegeben und ersetzt dort die Funktionsbezugnahme.
Die FUNCTION-Anweisung hat folgenden Aufbau:

```
[tp] FUNCTION nam ([p[,p]...])
```

Dabei ist:

tp - Angabe des Datentyps (siehe Abschnitt 2.1.),
nam - Name des FUNCTION-Unterprogramms,
p - formaler Parameter (siehe Abschnitt 5.1.).

Ist tp von der Form CHARACTER*l, dann muß l ein ganzzahliger Konstantenausdruck sein, der aber keine Namen von Konstanten enthalten darf. Alle anderen Datentypen gemäß Abschnitt 2.1. sind angebbar. Wird CHARACTER*(*) als tp benutzt, so übernimmt das Funktionsunterprogramm die Länge des Funktionswerts aus der aufrufenden Programmeinheit. Das ist für den Programmierer der beste Weg, die geforderte Übereinstimmung der Längen von Funktionsbezugnahme in der aufrufenden Programmeinheit und Funktionswert im Funktionsunterprogramm zu sichern. Wird tp nicht benutzt, so kann der Datentyp des Funktionswerts auch in einer nachfolgenden Typvereinbarungsanweisung festgelegt werden, oder die Standardtypvereinbarung

wird wirksam.
Alternative Rücksprünge sind als formale Parameter eines FUNCTION-Unterprogrammes verboten.
Der Aufruf eines FUNCTION-Unterprogrammes erfolgt durch eine Funktionsbezugnahme als Operand in einem Ausdruck.
Sie hat folgendes Aussehen:

```
nam([p[,p]...])
```

Dabei ist

- nam - Name eines FUNCTION-Unterprogrammes oder eines formalen Parameters, der mit einem FUNCTION-Unterprogramm verbunden ist.
- p - aktueller Parameter (siehe Abschnitt 5.1.)

Der Name "nam" des FUNCTION-Unterprogrammes wird auch in der aufrufenden Programmeinheit zur Bestimmung des Datentypes des Wertes verwendet, den die Funktionsbezugnahme zur Ausführungszeit repräsentiert. Dazu kann "nam" in einer Typvereinbarungsanweisung auftreten. Ist das nicht der Fall, kommt es anhand des Anfangsbuchstabens von nam zu einer Standardtypvereinbarung.
Auf jeden Fall muß aber der Datentyp des FUNCTION-Unterprogrammes bei der Funktionsbezugnahme der gleiche sein wie in der Vereinbarung des FUNCTION-Unterprogrammes. Hat das FUNCTION-Unterprogramm den Datentyp CHARACTER, muß auch die Länge bei der Funktionsbezugnahme und bei der Vereinbarung des FUNCTION-Unterprogrammes übereinstimmen. Aktueller Parameter darf kein Zeichenkettenausdruck sein, in dem Verkettungsoperationen mit Zeichenketten mit übernommener Länge auftreten (siehe Abschnitt 5.1.3.).
Die Ausführung einer Funktionsbezugnahme auf ein FUNCTION-Unterprogramm führt zu folgenden Aktionen:
- Berechnung der Ausdrücke, die aktuelle Parameter sind.
- Verbindung der aktuellen mit den formalen Parametern.
- Ausführung des FUNCTION-Unterprogramms und abschließend Bereitstellung des Funktionswerts anstelle der Funktionsbezugnahme. Die Ausführung beginnt mit der ersten ausführbaren Anweisung hinter der FUNCTION- oder einer ENTRY-Anweisung (siehe Abschnitt 5.2.4.).

Eine FUNCTION-Anweisung muß die erste Anweisung eines FUNCTION-Unterprogrammes sein. Ein solches Unterprogramm darf keine SUBROUTINE-, BLOCK DATA- und PROGRAM-Anweisung enthalten. ENTRY-Anweisungen dagegen sind gestattet, um mehrere Eingangspunkte in ein FUNCTION-Unterprogramm zu ermöglichen (siehe Abschnitt 5.2.4.).
Anschließend zwei Beispiele für FUNCTION-Unterprogramme.

```
      FUNCTION NULL(A)
      X = 1.0
    2 EX = EXP(X)
      EMINX = 1./EX
      NULL = ((EX+EMINX)*.5+COS(X)-A)/((EX-EMINX)*.5-SIN(X))
      IF ((ABS(X)-NULL) .LT. 1E-6) RETURN
      X = NULL
      GO TO 2
      END
```

Dieses Beispiel verwendet die Iterationsmethode nach NEWTON-RAPHSON zur Berechnung der Nullstelle der Funktion:

$$F(x) = \cosh(x)+\cos(x)-A = 0$$

Der Wert von A wird als Parameter übergeben. Die Iterationsformel für die Nullstelle lautet:

$$x(i+1) = x(i) - \left(\frac{\cosh(x(i))+\cos(x(i))-A}{\sinh(x(i))-\sin(x(i))} \right)$$

Diese Berechnung wird solange wiederholt, bis die Differenz zwischen x(i) und x(i+1) kleiner als 1.0E-6 ist. Dabei werden die Standardfunktionen EXP, SIN, COS und ABS (siehe Anlage 2) verwendet.
Das zweite Beispiel soll eine CHARACTER*(*) FUNCTION mit Vereinbarung und Anwendung zeigen.

```
      CHARACTER*(*) FUNCTION REPEAT(ARG)
      CHARACTER ARG
      DO 10 I = 1,LEN(REPEAT)
   10 REPEAT(I:I) = ARG
      RETURN
      END
```

Innerhalb einer Programmeinheit müssen alle Bezugnahmen auf eine CHARACTER*(*) FUNCTION mit der gleichen Länge erfolgen. Im folgenden Aufruf wird REPEAT mit der Länge 1000 aufgerufen:

```
      CHARACTER*1000 REPEAT,AKETTE,ZKETTE
      AKETTE = REPEAT('A')
      ZKETTE = REPEAT('Z')
```

Innerhalb des gleichen ausführbaren Programmes, aber in einer anderen Programmeinheit, kann REPEAT mit einer anderen Länge aufgerufen werden, beim folgenden Aufruf zum Beispiel mit der Länge 2.

```
      CHARACTER*6 ZIEL,REPEAT*2
      ZIEL = REPEAT('A')//REPEAT('B')//REPEAT('C')
```

5.2.3. SUBROUTINE-Unterprogramme

Ein SUBROUTINE-Unterprogramm ist eine Programmeinheit, die mit einer SUBROUTINE-Anweisung beginnt.
Es hat folgende Struktur:

```
SUBROUTINE-Anweisung
 .
 .
 .
[RETURN-Anweisung]
 .
 .
 .
END
```

Der Aufruf eines SUBROUTINE-Unterprogramms erfolgt mit einer CALL-Anweisung (siehe Abschnitt 4.9.). Die Rückgabe der Steuerung an die aufrufende Programmeinheit erfolgt durch eine RETURN- (siehe Abschnitt 4.10.) oder END-Anweisung.
Die SUBROUTINE-Anweisung hat folgenden Aufbau:

```
SUBROUTINE nam [([p[,p]...])]
```

Dabei ist:

nam - Name des SUBROUTINE-Unterprogrammes
p - formaler Parameter (siehe Abschnitt 5.1.)

Ist der formale Parameter eines SUBROUTINE-Unterprogrammes eine Zeichenkette mit übernommener Länge (siehe Abschnitt 5.1.3.), dann darf diese Variable außer in Zeichenkettenergibtanweisungen nicht in Verkettungsoperationen verwendet werden.
Das folgende Beispiel enthält ein SUBROUTINE-Unterprogramm zur Berechnung des Volumens regulärer Polyeder bei vorgegebener Anzahl der Flächen und der Kantenlänge. Es verwendet die berechnende GO TO-Anweisung um festzustellen, ob das Polyeder ein Tetraeder, ein Würfel, ein Oktaeder, ein Dodecaeder, oder ein Icosaeder ist. Gleichzeitig realisiert diese GO TO-Anweisung den Sprung an die Stelle, wo das zugehörige Volumen berechnet wird. Ist die Anzahl der gegebenen Flächen nicht 4, 6, 8, 12 oder 20, dann gibt das SUBROUTINE-Unterprogramm eine Fehlermeldung aus.

Hauptprogramm

```
      PROGRAMM TEST
      COMMON ANZ,KANTE,VOLUM
      READ *,ANZ,KANTE
      CALL POLY
      PRINT *,'VOLUMEN =',VOLUM
      STOP
      END
```

Unterprogramm

```
      SUBROUTINE POLY
      COMMON ANZ,KANTE,VOLUm
      CUBUS = KANTE**3
      GO TO ( 6,6,6,1,6,2,6,3,6,6,6,4,6,6,6,6,6,6,6,5 ), ANZ
      GO TO 6
 1    VOLUM = CUBUS*0.11785
      RETURN
 2    VOLUM = CUBUS
      RETURN
 3    VOLUM = CUBUS*0.47140
      RETURN
 4    VOLUM = CUBUS*7.66312
      RETURN
 5    VOLUM = CUBUS*2.18170
      RETURN
 6    PRINT 100,ANZ
100   FORMAT ('KEIN REGULAERES POLYEDER HAT',I3,'FLAECHEN'/)
      VOLUM = 0.0
      RETURN
      END
```

Das folgende Beispiel zeigt die Verwendung von alternativen Rücksprungparametern. Mit ihnen wird bestimmt, wohin die Steuerung in die aufrufende Programmeinheit zurückgegeben wird. Die Parameterliste der SUBROUTINE-Anweisung enthält zwei alternative Rücksprungparameter, die mit den aktuellen Parametern *10 und *20 in der aktuellen Parameterliste der CALL-Anweisung verbunden sind. Die Auswahl der RETURN-Anweisung erfolgt in Abhängigkeit von den für Z im Unterprogramm berechneten Wert. Hat Z einen Wert kleiner als Null, wird der normale Rücksprung benutzt. Ist der Wert gleich Null, dann wird zur Anweisung mit der Marke 10, im Falle größer als Null zur Anweisung mit der Marke 20 im Hauptprogramm zurückgekehrt.

Hauptprogramm

```
          CALL  CHECK(A,B,*10,*20,C)
          PRINT *,'WERT KLEINER NULL'
          GOTO 30
    10    PRINT *,'WERT GLEICH NULL'
          GOTO 30
    20    PRINT *,'WERT GROESSER NULL'
    30    CONTINUE
```

SUBROUTINE-Unterprogramm

```
          SUBROUTINE CHECK (X,Y,*,*,Q)
          .
          .
          .
          IF(Z) 60,70,80
    60    RETURN
    70    RETURN 1
    80    RETURN 2
          END
```

5.2.4. ENTRY-Anweisung

Die ENTRY-Anweisung ermöglicht sekundäre Eingangspunkte in ein Unterprogramm. Sie ist nicht ausführbar und kann innerhalb eines FUNCTION- oder SUBROUTINE-Unterprogramms nach der FUNCTION- oder SUBROUTINE-Anweisung stehen. Wird ein Unterprogramm über einen ENTRY-Namen aufgerufen, beginnt die Abarbeitung des Unterprogrammes mit der ersten ausführbaren Anweisung nach der ENTRY-Anweisung.
Die ENTRY-Anweisung hat die Form:

```
ENTRY nam [([p[,p]...])]
```

Dabei ist:

nam - ENTRY-Name
p - formaler Parameter (siehe Abschnitt 5.1.)

Wird die ENTRY-Anweisung in einem SUBROUTINE-Unterprogramm verwendet, ist der ENTRY-Name auch ein Name des SUBROUTINE-Unterprogrammes. Tritt eine ENTRY-Anweisung in einem FUNCTION-Unterprogramm auf, dann ist der ENTRY-Name auch ein Name des FUNCTION-Unterprogrammes.
In einem FUNCTION-Unterprogramm ist es nicht gestattet, in der Liste der formalen Parameter einer ENTRY-Anweisung einen alternati-

ven Rücksprung anzugeben.
Die Liste der formalen Parameter einer ENTRY-Anweisung kann sich in Reihenfolge, Anzahl, Typ und Namen der formalen Parameter von denen der FUNCTION-, SUBROUTINE- oder anderer ENTRY-Anweisungen der gleichen Programmeinheit unterscheiden. Ein formaler Parameter kann in ausführbaren Anweisungen nur hinter der ENTRY-Anweisung verwendet werden, die den formalen Parameter enthält.
Eine ENTRY-Anweisung darf nicht innerhalb einer Block-IF-Konstruktion oder einer DO-Schleife auftreten.

ENTRY im FUNCTION-Unterprogramm

In einem FUNCTION-Unterprogramm ist die Variable, die den Namen des FUNCTION-Unterprogrammes hat, mit allen Variablen assoziiert, die einen ENTRY-Namen haben. Das bedeutet zum Beispiel, wenn einer Variablen, die einen ENTRY-Namen hat, ein Wert zugewiesen wird, dann sind alle assoziierten Variablen vom gleichen Datentyp als definiert zu betrachten. Alle assoziierten Variablen mit anderem Datentyp sind in dem Falle undefiniert. Eine Ausnahme macht der Datentyp CHARACTER. Er erfordert die Übereinstimmung des Datentyps und der Längenangabe für alle assoziierten Variablen.
Der Aufruf eines FUNCTION-Unterprogrammes über einem ENTRY-Namen erfolgt durch eine Funktionsbezugnahme (siehe Abschnitt 5.2.2.).
Bei der Ausführung einer RETURN-Anweisung oder am Ende eines FUNCTION-Unterprogrammes muß der Name, unter dem das Unterprogramm aufgerufen wurde, definiert sein, d.h. ihm muß wenigstens einmal ein Wert zugewiesen worden sein.
Der ENTRY-Name eines FUNCTION-Unterprogrammes darf in keiner Anweisung vor der ENTRY-Anweisung auftreten, außer in einer Typvereinbarung.

Das folgende Beispiel zeigt ein FUNCTION-Unterprogramm mit mehreren Eingangspunkten:

```
      REAL FUNCTION TANH(A)
C ANWEISUNGSFUNKTION FUER SINH
      ASINH(B) = EXP(B)-EXP(-B)
C ANWEISUNGSFUNKTION FUER COSH
      ACOSH(B) = EXP(B)+EXP(-B)
C BERECHNUNG TANH
      TANH = ASINH(A)/ACOSH(A)
      RETURN
C BERECHNUNG SINH
      ENTRY SINH(A)
      SINH = ASINH(A)/2.0
      RETURN
C BERECHNUNG COSH
      ENTRY COSH(A)
      COSH = ACOSH(A)/2.0
      RETURN
      END
```

Je nach Aufruf mit SINH(X), COSH(X) oder TANH(X) berechnet diese Funktion den hyperbolischen Sinus, Cosinus oder Tangens von X.

ENTRY im SUBROUTINE-Unterprogramm

Der Aufruf eines SUBROUTINE-Unterprogrammes über einen ENTRY-Namen erfolgt durch die CALL-Anweisung.

```
Hauptprogramm                    SUBROUTINE-Unterprogramm
-------------                    ------------------------

.                                SUBROUTINE UPRO(U,V,W)
.                                .
.                                .
CALL UPRO1(A,B,C)                .
.                                ENTRY UPRO1(X,Y,Z)
.
.
```

In diesem Beispiel wird durch die CALL-Anweisung das Unterprogramm UPRO über den ENTRY-Namen UPRO1 aufgerufen. Die Abarbeitung des Unterprogrammes beginnt in der ersten ausführbaren Anweisung nach ENTRY UPRO1(X,Y,Z) unter Verwendung der aktuellen Parameter (A,B,C) aus der CALL-Anweisung.
Ein alternativer Rücksprungparameter kann auch in der ENTRY-Anweisung angegeben werden.

Zum Beispiel:

```
      SUBROUTINE UPRO(X,Y)
       .
       .
       .
      ENTRY UPRO1(K,J,*,X,*)
       .
       .
       .
      RETURN 1
      RETURN 2
      END
```

Der Aufruf über den ENTRY-Namen UPRO1 könnte hier also lauten:
CALL UPRO1(I,M,*10,A,*20)
In diesem Beispiel würde die Anweisung RETURN 1 die Steuerung an die Anweisung 10 und RETURN 2 die Steuerung an die Anweisung 20 in der aufrufenden Programmeinheit zurückgeben.

5.2.5. Aufruf von SUBROUTINE-Unterprogrammen

Die CALL-Anweisung veranlaßt die Abarbeitung eines SUBROUTINE-Unterprogrammes (siehe Abschnitt 5.2.3.).
Die CALL-Anweisung hat die Form:

```
CALL nam [([p][,p]]...)]
```

Dabei ist

nam - der Name eines SUBROUTINE-Unterprogrammes oder eines formalen Parameters, der mit einem SUBROUTINE-Unterprogramm verbunden ist,
p - ein aktueller Parameter.

Aktueller Parameter kann in der CALL-Anweisung alles sein, was in Abschnitt 5.1. als aktueller Parameter aufgeführt wird, mit der Einschränkung, daß ein Zeichenkettenausdruck keine Verkettungsoperationen mit Zeichenketten mit übernommener Länge (siehe Abschnitt 5.1.3.) enthalten darf. Es sei hier noch einmal ausdrücklich darauf hingewiesen, daß alternative Rücksprünge nur als aktuelle Parameter in einer CALL-Anweisung verwendet werden dürfen (siehe Abschnitt 5.1.3.).

Die Ausführung einer CALL-Anweisung führt zu folgenden Aktionen:

- Berechnung der Ausdrücke, die aktuelle Parameter sind,
- Verbindung der aktuellen mit den formalen Parametern,
- Ausführung des SUBROUTINE-Unterprogrammes, beginnend mit der ersten ausführbaren Anweisung hinter der SUBROUTINE- oder ENTRY-Anweisung (siehe Abschnitt 5.2.4.).

Die CALL-Anweisung gilt als beendet, wenn die Steuerung aus dem aufgerufenen Unterprogramm zurückgegeben wird.

5.2.6. Rücksprung aus Unterprogrammen

Die Rückgabe der Steuerung aus einem Unterprogramm an die aufrufende Programmeinheit erfolgt durch die Ausführung einer RETURN- oder END-Anweisung. Die RETURN-Anweisung kann in FUNCTION- oder SUBROUTINE-Unterprogrammen verwendet werden und darf in einem Unterprogramm mehrfach auftreten.
Die RETURN-Anweisung in einem FUNCTION-Unterprogramm hat die Form:

```
RETURN
```

In einem SUBROUTINE-Unterprogramm ist

```
RETURN [i]
```

möglich.
Dabei ist

i - ein ganzzahliger Ausdruck, dessen Wert anzeigt, daß der i-te alternative Rücksprung aus der Liste der aktuellen Parameter einer CALL-Anweisung benutzt werden soll. Für den Wert von i muß gelten
$1 <= i <= n$,
wobei n die Anzahl von Sternen (*) in der Liste der formalen Parameter der SUBROUTINE- oder ENTRY-Anweisung ist.

Die Ausführung einer RETURN-Anweisung ohne alternativen Rücksprung bewirkt, daß in der aufrufenden Programmeinheit die Anweisung die Steuerung erhält, welche die Funktionsbezugnahme enthält oder der CALL-Anweisung folgt.
Wird eine RETURN-Anweisung mit alternativem Rücksprung ausgeführt, so erhält in der aufrufenden Programmeinheit die Anweisung die Steuerung, welche mit dem i-ten alternativen Rücksprung-Parameter der CALL-Anweisung verbunden ist.
Ist $i<1$ oder i n, dann wird i ignoriert und wie bei einer RETURN-Anweisung ohne alternativen Rücksprung verfahren, das heißt die erste ausführbare Anweisung nach der CALL-Anweisung erhält die

Steuerung.
Die Ausführung einer RETURN- oder END-Anweisung in einem Unterprogramm bewirkt, daß die Verbindung zwischen den aktuellen und formalen Parametern gelöst wird. Sie bewirkt aber auch, daß alle Objekte des Unterprogrammes als undefiniert zu betrachten sind, sofern sie nicht im Unterprogramm
- in einer SAVE-Anweisung verwendet werden oder
- als Objekte eines unbenannten COMMON-Bereiches vereinbart werden oder
- mit einer DATA-Anweisung initialisiert und im Verlauf des Unterprogrammes nicht neu definiert (z.B. durch Ergibtanweisung) oder undefiniert (z. B. durch EQUIVALENCE-Anweisung) werden oder
- als Objekte eines benannten COMMON-Bereiches vereinbart werden und dieser COMMON-Bereich auch in wenigstens einer Programmeinheit, die das Unterprogramm direkt oder indirekt aufruft, erscheint.

Die Objekte eines benannten COMMON-Bereiches, der im Hauptprogramm vereinbart wurde, können durch die Ausführung beliebiger RETURN- oder END-Anweisungen im ausführbaren Programm nicht undefiniert werden.

5.3. Standardfunktionen

FORTRAN77 bietet zur Lösung mathematischer und Textverarbeitungsprobleme Standardfunktionen. Die Anlage 2 enthält eine Übersicht aller verfügbaren Standardfunktionen. Der Aufruf einer FORTRAN 77-Standardfunktion erfolgt auf die gleiche Weise wie der Aufruf eines FUNCTION-Unterprogrammes (siehe Abschnitt 5.2.3.).
Die Anlage 2 enthält für jede Standardfunktion den Datentyp der Standardfunktion selbst und den der zugehörigen aktuellen Parameter. Der Abschnitt 5.3.3. beschreibt die Standardfunktionen zur Textverarbeitung.

Bezugnahmen auf Standardfunktionen

Gewöhnlich wird über die Namen aus Anlage 2 eine FORTRAN77-Standardfunktion aufgerufen. Diese Namen sind jedoch nicht für Standardfunktionen reserviert und man kann sich auf ein nutzergeschriebenes Unterprogramm mit dem Namen einer Standardfunktion beziehen, wenn dieser Name in einer EXTERNAL-Anweisung auftritt. Außer in diesem Fall, wo ein Standardfunktionsname in einer EXTERNAL-Anweisung steht, sind die Standardfunktionsnamen lokal zu der Programmeinheit, in der die Bezugnahme auf die Standardfunktion erfolgt. In anderen Programmeinheiten können solche Namen wieder zu anderen Zwecken benutzt werden. Es ist aber nicht möglich, innerhalb einer Programmeinheit auf eine Standardfunktion und ein nutzergeschriebenes Unterprogramm gleichen Namens Bezug zu nehmen.

