UDOS–Software
Programmiersprachen

EAW*electronic*

P8000

Diese Dokumentation wurde von einem Kollektiv des Kombinates

VEB ELEKTRO-APPARATE-WERKE
BERLIN-TREPTOW "FRIEDRICH EBERT"

erarbeitet.

Herausgeber:

Kombinat
VEB ELEKTRO-APPARATE-WERKE
BERLIN-TREPTOW "FRIEDRICH EBERT"
Hoffmannstrasse 15-26
BERLIN
1193

WAE/03-0103-01

BG 117/05/87

Ausgabe: 12/86

Die vorliegende Dokumentation unterliegt nicht dem Aenderungsdienst.

Spezielle Hinweise zum aktuellen Stand der Softwarepakete befinden sich in README-Dateien auf den entsprechenden Vertriebsdisketten.

Dieser Band enthaelt folgende Unterlagen:

- U880-BASIC Benutzerhandbuch
  (Bearbeiter: J. Kubisch)

- U880-PASCAL Benutzerhandbuch
  (Bearbeiter: U. Oefler)

- U880-FORTRAN Benutzerhandbuch
  (Bearbeiter: L. Mielenz)

- U880-PLZ/SYS Benutzerhandbuch
  (Bearbeiter: H. Pille)

U S S O - B A S I C

Benutzerhandbuch

VORWORT

Die vorliegende Anwenderdokumentation gibt eine gestraffte Beschreibung der Programmiersprache BASIC. Sie ist als Arbeitsgrundlage fuer den BASIC-kundigen Nutzer gedacht. Eine ausfuehrliche Sprachbeschreibung kann der folgenden Literatur entnommen werden:

- BASIC-Interpreter fuer UDOS 1526. Anwenderdokumentation. VEB Robotron-Buchungsmaschinenwerk Karl-Marx-Stadt

- Mueller, S.: Programmieren mit BASIC. Reihe Automatisierungstechnik, Bd. 216. 2. bearb. Auflage. Berlin: VEB Verlag Technik 1986

- Strelocke, K.; Hoffmann, P.: Die Dialogprogrammiersprache BASIC: Sprachbeschreibung und Anwendungsbeispiele. Berlin: Verlag Die Wirtschaft 1981

- Werner, D.: BASIC fuer Mikrorechner. Programmentwicklung. Sprachelemente. Anwendungen. Berlin: VEB Verlag Technik 1986

Inhaltsverzeichnis Seite

1. U880 BASIC-Interpreter

Die Programmiersprache BASIC (Abk. fuer Beginner's All purpose Symbolic Instruction Code) ist eine der am weitesten verbreiteten Dialogsprachen. Neben den Programm-anweisungen, in denen die Aufgabe formuliert wird, gibt es eine Reihe von Kommandos, die das Erstellen, Testen, Aendern und Abarbeiten der Programme im Dialogbetrieb ermoeglichen. Dieses Konzept gestattet einen schnellen und unkomplizierten Einstieg in die Programmierung. Da das Programm mittels eines Interpreters ausgefuehrt wird, kann es ohne vorherige Uebersetzung abgearbeitet werden.
Durch die Verarbeitung von arithmetischen Ausdruecken und die Moeglichkeit der Zeichenkettenmanipulation eignet sich BASIC sowohl fuer wissenschaftlich-technische als auch fuer kommerzielle Aufgabenstellungen kleinen bis mittleren Umfangs. Die hier beschriebene BASIC-Version fuer den U880 unter dem Betriebssystem UDOS nutzt eine Vielzahl der Moeglichkeiten des Systems (z.B.: Dateiarbeit, Zuweisung von Ein-/Ausgabestroemen u.a.m.) konsequent aus und gestattet daher dem Anwender ein relativ einfaches Einbinden dieser Faehigkeiten in seine Programme.

2. Sprachelemente

Ein BASIC-Programm ist zeilenorientiert. Jede Zeile beginnt mit einer Zeilennummer. Beim Start eines Programms arbeitet der Interpreter die Programmzeilen in der Reihenfolge ihrer Zeilennummern unabhaengig von der Stellung im Quelltext ab (Ausnahmen: Spruenge, Unterprogramme, Schleifen).

Dies ist ein Beispiel fuer ein Programm
mit nur einer Anweisung:

```
>100 PRINT "5*10 = ";5*10
```

100 ist die Anweisungsnummer. PRINT ist das Schluesselwort, das dem Interpreter die Art der auszufuehrenden Aktion mitteilt. In diesem Fall wird die Zeichenkette "5*10=" und der Wert des folgenden Ausdrucks ausgegeben. 5*10 ist ein arithmetischer Ausdruck, der durch den Interpreter ausgewertet wird. Normalerweise enthaelt ein Programm mehr als eine Anweisung.

Diese vier Anweisungen sind ein Programm:

```
>10 INPUT A,B
>20 LET C=A+B
>30 PRINT
>40 PRINT A;"+";B;"=";C
```

Dieses Programm, das die Summe zweier Zahlen berechnet, ist in der Reihenfolge seiner Ausfuehrung gezeigt. Es koennte in beliebiger Reihenfolge eingegeben werden, wenn die jeder Anweisung zugeordneten Anweisungsnummern nicht geaendert werden.

Falls eine Programmzeile geaendert werden soll, so ist eine Zeile mit der gleichen Zeilennummer neu einzugeben. Bei der

Eingabe der Programmzeilen erfolgt eine Pruefung auf syntaktische Richtigkeit. Sinnvollerweise werden die Zeilen mit einem Inkrement groesser 1 numeriert, um die Moeglichkeit des Einfuegens von Befehlsfolgen zu erhalten. Die Eingabe nur einer Zeilennummer streicht die entsprechende Programmzeile. Die Anweisungen haben ein freies Format, d.h., Leerzeichen werden ignoriert. Der Aufruf des BASIC-Interpreters erfolgt vom UDOS aus durch Eingabe folgender Kommandozeile:

```
%BASIC
 (Meldung von BASIC)
 ...
 ...
>
```

Der Interpreter zeigt seine Dialogbereitschaft durch Ausgabe des Zeichens ">" an. Eine erwartete Eingabe waehrend der Ausfuehrung einer INPUT-Anweisung wird durch das Zeichen "?" angezeigt. Fehler- und Warnmeldungen werden durch eine Nummer gekennzeichnet. Die Erlaeuterungen dazu sind dem Anhang zu entnehmen.
Der BASIC-Interpreter arbeitet intern mit einem Gleitkommapaket, das eine 13stellige Mantisse realisiert.

Kommandos realisieren Steuerfunktionen, die Aktionen werden direkt ausgefuehrt. Anweisungen dagegen dienen zur Formulierung des Problems, sie werden erst bei Programmabarbeitung ausgefuehrt. Aehnlich den Kommandos koennen aber auch einige Anweisungen fuer Testzwecke ebenfalls direkt abgearbeitet werden. Sie muessen dann ohne Zeilennummer eingegeben werden.

2.1. Kommandos

Die Kommandos unterteilen sich in drei Gruppen: Programmausfuehrungs-, Editier- und diskettenorientierte Kommandos. Sie werden direkt hinter das Bereitschaftszeichen ">" gesetzt und koennen mit ihren ersten drei Buchstaben abgekuerzt werden. Sie werden unmittelbar ausgefuehrt.
Bei der Angabe der Kommandosyntax werden folgende Konventionen benutzt:

GROSS-BUCHSTABEN	Schluesselworte, die genau so geschrieben werden muessen
[ ]	umschliessen optionale Begriffe
( )	umschliessen vorgegebene Begriffe
|	trennt Alternativen, von denen eine gewaehlt werden muss

2.1.1. Programmausfuehrungskommandos

Die Programmausfuehrungskommandos unterstuetzen die Fehlersuche in einem Programm. Die Programmausfuehrung kann entweder durch von-Hand-Eingriff oder unter Programmkontrolle unterbrochen werden. Variablen koennen beobachtet und/oder geaendert werden, Programmteile koennen gezeigt oder geaendert werden, danach kann die Ausfuehrung fortgesetzt werden.

Programmausfuehrungskommandos:

RUN[-Zeilennummer]	Start eines BASIC-Programms bei der ersten ausfuehrbaren Anweisung, oder, falls angegeben, bei der spezifizierten Programmzeile.
CONTINUE[-Zeilennr]	Weiterlauf eines unterbrochenen Programms (durch eine STOP-Anweisung oder Betaetigen der DEL-Taste) bei der naechsten Anweisung, oder, falls angegeben, bei der spezifizierten Programmzeile.
STEP	Weiterlauf des Programms bis zum Beginn der naechsten Anweisung. ("Einzelschrittbetrieb")
QUIT	Verlassen des BASIC-Interpreters und Rueckkehr ins Betriebssystemniveau. Alle geoeffneten Dateien werden geschlossen und der Benutzerarbeitsbereich geloescht.

2.1.2. Editierkommandos

Die folgenden Editierkommandos beziehen sich immer auf das aktuelle Programm, das sich im Arbeitsspeicher (RAM) des BASIC-Interpreters befindet.

Editierkommandos:

LIST[-Bereich]	Vollstaendiges oder teilweises Auslisten des aktuellen Programms:
LIS-n	nur der Programmzeile n
LIS-n,	alle Programmzeilen ab n bis zum Ende
LIS-,n	alle Programmzeilen von der ersten bis zur Zeile n (einschliesslich)
LIS-n,m	von Programmzeile n bis m (einschliesslich)
LIS-,	des gesamten Programms

Beispiele:

```
>LIST
```

Das ganze aktuelle Programm wird aufgelistet.

```
>LIST-1,100
```

Die Anweisungen 1 bis 100 des aktuellen Programms werden aufgelistet.

Bemerkung:

Eine Auflistung kann durch Betaetigung der DEL-Taste abgebrochen werden. Die Kontrolle kehrt ans BASIC-System zurueck. Die Auflistung kann auch durch Druck der "?"-Taste unterbrochen werden. Der erneute Druck von "?" setzt die Ausgabe fort.

NEW[-Pufferanz.] Loeschen des aktuellen Programms. Dabei koennen fuer die Dateiarbeit benoetigte Puffer (0 bis 15) zu je 512 Byte reserviert werden.

Wenn die Zahl der anzulegenden Puffer (0-15) nicht angegeben ist, werden zwei Puffer angelegt. Ein Puffer wird fuer jede offene Datei benoetigt. Ein Puffer wird fuer Dateikommandos benoetigt.

Beispiel:

```
>NEW-7
```

Das aktuelle Programm wird geloescht, sieben Puffer werden angelegt und ein neues aktuelles Programm kann in den Benutzer-Arbeitsbereich eingegeben werden. Das SIZE-Kommando zeigt die Zahl der angelegten Puffer.

```
>SIZ:      AVAIL=9213 PROG=0 VAR=0 BUF=7
```

DELETE[-Bereich] Streichen von Programmzeilen. Der Bereich kann analog zum LIST-Kommando angegeben werden.

RENUMBER[-neuerst[,delta[,alterst[,altletzt]]]]

Umnumerierung eines gesamten Programms, oder, falls angegeben, eines durch alterst (voreingestellt:1) und altletzt (voreingestellt:9999) spezifizierten Programmbereichs. Dabei geben neuerst (voreingestellt:10) die neue erste Zeilennummer und delta (voreingestellt:10) die Schrittweite an. Falls sie nicht spezifiziert werden, so wird mit den

voreingestellten Werten gearbeitet. Jede Anweisung, die auf eine umnumerierte Anweisung verweist, wird sinngemaess geaendert. Bei Ausfuehrung dieses Kommandos muss das Programm RENUMBER.PROG.BP auf der Diskette verfuegbar sein. Das Kommando erstellt die Dateien PROG.BEFORE.REN.BP (enthaelt das alte Programm) und NEW.REN.PROG.BP (enthaelt das umnumerierte Programm).

Beispiel:

```
>RENUMBER
```

Die Anweisungen des aktuellen Programms werden in 10-er-Schritten beginnend mit 10 umnumeriert.

```
>REN-3,7,50,250
```

Die Anweisungeh mit den alten Nummern 50 bis 250 werden umnumeriert. Kleinste Nummer ist 3, Schrittweite ist 7.

SIZE — Ausgabe des Status des aktuellen Programms. Dabei werden die Anzahl der verfuegbaren Bytes (AVAIL=...), die Groesse des durch das Programm (PROG=...), sowie durch Variable (VAR=...) belegten Speichers in Bytes und die Anzahl der angelegten Pufferbereiche (BUF=...)ausgegeben.

CLEAR — Ruecksetzen aller durch ein Programm realisierten Zuweisungen. Alle Variablen werden undefiniert, alle haengenden Funktionsaufrufe, GOSUBs und FORs werden rueckgesetzt, alle Dateien geschlossen. CLEAR wird vom RUN-Kommando automatisch generiert.

Beispiel:

```
>SIZ:   AVAIL=9460 PROG=36 VAR=24 BUF=2
>CLEAR
>SIZ:   AVAIL=9520 PROG=0  VAR=0  BUF=2
```

Beispiele mit Editier-Kommandos:

Der Benutzer gibt ein Programm ein, macht einen Fehler in Zeile 30 und gibt die Zeile neu ein.

```
>10 INPUT   A,B,C,D,E
>20 REM..EINGABE 5 WERTE
```

```
>30 LET R=(A+B)/5
>40 REM..S=Mittelwert der 5 Eingabewerte
>50 PRINT S
>30 LET S=(A+B+C+D+E)/5
```

LIST listet das Programm richtig auf:

```
>LIST
  10 INPUT A,B,C,D,E
  20 REM..Eingabe 5 Werte
  30 LET S=(A+B+C+D+E)/5
  40 REM..S=Mittelwert der 5 Eingabewerte
  50 PRINT S
```

SIZE gibt die Laenge in Bytes an:

```
>SIZ:   AVAIL=11648 PROG=125 VAR=0 BUF=2
```

Die Kommentarzeilen werden geloescht und das Programm aufgelistet:

```
>DEL-20
>DEL-40
>LIST
  10 INPUT A,B,C,D,E
  30 LET S=(A+B+C+D+E)/5
  50 PRINT S
>SIZ:   AVAIL=11710 PROG=63 VAR=0 BUF=2
```

Als naechstes wird das Programm umnumeriert und erneut aufgelistet:

```
>RENUMBER
>LIST
  10 INPUT A,B,C,D,E
  20 LET S=(A+B+C+D+E)/5
  30 PRINT S
```

Das Programm wird geloescht. Wenn nun LIST gegeben wird, gibt es kein aktuelles Programm; der Computer bringt ein ">" und erwartet weitere Eingabe:

```
>NEW
>LIST
>
```

2.1.3. Diskettenbezogene Kommandos

Um eingegebene BASIC-Programme fuer eine spaetere Verwendung zu speichern und vorhandene Programme zu laden, existieren eine Reihe von diskettenbezogenen Kommandos. Die so erstellten Dateien erhalten zur zusaetzlichen Kennzeichnung dabei automatisch die Endung ".BP". Diese Endung wird bei Manipulationen an Programmdateien im BASIC-System nicht benutzt. Die Endung muss aber bei Manipulationen solcher Programmdateien ausserhalb des BASIC-Systems angegeben werden.

Diskettenbezogene Kommandos:

SAVE-Programmname	Speichert eine Kopie des aktuellen Programms in verdichteter Form unter dem angegebenen Dateinamen auf Diskette ab. Diese Form ist kompakter als die lesbare Form des Quellprogramms und erlaubt somit ein schnelleres Laden.
ASAVE-Programmname	Speichert eine Kopie des aktuellen Programms in Quelltextdarstellung unter dem angegebenen Dateinamen auf Diskette ab. Die so erstellten Dateien koennen mit UDOS-Dienstprogrammen (z.B.: Editor) weiter bearbeitet werden.
RSAVE-Programmname	Analog SAVE und ASAVE, die angegebene Datei muss bereits existieren. Sie wird ueberschrieben. Die Abspeicherung in verdichteter oder Quellform richtet sich nach dem alten Inhalt der Datei.
GET-Programmname	Laden des angegebenen BASIC-Programms von Diskette in den Arbeitsbereich. Es ersetzt das aktuelle Programm.
XEQ-Programmname	Laden des angegebenen BASIC-Programms und starten des Programms. Entspricht GET-Kommando gefolgt von RUN.
APPEND-Programmname	Anhaengen des angegebenen Programms an das aktuelle Programm im Arbeitsspeicher. Das zu ladende Programm muss in Quelltextdarstellung (ASAVE-Form) vorliegen.

Um sich vom BASIC-Niveau aus einen Ueberblick ueber vorhandene Dateien und deren Inhalt verschaffen zu koennen, dienen die Programme CAT.BP (Auflisten von Dateien) und FLIST.BP (Ausgabe des Inhalts einer Datei). Die Programmlistings sind im Abschn. 4. angegeben. Die Programme koennen mit dem XEQ-Kommando vom BASIC-Interpreter aus abgearbeitet werden.

2.2. Ausdruecke

Ein Ausdruck verbindet Konstanten, Variablen oder Funktionen durch Operatoren zu einer geordneten Folge, deren Auswertung einen Wert ergeben muss.

2.2.1. Konstanten

Als Konstanten koennen Zahl- oder Zeichenkettenkonstanten verwendet werden. Zahlkonstanten (Integer- oder Gleitkommazahlen) koennen im Bereich von 10 hoch -128 bis 10 hoch +128 liegen. Zur Darstellung von Gleitkommazahlen kann auch die E-Notation (z.B.: 12.34567E-89) verwendet werden.
Zeichenkettenkonstanten bestehen aus einer Folge von ASCII-Zeichen (ausser " und <), die in Anfuehrungszeichen eingeschlossen sind (z.B.: "HALLO").

2.2.2. Variablen

Eine Variable ist ein Name, dem ein Wert zugewiesen werden kann. Es gibt numerische- und Zeichenkettenvariablen.
Numerische Variablen unterteilen sich in reelle Variablen (durch einen einzigen Buchstaben A...Z oder einen Buchstaben direkt gefolgt von einer Ziffer A0...Z9 gekennzeichnet und intern als Gleitkommazahl dargestellt) und ganzzahlige Variablen (Kennzeichnung analog den reellen Variablen, aber mit dem Zusatz "%", z.B.: I1% und intern als 16-Bit-Ganzzahl dargestellt).
Zeichenkettenvariablen koennen als Wert eine Zeichenkette besitzen. Ihre Kennzeichnung ist analog den reellen Variablen, aber mit dem Zusatz "$", z.B.: A0$. Zeichenkettenvariablen muessen vor ihrer Benutzung durch eine DIM-Anweisung (s. Abschn. 2.4.2.) vereinbart werden.
Die Einschraenkung der Variablennamen auf einen Buchstaben, eventuell noch durch eine Ziffer ergaenzt, behindert die Selbstdokumentation der Variablenbezeichnungen. Deshalb sollten ausreichend Kommentare zur Variablenverwendung in die Quelltexte integriert werden.

Eine Variable kann ein Feld (maximal zweidimensional) kennzeichnen. Felder koennen numerische Werte (ganzzahlig oder reell) oder Zeichenketten enthalten. Auf die Elemente kann ueber Indizierung zugegriffen werden. Als Index kann eine ganzzahlige Konstante, eine Variable oder ein Ausdruck, der auf einen ganzzahligen Wert gerundet wird, auftreten. Die Indexliste steht, in Klammern eingeschlossen, hinter dem Variablennamen. Sie kann aus einem oder aus zwei, durch Komma getrennten, Indizes bestehen. Der erste Index bezeichnet stets die Zeilen-, der zweite die Spaltennummer.
Eine einfache numerische Variable kann denselben Namen haben wie ein numerisches Feld, ohne dass dies zu Komplikationen fuehrt. Numerische Felder muessen vor ihrer Benutzung durch eine DIM-Anweisung vereinbart werden, wenn die Dimension groesser als 10 Zeilen (bei eindimensionalen Feldern) oder groesser als 10 Zeilen und 10 Spalten (bei zweidimensionalen Feldern) ist. Der gerundete Wert eines Indizes muss ein Wert zwischen 1 und dem maximalen Zeilen- oder Spaltenwert sein.
Zeichenkettenfelder koennen nur eindimensional sein. Sie muessen immer durch eine DIM-Anweisung vereinbart werden. Der Name einer einfachen Zeichenkettenvariablen darf im Gegensatz zu den numerischen Variablen nicht mit dem Namen eines Zeichenkettenfeldes uebereinstimmen!

2.2.3. Funktionen

Eine Funktion ist eine durch einen Namen gekennzeichnete Operation, die einen oder mehrere Parameterwerte benutzt, um einen einzelnen Wert zu berechnen.
Funktionsnamen koennen aus mehreren Buchstaben bestehen. Der Anhang "%" kennzeichnet eine ganzzahlige numerische Funktion und der Anhang "$" eine Zeichenkettenfunktion.
Da eine Funktion nur einen Wert liefert, kann sie in Ausdruecken wie eine Konstante oder Variable verwendet werden. Der Aufruf einer Funktion erfolgt durch Angabe des Namens, gefolgt von den in runden Klammern eingeschlossenen, durch Komma getrennten, aktuellen Parametern (z.B.: INT(A)).
Dem Anwender stehen eine Reihe von fest eingebauten Funktionen (s. Abschn. 2.5.) zur Verfuegung. Fuer spezielle Faelle koennen auch selbst Funktionen definiert werden. Neben den numerischen Funktionen sind vor allem die Zeichenkettenfunktionen hervorzuheben.

2.2.4. Operatoren

Operatoren fuehren eine mathematische, logische oder Zeichenkettenoperation auf einem (unaeres Plus oder Minus als Vorzeichen vor einem Wert) oder zwei Operanden aus und liefern einen Wert.

Art	Notation		Ergebnis
arithmet.Operatoren	+		numerischer Wert
	-		
	*		
	/		
	^		
Vergleichsoperatoren	=		0 fuer falsch
	<		oder
	>		1 fuer wahr
	<=		
	>=		
	<>	(ungleich)	
logische Operatoren	&	(UND)	0 fuer falsch
	!	(ODER)	oder
		(KOMPLEMENT)	1 fuer wahr
Zeichenkettenoperator	+	(Verkettung)	Zeichenkette

Die Reihenfolge der Ausfuehrung der Operationen wird durch die Rangordnung der Operatoren bestimmt.

Hierarchie der Operatoren:

unaeres +, unaeres -, ^ hoechste Prioritaet

*, /
binaeres +, binaeres -
Vergleichsoperatoren (=,<,>,<=,>=,<>)
&, ! niedrigste Prioritaet

Bei gleicher Rangordnung erfolgt die Berechnung von links nach rechts. Runde Klammern koennen verwendet werden, um die Berechnungsreihenfolge zu veraendern. Bei Schachtelung von Klammern werden die geklammerten Ausdruecke von innen nach aussen aufgeloest.

2.3. Anweisungen

Der Programmalgorithmus wird durch Anweisungen realisiert. Jede Anweisung fuehrt eine bestimmte Funktion aus. Ihr geht stets eine Anweisungsnummer voraus, durch die die Reihenfolge der Abarbeitung festgelegt wird.

2.3.1. Wertzuweisungsanweisung

Mit dieser Anweisung wird einer Variablen ein Wert zugewiesen, der in Form eines Ausdrucks, einer Konstanten, einer Zeichenkette oder einer anderen Variablen desselben Typs vorgegeben sein kann. Sie hat folgende Syntax:

```
Variable = Ausdruck          oder
LET Variable = Ausdruck
```

Das Wort LET kann bei der Eingabe weggelassen werden. In einer Anweisung (Zeile) koennen mehrere Zuweisungen auftreten, wenn sie durch Kommas getrennt werden. Das Gleichheitszeichen ist hierbei kein Vergleichsoperator, sondern ein Zuweisungsoperator, d.h., die Variable auf der linken Seite erhaelt den Wert des Ausdrucks auf der rechten Seite.

Beispiel:

```
50 N=0
60 LET N=N+1
70 LET A(N)=N
```

Die Anweisungen 50 bis 70 setzten das Feldelement A(1) auf 1. Durch Wiederholung der Anweisungen 60 und 70 wuerde jedes Feldelement auf den Wert seines Index gesetzt.

2.3.2. END/STOP-Anweisung

Die END/STOP-Anweisungen beenden die Programmabarbeitung. Sie bestehen nur aus den Woertern:

```
END
STOP
```

Die END-Anweisung bewirkt ein Abschliessen aller Dateien (close-request) und fuehrt zur Ausgabe von "READY" und Anzeige der Dialogbereitschaft.
Die STOP-Anweisung haelt den Programmablauf an. Es erfolgt die Ausgabe von "STOP AT nnnn" (nnnn - Zeilennummer der STOP-Anweisung). Das Programm kann mit Hilfe eines Programmablaufsteuerkommandos an dieser Stelle fortgesetzt werden (s. Abschn. 2.1.1).
Ist das Programm eine direkte Folge von Anweisungen und ist die letzte Anweisung des Programms auch tatsaechlich die letzte, die ausgefuehrt wird, kann das END entfallen. Es erscheint ebenfalls die Meldung "READY", jedoch bleiben offene Dateien offen.

2.3.3. Programmschleifen: FOR...NEXT-Anweisung

Ein wesentliches Element von Programmen sind Schleifenanweisungen, mit denen eine Gruppe von Anweisungen wiederholt ausgefuehrt werden kann. Im BASIC existiert hierfuer die FOR/NEXT-Anweisung.
Sie hat folgende Syntax:

```
FOR Variable_A = Ausdruck_1 TO Ausdruck_2 [STEP Ausdruck_3]
        .
        .
NEXT Variable_A
```

Die Variable_A ist dabei entweder eine reelle oder ganzzahlige Variable. Sie erhaelt zu Beginn den Wert vom Ausdruck_1. Die zwischen der FOR- und NEXT-Anweisung liegenden Anweisungen werden iterativ so lange ausgefuehrt, bis der Wert der Variablen_A den Wert des Ausdrucks_2 ueberschreitet. Falls kein STEP-Element (Schrittweite) angegeben wird, so wird der Wert der Variablen_A bei jedem Wiederholungsschritt um 1, ansonsten um den Wert des Ausdrucks_3, erhoeht.
Die Schleife wird durch die NEXT-Anweisung abgeschlossen, wobei die Variable hinter NEXT dieselbe sein muss, wie die in der zugehoerigen FOR-Anweisung. FOR/NEXT-Schleifen koennen auch ineinander verschachtelt werden, sie duerfen sich dabei aber nicht ueberlappen.