Generische Funktionsbezugnahmen

Eine generische Standardfunktion ist eine Gruppe von Standardfunktionen mit gleichen Aufgaben, aber unterschiedlichen Datentypen der Parameter. Der Programmierer darf für jede Funktion der Gruppe denselben generischen Funktionsnamen benutzen und kann es dem Compiler überlassen, auf Grund des Datentyps der aktuellen Parameter den korrekten speziellen Funktionsnamen auszuwählen. Ein Beispiel für eine generische Funktion ist die Standardfunktion SIN. Hat zum Beispiel A den Datentyp REAL, dann bezieht sich die Funktionsbezugnahme SIN(A) auf die Sinus-Funktion mit dem Datentyp REAL. Ist D aber vom Datentyp DOUBLE PRECISION, dann bezieht sich SIN(D) auf die Sinus-Funktion mit dem Datentyp DOUBLE PRECISION. Es ist nicht notwendig, hierfür den gleichwertigen speziellen Funktionsaufruf DSIN(D) zu verwenden.
Die Auswahl einer speziellen Standardfunktion erfolgt für jede Bezugnahme auf einen generischen Standardfunktionsnamen unabhängig. Es ist also möglich, in einundderselben Programmeinheit SIN(A) und SIN(D) zu schreiben.
Die generischen Standardfunktionsnamen sind der Anlage 2 zu entnehmen. Dabei ist zu beachten, daß sie nur für die angegebenen Parameterdatentypen verwendbar sind.
Die Namen der generischen Standardfunktionen sind nicht zur Auswahl spezieller Standardfunktionen verwendbar, wenn sie schon als Name einer Anweisungsfunktion, eines formalen Parameters, als COMMON-Block-Name, als Variablen- oder als Feldname verwendet werden.
Das Auftreten eines generischen Standardfunktionsnamens in einer INTRINSIC-Anweisung berührt in keiner Weise die generischen Eigenschaften dieses Namens. Dagegen ist es verboten, einen generischen Standardfunktionsnamen als aktuellen Parameter anzugeben.
Generische Standardfunktionsnamen sind lokal zu der Programmein-

heit, in der auf sie Bezug genommen wird. In anderen Programmeinheiten können sie zu anderen Zwecken benutzt werden.

Standardfunktionen zur Textverarbeitung

Zur Textverarbeitung existieren Standardfunktionen für die Arbeit mit Zeichenketten und für lexikalische Vergleiche.

FORTRAN77 bietet vier Zeichenketten-Standardfunktionen:

LEN - Die LEN-Standardfunktion gibt die Länge eines Zeichenkettenausdruckes zurück.
Ihr Aufruf hat die Form:

```
LEN (c)
```

wobei c ein Zeichenkettenausdruck ist.
Der zurückgegebene Wert gibt die Zahl der aufeinanderfolgenden Zeichen an, die der Zeichenkettenausdruck belegt.

INDEX - Die INDEX-Standardfunktion sucht in einer Zeichenkette c1 nach der Teilkette c2, und gibt, wenn sie die Teilkette gefunden hat, die Startposition der Teilkette c2 in der Zeichenkette c1 zurück. Tritt c2 in c1 mehrfach auf, wird die Position des ersten Auftretens von links zurückgegeben. Ist c2 nicht in c1 enthalten, wird der Wert Null zurückgegeben.
Der Aufruf hat die Form:

```
INDEX(c1,c2)
```

Dabei ist:

- c1 - ein Zeichenkettenausdruck, in dem nach der Teilkette c2 gesucht wird.
- c2 - ein Zeichenkettenausdruck, der die Teilkette darstellt, deren Startposition in c1 zu bestimmen ist.

ICHAR - Die ICHAR-Standardfunktion wandelt ein Zeichen in eine Zahl um. Dabei entspricht der Wert der ganzen Zahl der Position des Zeichens in der Sortierfolge des Codes (siehe [1]). Die Zahl 0 entspricht also dem 1. Zeichen, die Zahl n-1 dem letzten Zeichen dieser Sortierfolge, wobei n die Anzahl aller Zeichen in der Sortierfolge ist.
Der Aufruf von ICHAR hat folgende Form:

```
ICHAR(c)
```

Dabei ist c ein Zeichenkettenausdruck der Länge 1.

CHAR - Die CHAR-Standardfunktion wandelt einen ganzzahligen Wert in eine Zeichenkette der Länge 1 um. Das zurückgegebene Zeichen ist das i-te Zeichen in der Sortierfolge des Codes (siehe [1]).
Der Aufruf von CHAR hat folgendes Aussehen:

```
CHAR(i)
```

Dabei ist i ein ganzzahliger Ausdruck.

Für den lexikalischen Vergleich von Zeichenkettenausdrücken existieren vier Standardfunktionen. Grundlage des Vergleichs ist die ASCII-Codierung dieser Ausdrücke.
Der Aufruf der Standardfunktionen für den lexikalischen Vergleich hat folgendes Aussehen:

```
fkt(c1,c2)
```

Dabei ist:

fkt - einer der Namen LLT, LLE, LGT oder LGE
c1, c2 - Zeichenkettenausdrücke

Die Standardfunktion gibt den logischen Wert .TRUE. zurück, wenn die Vergleichsbedingung LT, LE, GT oder GL (siehe Abschnitt 2.7.3.) zwischen dem ersten und zweiten Parameter des Funktionsaufrufes erfüllt ist. Andernfalls gibt sie den Wert .FALSE. zurück.

5.4. BLOCK DATA-Unterprogramme

BLOCK DATA-Unterprogramme sind Programmeinheiten, die mit einer BLOCK DATA-Anweisung beginnen. Sie dürfen nur IMPLICIT-, PARAMETER-, DIMENSION-, COMMON-, EQUIVALENCE-, SAVE-, DATA- und Typvereinbarungsanweisungen enthalten. Abgeschlossen wird ein BLOCK DATA-Unterprogramm durch eine END-Anweisung.
Die BLOCK DATA-Anweisung hat folgenden Aufbau:

```
BLOCK DATA[nam]
```

Dabei ist:

nam - der Name des BLOCK DATA-Unterprogrammes

BLOCK DATA-Unterprogramme sind nicht aufrufbar. Sie dienen lediglich der Initialisierung von Objekten in benannten COMMON-Bereichen.
Ein ausführbares Programm darf nur ein unbenanntes BLOCK DATA-Unterprogramm enthalten.
Einundderselbe COMMON-Block darf nicht in mehreren BLOCK DATA-Unterprogrammen auftreten.
Soll irgendeinem Objekt eines benannten COMMON-Bereiches mit der DATA-Anweisung ein Anfangswert zugewiesen werden, so ist unbedingt zu beachten, daß alle Objekte des COMMON-Bereiches vollständig vereinbart sein müssen, auch wenn nicht alle Objekte in einer DATA-Anweisung auftreten. Es führt also zu einem Fehler, wenn bei-

spielsweise das Objekt A eines COMMON-Bereiches ein Feld mit der Dimension (1:10) und dem Datentyp DOUBLE PRECISION sein soll und folgendes BLOCK DATA-Unterprogramm existiert:

```
BLOCK DATA
INTEGER B,C
COMMON /CBER/ A,B,C
DATA B/10/
END
```

Die Anweisung:

```
DOUBLE PRECISION A(1:10)
```

ist in das BLOCK DATA-Unterprogramm aufzunehmen.

6. E/A-Anweisungen

E/A-Anweisungen erlauben den Datentransport zwischen dem internen Speicher und externen oder internen Dateien. Dabei können die Daten während der Ein- oder Ausgabe aufbereitet werden. Außerdem können durch E/A-Anweisungen die Eigenschaften von Dateien festgelegt werden, diese Eigenschaften abgefragt werden, Dateien positioniert werden und Verbindungen zwischen Dateien und externen Geräten hergestellt werden.
Die Ein- und Ausgabe und somit auch die in den E/A-Anweisungen benötigten E/A-Parameter sind zum Teil vom Betriebssystem abhängig. Auskunft darüber gibt [1].

6.1. Dateien und ihre Verarbeitung durch ein FORTRAN77-Programm

6.1.1. Sätze, Dateien, Einheiten

Ein Satz ist eine Folge von Zeichen oder eine Folge von Werten. Unformatisierte Sätze enthalten die Werte in der Form, wie sie vom Rechner in numerischen, logischen oder anderen Operationen benutzt wird. Formatisierte Sätze enthalten die Werte in einer Form, die sich durch Aufbereitung mit Hilfe von E/A-Anweisungen in die rechnerinterne Form überführen läßt oder die durch Aufbereitung aus der rechnerinternen Form entstanden ist. Zum Beispiel wird man in Drucklisten für numerische Werte keine rechnerinterne Form wählen, sondern eine aufbereitete, besser lesbare Darstellung.
Eine Datei ist eine Folge von Sätzen. Sie wird durch folgende Eigenschaften charakterisiert:

- Zugriffsart (sequentieller Zugriff oder Direktzugriff auf die Sätze)
- Satzlänge (feste Länge oder variable Länge)
- Art der Sätze (formatisiert oder unformatisiert)
- Positionierung (Festlegung eines aktuellen Satzes)
- Dateiname

Die Zugriffsart bestimmt die Reihenfolge, in der die Sätze einer Datei verarbeitet werden. Es gibt Dateien, die nur sequentiellen Zugriff erlauben (z.B. Dateien auf Magnetbändern) und Dateien, die sequentiellen Zugriff und Direktzugriff erlauben (z.B.Dateien auf Magnetplatten). Welche Zugriffsart für eine Datei benutzt wird, wird bei der Verbindung einer Datei mit einer Einheit festgelegt. Beim sequentiellen Zugriff ist die Reihenfolge der Verarbeitung die Reihenfolge, in der die Sätze geschrieben worden sind. Beim Direktzugriff erhält jeder Satz der Datei eine eindeutige ganzzahlige positive Zahl, die Satznummer, zugeordnet. Beim direkten Le-

sen oder Schreiben werden die einzelnen Sätze über ihre Satznummer identifiziert. Dadurch ist es möglich, die Sätze in beliebiger Reihenfolge zu verarbeiten, z.B. den Satz 4 zu schreiben, obwohl die Sätze 1-3 noch nicht geschrieben wurden. Sequentieller Zugriff auf eine Datei, die auch Direktzugriff gestattet, bedeutet, daß die Sätze in aufsteigender Folge der Satznummern verarbeitet werden.
Die Satzlänge wird bei der Erzeugung der Sätze festgelegt und gibt die Anzahl der Zeichen bzw. die Anzahl der Bytes an, aus denen Sätze bestehen. FORTRAN77 unterstützt Dateien mit fester Satzlänge (alle Sätze der Datei haben eine einheitliche Satzlänge) und mit variabler Satzlänge. Direktzugriff ist nur zu Dateien mit fester Satzlänge möglich. Die Sätze einer Datei sind entweder alle formatisiert oder alle unformatisiert.
Eine Einheit ist ein Mittel der Sprache FORTRAN77 für den Zugriff auf eine Datei. Erst wenn eine Datei mit einer Einheit verbunden ist, kann diese Datei verarbeitet werden. Die Verbindung einer Datei mit einer Einheit wird durch eine OPEN-Anweisung hergestellt und durch eine CLOSE-Anweisung wieder aufgehoben oder endet automatisch bei Beendigung der Programmausführung. Eine OPEN-Anweisung kann auch benutzt werden, um eine neue Datei anzulegen.
FORTRAN77 unterstützt zwei Standarddateien und die zugehörigen Einheiten. Diese Standarddateien für die formatisierte sequentielle Eingabe bzw. Ausgabe sind bei Beginn eines ausführbaren Programms mit ihren Einheiten verbunden und erfordern keine OPEN-Anweisung. Auf diese Standarddateien wird zugegriffen, wenn im UNIT-Parameter für die Einheitennummer ein Stern angegeben wird. Außer den Standarddateien können noch weitere Dateien bei Programmbeginn mit ihren Einheiten verbunden sein.

6.1.2. Einheitennummer und ihre Verbindung mit einer Datei

Damit FORTRAN77-Programme weitgehend unabhängig vom speziellen Rechner und Betriebssystem sind, beziehen sich alle E/A-Anweisungen über Einheitennummern auf Einheiten. Welche Dateien mit diesen Einheiten verarbeitet werden sollen und auf welchen Geräten sich diese Dateien befinden bzw. wo sie angelegt werden sollen, wird entweder vor Ausführung des Programmes festgelegt (siehe [1]), oder die Verbindung zur Datei wird erst bei Ausführung einer OPEN-Anweisung hergestelt.
Eine Einheit darf zur gleichen Zeit höchstens mit einer Datei verbunden sein und umgekehrt. Bestehende Verbindungen können durch CLOSE-Anweisungen aufgehoben werden. Dabei muß die CLOSE-Anweisung nicht in derselben Programmeinheit sein wie eine evtl. vorangegangene OPEN-Anweisung für dieselbe Datei. Nach einer CLOSE-Anweisung sind Einheit und Datei wieder frei und können an anderer Stelle des ausführbaren Programms durch OPEN-Anweisungen erneut miteinander oder mit einer anderen Einheit bzw. Datei verbunden

werden. Dabei verlangt die Wiederherstellung einer Verbindung, die schon vor der Ausführung des Programms festgelegt wurde, keine OPEN-Anweisung ,wenn sich die Dateieigenschaften nicht ändern sollen. Eine Einheit, die nicht verbunden ist, darf nur in OPEN-, CLOSE- und INQUIRE-Anweisungen benutzt werden. Auf eine Datei, die nicht verbunden ist, kann nur in einer OPEN- oder INQUIRE-Anweisung mit Hilfe ihres Dateinamens Bezug genommen werden. INQUIRE-Anweisungen liefern dem Programm Informationen über Dateien, Einheiten und Verbindungen zwischen ihnen.
Bei Programmende noch bestehende Verbindungen zwischen Einheiten und Dateien werden implizit aufgelöst, als wäre für sie eine CLOSE-Anweisung ohne Angabe des STATUS-Parameters ausgeführt worden.

6.1.3. Interne Dateien

Interne Dateien sind ein Mittel, um Daten im internen Speicher zu transportieren und dabei aufzubereiten. Eine interne Datei ist eine Variable, ein Feldelement oder ein Feld vom Datentyp CHARACTER oder eine Teilkette. Ist eine interne Datei ein Feld, so ist jedes Feldelement ein Satz. Die Reihenfolge der Sätze entspricht der Reihenfolge der Elemente im Feld. In allen anderen Fällen hat die interne Datei nur einen Satz.
Interne Dateien können nur in READ- oder WRITE-Anweisungen benutzt werden. Die Ein- oder Ausgabe muß sequentiell und format-gesteuert sein. LIST-gesteuerte Ein- oder Ausgabe ist nicht möglich. Die interne Datei darf keine im FMT-Parameter angegebene Variable enthalten und auch nicht durch eine EQUIVALENCE-Anweisung mit solch einer Variablen verbunden sein. Eine interne Datei wird vor jedem Datentransport auf den Anfang des ersten Satzes positioniert. Das bedeutet, daß auch ein Zugriff auf mehrere Sätze der Datei durch eine einzige READ oder WRITE-Anweisung erfolgen muß.
Eine Variable, ein Feldelement oder eine Teilkette als Satz einer internen Datei wird definiert, wenn dieser Satz geschrieben wird. Ist die Anzahl der Zeichen, die in den Satz geschrieben werden, kleiner als die Satzlänge, wird der Rest des Satzes mit Leerzeichen aufgefüllt. Die Variable, das Feldelement oder die Teilkette muß definiert sein, wenn der entsprechende Satz gelesen werden soll. Dabei können die Sätze einer internen Datei nicht nur durch WRITE-Anweisungen definiert werden, sondern z.B. auch durch Ergibtanweisungen.

6.1.4. Fehlerbehandlung

Während der Ausführung von E/A-Anweisungen können Bedingungen eintreten, die eine ordnungsgemäße Beendigung der Anweisung unmöglich machen.

Das kann beispielsweise in folgenden Fällen eintreten:

- Bei der Berechnung von Ausdrücken in der Datenliste werden Funktionen verwendet, deren Berechnung die Ausführung von E/A-Anweisungen fordert.
- Ein Datenlistenelement paßt nicht zu einem Formatlistenelement.
- Die Datei ist nicht für direkten Zugriff verbunden. Die E/A-Anweisung fordert aber einen direkten Zugriff.
- Die Dateiende-Bedingung tritt ein.

Die Behandlung solcher Zustände kann durch das FORTRAN77-Programm selbst erfolgen, wenn in der Liste der Steuerinformationen der E/A-Anweisung der IOSTAT-Parammeter angegeben ist. Weiterhin stehen dazu der END- und der ERR-Parameter zur Verfügung.
Ist der IOSTAT-Parameter angegeben, erhält unabhängig vom Vorhandensein des END- oder ERR-Parameters, die im IOSTAT-Parameter angegebene ganzzahlige Variable oder das angegebene ganzzahlige Feldelement einen den eingetretenen Zustand dokumentierenden Wert. Eine Liste dieser Werte ist in [1] enthalten.
Tritt ein Fehlerzustand auf, so wird die Arbeit der E/A-Anweisung abgebrochen, und die Position der Datei wird unbestimmt. Ist der ERR-Parameter angegeben, so wird die Ausführung des Programmes mit der Anweisung fortgesetzt, deren Marke im ERR-Parameter angegeben ist. Ist weder der ERR- noch der IOSTAT-Parameter angegeben, wird die Ausführung des Programms beendet, sonst wird die Arbeit mit der nächsten ausführbaren Anweisung fortgesetzt. Analog gilt: Wird während einer READ-Anweisung Dateiende festgestellt, ohne daß gleichzeitig ein Fehlerzustand aufgetreten ist, so wird die READ-Anweisung abgebrochen. Ist der END-Parameter angegeben, so wird das Programm mit der dort angezeigten Anweisung fortgesetzt. Ist weder der END- noch der IOSTAT-Parameter angegeben, wird die Ausführung des Programmes beendet.

6.2. Überblick über die E/A-Parameter

Die in E/A-Anweisungen auftretenden E/A-Parameter dienen folgenden Zwecken:

- Steuerung der Auführung der E/A-Anweisung
- Erhalt von Informationen über die erfolgreiche oder nicht erfolgreiche Ausführung der E/A-Anweisung
- Erhalt von Informationen über eine Datei und über eine eventuell vorhandene Verbindung der Datei zu einer Einheit

Ein E/A-Parameter ist ein Element der Liste der Steuerinformationen und besteht aus einem Schlüsselwort, dem Gleichheitszeichen und entweder einem Ausdruck, dessen Wert für die Ausführung der E/A-Anweisung von Bedeutung ist oder einem Objekt der Programmein-

heit, die die E/A-Anweisung enthält. Dieses Objekt erhält durch die Ausführung der E/A-Anweisung einen Wert. Die E/A-Parameter stehen in beliebiger Reihenfolge, werden durch Kommas getrennt und die gesamte Liste der Steuerinformationen wird von einem Klammernpaar eingeschlossen. Jeder E/A-Parameter darf höchstens einmal angegeben werden.
Enthält die Liste der Steuerinformationen einen UNIT-Parameter oder einen UNIT- und einen FMT-Parameter, können die Schlüsselworte UNIT und FMT mit dem Gleichheitszeichen weggelassen werden, wenn der UNIT-Parameter das erste und der FMT-Parameter das zweite Element der Liste der Steuerinformationen ist.
Die PRINT-Anweisung enthält nur den FMT-Parameter. Die READ-Anweisung kann sowohl mit einer Liste der Steuerinformationen als auch nur mit einem FMT-Parameter geschrieben werden.
In der BACKSPACE-, REWIND- und ENDFILE-Anweisung kann der UNIT-Parameter ohne die Zeichenfolge UNIT= und ohne das die Liste der Steuerinformationen umschließende Klammernpaar angegeben werden, wenn er das einzigste Element der Liste der Steuerinformationen ist.

Tabelle 6-1 E/A-Parameter

Schlüsselwort des E/A-Parameters	E/A-Anweisung								
	READ		WRITE	PRINT	OPEN	CLOSE	INQUIRE	BACKSPACE REWIND ENDFILE	
	mit cilist	ohne cilist							
UNIT	A		A		A	A	G	B	
FMT	C	D	C	D					
REC	E		E						
END	F								
ERR IOSTAT	F		F		F	F	F	F	
FILE					F		G		
STATUS					F	F			
ACCESS					F		F		
FORM									
BLANK									
RECL					E		F		
EXIST							F		
OPENED									
NUMBER									
NAMED									
NAME									
SEQUENTIAL									
DIRECT									
FORMATTED									
UNFORMATTED									
NEXTREC									

Erläuterung zur Tabelle 6-1.

cilist - Liste der Steuerinformationen
A - Parameter muß angegeben werden.
Wird die Zeichenfolge UNIT= weggelassen, muß der Parameter am Anfang der Liste der Steuerinformationen stehen
B - Parameter muß angegeben werden.
Wird die Zeichenfolge UNIT= weggelassen, muß der Parameter am Anfang der Liste der Steuerinformationen stehen.
Wird die Zeichenfolge UNIT= weggelassen und handelt es sich um den einzigsten Parameter der Liste der Steuerinformationen, können die diese Liste umschließenden Klammern weggelassen werden.
C - Parameter ist wahlweise angebbar.
Wird die Zeichenfolge FMT= weggelassen, muß der Parameter der zweite in der Liste der Steuerinformationen angegebene Parameter sein und der UNIT-Parameter muß ohne die Zeichenfolge UNIT= angegeben worden sein.
D - Parameter muß ohne Zeichenfolge FMT= angegeben werden und wird nicht in Klammern eingeschlossen.
E - Parameter muß angegeben werden, wenn zu den Sätzen der Datei direkt zugegriffen werden soll.
F - Parameter ist wahlweise angebbar.
G - Parameter müssen alternativ zueinander angegeben werden (entweder UNIT- oder FILE-Parameter)

6.3. Anweisungen zur Eingabe und Ausgabe von Daten

Durch eine READ-Anweisung werden Werte aus einer Datei gelesen und den in der Eingabedatenliste aufgeführten Elementen zugewiesen. WRITE- oder PRINT-Anweisungen bewirken, daß Werte entsprechend der Ausgabedatenliste und des FMT-Parameters in eine Datei gebracht werden. Eine WRITE- oder PRINT-Anweisung für eine Datei, die noch nicht existiert, bewirkt, daß diese Datei angelegt wird. Eine PRINT-Anweisung bezieht sich immer auf die Standardausgabeeinheit. Dieselbe Einheit kann in einer WRITE-Anweisung über UNIT = * erreicht werden. Die Ausgabe mittels der PRINT-Anweisung ist stets sequentiell und liefert formatisierte Sätze.
Eine spezielle Form der Ausgabe ist der Druck von Daten. Der Druck ist die sequentielle Ausgabe formatisierter Sätze auf speziellen Geräten, welche das erste Zeichen eines Satzes nicht ausgeben, sondern zur Steuerung des Zeilenvorschubs vor dem Druck der restlichen Zeichen des Satzes benutzen. Tabelle 6-2 gibt Auskunft über die möglichen Steuerzeichen für einen Drucker.
Im allgemeinen wird zum Drucken die PRINT-Anweisung benutzt. Jedoch ist auch Drucken mit Hilfe von WRITE-Anweisungen möglich, ebenso wie eine PRINT-Anweisung nicht unbedingt ein Drucken bewirkt. Entscheidend für das Drucken ist einzig und allein, ob die

Einheit, auf die sich die PRINT- oder WRITE-Anweisung bezieht, mit einem Drucker verbunden ist.

Tabelle 6-2 Druckersteuerzeichen

Steuerzeichen	Aktion vor dem Drucken des Satzes
Leerzeichen	Vorschub um eine Zeile
0	Vorschub um zwei Zeilen
1	Vorschub zur ersten Zeile auf der nächsten Seite
+	kein Vorschub

Eine E/A-Anweisung kann nur einen unformatisierten Satz, aber einen oder mehrere formatisierte Sätze übertragen. Formatisierte Sätze werden bei der Ausgabe in interne Dateien oder Direktzugriffsdateien bis zum Satzende mit Leerzeichen aufgefüllt, wenn die Ausgabedatenliste einen Satz nicht zu Ende füllen kann. In allen anderen Fällen ist entweder der Rest des Satzes undefiniert, oder es wird ein verkürzter Satz geschrieben.
Die E/A-Anweisungen zur Ein- und Ausgabe von Daten haben folgende Formen:

```
READ (cilist) [idlist]
READ f        [,idlist]
WRITE (cilist) [odlist]
PRINT f       [,odlist]
```

Die einzelnen Teile dieser Anweisung sind:

- Liste der Steuerinformationen 'cilist' (siehe Abschnitt 6.3.1.)
- FMT-Parameter 'f', der auch in der Liste der Steuerinformationen enthalten sein kann, zur Bestimmung der Art der Datenaufbereitung
- Eingabedatenliste 'idlist' mit den einzulesenden oder Ausgabedatenliste 'odlist' mit den auszugebenden Daten (siehe Abschnitt 6.3.2.)

Weiterhin gelten folgende Bedingungen:

- Enthält die READ-Anweisung keine Liste der Steuerinformationen. erfolgt die Eingabe von der gleichen Datei, als wenn UNIT=* in der Liste der Steuerinformationen angegeben wäre.
- Existiert keine Eingabedatenliste 'idlist', werden keine Daten eingelesen. Es werden lediglich die Ausdrücke der Liste der Steuerinformationen ausgewertet und Dateipositionierungsoperationen ausgeführt.
- Existiert keine Ausgabedatenliste 'odlist', werden die Ausdrücke der Liste der Steuerinformationen ausgewertet, Dateipositionierungsoperationen ausgeführt, sowie Zeichenkettenkonstanten ausgegeben, wenn diese im FMT-Parameter angegeben worden sind.

6.3.1. Steuerinformationen der READ-, WRITE- und PRINT-Anweisung

UNIT = u 'u' gibt an, an welche Einheit oder an welche interne Datei sich die E/A-Anweisung richtet, 'u' ist
- ein ganzzahliger Ausdruck mit einem Wert größer oder gleich Null. Der Wert des Ausdrucks liefert die Einheitennummer.
- ein Stern '*'. Dann ist eine Standardeinheit für formatisierten sequentiellen Zugriff gewünscht. Die bei Angabe von '*' verwendeten Einheiten können [1] entnommen werden.
- der Name einer Variablen, eines Feldes oder eines Feldelementes vom Datentyp CHARACTER oder der Name einer Teilkette. Diese Elemente werden von der E/A-Anweisung als interne Datei benutzt.

FMT = f 'f' gibt an, ob die format- oder list-gesteuerte Datenaufbereitung verwendet werden soll. Fehlt der FMT-Parameter, erfolgt keine Datenaufbereitung, d.h. es wird durch die E/A-Anweisung ein unformatisierter Satz erzeugt oder gelesen.
Für die format-gesteuerte Datenaufbereitung können folgende Angaben gemacht werden (siehe Abschnitt 7.):
- Marke einer FORMAT-Anweisung, die in der gleichen Programmeinheit wie die E/A-Anweisung enthalten ist
- Name einer ganzzahligen Variablen, der mittels einer ASSIGN-Anweisung die Marke einer FORMAT-Anweisung zugewiesen wurde
- Name eines Feldes vom Typ CHARACTER, wobei die Verkettung aller Feldelemente der Wert des FMT-Parameters ist
- Ausdruck vom Typ CHARACTER, wobei in einer Verkettung kein Operand enthalten sein darf, dessen Längenangabe '(*)' ist, sofern er nicht der Name einer symbolischen Konstanten ist

Die list-gesteuerte Datenaufbereitung, die nicht für die Verarbeitung von internen Dateien oder für die Verarbeitung von Dateien durch direkten Zugriff verwendet werden kann, wird durch Angabe eines Sterns '*' gefordert.

REC = rn 'rn' gibt die Satznummer des mittels direktem Zugriff zu verarbeitenden Satzes an.
'rn' muß ein ganzzahliger Ausdruck mit positivem Wert sein.

END = me 'me' muß die Marke einer ausführbaren Anweisung sein. Mit dieser Anweisung wird die Ausführung des Programmes fortgesetzt, wenn bei Ausführung der READ-Anweisung die Dateiende-Bedingung auftritt. Die Dateiende-Bedingung tritt auf, wenn versucht wird, einen Satz zu lesen, der hinter dem letzten Satz einer internen Datei liegt, oder wenn beim Verarbeiten des letzten Satzes einer für sequentiellen Zugriff verbundenen Datei das Dateiende erreicht wird.

ERR = mr 'mr' muß die Marke einer ausführbaren Anweisung sein. Mit dieser Anweisung wird die Ausführung des Programmes fortgesetzt, wenn bei der Eingabe oder bei der Ausgabe ein Fehler auftritt.
Nach Auftreten eines Fehlers ist die aktuelle Dateiposition unbestimmt. Ebenfalls sind unbestimmte Werte in den Eingabedatenlistenelementen enthalten. Ist in der E/A-Anweisung weder der ERR-Parameter noch der IOSTAT-Parameter vorhanden und es tritt ein Fehler auf, wird die Ausführung des Programmes abgebrochen.

IOSTAT=ios 'ios' muß der Name einer ganzzahligen Variablen oder der Name eines ganzzahligen Feldelementes sein. Bei der Ausführung der E/A-Anweisung erhält 'ios' einen Wert, der die Ausführung der E/A-Anweisung dokumentiert.
Dieser Wert ist:
- 0, wenn kein Fehler erkannt wurde und die Dateiende-Bedingung nicht aufgetreten ist,
- negativ, wenn die Dateiende-Bedingung aufgetreten ist und kein Fehler erkannt wurde,
- positiv, wenn ein Fehler erkannt wurde.

Die Liste der möglichen Werte ist in [1] enthalten.

6.3.2. Datenlisten der READ-, WRITE- und PRINT-Anweisungen

Eine Eingabedatenliste kann

- Namen von Variablen,
- Feldelemente,
- Teilketten,
- Namen von Feldern und
- implizite DO-Listen

enthalten.

Eine Ausgabedatenliste kann

- Ausdrücke,
- Namen von Variablen,
- Feldelemente,
- Teilketten,
- Namen von Feldern und
- implizite DO-Listen

enthalten.

Die Angabe des Namens eines Feldes ist äquivalent zu der Angabe aller Feldelemente in der Reihenfolge, die durch die Anordnung der Feldelemente im Speicher gegeben ist. Der Name eines Feldes darf kein Feld bezeichnen, das ein formaler Parameter von angenommener Größe ist.
In einem Ausdruck darf keine Verkettung vorhanden sein, bei der ein Operand die Längenangabe '(*)' hat und nicht der Name einer symbolischen Konstanten ist.
In einer READ-Anweisung darf ein Element der Eingabedatenliste nicht Teil der Angabe im FMT-Parameter sein und darf auch durch eine EQUIVALENCE-Anweisung nicht mit dieser Angabe verbunden sein. Das gleiche gilt für ein Element einer Eingabe- oder Ausgabedatenliste und für eine im UNIT-Parameter angegebene interne Datei.
Eine implizite DO-Liste hat das Format:

```
(dlist, do_var = anf_wert, end_wert[,sw])
```

Es gilt:

dlist	ist eine Eingabedatenliste, wenn die implizite DO-Liste in einer READ-Anweisung enthalten ist, sonst eine Ausgabedatenliste.
do_var anf_wert end_wert sw	entsprechen den Elementen einer DO-Anweisung.

Die Datenliste einer impliziten DO-Liste darf wiederum implizite DO-Listen enthalten.
Der Schleifenzähler und die Werte für die DO-Variable do_var werden nach den gleichen Regeln berechnet, die für die DO-Anweisung gelten.
Die DO-Variable oder eine durch eine EQUIVALENCE-Anweisung mit der DO-Variablen verbundene Variable, darf nicht in der Datenliste der impliziten DO-Liste enthalten sein. Enthält eine E/A-Anweisung eine implizite DO-Liste, wird die in ihr enthaltene Datenliste so oft abgearbeitet, wie durch den Schleifenzähler gefordert wird. Dabei wird bei jedem Auftreten der DO-Variablen ihr aktueller Wert verwendet.

6.3.3. Beispiele für READ-, WRITE- und PRINT-Anweisungen

```
1.
   100 FORMAT(I5,F8.2)
       READ 100, IVAR,REAL
       READ(5,100) IVAR,REAL
2.
       READ(3,'(I5,A9)',END=20,IOSTAT=IOS) IVAR,ZK
3.
       READ 200,IV,(IFELD(I),I=1,IV)
4.
       PRINT
       100,A,((((FELD(I,J,K,L),I=1,20),J=1,5,1),K=7,1,-1),L=1,9)
5.
       WRITE (ERR=90,IOSTAT=IOS,UNIT=IUNIT) 'ABCDE'//ZK
       WRITE(FMT=100,UNIT=7) VAR,B,C,D, (LIST(K),K=19,5,-3),F,G,H
```

6.4. OPEN-Anweisung

Mittels der OPEN-Anweisung kann eine Verbindung zwischen einer Einheit und einer Datei hergestellt und eine bestehende Verbindung modifiziert werden (siehe auch 6.4.2).
Die OPEN-Anweisung hat die Form:

```
OPEN (olist)
```

Die Liste der Steuerinformationen 'olist' muß einen UNIT-Parameter und kann je einen ERR-, IOSTAT-, FILE-, STATUS-, ACCESS-, FORM-, RECL- und BLANK-Parameter enthalten (siehe Abschnitt 6.4.1.).

6.4.1. E/A-Parameter der OPEN-Anweisung

UNIT = u
'u' ist ein ganzzahliger Ausdruck, dessen Wert (0 oder positiv) eine zulässige Einheitennummer sein muß.
Nach erfolgreicher Beendigung der Ausführung der OPEN-Anweisung ist in allen Programmeinheiten des ausführbaren Programmes die Einheit mit einer Datei verbunden, solange keine CLOSE-Anweisung für diese Einheit ausgeführt wird.
Ist die Einheit bereits mit einer Datei verbunden, wird die Verbindung gemäß Abschnitt 6.4.2. modifiziert.

ERR = mr
'mr' muß die Marke eine ausführbaren Anweisung sein.
Mit dieser Anweisung wird die Ausführung des Programms fortgesetzt, wenn bei der Ausführung der OPEN-Anweisung ein Fehler auftritt.
Ist weder der ERR-Parameter noch der IOSTAT-Parameter angegeben und es tritt ein Fehler auf, wird die Ausführung des Programmes abgebrochen.

IOSTAT=ios
'ios' muß der Name einer ganzzahligen Variablen oder der Name eines ganzzahligen Feldelementes sein. Bei der Ausführung der OPEN-Anweisung erhält 'ios' einen Wert, der die Ausführung dokumentiert:
Dieser Wert ist:
- 0, wenn kein Fehler erkannt wurde,
- positiv, wenn ein Fehler erkannt wurde.
Die Liste der möglichen Werte ist in [1] enthalten.

FILE = fn
'fn' muß ein Ausdruck vom Typ CHARACTER sein, dessen Wert (ohne abschließende Leerzeichen) ein gültiger Dateiname ist und die mit einer Einheit zu verbindende Datei benennt.
Fehlt der FILE-Parameter und es besteht noch keine Verbindung der Einheit zu einer Datei, wird eine temporäre Datei gemäß [1] verwendet, die nur für die Zeit dieser Verbindung existiert. Fehlt der FILE-Parameter und es besteht bereits eine Verbindung zu einer Datei, wird die OPEN-Anweisung gemäß Abschnitt 6.4.2. ausgeführt.
Die gültigen Dateinamen sind in [1] beschrieben.

STATUS=st 'st' muß ein Ausdruck vom Typ CHARACTER sein, der ohne abschließende Leerzeichen die Werte
OLD, NEW, SCRATCH, UNKNOWN
haben darf, und Aussagen über den Status der zu verbindenden Datei macht.

OLD - Der FILE-Parameter muß angegeben werden und die in ihm angegebene Datei muß bereits existieren.

NEW - Der FILE-Parameter muß angegeben werden und die in ihm angegebene Datei darf noch nicht existieren.

SCRATCH - Der FILE-Parameter darf nicht angegeben werden und die zu verbindende Einheit darf nicht mit einer Datei verbunden sein. Die Einheit wird dann mit einer temporären Datei gemäß [1] verbunden, wobei diese Datei nur für die Zeit ihrer Verbindung mit einer Einheit existiert.

UNKNOWN - Ist der Dateiname der Datei, zu der eine Verbindung hergestellt werden soll (entweder durch eine bereits bestehende Verbindung oder durch die Angabe im FILE-Parameter) bereits bekannt und diese Datei existiert noch nicht, wird der Status NEW angenommen. Existiert diese Datei bereits, gilt der Status OLD. Ist kein Dateiname bekannt, wird der Status SCRATCH angenommen.

Fehlt der STATUS-Parameter oder hat er einen ungültigen Wert, wird UNKNOWN angenommen.

ACCESS=ac 'ac' muß ein Ausdruck vom Typ CHARACTER sein, der ohne abschließende Leerzeichen die Werte SEQUENTIAL und DIRECT haben darf, und die Methode des Zugriffes zu den Sätzen der zu verbindenden Datei festlegt.
Fehlt der ACCESS-Parameter, wird angenommen, daß sequentiell zugegriffen werden soll.

FORM = fm 'fm' muß ein Ausdruck vom Typ CHARACTER sein, der ohne abschließende Leerzeichen die Werte FORMATTED oder UNFORMATTED haben darf, und die Art der Sätze festlegt, die die zu verbindende Datei hat oder erhalten soll. Damit wird auch festgelegt, ob entweder format- oder list-gesteuerte Datenaufbereitung oder keine Datenaufbereitung bei der Ein- und Ausgabe erlaubt ist.
Fehlt der FORM-Parameter wird UNFORMATTED angenommmen, wenn die Datei für direkten Zugriff verbunden wird, und es wird FORMATTED angenommen, wenn die Datei für sequentiellen Zugriff verbunden wird.

RECL = rl 'rl' muß ein ganzzahliger Ausdruck sein, dessen Wert die Länge der Sätze der Datei in Bytes festlegt.
Der RECL-Parameter muß angegeben werden und 'rl' muß positiv sein, wenn die Datei für direkten Zugriff verbunden wird.
Für sequentiellen Zugriff darf der RECL-Parameter nicht benutzt werden.

BLANK = bl 'bl' muß ein Ausdruck vom Typ CHARACTER sein, der ohne abschließende Leerzeichen die Werte NULL und ZERO haben darf und die Interpretation von Leerzeichen in numerischen Eingabefeldern beim Lesen von formatisierten Sätzen festlegt, wenn diese Leerzeichen keine führenden Leerzeichen sind. Der Wert von 'bl' wird bei allen Zugriffen auf diese Datei benutzt, sofern er nicht in einer READ-Anweisung durch die Formatelemente BN oder BZ überschrieben wird (siehe Abschnitt 7.).
NULL - Leerzeichen werden ignoriert.
Enthält ein numerisches Eingabefeld nur Leerzeichen, wird der Wert 0 eingelesen.
ZERO - Leerzeichen in einem numerischen Eingabefeld werden als Zeichen 0 interpretiert.
Fehlt der BLANK-Parameter wird BLANK = NULL angenommen.

6.4.2. OPEN-Anweisung mit schon verbundener Einheitennummer

Das Resultat einer OPEN-Anweisung für eine Einheit, die schon mit einer Datei verbunden ist, ist abhängig von der Angabe des FILE-Parameters und von der Existenz der Datei.
Wurde in der OPEN-Anweisung kein FILE-Parameter angegeben und die Einheit ist bereits mit einer Datei verbunden, bleibt diese Verbindung bestehen.
Soll die Einheit mit der gleichen Datei verbunden werden, mit der sie schon verbunden ist, und diese Datei existiert bereits, wird nur ein eventuell anderer Wert des BLANK-Parameters für diese Verbindung wirksam.
Existiert die Datei jedoch noch nicht und die Verbindung wurde vor Beginn der Ausführung des Programmes hergestellt, erhält die Datei alle durch die OPEN-Anweisung geforderten Eigenschaften.
Ist jedoch im FILE-Parameter der Name einer anderen Datei angegeben, als der Datei, mit der die Einheit entweder schon bei Beginn der Ausführung des Programmes oder nach Ausführung einer OPEN-Anweisung verbunden ist, so wird die bestehende Verbindung gelöst und dann die neue Verbindung hergestellt. Die Lösung der Verbindung erfolgt wie durch eine CLOSE-Anweisung ohne Angabe eines STATUS-Parameters.

6.5. CLOSE-Anweisung

Mittels der CLOSE-Anweisung wird eine bestehende Verbindung zwischen einer Einheit und einer Datei aufgehoben.
Die CLOSE-Anweisung hat die Form:

CLOSE (clist)

wobei die Liste der Steuerinformationen 'clist' die im folgenden beschriebenen E/A-Parameter enthält. Dabei muß ein UNIT-parameter und kann jeweils ein IOSTAT-, ERR- oder STATUS-Parameter angegeben werden.

UNIT = u 'u' muß ein ganzzahliger Ausdruck sein, dessen Wert (0 oder positiv) die Einheitennummer der zu lösenden Verbindung ist.
Existiert keine derartige Verbindung, hat die Ausführung der CLOSE-Anweisung keine Auswirkung auf irgendeine Datei.

ERR = mr, IOSTAT = ios
Für den ERR- und für den IOSTAT-Parameter gilt das für diese Parameter bei der OPEN-Anweisung (Abschnitt 6.4.1.) beschriebene.

STATUS = st 'st' muß ein Ausdruck von Typ CHARACTER sein, der ohne abschließende Leerzeichen die Werte KEEP und DELETE annehmen darf und Aussagen über die Existenz der Datei nach der Auflösung ihrer Verbindung mit einer Einheit macht.

- KEEP - Eine vor Ausführung der CLOSE-Anweisung existierende Datei existiert auch weiterhin. Für eine temporäre Datei (mit dem Status SCRATCH) darf KEEP jedoch nicht angegeben werden.
- DELETE - Nach Ausführung der CLOSE-Anweisung existiert die mit der Einheit verbundene Datei nicht mehr.

Wird der STATUS-Parameter weggelassen, so wird DELETE angenommen, wenn die Verbindung zu einer Datei mit dem Status SCRATCH gelöst wird, anderenfalls wird KEEP angenommen.

6.6. BACKSPACE-, ENDFILE- und REWIND-Anweisung

Nach Ausführung der Dateipositionierungsanweisungen BACKSPACE und REWIND können schon verarbeitete Sätze von Dateien mit sequentiellem Zugriff nochmals verarbeitet werden. Mittels der ENDFILE-Anweisung kann das Ende einer Datei für sequentiellen Zugriff markiert werden, so daß bei nachfolgenden Eingabevorgängen das Ende der Datei erkannt werden kann. Die Positionierungsanweisungen sind nur für Dateien möglich, auf die sequentiell zugegriffen wird. Explizite Positionierung interner Dateien ist nicht möglich. Die Form dieser Anweisungen ist:

```
BACKSPACE u
BACKSPACE (fplist)
REWIND    u
REWIND    (fplist)
ENDFILE   u
ENDFILE   (fplist)
```

wobei die Liste der Steuerinformationen 'fplist' einen UNIT-Parameter enthalten muß und jeweils einen IOSTAT- und ERR-Parameter enthalten kann. Die Liste der Steuerinformationen reduziert sich zu dem im UNIT-Parameter anzugebenden Ausdruck, wenn nur der UNIT-Parameter angegeben werden soll.
Die Steuerinformationen sind folgende:

UNIT = u 'u' muß ein ganzzahliger Ausdruck sein, dessen Wert (0 oder positiv) die Nummer einer Einheit ist, die mit einer Datei für sequentiellen Zugriff verbunden ist.

ERR = mr, IOSTAT = ios
Für den ERR- und für den IOSTAT-Parameter gilt das für diese Parameter bei der OPEN-Anweisung (siehe Abschnitt 6.4.1.) beschriebene.

BACKSPACE-Anweisung

Nach Ausführung einer BACKSPACE-Anweisung kann der zuletzt verarbeitete Satz einer Datei nochmals verarbeitet werden, d.h. er kann überschrieben und er kann nochmals gelesen werden.
Wurde noch kein Satz der Datei verarbeitet, hat die Ausführung einer BACKSPACE-Anweisung keine Auswirkungen darauf, welcher Satz als nächster zu verarbeiten ist.
Nach Ausführung einer ENDFILE-Anweisung müssen für diese Datei zwei BACKSPACE-Anweisungen ausgeführt werden, damit der letzte Satz der Datei erneut verarbeitet werden kann.
Wurde der zuletzt geschriebene Satz durch list-gesteuerte Datenaufbereitung erzeugt, darf keine BACKSPACE-Anweisung ausgeführt werden.

REWIND-Anweisung

Nach Ausführung einer REWIND-Anweisung kann der erste Satz der Datei überschrieben oder erneut gelesen werden oder, wenn er noch nicht existiert, erzeugt werden.

ENDFILE-Anweisung

Die ENDFILE-Anweisung schreibt einen Endekennsatz als nächsten Satz auf die Datei. Nach der Ausführung einer ENDFILE-Anweisung muß eine BACKSPACE- oder REWIND-Anweisung ausgeführt werden, bevor wieder Sätze der Datei gelesen oder überschrieben werden können oder die Datei erweitert werden kann. Erlaubt diese Datei auch direkten Zugriff, so können bei Direktzugriff nur die Sätze gelesen werden, die vor dem Endekennsatz geschrieben wurden. Eine ENDFILE-Anweisung für eine Datei, die mit einer Einheit verbunden ist, aber noch nicht existiert, legt diese Datei an.