Beispiel:

```
10 REM..ERLAUBTE VERSCHACHTELUNG
20 DIM Y(7,16)
30 FOR A=1 TO 7 STEP 2
40   FOR B=1 TO 16 STEP 2
50     LET Y(A,B)=-1
```

```
60 NEXT B
70 NEXT A
```

2.3.4. Programmverzweigung: GOTO/ON...GOTO-Anweisungen

Zur Programmverzweigung existiert eine Einfach- und eine Mehrfachverzweigungsanweisung. Sie haben folgendes Format:

```
GOTO Anweisungsnummer
ON Ausdruck GOTO Anw.nr.1, ..., Anw.nr.N
```

Die GOTO-Anweisung ist ein unbedingter Sprung zu der angegebenen Programmzeile. Falls unter der angegebenen Zeilennummer keine ausfuehrbare Anweisung (z.B.: ein Kommentar) steht, so wird mit der naechstfolgenden ausfuehrbaren Anweisung fortgesetzt. Mit der GOTO-Anweisung darf nicht in eine Funktionsdefinition gesprungen werden (auch nicht heraus).
Die Mehrfachverzweigung ON/GOTO verzweigt entsprechend des Wertes des Ausdrucks, der auf ON folgt. Falls der Wert 1 ist, so wird bei der ersten hinter GOTO angegebenen Anweisungsnummer fortgesetzt, bei 2 bei der zweiten usw. Wenn der Ausdruckswert kleiner 1 oder groesser der Anzahl der angegebenen Anweisungsnummern ist, so wird die gesamte Anweisung ignoriert. Wenn der Wert des Ausdrucks nicht ganzzahlig ist, so wird er auf die naechste ganze Zahl gerundet.

Beispiel:

Das folgende Beispiel zeigt ein einfaches GOTO in Zeile 45, 55 und 65 sowie einen Verteilsprung in Zeile 30.

```
10 LET I=0
20 LET I=I+1
30 ON I GOTO 40,50,60,70
40 PRINT "DER WERT VON I IST 1"
45 GOTO 20
50 PRINT "DER WERT VON I IST 2"
55 GOTO 20
60 PRINT "DER WERT VON I IST 3"
65 GOTO 20
70 PRINT "DER WERT VON I ist 4"
75 END

>RUN
DER WERT VON I IST 1
DER WERT VON I IST 2
DER WERT VON I IST 3
DER WERT VON I IST 4
```

Wenn es gestartet wird, druckt das Programm den Wert von I fuer jedes ON...GOTO.

2.3.5. Unterprogramme: GOSUB...RETURN-Anweisungen

Haeufig sich wiederholende Programmteile koennen auch in BASIC in Form von Unterprogrammen realisiert werden. Dazu dienen die GOSUB- und die RETURN-Anweisung. Die Syntax der GOSUB-Anweisung ist analog zur GOTO-Anweisung:

```
GOSUB Anweisungsnummer
ON Ausdruck GOSUB Anw.nr.1, ..., Anw.nr.N
```

Die RETURN-Anweisung besteht nur aus dem Wort

```
RETURN
```

Sie bewirkt den Ruecksprung zu der Anweisung, die auf die GOSUB-Anweisung folgt.
Unterprogramme koennen verschachtelt werden, d.h., aus einem Unterprogramm kann ein weiteres aufgerufen werden. Das jeweilige RETURN fuehrt zum aufrufenden Unterprogramm zurueck, und zwar zur Anweisung, die auf GOSUB folgt.

Beispiel:

Das Beispiel zeigt einen Mehrfach-Unterprogrammsprung in Zeile 20. Das dritte ausgefuehrte Unterprogramm enthaelt einen geschachtelten Aufruf.

```
 10 FOR A=1 TO 3
 20 ON A GOSUB 50, 80, 110
 30 NEXT A
 40 END
 50 REM: Erstes Unterprogramm
 60 PRINT "Erster UP-Aufruf"
 70 RETURN
 80 REM: Zweites Unterprogramm
 90 PRINT "Zweiter UP-Aufruf"
100 RETURN
110 REM: Drittes Unterprogramm
120 REM: Es enthaelt einen geschachtelten Aufruf
130 PRINT "Dritter UP-Aufruf"
140 GOSUB 170
145 PRINT "Ende des dritten UP-Aufrufs"
150 RETURN
160 REM: Anweisung 150 bringt Ruecksprung nach 30
170 REM: Erste Anweisung in geschachtelten Aufruf
180 PRINT "      Geschachtelter Aufruf"
190 RETURN
200 REM: Anweisung 190 bringt Ruecksprung nach 145
> RUN
Erster UP-Aufruf
Zweiter UP-Aufruf
Dritter UP-Aufruf
      Geschachtelter Aufruf
Ende des dritten UP-Aufrufs
```

2.3.6. Bedingte Programmverzweigung: IF...THEN-Anweisungen

Eine Programmverzweigung in Abhaengigkeit von bestimmten Bedingungen kann mit Hilfe der IF/THEN-Anweisung realisiert werden. Sie hat folgende Syntax:

```
(a) IF Ausdruck THEN Anweisungsnummer [ELSE-Zweig]
(b) IF Ausdruck THEN Anweisung [ELSE-Zweig]
(c) IF Ausdruck THEN DO
                     Anweisungen
                        .
                        .
    DOEND [ELSE-Zweig]
```

Der optionale ELSE-Zweig kann wiederum eine der drei folgenden Formen haben:

```
(a) ELSE Anweisungsnummer
(b) ELSE Anweisung
(c) ELSE DO
         Anweisungen
            .
            .
         DOEND
```

Dabei muss der ELSE-Zweig eine eigene Anweisungsnummer erhalten.
Falls der nach IF folgende Ausdruck wahr ist (d.h., der Wert ist ungleich Null), so werden die zum THEN-Zweig gehoerenden Anweisungen ausgefuehrt. Falls der Wert des Ausdrucks gleich Null ist (d.h. falsch) und ein ELSE-Zweig vorhanden ist, so werden die dazu gehoerenden Anweisungen ausgefuehrt. Falls der ELSE-Zweig fehlt, so wird mit der nach der IF-Anweisung folgenden Anweisung fortgesetzt.
Sollen auf THEN bzw. auf ELSE eine Reihe von Anweisungen folgen, so werden diese in DO und DOEND eingeschlossen. Jede dieser Anweisungen traegt eine Anweisungsnummer. Eine FOR-Anweisung kann Bestandteil einer DO/DOEND-Gruppe sein. Dann muss aber auch das dazugehoerige NEXT darin enthalten sein.
IF-Anweisungen koennen ebenfalls verschachtelt werden. Sie muessen dann innerhalb einer DO/DOEND-Gruppe erscheinen. In einem solchen Fall gehoert jedes ELSE zu dem naechsten vorangegangenen IF der gleichen Verschachtelungstiefe. (Entweder befindet sich das entsprechende IF unmittelbar vor dem ELSE, oder vor dem ELSE steht ein DOEND. Dann ist fuer dieses DOEND das dazugehoerige DO zu suchen. Dort findet man das zu ELSE gehoerende IF.)

Beispiel:

Die folgenden Beispiele zeigen geschachtelte IF...THEN Anweisungen:

```
10 INPUT P, Q, R
15 PRINT
20 IF (P+10)=(Q+5) THEN DO
30   LET P=Q
```

```
 40   IF P>R THEN LET P=P-R
 50   ELSE LET P=P+R
 60 DOEND
 70 PRINT P, Q, R
>RUN
 ?20,25,40

 65          25          40


 10 INPUT A,B,C
 15 PRINT
 20 IF A>B THEN DO
 30   IF B>C THEN DO
 40     IF C=10 THEN DO
 50       LET A=A+1
 60       GOTO 200
 70     DOEND
 80     ELSE GOTO 220
 90   DOEND
100   ELSE DO
110     IF C=10 THEN LET B=C+A
120     ELSE LET C=B-A
130     GOTO 180
140   DOEND
150 DOEND
160 PRINT "A<60>B, A=",A
170 GOTO 230
180 PRINT "A>B,B<60>C, B=";B
190 GOTO 230
230 PRINT "A>B>C,C=10"
210 GOTO 230
220 PRINT "A>B>C,C<60>>10,C=";C
230 END

>RUN
?10,15,20,
A<B,A=10

>RUN
?15,5,10
A>B,B<C,B=25

>RUN
?20,15,5
A>B>C,C<>10,C=5
```

2.3.7. INPUT-Anweisung

Da BASIC in seiner Grundform eine dialogorientierte Sprache ist, erfolgen Ein-/Ausgaben standardmaessig ueber Tastatur und Bildschirm. Zur Realisierung von Dialogeingaben dient die INPUT-Anweisung. Sie hat folgende Syntax:

```
INPUT Variablenliste                                    oder
INPUT Zeichenkettenkonstante, Variablenliste
```

Die Variablen innerhalb der Liste muessen durch Kommas getrennt werden. Bei der ersten Form zeigt ein Fragezeichen an, dass Eingabewerte erwartet werden.
Bei der zweiten Form wird das Fragezeichen durch die angegebene Zeichenkette ersetzt (z.B.: INPUT "Bitte Wert fuer S: ",S).
Falls mehrere Eingabewerte erwartet werden, so sind diese ebenfalls durch Kommas getrennt einzugeben. Waehrend der Eingabe erfolgt eine Typpruefung.

Zur Eingabe vorgesehene Zeichenketten koennen in Anfuehrungszeichen eingeschlossen sein oder nicht. Wenn nicht, werden Leerzeichen vorn und hinten ignoriert. Das Element endet bei einem Komma bzw. am Ende der Eingabezeile.

Beispiele:

```
10 DIM C$[25]
20 INPUT A,B,C$
25 PRINT
30 X=A*B^2
40 PRINT C$;X
```

```
> RUN

?4,7,"X=A MAL B QUADRAT, X="
X=A MAL B QUADRAT, X=196
```

```
10 INPUT "EINGABEWERT FUER DEN RADIUS ",R
20 X=3,14*R^2
30 PRINT "FLAECHE VON X=",X

> RUN

EINGABEWERT FUER DEN RADIUS 25
FLAECHE VON X =        1962.5
```

2.3.8. PRINT-Anweisung

Zur Ausgabe von Daten oder Zeichenketten dient die PRINT-Anweisung. Sie hat folgendes Format:

```
PRINT   Ausgabeliste
```

Anstelle des Schluesselwortes PRINT kann als Abkuerzung auch der Doppelpunkt (z.B.: 100 : A,B;) angegeben werden.
Eine PRINT-Anweisung ohne Ausgabeliste bewirkt einen einfachen Zeilenvorschub.
Die Ausgabeliste kann aus numerischen oder Zeichenkettenausdruecken bestehen. Die Elemente der Liste muessen durch Komma oder Semikolon getrennt werden. Ein Komma veranlasst die Ausgabe zu Beginn der naechsten Tabulatorposition (implizit existieren 5 Tabulatorpositionen, alle 14 Zeichen), bei einem Semikolon erfolgt die Ausgabe dicht aufeinanderfolgend.
Die Voreinstellung fuer die Laenge einer Ausgabezeile (70) kann durch die SYSTEM-Anweisung "LINELEN" veraendert werden

(s. Abschn. 2.3.12.).
Sofern die Ausgabeliste n i c h t mit Komma oder Semikolon abgeschlossen ist, wird nach Ausfuehrung von PRINT ein Wagenruecklauf mit Zeilenschaltung ausgefuehrt. Andernfalls erfolgt die naechste PRINT Anweisung auf derselben Zeile.

Beispiel:

```
10 LET A=1,B=2,C=3,D=4,E=5
20 PRINT A,C*D,E-B*B
30 PRINT
40 PRINT "A/(B-C)=";A/(B-C),
50 PRINT "E+D=";E+D

> RUN

 1             12            1

A/(B-C)=-1 E+D=9
```

Numerische Daten koennen auch formatiert ausgegeben werden. Dazu dient die PRINT/USING-Anweisung. Sie hat folgendes Format:

```
PRINT USING "Zeichenkettenkonstante", numerischer_Ausdruck
```

Die Formatbeschreibung erfolgt durch den nach USING folgenden Zeichenkettenausdruck (z.B.: PRINT USING "###.DD",A). Durch ihn wird die Form festgelegt, in der das Datenelement ausgegeben wird. Das Komma in der PRINT/USING-Anweisung ist nur Trennzeichen. Es hat keinen Einfluss auf das Format wie in PRINT.
Wenn der Ausdruck und die Spezifikation nicht zueinander passen, wird das Programm mit einer Fehlermeldung abgebrochen. Wenn der Wert des numerischen Ausdrucks groesser ist, als mit dem Format ausgebbar, wird die Zahl ohne Formatierung ausgegeben. In diesem Fall werden zwei Sternchen vorgestellt und die Ausgabe erfolgt auf der naechsten Zeile. Das Programm wird fortgesetzt.
Bei Ausgabe einer Gleitkommazahl mittels einer INTEGER-Format-Spezifikation wird auf die naechste ganze Zahl gerundet.

Beispiel:

```
 10 A=12.46
 20 PRINT USING"+DDDD.DDD",A
 30 PRINT USING"###DPDD+",A
>RUN
 + 0012.460
    12.46+
```

Format-Zeichenketten:

Im folgenden werden die Format-Zeichen und ihre Funktion angegeben:

Zeichen	Ausgabe	Kommentar
#	Ziffer oder Leerzeichen oder "-"	Ausgabe eines Leerzeichens, sofern fuehrende oder nachfolgende Stelle Null. "-" Vorzeichen ergibt sich, wenn Zahl negativ.
D	Ziffer	Ausgabe einer Null, sofern fuehrende oder nachfolg. Stelle Null.
+	"+" oder "-"	Vorzeichen
-	"-" oder Leerzeichen	Leerzeichen, wenn die Zahl nicht negativ ist.
n	"n" oder Ziffer	Ziffern an den Stellen mit dem Zeichen "n" ueberschreiben dieses Zeichen. Es wird nur ein n vor der Ziffernfolge ungleich Null ausgegeben, an den anderen Stellen erscheint anstelle von n das Leerzeichen.
*	"*" oder Ziffer	Ausgabe eines "*" in fuehrenden oder nachfolgenden Stellen, sofern Wert Null, sonst Ausgabe der Ziffer.
P	"."	gibt die Position des Dezimalpunktes in einer Zahl an
.	"." oder Leerzeichen	Ausgabe eines ".", wenn es keine ganze Zahl ist, ansonsten Ausgabe Leerzeichen. Eine Zahl ist dann keine ganze Zahl, wenn Ziffern auf der rechten Seite nach dem Dezimalpunkt ausgegeben werden sollen.
,	"," oder Leerzeichen	Ausgabe Leerzeichen, wenn links vor dem Komma keine Ziffern stehen, sonst Komma.
^^^^	Esdd	Fuer s steht "+" oder "-", dd sind Ziffern; veranlasst, die Zahl in Exponentenform auszugeben.
^^^^^	Esddd	Wie bei "^^^^", nur mit dem Unterschied, dass hier der Exponent aus drei Ziffern besteht.

Anmerkungen:

1. "P" oder "." darf nur einmal vorkommen.

2. "," darf nicht rechts von einem "P", "," oder "^" stehen.

3. ^^^^ und ^^^^^ koennen nur auf der rechten Seite einer Format-Zeichenkette stehen.

4. Wenn "+" oder "-" auftreten, wird das "#" niemals als Vorzeichen verwendet.

5. Wenn kein "+" oder "-" auftritt, dann wird ein "#" als Vorzeichen in der Formatzeichenkette verwendet.

6. Wenn kein "P" oder "." auftritt, wird angenommen, dass der Dezimalpunkt rechts von der "#"- oder "D"- Zeichenkette steht.

•

Folgende Beispiele sollen die Kombinationsmoeglichkeiten verdeutlichen:

Formatzeichenkette	Zahl	Ausgabe
###.##	1234.567	234.56
###.DD	1.2	1.20
+###	12.3	+ 12
##-	-78	78-
+##.DD	0	+ 0.00
###P##	123	123.
¤##.DD	6.7	¤ 6.70
***.DD	12.3	*12.30
##,###,###.DD	12345678	12,345,678.00
#.###^^^^ (2 stelliger Exponent)	234.5	2.345E+02
##.DD^^^^^ (3 stelliger Exponent)	-34E+123	-34.00E+123

2.3.9. READ/DATA/RESTORE-Anweisungen

Daten koennen nicht nur durch Eingaben Variablen zugewiesen werden. Es besteht auch die Moeglichkeit, innerhalb des Programms Daten zu vereinbaren und diese dann Variablen zuzuweisen. Dazu dienen die READ/DATA/RESTORE-Anweisungen. Sie haben folgende Syntax:

```
DATA Konstante_1, ..., Konstante_N
READ Variablenliste
RESTORE [Zeilennummer]
ON Ausdruck RESTORE Zeilennr.1, ..., Zeilennr.N
```

Mit der READ-Anweisung werden Daten, die in DATA- Anweisungen angegeben sind, den Variablen zugewiesen, die die READ-Anweisung auffuehrt. Es sind mehrere DATA-Anweisungen in einem Programm moeglich. Alle in ihnen angegebenen Konstanten bilden eine gemeinsame Datenliste. In der Reihenfolge aufsteigender Anweisungsnummern werden sie hintereinander abgelegt. Der Zugriff erfolgt ueber einen internen Zeiger, der immer die naechste zuzuweisende Konstante adressiert (steht zu Beginn auf der ersten Konstanten der ersten DATA-Anweisung). Die Zuweisung erfolgt in der Reihenfolge der Konstantendefinitionen.
Die RESTORE-Anweisung kann vor einer READ-Anweisung zur Veraenderung dieses Zeigerwertes benutzt werden. Falls keine Zeilennummer angegeben ist, so wird der Zeiger wieder auf die erste DATA-Anweisung positioniert, ansonsten wird er auf die erste Konstante der DATA-Anweisung auf der angegebenen Zeilennummer gesetzt. Analog zur ON/GOTO-Anweisung kann dabei eine Mehrfachverzweigung fuer die RESTORE-Anweisung benutzt werden.
Einer numerischen Variablen kann ueber die DATA-Liste nur ein Zahlenwert zugeordnet werden. Bei einer Zeichenkettenvariablen kann das DATA-Element beliebig sein.

Beispiel:

Hier werden dieselben Konstanten nochmal in einen zweiten Satz von Variablen gelesen.

```
  5 DIM A%[3],C%[3], D%[3],E%[3],B%[3]
 10 DATA 3,5,7
 20 READ A,B,C
 30 READ A%,B%
 40 DATA ABC,DEF
 50 RESTORE 40
 60 READ C%,D%
 70 RESTORE
 80 READ D,E,F,E%
 90 PRINT A,B,C
100 PRINT A%+B%,C%+D%
110 PRINT D,E,F,E%

> RUN
  3        5        7
ABCDEF     ABCDEF
  3        5        7        ABC
```

2.3.10. Kommentare: REM-Anweisung

Gerade bei BASIC-Programmen ist eine Kommentierung der Aktionen innerhalb eines Programms besonders wichtig. Dazu dient die REM-Anweisung.

```
REM beliebiger_Text
```

Alle mit dem Schluesselwort REM beginnenden Zeilen werden als Kommentarzeilen betrachtet und haben keinen Einfluss auf den Programmablauf. Kommentare koennen auch hinter jeder Anweisung (ausser nach DATA-Anweisungen) stehen. Sie muessen dann mit dem Zeichen "\" beginnen. (Dazu ist die Eingabe von zwei Backslashzeichen notwendig.)

Beispiel:

```
100 REM:... BEISPIEL FUER KOMMENTAR
110 PRINT A*3.4  \ Hier kann ein Kommentar stehen
```

2.3.11. RANDOMIZE-Anweisung

Im BASIC-Interpreter existiert die Funktion RND zur Erzeugung von Zufallszahlen. Die RANDOMIZE-Anweisung, mit dem Format:

```
RANDOMIZE
```

startet die Funktion mit einem neuen Anfangswert. Dadurch kann verhindert werden, dass in Programmablaeufen stets die gleichen "Pseudozufallszahlen" generiert werden.

2.3.12. SYSTEM-Anweisungen

Zur Spezifizierung von bestimmten Systemfunktionen und Parametern dient die SYSTEM-Anweisung. Sie hat folgende Syntax:

```
SYSTEM Operation
```

Fuer Operation koennen dabei folgende Zeichenketten mit moeglichen Parametern stehen:

"WARNOFF"	Unterdrueckung aller Warnmeldungen
"WARNON"	Ausgabe aller Warnmeldungen (ist implizit gesetzt)
"INDENTOFF"	stellt das automatische Einruecken beim Auslisten ab
"INDENTON"	automatisches Einruecken beim Auslisten (ist implizit gesetzt)
"ASINOFF"	analog zu "INDENTOFF" fuer ASAVE-Dateien (ist implizit gesetzt)

"ASINON" analog "INDENTON" fuer ASAVE-Dateien

"LINELEN" n Setzen der Zeilenlaenge fuer Ausgabezeile (Wert zwischen 1 und 255; implizit: 70)

"AUTOCROFF" Ausschalten des automatischen Zeilenvorschubs

"AUTOCRON" Einschalten des automatischen Zeilenvorschubs

Beispiel:

```
SYSTEM "LINELEN",132
```

Die Druckzeilenlaenge wird auf 132 Zeichen festgesetzt.

2.4. Zeichenketten und Felder

Eine Zeichenkette besteht aus einer in Anfuehrungszeichen eingeschlossenen Folge von ASCII-Zeichen (maximal 255 Zeichen lang), ausser den Zeichen " und < . Diese Zeichen und die nicht druckbaren Zeichen koennen durch ihr numerisches Aequivalent, eingeschlossen in spitze Klammern, dargestellt werden. Die Notation <Dezimalzahl> steht dann innerhalb einer Zeichenkette fuer ein Zeichen (z.B.: <34> fuer "; <60> fuer < ; <9> fuer Tabulator und <13> fuer Zeilenvorschub; "NR<9>NAME<9>ABT<13>").
Zeichenkettenvariablen enthalten nach ihrer Vereinbarung durch eine DIM-Anweisung die leere Zeichenkette.

2.4.1. Teilzeichenketten

Es besteht die Moeglichkeit der Verarbeitung von Teilzeichenketten. Dazu wird folgende Notation verwendet:

Bei einfachen Zeichenkettenvariablen:

Zeichenkettenname(erste_Zeichenposition)

entspricht der Teilzeichenkette ab dem spezifizierten Zeichen (Position)

Zeichenkettenname(erste_Zeichenpos.,letzte_Zeichenpos.)

entspricht der Teilzeichenkette, die aus den zwischen den angegebenen Positionen liegenden Zeichen gebildet wird.

(z.B.: wenn A$="VORNAME NAME ABT" ist, dann sind A$(9)="NAME ABT" und A$(9,12)="NAME")

Bei Zeichenkettenfeldern:

Feldname(Elementnummer)

entspricht dem angegebenen Element des Zeichenkettenfeldes

Feldname(Elementnr.,erste_Zeichenposition)
Feldname(Elementnr.,erste_Zeichenpos.,letzte_Zeichenp.)

entspricht der spezifizierten Teilzeichenkette des angegebenen Elements des Feldes

(z.B.: wenn B¤ ein Feld von 10 Zeichenketten der Laenge 25 ist, dann sind:

B¤(6)	die gesamte 6.Zeichenkette
B¤(3,11)	von der 3.Zeichenkette die letzten 15 Zeichen
B¤(7,1,20)	von der 7.Zeichenkette nur die ersten 20 Zeichen)

Anstelle der runden Klammern koennen zur Kennzeichnung der Teilelemente auch eckige Klammern verwendet werden.

2.4.2. Speichervereinbarung: DIM-Anweisung

Felder von numerischen Groessen werden ueber die DIM-Anweisung vereinbart. Dies ist notwendig, wenn sie mehr als 10 Elemente (oder 10 x 10 Elemente bei zweidimensionalen Feldern) enthalten. Die DIM-Anweisung hat folgende Syntax:

DIM Variable(ganze_Zahl), Variable(ganze_Zahl), ...
DIM Variable(ganze_Zahl,ganze_Zahl), ...

Die erste Form gilt fuer eindimensionale und die zweite Form fuer zweidimensionale Felder. Der Typ (ganzzahlig oder reell) wird durch den Variablennamen fixiert. Die Zaehlung der Indizes beginnt bei 1 (z.B.: besteht das Feld A(25) aus 25 reellen Zahlen und K%(5,10) ist ein Feld von ganzen Zahlen der Dimension 5 Zeilen und 10 Spalten). Nach der Feldvereinbarung besitzen alle Elemente den Wert Null.
Im Gegensatz zu numerischen Groessen muessen jede einfache Zeichenkettenvariable und jedes Zeichenkettenfeld (maximal eindimensional) ueber eine DIM-Anweisung vereinbart werden. Die erste Form der DIM-Anweisung vereinbart dann eine einfache Zeichenkettenvariable (der Parameter gibt die maximale Laenge der Zeichenkette an) und die zweite Form ein eindimensionales Zeichenkettenfeld, wobei der erste Parameter die Anzahl der Zeichenketten und der zweite Parameter die maximale Laenge der Zeichenketten definiert.

Beispiel:

DIM A¤(15)	A¤ ist eine Zeichenkette aus max. 15 Zeichen
DIM F¤(5,10)	F¤ ist ein Feld aus 5 Zeichenketten der Laenge 10

Fuer Zeichenketten existiert ein Verkettungsoperator (+). Diese Operation kann in Verbindung mit der Teilzeichenkettenbildung und den zur Verfuegung stehenden Zeichenkettenfunktionen zu vielfaeltigen Zeichenkettenmanipulationen angewendet werden.
(z.B.: wenn N¤ die Zeichenkette "PETER MEIER" darstellt, so druckt die Anweisung: PRINT "HALLO"+N¤(1,5) die Zeichenkette "HALLO PETER" aus)

Daneben koennen Zeichenketten auch mit den Vergleichsoperatoren zur Bildung von logischen Ausdruecken verknuepft werden. Dabei sind zwei Zeichenketten nur dann gleich, wenn sie gleich lang sind und in jedem Zeichen uebereinstimmen. Eine Zeichenkette ist kleiner als eine andere, wenn das erste verschiedene Zeichen einen kleineren numerischen Wert hat als das entsprechende Zeichen der anderen Zeichenkette.

2.4.3. Zeichenketteneingabe: LINPUT-Anweisung

Speziell zur Eingabe von Zeichenketten existiert noch die LINPUT-Anweisung. Sie hat folgende Form:

```
LINPUT Zeichenkettenvariable                          oder
LINPUT Zeichenkettenkonstante,Zeichenkettenvariable
```

Im Unterschied zur INPUT-Anweisung werden dabei alle eingegebenen Zeichen (einschliesslich Anfuehrungszeichen, Kommas und Leerzeichen) der Zeichenkettenvariable zugewiesen. Man erspart sich so die Anfuehrungszeichen fuer die Kennzeichnung der Zeichenkette. Die Eingabe wird durch das Wagenruecklaufzeichen (CR) abgeschlossen.
Falls die Anzahl der eingegebenen Zeichen groesser als die Dimension der Zeichenkettenvariable ist, so wird der Rest weggeschnitten.

2.5. Funktionen

Im BASIC-Interpreter sind eine Reihe von Funktionen fest integriert. Daneben kann der Anwender fuer seine Problemloesungen auch eigene Funktionen definieren(s. 2.5.3.).