6.7. INQUIRE-Anweisung

Mittels einer INQUIRE-Anweisung werden Informationen über bestehende Verbindungen zwischen einer Einheit und einer Datei, aber auch Informationen über eine nicht mit einer Einheit verbundene Datei dem Programm zur Verfügung gestellt.
Die INQUIRE-Anweisung hat die Form:

INQUIRE (iqlist)

wobei die Liste der Steuerinformationen entweder einen UNIT-Parameter oder einen FILE-Parameter enthalten muß und jeweils einen IOSTAT-, ERR-, EXIST-, OPENED-, NUMBER-, NAMED-, NAME-, ACCESS-, SEQUENTIAL-, DIRECT-, FORM-, FORMATTED-, UNFORMATTED-, RECL-, NEXTREC- und BLANK-Parameter enthalten kann.
Wird der UNIT-Parameter angegeben, informieren die Werte der angegebenen E/A-Parameter (außer FILE-, IOSTAT- und ERR-Parameter) über die im UNIT-Parameter angegebene Einheit und sofern diese mit einer Datei verbunden ist, über diese Verbindung und über die Datei.
Wird der FILE-Parameter angegeben, geben die Werte der angegebenen E/A-Parameter (außer UNIT-, IOSTAT- und ERR-Parameter) Auskunft über die im FILE-Parameter angegebene Datei und über deren Verbindung zu einer Einheit, falls diese Verbindung existiert.
In den folgenden Erläuterungen der E/A-Parameter wird eine INQUIRE-Anweisung mit UNIT-Parameter als INQUIRE/U-Anweisung und eine INQUIRE-Anweisung mit FILE-Parameter als INQUIRE/F-Anweisung bezeichnet.
Die Zuweisung der Werte erfolgt gemäß den Regeln der Ergibtanwei-

sung.

Folgende Parameter sind angebbar:

UNIT = u 'u' muß ein ganzzahliger Ausdruck sein, dessen Wert (0 oder positiv) die Nummer der Einheit ist, über die Informationen gegeben werden sollen.

FILE = fn 'fn' muß ein Ausdruck vom Typ CHARACTER sein, dessen Wert (ohne abschließende Leerzeichen) der Name der Datei ist, über die Informationen gegeben werden sollen.

ERR = mr, IOSTAT = ios
Für den ERR- und für den IOSTAT-Parameter gilt das für diese Parameter bei der OPEN-Anweisung (Abschnitt 6.4.1.) beschriebene.

EXIST = ex
OPENED = op
NAMED = nmd
'ex', 'op' und 'nmd' müssen logische Variable oder logische Feldelemente sein. Sie erhalten bei Ausführung einer INQUIRE-Anweisung Werte gemäß folgender Übersicht:

Parameter	Wert	Bedeutung bei Verwendung in	
		INQUIRE/U-Anweisung	INQUIRE/F-Anweisung
EXIST	TRUE	Einheit existiert	Datei existiert
	FALSE	Einheit existiert nicht	Datei existiert nicht
OPENED	TRUE	Verbindung zu Datei existiert.	Verbindung zu Einheit existiert.
	FALSE	keine Verbindung	keine Verbindung
NAMED	TRUE	verbundene Datei hat einen Namen	Dateiname ist gültig
	FALSE	verbundene Datei hat keinen Namen (STATUS = SCRATCH)	Dateiname ist nicht gültig

NUMBER = num
RECL = re
NEXTREC = nr

'num', 're' und 'nr' müssen ganzzahlige Variable oder ganzzahlige Feldelemente sein.
Sie erhalten bei Ausführung einer INQUIRE-Anweisung Werte gemäß folgender Übersicht. Sind die angegebenen Bedingungen nicht erfüllt, sind diese Werte unbestimmt.

Parameter	Wert	Bedingungen
NUMBER	Einheiten-nummer	Die im FILE-Parameter benannte Datei ist mit dieser Einheit verbunden.
RECL	Satzlänge (siehe Abschnitt 6.1.1.)	Verbindung zwischen Einheit und Datei existiert für direkten Zugriff (siehe RECL-Parameter der OPEN-Anweisung in Abschnitt 6.4.1.)
NEXTREC	n+1	Verbindung zwischen Einheitennummer und Datei existiert für direkten Zugriff bisherige fehlerfreie Arbeit mit dieser Verbindung n ist Wert des REC-Parameters der zuletzt ausgeführten READ- oder WRITE-Anweisung
	1	bisher keine READ- oder WRITE-Anweisung ausgeführt, aber Verbindung zwischen Einheit und Datei existiert.

NAME = dsn 'dsn' muß eine Variable oder ein Feldelement vom Typ CHARACTER sein und erhält bei Ausführung der INQUIRE/U-Anweisung den Namen einer Datei, wenn zu dieser Datei eine Verbindung mit der im UNIT-Parameter angegebenen Einheit existiert und wenn diese Datei einen Namen hat. Existiert keine Verbindung oder hat die Datei keinen Namen, ist der Wert unbestimmt.
Wird der NAME-Parameter in einer INQUIRE/F-Anweisung verwendet, erhält 'dsn' einen Wert, der ein evtl. vervollständigter Dateiname ist (siehe [1]).
Erhält 'dsn' einen Wert, kann dieser Wert im FILE-Parameter einer OPEN-Anweisung verwendet werden.

ACCESS = ac
FORM = fm
BLANK = bl
SEQUENTIAL = seq
DIRECT = dir
FORMATTED = fmt
UNFORMATTED = unf

'ac', 'fm', 'bl', 'seq', 'dir', 'fmt' und 'unf' müssen Variable oder Feldelemente vom Typ CHARACTER sein und erhalten die in der folgenden Übersicht angegebenen Werte, wenn die ebenfalls angegebenen Bedingungen erfüllt sind, also eine erwähnte Verbindung zwischen Einheit und Datei existiert und auch die Datei existiert, über die Aussagen gemacht werden sollen.
Sind die Bedingungen nicht erfüllt, sind die Werte unbestimmt.

Parameter	Wert	Bedingungen
ACCESS	SEQUENTIAL	Verbindung für sequentiellen Zugriff
	DIRECT	Verbindung für direkten Zugriff
FORM	FORMATTED	Verbindung für format- oder list-gesteuerte Datenaufbereitung
	UNFORMATTED	Verbindung erlaubt nur Ein- und Ausgabe ohne Datenaufbereitung.
BLANK	NULL	Verbindung wurde mit BLANK=NULL in der OPEN-Anweisung, ohne BLANK-Parameter oder vor Beginn der Ausführung des Programms, erzeugt.
	ZERO	Verbindung wurde mit BLANK=ZERO in der OPEN-Anweisung erzeugt.
SEQUENTIAL	YES	Sequentieller Zugriff zur Datei ist erlaubt.
	NO	Sequentieller Zugriff zur Datei ist nicht erlaubt.
	UNKNOWN	Es ist nicht feststellbar, ob sequentieller Zugriff erlaubt ist.

Parameter	Wert	Bedingungen
DIRECT	YES	Direkter Zugriff zur Datei ist erlaubt.
	NO	Direkter Zugriff zur Datei ist nicht erlaubt.
	UNKNOWN	Es ist nicht feststellbar, ob direkter Zugriff erlaubt ist.
FORMATTED	YES	Bereits existierende Sätze der Datei sind formatisiert.
	NO	Bereits existierende Sätze der Datei sind unformatisiert.
	UNKNOWN	Typ der Sätze der Datei ist nicht feststellbar.
UNFORMATTED	YES	Bereits existierende Sätze der Datei sind unformatisiert.
	NO	Bereits existierende Sätze der Datei sind formatisiert.
	UNKNOWN	Typ der Sätze der Datei ist nicht feststellbar.

7. FORMAT-Anweisung, format- und list-gesteuerte Datenaufbereitung

Weil formatisierte Sätze die Daten in einer Form enthalten, die nicht der rechner-internen Darstellung dieser Daten entspricht (siehe Abschnitt 6.) muß bei der Ein- oder Ausgabe von formatisierten Sätzen eine Datenaufbereitung erfolgen.
Die Art der Datenaufbereitung wird durch die FMT-Parameter in READ-, PRINT- und WRITE-Anweisungen festgelegt (siehe Abschnitt 6.). Besteht der FMT-Parameter aus dem Stern (*) wird die list-gesteuerte Datenaufbereitung gefordert. Andere Angaben im FMT-Parameter verbinden die E/A-Anweisung mit einer Liste von Formatelementen und fordern damit die format-gesteuerte Datenaufbereitung. Durch diese Formatelemente wird die Struktur der Daten in den formatisierten Sätzen beschrieben.
Bei der Eingabe mittels list-gesteuerter Datenaufbereitung muß das Format der Daten in den formatisierten Sätzen gewissen Regeln genügen. Bei der Ausgabe mittels list-gesteuerter Datenaufbereitung werden die Daten in ein zum Datentyp passendes Format umgewandelt.
Bei der Programmierung zu berücksichtigende Unterschiede zwischen Ein- und Ausgabeformat bei der list-gesteuerten Datenaufbereitung bestehen lediglich für Daten vom Typ CHARACTER.

7.1. FORMAT-Anweisung

Eine Format-Anweisung definiert eine Liste von Formatelementen zur Beschreibung des Formates von formatisierten Sätzen.
Die FORMAT-Anweisung hat die Form:

```
marke FORMAT(formatliste)
```

Mittels der Marke der FORMAT-Anweisung kann die Liste von Formatelementen 'formatliste' für die format-gesteuerte Datenaufbereitung in beliebig vielen E/A-Anweisungen verwendet werden. Dazu wird im FMT-Parameter der E/A-Anweisung die Marke oder eine ganzzahlige Variable angegeben. Dieser ganzzahligen Variablen muß mittels einer ASSIGN-Anweisung die Marke zugewiesen worden sein.
Für die Liste der Formatelemente 'formatliste' gelten folgende Regeln:

- Die Liste kann Formatelemente und interne Formatlisten enthalten.
- Es gibt wiederholbare und nicht wiederholbare Formatelemente (siehe Abschnitt 7.3.).
- Wiederholbare Formatelemente und interne Formatlisten können durch einen davorstehenden Wiederholungsfaktor, eine ganzzahlige Konstante ohne Vorzeichen, ergänzt werden.

- Die Angabe eines Wiederholungsfaktors vor einem Formatelement oder einer internen Formatliste ist identisch mit der mehrfachen Angabe des Formatelementes oder der internen Formatliste.
- Nichtwiederholbare Formatelemente können wiederholt werden, wenn sie in internen Formatlisten auftreten, vor denen ein Wiederholungsfaktor steht.
- Eine interne Formatliste ist eine von einem Klammernpaar eingeschlossene Liste von Formatelementen und internen Formatlisten.
- Formatelemente und interne Formatlisten werden gewöhnlich durch Kommata voneinander getrennt.
  Nach einem P-Formatelement braucht jedoch kein Komma zu stehen, wenn unmittelbar ein F-, E-, D- oder G-Formatelement folgt. Ein '/'-Formatelement oder ein ':'-Formatelement braucht ebenfalls nicht durch Kommata von vorhergehenden oder nachfolgenden Formatelementen oder internen Formatlisten getrennt zu werden.
- Außer in den Formatelementen, die Zeichenkettenkonstanten definieren (siehe Abschnitt 7.3.3.1.), dürfen Leerzeichen vor und nach und auch in den Formatelementen in beliebiger Zahl auftreten.

Die Liste der Formatelemente 'formatliste' kann weggelassen werden oder kann nur nichtwiederholbare Formatelemente enthalten. Eine E/A-Anweisung, die eine derartige FORMAT-Anweisung verwendet, darf dann keine Datenlistenelemente enthalten.

7.2. Zeichenketten im FMT-Parameter

Die format-gesteuerte Datenaufbereitung kann durch eine Angabe vom Typ CHARACTER im FMT-Parameter einer E/A-Anweisung gemäß Abschnitt 6.3.1. gefordert werden.
Der Wert des FMT-Parameters wird dabei durch den Wert des Ausdruckes vom Typ CHARACTER oder durch die Verkettung aller Feldelemente des angegebenen Feldes vom Typ CHARACTER bestimmt. Der Ausdruck besteht im einfachsten Fall nur aus einer Variablen oder einem Feldelement vom Typ CHARACTER oder aus einer Zeichenkettenkonstanten.
Der Wert des FMT-Parameters muß aus einer gedachten Verkettung folgender Teile bestehen:
- einer öffnenden Klammer, der beliebig viele Leerzeichen vorangehen dürfen,
- einer Liste von Formatelementen, für die die im Abschnitt 7.1. angegebenen Regeln gelten,
- einer schließenden Klammer, der beliebige Zeichen folgen dürfen.

7.3. Format-gesteuerte Datenaufbereitung

7.3.1. Überblick über die Formatelemente und ihre Interpretation

Ein wiederholbares Formatelement beschreibt eine Zeichenfolge, die bei der Ausgabe eines logischen oder ganzzahligen Wertes oder bei der Ausgabe eines Gleitkomma- oder Zeichenkettenwertes zu erzeugen ist, oder es beschreibt die Zeichenfolge, deren Inhalt, nach einer eventuell notwendigen Datenumwandlung, einem Objekt des Programmes zuzuweisen ist. Ein solches Objekt ist entweder eine Variable oder ein Feldelement.
Eine durch ein wiederholbares Formatelement beschriebene Zeichenfolge definiert einen Eingabe- oder einen Ausgabebereich des formatisierten Satzes. Die durch ein I-, F-, D-, E- oder G-Formatelement beschriebenen Eingabe- oder Ausgabebereiche werden numerische Eingabebereiche bzw. numerische Ausgabebereiche genannt.
Ein Ein- oder Ausgabebereich beginnt dabei an der aktuellen Position im Satz. Die aktuelle Position ist bei Beginn der Verarbeitung eines Satzes das Byte 1. Sie wird durch wiederholbare Formatelemente und durch Zeichenkettenkonstanten in der Formatliste verändert und kann durch T-, TL-, TR- und X-Formatelemete beeinflußt werden.
Auf die Verarbeitung von numerischen Eingabefeldern und auf die Erzeugung von numerischen Ausgabefeldern haben zusätzlich andere nicht wiederholbare Formatelemente Einfluß. Ein BN- oder BZ-Formatelement hat Bedeutung bei der Eingabe, ein S-, SP- oder SS-Formatelement hat Bedeutung bei der Ausgabe und das kP-Formatelement hat Bedeutung bei Ein- und Ausgabe.
Für die Ausführung einer format-gesteuerten E/A-Anweisung müssen den Elementen der Datenliste wiederholbare Formatelemente zugeordnet werden. Dabei ist jedem Element der Datenliste oder bei Angabe eines Feldnamens jedem Feldelement ein Formatelement zuzuordnen. Hat das Datenlistenelement den Typ COMPLEX, erfordert das Datenlistenelement oder ein Feldelement zwei wiederholbare Formatelemente. Die Art des Formatelementes ist abhängig vom Typ des Datenlistenelementes.
In der folgenden Übersicht sind alle wiederholbaren Formatelemente sowie die zu ihnen passenden Datentypen angegeben.

Typ des Wertes oder des Objektes	Verwendbare Formatelemente
LOGICAL	Lw
INTEGER	Iw, Iw.m
REAL, DOUBLE PRECISION und COMPLEX (2 Formatelemente notwendig)	Fw.d, Dw.d, Ew.d, Ew.dEe, Gw.d, Gw.dEe
CHARACTER	A, Aw

Durch die Kennbuchstaben L, I, F, D, E, G und A wird die Art der Datenaufbereitung festgelegt, während die ganzzahligen Konstanten w, m, d und e Längen oder Teillängen der zu verarbeitenden Zeichenfolgen festlegen.
Die Zuordnung der wiederholbaren Formatelemente zu den Datenlistenelementen erfolgt von links nach rechts. Diese Zuordnung erfolgt unter Berücksichtigung von angegebenen Wiederholungsfaktoren. Zwischen zwei wiederholbaren Formatelementen können beliebig viele nichtwiederholbare Formatelemente stehen, auch wenn die wiederholbaren Formatelemente zu einem Datenlistenelement (Typ COMPLEX oder ein Feld) gehören. Wird vor der Zuordnung eines wiederholbaren Formatelementes ein nicht wiederholbares Formatelement erkannt, wird die zu diesem Formatelement gehörende Operation ausgeführt.

Die nicht wiederholbaren Formatelemente ermöglichen:
- die Ausgabe von Zeichenkettenkonstanten ('x...', nHx...)
- die Änderung der aktuellen Position im Satz (Tc, TLn, TRn, nX)
- die Angabe eines Skalierungsfaktors, wirksam bei der Verarbeitung von numerischen Ein- und Ausgabebereichen (kP)
- den Beginn der Verarbeitung des nächsten Satzes, auch wenn der vorhergehende nicht komplett verarbeitet wurde (/)
- die Beendigung der Abarbeitung der Formatliste, wenn kein Datenlistenelement mehr vorhanden ist (:)

- die Definition der Interpretation von Leerzeichen in numerischen Eingabebereichen (BN, BZ)
- das Einfügen eines Pluszeichen in einen numerischen Ausgabebereich bei positivem Wert (S, SP, SS)

Die Buchstaben H, T, TL, TR, X, P, BN, BZ, S, SP, und SS, sowie die Zeichen ' (Apostroph), / und : kennzeichnen die Art des Formatelementes, x repräsentiert ein im Quelltext und auf der Ausgabedatei darstellbares Zeichen, während n, c und k Konstanten sind (n, c - ganzzahlig, ohne Vorzeichen, verschieden von 0; k - ganzzahlig).
Die Funktionen von nicht wiederholbaren Formatelementen werden auch dann noch ausgeführt, wenn keine Zuordnung eines wiederholbaren Formatelementes zu einem Datenlistenelement mehr benötigt wird, d.h. wenn die Datenliste abgearbeitet ist. In diesem Fall wird bei Erkennen eines :-Formatelementes die Ausführung der E/A-Anweisung beendet.
Wird ein :-Formatelement erkannt, und die Datenliste enthält noch unverarbeitete Elemente, wird das :-Formatelement ignoriert.
Enthält die Datenliste noch unverarbeitete oder nicht vollständig verarbeitete Elemente und die Abarbeitung der Formatliste ist an deren rechtem Ende angekommen, wird die Datei so positioniert, als ob ein /-Formatelement abgearbeitet wird, und von der Formatliste wird der Teil nochmals abgearbeitet, der mit der letzten internen Formatliste, einschließlich deren Wiederholungsfaktor, beginnt.
Die letzte interne Formatliste ist dabei die interne Formatliste, die als letzte beeendet wird, deren schließende Klammer also am weitesten rechts in der Formatliste steht. Enthält die Formatliste keine interne Formatliste, beginnt die erneute Abarbeitung der Formatliste an ihrem ersten Formatelement. Die nochmalige Abarbeitung der ganzen Formatliste oder von Teilen derselben, hat keinen Einfluß auf den gültigen Skalierungsfaktor, auf die Darstellung des Pluszeichens in numerischen Ausgabebereichen, sowie auf die Interpretation von Leerzeichen in numerischen Eingabebereichen, solange keine P-, S-, SP-, SS-, BN- oder BZ-Formatelemente eine Änderung bewirken.

7.3.2. Wiederholbare Formatelemente

7.3.2.1. I-Formatelement

Das I-Formatelement kann Elementen der Datenliste des Typs INTEGER zugeordnet werden und hat die Form

Iw [.m],

wobei w und m ganzzahlige Konstanten (w 0) sein müssen.

Für den durch ein I-Formatelement beschriebenen numerischen Ein- oder Ausgabebereich gelten folgende Regeln:

Eingabe:
- w ist die Länge des Eingabebereiches, der eine ganzzahlige Konstante enthalten muß.
- Die ganzzahlige Konstante besteht aus einer Ziffernfolge, der ein Vorzeichen vorangehen kann.
- Die Interpretation von Leerzeichen im Eingabebereich ist vom Wert des BLANK-Parameters einer eventuell für die Datei ausgeführten OPEN-Anweisung und von bereits abgearbeiteten BN- und BZ-Formatelementen abhängig (siehe Abschnitt 7.3.)
- Enthält der Eingabebereich nur Leerzeichen, wird der Wert 0 eingelesen.
- Die Angabe .m ist bedeutungslos.

Ausgabe:
- w ist die Länge des Ausgabebereiches, in dem der Wert in Form einer ganzzahligen Konstanten rechtsbündig mit führenden Leerzeichen eingetragen wird.
- m gibt die Zahl der mindestens auszugebenden Ziffern an, die notfalls durch Ausgabe führender Nullen erreicht wird.
- Die Ausgabe eines Pluszeichens, wenn der auszugebende Wert positiv ist oder den Wert 0 hat, erfolgt nur nach vorheriger Abarbeitung eines SP-Formatelementes (siehe Abschnitt 7.3.3.7.).
- Ist die Länge des Ausgabebereiches für die Aufnahme aller notwendigen Zeichen zu klein, wird er mit dem Zeichen '*' gefüllt.
- Ist m nicht angegegeben, besteht das Ausgabefeld mindestens aus einer Ziffer (Ziffer 0 bei Ausgabe des Wertes 0).
- Ist m=0 und der auszugebende Wert ist ebenfalls 0, besteht das Ausgabefeld nur aus Leerzeichen.

Beispiele:

Wert	Formatelement	Erzeugte Zeichenfolge
523	I5	bb523
523	I7.7	0000523
-411	I4.1	-411
301	SP,I5	b+301

Angegebene Zeichenfolge	Formatelement	Eingelesener Wert
b5b23	I5	523
0b4	I3.3	4
3b1	BZ,I3	301

7.3.2.2. F-, E-, D- und G-Formatelement

Mit den F-, E-, D- und G-Formatelementen können reelle Daten einfacher und doppelter Genauigkeit, sowie komplexe Daten aufbereitet werden. Komplexe Daten verlangen jedoch zwei Formatelemente der hier beschriebenen Art.

Die Formatelemente

```
Fw.d
Ew.d[Ee]
Dw.d
Gw.d[Ee]
```

beschreiben numerische Ein- und Ausgabebereiche, wobei w, d und e ganzzahlige Konstanten ohne Vorzeichen (w, e verschieden von 0) sein müssen.

Die Zeichenfolge im Eingabebereich muß folgenden Regeln genügen:
- w ist die Länge des Eingabebereiches, der eine reelle Konstante enthalten muß.
- Die Interpretation von Leerzeichen im Eingabebereich ist vom Wert des BLANK-Parameters einer eventuell für die Datei ausgeführten OPEN-Anweisung und von bereits abgearbeiteten BN- und BZ-Formatelementen abhängig (siehe Abschnitt 7.3.).
- Besteht das Eingabefeld nur aus Leerzeichen, wird der Wert 0 eingelesen.
- Eine reelle Konstante besteht aus einer Ziffernfolge, der ein Vorzeichen vorangehen und die einen Dezimalpunkt enthalten kann.
- Enthält die Ziffernfolge keinen Dezimalpunkt, werden die rechts stehenden d Ziffern als gebrochener Teil der Dezimalzahl betrachtet, wobei, falls notwendig, führende Nullen ergänzt werden.
- Enthält die Ziffernfolge einen Dezimalpunkt, wird die Angabe von d ignoriert.
- Die Angabe Ee hat keine Bedeutung.
- Der beschriebenen reellen Konstanten kann im Eingabebereich ein dezimaler Exponent folgen. Dieser Exponent muß entweder eine ganzzahlige Konstante mit einem Vorzeichen sein oder muß einer der Buchstaben E und D gefolgt von einer ganzzahligen Konstanten sein.
- Enthält der Eingabebereich keinen Exponenten, wird bei der Bestimmung der internen Darstellung des einzulesenden Wertes der gültige Skalierungsfaktor berücksichtigt. Ist dieser ungleich 0 (d.h. zuletzt wurde ein P-Formatelement kP (mit k verschieden von 0) abgearbeitet, siehe auch Abschnitt 7.3.3.3.), wird der Variablen oder dem Feldelement der Quotient aus dem im Eingabebereich dargestellten Wert und 10**k zugewiesen.

- Ein gültiger Skalierungsfaktor ist ohne Bedeutung, wenn der Eingabebereich einen Exponenten enthält.

Ein Ausgabebereich für eines der Formatelemente F, D und E hat das in der folgenden Übersicht angegebene Format, wenn der gültige Skalierungsfaktor 0 ist, das heißt noch kein P-Formatelement oder zuletzt ein Formatelement OP (siehe Abschnitt 7.3.3.3.) abgearbeitet wurde.

Fw.d	[b]... [+ / -] s... DP z... d Ziffern w Zeichen
Ew.d Dw.d	[b]... [+ / -] [0] DP z... {+ / -} zzz d Ziffern w Zeichen
Ew.dEe	[b]... [+ / -] [0] DP z... E {+ / -} z... d Ziffern e Ziffern w Zeichen

mit:
b - Leerzeichen, DP - Dezimalpunkt, s, z - Dezimalziffer.
Ziffernfolgen 's...' haben keine führenden Nullen, während die
Ziffernfolgen 'z...' führende Nullen haben dürfen.

Die Datenaufbereitung durch das G-Formatelement erfolgt abhängig vom absoluten Betrag N des auszugebenden Wertes wie bei der Datenaufbereitung durch ein E- oder F-Formatelement.
Die Folge von Formatelementen, die anstelle eines G-Formatelementes verwendet wird, ist in der folgenden Übersicht dargestellt.

	Gw.d	Gw.dEe
N < 0.1	Ew.d	Ew.dEe
10**(n-1)<=N<10**n	Fs.t,4('b')	Fu.t,m('b')
N<=10**d	Ew.d	Ew.dEe

mit:
n, s, u, t, m - ganzzahlig
n=0,1,2,...,d
s=w-4
u=w-e-2
t=d-n
m=e+2

Für die erzeugten Zeichenfolgen gelten folgende Bedingungen:
- Ein Pluszeichen wird nur dann ausgegeben, wenn es zur Trennung von Mantisse und Exponent notwendig ist oder wenn zuletzt ein SP-Formatelement abgearbeitet wurde (siehe Abschnitt 7.3.3.7.).
- Die erste Ziffer einer Mantisse ist nur dann die Ziffer 0, wenn der Betrag der Mantisse kleiner als 1 ist und diese 0 steht nur dann vor dem Dezimalpunkt, wenn es die Größe des Ausgabebereiches zuläßt.
- Die Mantisse wird gemäß den allgemeinen Rundungsregeln aus dem internen Wert gebildet.

Ein gültiger Skalierungsfaktor k mit k verschieden von 0 (d.h. es wurde ein P-Formatelement kP abgearbeitet, siehe Abschnitt 7.3.3.3.) hat folgende Auswirkungen:

- Bei Verwendung des F-Formatelementes repräsentiert die ausgegebene Zeichenfolge einen Wert, der das Produkt aus dem auszugebenden Wert und 10**k ist.
- Bei der Verwendung des E- oder des D-Formatelementes wird die Normalisierung des auszugebenden Wertes beeinflußt, indem die ausgegebene Mantisse mit 10**k multipliziert und vom ausgegebenen Exponenten k subtrahiert wird.
  Gilt $-d < k = 0$
  enthält der Ausgabebereich hinter dem Dezimalpunkt !k! führende Nullen und d-!k! gültige Ziffern.
  Gilt aber $0 < k < d+2$,
  enthält der Ausgabebereich k gültige Ziffern vor und d-k+1 gültige Ziffern nach dem Dezimalpunkt.
  Andere Werte für den Skalierungsfaktor sind nicht erlaubt.
- Resultiert die Verwendung des G-Formatelementes in einer Datenaufbereitung mittels eines F-Formatelementes, hat ein gültiger Skalierungsfaktor keine Bedeutung. Wird dagegen die Datenaufbereitung mittels E-Formatelement verwendet, hat ein gültiger Skalierungsfaktor den gleichen Effekt, als ob das E-Formatelement direkt angegeben worden wäre.

Werden zur Darstellung des auszugebenden Wertes mehr Zeichen benötigt als der Ausgabebereich oder der Teilbereich für den Exponenten aufnehmen kann, wird der gesamte Ausgabebereich mit dem Zeichen '*' gefüllt (siehe auch Abschnitt 7.3.3.7.).

Beispiele:

Angegebene Zeichenfolge	Formatelement	Eingelesener Wert
00523	F5.4	0,0523
00523	F5.0	523
0052.3	F6.4	52,3
00523	E5.2	5,23
523+1	E5.2E2	52,3
523E2	D5.0	52300
00523	-3PD5.0	523000

Auszugebender Wert	Formatelement	Erzeugte Zeichenfolge
12345	F6.0	b12345
12,345	F8.4	b12.3450
12,345	E8.3	.123+002
12,345	E9.4	.1234+002
12,345	E9.4E1	b0.1234E2
0,012345	E9.4E1	0.1234E-1
12,345	SP,E9.4E1	0.1234E+2
-12,345	G9.2	-12.3bbbb
-12,345	G9.1	b-0.1+002

7.3.2.3. L-Formatelement

Das Formatelement

Lw (w - ganzzahlig und größer 0)

dient zur Datenaufbereitung von logischen Werten.

Eingabe:

- w gibt die Länge des Eingabebereiches an, das eine Zeichenfolge enthält, die als logische Konstante interpretiert werden kann.

- Eine Zeichenfolge kann als logische Konstante gedeutet werden, wenn sie nach Leerzeichen und einem Dezimalpunkt einen der Buchstaben T oder F enthält.
- Führende Leerzeichen und der Dezimalpunkt können entfallen.
- Die den Zeichen T oder F folgenden Zeichen sind ohne Bedeutung.
- Durch den Buchstaben T erhält die einzulesende logische Variable oder das einzulesende logische Feldelement den Wert 'TRUE', anderenfalls den Wert 'FALSE'.

Ausgabe:

- Das Ausgabefeld der Länge w enthält rechtsbündig den Buchstaben T, falls der auszugebende Wert 'TRUE' ist, anderenfalls den Buchstaben F.
- Die restlichen Zeichen des Ausgabefeldes enthalten Leerzeichen.

7.3.2.4. A-Formatelement

Das Formatelement

A[w] (w - ganzzahlig und größer 0)

dient zur formatisierten Ein- oder Ausgabe von Daten des Typs CHARACTER.
Die Länge des Ein- oder Ausgabebereiches ist w oder entspricht der Länge des Datenlistenelementes, wenn w nicht angegeben worden ist.
Die Datenaufbereitung zwischen Ein- oder Ausgabebereich und Datenlistenelement erfolgt so, als ob die folgenden Ergibtanweisungen ausgeführt werden.
Hierbei seien:

VAR - das Datenlistenelement der Länge n,
EAB - der Ein- oder Ausgabebereich der Länge w und
LEER- eine CHARACTER-Variable ausreichender Länge mit Leerzeichen als Inhalt.

Eingabe:

```
n > w :   VAR = EAB // LEER (1 : n-w)
n = w :   VAR = EAB
n < w :   VAR = EAB (w-n+1:w)
```

Ausgabe:

```
w > n :   EAB = LEER (1:w-n) // VAR
w = n :   EAB = VAR
w < n :   EAB = VAR (1 : w)
```

7.3.3. Nicht wiederholbare Formatelemente

7.3.3.1. Zeichenkettenkonstante

Eine Zeichenkettenkonstante in der Formatliste wird unverändert an der aktuellen Position in den formatisierten Satz übertragen, wenn die Formatliste von einer PRINT- oder WRITE-Anweisung verwendet wird. Von einer READ-Anweisung darf ein derartiges Formatelement nicht verwendet werden.
Die Zeichenkettenkonstante kann eine in ein Apostroph-Paar eingeschlossene Zeichenfolge oder eine Hollerith-Konstante sein.
In einer durch ein Apostroph-Paar eingeschlossenen Zeichenkettenkonstanten sind alle auf dem Rechner darstellbaren Zeichen gültig, also auch das Leerzeichen. Ein auszugebendes Apostroph ist jedoch doppelt anzugeben.
Eine Hollerith-Konstante hat folgenden Aufbau:

nH x1, x2, ..., xn

Die ganzzahlige Konstante n gibt die Anzahl der dem Buchstaben H unmittelbar folgenden und auf dem Rechner darstellbaren Zeichen x1, x2, ..., xn an. In dieser Zeichenfolge vorkommende Leerzeichen gehören dabei zur Hollerith-Konstanten. Diese Zeichenfolge bildet den Wert der Hollerith-Konstanten.

7.3.3.2. Änderung der aktuellen Position im Satz

Die aktuelle Position im Satz, d.h. der mögliche Beginn eines Ein- oder Ausgabebereiches, kann durch die Positionierungsformatelemente

Tc
TLn
TRn
nX

geändert werden (n, c - ganzzahlige Konstanten, 0).

Durch ein Formatelement Tc wird für den Beginn des nächsten Ein- oder Ausgabebereiches das Byte c des gerade verarbeiteten Satzes festgelegt. Durch ein Formatelement TLn erfolgt eine Verschiebung der aktuellen Position um n Bytes nach links und durch eines der Formatelemente TRn oder nX eine Verschiebung um n Bytes nach rechts.
Die Positionierungsformatelemente haben nur dann Einfluß auf die Länge des zu erzeugenden Satzes, wenn danach eine Datenaufbereitung durch wiederholbare Formatelemente erfolgt oder wenn danach

in der Formatliste eine Zeichenkettenkonstante auftritt. Ebensowenig hat die Länge eines eingelesenen Satzes Einfluß auf die möglichen Werte in den Positionierungsformatelementen.
Das bedeutet, daß nach Positionierungsformatelementen die aktuelle Position außerhalb des Satzes liegen darf. An dieser Position darf dann allerdings kein Ein- oder Ausgabebereich beginnen. Bei der Ausgabe wird die Länge des Satzes, wenn sie nicht konstant ist, durch den Ausgabebereich festgelegt, dessen Ende die höchste Position im Satz hat. Positionierungsformatelemente haben keinen Einfluß auf den Inhalt eines Ausgabesatzes. Werden Satzpositionen jedoch durch keinen Ausgabebereich erfaßt, besteht ihr Inhalt aus Leerzeichen.

7.3.3.3. Skalierungsfaktor

Ein Skalierungsfaktor wird durch das Formatelement

kP

(k - ganzzahlige Konstante) definiert.

Ein gültiger Skalierungsfaktor hat bei der Datenaufbereitung durch F-, E-, D- und G-Formatelemente Einfluß auf den Wert einer einzulesenden Ziffernfolge, wenn der Eingabebereich keinen Exponenten enthält. In auszugebenden Werten kann durch einen Skalierungsfaktor in der Darstellung der Mantisse die Anzahl der gültigen Ziffern vor dem Dezimalpunkt oder die Anzahl der führenden Nullen nach dem Dezimalpunkt beeinflußt werden.
Die Wirkungsweise eines gültigen Skalierungsfaktors für die genannten Formatelemente ist bei der Erläuterung dieser Formatelemente im Abschnitt 7.3.2.2. beschrieben.

7.3.3.4. Verarbeitung eines neuen Satzes

Bei Erreichen eines Formatelementes / wird die Datenaufbereitung für den aktuellen Satz beendet.

Für die Eingabe bedeutet das:
- Der noch nicht verarbeitete Teil des Satzes wird übergangen.
- Dabei kann auch ein ganzer Satz übergangen werden.
- Es wird auf den Anfang des nächsten Satzes positioniert. Dieser Satz wird der aktuelle Satz.
- Wenn die Datei für direkten Zugriff verbunden ist, wird die aktuelle Satznummer um 1 erhöht.

Für die Ausgabe gilt:
- Der durch die evtl. vorangehenden Datenaufbereitungen gebildete Satz wird ausgegeben, d.h. er wird erzeugt.
- Dieser Satz wird der letzte und aktuelle Satz der Datei. Für eine interne Datei oder eine Datei mit Direktzugriff wird dieser Satz bei Bedarf mit Leerzeichen aufgefüllt.
- Bei sequentieller Ausgabe kann auch ein leerer Satz (ein Satz der keine Zeichen enthält) ausgegeben werden.
- Wenn die Datei für direkten Zugriff verbunden ist, wird die aktuelle Satznummer um 1 erhöht.

7.3.3.5. Beendigung der Abarbeitung einer Formatliste

Bei Erreichen des Formatelementes : wird die Abarbeitung der Formatliste beendet, wenn sämtliche Elemente der Datenliste bereits Formatlistenelementen zugeordnet wurden. Ein Formatelement ":" ist bedeutungslos, wenn weitere Datenlistenelemente existieren.

7.3.3.6. Leerzeichen in numerischen Eingabebereichen

Die Interpretation von Leerzeichen in numerischen Eingabebereichen wird durch den BLANK-Parameter der OPEN-Anweisung und durch die Formatelemente BN und BZ beeinflußt.
Nach einem Formatelement BN haben Leerzeichen keinen Einfluß auf den dargestellten Wert. Nach einem Formatelement BZ wird ein Leerzeichen im Eingabebereich als Ziffer 0 interpretiert.
Wurde in der OPEN-Anweisung für die Datei der BLANK-Parameter angegeben, so wirkt dieser so, als ob als erstes Formatelement jeder READ-Anweisung für diese Datei das BN-Formatelement bei BLANK=NULL und das BZ-Formatelement bei BLANK=ZERO angegeben worden wäre.
Wurde in dieser OPEN-Anweisung der BLANK-Parameter nicht angegeben oder wurde keine OPEN-Anweisung ausgeführt, werden Leerzeichen in numerischen Eingabefeldern ignoriert.
Der eingelesene Wert ist jedoch immer 0, wenn der Eingabebereich nur aus Leerzeichen besteht.

7.3.3.7. Pluszeichen in numerischen Ausgabebereichen

Die Angabe eines Pluszeichens bei positivem Wert kann durch eines der Formatelemente S, SP oder SS beeinflußt werden.
Bei Beginn der Ausführung einer WRITE- oder PRINT-Anweisung und nach Auftreten eines S- oder SS-Formatelementes werden keine Pluszeichen bei der Datenaufbereitung durch I-, F-, D-, E- und G-Formatelemente erzeugt, falls das Pluszeichen nicht zur Trennung von Mantisse und Exponent dient.

Nach Auftreten eines SP-Formatelementes gehört das Pluszeichen zu den unbedingt zur Darstellung des auszugebenden Wertes notwendigen Zeichen.
Bei der Ausführung einer READ-Anweisung haben die Formatelemente S, SP und SS keine Bedeutung.

7.4. List-gesteuerte Datenaufbereitung

Durch Angabe des Zeichens '*' im FMT-Parameter von READ-, WRITE und PRINT-Anweisungen kann die list-gesteuerte Datenaufbereitung gefordert werden. Jede Anweisung beginnt dabei mit der Verarbeitung eines neuen Satzes.
Durch die list-gesteuerte Datenaufbereitung werden formatisierte Sätze erzeugt, die die auszugebenden Werte in einer Form enthalten, wie sie durch die list-gesteuerte Datenaufbereitung auch wieder eingelesen werden können. Eine Ausnahme bildet dabei die Aus- und Eingabe von Zeichenkettenwerten.
Durch Eingabe mit list-gesteuerter Datenaufbereitung können jedoch Daten eingelesen werden, deren Form mannigfaltiger ist als sie durch Ausgabe mit der list-gesteuerten Datenaufbereitung erzeugt werden können.
Das Format, das eine Zeichenfolge in einem formatisierten Satz haben muß, damit aus dieser Zeichenfolge ein Wert für das zu definierende Objekt der Datenliste der READ-Anweisung abgeleitet werden kann, ist bei der list-gesteuerten Datenaufbereitung von dem Datentyp des Objektes abhängig. Auch die bei der Ausgabe erzeugte Zeichenfolge ist nur vom Datentyp abhängig.

7.4.1. Formate für die list-gesteuerte Eingabe

Formatisierte Sätze, die mittels list-gesteuerter Datenaufbereitung eingelesen werden sollen, müssen Folgen von Werten enthalten, die durch Trennzeichen voneinander getrennt sind.
Trennzeichen sind die Zeichen ',' und '/' sowie das Leerzeichen.
Als Trennzeichen zwischen zwei Werten gilt auch:
- ein Komma, dem sowohl beliebig viele Leerzeichen vorausgehen als auch folgen können.
- ein Schrägstrich, dem sowohl beliebig viele Leerzeichen vorausgehen als auch folgen können. Hierbei wird nach Eingabe des vorangegangenen Wertes die Ausführung der READ-Anweisung mit list-gesteuerter Datenaufbereitung beendet. Alle weiteren Elemente der Datenliste bleiben also unverändert.
- eine beliebig lange Folge von Leerzeichen.
- ein Satzende, wenn es nicht nach einem anderen Trennzeichen und nicht in einer Zeichenkettenkonstanten auftritt. Das Satzende wird nämlich nur als einfaches Leerzeichen betrachtet. Endet der Satz also mit einer einem Trennzeichen äquivalenten Zeichenfol-

ge, wirkt das Satzende nicht als zusätzliches Trennzeichen. Alle dem letzten eingelesenen Wert folgenden Leerzeichen gelten ebenfalls als Trennzeichen.
Gehen dem ersten Wert im ersten eingelesenen Satz nur Leerzeichen voran, haben diese Leerzeichen keine Bedeutung, bilden also kein Trennzeichen.
Ein einzulesender Wert ist entweder in Form einer Konstanten oder als Nullwert anzugeben. Der Wert der Konstanten wird dabei dem Datenlistenelement zugewiesen. Wird aber ein Nullwert eingelesen, bleibt der Wert des Datenlistenelementes unverändert oder er bleibt undefiniert. Wenn das Datenlistenelement ein Feld ist, müssen Konstanten oder Nullwerte für jedes Feldelement bereitgestellt werden. Soll der gleiche Wert mehreren Datenlistenelementen oder Feldelementen zugewiesen werden, oder sollen mehrere Nullwerte bereitgestellt werden, kann die Konstante oder der Nullwert mit einem Wiederholungsfaktor im Eingabesatz angegeben werden.
Die Zeichenfolge zwischen zwei Trennzeichen kann also folgende Form haben:

```
c
r*c
n
r*
```

Hierbei ist c die einzulesende und datentypabhängige Konstante und r der Wiederholungsfaktor in Form einer ganzzahligen Konstanten (ohne Vorzeichen). n repräsentiert einen Nullwert. Die Form r* ist äquivalent zur Angabe von r Nullwerten. Ein einzelner Nullwert ist dadurch gekennzeichnet, daß zwischen zwei Trennzeichen oder vor dem ersten Trennzeichen des ersten Satzes kein Zeichen auftritt. Als Trennzeichen kommen deshalb nur Komma und Schrägstrich in Frage.
Die Form der einzulesenden Konstanten ist datentypabhängig und wird in der folgenden Übersicht beschrieben. Dazu werden nach Möglichkeit die Formatelemente (siehe Abschnitt 7.3.2.) benutzt, die verwendet werden müßten, wenn die Datenaufbereitung formatgesteuert erfolgen würde.

Datentyp	Äquivalente Formatelemente
CHARACTER*w	Der Wert im Eingabesatz muß die Form einer von einem Apostrophpaar eingeschlossenen Zeichenkettenkonstanten haben. Diese Zeichenkettenkonstante kann in einem Folgesatz fortgesetzt werden. Das Satzende darf jedoch nicht zwischen zwei aufeinanderfolgenden Apostrophen auftreten. Zwei aufeinander folgende Apostrophe sind zur Darstellung eines Apostrophes notwendig. Die Zuweisung zum Datenlistenelement erfolgt wie in einer Ergibtanweisung.
INTEGER*w	Iw
LOGICAL*w	Lw
REAL	Fw.0
DOUBLE PRECISION	Fw.0
COMPLEX	Der Wert im Eingabesatz muß die Form einer Konstanten vom Typ COMPLEX haben, also ein von einem runden Klammernpaar eingeschlossenes und von einem Komma getrenntes Paar von Konstanten, die durch ein F-Formatelement verarbeitbar sind. Diese Konstanten können von beliebig vielen Leerzeichen umgeben sein. Ein Satzende kann zwischen diesen beiden Konstanten auftreten.

w - Länge der Zeichenfolge zwischen den Trennzeichen.

7.4.2. Formate für die list-gesteuerte Ausgabe

Bei der Ausgabe von formatisierten Sätzen mittels der list-gesteuerten Datenaufbereitung werden Folgen von Konstanten ausgegeben. Dabei beginnt jeder Satz mit einem Leerzeichen als Steuerzeichen. Nur Konstanten vom Typ COMPLEX oder CHARACTER werden in Folgesätzen fortgesetzt, aber nur dann, wenn die realisierbare Satzlänge der Datei zu klein ist, um die Konstante in einem Satz auszugeben.

Die Form der auszugebenden Konstanten ist datentypabhängig und wird in der folgenden Übersicht beschrieben. Dazu werden die Formatelemente (siehe Abschnitt 7.3.2.) benutzt, bei deren Verwendung das gleiche Format erzeugt würde.

Datentyp	Äquivalente Formatelemente
CHARACTER*n	A
LOGICAL*n	L4
INTEGER*n	I15
REAL	1P,SP, E20.8E4
DOUBLE PRECISION	1P, SP, E20.8E4
COMPLEX	1P, SP, 'b(', E18.8E4, ',', E18.8E4,')'

Es ist zu beachten, daß mittels list-gesteuerter Datenaufbereitung ausgegebene Werte des Typs CHARACTER nicht unmittelbar mittels list-gesteuerter Datenaufbereitung wieder eingelesen werden können, da kein begrenzendes Apostrophpaar ausgegeben wird und auch ein Apostroph im Wert bei der Ausgabe nicht verdoppelt wird.