2.5.1. Numerische Funktionen

ABS(Ausdruck)	Liefert den Absolutbetrag des Ausdruckswertes
ATN(Ausdruck)	Liefert den Arcustangens des Ausdruckswertes (Ergebnis liegt im Bereich von -pi/2 bis +pi/2)
COS(Ausdruck)	Kosinusfunktion
EXP(Ausdruck)	Liefert den Wert e hoch Ausdruckswert (e=2,718281828)

INT(Ausdruck)	Liefert die groesste ganze Zahl, die kleiner oder gleich dem Wert des Ausdrucks ist
LOG(Ausdruck)	Liefert den natuerlichen Logarithmus des Ausdrucks (der Wert des Ausdrucks muss groesser Null sein)
RND	Liefert eine Pseudozufallszahl zwischen 0 und 1
SGN(Ausdruck)	Liefert das Vorzeichen des Ausdruckswertes: 1, wenn Wert > 0 0, " " = 0 -1, " " < 0
SIN(Ausdruck)	Sinusfunktion
SQR(Ausdruck)	Liefert die Quadratwurzel des Ausdruckswertes (Wert des Ausdrucks muss groesser gleich Null sein)
TAN(Ausdruck)	Tangensfunktion

2.5.2. Zeichenkettenfunktionen

Neben den numerischen Funktionen existiert noch ein Satz von fest eingebauten Zeichenkettenfunktionen. Dabei kennzeichnet die Endung $ beim Funktionsnamen, dass ein Zeichenkettenwert geliefert wird.

CHR$(ganzzahliger_Ausdruck)
Liefert das ASCII-Zeichen, das den Wert des Ausdrucks (0-255) besitzt (z.B.: ist CHR$(66) gleich dem Zeichen "B").

ASC(Zeichenkettenausdruck)
Liefert den numerischen Wert des ersten Zeichens des Ausdrucks (z.B.: ist ASC("A") gleich 65).

LEN(Zeichenkettenausdruck)
Liefert die Laenge der als Parameter uebergebenen Zeichenkette.

POS(Zeichenkette_A,Zeichenkette_B)
Liefert die Startposition der Teilzeichenkette_B in der Zeichenkette_A (z.B.: ist POS("ABCDEFG","EF") gleich 5). Falls die zweite Zeichenkette keine Teilkette der ersten ist, so wird der Wert Null geliefert.

VAL(Zeichenkettenausdruck)
Liefert den numerischen Wert, der durch die Zahlen in der Zeichenkette dargestellt wird (z.B.: ist VAL("123AB") gleich 123). Dabei kann auch die E-Notation benutzt werden (z.B.: wenn A¤="ABC3EXY" ist, so liefert VAL(A¤(4,5)+"2") den Wert 300).

STR¤(numerischer_Ausdruck)
Bildet das Zeichenkettenaequivalent zum numerischen Wert (z.B.: ist STR¤(123) gleich der Zeichenkette "123").

LEFT¤(Zeichenkettenausdruck,ganzzahliger_Ausdruck)
Liefert die linke Teilzeichenkette des ersten Parameters. Die Laenge wird durch den Wert des zweiten Parameters spezifiziert.
(z.B.: ist LEFT¤("PETER MEIER",5) gleich der Zeichenkette "PETER")

RIGHT¤(Zeichenkettenausdruck,ganzzahliger_Ausdruck)
Liefert die rechte Teilzeichenkette, beginnend mit dem n-ten Zeichen, das durch den ganzzahligen Ausdruck festgelegt wird.
(z.B.: ist RIGHT¤("ABCDEFGHIJ",4) gleich der Zeichenkette "DEFGHIJ")

SEG¤(Zeichenkettenausdr.,ganz._Ausdruck,ganz._Ausdruck)
Liefert eine Teilzeichenkette des ersten Parameters. Die Position des ersten Zeichens wird durch den Wert des zweiten Parameters und die des letzten Zeichens durch den Wert des dritten Parameters spezifiziert.
(z.B.: ist SEG¤("1020 BERLIN STR xyz",6,11) gleich der Zeichenkette "BERLIN")

2.5.3. Anwenderfunktionen

Anwenderfunktionen werden innerhalb eines Programms durch eine DEF-Anweisung definiert und koennen wie eingebaute Funktionen verwendet werden. Die Funktionsnamen bestehen aus den Buchstaben "FN" gefolgt von einem normalen Variablennamen. Dabei kennzeichnet der Variablenname den Funktionstyp, d.h., die Funktion liefert einen ganzzahligen Wert, wenn der Name mit "%" endet, eine Zeichenkette, wenn er mit "¤" endet, und sonst einen reellen Wert.
Die Parameter in einer Funktionsdefinition sind formale Parameter. Bei Aufruf der Funktion werden sie durch die aktuellen Parameter des Funktionsaufrufs, entsprechend ihrer Position in der Parameterliste, ersetzt. Bei der

Definition von Anwenderfunktionen wird zwischen ein- und mehrzeiligen Funktionen unterschieden.

- einzeilige Funktionsdefinitionen:

```
DEF Fkt.name (formale_Parameterliste) = Ausdruck  oder
DEF Fkt.name = Ausdruck
```

Beispiel:

Bei Aufruf der Funktion:

```
30  DEF FNB%(A%,X2%)=A%*X2%÷(A%+X2%)
```

die einen ganzzahligen Wert liefert, koennen die aktuellen Parameter Variablen sein:

```
500 LET X%=4,Y%=2
510 PRINT FNB%(X%,Y%)
>RUN
 14
```

oder Zahlenwertausdruecke:

```
520 PRINT FNB%(4,2)
>RUN
 14
```

- mehrzeilige Funktionsdefinitionen:

```
DEF Fkt.name (formale_Parameterliste)          oder
DEF Fkt.name
        Anweisungen
          .
          .
RETURN Ausdruck
FNEND
```

Der Typ des Ausdrucks muss mit dem Funktionstyp (durch Namen festgelegt) vereinbar sein. Die formalen Parameter sind lokale Groessen innerhalb der Funktionsdefinition, d.h., sie stehen in keiner Beziehung zu Programmvariablen gleichen Namens.
Bei einer mehrzeiligen Funktionsdefinition duerfen im Funktionskoerper keine weiteren Funktionsdefinitionen auftreten. Aufrufe anderer Funktionen (auch rekursiv der eigenen Funktion) sind dagegen erlaubt. Der Funktionskoerper muss eine abgeschlossene Einheit bilden, d.h., Schleifen muessen vollstaendig im Koerper liegen und es duerfen keine Sprueenge aus dem Funktionskoerper heraus auftreten.

Beispiel:

Die folgende mehrzeilige Funktion liefert einen Zeichenkettenwert, der das Spiegelbild des Zeichenkettenwertes ist, der ueber den aktuellen Parameter eingegeben wird:

```
10 DEF FNR¤(A¤)
20 REM...FNR¤ LIEFERT DAS GESPIEGELTE A¤
30 IF LEN(A¤)<=1 THEN RETURN A¤
40 RETURN FNR¤(A¤[2]+A¤[1,1]
50 FNEND
```

Bei Aufruf dieser Funktion kann der aktuelle Parameter eine Zeichenketten-Variable sein:

```
60 DIM X¤[5]
70 X¤="12345"
80 PRINT "FNR¤ RETURNS:";FNR¤(X¤)
>RUN
 FNR¤ RETURNS:54321
```

2.6. Dateiarbeit

Neben der standardmaessigen Zuordnung der Ein-/Ausgabe zu Tastatur und Bildschirm koennen innerhalb von BASIC-Programmen die Ein-/Ausgabestroeme auch anderen peripheren Geraeten zugeordnet werden. Entsprechende Treiberprogramme muessen dazu im Betriebssystem vorhanden sein (z.B.: fuer die Druckerausgabe). Sie muessen vor Start des BASIC-Interpreters aktiviert werden, wenn sie in BASIC-Programmen verwendet werden sollen.

```
%ACTIVATE ¤PRINTER
%BASIC
   .
   .
```

2.6.1. Zuweisung von Ein-/Ausgabekanaelen: FILE-Anweisung

Der entsprechende Ein-/Ausgabekanal wird im Programm durch eine FILE-Anweisung eroeffnet und einer logischen Geraetenummer (zwischen 1 und 15) zugeordnet. Bei Ausgaben zum Beispiel wird einfach vor der Ausgabeliste die Geraetenummer gefolgt von einem Semikolon angegeben.

```
10  FILE #1;"¤PRINTER"
   .
   .
100 PRINT #1;"DIESE AUSGABE GEHT UEBBER DEN DRUCKER"
```

2.6.2. Oeffnen von Dateien: FILE-Anweisung

Ebenso koennen externe Daten als Inhalt von Dateien, die auf Disketten abgelegt sind, verarbeitet werden. Dazu werden die Moeglichkeiten des Betriebssystems ausgenutzt, die es gestatten, dass Standard-Ein-/Ausgabeanforderungen des DOS-Teils durch BASIC-Anweisungen ausgefuehrt werden. Es koennen sowohl Dateien vom Typ ASCII (fuer Zeichenketten) als auch binaere Dateien (fuer Daten) manipuliert werden.

Um vom Programm aus auf eine Datei zugreifen zu koennen,

muss sie offen sein. Fuer jede offene Datei wird ein Puffer benoetigt. Das NEW-Kommando (s. Abschn. 2.1.2.) wird zur Anlage der Puffer benutzt.
Fuer jede zu oeffnende Datei wird eine logische Verbindung zwischen der in den Zugriffsanweisungen benutzten Dateinummer und dem Dateinamen hergestellt. Die Dateinummer ist eine ganze Zahl zwischen 1 und 15.
Die Verbindung zwischen Dateiname und Dateinummer wird durch die FILE-Anweisung realisiert. FILE bewirkt, dass einem Dateinamen eine Dateinummer zugeordnet wird. Wenn der Dateinummer bereits eine Datei zugeordnet ist, wird diese geschlossen.

Die FILE-Anweisung hat folgendes Format:

FILE #n;Namen_Optionenkette ,Returnvariable

Die Namen_Optionenkette ist eine Zeichenkette, die den Dateinamen und moegliche, durch Semikolon getrennte, Optionen enthaelt.

Moegliche Optionen:

REC=Konstante	Legt Satzlaenge der Datei fest (impl.: 128). Falls die Datei bereits mit einer anderen Satzlaenge existiert, so wird diese auf den angegebenen Wert veraendert.
RL=Konstante	Legt bei einer neu anzulegenden Datei die Satzlaenge fest (impl.: 128).
RND	Datei ist fuer wahlfreien Zugriff vorgesehen.
ACC=Option	Legt den Typ der Eroeffnungsanforderung fest. Die Option kann sein:
IN	zur Eingabe, d.h., Datei muss bereits existieren
OUT	zur Ausgabe, falls Datei bereits existiert, so wird der Inhalt geloescht
NEW	zur Ausgabe, falls Datei bereits existiert, so wird Fehler 1 generiert
UPD	zur Aktualisierung einer bereits existierenden Datei (Dateizeiger wird auf den Anfang gesetzt, voreingestellt)
APP	Zeiger wird ans Ende einer bereits existierenden Datei gesetzt.

Die Fehlerbehandlung waehrend der Dateiarbeit kann ueber zwei Wege erfolgen.
Erstens: Es werden auftretende Fehler normal ueber den Bildschirm angezeigt. Sie koennen aber auch ueber eine TRAP-Anweisung (s. Abschn. 2.8.) abgefangen werden.

Zweitens: Es werden keine Fehlermeldungen generiert, wenn innerhalb der FILE-Anweisung eine Returnvariable angegeben

wurde. Die Auswertung der Returnvariablen muss dann selbstaendig erfolgen. Sie kann folgende Werte annehmen:

0	Operation fehlerfrei ausgefuehrt
1	Datei existiert bereits (bei ACC=NEW)
2	Datei existiert nicht (bei ACC=IN)
3	unzulaessiger Dateiname
4	unerlaubte Operation
5	unzulaessige Satzlaenge (bei RL= oder REC=)
6	nicht genuegend Pufferspeicher vorhanden

Beispiel:

```
FILE #1;"XFILE;ACC=IN"
FILE #2;"YFILE;ACC=APP;RND",C%
FILE #3;"ZFILE"
```

XFILE wird geoeffnet und muss schon existieren. YFILE wird fuer wahlfreien Zugriff geoeffnet, der Zeiger wird auf das Ende der Datei gesetzt und C% enthaelt den Returnzustand nach Ausfuehrung der Anweisung. ZFILE wird geoeffnet, wobei der Zeiger auf den Anfang der Datei gesetzt wird. Falls ZFILE noch nicht existiert, wird sie angelegt.

2.6.3. Schliessen von Dateien: CLOSE-Anweisung

Zum Ende eines Programms werden alle Dateien automatisch abgeschlossen. Waehrend der Programmausfuehrung kann dies durch eine CLOSE-Anweisung auch direkt fuer die spezifizierten Dateien (ueber ihre logischen Nummern) erfolgen.

```
CLOSE #n,#m,...
```

Falls kein Parameter angegeben wird, so werden alle eroeffneten Dateien abgeschlossen. Die CLOSE-Anweisung findet Verwendung, wenn der Puffer-Speicherbereich fuer andere Dateien benoetigt wird.

2.6.4. Loeschen von Dateien: ERASE-Anweisung

Mit der ERASE-Anweisung:

```
ERASE "Dateiname"[,Returnvariable]
```

wird die Datei mit dem angegebenen Namen gestrichen. Der Dateiname ist ein Zeichenkettenausdruck. Die Returnvariable enthaelt im Anschluss an die Ausfuehrung der ERASE-Anweisung ein Ergebnis. Sie kann dabei folgende Werte annehmen:

0	Operation fehlerfrei ausgefuehrt
1	Datei konnte nicht gestrichen werden
2	Benutzer darf diese Datei nicht streichen
3	angegebene Datei existiert nicht

2.6.5. Dateizugriff

Fuer den Dateizugriff gibt es zwei Moeglichkeiten. Einmal den sequentiellen Zugriff, dabei erfolgt die Adressierung der zu lesenden oder zu schreibenden Elemente ueber einen internen Dateizeiger, der nach jedem Zugriff automatisch auf das naechste Element zeigt. Dieser Zeiger kann ueber die SPACE- und RESTORE-Anweisungen positioniert werden (s. Abschn. 2.6.6./2.6.7.)
Bei der zweiten Zugriffsart, dem wahlfreien Zugriff, kann zusaetzlich die Satznummer spezifiziert werden, ab der der Zugriff beginnt. Beide Zugriffsarten koennen auch gemischt auftreten.
Wird wahlfreier Zugriff beabsichtigt, muss das in der FILE-Anweisung fuer die Datei angegeben werden (s. Abschn. 2.6.2.).

2.6.5.1. Sequentielles READ/INPUT/LINPUT von Dateien

Mit den Anweisungen READ/INPUT/LINPUT fuer sequentiellen Zugriff werden Daten von Dateien in Zahlwert- und Zeichenkettenvariablen gelesen. Das erste Element, das gelesen wird, ist das, welches auf die augenblickliche Zeigerposition folgt, d.h. direkt im Anschluss an den letzten Zugriff.
Wie bei sequentiellem PRINT (s. Abschn. 2.6.5.2.) werden Satzgrenzen ignoriert und die Liste der einzulesenden Datenelemente kann mitten in einem Satz anfangen und mitten in einem anderen aufhoeren.
Die sequentiellen Datei-Lesezugriffe haben folgendes Format:

```
READ #n;Variablenliste
INPUT #n;Variablenliste
LINPUT #n;Zeichenkettenvariable
```

Die READ #-Anweisung fuer sequentiellen Dateizugriff bringt binaere Daten von der angegebenen Datei in die Variablen der Variablenliste. Die Zahl der uebertragenen Bytes ist genau die, die zur Fuellung der Variablen benoetigt wird. Es wird keinerlei Umwandlung oder Typkontrolle durchgefuehrt. Die Zahl der uebertragenen Bytes haengt vom Datentyp ab:

Datentyp	Zahl der Bytes auf der Datei
Integer	2
Real	8 (bei BCD-Arithmetik-BASIC)
Zeichenkette	aktuelle Zahl der gelieferten Zeichen: logische (physische Groesse bei READ) Groesse der Zeichenkette, wenn keine Teilkette oder angegebene Groesse bei Teilkette

Wird eine Zeichenkette von einer binaeren Datei gelesen,

haengt die Zahl der gelesenen Zeichen von der Form der Variablen ab. Wenn z.B. A$ eine einfache Zeichenkettenvariable ist, dann:

```
READ #1;A$          liest die physische Laenge von A$
READ #1;A$(I)       liest die physische Laenge der Teilkette, die bei I anfaengt
READ #1;A$(I,J)     liest J-I+1 Zeichen in die Teilkette, die bei I anfaengt
```

Die INPUT #-Anweisung fuer sequentielle Datei-Eingabe liest ASCII-Daten von einer Datei in derselben Weise wie durch die INPUT-Anweisung Daten ueber die Tastatur eingegeben werden. Jedes Datenelement aus der angegebenen Datei wird in eine Variable der Variablenliste gelesen, das erste Element in die erste Variable, das zweite in die zweite, und so weiter. Die einzulesenden Datenelemente muessen durch Komma getrennt sein.

Das Ziel eines Zeichenkettenwertes muss eine Zeichenkettenvariable sein. Das Ziel eines Zahlenwertes muss eine Zahlenwertvariable sein. Andernfalls tritt ein Fehler auf. Wenn der einzulesende Zahlenwert nicht denselben Typ hat wie die Variable, wird Typumwandlung durchgefuehrt.

Bei Eingabe von Zeichenketten mit INPUT # werden ueberfluessige Zeichen ueberlesen, falls die Zeichenkettenvariable nicht lang genug ist, das ganze Datenelement aufzunehmen.

Die LINPUT#-Anweisung liest eine Zeichenkettenvariable bis zu einem Wagenruecklaufzeichen (CR).

Wenn eine EOF-Bedingung auftritt, bleiben die Variablen unveraendert und die EOF-Funktion (s. Abschn. 2.6.8.) bekommt den Wert "wahr".

2.6.5.2. Sequentielles PRINT und WRITE auf Dateien

Die sequentiellen PRINT-und WRITE-Anweisungen fuer Dateien schreiben Datenelemente auf eine Datei, beginnend bei der augenblicklichen Zeigerposition. Die Datenelemente koennen Zahlenwert- oder Zeichenkettenausdruecke sein.

Die sequentiellen Ausgabeanweisungen auf Dateien haben folgende Form:

```
PRINT #n[;Ausgabeliste]
WRITE #n;Ausgabeliste
```

Die Ausgabeliste ist eine Folge von Zahlwert- und/oder Zeichenkettenausdruecken. Die Regeln fuer den Aufbau einer solchen Liste sind dieselben wie die fuer die PRINT-Anweisung in Abschn. 2.3.8. beschriebenen.
Wird die Ausgabeliste bei der PRINT#-Anweisung weggelassen,

wird ein RETURN auf die Datei geschrieben, sofern es sich um eine ASCII-Datei handelt. Ansonsten wird die Anweisung ignoriert (Zeilenvorschub wie bei einer PRINT-Anweisung).

Jedes Element der Ausgabeliste wird in der Reihenfolge auf die Datei geschrieben, die durch die Anweisung vorgegeben ist. Die Elemente werden ab der Position geschrieben, auf die der Zeiger gerade weist. Daten, die sich schon dort befinden, werden ueberschrieben.
Satzgrenzen werden ignoriert. Eine sequentielle Ausgabe kann mitten in einem Satz beginnen und mitten in einem anderen aufhoeren.
Die durch PRINT# geschriebenen Daten sind genau die ASCII-Zeichen, die bei Ausfuehrung eines gleich aufgebauten PRINT zur Ausgabe ueber Bildschirm oder Drucker kommen.
Es ist zu beachten, dass mit einer INPUT#-Anweisung nicht diejenigen Daten zurueckgelesen werden koennen, die zuvor mit einer PRINT#-Anweisung auf eine Datei geschrieben wurden, wenn nicht Kommas zwischen die Elemente gesetzt und Zeichenketten mit Anfuehrungszeichen geklammert wurden.

Eine sequentielle WRITE#-Anweisung bringt binaere Daten aus Elementen der Ausgabeliste auf die angegebene Datei.
Der Umfang in (Bytes) der uebertragenen Daten haengt von der Groesse der Dateielemente ab (s. Datentyptabelle in Abschn. 2.6.5.1.). Es wird keine Typumwandlung durchgefuehrt und es ist nicht moeglich, Struktur oder Typ der Daten auf der Datei aus der Information allein zu ermitteln. So geschriebene Daten werden mit der READ#-Anweisung wieder eingelesen.

2.6.5.3. READ/INPUT/LINPUT von wahlfreien Dateien

Die Anweisungen READ/INPUT/LINPUT fuer Dateien mit wahlfreiem Zugriff lesen Datenwerte ab einem bestimmten Satz einer bestimmten Datei und weisen diese Werte Variablen zu.

Die Anweisungen haben folgende Form:

```
READ #n,Satznummer;Variablenliste
INPUT #n,Satznummer;Variablenliste
LINPUT #n,Satznummer;Zeichenkettenvariable
```

Die Datei- und Satznummer sind ganzzahlige Ausdruecke. Uebersteigt die Satznummer die Anzahl der Saetze, aus der die Datei besteht, so erscheint die EOF-Bedingung.

2.6.5.4. PRINT und WRITE auf wahlfreie Dateien

Die Anweisungen PRINT und WRITE fuer Dateien mit wahlfreiem Zugriff schreiben eine Liste von Datenelementen auf eine angegebene Datei. Die Ausgabe beginnt bei einem in der Anweisung angegeben Satz. Daten, die vor oder hinter dem angegebenen Gebiet liegen, werden nicht veraendert.

Die Anweisungen haben folgende Form:

```
PRINT #n,Satznummer;Ausgabeliste
WRITE #n,Satznummer;Ausgabeliste
```

Die Ausgabeliste hat dasselbe Format wie bei PRINT# auf sequentielle Dateien. Hier darf sie jedoch nicht entfallen.

PRINT# und WRITE# fuer wahlfreie Dateien positionieren den Zeiger auf den Anfang des angegebenen Satzes und schreiben dann den Inhalt der Ausgabeliste. Der erste Satz der Datei hat die Nummer 0.
PRINT#-Anweisungen fuer sequentielle und wahlfreie Dateien koennen benutzt werden, um auf dieselbe Datei zu schreiben. Dasselbe gilt fuer WRITE#. Ein sequentielles PRINT#, das auf ein wahlfreies PRINT# folgt, wird seine Daten in unmittelbarem Anschluss an die vorhergegangenen Daten setzen.

Beispiel fuer wahlfreie Dateien:

Das folgende Beispiel arbeitet mit wahlfreiem Dateizugriff. Zwei Dateien werden aufgebaut und Daten in sie hineingeschrieben. Einzelne Zeilen beider Dateien werden dann untereinander ausgetauscht, schliesslich die Originaldaten wiederhergestellt und verglichen.
Die Nachricht "TEST SUCCESSFULL" wird ausgegeben, wenn das Pruefergebnis "Gleichheit beider Dateien" war. Anschliessend werden beide Dateien geschlossen und geloescht.

```
 10 FILE #1;"P1"
 20 FILE #2;"P2"
 30 FOR I=1 TO 100
 40   PRINT #1;I,I,I
 50 NEXT I
 60 DIM A%(128),B%(128)
 70 FOR I=0 TO 24
 80   READ #1,I;A%
 90   WRITE #2;A%
100 NEXT I
110 FOR I=1 TO 2
120   FOR J=0 TO 10
130     READ#1,10-J;A%
140     READ#1,10+J;B%
150     WRITE#1,10-J;B%
160     WRITE#1,10+J;A%
170   NEXT J
180 NEXT I
190 RESTORE #1
200 FILE #1;"P1"
210 FILE #2;"P2"
220 FOR J=20 TO 0 STEP -1
230   READ #1,J;A%
240   READ #2,J;B%
250   IF A% <>B% THEN 400
260 NEXT J
270 PRINT "TEST SUCCESSFUL"
280 GOTO 410
400 PRINT "TEST FAILED"
```

```
410 CLOSE #1
420 CLOSE #2
430 ERASE #"P1"
440 ERASE #"P2"
```

2.6.6. SPACE-Anweisung

Mit der SPACE-Anweisung kann der Dateizeiger auf die gewuenschte Position innerhalb der Datei gesetzt werden. Dies kann durch Angabe einer relativen Entfernung (vor- oder rueckwaerts) oder durch Angabe eines Suchzeichens erfolgen.

```
SPACE #n;Bewegungszaehler
SPACE #n;Bewegungszaehler,Suchzeichen[,Returnvariable]
```

Der Dateizeiger wird um die durch den Bewegungszaehler festgelegte Anzahl Bytes versetzt, bei negativem Bewegungszaehler in Richtung Dateianfang. Wird dabei das Dateiende bzw. der -anfang erreicht, endet die Zeigerbewegung. Die EOF-Funktion zeigt an, ob die END-OF-FILE Marke erreicht wurde oder nicht.
Wird ein Suchzeichen (Zeichenkette der Laenge 1) vorgegeben, so wird in Richtung Dateiende danach gesucht. Der Suchvorgang wird beendet, wenn die angegebene Zahl von Bytes zurueckgelegt wurde, egal ob das Suchzeichen gefunden wurde oder nicht.
Die Returnvariable enthaelt als Wert die Anzahl der Bytes, um die der Dateizeiger tatsaechlich bewegt wurde.

2.6.7. RESTORE-Anweisung

Ebenfalls zur Dateipositionierung kann die RESTORE-Anweisung verwendet werden.

```
RESTORE #n
RESTORE #n;Satznummer
```

In der ersten Form wird der Dateizeiger auf den Anfang der Datei gesetzt, in der zweiten auf den angegebenen Satz einer Datei mit wahlfreiem Zugriff.

Beispiel:

```
 5 FILE #1;"AFILE"
10 FILE #2;"BFILE"
20 PRINT #1;123.4
30 WRITE #2;567.8
40 RESTORE #2
50 RESTORE #1
60 INPUT #1;C
70 READ #2;D
80 PRINT C,D
>RUN
123.4           567.8
```

Wenn die RESTORE-Anweisungen ausgefuehrt werden, wird der

Zeiger von Datei 2 auf den Anfang der Datei zurueckgesetzt. Dann wird der Zeiger von Datei 1 auf den Anfang der Datei gesetzt.
Auf Datei 1 wird als ASCII-Datei zugegriffen, wobei INPUT- und PRINT-Anweisungen benutzt werden.
Auf Datei 2 wird als binaere Datei mit READ- und WRITE-Anweisungen zugegriffen.

2.6.8. EOF-Anweisung

Im Zusammenhang mit der Positionierung des Dateizeigers wird die EOF-Funktion (end of file) verwendet.

EOF(n)

Sie zeigt mit dem Wert 1 an, dass das Ende der Datei mit der logischen Nummer n erreicht ist. Ansonsten liefert sie den Wert Null.

2.6.9. TRUNCATE-Anweisung

Mit der TRUNCATE-Anweisung koennen Dateien verkuerzt werden. Sie hat die Form:

TRUNCATE #n

Durch diese Anweisung wird jedes Byte, das sich hinter der momentanen Dateizeigerposition befindet, gestrichen. Das Byte vor dem Zeiger wird das letzte der Datei.

2.7. Programm-Segmentierung

Die Groesse eines Anwenderprogramms ist durch den im BASIC verfuegbaren Arbeitsspeicher (siehe SIZE-Kommando) begrenzt. Um auch groessere Programme abzuarbeiten, wurde die Moeglichkeit der Teilung des Programms in mehrere Segmente geschaffen. Alle Programmsegmente muessen auf der Diskette abgelegt sein und werden von dort in den Arbeitsspeicher geladen.

Zum Aufruf eines anderen Programms wird die CHAIN-Anweisung benutzt. Die COM-Anweisung ermoeglicht die gemeinsame Verwendung von Variablen durch mehrere Programme.

2.7.1. CHAIN-Anweisung

Die CHAIN-Anweisung beendet das aktuelle Programm und beginnt die Ausfuehrung eines anderen Programms. Sie hat folgendes Format:

CHAIN Zeichenketten-Ausdruck

Der ausgewertete Zeichenketten-Ausdruck ergibt den Namen eines UDOS-BASIC Programms. Das aktuelle Programm wird

beendet, und das angegebene BASIC-Programm wird geladen und gestartet. Es erfolgt keine automatische Rueckkehr zu dem aufrufenden Programmsegment.
Das aufgerufene Programm kann seinerseits ein anderes Programm mittels CHAIN aufrufen, einschliesslich desjenigen, von dem es selbst aufgerufen wurde.
Nur in COM-Anweisungen deklarierte Variablen werden waehrend der CHAIN-Operation gerettet. Alle Variablen und Felder des aktuellen Programms, die nicht in COM deklariert wurden, gehen verloren, wenn das neue Programm seine Ausfuehrung beginnt.
Alle im aktuellen Programm eroeffneten Dateien bleiben offen.

Beispiel:

```
>10 PRINT "HALLO PIT"
>20 CHAIN "PIT"
>ASAVE-PAT
>NEW
>10 PRINT "HALLO PAT"
>ASAVE-PIT
>NEW
>XEQ-PAT
 HALLO PIT
 HALLO PAT
```

Das Hauptprogramm PAT ruft das Programm PIT mit der CHAIN-Anweisung in Zeile 20 auf. Danach beginnt die Ausfuehrung von PIT. Sie endet bei der letzten Zeile von PIT.
Es wird kein Variablenwert aus PAT gerettet, nachdem CHAIN "PIT" ausgefuehrt wurde.

2.7.2. COM-Anweisung

Die COM-Anweisung wird benutzt, um Datenwerte zwischen segmentierten Programmen zu uebergeben. Die in einer COM-Anweisung aufgefuehrten Variablen werden in einem gemeinsamen Speicherbereich abgelegt, so dass auf Werte, die diesen Variablen in einem Programm zugewiesen wurden, in anderen Programmen zugegriffen werden kann (sofern Programmwechsel mittels der CHAIN-Anweisung erfolgt).
COM-Anweisungen muessen in einem Programm vor jeder anderen Anweisung stehen (Ausnahme: REM-Anweisungen). Jede Dimensionierung von Variablen wird innerhalb der COM-Anweisung vorgenommen. Variable, die in einer COM-Anweisung auftreten, duerfen nicht in einer DIM-Anweisung in demselben Programm vorkommen.
Die COM-Anweisung hat folgendes Format:

```
COM Liste
```

Die Liste besteht aus Variablendeklarationen. Einfache Variablen werden durch den Variablennamen, Felder durch den Feldnamen und die Feldgrenzen angegeben. Die Feldgrenzen werden wie in einer DIM-Anweisung angegeben.
Der Typ der in der Liste deklarierten Variablen wird mit

REAL angenommen, es sei denn, der Variablenname endet auf "n" (bei Zeichenkettenvariablen) oder "%" (bei INTEGER-Variablen). Felder und einfache Variablen, die mit COM deklariert sind, werden mit 0 vorbesetzt. Zeichenketten, die mit COM deklariert sind, werden als leere Zeichenkette vorbesetzt.
Die COM-Listen brauchen in verschiedenen Programmen nicht die gleiche Reihenfolge der Variablennamen zu besitzen. Aber Variablen, die von verschiedenen Programmen aus benutzt werden, muessen gleich sein in Namen und Dimensionierung.

2.8. TRAP-Anweisung

Die TRAP-Anweisung ist ein Mittel, um waehrend der Abarbeitung eines BASIC-Programms auftretende Fehlerbedingungen abfangen und auf externe Interrupts reagieren zu koennen, indem zu einer spezifizierten Marke (Zeilennummer) verzweigt wird. Sie kann folgendes Format haben:

```
TRAP Bedingung TO Anweisungsnummer
TRAP Bedingung OFF
TRAP ESC
```

Fuer Bedingung kann eines der Symbole ESC, ERR, EOF, KEYS oder EXT stehen.

In der erstgenannten Form wird eine TRAP eingebaut, die bei Auftreten der vorgegebenen Bedingung eine Verzweigung zu der angegebenen Anweisung bewirkt. Falls eine solche TRAP bereits eingerichtet worden war, so wird lediglich das Sprungziel auf die Anweisungsnummer geaendert, das in der neuen TRAP angegeben wurde.

Die zweite Form der TRAP-Anweisung loescht alle bisher fuer die angegebene Bedingung eingerichteten TRAPs.

Die dritte oben angegebene Form der TRAP-Anweisung dient nicht zum Einrichten einer TRAP, sondern zum Sperren der ueblichen Funktionen der ESC-Taste und zum Loeschen einer eventuell vorher gegebenen TRAP mit ESC-Bedingung.
Die ESC-Taste hat dann nur noch eine Funktion wie jede andere Taste auch und bewirkt nicht mehr den Abbruch einer Programmausfuehrung. Dadurch kann die TRAP-ESC-Anweisung zur Pruefung herangezogen werden, ob die ESC-Taste gedrueckt wurde.
Die Benutzung dieser Funktion sollte nur in Ausnahmefaellen und mit groesster Vorsicht erfolgen, da bei Verwendung von TRAP ESC ausser dem hardwaremaessigen Ruecksetzen keine Moeglichkeit besteht, aus einer Endlosschleife herauszukommen.
Durch die TRAP-Anweisung koennen in UDOS-BASIC-Programmen fuenf verschiedene Sonderfaelle gezielt behandelt werden. Die Behandlung erfolgt durch einen Sprung zu der in der TRAP-Anweisung angegebenen Anweisung, anstelle der Ausfueh-

rung der naechsten im Programm folgenden Anweisung.

Die Zeilennummer der letzten vor der Ausfuehrung der TRAP-Anweisung ausgefuehrten Anweisung ist dem Programmierer ueber die TRAP-Funktion zugaenglich.
Darueber hinaus sind auch noch weitere Informationen ueber die Bedingung, die die Verzweigung bewirkt hatte, zugaenglich, und zwar durch Benutzung der nachfolgend beschriebenen Funktionen.
TRAP-Anweisungen koennen an jeder beliebigen Stelle des Programms stehen.
Die TRAPs werden ausschliesslich zwischen der fertigen Ausfuehrung eines Befehls und dem Beginn der Ausfuehrung der naechstfolgenden Anweisung aktiviert.

Die folgenden fuenf Bedingungen koennen als Bedingung fuer eine TRAP benutzt werden:

ESC	Betaetigung der ESC-Taste
ERR	Auftreten eines Laufzeitfehlers in der Programmausfuehrung
EOF	Erreichen des Dateiendes
KEYS	Betaetigen einer der Bedienkonsoltasten (nicht ESC)
EXT	eine vom Anwender extern definierte Bedingung

Die Bedingung KEYS:

Wenn waehrend des normalen Programmablaufs irgendwelche Tasten an der Bedienkonsole gedrueckt werden, ohne dass das Programm an dieser Stelle eine Eingabe erwartet, so werden die den entsprechenden Tasten zugeordneten Werte in einen speziell hierfuer eingerichteten Puffer geschrieben, unabhaengig davon, ob eine TRAP mit der Bedingung KEYS eingerichtet wurde oder nicht.
Der Inhalt dieser Pufferspeicherzelle kann ueber die KEY-Funktion ausgelesen werden.

Die Bedingung EXT:

Eine TRAP mit dieser Bedingung wird aktiv, wenn ein zu BASIC externes Programm das "externe Interrupt-Flag" innerhalb BASIC setzt. Bei Einrichten dieser TRAP wird dieses Flag geloescht. Ein externes Anwenderprogramm kann auf dieses Flag zugreifen. Auf der Speicherstelle BSTART+2 (Adresse 4008H) steht der Zeiger zu diesem Flag. Ueber die CALL-Anweisung kann der Uebergang zwischen einem BASIC-Programm und Programmen anderer Sprachen erfolgen.

Die Bedingung ESC:

Sowohl die TRAP mit ESC-Bedingung als auch die Anweisung TRAP ESC sperren die Normalfunktion der ESC-Taste. Hier wird nochmals darauf hingewiesen, dass die Anweisung TRAP ESC nicht zum Einrichten einer TRAP zum Verzweigen auf ein bestimmtes Sprungziel dient. Die Funktion der TRAP mit ESC-Bedingung entspricht der der TRAP mit KEYS-Bedingung.
Nach der Ausfuehrung einer jeden BASIC-Anweisung prueft der BASIC-Interpreter, ob die ESC-Taste gedrueckt wurde. Die Funktion ESC hat den Wert 0, wenn die ESC-Taste nicht gedrueckt worden war und den Wert 1, wenn diese gedrueckt wurde.
Normalerweise wird durch das Druecken der ESC-Taste die Ausfuehrung des BASIC-Programms abgebrochen. Dies ist nicht mehr der Fall, wenn eine TRAP-ESC-Anweisung ausgefuehrt worden war.

Die Bedingung ERR:

Eine Verzweigung auf Grund von Fehlern in der Ausfuehrung eines Anwenderprogramms kann durch die TRAP-Bedingung ERR erfolgen. Bei Eintreten dieser Fehlerbedingung wird von der TRAP die Fehlernummer ueber die ERR-Funktion uebergeben. Entsprechend wird die Zeilennummer der zugehoerigen Anweisung, die den Fehler hervorgerufen hatte, ueber die ERR-Funktion dem Programmierer zugaenglich gemacht.

Das Auftreten eines jeden Fehlers vor der Initialisierung der TRAP fuehrt zur Beendigung der Ausfuehrung des Programms mit einer normalen Fehlermeldung.

Die Bedingung EOF:

Normalerweise wird die Variablenliste einer READ, INPUT oder LINPUT-Anweisung nicht verarbeitet, wenn eine solche Anweisung auf die EOF-Bedingung stoesst und die EOF(n)-Funktion bekommt fuer die zugehoerige Datei n den Wert 1 zugeordnet. Eine darauffolgende READ, INPUT oder LINPUT-Anweisung fuer die gleiche Dateinummer ruft eine Fehlermeldung hervor. Wenn ueber die Anweisung TRAP EOF TO marke eine EOF TRAP initialisiert wurde, so bewirkt die Ausfuehrung des ersten Eingabebefehls, der auf die EOF-Bedingung stoesst, statt dessen eine Verzweigung zu der in der TRAP-Anweisung spezifizierten Marke.

Das Verhaeltnis von Ablaufumgebung und TRAPs:

Jedesmal, wenn eine Funktion, die aus mehreren Programmzeilen besteht, aufgerufen wird, wird eine neue Ablaufumgebung geschaffen. Jede dieser Ablaufumgebungen kann ihre eigenen lokalen TRAPs haben, die dahingehend lokal definiert sein koennen, dass sie in dieser Umgebung eingerichtet oder deaktiviert werden koennen, ohne andere Ablaufumgebungen zu beeinflussen.
Das Auftreten von ERR- oder EOF-Bedingungen in einer bestimmten Ablaufumgebung hat Einfluss auf die

Programmausfuehrung, wenn die entsprechende TRAP in der momentan behandelten Ablaufumgebung nicht existiert, jedoch in anderen Ablaufumgebungen aktiviert worden war.
In diesem Fall werden solche Ablaufumgebungen in der umgekehrten Reihenfolge in der sie aufgebaut worden waren zurueckgestellt, bis eine Ablaufumgebung gefunden wird, in der eine aktive ERR- oder EOF-TRAP vorkommt. In dieser Ablaufumgebung wird auf diese TRAP verzweigt. Die Zeilennummer, die von der TRP-Funktion geliefert wird, identifiziert die Anweisung, die als letzte ueber die ERR- oder EOF-Bedingung die TRAP aktiviert hatte.

Funktionen, die der TRAP-Behandlung zugeordnet sind:

TRP liefert den ganzzahligen Wert der letzten Zeilennummer, die vor Eintreten der TRAP-Bedingung zur Ausfuehrung gekommen war

ESC uebergibt den Wert 1, falls die ESC-Taste gedrueckt worden war, sonst den Wert 0. Mit der ESC-Bedingung ist eine Verriegelung gekoppelt, so dass ESC den Wert 1 genau einmal liefert, wenn die ESC-Taste oefters gedrueckt wurde. ESC kann in Verbindung mit TRAP EOF-Anweisung benutzt werden

KEYS$ liefert eine Zeichenfolge, die den den Tasten entsprechenden Werte ausser ESC uebermittelt, die waehrend der Programmausfuehrung gedrueckt worden waren. Dabei sind nicht die Eingaben beruecksichtigt, die waehrend irgend einer Eingabeanweisung erfolgten. Waren bei der Programmausfuehrung keine Tasten der Bedienkonsole gedrueckt worden, so liefert diese Funktion den Wert 0.
Auch hier ist eine Software-Verriegelung implementiert, so dass die den Tasten zugeordneten Werte genau einmal beruecksichtig werden.

ERR liefert die Fehlernummer zur Kennzeichnung des Fehlers, der die letzte ERR-Bedingung erfuellt hatte. Diese Funktion wird zusammen mit der TRAP-ERR-Anweisung benutzt.

Beispiel zur Anwendung der TRAP-Funktion:

```
10   TRAP ERR TO 1000
      .
      .
1000 PRINT "*** FEHLER IN ZEILE: ";TRP;" ***"
1010 PRINT "*** FEHLERNUMMER: ";ERR;" ***"
1020 STOP
```

3. Kopplung mit Assemblerprogrammen

Innerhalb eines BASIC-Programms koennen in Assemblersprache geschriebene Prozeduren ueber die CALL-Anweisung aufgerufen werden. Sie hat folgendes Format:

CALL "Prozedurname",E.-parameterliste;A.-parameterliste

Nach dem Prozedurnamen (Zeichenkettenausdruck) kann eine Liste von an die Prozedur zu uebergebenden Parametern erscheinen. Eine moegliche Liste von Parametern, die von der Prozedur an BASIC zurueckgegeben werden, wird durch ein Semikolon eingeleitet. Die Elemente der Parameterlisten koennen sowohl numerische Groessen (reell oder ganzzahlig) als auch Zeichenketten sein. Sie werden durch Kommas voneinander getrennt. Jede Liste kann entfallen.
Bei der Kopplung einer Assemblerprozedur mit einem BASIC-Programm sind zwei Aspekte wichtig, einmal die Bekanntmachung der externen Prozedurnamen an BASIC (dies erfolgt vor dem Start von BASIC) und das Parameteruebergabeschema beim Aufruf der Prozedur. Zum erste Punkt muessen innerhalb der Assemblerprozedur zwei Tabellen aufgebaut werden (siehe Bild 1).

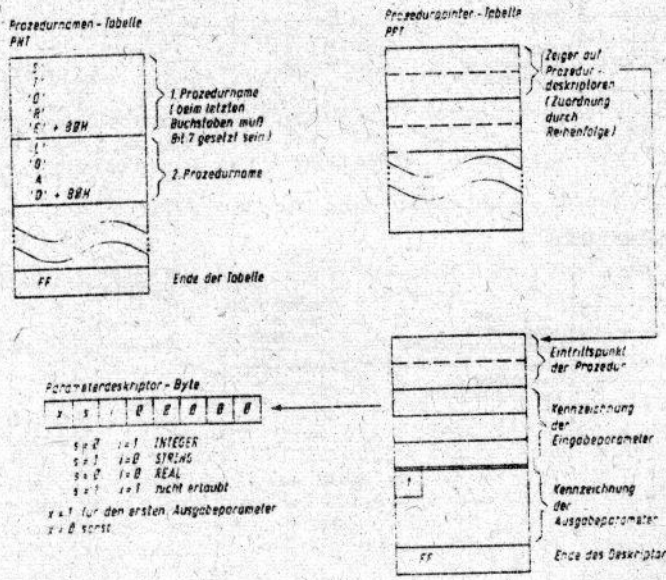

Bild 1: Aufbau der PNT- und PPT-Tabellen
Diese enthalten einmal alle Namen der von BASIC aus aufruf-

baren Prozeduren (Prozedurnamen-Tabelle: PNT) und zum anderen die Adressen (Prozedurpointer-Tabelle: PPT) von Prozedurbeschreibungsbloecken. Fuer jede in der PNT eingetragene Prozedur muss es einen Prozedurbeschreibungsblock geben, in den der Eintrittspunkt der Prozedur und die Anzahl und der Typ der Parameter eingetragen werden.
Ganze Zahlen werden intern durch zwei Byte dargestellt. Reelle Zahlen werden intern durch eine 8 Byte grosse BCD-Gleitkommazahl repraesentiert. Zeichenketten werden durch ihre ASCII-Werte (ein Zeichen ein Byte) dargestellt. Vor dem ersten Zeichen wird in 2 mal 2 Byte zweimal (!) die Laenge der Zeichenkette angegeben.

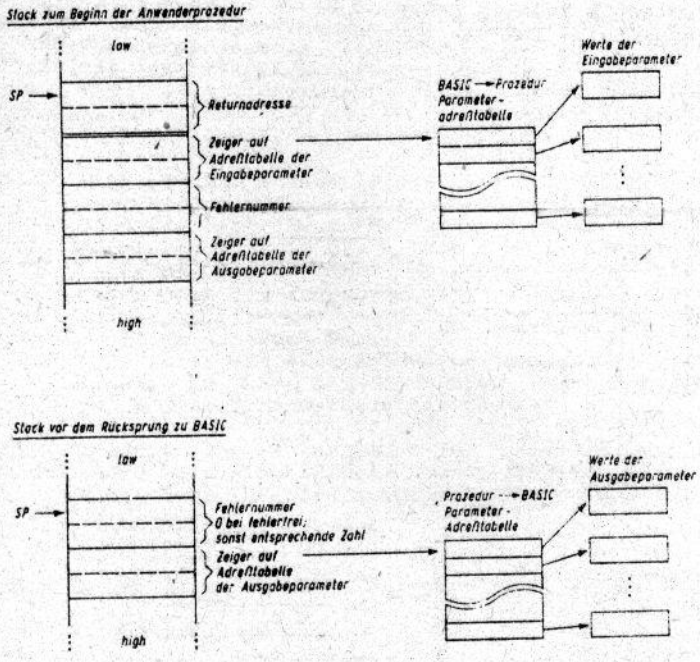

Bild 2: Parameteruebergabeschema bei der Kopplung von BASIC mit externen Prozeduren

Die Parameteruebergabe zwischen dem BASIC-Programm und der

externen Prozedur erfolgt ueber den Stack (s. Bild 2).
Die Vorbereitung des Stacks erfolgt bevor der CALL ausgefuehrt wird durch BASIC (ueber Push-Befehle).
Zunaechst werden 2*2 Byte reserviert. In diese muessen im externen Assemblerprogramm vor der Rueckkehr zum BASIC-Programm der Zeiger auf die Adresstabelle der Ausgabeparameter und der Fehlerkode eingetragen werden.
Die naechsten zwei Byte enthalten den Zeiger, der auf die Adresstabelle der Eingabeparameter weist.
Die obersten beiden Bytes des Stack enthalten nach dem Aufruf des Assemblerprogramms die Rueckkehradresse.
Vor der Rueckkehr zu BASIC, die ueber JUHP erfolgt, muss der Stackpointer auf den L-Teil des Fehlerkodes weisen. Das anschliessende Beispiel macht die Verfahrensweise deutlich.

Waehrend das Einrichten der Adresstabelle der Eingabeparameter und das Bereitstellen der Eingabeparameter-Werte durch BASIC erfolgt (abgeleitet aus der CALL-Anweisung), muessen im externen Assemblerprogramm das Einrichten der Adresstabelle fuer die Ausgabeparameter und das Bereitstellen der Ausgabeparameter-Werte durch den Anwender vorgenommen werden.

Der BASIC-Interpreter muss vor dem Start ueber die Existenz der beiden Tabellen (PNT und PPT) informiert werden. Dies geschieht ueber folgende Befehlsfolge zum Beginn der externen Prozedur:

```
LD HL,NAMETAB      ;Adresse von PNT
LD DE,POINTAB      ;Adresse von PPT
JP 4006H           ;BASIC-Startadresse (BSTART)
```

Die externe Prozedur sollte auf die hoechstmoegliche Adresse gebunden werden, um fuer den BASIC-Interpreter noch einen moeglichst grossen Arbeitsbereich zu erhalten.

Der Start von BASIC erfolgt dann ueber die externe Prozedur:

```
%USERPROZEDUR,      Laden der externen Prozedur, aber
                    nicht starten

%BASIC,X C000       (Startadresse der Prozedur: C000)
                    Laden von BASIC und starten der
                    Anwenderprozedur
```

Zu beachten ist noch, dass bei Aufruf der Prozedur der Stackpointer (SP) auf den internen BASIC-Stackbereich gesetzt ist. Die dort maximal garantierte Groesse betraegt 30 Byte. Bei groesserem Bedarf muss deshalb mit einem eigenen Stackbereich gearbeitet werden. Vor der Rueckkehr zu BASIC ist der Stackpointer wieder auf die Adresse des BASIC-Stackbereichs zu setzen.

Beispiel fuer Aufruf eines externen Assemblerprogramms:

BASIC-Programm:

```
10 FOR I=1 TO 5
20 CALL "ONE",I%;I1%
30 PRINT I%,I1%
40 NEXT I%
```

Externes Assemblerprogramm:
(gebunden ab 0C000H)

```
                GLOBAL   START

BSTART:   EQU   4006H           ;Startadr. BASIC, wenn PNT
                                ;und PPT uebergeben werden
                                ;sollen
START:    LD    HL,NAME         ;Adresse von PNT
          LD    DE,PTAB         ;Adresse von PPT
          JP    BSTART

NAME:     DEFB  'O'             ;Prozedur-Namen-Tabelle
          DEFB  'N'             ;(PNT)
          DEFB  'E'+80H         ;Bit 7=1 beendet Namen
          DEFB  0FFH            ;beendet PNT

PTAB:     DEFW  PD1             ;Prozedur-Zeiger-Tabelle
                                ;(PPT)

PD1:      DEFW  ONE             ;Adresse der Prozedur
          DEFB  20H             ;Eingabeparameter: INTEGER
          DEFB  0A0H            ;Ausgabeparameter: INTEGER
                                ;(Bit 7=1)
          DEFB  0FFH            ;Ende der Proz.beschr.(PD)

     ;PROZEDUR ONE
     ;CALL "ONE",X%;Y%
     ;RETURNS Y% = 2*X%

ONE:      LD    IX,0
          ADD   IX,SP
          PUSH  IX              ;Stackpointer-Adresse

          LD    H,(IX+3)
          LD    L,(IX+2)
          PUSH  HL
          POP   IY              ;IY zeigt auf Adresstabelle
                                ;der Eingabeparameter

          LD    HL,PPT1         ;Zeiger fuer Adresstabelle
                                ;der Ausgabeparameter
          LD    (IX+7),H
          LD    (IX+6),L
          PUSH  HL
          POP   IX              ;IX zeigt auf Adresstabelle
```

```
        LD      H,(IY+1)        ;Setzen HL AUF 1.Eingabepa-
        LD      L,(IY+0)        ;rameteradresse

        LD      E,(HL)          ;Holen 2 Byte INTEGER-
        INC     HL              ;Eingabeparameter
        LD      D,(HL)
        EX      DE,HL
        ADD     HL,HL           ;HL:=2*X%

        LD      (OUT),HL        ;Ablegen Ausgabeparameter
        LD      HL,OUT          ;Adr. des Ausgabeparameters
        LD      (IX+1),H        ;in Parameter-Pointer-
        LD      (IX+0),L        ;Tabelle eintragen

        LD      HL,0            ;Error-Kode=0 setzen

        POP     IX              ;Urspruengl. Stackpointer-
                                ;Adresse
        LD      (IX+5),H        ;Fehler-Kode eintragen
        LD      (IX+4),L

        POP     HL              ;BASIC-Eeturnadresse
        POP     BC              ;Eingabeparameter-Pointer
                                ;freigeben
        JP      (HL)            ;Ruecksprung zu BASIC
```

im RAM-Bereich:

```
PPT1:   DEFS    2               ;Parameterpointer, Zeiger
                                ;zeigt auf Ausgabepara-
                                ;meter (OUT)
OUT:    DEFS    2               ;Ausgabeparameter, 2 Byte
                                ;fuer einen INTEGER-Wert
```

Aufruf unter UDOS:

```
    %START,                 ;Laden der externen Prozedur, aber
                            ;nicht starten
    %BASIC,X C000           ;Laden BASIC und Starten der ex-
                            ;ternen Anwenderprozedur
```

4. Programmbeispiele

Zur Demonstration werden die Programmlistings der im Abschn. 2.1.3. erwaehnten Programme FLIST.BP und CAT.BP angegeben.

Das Programm FLIST erwartet die Eingabe eines Dateinamens, eroeffnet diese Datei und gibt den Inhalt der Datei ueber den Bildschirm aus. Falls als erstes Zeichen des Dateinamens ein Doppelkreuz angegeben wird, so erfolgt zusaetzlich noch eine Ausgabe der Zeilennummern. Das Einlesen des Inhalts der Datei (sie muss vom Typ ASCII sein) erfolgt

zeilenweise.