Beispiel:

```
CHARACTER ZK*5
DIMENSION ZK(1:5)
LOGICAL   LOG
PRINT     *, LOG,  INTV,  ZK, XR
```

Statt der PRINT-Anweisung ist folgendes Anweisungspaar möglich:

```
      PRINT     100, LOG, INTV, ZK, XR
  100 FORMAT    (L4, I15, 5A, 1P, SP, E20.8E4)
```

Anlage 1: Syntaxregeln

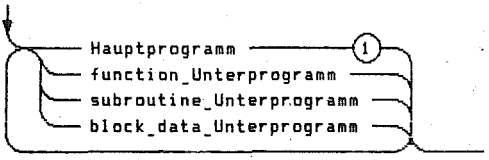
In dieser Anlage wird die Syntax von FORTRAN77 in einer graphisch orientierten Notation dargestellt.
Diese Notation dient in erster Linie einer guten Lesbarkeit, bildet aber keine Basis für eine syntaktische Analyse von FORTRAN77.
Diese Notation gibt keine Auskunft über

- Benutzung von Leerzeichen,
- die Notation von Anweisungen in Anfangs- und Fortsetzungszeilen,
- Kommentarzeilen und
- kontextabhängige Spracheigenschaften.

In jedem Falle ist die in den Abschnitten 1. bis 7. enthaltene Sprachbeschreibung bindend.
In der graphisch orientierten Notation sind folgende Konventionen zu berücksichtigen:

- Folgen von Kleinbuchstaben mit eventuell eingefügten Unterstreichungszeichen stellen syntaktische Einheiten dar.
- Folgen von Großbuchstaben und Sonderzeichen sind als Terminale der Sprache zu deuten.
- Die Syntaxdiagramme sind in Analogie zu Gleisplänen zu deuten und sind somit selbsterklärend. Eine Zahl n in einem Halbkreis gibt an, daß der entsprechende Zweig maximal n-mal durchlaufen werden kann. Eine Zahl n in einem Vollkreis gibt an, daß der entsprechende Zweig exakt n-mal durchlaufen werden muß.
- Für die syntaktischen Einheiten "Vorzeichen", "Ziffer", "Buchstabe" und "Hochkomma" gelten die in Tabelle 1 getroffenen Festlegungen.

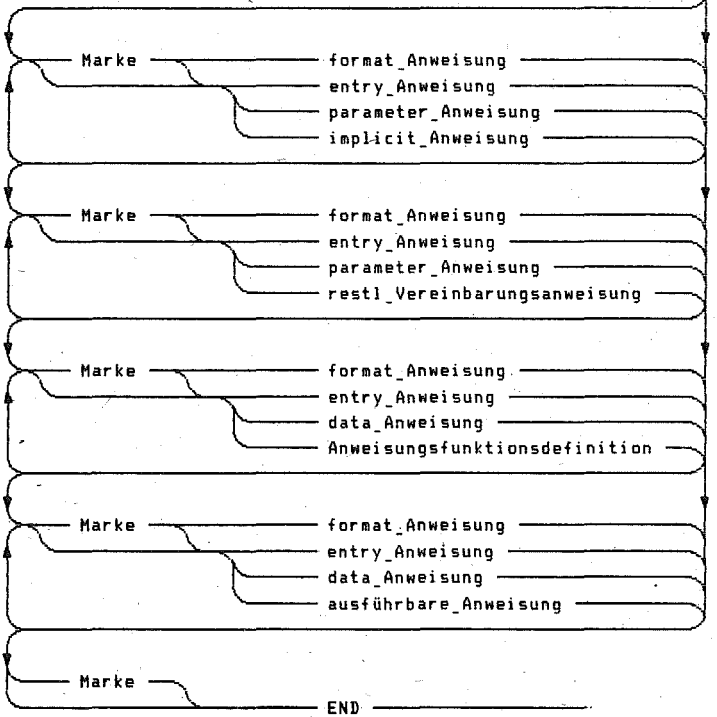
1 Ausführbares_Programm:



(1) Ein ausführbares Programm muß genau ein Hauptprogramm enthalten. Ein ausführbares Programm kann externe Prozeduren enthalten, die nicht in FORTRAN geschrieben sind.

2 Hauptprogramm: program_Anweisung

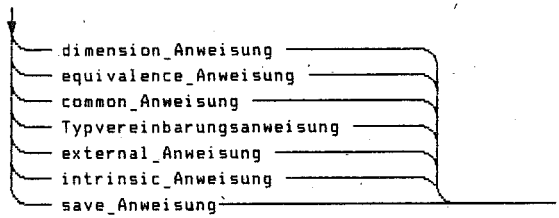
3 function_Unterprogramm: function_Anweisung
4 subroutine_Unterprogramm: subroutine_Anweisung
5 block_data_Unterprogramm: block_data_Anweisung



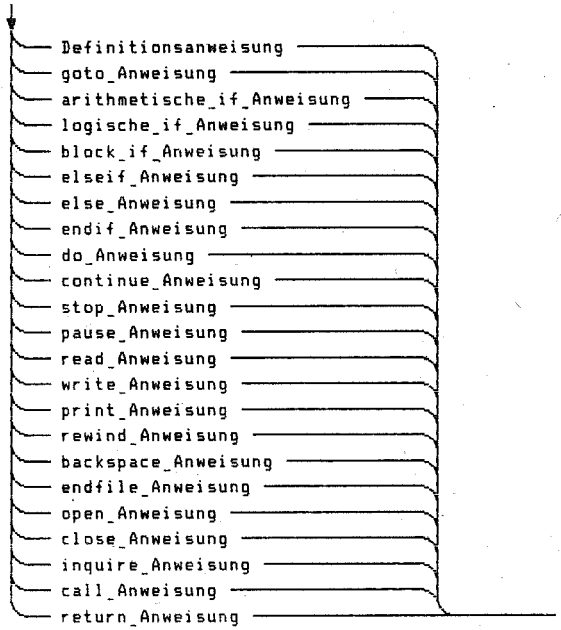
(2) Ein Hauptprogramm darf keine ENTRY- oder RETURN-Anweisung enthalten.

(5) Ein BLOCK DATA-Unterprogramm darf nur BLOCK DATA-, IMPLICIT-, PARAMETER-, DIMENSION-, COMMON-, SAVE-, EQUIVALENCE-, DATA-, END- und Typvereinbarungsanweisungen enthalten.

6 restliche_Vereinbarungsanweisungen:



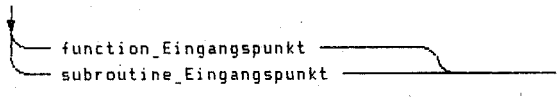
7 Ausführbare_Anweisung:



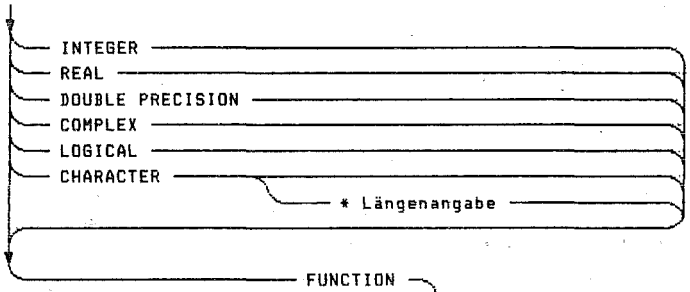
(7) Die END-Anweisung ist auch eine ausführbare Anweisung und muß die letzte Anweisung einer Programmeinheit sein.

8 program_Anweisung: ——— PROGRAM program_Name ———

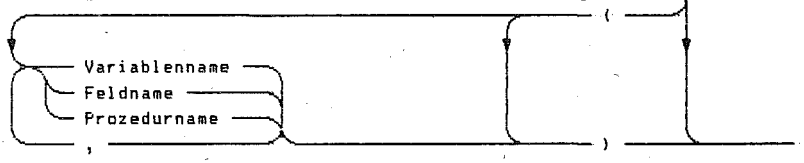
9 entry_Anweisung:



10 function_Anweisung:



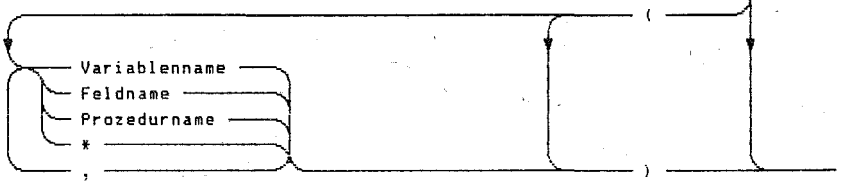
11 function_Eingangspunkt: — ENTRY — function_Name



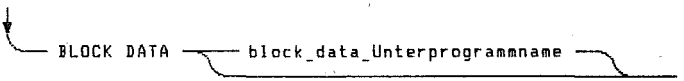
(11) In einer FUNCTION-Anweisung müssen Klammern auftreten.

12 subroutine_Anweisung: — SUBROUTINE

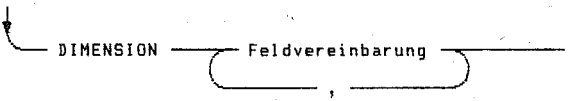
13 subroutine_Eingangspunkt: — ENTRY — subroutine_Name



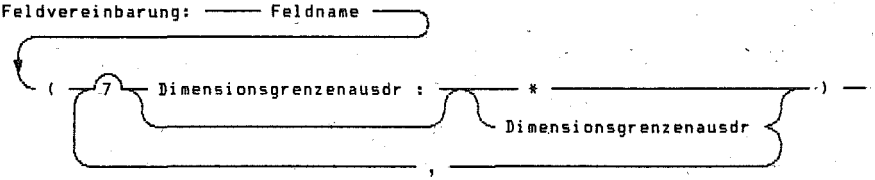
14 block_data_Anweisung:



15 dimension_Anweisung:

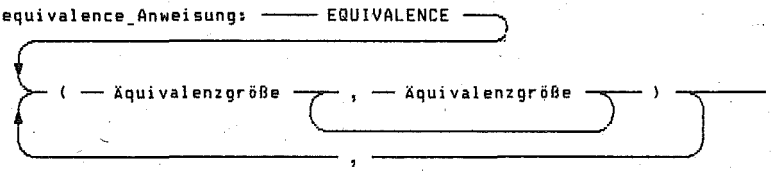


16 Feldvereinbarung:

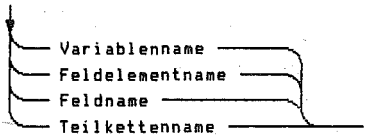


(16) Der Stern kann nur in einer Feldvereinbarung für einen formalen Parameter enthalten sein.

17 equivalence_Anweisung:

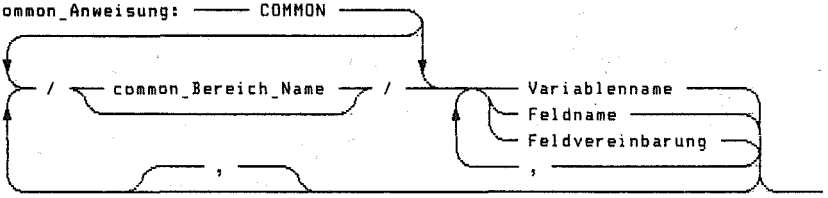


18 Äquivalenzgröße:

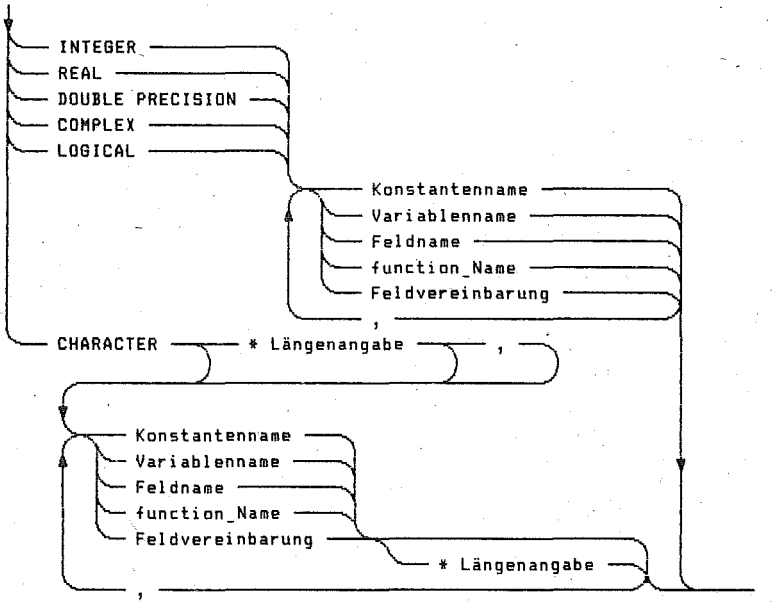


(18) Der Index- oder Teilkettenausdruck in einer EQUIVALENCE-Anweisung muß ein ganzzahliger Konstantenausdruck sein.

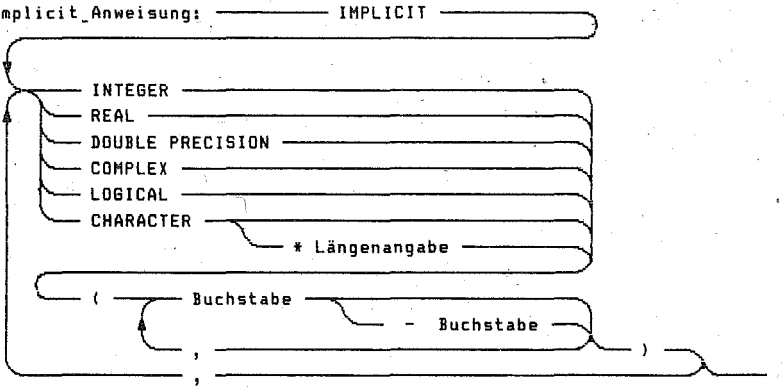
19 common_Anweisung:



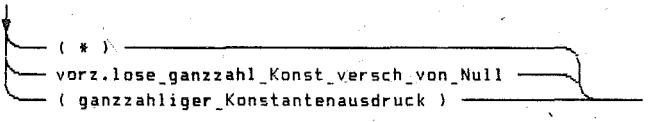
20 Typvereinbarungs_Anweisung:



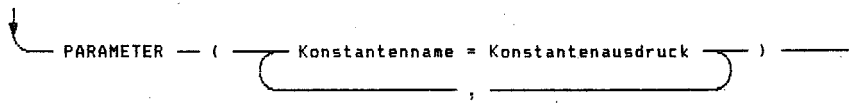
21 implicit_Anweisung:



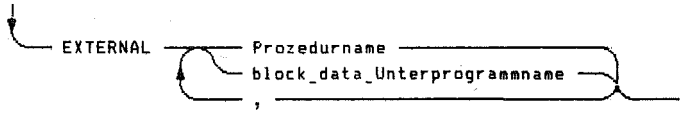
22 Längenangabe:



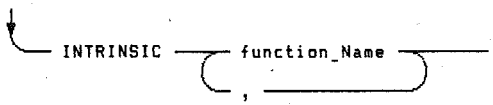
23 parameter_Anweisung:



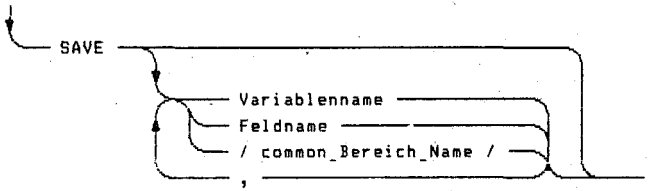
24 external_Anweisung:



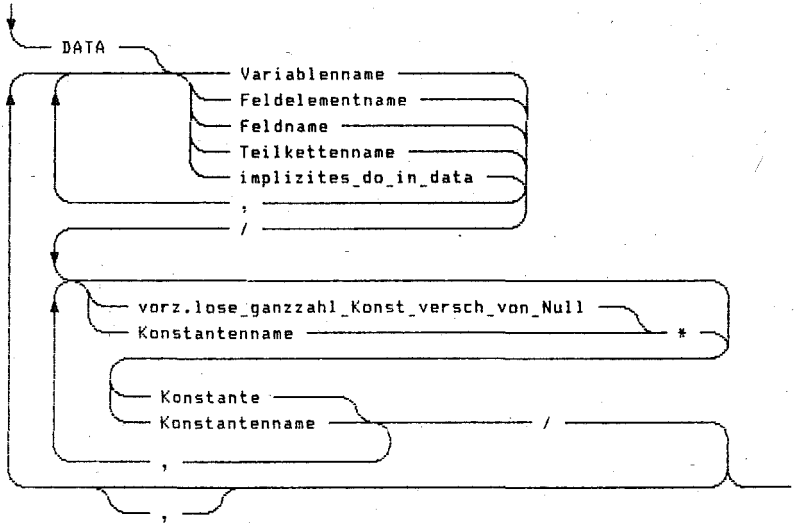
25 intrinsic_Anweisung:



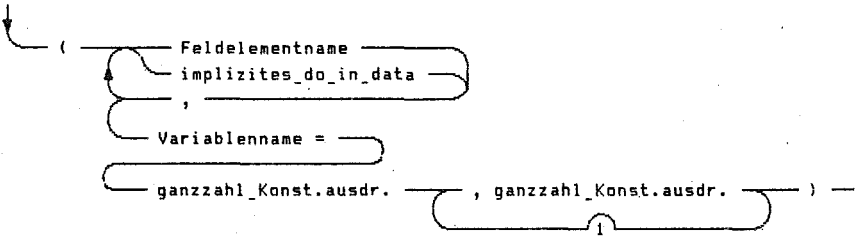
26 save_Anweisung:



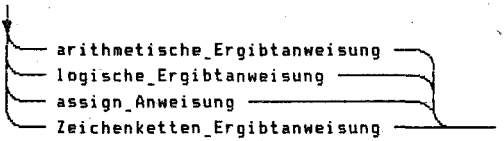
27 data_Anweisung:



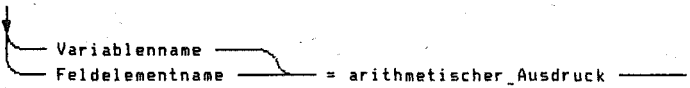
28 implizites_do_in_data:



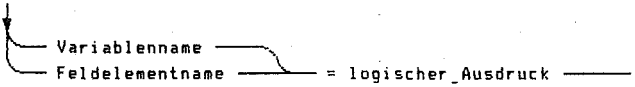
29 Definitionsanweisung:



29.1 arithmetische_Ergibtanweisung:

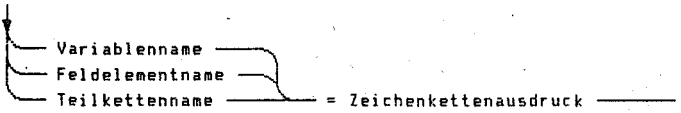


29.2 logische_Ergibtanweisung:

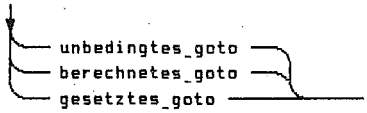


29.3 assign_Anweisung:

29.4 Zeichenketten_Ergibtanweisung:

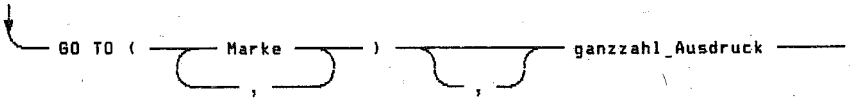


30 goto_Anweisung:

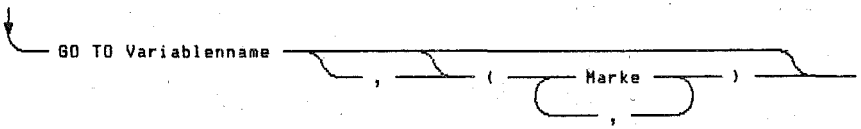


31 unbedingtes_goto: GO TO Marke

32 berechnetes_goto:



33 gesetztes_goto:



34 arithmetische_if_Anweisung:

IF (gemischter_arithmetischer_Ausdruck) Marke , Marke , Marke

35 logische_if_Anweisung:

IF (logischer_Ausdruck) ausführbare_Anweisung

(35) Die in einer logischen IF-Anweisung enthaltene ausführbare Anweisung darf keine DO-, Block-IF-, ELSEIF-, ELSE-, ENDIF-, END- oder andere logische IF-Anweisung sein.

36 block_if_Anweisung:

IF (logischer_Ausdruck) THEN

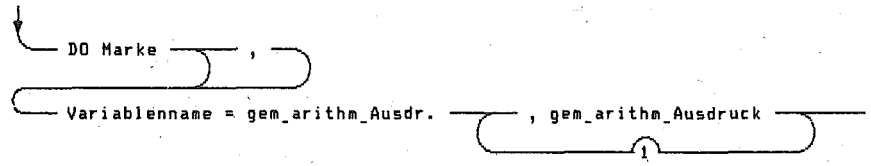
37 elseif_Anweisung:

ELSE IF (logischer_Ausdruck) THEN

38 else_Anweisung: ELSE

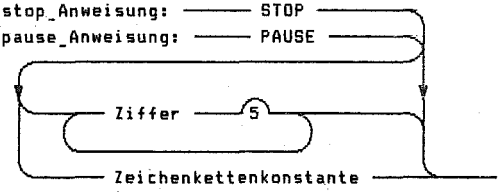
39 endif_Anweisung: END IF

40 do_Anweisung:

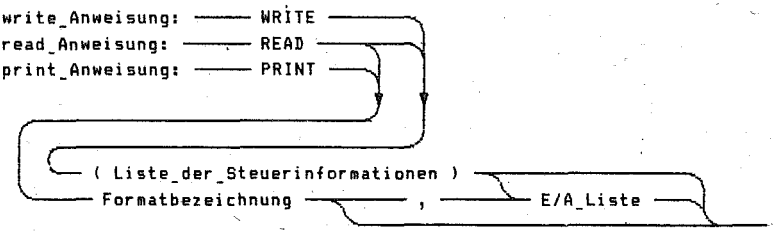


41 continue_Anweisung: CONTINUE

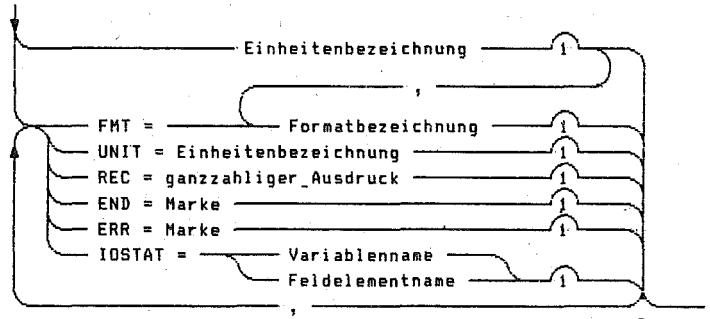
42 stop_Anweisung: STOP
43 pause_Anweisung: PAUSE



44 write_Anweisung: WRITE
45 read_Anweisung: READ
46 print_Anweisung: PRINT

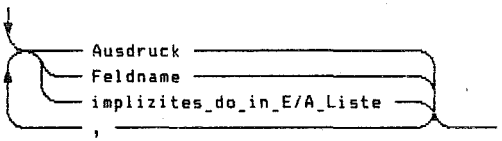


47 Liste_der_Steuerinformationen:



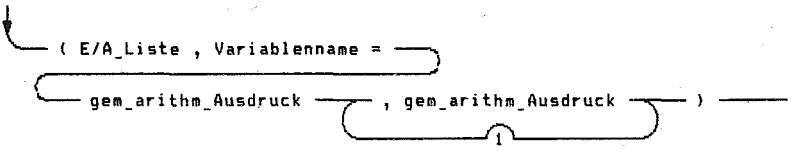
(47) Die Liste der Steuerinformationen muß genau eine Einheitenbezeichnung enthalten. Die Angabe END= darf nicht in einer WRITE-Anweisung auftreten.