Programm FLIST.BP:

```
 10 DIM A$(200)
 20 INPUT "FILE:",A$
 30 IF A$(1,1)="#" THEN DO
 40 LET A$=A$(2)
 50 LET L%=1
 60 DOEND
 70 ELSE LET L%=0
 80 FILE #1;A$+";ACC=IN",R%
 90 IF R%<>0 THEN 180
100 PRINT
110 LET I%=0
120 LET I%=I%+1
130 LINPUT #1;A$     \ LIEST BIS ZUM CR
140 IF EOF(1) THEN 200
150 IF L% THEN PRINT I%;"<9>";A$
160 ELSE PRINT A$
170 GOTO 120
180 PRINT
190 PRINT "*** DATEIFEHLER: ";R%;" ***"
200 END
```

Das Programm CAT ermittelt die Namen aller auf einer Diskette befindlichen Dateien. Es erwartet die Eingabe einer Laufwerknummer. Danach kann noch eine Dateinamenendung (z.B.: .BP) spezifiziert werden. Es werden dann nur die Dateien, deren Namen mit der eingegebenen Zeichenkette endet aufgelistet.
Das Programm eroeffnet das Inhaltsverzeichnis der angegebenen Diskette und liest den Inhalt satzweise (jeweils 128 Byte gleich ein Satz der Datei DIRECTORY) ein. Der Aufbau eines Satzes des Inhaltsverzeichnisses einer Diskette kann mit Hilfe des UDOS-Dienstprogramms FILE.DEBUG ermittelt werden. Dort sind die Dateinamen und der Verweis (Diskadresse) auf den Beschreibungssatz der Datei abgelegt (1 Byte Dateinamenlaenge, Dateiname maximal 32 Byte, 2 Byte Zeiger).
Die Variable N erhaelt jeweils die Dateinamenlaenge. In der Zeile 270 wird der Test auf Uebereinstimmung mit der eingegebenen Endung durchgefuehrt, und in der Zeile 280 erfolgt die Ausgabe der Dateinamen.

Programm CAT.BP:

```
 10 REM BASIC CAT-PROGRAMM
 20 REM 10/84
 30 TRAP ERR TO 330
 40 DIM A$(128)
 50 DIM D$(10)
 60 DIM E$(20)
```

```
 70 LINPUT "LAUFWERK: ",D¤
 80 TRAP ERR TO 110
 90 LET D¤=":"+STR¤(VAL(D¤))
100 GOTO 130
110 LET D¤=""
120 TRAP ERR TO 330
130 PRINT
140 LINPUT "ENDUNG: ",E¤
150 PRINT
160 FILE #1;"¤ZDOS"+D¤+"/DIRECTORY;ACC=IN"
170 READ #1;A¤     \ SATZ EINLESEN
180 IF EOF(1) THEN 320
190 LET L=1    \ STARTPUNKT IN DER ZEICHENKETTE
200 LET N=ASC(A¤(L,L))    \ LAENGE DES DATEINAMENS
210 IF N=255 THEN  170    \ KEIN WEIT. DAT.NAME IM SATZ
220 IF N>128 THEN  DO    \ SECRET-BIT GESETZT (BIT 7)
230 LET N=N-128     \ BIT RUECKSETZEN
240 GOTO 300    \ DATEINAMEN NICHT AUSGEBEN
250 DOEND
260 IF N=0!N=127 THEN 170    \ FEHLERHAFTE EINTRAGUNGEN
270 IF A¤(L+N+1-LEN(E¤),L+N)<>E¤ THEN 300 \ VERGL.ENDG.
280 PRINT A¤(L+1,L+N);   \ AUSGABE DES DATEINAMENS
290 PRINT,    \ AUF NAECHSTE TABULATOR-POSITION
300 LET L=L+1+N+2    \ NAECHSTE DATEINAMENSEINTRAGUNG
310 GOTO 200
320 END
330 PRINT "CAT NICHT MOEGLICH, FEHLER: ";ERR
340 END
```

Anhang: Liste der Fehlernummern und Erlaeuterung

Gruppe 1: Uebersetzungszeitfehler

Fehlernummer	Erlaeuterung
1	Fehlerhafte Anweisungsnummer
2	Eingabe nicht erkennbar
3	Zahl der oeffnenden und schliessenden Klammern verschieden
4	Literal zu lang
5	Unzulaessige Anweisung im zweiten Teil
6	Ausdruck zu komplex
7	Fehlerhaftes Element im Ausdruck
8	Fehlendes schliessendes Anfuehrungszeichen
9	Falscher Anwenderfunktionsname
10	Zeichen nach Anweisungsende
11	Fehlendes "#"
12	Fehler im Ausdruck fuer die Dateibeschreibung
13	Fehlendes ";"
14	Fehlender oder falscher Dateiname
15	Unzulaessige Return-Variable
16	Unzulaessiger Ausdruck fuer die Datensatznummer
17	Fehlende oder unzulaessige Anweisungsnummer
18	Unzulaessiger Auswahlausdruck
19	Unzulaessiger Funktionsname
20	Unzulaessige THEN, ELSE-Klausel
21	Unzulaessiges Zuweisungsobjekt
22	Fehlender Zuweisungsoperatur
23	Unzulaessiger Ausdruck
24	Unzulaessige Variable
25	Unzulaessige Listenelemente
26	Unzulaessiger formaler Parameter
27	Als FOR/NEXT-Index sind keine einfachen Variablen angegeben
28	Unzulaessige "USING"-Zeichenkette
29	LINPUT-Variable muss vom Typ Zeichenkette sein
30	Fehlendes "="
31	Fehlender oder unzulaessiger Anfangswert
32	Fehlendes "TO"
33	Fehlender oder unzulaessiger Endwert
34	Fehlende oder unzulaessige Schrittweite
35	Fehlerhafte Aufgliederung
36	Fehlendes "THEN"
37	Unzulaessige Funktionsdefinition
38	Kann im Tischrechnermode nicht ausgefuehrt werden
39	Unzulaessiges TRAP-Objekt
40	Kommando nicht erlaubt in Verbindung mit Datei
41	Kompilierte Datei in unzulaessigem Zusammenhang

Fehlernummer	Erlaeuterung
42	Undefinierte Umgebung, keine Fortsetzung moeglich
43	Keine BASIC-Datei
44	Unzulaessiger Parameter

Gruppe 2: Programmstrukturfehler

Fehlernummer	Erlaeuterung
50	Bezugnahme auf undefinierte Variablen oder undimensioniertes Feld
51	Bezugnahme auf nicht existierende Zeilennummer
52	Bezugnahme auf undefinierte Funktion
53	Bezugnahme auf nicht deklarierte Dateinummer
54	Geschachtelte Funktionsvereinbarungen sind unzulaessig
55	Unzulaessige Zahl von Puffern
56	Unzulaessige Dateinummer
57	Unpaarige Zahl von DO/DOEND's
58	RETURN ohne vorhergehendes GOSUB
59	FNEND ohne RETURN
60	NEXT ohne FOR
61	Unzulaessige Verschachtelung von FOR/NEXT
62	NEXT nicht im selben Block mit FOR
63	INPUT/READ kann keine Funktionen einlesen
64	Fehlerhafter Anwenderfunktionsaufruf
65	Typ unpassend
66	Dimension zu gross
67	Zeichenkette darf nicht rueckdimensioniert werden
68	Unkorrekte Zahl von Argumenten bzw. Indizes
69	Unkorrekte Zahl von CALL/SYSTEM-Parametern
70	Bezugnahme auf eine undefinierte Prozedur
71	Unzulaessige Begrenzungskette
72	Unzulaessiger TAB-Gebrauch
73	Unzulaessige TRAP-Situation

Gruppe 3: Systembegrenzungen und -stoerungen

Fehlernummer	Erlaeuterung
80	Symboltabelle gefuellt
81	Zu viele Dateien eroeffnet
82	Speicherueberlauf
83	Runtime-Stack-Ueberlauf
84	DO-Verschachtelung zu tief
85	Unvollstaendiges Betriebssystem
86	Merkmal nicht implementiert
87	Interpreter-Fehler
88	Interface-Fehler zum Betriebssystem

Gruppe 4: Fehler in Verbindung mit Feldern und Zeichenketten

Fehlernummer	Erlaeuterung
100	Argument ausserhalb eines Wertebereiches
101	Unzulaessige Teilzeichenkettenbeschreibung
102	Indexvariable ausserhalb eines Wertebereiches
103	Zweite Indexvariable ausserhalb eines Wertebereiches
104	Versuch, die Dimension zu vergroessern
105	Fehlende Indizes

Gruppe 5: Ein-/Ausgabe-Fehler

Fehlernummer	Erlaeuterung
120	Datei existiert nicht
121	Datei existiert schon
122	Positionierung vor Beginn der Datei
123	Positionierung ueber Ende der Datei hinaus
124	Ausserhalb von DATA
125	Unzulaessiger Dateiname
126	Unzulaessiger Dateityp
127	Dateischutzfehler
128	Datei bereits eroeffnet
129	Nicht zugewiesene Ein-/Ausgabe
130	Datei nicht eroeffnet
131	Dateifehler
132	Diskettenfehler
133	Diskette nicht bereit
134	Diskette voll
135	Ungueltige Operation
136	Numerischer Umwandlungsfehler
137	Unzureichende Eingabe
138	Dateistruktur falsch

Gruppe 6: Warnungen

Fehlernummer	Erlaeuterungen
140	Unzulaessige Zahl
141	Ueberlauf
142	Warnung: Zahl zu klein
143	Division durch Null
144	Quadratwurzel von einer negativen Zahl
145	Logarithmus von einer negativen Zahl oder Null
146	Zeichenkette wird waehrend Zuweisung abgeschnitten
147	Format zu klein, um Zahl aufzunehmen
160	Unzulaessiges Formatzeichen
161	Unzulaessiges Exponentenfeld
162	keine Ziffernpositionen
163	keine Formatzeichenkette

U 8 8 0 - P A S C A L

Benutzerhandbuch

Vorwort

Die vorliegende Anwenderdokumentation enthaelt die notwendigen Informationen zur Uebersetzung und Abarbeitung von PASCAL-Quellprogrammen unter dem Betriebssystem UDOS. Es werden die Einschraenkungen und Erweiterungen dieser PASCAL-Version erlaeutert und einige Hinweise zur Handhabung gegeben.
Diese Schrift ist keine PASCAL-Sprachbeschreibung. Fuer eine ausfuehrliche Beschreibung sei auf existierende Literatur verwiesen:

Jensen, K.; Wirth, N.
PASCAL User Manual and Report (second edition)
Springer Verlag, New York 1976

Paulin, G.; Schiemangk, H.
Programmieren mit PASCAL (2. Aufl.)
Akademie-Verlag, Berlin 1986

PASCAL fuer das Betriebssystem UDOS 1526
VEB Robotron-Buchungsmaschinenwerk
Karl-Marx-Stadt 1984

Das PASCAL-System benutzt einige Moeglichkeiten des Betriebssystems UDOS, wie reihenweise Ein/Ausgabe zur Dateiarbeit und die Speicherverwaltung (memory manager). Eine umfassende Beschreibung dieser Moeglichkeiten ist der Anwenderdokumentation zum Betriebssystem zu entnehmen.

Inhaltsverzeichnis

1. Das PASCAL-System

1.1. Ueberblick

Das PASCAL-System besteht aus drei Programmen: einem PASCAL-Compiler, einem Post-Prozessor und einem Interpreter. Der Compiler, selbst ein PASCAL-Programm, erzeugt P-Kode, einen symbolischen Zwischenkode fuer einen staokorientierten Rechner. Der Post-Prozessor konvertiert den vom Compiler erzeugten P-Kode zur Abarbeitung auf kleineren Rechnern in einen verdichteten Objektkode, der dann unter Steuerung des Interpreters abgearbeitet wird.

1.2. Ablaufskizze

Um ein PASCAL-Programm abzuarbeiten, muss das Quellprogramm vom Compiler uebersetzt werden. Der vom Compiler erzeugte P-Kode ist die Eingabe fuer den Post-Prozessor, der den verdichteten P-Kode erzeugt. Dieser verdichtete P-Kode wird dann interpretativ abgearbeitet. Das Bild 1.1 verdeutlicht diesen Prozess.

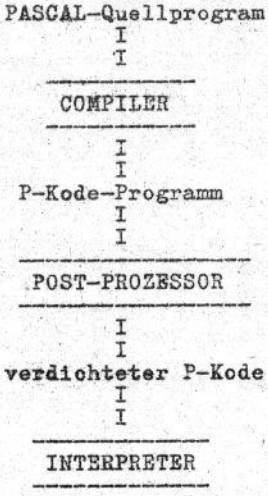

Bild 1.1 PASCAL-Ablaufuebersicht

2. Laufzeitumgebung

2.1 Speicherzuweisung

Die moegliche Groesse eines zu uebersetzenden und abzuarbeitenden PASCAL-Programms haengt von der Groesse des verfuegbaren zusammenhaengenden Speicherbereichs ab. Deshalb sollten alle vom Anwender waehrend der Arbeit mit PASCAL

noch benoetigten Programme (E/A-Treiber, ec.) auf die hoechst moeglichen Adressen geladen werden. Dadurch wird der groesstmoegliche zusammenhaengende Speicherbereich fuer Anwenderprogramme zur Verfuegung gestellt. Das folgende Bild verdeutlicht den Speicheraufbau bei der Uebersetzung oder Abarbeitung von PASCAL-Programmen unter dem Betriebssystem UDOS:

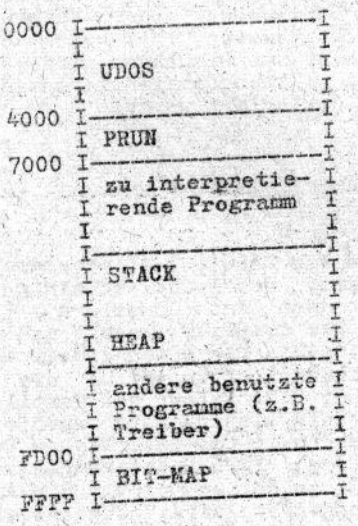

Bild 2.1 Speicherzuordnung

Das zu interpretierende Programm ist der verdichtete P-Kode entweder eines PASCAL-Anwenderprogramms, des Compilers oder des Post-Prozessors. Der von PRUN benoetigte Speicherbereich besteht aus dem Interpreter, einem vom Interpreter benutztem E/A-Paket und einem kleinen Startmodul, der das zu interpretierende Programm vor dem Aufruf des Interpreters einliest. Speicherbereich wird weiterhin fuer den PASCAL-Stack- und -Heap-Bereich benoetigt. Dieser Bereich wird folgendermassen dynamisch zugewiesen:

(1) Bei Aufruf von PRUN veranlasst der Startmodul eine Anforderung an den UDOS Memory Manager zur Ermittlung des groessten verfuegbaren Speichersegments.

(2) Das zu interpretierende Programm wird an den Anfang dieses Bereichs eingelesen.

(3) Der Rest dieses Bereichs wird fuer den Stack- und Heapaufbau benutzt. Der Stack waechst aufwaerts zu den hoeheren Adressen und der Heap waechst rueckwaerts zu den niederen Adressen.

Um so groesser der verfuegbare Stack/Heap-Bereich, desto groesser koennen die abzuarbeitenden Programme sein. PRUN wird auf die niedrigst moegliche Adresse, also im Anschluss an UDOS geladen. Wie aus dem Bild der Speicherzuordnung ersichtlich wird, bleibt kein Platz fuer Ein/Ausgabepuffer, d.h., nach einmaligem Aufruf zur Speicherzuordnung uebernimmt das PASCAL-System die Steuerung der Speicherzuweisung. Beim Eroeffnen einer Datei liegt der Pufferbereich im PASCAL-Stackbereich. Beim Schliessen einer Datei wird dieser Bereich nicht sofort freigegeben, erst bei Abschluss der entsprechenden Prozedur oder des Programms wird dieser Bereich freigegeben. Deshalb sollten Dateien in dem Niveau eroeffnet werden, in dem sie deklariert werden.

3. PASCAL-Compiler

3.1. Compiler-Ein-/Ausgabe

Vor Aufruf des Compilers muss ein PASCAL-Quellprogramm mit dem UDOS-Editor erstellt werden. Dabei sollten einige Einschraenkungen und Erweiterungen des Compilers (siehe Abschn. 7) beachtet werden. Der Compiler erzeugt wahlweise eine Listing- und eine Objektdatei. Die Namen dieser beiden Dateien werden in der Kommandozeile zum Aufruf des Compilers angegeben (siehe Abschn. 3.2. Aufruf des Compilers). Ein einfaches Beispiel fuer ein Listing wird nachfolgend angegeben. Die Listingdatei enthaelt die numerierten Quellzeilen und eine Anzeige des Verschachtelungsniveaus. Daneben existiert eine Spalte, in der kummulativ die Anzahl der erzeugten P-Anweisungen angegeben wird. Zum Abschluss eines jeden Listings werden die Anzahl der erkannten Fehler, die Anzahl der gelesenen Zeilen, die Anzahl der uebersetzten Prozeduren und die Anzahl der erzeugten P-Anweisungen angegeben. Erkannte Fehler werden unter der entsprechenden Quellzeile angezeigt. Die Fehlernummern des Compilers und ihre Erlaeuterungen sind im Anhang A zusammengefasst.

Der Compiler erzeugt standardmaessig eine Listing- und Objektdatei. Falls diese Erzeugung nicht gewuenscht wird ist es notwendig im Quellprogramm einen Kommentar der Form: (*¤L-,C-*) unterzubringen (L fuer Listing- und C fuer P-Kodedatei; - unterdrueckt Erzeugung, + erlaubt Erzeugung).

Bei der Unterdrueckung der Listingerzeugung wird noch eine kurze Ausgabe, die aus dem Listingkopf und den oben beschriebenen Abschlussmeldungen besteht generiert. Zusaetzlich wird fuer jede verarbeitete Quellzeile das Zeichen "+" ausgegeben. Nach jeweils 25 Zeilen wird zusaetzlich die Anzahl der bis dahin verarbeiteten Quellzeilen ausgegeben. Ein einfaches Programmlisting ist im folgenden Bild dargestellt.

```
LINE  P_LC LVL    PASCAL   2.0
   1     4  1) program BSP(input, output);
   2     4  1)
   3     6  1) var      X : integer;
   4    26  1)          Y : array[1..10] of integer;
   5    26  1)
   6     0  1) begin
   7    17  1)          readln (X, Y[5]);
   8    30  1)          writeln (X, Y[5]);
   9    26  1) end.

****    NO SYNTAX ERROR(S) DETECTED.
****     9 LINE(S) READ,    0 PROCEDURE(S) COMPILED,
****    31 P-INSTRUCTIONS GENERATED.
```

Zusaetzlich zur Zeilennummern- und Niveauspalte ist die P-Kode-Zuweisungsspalte (P_LC) folgendermassen zu interpretieren:

Zeilen-Nr.	Erlaeuterung
1	Vereinbarungsteil des Programms, 4 zeigt den Wert des P-Kode-Zuweisungszaehlers (LC) an. LC beginnt bei 0. Die Programmzeile zeigt an, dass zwei E/A-Dateien benoetigt werden. Entsprechend wurde der LC erhoeht, um die notwendigen 2 Byte Stackbereich fuer jede Datei zu reservieren.
2	Der LC wurde um zwei Byte erhoeht, da zwei Byte fuer die in der Zeile vereinbarte Variable X benoetigt werden.
3	Der LC wurde um 20 erhoeht, da die Groesse des Feldes Y 20 ist (Erklaerung siehe Abschn. 6).
4	Das Schluesselwort BEGIN kennzeichnet das Ende des Vereinbarungsteils, d.h., der P-Befehlszaehler wird fuer diese Prozedur, in diesem Fall das Hauptprogramm, neu initialisiert.
5	Nach Verarbeitung von READLN wurden 17 P-Anweisungen erzeugt.
6	Weitere 13 P-Anweisungen wurden nach Verarbeitung von WRITELN erzeugt.
7	Die letzte Zeile des Programms oder der Routine zeigt den kummulativen Wert des LC an.

3.2. Aufruf des Compilers

Start des Compilerlaufs durch Eingabe von:

```
%PRUN PASCAL,Dateiname,Dn1,Dn2,Dn3
```

wobei Dateiname die Quelldatei, Dn1 die Listingdatei, Dn2 die Table-Datei und Dn3 die P-Kode-Datei kennzeichnet. Dn2 und Dn3 sind Eingabedateien fuer den Post-Prozessor. Wenn die Ausgabedateien bereits existieren, dann werden sie ueberschrieben.
Durch Weglassen von Dn1 und/oder Dn2, Dn3 ist eine direkte Verbindung des Ausgabestroms mit der Bedienerkonsole moeglich, ohne dass eine UDOS-Datei angelegt wird.

z.B.: %PRUN PASCAL,TESTPROG,,TEMP.T,TEMP.P

uebersetzt das Programm TESTPROG, gibt das Listing ueber die Konsole aus und erzeugt die Table-Datei TEMP.T und die P-Kode-Datei TEMP.P

3.3. Logische E/A-Einheiten

Der Compiler benutzt folgende logische E/A-Einheiten:

2 (CONOUT)	falls Listingausgabe ueber Konsole erfolgt
5	Compiler
5-8	Quellprogramm, Ausgabelisting, Table- und P-Kode-Datei

Wenn eine Listingdatei mit der Konsole verbunden ist, dann werden die logischen Einheiten 5, 7 und 8 benutzt, anstelle von 5 fortlaufend.

4. Post-Prozessor

4.1. Post-Prozessor-Ein-/Ausgabe

Der Post-Prozessor erzeugt einen verdichteten Objektkode als Eingabe fuer den Interpreter und eine Listingausgabe, die die Abarbeitung des Post-Prozessors dokumentiert. Diese Ausgabe besteht aus:

- Kopfzeile als Meldung des Post-Prozessors
- Der Name einer jeden verarbeiteten Routine, gefolgt durch mehrere Zeichen "+", ein Zeichen jeweils fuer 20 P-Anweisungen innerhalb der Routine
- Abschlusszeile, die die Anzahl der P-Anweisungen und die Anzahl der von diesen Anweisungen belegten Bytes enthaelt.

4.2. Aufruf des Post-Prozessors

Aufruf von UDOS aus durch Eingabe von:

```
>PRUN PASM,Dn1,Dateiname,Dn2,Dn3
```

wobei Dn1 die zu verarbeitende P-Kode-Datei und Dn3 die Table-Datei kennzeichnet. Beide wurden vom Compiler erzeugt. Dateiname kennzeichnet die Listingausgabedatei des Post-Prozessors und Dn2 die Datei, die den verdichteten P-Kode enthaelt. Falls diese Dateien bereits existieren, so werden sie ueberschrieben. Durch Weglassen des Dateinamens ist es moeglich die Listingausgabe des Post-Prozessors direkt auf die Konsole zu legen.

z.B.: >PRUN PASM,TEMP.P,,TEMP.T,TESTDSP.POBJ

4.3. Logische E/A-Einheiten

Der Post-Prozessor benutzt folgende logische Einheiten:

2 (CONOUT)	falls Listingausgabe ueber Konsole erfolgt
5	Post-Prozessor
5 - 8	P-Kode-Datei, Ausgabelisting, Table-Datei und Objektdatei

Wenn die Listingdatei mit der Konsole verbunden ist, dann werden die logischen Einheiten 5, 7 und 8 benutzt, anstelle von 5 fortlaufend.

5. Der Interpreter

5.1. Funktion

Zur Abarbeitung eines PASCAL-Programms liest ein Ladeprogramm die zu interpretierende Objektdatei in den Speicher ein und startet das Interpreterprogramm. Der Interpreter fuehrt jede verdichtete P-Anweisung aus, d.h., das Programm wird abgearbeitet.

5.2. Abarbeitung eines PASCAL-Programms

Nach Ausfuehrung des Compiler- und des Post-Prozessor-Laufs wird die Abarbeitung eines PASCAL-Programms durch folgende Kommandozeile gestartet:

```
>PRUN Dn1,Dateinamen
```

wobei Dn1 das abzuarbeitende Programm (Datei mit verdichtetem P-Kode) kennzeichnet und nachfolgend noch Dateinamen folgen koennen, die waehrend der Abarbeitung von Dn1 erzeugt oder benutzt werden (maximal 10 Stueck). CONIN und CONOUT koennen durch Angabe nur des Kommas vereinbart werden (siehe nachfolgendes Beispiel).

Beispiel:

```
%PRUN TESTBSP.POBJ,IN1,,OUT1
```

arbeitet das Programm TESTBSP.POBJ ab, dieses Programm korrespondiert mit drei Dateien (1. Datei: IN1, 2. Datei: die Konsole, 3. Datei: OUT1).
Weitere Informationen zu Ein/Ausgabedateien und der Verbindung von Quellprogramm-Dateinamen und UDOS-Dateinamen werden im Abschn. 6.9. gegeben.
Um den Uebersetzungsprozess zu vereinfachen, wird die Benutzung einer UDOS-Kommandodatei empfohlen. Beispiele fuer solche Dateien sind:

```
V;PRUN PASCAL,#1,,TEMP.T,TEMP.P;B
ECHO PASM?-->bel.Taste oder Quit?-->ESC-Taste
PAUSE;V;PRUN PASM,TEMP.P,,TEMP.T,#1.POBJ;DELETE TEMP.* Q=N;B
```

```
V;PRUN PASCAL,#1,#1.L,TEMP.T,TEMP.P
PRUN PASM,TEMP.P,,TEMP.T,#1.POBJ
DELETE TEMP.P TEMP.T Q=N;B
```

```
V;PRUND PASCAL,#1,,TEMP.T,TEMP.P
PRUND PASM,TEMP.P,,TEMP.T,#1.POBJ;
DELETE TEMP.P TEMP.T Q=N;B
```

Wichtig dabei ist, dass das DO-Kommando auf die hoechstmoegliche Adresse im System gelegt wird !

5.3. Logische E/A-Einheiten

Der Interpreter benutzt folgende logische E/A-Einheiten:

1 (CONIN)	wenn in Kommandozeile angegeben
2 (CONOUT)	" " " "
5	das zu interpretierende Programm
5 - 20	Anwenderprogrammdateien

Die logischen Einheiten 5 bis 20 sind fuer Anwenderdateien verfuegbar. Sie werden alle benutzt, wenn in der PROGRAM-Anweisung 16 Dateien angegeben sind und alle mit UDOS-Dateinamen verbunden sind (d.h. nicht mit der Konsole). Die logischen Einheiten werden von 5 fortlaufend zugewiesen. Bsp: Falls drei Dateien angegeben sind und nur die mittlere mit der Konsole verbunden wurde, so werden fuer die restlichen Dateien die logischen Einheiten 5 und 7 verwendet.

6. Definierte Datentypen

6.1. Char

Der Standardtyp CHAR ist definiert als:

```
TYPE CHAR = MINCHAR..MAXCHAR;
```

wobei ORD(MINCHAR)=0 und ORD(MAXCHAR)=127. Variablen vom Typ CHAR belegen 16 Bit (high-Byte=0, low-Byte=ASCII-Zeichen).

6.2. Integer

Der Standardtyp INTEGER ist definiert als:
TYPE INTEGER = -32766..32767;
Integergroessen sind in 16-Bit-Zweierkomplementarithmetik implementiert.

6.3. Real

Das Programm PRUN enthaelt ein mathematisches Paket zur Verarbeitung von reellen Zahlen. Dieses Paket arbeitet mit einer 32-Bit-Gleitkommadarstellung; duales GK-Paket (4 Byte):

BYTE1	BYTE2	BYTE3	BYTE4
VZ-Bit 0=+ 1=-	23-Bit-Mantisse (normalisiert) d.h., Exponent wird so eingestellt, dass MSB der Mantisse 1 ist, oder die Mantisse leer ist.		8-Bit-Exponent (excess 128) d.h. realer Exponent = Exponentenfeld - 128

Eine andere Version: PRUND arbeitet mit einer 64-Bit-Gleitkommadarstellung; BCD GK-Paket (8 Byte):

BYTE1	BYTE2	BYTE3	BYTE4	...	BYTE8
VZ-Bit 0=+ 1=-	13-BCD-Ziffern-Mantisse (normalisiert) d.h., Exponent so eingestellt, dass das hoeherwertigste BCD-Zeichen ungleich 0 ist, oder ganze Mantisse leer ist.				Exponent (excess 128)

Falls die Anwenderprogramme viel Speicherbereich benoetigen und keine reellen Zahlen benutzen, dann sollte die Version PRUNI verwendet werden. (enthaelt selbe Funktionen wie PRUN und PRUND, ausser der Verarbeitung von reellen Zahlen)

ACHTUNG: Es muss die gleiche Version des Interpreters (PRUN, PRUND oder PRUNI) in allen drei Stufen der Uebersetzung/Abarbeitung verwendet werden, ansonsten treten Fehler auf.

6.4. Boolean

Ein logischer Wert wird intern in 16 Bit gespeichert:
high-Byte = 0
low-Byte = 0 fuer FALSE, 1 fuer TRUE

6.5. Zeiger

Eine Zeigervariable wird intern in 16 Bit gespeichert. Der Wert NIL wird durch 0 repraesentiert.

6.6. Mengen

Mengen werden in einer variablen Anzahl von Bytes, von 4 (2 Woerter) bis 512 (256 Woerter) gespeichert. Die Anzahl ist fuer jede Menge fixiert und wird bei der Mengenvereinbarung ermittelt. Die Anzahl der Bytes ist immer gerade. Vor der aktuellen Menge stehen zwei Byte, die die Anzahl der Woerter der Menge enthalten.
kleinste Menge: enthaelt max. 16 Bits/Elemente, wird durch zwei Woerter dargestellt, erste enthaelt Laenge (=1).
groesste Menge: enthaelt max. 4080 Bits/Elemente, wird durch 256 Woerter dargestellt, erste enthaelt Laenge (=255). 16 Bits pro Wort x 255 = 4080 Elemente
Somit muss die Kardinalzahl des Basistyps und der Bereich der Indexwerte der Menge zwischen 0 und 4079 liegen. Jedes Element in einer Menge wird durch ein Bit repraesentiert.

Beispiel:

```
VAR s: SET OF 0..23;
     .
     .
BEGIN
      s := [7,8];
     .
     .
```

s hat folgende Darstellung im Speicher:
(# ist Adresse von s)

```
#    : 0000 0000    high Byte der Laenge
#+1  : 0000 0010    Laenge = 2 Woerter
#+2  : 0000 0000
#+3  : 0000 0000
#+4  : 0000 0001    Bit 8 gesetzt
#+5  : 1000 0000    Bit 7 gesetzt
```

ACHTUNG: Es ist guenstiger Teilbereichstypen anstelle von Skalartypen bei der Mengenvereinbarung zu verwenden!

Beispiel:

```
VAR x : SET OF 0..79;        reserviert 12 Byte
VAR x : SET OF INTEGER;      reserviert 512 Byte
```

6.7. Skalar- und Teilbereichstypen

Wenn ein Datentyp aus einer aufgezaehlten Menge von Bezeichnern besteht, dann wird eine Variable dieses Typs im Speicher durch eine ganze Zahl dargestellt. Der Wert dieser Zahl ist die Ordinalzahl (Position) des Bezeichners, den sie repraesentiert.

6.8. Felder und Records

Felder und Records werden ab geraden Speicherplatzgrenzen (von Low- zu High-Adresse wachsend) abgelegt. Das Schluesselwort PACKED wird ignoriert.

6.9. Dateitypen

In einem PASCAL-Quellprogramm koennen maximal 16 Dateien zur gleichen Zeit eroeffnet sein. Es sind nur Character-Dateien gestattet. Die vom PASCAL-Programm benutzten Dateien muessen in der ersten Zeile des Programms aufgefuehrt werden, ihre Namen sind abzukuerzen. Zusaetzlich muessen alle Dateien (ausser INPUT und OUTPUT) vom Typ TEXT sein und durch eine RESET- oder REWRITE-Anweisung eroeffnet sein, bevor sie benutzt werden.
Es gibt zwei Moeglichkeiten eine Verbindung zwischen UDOS-Dateien und internen PASCAL-Dateien herzustellen. Sie unterscheiden sich durch den Typ der verwendeten RESET/REWRITE-Anweisung.

6.9.1. Verbindung von PASCAL- und UDOS-Dateien ueber Kommandozeile

Mit dieser Methode wird eine Verbindung entsprechend der angegebenen Reihenfolge zwischen den in der Kommandozeile aufgefuehrten Dateien und den internen PASCAL-Dateien hergestellt. Wenn in der Kommandozeile anstelle eines Dateinamens zwei Kommas dicht stehen, dann wird die entsprechende Datei mit dem Konsoltreiber verbunden. Beachte, dass eine Standard-RESET/REWRITE-Anweisung fuer Dateien die bereits eroeffnet sind ein Ruecksetzen des Zeigers auf den Beginn der Datei bedeutet (die Datei wird nicht abgeschlossen). Bei Ausgabedateien geht dann zusaetzlich der augenblickliche Inhalt verloren. Folgende Beispiele zeigen diese Variante der Dateiarbeit:

```
program BSP1(F);      (* BSP1: erstellen einer Datei durch *)
  var F : text;       (*       ein Pascal-Programm         *)
begin                 (* Variante 1: Dateiname wird ausser-*)
  rewrite(F);         (*       halb des Quellprogramms fix.*)
 writeln(F,'TEXT');(* PRUN BSP1A.POBJ,OUTDATEI             *)
end(*BSP1*).


programm COPY(I,O); (* Variante 1: Dateinamen werden      *)
  var I,O:text;     (* ausserhalb des Quellprogramms fix.*)
begin               (* PRUN BSP2A.POBJ,INDAT,OUTDAT      *)
  reset(I); rewrite(O);
  while not eof(I)
  do begin O^ := I^;
       put(O); get(I);
     end;
end(*COPY*).
```

6.9.2. Interne Angabe eines Dateinamens

Diese Methode der Verbindung einer PASCAL-Datei mit einer UDOS-Datei besteht in der Angabe des Dateinamens innerhalb des Quellprogramms durch:

RESET (F,STRING) oder REWRITE (F,STRING)

wobei F die interne Dateibezeichnung ist und STRING den Dateinamen festlegt. STRING muss eine konstante Zeichenkette oder Feld von Zeichen sein. Wichtig ist, dass der Dateiname durch ein Leerzeichen begrenzt wird (siehe Beispiele). Mit dieser erweiterten RESET/REWRITE-Anweisung koennen keine Dateien eroeffnet werden, die mit der Konsole verbunden sind. Dagegen ist es moeglich durch zusaetzliche erweiterte RESET/REWRITE Anweisungen Dateien zu schliessen und neue Dateien zu eroeffnen.

Beispiele:

```
program BSP1(F);     (*BSP1: erstellen einer Datei durch *)
  var F: text;       (*       ein Pascal-Programm        *)
begin                (* Variante 2: Dateinamen werden im *)
  rewrite(F,'OUTDATEI '); (* Quellprogramm fixiert       *)
  writeln(F,'TEXT:ABCD');
end(*BSP1*).


program COPY(I,O);  (* Variante 2: Dateinamen werden     *)
  var I,O: text;    (* im Quellprogramm fixiert          *)
begin
    reset(I,'INFILE '); rewrite(O,'OUTFILE ');
    while not eof(I)
    do begin O^:=I^; put(O); get(I);
       end;
end(*COPY*).
```

```
program BSP3(A,B);  (* Dateiname wird im Quellprogramm  *)
  var               (* durch Variable STRING gebildet  *)
    A,B     :text;
    I       :integer;
    STRING  :array[1..33] of char;
begin
     reset(A); I:=1;
     while not(eoln(A)) do
       begin read(A,STRING[I]); I:=I+1; end;
     STRING[I]:=' ';      (* Leerzeichen ist wichtig! *)
     rewrite(B,STRING);
     writeln(B,'diese Datei wurde gerade eroeffnet');
end(*BSP3*).
```

6.9.3. Hinweise zur Eroeffnung von Dateien

- Die Benutzung der erweiterten Form der RESET/REWRITE-Anweisung auf eine in einem hoeheren Niveau eroeffnete Datei ist nicht gestattet.

- Alle Dateien, die mit der selben logischen Einheit (d.h. mit dem selben PASCAL-Dateinamen) verbunden sind, muessen die gleiche Recordlaenge besitzen.

- Wenn eine Datei mit der Konsole verbunden wurde, dann kann sie nicht mehr mit der erweiterten Form der RESET-/REWRITE Anweisung eroeffnet werden.

7. Einschraenkungen und Erweiterungen

7.1. Einschraenkungen

Die vorliegende Compilerversion enthaelt folgende Einschraenkungen:

- Es sind nur PASCAL-Textdateien gestattet (character files).

- Prozeduren oder Funktionen koennen nicht als Parameter anderen Prozeduren oder Funktionen uebergeben werden.

- Eine GOTO-Anweisung zu einer Adresse ausserhalb einer Prozedur ist nicht gestattet.

- Die Standardprozeduren PACK, UNPACK und PAGE sind nicht implementiert. Letztere kann durch WRITE(F,CHR(12)); ersetzt werden, wobei F eine Datei ist.

- Bei der Ausgabe einer Zahl kann eine Feldlaenge angegeben werden. Bei reellen Zahlen der Laenge 32 Bit muessen jedoch mindestens 14 Zeichen; bei reellen Zahlen der Laenge 64 Bit mindestens 21 Zeichen ausgegeben werden. Eine kleinere Feldlaenge wird ignoriert. Bei der Ausgabe von reellen Zahlen wird der zweite Datenfeldlaengenparameter ignoriert.

- Prozeduren sollten nicht mehr als 400 bis 500 Quellzeilen besitzen (d.h. nicht mehr als 4KByte umfassen).

- Kommentare werden anstelle von "{" und "}" durch die Zeichenpaare "(*" und "*)" gekennzeichnet.
  Bsp: (* dies ist ein Kommentar *)

- Die Prozedur-Verschachtelungstiefe ist auf 7 begrenzt.

- Eine PASCAL-Quellzeile kann maximal 80 Zeichen enthalten.

- Bereichs- und Indextests sind nicht implementiert.

- Arithmetischer Ueberlauf wird nicht angezeigt.

7.2. Erweiterungen

Die implementierten Erweiterungen enthalten vier neue Standardprozeduren (TRAP, EXIT, MARK, RELEASE). Diese und weitere Moeglichkeiten werden nachfolgend erlaeutert.

7.2.1. TRAP-Funktion

Die Funktion TRAP (I: integer; VAR R: bel. Typ) wurde in die Menge der vordefinierten Prozeduren mit aufgenommen, um den Aufruf eines externen Assemblerprogramms vom PASCAL-Programm aus zu ermoeglichen. Der erste Parameter wird vom Compiler als Funktionskode benutzt. Der zweite Parameter (bel. Typ einschliesslich array oder record) kann durch das externe Assemblerprogramm vor Rueckkehr ins PASCAL-Programm modifiziert werden.
Es koennen beliebig viele externe Assemblerroutinen aufgerufen werden. Der erste Parameter dient dabei zur Unterscheidung, welche Routine aufgerufen wird (I muss groesser 0 sein !). Die Verbindung zu den externen Assemblerroutinen wird ueber ein kleines Startprogramm realisiert, das vor Start des PASCAL-Systems abgearbeitet werden muss.

Beispiel:

```
        PSTART    EQU  4000H      ;PASCAL-System Startadr.
        USER      LD   HL,PTRTAB
                  JP   PSTART
        PTRTAB    DEFW PROG1      ;Startadressen der Ass.-
                  DEFW PROG2      ;programme
                       .
                       .
                  DEFW PROGn
```

Der erste Parameter der TRAP-Anweisung entspricht der Position in der Startadressentabelle der Assemblerprogramme.
TRAP (2,X);
uebergibt die Steuerung an Assemblerprogramm PROG2 im oberen Beispiel.

Beim Schreiben von Assemblerprogrammen sollten folgende Aspekte beruecksichtigt werden:

- Beim ersten Aufruf des Assemblerprogramms enthaelt das Registerpaar HL die Adresse des High-Byte des Zeigers auf den zweiten Parameter ("X" im oberen Beispiel). Das bedeutet, nach dem Befehl INC HL zeigt das Registerpaar HL auf das Low-Byte des Zeigers.

- Die Assemblerroutine muss mit einem RET-Befehl abgeschlossen werden. Vorher muss der Akkumulator gesetzt werden, um eine fehlerfreie oder fehlerhafte Aktion anzuzeigen.
  A=0 bedeutet fehlerfrei
  A=n zeigt Fehler an
  (entspricht EXIT(A), wobei A Inhalt des Akkumulators)

- Der fuer den Nutzer aktuelle Stackbereich ist der Stackbereich des Interpreters (200H Byte reserviert). Es gibt daher keine Garantie, ob genuegend Platz fuer den Anwender zur Verfuegung steht. Da der Interpreter keine Tests auf Stackueberlauf durchfuehrt, wird dem Anwender empfohlen, bei Ausfuehrung einer externen Assemblerroutine einen eigenen Stackbereich zu benutzen.

Nach Uebersetzen der Assemblerprogramme sollten sie auf die hoechst moeglichen Adressen gelegt werden (siehe Abschn. 2). Das dazugehoerige PASCAL-Programm ist bis zum Post-Prozessor wie vorher beschrieben zu uebersetzen. Zum Aufruf des Programms ist dann aber folgende Kommandozeile zu verwenden:

```
%PRUN,USER BSP.POBJ ,Dateinamen
```

wobei USER der Name der Anwender-Assembler-Prozedurdatei und BSP.POBJ der Dateiname des verdichteten P-Kodes ist. Optional koennen noch weitere Dateinamen fuer Ein/Ausgaben (wie im Abschn. 5.2. beschrieben) folgen.

Beispiel 1:
In diesem Beispiel wird zum Wert von 'X' 15 addiert.

PASCAL-Programm:

```
program P(input,output);
  var X : integer;
begin
     read(X); TRAP(1,X); writeln('X+15=',X);
end.
```

Assemblerprogramm:

```
PSTART    EQU  4000H      ;PASCAL-STARTADRESSE

USR       LD HL,PTRTABLE
          JP PSTART

PTRTABLE  DEFW RTN1

RNT1      LD D,(HL)
          INC HL
          LD E,(HL)
          EX DE,HL
          LD A,15
          INC HL          ;ZEIGT AUF LOW-BYTE VON X
          ADD A,(HL)
          LD (HL),A
          JR NC,CONTIN    ;MUSS HIGH-BYTE INCR. WERDEN ?
          DEC HL          ;JA
          INC (HL)
CONTIN    LD A,0          ;KENNZEICHNUNG FEHLERFREI
          RET
```

Beispiel 2:
Im folgenden Beispiel werden zwei Zeichenketten auf Gleichheit ueberprueft und das Ergebnis in 'EQUAL' abgelegt.

PASCAL-Programm:

```
program P(input,output);
  var R : record
          EQUAL : boolean;
          STR1  : array[1..10] of char;
          STR2  : array[1..10] of char;
          end;
      I : integer;
begin
     for I:=1 to 10 do
          read(R.STR1[I]);
     readln;
     for I:=1 to 10 do
          read(R.STR2[I]);
     readln;
     trap(1,R); write('die Zeichenketten sind');
     if not R.EQUAL then write('nicht ');
     writeln('gleich.');
end.
```

Assemblerprogramm:

```
PSTART    EQU  4000H
USR       LD HL,PTRTABLE
          JP PSTART
PTRTABLE  DEFW RTN1
RTN1      LD D,(HL)
          INC HL
          LD E,(HL)        ;DE ZEIGT AUF RECORD R (EQUAL)
          PUSH DE          ;RETTEN ADRESSE VON EQUAL
          INC DE
          INC DE           ;ZEIGT AUF STR1
          LD H,D
          LD L,E
          LD BC,20         ;ADDITION GROESSE VON STR1
          ADD HL,BC        ;HL ZEIGT AUF STR2
          LD B,10          ;10 MAL VERGLEICHEN
LOOP      INC HL           ;UEBERGEHEN HIGH-BYTES
          INC DE
          LD A,(DE)
          CP (HL)
          JR NZ,OUT        ;NZ: UNGLEICH
          INC HL
          INC DE
          DJNZ LOOP
OUT       LD A,0           ;FEHLERFREIE AKTION
          POP HL           ;ADRESSE VON EQUAL
          LD (HL),A        ;HIGH-BYTE IST NULL
          INC HL
          LD (HL),A        ;EQUAL FALSE SETZEN
          JR NZ,OUT2
          LD (HL),1        ;EQUAL TRUE SETZEN
OUT2      RET
```

7.2.2. EXIT-Funktion

Die Funktion EXIT(I:integer) wurde in die Menge der vordefinierten Prozeduren mit aufgenommen. EXIT beendet das PASCAL-Programm und kann an beliebiger Stelle im Programm stehen. Das niederwertige Byte des Wertes der Integergroesse wird vor Beendigung in den Akkumulator gebracht.

7.2.3. Andere Erweiterungen

Andere Spracherweiterungen sind:

(1) neue Standardprozeduren MARK und RELEASE, die folgendes leisten:
MARK(P) Markiert den Heap-Bereich auf die angegebene Position (P ist Zeigertyp).
RELEASE(P) Gibt die Bereiche, die durch neue Anweisungen seit der entsprechenden MARK(P) Anweisung belegt wurden wieder frei.
Die Variable P von MARK(P) sollte bis nach Ausfuehrung der entsprechenden RELEASE(P) Anweisung nicht geaendert werden.

(2) Zusaetzlich zu den Standardprozeduren RESET und REWRITE besteht die Moeglichkeit, ueber die erweiterte Form dieser Prozeduren Dateien zu eroeffnen, deren Namen innerhalb des PASCAL-Programms festgelegt werden. (weitere Informationen siehe Abschn. 6.9.)

(3) Beim Lesen eines Zeichens fuehrt der Compiler, wenn es nicht notwendig ist, keinen Vorzugriff in die Puffervariable f aus (wobei f eine PASCAL-Datei ist). Deshalb braucht der Nutzer das naechste Zeichen nicht zu liefern, bevor es benoetigt wird.

(4) Die CASE-Anweisung kann einen OTHERWISE-Zweig enthalten der ausgefuehrt wird, wenn keine Uebereinstimmung mit den vorliegenden Werten des Selektors festgestellt wird. Die Marke OTHERWISE und die dazugehoerigen Anweisungen muessen die letzte innerhalb der CASE-LABEL-Liste sein.

(5) Falls der Stackpointer groesser als der Heappointer wird (siehe Abschn. 2.2.), dann erfolgt die Ausgabe der Nachricht "STACK/HEAP COLLISION" und die Ausfuehrung wird beendet.

(6) Die Eingabe von ESCAPE bricht die Abarbeitung des Compilers und des Post-Prozessors ab.

(7) TAB-Zeichen werden vom Compiler akzeptiert und wie Leerzeichen behandelt.

Anhang A Compiler-Fehlernummern und Erlaeuterungen

Im folgenden sind die Compiler Fehlernummern und ihre Bedeutung aufgefuehrt. Die Fehlernummern und die entsprechenden Beschreibungen sind aus dem "PASCAL User Manual and Report" abgeleitet und an einigen Stellen veraendert worden.

Nummer	Erlaeuterung
1	Fehler beim einfachen Typ
2	Fehlender Bezeichner (identifier)
3	"PROGRAM" fehlt
4	")" fehlt
5	":" fehlt
6	Illegales Symbol
7	Fehler in der Parameterliste
8	"OF" fehlt
9	"(" fehlt
10	Fehler in Typvereinbarung
11	" " fehlt
12	" " fehlt
13	"END" fehlt
14	":" fehlt
15	Integer fehlt
16	"=" fehlt
17	"BEGIN" fehlt
18	Fehler im Vereinbarungsteil
19	Fehler in Feldliste
20	"," fehlt
21	";" fehlt
50	Fehler in Konstante
51	":=" fehlt
52	"THEN" fehlt
53	"UNTIL" fehlt
54	"DO" fehlt
55	"TO"/"DOWNTO" fehlt
56	"IF" fehlt
58	Fehler im Faktor
59	Fehler in Variable
101	Bezeichner wurde zweimal vereinbart
102	Untere Grenze groesser als obere
103	Bezeichner nicht aus der passenden Klasse
104	Bezeichner nicht vereinbart
105	Vorzeichen nicht erlaubt
106	Fehlende Zahl
107	Nicht kompatibler Unterbereichstyp
108	Datei nicht erlaubt
109	Typ darf nicht REAL sein
110	Feldtyp muss Skalar oder Unterbereich sein
111	Nicht kompatibel mit Feldtyp
112	Indextyp darf nicht REAL sein
113	Indextyp muss Skalar oder Unterbereich sein
114	Basistyp darf nicht REAL sein
115	Basistyp muss Skalar oder Unterbereich sein
116	Typfehler eines Standard-Prozedur-Parameter

117 Nicht befriedigter Vorwaertsverweis
118 Vorwaertsverweis eines Typbezeichners in Variablendeklaration
119 Vorwaerts deklariert; Wiederholung der Parameterliste nicht erlaubt
120 Funktionsergebnistyp muss Skalar, Unterbereich oder Zeiger sein
121 Datei-Wertparameter nicht erlaubt
122 Vorwaerts deklarierte Funktion; Wiederholung des Ergebnistyps nicht erlaubt
123 Fehlender Ergebnistyp in Funktionsvereinbarung
124 nur F-Format fuer REAL moeglich
125 Fehler im Typ der Standard-Funktionsparameter
126 Anzahl der Par. stimmt nicht mit Vereinbarung ueberein
127 Falsche Parametersubstitution
128 Typ des Ergebnisses der Funktion stimmt nicht mit Vereinbarung ueberein
129 Typkonflikt der Operanden
130 Ausdruck ist nicht vom gesetzten Typ
131 Nur Test auf Gleichheit ist erlaubt
132 Strenge Inklusion nicht erlaubt
133 Dateivergleich nicht gestattet
134 Falscher Typ der Operanden
135 Typ der Operanden muss BOOLEAN sein
136 Typ der Mengenelement muss Skalar oder Unterbereich sein
137 Typen der Mengenelemente nicht kompatibel
138 Typ der Variablen ist kein Feld
139 Indextyp ist nicht kompatibel mit Vereinbarung
140 Variablentyp ist kein RECORD
141 Typ der Variablen muss Datei oder Zeiger sein
142 Falsche Parametersubstitution
143 Falscher Typ der Schleifensteuervariablen
144 Falscher Typ des Ausdrucks
145 Typkonflikt
146 Zuweisung der Dateien nicht gestattet
147 Markentyp inkompatibel mit ausgewaehltem Ausdruck
148 Unterbereichsgrenzen muessen Skalar sein
149 Indextyp darf nicht INTEGER sein
150 Zuweisung zur Standardfunktion nicht gestattet
151 Zuweisung zur formalen Funktion nicht gestattet
152 Kein solches Feld in diesem RECORD
153 Typfehler beim Lesen
154 Aktueller Parameter muss Variable sein
155 Steuervariable muss entweder formal oder nicht lokal sein
156 Mehrfach definierte CASE-Marke
157 Zu viele CASE-Faelle in der CASE-Anweisung
158 Fehlen der entsprechenden Variantenvereinbarung
159 Real- oder String-Felder nicht erlaubt
160 Vorherige Deklaration war nicht vorwaerts
161 Wieder vorwaerts deklariert
162 Parametergroesse muss konstant sein
163 Fehlende Variante in Vereinbarung
164 Substitution Standardprozedur/-Funktion nicht erl.
165 Mehrfach definierte Marke
166 Mehrfach deklarierte Marke

167	Nicht deklarierte Marke
168	Nicht definierte Marke
169	Fehler in Basismenge
170	Fehlender Wertparameter
171	Standarddatei wurde neu deklariert
172	Nicht deklarierte externe Datei
174	Fehlende PASCAL-Prozedur oder Funktion
201	Fehler in REAL-Konstante: fehlendes digit
202	String-Konstante darf nicht Quelltextzeile ueberschreiten
203	INTEGER-Konstante ueberschreitet Bereich
250	Zu viele verschachtelte Bereiche von Bezeichnern
251	Zu viele verschachtelte Prozeduren und/oder Funktionen
253	Prozedur zu lang
255	Zu viele Fehler in dieser Quellzeile
256	Zu viele externe Verweise
257	Zu viele externe Groessen
258	Zu viele lokale Dateien
259	Ausdruck zu kompliziert
260	Datenstruktur zu lang, offset uebersteigt 32000
300	Division durch Null
302	Indexausdruck ist ausserhalb des Bereichs
303	Zuzuweisender Wert ist ausserhalb des Bereichs
304	Elementarausdruck ist ausserhalb des Bereichs
398	Implementationseinschraenkung

Anhang B Post-Prozessor Fehlernummern und Erlaeuterungen

Der Post-Prozessor kann waehrend der Abarbeitung Fehlernachrichten erzeugen. Diese Fehler sollten vor Aufruf des Interpreterprogramms korrigiert werden.

Fehler	Erlaeuterung
102	LOAD-Offset zu gross
104	STORE-Offset zu gross
105	Zeichenkette zu lang fuer Vergleich (=512 Byte)
106	LOAD-Adressen-Offset zu gross
110	INDIRECT-Offset zu gross
112 *	Nicht definierte Prozedur
113 *	Nicht definierte Standardprozedur
114	MOVE-Operand zu gross (=512 Byte)
118	Zu viele Marken
119 *	Mehrfache Markendefinition
125 *	Nicht korrekte Variablenaufstellung (ungerade Versch.)
127	Prozedurverschachtelungsniveau zu tief (7 Ebenen)
141 *	Ungueltiger Befehl
146	"NEW" Operand zu gross (=256 Byte)
160 *	Nicht def./falscher Operationskode

Die ohne "*" angefuehrten Fehlernummern zeigen einen implementationsbedingten Fehler an. In diesen Faellen kann der Nutzer nur durch Aenderungen im Quellprogramm, die die Einschraenkungen (siehe Abschn. 7.1.) beruecksichtigen, diese Situation vermeiden.

U 8 8 0 - F O R T R A N

Benutzerhandbuch

Inhaltsverzeichnis Seite

Seite

1. Einfuehrung

FORTRAN (FORmula TRANslator) ist eine allgemeine problemorientierte Programmiersprache. Die Syntax dieser Sprache fordert vom Anwender, dass er seine zu loesenden Probleme in einer Folge exakter Anweisungen formuliert. Diese Folge von derartigen Anweisungen wird als Quellprogramm bezeichnet, welches dann durch einen FORTRAN-Compiler in die Maschinensprache des Computers, auf dem das Programm ausgefuehrt werden soll, uebersetzt wird. Im folgenden Text wird die FORTRAN-Sprache speziell fuer Mikrorechnersysteme auf der Basis des Mikroprozessors U880 beschrieben. Diese FORTRAN-Variante hat gegenueber dem Standard-FORTRAN einige Abweichungen (Erweiterungen bzw. Einschraenkungen).