48 E/A_Liste:

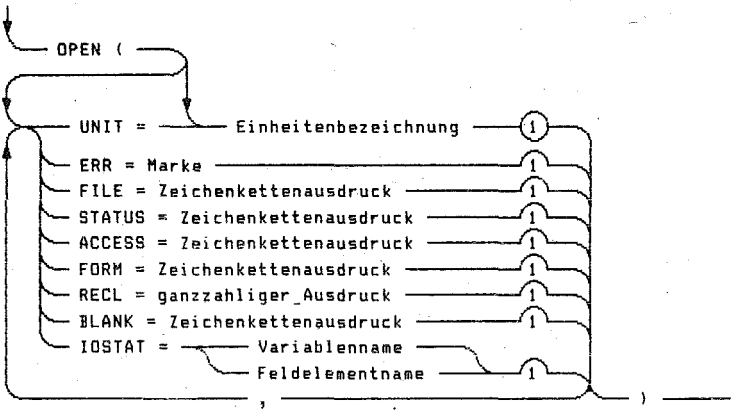


(48) In einer READ-Anweisung muß der E/A-Listenausdruck ein Variablenname, Feldelementname oder Teilkettenname sein.

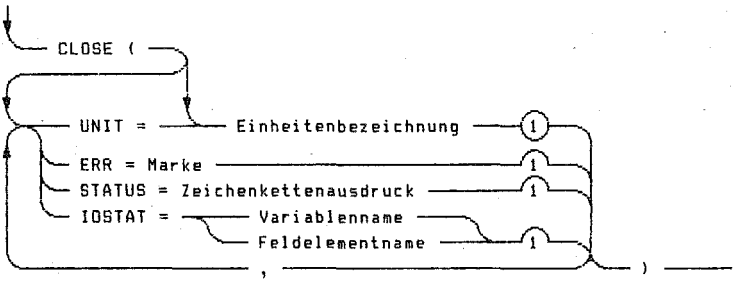
49 implizites_do_in_E/A_Liste:



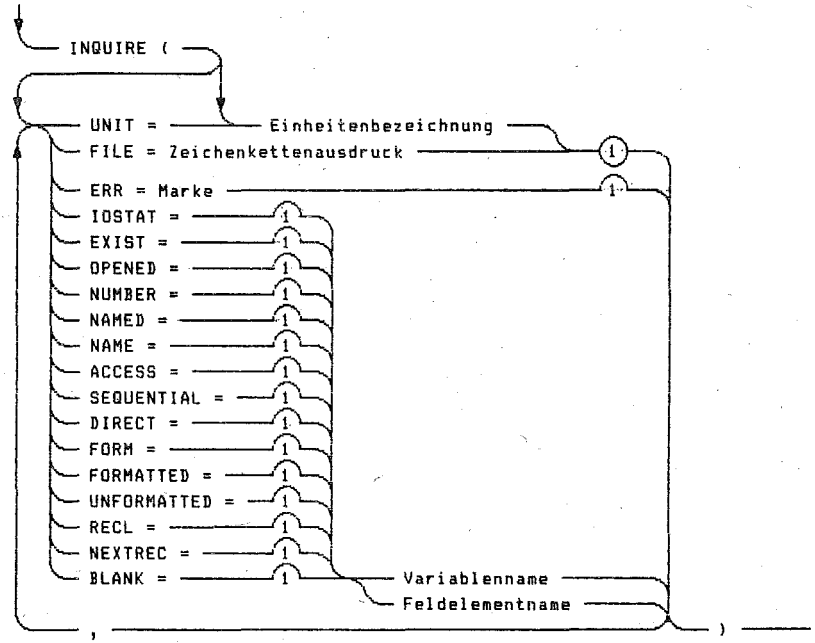
50 open_Anweisung:



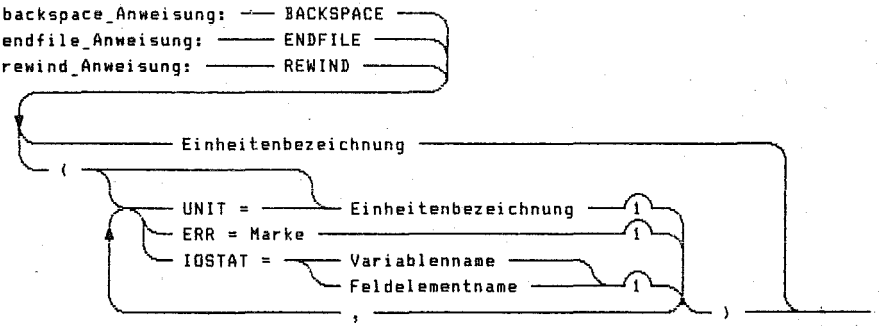
51 close_Anweisung:



52 inquire_Anweisung:

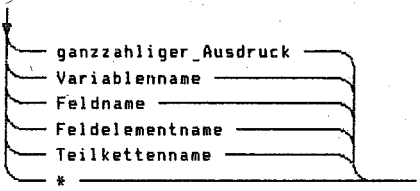


53 backspace_Anweisung:
54 endfile_Anweisung:
55 rewind_Anweisung:



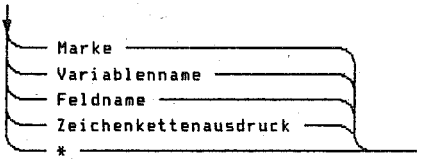
(53,54,55) BACKSPACE-, ENDFILE- und REWIND-Anweisungen müssen eine Einheitenbezeichnung enthalten.

56 Einheitenbezeichnung:



(56) Die Einheitenbezeichnung muß vom Datentyp INTEGER oder CHARACTER oder ein Stern sein.

57 Formatbezeichnung:

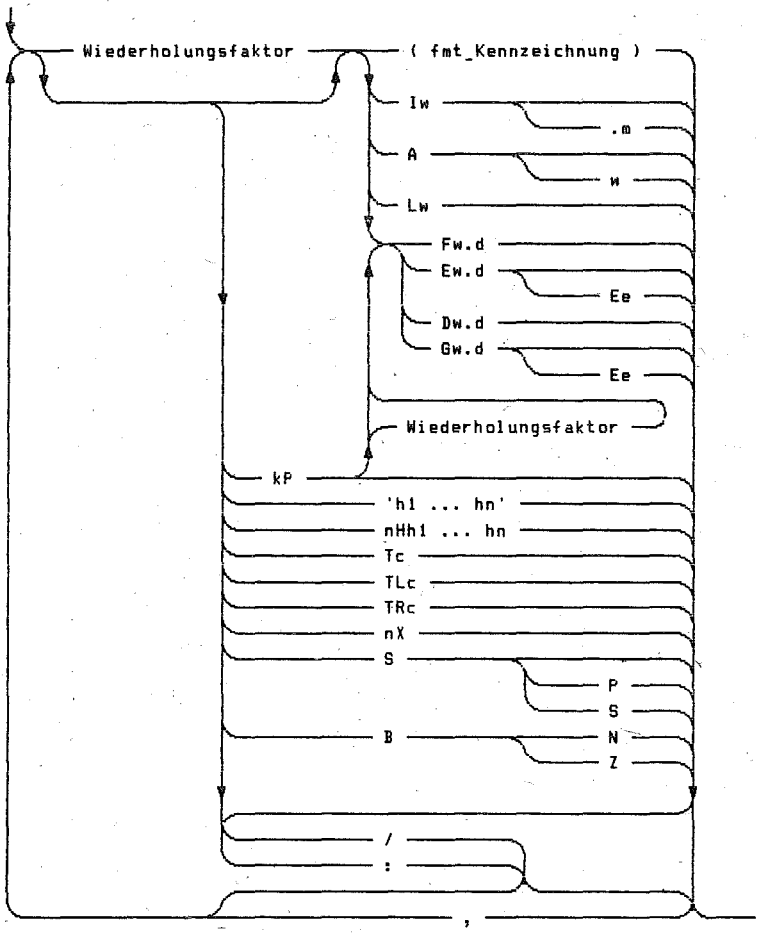


(57) Ist die Formatbezeichnung ein Variablenname oder Feldname, so muß sie vom Datentyp INTEGER oder CHARACTER sein.

58 format_Anweisung: ——— FORMAT format_Kennzeichnung ———

59 format_Kennzeichnung: —— (——— fmt_Kennzeichnung ————) ———

60 fmt_Kennzeichnung:



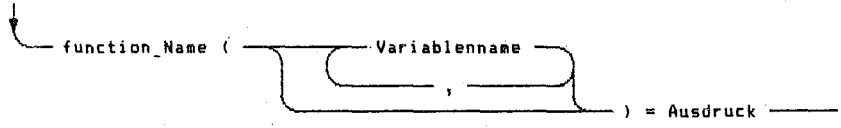
61 Wiederholungsfaktor:
62 w:
63 e:
64 n:
65 c: vorz.lose_ganzzahl_Konstante_versch_von_Null

66 d:
67 m: vorzeichenlose_ganzzahlige_Konstante

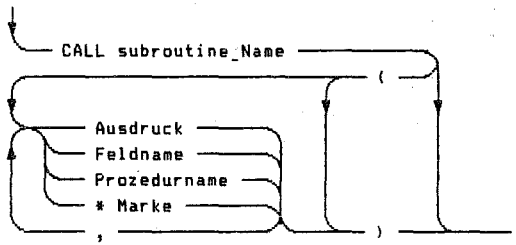
68 k: ganzzahlige_Konstante

69 h: gültiges_Zeichen_des_Betriebssystems

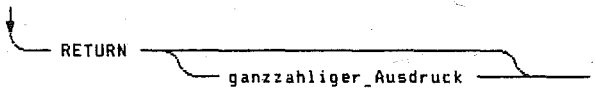
70 Anweisungsfunktionsdefinition:



71 call_Anweisung:

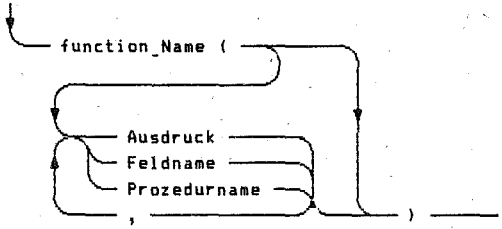


72 return_Anweisung:

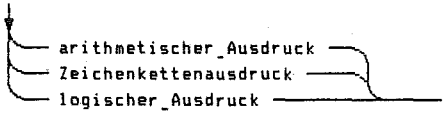


(72) In einem FUNCTION-Unterprogramm ist ein alternativer Rücksprung verboten.

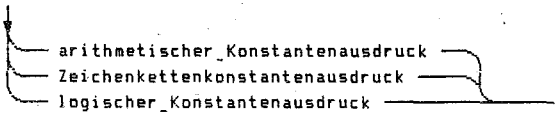
73 Funktionsbezugnahme:



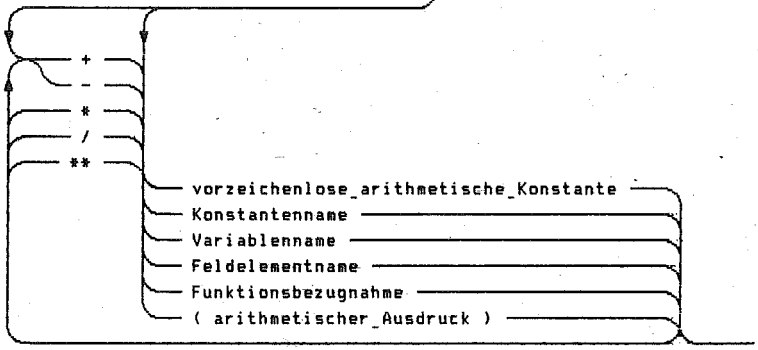
74 Ausdruck:



75 Konstantenausdruck:



76 arithmetischer_Ausdruck:
77 ganzzahliger_Ausdruck:
78 gemischter_arithmetischer_Ausdruck:

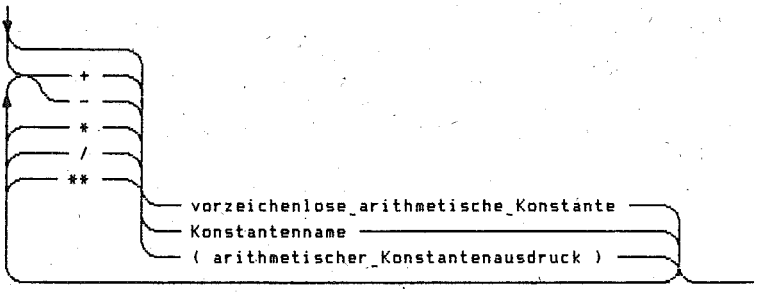


(76) Ein Konstantenname, Variablenname, Feldelementname oder eine Funktionsbezugnahme in einem arithmetischen Ausdruck muß vom Datentyp INTEGER, REAL, DOUBLE PRECISION oder COMPLEX sein.

(77) Ein ganzzahliger Ausdruck ist ein arithmetischer Ausdruck vom Datentyp INTEGER.

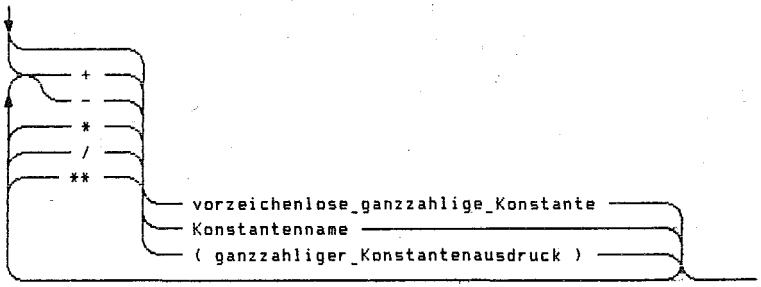
(78) Ein gemischter arithmetischer Ausdruck ist ein Ausdruck vom Datentyp INTEGER, REAL oder DOUBLE PRECISION.

79 arithmetischer_Konstantenausdruck:



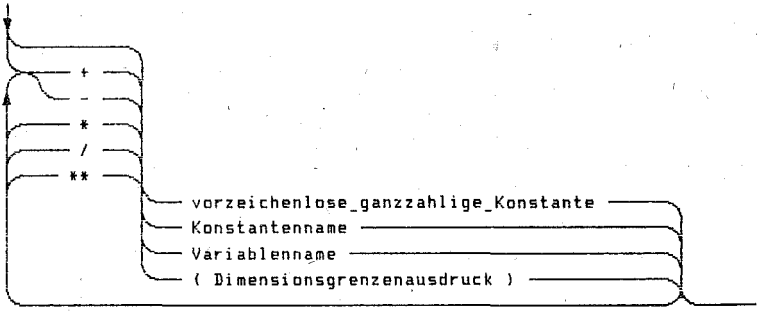
(79) Ein Konstantenname in einem arithmetischen Konstantenausdruck muß vom Datentyp INTEGER, REAL, DOUBLE PRECISION oder COMPLEX sein. Ein Exponent in einem arithmetischen Konstantenausdruck muß vom Datentyp INTEGER sein.

80 ganzzahliger_Konstantenausdruck:



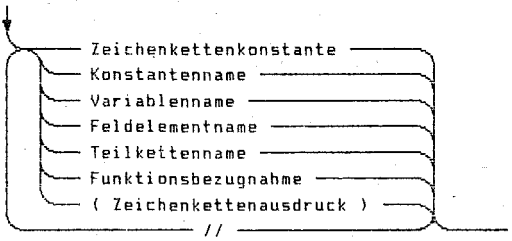
(80) Ein Konstantenname in einem ganzzahligen Konstantenausdruck muß vom Datentyp INTEGER sein.

81 Dimensionsgrenzenausdruck:



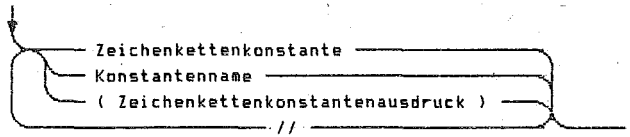
(81) Jeder Variablenname in einem Dimensionsgrenzenausdruck muß vom Datentyp INTEGER sein und muß ein formaler Parameter oder Element eines COMMON-Bereiches sein.

82 Zeichenkettenausdruck:



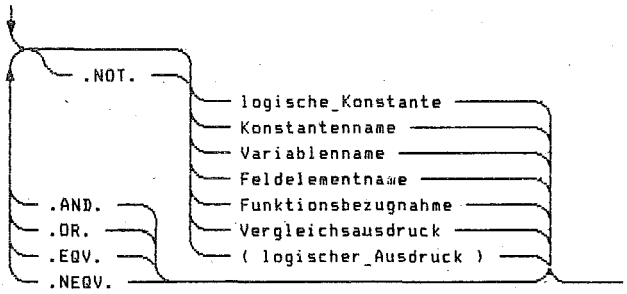
(82) Ein Konstantenname, Variablenname, Feldelementname oder eine Funktionsbezugnahme in einem Zeichenkettenausdruck muß vom Datentyp CHARACTER sein.

83 Zeichenkettenkonstantenausdruck:



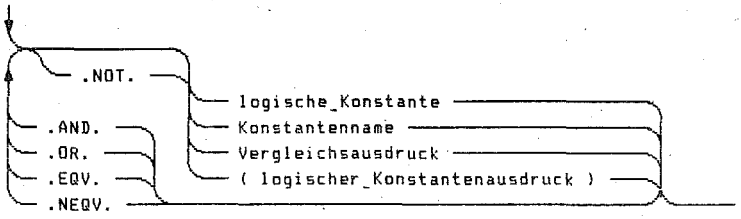
(83) Ein Konstantenname in einem Zeichenkettenkonstantenausdruck muß vom Datentyp CHARACTER sein.

84 logischer_Ausdruck:



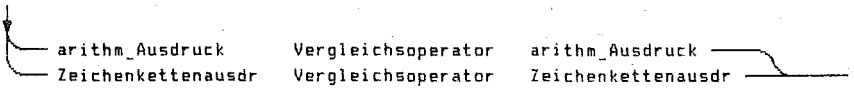
(84) Ein Konstantenname, Variablenname, Feldelementname oder eine Funktionsbezugnahme in einem logischen Ausdruck muß vom Datentyp LOGICAL sein.

85 logischer_Konstantenausdruck:



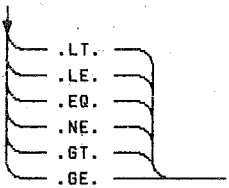
(85) Ein Konstantenname in einem logischen Konstantenausdruck muß vom Datentyp LOGICAL sein.

86 Vergleichsausdruck:

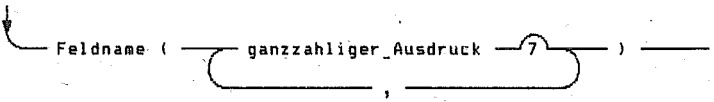


(86) Ein arithmetischer Ausdruck vom Datentyp COMPLEX ist nur erlaubt, wenn der Vergleichsoperator .EQ. oder .NE. ist.

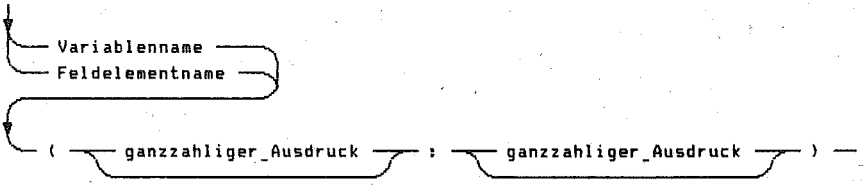
87 Vergleichsoperator:



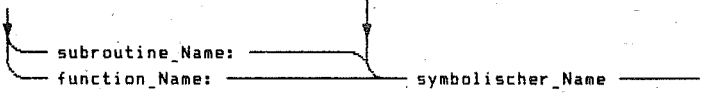
88 Feldelementname:



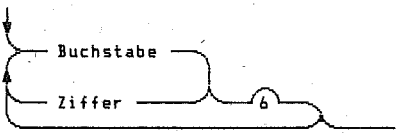
89 Teilkettenname:



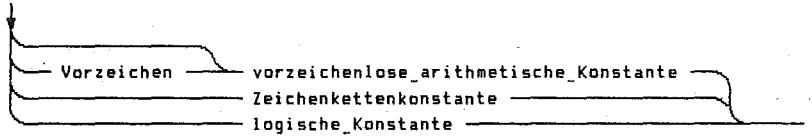
90 Konstantenname:
91 Variablenname:
92 Feldname:
93 common_Bereich_Name:
94 program_Name:
95 block_data_Unterprogrammname:
96 prozedurname:
97 subroutine_Name:
98 function_Name:



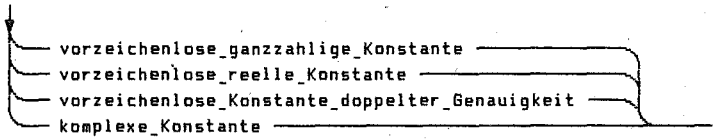
99 symbolischer_Name:



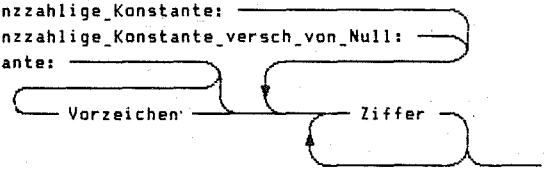
100 Konstante:



101 vorzeichenlose_arithmetische_Konstante:

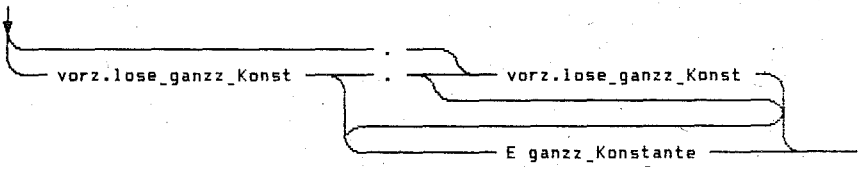


102 vorzeichenlose_ganzzahlige_Konstante:
103 vorzeichenlose_ganzzahlige_Konstante_versch_von_Null:
104 ganzzahlige_Konstante:

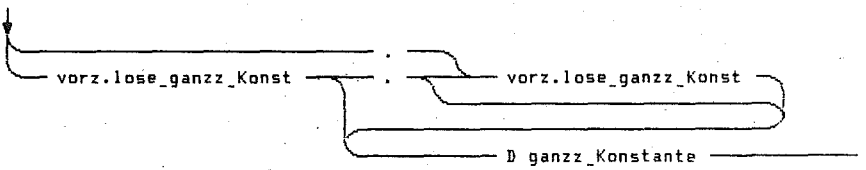


(103) Eine vorzeichenlose ganzzahlige Konstante verschieden von Null muß eine Ziffer verschieden von Null enthalten.

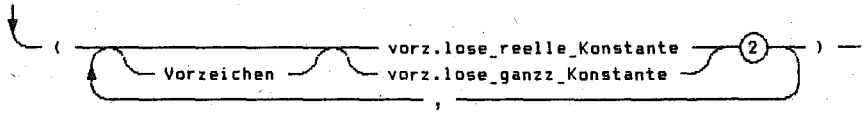
105 vorzeichenlose_reelle_Konstante:



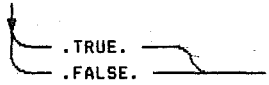
106 vorzeichenlose_Konstante_doppelter_Genauigkeit:



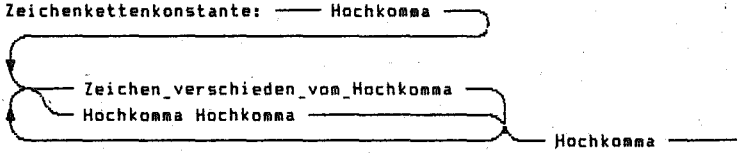
107 komplexe_Konstante:



108 logische_Konstante:

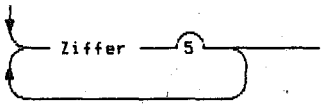


109 Zeichenkettenkonstante:



(109) Ein Hochkomma innerhalb einer Zeichenkette wird durch zwei aufeinanderfolgende Hochkommas ohne dazwischenliegendes Leerzeichen dargestellt.

110 Marke:



(110) Eine Marke muß eine Ziffer verschieden von Null enthalten.

Metasprachliche Bezeichnung	In der Beschreibung verwendeter Begriff	Definierende Syntaxregel
arithm_Ausdruck	arithmetischer Ausdruck	86
ganzzahl_Ausdruck	ganzzahliger Ausdruck	32
ganzz_Konstante	ganzzahlige Konstante	105, 106
ganzzahl_Konst.ausdr.	ganzzahliger Konstantenausdruck	28
ganzzahl_Konstantenausdruck	ganzzahliger Konstantenausdruck	22, 80
gem_arithm_Ausdr.	gemischt arithmetischer Ausdruck	34, 80
gem_arithm_Ausdruck	gemischt arithmetischer Ausdruck	40, 49
restl_Vereinbarungsanweisung	restliche Vereinbarungs-Anweisung	5
vorz.lose_ganzz_Konstante	vorzeichenlose ganzzahlige Konstante	107
vorz.lose_ganzz_Konst	vorzeichenlose ganzzahlige Konstante	105, 106
vorz.lose_ganzzahl_Konst versch_von_Null	vorzeichenlose ganzzahlige Konstante verschieden von Null	22, 27
vorz.lose_ganzz_Konstante versch_von_Null	vorzeichenlose ganzzahlige Konstante verschieden von Null	65
vorz.lose_reelle_Konstante	vorzeichenlose reelle Konstante	107
Zeichenkettenausdr	Zeichenkettenausdruck	86

Anlage 2: Standardfunktionen

Standard-Funktion	Beschreibung	Bemerkung	Anzahl d. Parameter	Generischer Name	Spezieller Name	Datentyp Parameter	Datentyp Funktion
Datentyp-Umwandlung	Umwandlung in INTEGER int(a)	1	1	INT	-	INTEGER*2	INTEGER
					-	INTEGER	INTEGER
					INT	REAL	INTEGER
					IFIX	REAL	INTEGER
					IDINT	DOUBLE	INTEGER
					-	COMPLEX	INTEGER
	Umwandlung in REAL	2	1	REAL	-	INTEGER*2	REAL
					REAL	INTEGER	REAL
					FLOAT	INTEGER	REAL
					-	REAL	REAL
					SNGL	DOUBLE	REAL
					-	COMPLEX	REAL
	Umwandlung in DOUBLE PRECISION	3	1	DBLE	-	INTEGER*2	DOUBLE
					-	INTEGER	DOUBLE
					-	REAL	DOUBLE
					-	DOUBLE	DOUBLE
					-	COMPLEX	DOUBLE
	Umwandlung in COMPLEX	4	1 oder 2	CMPLX	-	INTEGER*2	COMPLEX
					-	INTEGER	COMPLEX
					-	REAL	COMPLEX
					-	DOUBLE	COMPLEX
					-	COMPLEX	COMPLEX

Anlage 2: Standardfunktionen (Teil 2)

Standard-Funktion	Beschreibung	Bemerkung	Anzahl d. Parameter	Generischer Name	Spezieller Name	Datentyp Parameter	Datentyp Funktion
Datentyp-umwandlung	Umwandlung in INTEGER	5	1		ICHAR	CHARACTER	INTEGER
	Umwandlung in CHARACTER	5	1	CHAR	- -	INTEGER*2 INTEGER	CHARACTER CHARACTER
Abschneiden des gebroch. Teiles	int(a)	1	1	AINT	AINT DINT	REAL DOUBLE	REAL DOUBLE
Nächste ganze Zahl	int(a+.5), wenn a>0 int(a-.5), wenn a<0		1	ANINT	ANINT DNINT	REAL DOUBLE	REAL DOUBLE
Nächste Zahl vom Typ INTEGER	int(a+.5), wenn a>0 int(a-.5), wenn a<0		1	NINT	NINT IDNINT	REAL DOUBLE	INTEGER INTEGER
Absoluter Wert	|a| (AR**2+ AI**2)**1/2	6	1	ABS	- IABS ABS DABS CABS	INTEGER*2 INTEGER REAL DOUBLE COMPLEX	INTEGER*2 INTEGER REAL DOUBLE REAL

Anlage 2: Standardfunktionen (Teil 3)

Standard-Funktion	Beschreibung	Bemerkung	Anzahl d. Parameter	Generischer Name	Spezieller Name	Datentyp Parameter	Datentyp Funktion
Divisionsrest	a1-int(a1/a2)*a2	1	2	MOD	-	INTEGER*2	INTEGER*2
					MOD	INTEGER	INTEGER
					AMOD	REAL	REAL
					DMOD	DOUBLE	DOUBLE
Vorzeichen-Übertragung	|a1|, wenn a2>=0		2	SIGN	-	INTEGER*2	INTEGER*2
	-|a1|, wenn a2<0				ISIGN	INTEGER	INTEGER
					SIGN	REAL	REAL
					DSIGN	DOUBLE	DOUBLE
Schwache Differenz	a1-a2, wenn a1>a2		2	DIM	-	INTEGER*2	INTEGER*2
	0, wenn a1<=a2				IDIM	INTEGER	INTEGER
					DIM	REAL	REAL
					DDIM	DOUBLE	DOUBLE
Produkt mit Datentyp DOUBLE PRECISION	a1*a2		2		DPROD	REAL	DOUBLE
Maximum	MAX (a1...an)	9	=2	MAX	-	INTEGER*2	INTEGER*2
					MAX0	INTEGER	INTEGER
					AMAX1	REAL	REAL
					DMAx1	DOUBLE	DOUBLE
					AMAX0	INTEGER	REAL
					MAX1	REAL	INTEGER

Anlage 2: Standardfunktionen (Teil 4)

Standard-Funktion	Beschreibung	Bemerkung	Anzahl d. Parameter	Generischer Name	Spezieller Name	Datentyp Parameter	Datentyp Funktion
Minimum	MIN (a1...an)	9	=2	MIN	-	INTEGER*2	INTEGER*2
					MINO	INTEGER	INTEGER
					AMIN1	REAL	REAL
					DMIN1	DOUBLE	DOUBLE
					AMINO	INTEGER	REAL
					MIN1	REAL	INTEGER
Länge	Länge einer Zeichenkette	11	1		LEN	CHARACTER	INTEGER
Position einer Teilkette	Position von Teilkette a2 in Kette a1	10	2		INDEX	CHARACTER	INTEGER
Imaginärteil einer komplexen Zahl	ai	6	1		AIMAG	COMPLEX	REAL

Anlage 2: Standardfunktionen (Teil 5)

Standard-Funktion	Beschreibung	Bemerkung	Anzahl d. Parameter	Generischer Name	Spezieller Name	Datentyp Parameter	Datentyp Funktion
konjug. komplexe Zahl	(ar,-ai)	6	1		CONJG	COMPLEX	COMPLEX
Quadratwurzel	(a)**1/2	13	1	SQRT	SQRT DSQRT CSQRT	REAL DOUBLE COMPLEX	REAL DOUBLE COMPLEX
Exponential-Funktion	e**A		1	EXP	EXP DEXP CEXP	REAL DOUBLE COMPLEX	REAL DOUBLE COMPLEX
Natürlicher Logarithmus	log(a)	13	1	LOG	ALOG DLOG CLOG	REAL DOUBLE COMPLEX	REAL DOUBLE COMPLEX
Logarithmus zur Basis 10	log10(a)	13	1	LOG10	ALOG10 DLOG10	REAL DOUBLE	REAL DOUBLE

Anlage 2: Standardfunktionen (Teil 6)

Standard-Funktion	Beschreibung	Bemerkung	Anzahl d. Parameter	Generischer Name	Spezieller Name	Datentyp Parameter	Datentyp Funktion
Sinus	sin(a)	14	1	SIN	SIN DSIN CSIN	REAL DOUBLE COMPLEX	REAL DOUBLE COMPLEX
Cosinus	cos(a)	14	1	COS	COS DCOS CCOS	REAL DOUBLE COMPLEX	REAL DOUBLE COMPLEX
Tangens	tan(a)	14	1	TAN	TAN DTAN	REAL DOUBLE	REAL DOUBLE
Arcus sinus	arcsin(a)	15, 18	1	ASIN	ASIN DASIN	REAL DOUBLE	REAL DOUBLE
Arcus cosinus	arccos(a)	15, 19	1	ACOS	ACOS DACOS	REAL DOUBLE	REAL DOUBLE

Anlage 2: Standardfunktionen (Teil 7)

Standard-Funktion	Beschreibung	Bemerkung	Anzahl d. Parameter	Generischer Name	Spezieller Name	Datentyp Parameter	Datentyp Funktion
Arcus tangens	Arctan(a)	16, 18	1	ATAN	ATAN DATAN	REAL DOUBLE	REAL DOUBLE
	arctan (a1/a2)	17, 20	2	ATAN2	ATAN2 DATAN2	REAL DOUBLE	REAL DOUBLE
Sinus hyperbolicus	sinh(a)		1	SINH	SINH DSINH	REAL DOUBLE	REAL DOUBLE
Cosinus hyperbolicus	cosh(a)		1	COSH	COSH DCOSH	REAL DOUBLE	REAL DOUBLE
Tangens hyperbolicus	tanh(a)		1	TANH	TANH DTANH	REAL DOUBLE	REAL DOUBLE

Anlage 2: Standardfunktionen (Teil 8)

Standard-Funktion	Beschreibung	Bemerkung	Anzahl d. Parameter	Generischer Name	Spezieller Name	Datentyp Parameter	Datentyp Funktion
lexikalischer Vergleich	a1>=a2	12	2		LGE	CHARACTER	LOGICAL
	a1>a2	12	2		LGT	CHARACTER	LOGICAL
	a1<=a2	12	2		LLE	CHARACTER	LOGICAL
	a1<a2	12	2		LLT	CHARACTER	LOGICAL

Bemerkungen:

1. Hat a den Datentyp INTEGER, gilt int(a) = a. Ist a vom Datentyp REAL oder DOUBLE PRECISION, gibt es zwei Fälle. Wenn |a| < 1, dann ist int(a) = 0; für |a| >= 1 ist int(a) die ganze Zahl, deren Wert nicht größer ist, als der Wert von a, unter Beibehaltung des Vorzeichens. Zum Beispiel gilt
   int (-3.7) = -3
   Ist a vom Datentyp COMPLEX, gilt int(a) für den Realteil von a unter Einhaltung obengenannter Regel.

2. Hat a den Datentyp REAL, dann ist REAL(a) = a. Wenn a vom Datentyp INTEGER, INTEGER*2 oder DOUBLE PRECISION ist, dann ist REAL(a) so genau, wie ein REAL-Element signifikante Ziffern von A aufnehmen kann. Ist a vom Datentyp COMPLEX, dann ist REAL(a) der Realteil von a.

3. Ist a vom Datentyp DOUBLE PRECISION, dann ist DBLE(a) = a. Wenn a vom Datentyp INTEGER, INTEGER*2 oder REAL ist, hat DBLE(a) eine solche Genauigkeit, wie a signifikante Ziffern hat. Hat a den Datentyp COMPLEX, dann ist DBLE(a) der Realteil von a in einer Genauigkeit, wie der Realteil von a signifikante Ziffern hat.

4. CMPLX kann einen oder zwei Parameter haben. Wird nur ein Parameter angegeben, kann dieser vom Datentyp INTEGER*2, INTEGER, REAL, DOUBLE PRECISION oder COMPLEX sein. Sind zwei Parameter angegeben, müssen beide vom gleichen Datentyp sein. Der Datentyp COMPLEX ist dann nicht gestattet.
   Ist a vom Datentyp COMPLEX, so gilt CMPLX(a) = a. Hat a den Datentyp INTEGER, INTEGER*2, REAL oder DOUBLE PRECISION, ist CMPLX(a) ein komplexer Wert mit dem Realteil REAL(a) und dem Imaginärteil Null.
   CMPLX(a1,a2) ist ein komplexer Wert mit dem Realteil REAL(a1) und dem Imaginärteil REAL(a2).

5. ICHAR realisiert die Umwandlung eines Zeichens in eine ganze Zahl, basierend auf der Position des Zeichens in der Sortierfolge des ASCII-Codes. Das erste Zeichen in der Sortierfolge hat die Position 0, das letzte die Position n-1, wobei n die Anzahl aller Zeichen in der Sortierfolge ist. Der Wert von ICHAR(a) ist eine ganze Zahl im Bereich 0<=ICHAR(a)<=n-1. Dabei ist a ein Parameter vom Datentyp CHARACTER und der Länge 1. Das Zeichen a muß eine Codierung im ASCII besitzen. Die Position von a im ASCII-Code entspricht dem Wert von ICHAR(a).
   CHAR(i) gibt das Zeichen mit der i.ten Position in der ASCII-Codefolge zurück. Der Wert ist vom Datentyp CHARACTER und der Länge 1. Der Parameter i muß ein ganzzahliger Aus-

druck mit einem Wert im Bereich 0<=i<=n-1 sein.

6. Ein komplexer Wert wird als geordnetes Paar von reellen Zahlen in der Form (ar,ai) angegeben. Dabei ist ar der Realteil und ai der Imaginäerteil.

7. Alle Winkel werden in Radian angegeben.

8. Das Resultat einer Standardfunktion mit dem Datentyp COMPLEX ist der Hauptwert.

9. Alle aktuellen Parameter in einer Standardfunktionsbezugnahme müssen vom gleichen Datentyp sein.

10. INDEX(a1,a2) gibt einen ganzzahligen Wert zurück, der die Startposition der Teilkette a2 in der Zeichenkette a1 darstellt. Tritt a2 mehrfach in a1 auf, wird die Position des ersten Auftretens zurückgegeben. Ist dagegen a2 in a1 gar nicht enthalten oder LEN(a1) < LEN(a2), dann wird der Wert Null zurückgegeben.

11. Der aktuelle Parameter von LEN muß zur Zeit der Ausführung der Standardfunktion keinen definierten Wert haben.

12. LGE(a1,a2) gibt den Wert .TRUE. zurück, wenn a1=a2 oder a1 in der Sortierfolge nach ASCII-Code hinter a2 folgt. Anderenfalls wird der Wert .FALSE. zurückgegeben.
LGT(a1,a2) gibt den Wert .TRUE. zurück, wenn a1 in der Sortierfolge nach ASCII-Code hinter a2 steht. Ansonsten wird .FALSE. zurückgegeben.
LLE(a1,a2) gibt den Wert .TRUE. zurück, wenn a1 = a2 oder a1 in der Sortierfolge nach ASCII-Code vor a2 steht. Anderenfalls wird .FALSE. zurückgegeben.
LLT(a1,a2) gibt den Wert .TRUE. zurück, wenn a1 in der Sortierfolge nach ASCII-Code vor a2 steht. Anderenfalls wird .FALSE. zurückgegeben.
Sind die aktuellen Parameter für eine der Standardfunktionen LGT, LLE, LGE oder LLT ungleich lang, wird der kürzere so behandelt, als wäre er rechtsseitig mit Leerzeichen aufgefüllt.

13. Der Parameter a muß ein Wert größer gleich Null sein.

14. Der Absolutwert des Parameters a muß nicht kleiner als 2*PI sein.

15. Der Absolutwert des Parameters a muß kleiner oder gleich 1 sein.

16. Ist der Wert des Parameters a positiv, ist auch der Funktionswert positiv.

17. Abhängig vom Wert des Parameters a1 führt die Standardfunktion ATAN2/DATAN2 folgende Operationen durch:

```
a1 > 0 :  ATAN(a1/a2)
a1 < 0 :  SIGN(PI-ATAN(ABS(a2/a1),a2))
a1 = 0 :  SIGN(PI/2,a2)
```

18. Der Funktionswert FW liegt im Bereich -PI/2 <= FW <= PI/2.

19. Der Funktionswert FW liegt im Bereich 0 <= FW <= PI .

20. Der Funktionswert FW liegt im Bereich -PI <= FW <= PI .

Anlage 3: Englisch-deutsche Fachwörterliste

adjustable_array	dynamisches Feld
apostrophe	Hochkomma
arithmetic_const_expr	arithmetischer Konstantenausdruck
arithmetic_expression	arithmetischer Ausdruck
arithmetic_if_statement	arithmetische IF-Anweisung
array_declarator	Feldvereinbarung
array_element_name	Feldelementname
array_name	Feldname
assigned_goto	gesetztes GOTO
assignment_statement	Definitionsanweisung (Ergibtanweisung)
assumed_size_array	Feld von angenommener Größe
backspace_statement	BACKSPACE-Anweisung
block_data_stamenet	BLOCK DATA-Anweisung
block_data_subprogram_name	BLOCK DATA-Unterprogrammname
block_data_subprogram	BLOCK DATA-Unterprogramm
block_if_statement	BLOCK IF-Anweisung
call_statement	CALL-Anweisung
character_const_expr	Zeichenkettenkonstantenausdruck
character_constant	Zeichenkettenkonstante
character_expression	Zeichenkettenausdruck
character_substring	Teilkette
close_statemenet	CLOSE-Anweisung
common_block_name	COMMON-Bereich-Name
common_statement	COMMON-Anweisung
complex_constant	komplexe Konstante
computed_goto	berechnetes GOTO
constant_expr	Konstantenausdruck
constant_name	Konstantenname
constant	Konstante
continue_statement	CONTINUE-Anweisung
control_info_list	Liste der Steuerinformationen
data_implied_do_list	implizites DO der DATA-Anweisung
data_statement	DATA-Anweisung
digit	Ziffer
dim_bound_expr	Dimensionsgrenzenausdruck
dimension_statement	DIMENSIOn-Anweisung
do_statement	DO-Anweisung
else_if_statement	ELSEIF-Anweisung
else_statement	ELSE-Anweisung
end_if_statement	ENDIF-Anweisung
endfile_statement	ENDFILE-Anweisung

entry_statement	ENTRY-Anweisung
equiv_entity	Äquivalenzgröße
equivalence_statement	EQUIVALENCE-Anweisung
executable_statement	ausführbare Anweisung
expression	Ausdruck
external_statement	EXTERNAL-Anweisung
fmt_specification	FMT-Kennzeichnung
format_identifier	Formatbezeichnung
format_specification	FORMAT-Kennzeichnung (Angabe)
format_statement	FORMAT-Anweisung
function_entry	FUNCTION-Eingangspunkt
function_name	FUNCTION-Name
function_reference	Funktionsbezugnahme
function_statement	FUNCTION-Anweisung
function_subprogram	FUNCTION-Unterprogramm
generic	generisch
goto_statement	GOTO-Anweisung
implicit_statement	IMPLICIT-Anweisung
inquire_statement	INQUIRE-Anweisung
int_constant_expr	ganzzahliger Konstantenausdruck
int_real_dp_expr	gemischter arithmetischer Ausdruck
integer_constant	ganzzahlige Konstante
integer_expr	ganzzahliger Ausdruck
intrinsic_statement	INTRINSIC-Anweisung
io_implied_do_list	implizites DO in E/A-Anweisungen
io_list	E/A-Liste
label	Marke
len_specification	Längenangabe
letter	Buchstabe
logical_const_expr	logischer Konstantenausdruck
logical_constant	logische Konstante
logical_expression	logischer Ausdruck
logical_if_statement	logische IF-Anweisung
main_program	Hauptprogramm
nonapostrophe_character	Zeichen verschieden vom Hochkomma
nonzero_unsigned_int_constant	vorzeichenlose ganzzahlige Konstante verschieden von Null
open_statement	OPEN-Anweisung
other_specification_statement	restliche Vereinbarungs-anweisungen
parameter_statement	PARAMETER-Anweisung

pause_statement	PAUSE-Anweisung
print_statement	PRINT-Anweisung
procedure_name	Prozedurname
processor_character	gültiges Zeichen des Betriebssystems
program_name	PROGRAM-Name
program_statement	PROGRAM-Anweisung
read_statement	READ-Anweisung
real_op	Vergleichoperator
relational_expression	Vergleichsausdruck
repeat_spec	Wiederholungsfaktor
return_statement	RETURN-Anweisung
rewind_statement	REWIND-Anweisung
save_statement	SAVE-Anweisung
sign	Vorzeichen
statement_function_statement	Anweisungsfunktionsdefinition
stop_statement	STOP-Anweisung
subroutine_entry	SUBROUTINE-Eingangspunkt
subroutine_name	SUBROUTINE-Name
subroutine_statement	SUBROUTINE-Anweisung
subroutine_subprogram	SUBROUTINE-Unterprogramm
substring_name	Teilkettenname
symbolic_name	symbolischer Name
type_statement	Typanweisung
unconditional_goto	unbedingtes GOTO
unit_identifier	logische Gerätenummer
unsigned_arithmetic_constant	vorzeichenlose arithmetische Konstante
unsigned_dp_constant	vorzeichenlose Konstante doppelter Genauigkeit
unsigned_int_constant	vorzeichenlose ganzzahlige Konstante
unsigned_real_constant	vorzeichenlose reelle Konstante
variable_name	Variablenname
write_statement	WRITE-Anweisung

Literaturverzeichnis

[1] C 1015-0200-1 M 3030 FOR77 Benutzungshinweise SCP 1700

oder

C 3015-0200-1 M 3030 FOR77 Benutzungshinweise DCP

Sachwortverzeichnis