2. FORTRAN-Programmform

FORTRAN-Quellprogramme bestehen aus den Programmkomplexen:

Hauptprogramm (MAIN-Programme)

und wahlweise aus einer Anzahl von:

Unterprogrammen (SUB-Programme)

Ein Programmkomplex enthaelt nichtausfuehrbare (Vereinbarungs-) und ausfuehrbare Anweisungen. Fuer die Notation dieser Anweisungen duerfen nur bestimmte Zeichen verwendet werden. Die Zeilenstruktur hat ein bestimmtes Format.

2.1. Zeichenmenge

Die Zeichenmenge besteht aus den:

Buchstaben: A,B,C,...X,Y,Z

Ziffern: 0,1,3,...7,8,9

Bei den Buchstaben wird nicht zwischen Gross- und Kleinbuchstaben unterschieden.

Alphanumerische Zeichen koennen aus allen Buchstaben und Ziffern gebildet werden.

Fuer die Darstellung von Hexadezimalziffern werden die Ziffern 0,1,2...7,8,9 bzw. die Buchstaben A,B,...F verwendet.

Sonderzeichen:
- _ blank
- = Gleichheitszeichen
- + Pluszeichen
- - Minuszeichen
- * Stern
- / Schraegstrich

(oeffnende Klammer
) schliessende Klammer
, Komma
. Dezimalpunkt
' Apostroph

Fuer arithmetische Operationen sind die folgenden Sonderzeichen zugelassen.

+ Addition von positiven Werten
- Subtraktion von negativen Werten
* Multiplikation
/ Division

2.2. Zeilenstruktur

Eine Zeile besteht aus max. 80 Zeichen bzw. Spalten, die in vier Felder aufgeteilt sind.

1.	Spalten	1... 5	Marken(label)feld
2.	"	6	Fortsetzungszeilenfeld
3.	"	7...72	Anweisungsfeld
4.	"	73...80	Identifikationsfeld

Das Indentifikationsfeld kann fuer beliebige Informationen genutzt werden. Der Inhalt dieser Spalten wird vom Compiler ignoriert.

2.3. Zeilentypen

Es wird zwischen vier Zeilentypen unterschieden.

1. Kommentarzeile

   Die Spalte 1 enthaelt den Buchstaben C.
   Die Spalten 2...72 koennen fuer beliebige Kommentare genutzt werden.
   Dieser Zeilentyp hat keinen Einfluss auf das Quellprogramm.

2. Anweisungszeile

   Die Spalten 1...5 koennen eine Marke enthalten, um die Zeile zu kennzeichnen.
   Die Spalte 6 muss 0 oder blank enthalten.
   Die Spalten 7...72 enthalten die gesamte oder Teile einer Anweisung. Der Beginn der Anweisung muss nicht in Spalte 7 sein.

3. Fortsetzungszeile

   Sie wird verwendet, wenn fuer eine Anweisung mehr als eine Zeile benoetigt wird.
   Die Spalten 1...5 werden ignoriert, ausser Spalte 1, wenn diese ein C enthaelt (Kommentarzeile).
   Spalte 6 muss ein von 0 und blank verschiedenes Zei-

chen enthalten.
Die Spalten 7...72 enthalten die Fortsetzung der Anweisung.
Die Anzahl der Fortsetzungszeilen ist beliebig.

4. Endzeile

Um dem Compiler das physische Ende eines jeden Programmkomplexes mitzuteilen, muss die letzte Zeile eines Programmkomplexes eine END-Zeile sein.
Die Spalten 1...5 koennen eine Marke enthalten.
Die Spalte 6 muss 0 oder bhank enthalten.
Die Spalten 7...72 enthalten das Wort END.

2.4. Marken

In den Spalten 1...5 einer Anweisungszeile bzw. einer Endzeile kann eine Marke enthalten sein. Von anderen Anweisungen kann auf diese Marke verwiesen werden. Eine Marke ist eine natuerliche Zahl von 1...99999. Fuehrende Nullen oder blank sind ohne Bedeutung. Eine bestimmte Marke darf nur einmal im Markenfeld auftreten.

2.5. Anweisungen

Die Anweisungen werden eingeteilt in ausfuehrbare und nicht ausfuehrbare Anweisungen. Ausfuehrbare Anweisungen werden vom Compiler in Objektprogrammbefehle umgewandelt. Diese werden durch den Linkprozess in ladefaehige Maschinenkodebefehle umgesetzt und loesen dann bei der Abarbeitung Aktionen aus. Es gibt drei Arten ausfuehrbarer Anweisungen.

Ergibtanweisungen
Steueranweisungen
Ein/Ausgabeanweisungen

Nichausfuehrbare Anweisungen spezifizieren dem Compiler die Art und Anordnung von Daten, geben Informationen ueber Ein- und Ausgabeformate und Dateninitialisierung an das Objektprogramm waehrend des Ladens und der Ausfuehrung von Programmen. Es gibt fuenf Arten nichtausfuehrbarer Anweisungen.

Vereinbarungsanweisungen
Formatanweisungen
Dateninitialisierungsanweisungen
Anweisungsfunktionsanweisungen
Unterprogrammanweisungen

2.6 Datentypen

Die zu verarbeitenden Daten koennen Konstanten oder Variablen sein. Es gibt vier verschiedene Typen.

1. INTEGER

```
Art       : ganze Zahl
Laenge    : 2 Bytes
Bereich   : -32768...32767 oder
            -2**15...2**15-1      (** entspricht 'hoch')
```

```
Beispiele:     13
              +0064
             -32768
              1 385
                138.5        falsch
```

2. REAL

```
Art       : reelle Zahl, Gleitkommaformat
Laenge    : 4 Bytes, wobei das Byte 1 der Exponent ist,
            Byte 2,3,4 die Mantisse.
Bereich   : +-i.fE+-e
(allg.)     i,f,e  sind vorzeichenlose natuerliche Zah-
            len; E+-e entspricht 10**+-e;
            fuer e gilt -38<=e<=+38
Bereich   : +-10**-38...+-10**38 oder
(konkret) +-2**-127...+-2**127
Wert des :  80H: Exponent=0 (gesamte Zahl=0)
Exponenten <80H:     "    <0
           >80H:     "    >0
```

```
Beispiele:  -.f            -.485
             i.            380.
            +i.f          +5956.3
             -.fE-e         -0.45E-2
            +i.Ee          +98.E2
            -i.fEe          -8.5E3
```

3. DOUBLE PRECISION

```
Art       : reelle Zahl, Gleitkommaformat
Laenge    : 8 Bytes
Bereich   : wie REAL, jedoch doppelte Genauigkeit.
            fuer E ist D zu notieren
```

```
Beispiele: -375.60D3
             0.5D-4
            90D6
```

4. LOGICAL

```
Art       : logischer Typ, 'Ein-Byte'-Darstellung der
            Wahrheitswerte 'TRUE' und 'FALSE'
Laenge    : 1 Byte
```

Werte : FALSE=0
TRUE =-1, es wird jedoch jeder Wert, der ungleich Null ist, ebenso behandelt.

Dieser Typ kann auch als 'Ein-Byte' vorzeichenlose ganze Zahl im Bereich -128...127 verwendet werden. Die Anwendungsregeln sind die gleichen wie fuer den Typ INTEGER.

Fuer die Typbezeichnung gilt folgendes:

BYTE, INTEGER*1, LOGICAL*1, LOGICAL	gleichwertig
INTEGER*2, LOGICAL*2, INTEGER	"
REAL, REAL*4	"
DOUBLE PRECISION, REAL*8	

2.7. Speichereinheiten

Der fuer die verschiedenen Datentypen zum Abspeichern benoetigte Speicherplatz wird in Speichereinheiten eingeteilt. Eine Speichereinheit besteht aus 4 Bytes. Damit ergeben sich fuer die Datentypen folgende Speichereinheiten:

INTEGER	:	1/2 Speichereinheit
LOGICAL	:	1/4 "
REAL	:	1 "
DOUBLE PRECISION	:	2

2.8. Konstante

Konstante koennen:

1. den Datentyp INTEGER, REAL, DOUBLE PRECISION und LOGICAL haben.
2. Zeichenkettenkonstante sein
3. Hexadezimalkonstante sein.

Zeichenketten sind in zwei Arten moeglich:

1. In Apostroph eingeschlossene Zeichenfolge.

   Beispiel: 'INHALT'

   Wenn in der Zeichenfolge ein Apostroph vorkommt, dann muss er doppelt geschrieben werden.

2. Es werden die Anzahl der Zeichen der Zeichenfolge, der Buchstabe H und die eigentliche Zeichenfolge notiert. Leerzeichen (blank) zaehlen mit.

   Beispiel: 11HI H H A L T

Hexadezimalkonstanten werden dargestellt, indem die hexadezimale Zahl in Apostrophe eingeschlossen und ein Z oder X vorangestellt wird.

Beispiel: Z'8FE' oder X'8FE'

2.9. Variable

Variable koennen den Datentyp INTEGER, REAL, DOUBLE PRECISION und LOGICAL haben. Sie sind durch symbolische Namen zu kennzeichnen. Die Namen bestehen aus max. sechs alphanumerischen Zeichen, wobei das erste Zeichen ein Buchstabe sein muss.

Die Festlegung des Typs erfolgt explizit oder implizit.

1. explizite Festlegung

   Der Typ kann durch eine Typvereinbarung festgelegt werden. (s.Abschn.4.3.1.)

2. implizite Festlegung

   Wenn der Typ nicht explizit festgelegt wurde, dann stellen symbolische Namen, die mit I,J,K,L,M,N beginnen, INTEGER-Variable dar, alle anderen REAL-Variable.

Variablen koennen durch eine DATA-Anweisung (s.Abschn.4.3.5.) oder waehrend der Programmabarbeitung numerische Werte zugewiesen werden. Unter dem Wert einer Variablen versteht man den entsprechend des vereinbarten Typs interpretierten aktuellen Inhalt der zugehoerigen Speichereinheit. Zeichenketten koennen jedem Variablentyp zugewiesen werden. (s.Abschn.5.3.6.)

2.10. Felder und Feldelemente

Als Feld wird eine nach einem bestimmten Schema angeordnete Menge von Daten bezeichnet. Das Schema der Datenanordnung kann geometrisch betrachtet drei Formen haben:

1. die Daten sind streckenfoermig angeordnet
2. " " " rechteckfoermig "
3. " " " quaderfoermig "

Die jeweilige Form hat die Bezeichnung Dimension 1, 2 oder 3. Die einzelnen Daten des Feldes werden als Feldelemente bezeichnet. Das Feld wird durch einen symbolischen Namen in der gleichen Weise wie eine Variable beschrieben, ausgenommen, dass ein Feldname durch eine Feldvereinbarung festgelegt werden muss. Den Feldelementen koennen durch eine DATA-Anweisung oder waehrend der Programmabarbeitung numerische Werte zugewiesen werden.

2.11. Indizierung

Auf ein bestimmtes Feldelement wird durch die Indizierung des Feldnamens Bezug genommen. Fuer den Begriff 'Feldelement' kann deshalb auch der Begriff 'indizierte Variable' verwendet werden. Der Index eines Feldelementes besteht aus ein, zwei oder drei in Klammern stehenden und durch Komma

getrennten Indexausdruecken. Die Anzahl der Indexausdruecke muss mit der in der Feldvereinbarung festgelegten Dimension uebereinstimmen. Das trifft nicht zu, wenn die EQUIVALENCE-Anweisung verwendet wird. (s.Abschn.4.3.4.) Indizes selbst duerfen nicht indiziert werden.

Felder koennen folgendermassen indiziert werden:

Beispiele:		
	X(I)	eindimensional
	B(L-2)	"
	K(20)	"
	Y(5*J-4,8)	zweidimensional
	S(U,V,W)	dreidimensional

3. Ausdruecke

Die Ausdruecke in FORTRAN sind vergleichbar mit den Formeln in der Mathematik. Es gibt arithmetische und logische Ausdruecke. Ein Ausdruck besteht aus einem einzigen Operanden oder aus einer Kette durch Operatoren verknuepfter Operanden.

3.1. Arithmetische Ausdruecke

Folgende Operanden sind moeglich:

Konstante
Variable
Feldelement
Funktion

Folgende Operatoren sind moeglich:

\+ Addition
\- Subtraktion
* Multiplikation
/ Division
** Potenzierung

Klammern koennen verwendet werden. Zu beachten ist aber, dass die Anzahl der oeffnenden gleich der Anzahl der schliessenden Klammern ist.

Die Reihenfolge bei der Berechnung von arithmetischen Ausdruecken ist folgende.

1. Klammerausdruecke (wenn Schachtelungen, dann die innerste zuerst)
2. Funktionsaufruf
3. Potenzierung
4. Multiplikation und Division
5. Addition und Subtraktion

Die Berechnung bei Operationen auf gleicher Ebene verlaeuft von links nach rechts.

Beispiel: Der Berechnungsausdruck fuer die Oberflaeche des Zylinders ist:

```
2*(PI*(D/2)**2)+PI*D*H
```

Die Reihenfolge der Berechnungsschritte ist:
(E1,E2...E6 sind die Zwischenschritte)

```
D/2  =E1
E1**2=E2
PI*E2=E3
2*E3 =E4
PI*D =E5
E5*H =E6
E4+E6=Wert des Ausdrucks
```

Eine direkte Potenzierung des Exponenten - also X**Y**Z - ist nicht statthaft.

Dieser Ausdruck ist so zu notieren:

(X**Y)**Z oder X**(Y**Z)

Der Datentyp des Ausdrucks wird durch die Datentypen der Operanden bestimmt. Sind diese Operanden nicht alle vom gleichen Typ, dann wird der Typ des Ausdrucks durch den Operanden mit dem hoechsten Typ bestimmt.

Die Datentyphierarchie vom hoechsten zum niedrigsten ist:

DOUBLE PRECISION, REAL, INTEGER, LOGICAL

3.2. Logische Ausdruecke

Logische Ausdruecke liefern die Werte .TRUE. oder .FALSE. Es gibt zwei Arten logischer Ausdruecke.

1. Eine Verknuepfung von logischen Operanden mit logischen Operatoren

   Logische Operanden sind: logische Konstante
   " Variable
   " Feldelemente
   " Funktion
   Vergleichsausdruecke

   Logische Operatoren sind: .AND. , .OR. , .XOR. , .NOT.

   Wenn A und B logische Ausdruecke sind, dann gilt fuer die Definition der logischen Operatoren:

A	B	.NOT.A	A.AND.B	A.OR.B	A.XOR.B
.FALSE.	.FALSE.	.TRUE.	.FALSE.	.FALSE.	.FALSE.
.FALSE.	.TRUE.	.TRUE.	.FALSE.	.TRUE.	.TRUE.
.TRUE.	.FALSE.	.FALSE.	.FALSE.	.TRUE.	.TRUE.
.TRUE.	.TRUE.	.FALSE.	.TRUE.	.TRUE.	.FALSE.

Logische Operatoren duerfen nicht direkt nebeneinanderstehen, ausgenommen der zweite Operator ist .NOT.

Beispiel: A.AND..NOT.B richtig
A.AND..OR.B falsch

2. Eine Verknuepfung von arithmetischen Ausdruecken mit Vergleichsoperatoren (Vergleichsausdruck)

Vergleichsoperatoren sind:

.LT.	kleiner als
.LE.	kleiner oder gleich
.EQ.	gleich
.NE.	nicht gleich
.GT.	groesser als
.GE.	groesser oder gleich

Beispiel: A.EQ.B
(A**J).GT.(ZAP*(RHO*TAU-ALPHA))

Die Reihenfolge bei der Berechnung von logischen Ausdruecken ist folgende:

1. Klammerausdruecke (wenn Schachtelungen, dann die innerste)
2. Funktionsaufrufe
3. Potenzierung
4. Multiplikation und Division
5. Addition und Subtraktion
6. .LT.,.LE.,.EQ.,.NE.,.GT.,.GE.
7. .NOT.,.AND.,.OR.,.XOR.

In logischen Ausdruecken koennen anstelle von Integerkonstanten Zeichenketten- und Hexadezimalkonstanten auftreten. Sie koennen deshalb nur 2 Bytes lang sein.

Beispiel: .NOT.A.GT.B
A.LT.X.AND.X.LE.B
A.AND..NOT.B.OR.C
SIN(X).LT.COS(X)
A.AND.B.OR..NOT.A.AND..NOT.B
A.OR.B*(I.GT.J) falsch

4. Anweisungen

4.1. Ergibtanweisungen

Syntax: Variable oder Feldelement=Ausdruck

Bei der Ausfuehrung wird der Wert des Ausdrucks berechnet und der Variablen oder dem Feldelement zugeordnet. Der Datentyp der linken Seite muss nicht mit dem der rechten Seite uebereinstimmen. Vor der Zuordnung des Ausdruckswertes zur linken Seite wird eine Typkonvertierung durchgefuehrt. Dabei koennen Teile der Information verloren gehen.

Bei Ergibtanweisungen muss die linke Seite einschliesslich des Gleichheitszeichens auf einer Zeile stehen.

Beispiel:
```
      A(5,3)=
     1     DO(7,2)+H
```

4.2. Steueranweisungen

Sollen in einem FORTRAN-Programm die ausfuehrbaren Anweisungen nicht in der gleichen Reihenfolge abgearbeitet werden, in der sie physisch im Quellprogramm notiert wurden, dann kann das mit Steueranweisungen erfolgen.

4.2.1. Sprunganweisungen

Es gibt 3 Typen von Sprunganweisungen:

1. Unbedingte Sprunganweisung

   Syntax: GOTO k

   k ist die Marke einer ausfuehrbaren Anweisung im selben Programmkomplex. Das Programm wird mit der Anweisung mit der Marke k fortgesetzt.

   Beispiel:
```
          GOTO 19
        7 A=B+C
          ,
          ,
       19 S=S+A
```

2. Berechnete Sprunganweisung

   Syntax: GOTO (k1,k2,...,kn),j

   k1,k2,... sind verschiedene Marken von ausfuehrbaren Anweisungen im selben Programmkomplex; j ist eine Integervariable im Bereich 1<=j<=n. Die Abarbeitung der berechneten Sprunganweisung bewirkt, dass als naechste die Anweisung abgearbeitet wird, welche durch die, in der Reihenfolge von links nach rechts gezaehlt, j-te Marke der Markenliste markiert wird. Die Bedingung ist, dass die Integervariable j im Bereich 1<=j<=n ist. Ist das nicht der Fall, dann wird mit der Anweisung nach GOTO fortgesetzt.

Beispiel:
```
      LABEL=4
      ,
      GOTO (7,11,18,20,25),LABEL
   20 X=Y+Z
```

3. Gesetzte Sprunganweisung

Syntax: GOTO j,(k1,k2,...,kn)

k1,k2,... sind verschiedene Marken von ausfuehrbaren Anweisungen. Die Reihenfolge ist unwichtig. j ist eine Integervariable. Ihr Wert ist als Marke zu interpretieren.
Dieser Sprunganweisung muss die folgende Anweisung vorausgehen:

ASSIGN i TO j

Die Marke i wird der Variablen j zugewiesen. Entspricht diese Marke einer der Marken k1,k2,... der **Sprunganweisung**, dann wird dorthin verzweigt.

Beispiel:
```
      ASSIGN 20 TO LABEL
      ,
      GOTO LABEL, (5,10,15,20,25)
   20 X=X*(A+B)
```

4.2.2. Bedingte Sprunganweisungen

Es gibt 2 Typen von bedingten Sprunganweisungen:

1. Arithmetische bedingte Sprunganweisung

Syntax: IF (A) m1,m2,m3

A ist ein arithmetischer Ausdruck. m1... sind verschiedene oder gleiche Marken von ausfuehrbaren Anweisungen im selben Programmkomplex. Bei welcher durch die Marken m1... markierten Anweisungen das Programm fortgesetzt wird, ist abhaengig vom Wert des Ausdrucks.

```
Ausdruck <0 :   Fortsetzung bei Marke m1
    "    =0 :        "      "    "  m2
    "    >0 :        "      "    "  m3
```

Beispiel: IF(X-3.) 1,1,2

2. Logische bedingte Sprunganweisung

Syntax: IF(u),s

u ist ein logischer Ausdruck. s eine ausfuehrbare

Anweisung, ausser einer weiteren logischen bedingten Anweisung bzw. einer Laufanweisung (s.Abschn.4.2.3.) Mit welcher Anweisung das Programm fortgesetzt wird, ist abhaengig vom Wert des logischen Ausdrucks.

Ausdruck =.TRUE. Fortsetzung mit s
" =.FALSE. " " Anweisung nach s

Wenn s eine Ergibtanweisung (V=A) ist, dann muss entweder 'V=' auf der selben Zeile wie IF(u) stehen oder auf der Fortzetzungszeile, wobei dann hinter IF(u) nur Leerzeichen stehen duerfen.

Beispiel: IF(Q.AND.R) I=J

oder

```
IF(Q.AND.R)
1   I=J
```

4.2.3. Laufanweisung

Die Laufanweisung ermoeglicht das wiederholte Abarbeiten einer Folge von Anweisungen, des sogenannten DO-Bereiches.

Syntax: DO k i=m1,m2,m3

k ist die Marke der letzten ausfuehrbaren Anweisung des DO-Bereiches. i (positive ganzzahlige oder logische Variable) ist die Laufvariable mit dem Anfangswert m1 (ganzzahlige Konstante oder Variable), m2 der Endwert; m3 die Schrittweite. Ist m3=1, dann kann m3=1 weggelaasen werden.

Die Abarbeitung einer Laufanweisung loest folgende Aktionen aus:

1. Der Laufvariablen i wird der Anfangswert m1 zugewiesen.
2. Der DO-Bereich wird bis zur letzten ausfuehrbaren Anweisung abgearbeitet und der Wert der Laufvariablen um den Wert der Schrittweite m3 erhoeht.
3. Ist der Wert der Laufvariablen nach der Erhoehung kleiner oder gleich dem zugehoerigen Endwert m2, dann werden die Aktionen von 2. wiederholt.
4. Ist der Wert der Laufvariablen groesser als der Endwert, dann ist die Laufanweisung abgearbeitet.

Die letzte ausfuehrbare Anweisung der Laufanweisung darf keine Anweisung sein, die aufgrund ihrer Funktion keine ordnungsgemaesse Abarbeitung der Laufanweisung zulaesst. Das sind:

Arithmetisches IF
GOTO
RETURN
PAUSE
STOP
ein weiteres DO

Ist die letzte Anweisung ein logisches IF, dann wird beim Wert .FALSE. die naechste Anweisung abgearbeitet; es werden also die Anweisungen des DO-Bereiches wiederholt. Ist der Wert .TRUE., dann wird die Anweisung des logischen IF ausgefuehrt und dann weiter wie bei .FALSE. Fuer die Art der Anweisung des logischen IF gilt das gleiche wie fuer die letzte Anweisung des DO-Bereiches.

Die Anweisungen eines DO-Bereiches, die logisch zwischen DO und der letzten Anweisung notiert werden, muessen nicht alle physisch zwischen DO und letzter Anweisung liegen. Sie koennen auch ausserhalb des physischen DO-Bereiches - im erweiterten Bereich - liegen.

Aus einem DO-Bereich darf herausgesprungen werden. Hineingesprungen werden darf nur in Form des Ruecksprungs aus dem erweiterten Bereich.

Eine Ineinanderverschachtelung von Laufanweisungen ist moeglich, wobei die innere Laufanweisung vollstaendig in der auesseren enthalten sein muss. Die letzte Anweisung kann von mehreren gemeinsam genutzt werden. Gesprungen werden darf zu dieser letzten Anweisung jedoch nur von der innersten Laufanweisung (oder vom erweiterten Bereich)

Beispiel:

```
      DO 50 I=10,327,3
        ,
        ,
      IF(U7-C*C)20,15,31
30      ,
50    A(I)=B(I)+C
        ,
        ,
20    C=C-0.5
      GOTO 50
31    C=C+0.125
      GOTO 30
        ,
        ,
15    C2=C2+.005
```

4.2.4. Leeranweisung

Die Abarbeitung der Leeranweisung loest keine Aktionen aus. Als naechste Anweisung des Programmkomplexes wird die physisch der Leeranweisung folgende Anweisung abgearbeitet.

Syntax: CONTINUE

Die Leeranweisung kann an beliebiger Stelle in das Programm eingefuegt werden. Sie dient vorwiegend zur Bezeichnung des Endes einer Laufanweisung, um die der letzten Anweisung des DO-Bereiches auferlegten Einschraenkungen aufzuheben.

```
Beispiel:   DO 100 I=1,25,2
                ,
                ,
            IF(A-B(I))100,3,3
        100 CONTINUE
                ,
                ,
          3 A=8+C**2
```

4.2.5. Pauseanweisung

Durch die Pauseanweisung kann die Ausfuehrung eines Programms unterbrochen werden.

Syntax: PAUSE C

Das optionale C ist eine Zeichenkette von maximal 6 Zeichen.

Wird bei der Programmabarbeitung PAUSE C erreicht, dann wird das Programm unterbrochen, und C erscheint auf dem Bildschirm.

Durch Bedienerkommando kann ueber die weitere Abarbeitung des Programms entschieden werden.

Programmfortsetzung: Betaetigen irgendeiner Taste
Programmabbruch : Betaetigen der Taste 'T'

4.2.6. Haltanweisung

Die Haltanweisung ist das logische Ende eines Programms.

Syntax: STOP C

Das optionale C ist eine Zeichenkette von maximal 6 Zeichen.

Wird bei der Programmabarbeitung STOP C erreicht, dann wird das Programm beendet, und C erscheint auf dem Bildschirm.

4.3. Vereinbarungsanweisungen

Vereinbarungsanweisungen sind nichtausfuehrbare Anweisungen. Sie liefern dem FORTRAN-Compiler Informationen, die zur Uebersetzung des Programms benoetigt werden (z.B. Datentypen von Variablen, Feldern und Felddimensionen, Speicherplatzzuordnungen festlegen und variablen Daten Anfangswerte zuordnen) Deshalb muessen diese Vereinbarungen am Anfang des Programms bzw. vor der ersten ausfuehrbaren Anweisung notiert werden.
Vor den Vereinbarungsanweisungen duerfen nur PROGRAMM-, SUBROUTINE-, FUNCTION- oder BLOCK DATA-Anweisungen stehen.

Es wird zwischen sechs Arten von Vereinbarungsanweisungen

unterschieden:

1. Typ	-	Vereinbarungsanweisung
2. Feld	-	"
3. Speicherplatz	-	"
4. Aequivalenz	-	"
5. Datenanfangswert	-	"
6. External	-	

4.3.1. Typvereinbarungsanweisungen

Typvereinbarungen dienen der expliziten Typzuweisung von Konstanten und Variablen. Fuer die Typen INTEGER und REAL ist die explizite Typvereinbarung nur dann notwendig, wenn die implizite (durch den ersten Buchstaben der Bezeichnung s.Abschn.2.9.) nicht zutrifft. Fuer die anderen Typen ist das die einzige Moeglichkeit, Bezeichnungen zu vereinbaren.

Syntax: t v1,v2,...

t ist ein Typ aus der unter Abschn.2.6. angegebenen Menge von Typbezeichnungen, also BYTE, INTEGER*1 usw. Die v1,v2,... sind Namen von Variablen, Feldern, Feldelementen oder Funktionen.

Beispiel: Die Variablen:
A,B,C sollen vom Typ ganzzahlig sein
I,J,K " " " " reell "
R,S,T " " " doppelgenau "

```
INTEGER A,B,C
REAL I,J,K
DOUBLE PRECISION R,S,T
```

4.3.2. Feldvereinbarungsanweisungen

Felder koennen durch Anwendung der Typvereinbarungsanweisung (s.Abschn.4.3.1.) vereinbart werden.

Beispiel: INTEGER A(3)

A ist der Feldname des eindimensionalen Feldes vom Typ INTEGER. Die Anzahl der Feldelemente betraegt 3.
Die Anordnung der Elemente im Speicher ist folgende :

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

REAL LI(3,2)

LI ist der Feldname des zweidimensionalen Feldes vom Typ REAL. Die Anzahl der Feldelemente betraegt 6.
Die Anordnung der Elemente im Speicher ist folgende:

```
LI(1,1), LI(2,1), LI(3,1)
LI(1,2), LI(2,2), LI(3,2)

LOGICAL TA(3,2,2)
```

TA ist der Feldname des dreidimensionalen Feldes vom Typ LOGICAL. Die Anzahl der Feldelemente betraegt 12.
Die Anordnung der Elemente im Speicher ist folgende:

```
TA(1,1,1), TA(2,1,1), TA(3,1,1)
TA(1,2,1), TA(2,2,1), TA(3,2,1)
TA(1,1,2), TA(2,1,2), TA(3,1,2)
TA(1,2,2), TA(2,2,2), TA(3,2,2)
```

Ein Element eines mehrdimensionalen Feldes kann auch mit einem Index (Feldnummer) bezeichnet werden. So ist z.B. TA(3,2,1)=TA(6), TA(2,1,2)=TA(8)

Allgemein gilt fuer die Feldnummer I des Elements A(i,j,k) eines durch A(l,m,n) vereinbarten Feldes :

I=i+l(j-1)+l-m(k-1)

Ist das Feld zweidimensional, dann ist in der Formel k durch 1 zu ersetzen.

Ist der Typ eines Feldes bereits implizit festgelegt, dann ist die folgende Anweisung zu verwenden:

Syntax: DIMENSION f1,f2,...fn

Die f1,f2... sind Feldbezeichnungen der Art fi(k1), fi(k1,k2), fi(k1,k2,k3)

fi ist der Name des Feldes und die k1,... sind die Dimensionen (ganzzahlige Konstante bzw. in Unterprogrammen ganzzahlige Variable) des Feldes.

Wurden Felder bereits durch eine Typvereinbarung definiert, dann darf die Felddefinition nicht noch in einer DIMENSION-Anweisung auftreten.

4.3.3. Speicherplatzanweisung

Mit Hilfe der Speicherplatzanweisung ist es moeglich, dass verschiedene Programmkomplexe auf physisch gleiche Speicherbloecke (Commonbloecke) zugreifen koennen. Damit kann Speicherplatz eingespart werden.

Syntax: COMMON /CB1/V1/CB2/V2.../CBn/Vn

CB1... sind die Speicherblocknamen, die V1... sind eine Liste von durch Komma getrennten Variablen-, Feld- oder Feldelementnamen.

Beispiel: Im Programmkomplex 1 sind im Block AREA die Variablen A,B,C und im Block FELD Z(3,2,2) zu speichern.

```
COMMON /AREA/A,B,C/FELD/Z(3,2,2)
```

Im Programmkomplex 2 sind im Block AREA die Variablen X,Y und im Block FELD das Feld U(2,2,2) zu speichern.

```
COMMON /AREA/X,Y/FELD/U(2,2,2)
```

Wie im Beispiel zu sehen ist, koennen die Speicherbloecke mit gleichen Namen in den Programmkomplexen unterschiedlich lang sein. Es muss aber der Programmkomplex mit der groessten Blocklaenge zuerst im Programm auftreten.

Ein Speicherblock ohne Blocknamen heisst nichtmarkiert. Es darf hoechstens ein einziger nichtmarkierter Block existieren. Einem nichtmarkierten Block wird vom FORTRAN-Compiler zur Darstellung eine interne Bezeichnung zugeordnet. Fehlt ein Blockname, dann sind zwei Schraegstriche zu schreiben. Folgt nach dem Schluesselwort COMMON ein nichtmarkierter Block, dann koennen die Schraegstriche entfallen. Ein Blockname darf in einer COMMON-Anweisung mehrmals vorkommen, die dazugehoerigen Listen bilden einen benannten Speicherblock.
Eine Feldbezeichnung in einer COMMON-Anweisung definiert dieses Feld. Ein solches Feld darf nicht noch einmal in einer Typvereinbarung oder DIMENSION-Anweisung definiert werden. Ein schon durch eine Typvereinbarung oder DIMENSION -Anweisung definiertes Feld wird in der COMMON-Anweisung nur mit dem Feldnamen angegeben.

Beispiel:
```
DIMENSION A(4),B(2,2)
COMMON A/BER/Z(2),C(3)//B
```

Der nichtmarkierte Block enthaelt dann A(1), A(2), A(3), A(4), B(1,1), B(2,1), B(1,2), B(2,2)
Der Block BER enthaelt dann Z(1), Z(2), C(1), C(2), C(3).

4.3.4. Aequivalenzanweisung

Eine Aequivalenzanweisung ordnet innerhalb eines Programmkomplexes allen in einer Liste enthaltenen Variablen oder Feldelementen einen gemeinsamen Speicherplatz zu.

Syntax: EQUIVALENCE (V1),(V2),...,(Vn)

V1,V2... sind Listen von durch Komma getrennten Variablen-, Feld- oder Feldelementnamen.

Beispiel: EQUIVALENCE (A,B,C)

Bei der Programmabarbeitung wird fuer A,B und C der gleiche Speicherplatz benutzt.

Die in den Listen auftretenden Feldnamen muessen zuvor als Felder vereinbart worden sein. Die Indizes eines Feldelements muessen ganzzahlige Konstante sein. Ein Element eines mehrdimensionalen Feldes kann auch durch die Feldnummer des Feldelementes gekennzeichnet werden.

Beispiel:
```
DIMENSION A(10),B(10)
EQUIVALENCE (A,B)
```

Fuer die Felder A und B werden die gleichen Speicherplaetze benutzt.

Variable koennen sowohl in einer Aequivalenz- als auch in einer Speicherplatzanweisung auftreten, wobei zwei oder mehrere Variable eines oder mehrerer Commonbloecke nicht durch eine Aequivalenzanweisung verbunden werden koennen.

Beispiel:
```
COMMON K/A,B,C
EQUIVALENCE (A,D)
```

A und D benutzen den ersten Platz im Speicherblock K.

Durch eine Aequivalenzanweisung kann die Groesse eines Commonblockes durch das Anfuegen von Elementen an das Ende des Blockes vergroessert werden.

Beispiel:
```
DIMENSION R(2,2)
COMMON /Z/W,X,Y
EQUIVALENCE (Y,R(3))
```

Es ergibt sich folgende Speicherbelegung:

Speicherplatz	Speicherblock Z	EQUIVALENCE
i	W	R(1,1)
i+1	X	R(2,1)
i+2	Y	R(1,2)
i+3		R(2,2)

Eine Erweiterung des Speicherblocks nach links ist nicht moeglich. In der Aequivalenzanweisung des letzten Beispiels darf z.B. anstelle R(3) nicht R(4) stehen.

Falsch ist auch die folgende Aequivalenzanweisung, da sie nicht widerspruchsfrei ist.

```
EQUIVALENCE (A(2),B(1)),(A(5),B(3))
```

4.3.5. Datenanfangswertanweisung

Mit Hilfe dieser Anweisung koennen Variablen und Feldern Anfangswerte zugewiesen werden.

Syntax: DATA V1/K1/,V2/K2/,...,Vn/Kn/

Die V1... sind Listen von durch Komma getrennte Variablen-, Feld- oder Feldelementnamen, die K1... sind Listen von Konstanten, die den Elementen der V1... Listen zugewiesen werden. Die Anzahl der Elemente der K-Liste muss mit der, der dazugehoerigen V-Liste uebereinstimmen. Sollten mehrere aufeinanderfolgende Elemente der V-Liste den gleichen Wert zugewiesen bekommen, dann kann n*k geschrieben werden.

Beispiel: DIMENSION C(7)
DATA A,B,C(1),C(3)/14.73,-8.1,2*7.5/

Die Anfangswerte koennen auch hexadezimale und Zeichenkettenkonstanten sein. Sie koennen Elementen beliebigen Typs zugeordnet sein.

Beispiel: DIMENSION HARY(2)
DATA HARY,B/4HTHIS,4H OK., 7.66/

Eine DATA-Liste darf keinen Namen enthalten, der in einer nachfolgenden Vereinbarungsanweisung nochmal auftritt. Einer Variablen eines nichtmarkierten Commonblocks kann kein Anfangswert zugewiesen werden. Eine Anfangswertzuweisung fuer Variable benannter Commonbloecke ist nur in BLOCK DATA-Programmkomplexen zulaessig.(s.Abschn.6.5.)

4.3.6. Externalanweisung

FORTRAN gestattet die Verwendung von Unterprogrammen. Diese Unterprogramme sind selbstaendige Programmkomplexe. Wird ein solches Unterprogramm aufgerufen, dann ist es mit Parametern (aktuelle Parameter) zu versorgen. Ist ein aktueller Parameter ein Unterprogrammname, muss dieser Name von den Namen ueblicher Variabler dadurch unterschieden werden, dass er in einer Externalanweisung erscheint.

Syntax: EXTERNAL UP-Name, UP-Name...

Beispiel: EXTERNAL SUM,AFUNC
,
CALL SUBR (SUM,AFUNC,X,Y)

5. Ein- und Ausgabeanweisungen

Die Ein- und Ausgabeanweisungen haben die Aufgabe, die Steuerung, und die Bedingungen der Datenuebertragung zwischen dem Speicher und den externen Geraeten zu definieren. Externe Geraete sind die Konsole, Diskette, Drucker u.a.

Es wird zwischen fuenf Arten von E/A-Anweisungen unterschieden.

1. Formatgebundene Ein- und Ausgabeanweisungen

2. Formatfreie Ein- und Ausgabeanweisungen zur Uebertragung binaerer Daten

3. Ein- und Ausgabeanweisungen zur Positionierung und Markierung von Dateien

4. ENCODE- und DECODE-Anweisungen fuer die Uebertragung von Daten zwischen den Speicherplaetzen

5. Formatanweisungen im Zusammenhang mit formatgebundenen Ein- und Ausgabeanweisungen.
Sie ermoeglichen die Konvertierung von Zeichenfolgen einer rechnerexternen Darstellungsform in die rechnerinternen Werte und umgekehrt. (Konvertierung und Rueckkonvertierung)

5.1. Formatgebundene Ein- und Ausgabeanweisungen.

Mit jeder Abarbeitung einer Eingabeanweisung wird ein neuer Datensatz von der Eingabedatei gelesen. Die Anzahl der zu lesenden Datensaetze wird bestimmt durch eine Liste V, deren Elemente verschiedenen Typs und von unbegrenzter Anzahl sein koennen und durch eine Formatanweisung spezifiziert werden. Es werden jedoch nur soviel Daten eingelesen, wie Elemente in der Liste V enthalten sind. Weitere Daten werden ignoriert.

Mit jeder Abarbeitung einer Ausgabeanweisung koennen mehrere Daten ausgegeben werden. Die Anzahl der auszugebenden Daten, welche von unterschiedlichem Typ sein koennen, wird spezifiziert in einer Liste V. Fuellen die auszugebenden Daten das angegebene Format nicht aus, werden Leerzeichen aufgefuellt.

```
Syntax: READ  (u,m, ERR=L1, END=L2) V
        WRITE (u,m, ERR=L1, END=L2) V
```

u ist die Nummer einer physischen und logischen Einheit. Diese Nummer ist eine vorzeichenlose Konstante oder eine ganzzahlige Variable, der vor der Ausfuehrung einer READ- oder WRITE- Anweisung ein Wert zugewiesen werden muss im Bereich von 1...255 . Die Einheiten 1,3,4 und 5 sind der Konsole zugeordnet, Einheit 2 ist der Drucker. Die Einheiten 6...10 sind den Diskettendateien zugeordnet.
Durch den Nutzer koennen diese Einheiten einschliesslich der bis 255 anders zugeordnet werden.
m ist die Marke einer Formatanweisung, welche alle notwendigen Informationen enthaelt, um die auf dem Eingabemedium gespeicherten Daten bei der Eingabe richtig zu interpretieren bzw. intern gespeicherte Daten in der gewuenschten Form auszugeben.
m kann auch ein Feldname sein. Zur Ausfuehrungszeit der READ- oder WRITE- Anweisung muss dann der Anfang dieses Feldes eine gueltige in Klammern eingeschlossene Formatanweisung enthalten.
ERR=L1 ist optional. L1 ist eine Marke, zu der verzweigt

wird, wenn ein Einlesefehler auftritt.
ERR=L2 ist optional. L2 ist eine Marke, zu der verzweigt wird, wenn das Dateiende erreicht wird.
V ist die Ein-/Ausgabeliste, deren durch Komma getrennte Listenelemente Namen von Variablen, Feldnamen oder Feldelementen sind. Die genaue Spezifikation der Liste wird in Abschn.5.2. beschrieben.

Beispiel: Es sind von der externen Einheit mit der Nummer 5 Daten zu lesen, welche durch die Formatanweisung mit der Marke 20 naeher beschrieben werden und den Variablen K,L,M und N zuzuordnen sind.

```
READ (5,20) K,L,M,N
```

Es sind die Werte der Variablen A,B,C und D, welche durch die Formatanweisung mit der Marke 10 naeher beschrieben werden, auf die externe Einheit mit der Nummer 2 auszugeben.

```
WRITE (2,10) A,B,C,D
```

Es sind von der externen Einheit mit der Nummer 6 Daten zu lesen, welche durch die Formatanweisung, die am Anfang des Feldes A steht, naeher beschrieben werden und den Feldern B und M zuzuordnen sind.

```
DIMENSION A(3),B(3),M(4)
DATA A/'(3F1','0.3,','4I6)'/
                ,
                ,
READ (6,A) B,M
```

Fuer die Ein- und Ausgabe alphanumerischer Informationen wird keine Liste benoetigt.

Beispiele:
```
READ (I,25)
25 FORMAT (10HABCDEFGHIJ)
```

Es werden 10 Zeichen vom Eingabegeraet I gelesen und an die Stelle der 10 Zeichen der Formatanweiwung eingesetzt.

```
WRITE (2,26)
26 FORMAT (12HH CONVERSION)
```

Der Text 'H CONVERSION' der Formatanweisung wird auf dem Ausgabegeraet 2 ausgegeben.

5.2. Eingabe-/Ausgabelisten-Spezifikation

Syntax: m1,m2,...,mn

m1,m2... sind die Listenelemente. Diese sind die Namen von Variablen, Feldern oder Feldelementen. Konstante duerfen nicht als Listenelement auftreten.

Es gibt drei Arten von Listenelementen:

1. Dem Listenelement ist genau eine Variable oder ein Feldelement zugeordnet.

   Beispiele: A
   C(26,1),R,K,D
   B,I(10,10),S,F(1,25)

2. Dem Listenelement sind mehrere Feldelemente zugeordnet.

   Beispiel: B

   Wenn B ein zweidimensionales Feld der Art B(j,k) ist, dann sind dem Listenelement B die Feldelemente B(1,1), B(2,1), B(3,1),..,B(1,2), B(2,2),...,B(j,k) zugeordnet.

3. Das Listenelement ist wie eine Laufanweisung zu deuten. Die Liste wird dann zyklische Liste genannt.

   Beispiel: (G(K),K=1,7,3)

   Dieser zyklischen Liste sind die Feldelemente G(1), G(4) und G(7) zugeordnet.

5.3. Formatanweisungen

Formatierte E/A-Anweisungen benoetigen fuer die Abarbeitung eine Formatanweisung. Eine Formatanweisung kann an beliebiger Stelle im Programm stehen. Es kann von mehreren verschiedenen E/A-Anweisungen auf die gleiche Formatanweisung zugegriffen werden. Die Abarbeitung der E/A-Anweisung bewirkt unter Benutzung der entsprechenden Formatanweisung den Aufruf eines interpretativ arbeitenden Formatkontrollprogramms. Es konvertiert entsprechend der E/A-Liste die Zeichenfolgen einer rechnerexternen Darstellungsform, die zu Datensaetzen zusammengestellt sind in die rechnerinternen Werte und umgekehrt (Konvertierung und Rueckkonvertierung).

Syntax: m FORMAT (Format,Format,...)

m ist die Marke der Formatanweisung; Format... ist eines der folgenden Formate:

Format	Konvertierung
I	ganzzahlige Daten
E,D,F	reelle "
L	logische "

G	ganzzahlige, reelle, logische Daten
A,H	Zeichenketten
X	Leerzeichenuebertragung

Die durch die Zeichenanzahl spezifizierte Zeichenfolge im Datensatz wird als Konvertierungsfeld bezeichnet. Die Zeichenanzahl w gibt die Zahl der externen Zeichen an, die bei der Eingabe in einem internen Wert konvertiert bzw. die Zahl der Zeichen, durch die ein interner Wert bei der Ausgabe dargestellt werden sollen. Durch d wird die Anzahl der Dezimalstellen spezifiziert. Bei den Formaten E,D,F und G kann ein Skalenfaktor nP vor die Formatbezeichnung gesetzt werden.

5.3.1. I-Format

Das I-Format ist fuer die Uebertragung ganzzahliger Daten zu verwenden.

Syntax: Iw 1<=w<=255

Fuer die Eingabe gilt:

Ein positives Vorzeichen ist optional. Fuehrende Leerzeichen sind ohne Bedeutung, die im Datenfeld werden jedoch als '0' interpretiert.

Beispiel:

Format	Eingabe	Variable	interner Wert
I4	_124	H	124
I4	-124	I	-124
I7	__6732_	J	67320
I4	1_2_	K	1020

Anweisungen:

```
READ (u,m) H,I,J,K
m FORMAT (I4,I4,I7,I4)
```

Fuer die Ausgabe gilt:

Die ganze Zahl wird einschliesslich eines eventuell vorhandenen Minusvorzeichens rechtsbuendig in das Datenfeld uebertragen. Bei Bedarf wird nach links mit Leerzeichen (blank) aufgefuellt. Ist die zur Darstellung der Zahl benoetigte Anzahl der Zeichen groesser als die notierte Feldweite w, wird das Datenfeld mit '*' gekennzeichnet.

Beispiel:

Format	Variable	interner Wert	Ausgabe
I5	A	+570	__570
I3	B	210	210
I6	C	-10013	-10013
I2	D	157	*7

Anweisungen:

```
WRITE (u,m) A,B,C,D
m FORMAT (I5,I3,I6,I2)
```

5.3.2. E-, D-, F-Format

Das E-, D- und F-Format ist fuer die Uebertragung von reellen Daten einfacher und doppelter Genauigkeit zu verwenden. Ob die Genauigkeit einfach oder doppelt ist, wird vom Typ des zugehoerigen Elements der E/A-Liste entschieden. Die Formate haben darauf keinen Einfluss.

Syntax: Ew.d, Dw.d, Fw.d 1<=w<=255, 0<=d<=w

Fuer die Eingabe im E-, D- und F-Format gilt:

Es wird zwischen drei Arten der Datennotation unterschieden:

1. Ist im Datenfeld ein Dezimalpunkt, dann bestimmt dieser die Anzahl der Dezimalstellen.

   Beispiel: Format Eingabe interner Wert
   E4.0 3.33 0.333*10**1

2. Ist im Datenfeld kein Dezimalpunkt und kein Exponent (E,D), dann sind die letzten rechten d Ziffern (oder Leerzeichen) des Datenfeldes der gebrochene Teil der Zahl und die davorstehenden Ziffern (oder Leerzeichen) der ganze Teil der Zahl.

   Beispiel: Format Eingabe interner Wert
   E8.5 -1322000 -0.1322*10**2

3. Ist im Datenfeld kein Dezimalpunkt aber ein Exponent, dann sind die letzten d Ziffern des Datenfeldes vor E oder D der gebrochene Teil der Zahl.

   Beispiel: Format Eingabe interner Wert
   E10.2 -12830E+03 0.1283*10**6

Fuehrende Leerzeichen haben keine Bedeutung. Ein Exponent ohne Ziffer davor ist verboten. Die Zahl ist Null, wenn im Datenfeld nur Leerzeichen sind.

Fuer die Ausgabe im E- und D-Format gilt:

Die auszugebenden Daten haben folgenden Aufbau:

1. Vorzeichen Minus (wenn Daten negativ)
2. Null und Dezimalpunkt
3. d Dezimalziffern
4. Buchstabe E
5. Vorzeichen des Exponenten (Minus oder blank)
6. zwei Exponentenziffern

Beispiel: Format	interner Wert	Ausgabe
E10.4	-1234567.	-.1234E_07
E7.3	56.93	*0.569E
E13.5	13.31	_0.13310E_02
E8.2	0.0	0.0_____

Fuer die Ausgabe im F-Format gilt:

Es werden insgesamt w Zeichen einschliesslich des Dezimalpunktes und eines eventuell vorhandenen Minuszeichens ausgegeben. Hinter dem Dezimalpunkt stehen d Zeichen.

Beispiel: Format	interner Wert	Ausgabe
F10.4	368.42	_368.4200
F7.2	1234.567	1234.57
F7.3	-5.6	-5.600
F5.0	1234.567	1234.
F5.0	0.12	____0.
F6.4	4739.76	*.7600

5.3.3. L-Format

Das L-Format ist fuer die Uebertragung von logischen Daten zu verwenden.

Syntax: Lw 1<=w<=255

Fuer die Eingabe gilt:

TRUE bzw. FALSE wird der Variablen zugeordnet, wenn im Datenfeld das erste vom Leerzeichen verschiedene Zeichen ein 'T' bzw. ein 'F' ist. Die 'T' bzw. 'F' eventuell folgenden Zeichen sind ohne Bedeutung.

Fuer die Ausgabe gilt:

Ein 'T' bzw. ein 'F' wird rechtsbuendig in das Datenfeld uebertragen, wenn der Wert des Listenelements ungleich Null bzw. Null ist. Davor erscheinen w-1 Leerzeichen.

Beispiel: Format	interner Wert	Ausgabe
L1	=0	F
L1	<>0	T
L5	<>0	____T
L7	=0	_____F

5.3.4. G-Format

Das G-Format ist fuer die Uebertragung ganzzahliger, reeller und logischer Daten zu verwenden. Wie die Daten dargestellt werden, wird durch den Typ des E/A-Listenelements bestimmt.

Syntax: Gw.s 1<=w<=255 0<=s<=w

s hat nur eine Bedeutung bei der Eingabe reeller Daten. Es entspricht d im E-, D- und F-Format.

Fuer die Eingabe gilt:

Unter Beruecksichtigung des Typs des E/A-Listenelements ist Gw.s gleichwertig mit:

```
Iw   : ganzzahligen Daten
Lw   : logischen     "
Ew.d : reellen       "
Dw.d :   "           "
Fw.d :   "           "
```

Fuer die Ausgabe gilt:

Unter Beruecksichtigung des Typs des E/A-Listenelements ist Gw.s gleichwertig mit:

```
Iw   : ganzzahligen Daten
Lw   : logischen     "
Ew.d : reellen       "     Wert.<0.1 oder Wert >=10**s
Dw.d :   "           "
F(w-4).(s-K),4* :    "     10**(K-1)<=Wert<10**K
                           0<=K<=s
```

Beispiel: Format interner Wert Ausgabe

```
  G8.3          0.0123       .123E_-1
  G8.3          123.0        123.____
 G11.3          1023.4       __0.102E_04
  G9.3          123.5        _123.____
```

5.3.5. Skalenfaktor nP

Der Skalenfaktor nP kann bei der Uebertragung von Daten im E-, D-, F- und G-Format verwendet werden.

Syntax: nP -127<=n<=127

Vor jeder Abarbeitung einer formatgebundenen Ein- und Ausgabeanweisung wird n automatisch auf 0 gesetzt. Wird der Skalenfaktor nP vor die Formatbezeichnung gesetzt, dann ist er auch fuer alle folgenden Formatbezeichnungen einer Formatanweisung solange gueltig, bis die Wirkung durch die Notation eines anderen Skalenfaktors veraendert wird. Die Notation von 0P hebt die Wirkung ganz auf.

Beispiel: n FORMAT (I4,4PE10.4,D8.2,2PE7.3,0PE10.5)

Fuer die Eingabe gilt:

Fuer das E-, D-, F- und G-Format hat der Skalenfaktor nur eine Wirkung, wenn kein Exponent bei der Eingabe erscheint. Ist ein Exponent vorhanden, dann ist der:

interne Wert=Eingabewert*10**-n

Fuer die Ausgabe im E- und D-Format gilt:

Ausgabewert=Mantisse des internen Wertes*10**n,
Exponent-n

Fuer die Ausgabe im F-Format gilt:

Ausgabewert=interner Wert*10**n

Fuer die Ausgabe im G-Format gilt:

Der Skalenfaktor wird ignoriert, wenn der innere Wert klein genug ist, um unter Verwendung der F-Konvertierung ausgegeben zu werden. Andernfalls ist die Wirkung die gleiche wie fuer die Ausgabe im E-Format.

5.3.6. A-Format

Das A-Format ist fuer die Uebertragung von Zeichenketten (Literalen) zwischen einem externen Speichermedium und einer Variablen beliebigen Typs zu verwenden. Durch die Typlaenge g des E/A-Listenelements wird die Anzahl der Zeichen festgelegt, die in den entsprechenden Speichereinheiten dargestellt werden koennen.

Syntax: Aw 1<=w<=255

Fuer die Eingabe gilt:

Es wird zwischen drei Faellen unterschieden:

1. w<=g Es werden die w Zeichen des Datenfeldes in die Speichereinheiten uebertragen.

2. w<g Es werden die w Zeichen des Datenfeldes linksbuendig in die Speichereinheiten uebertragen und die g-w Bytes der Speichereinheiten mit Leerzeichen aufgefuellt.

3. w>g Es werden die von links beginnend im Datenfeld stehenden w-g Zeichen uebersprungen und nur die folgenden g Zeichen uebertragen.

Beispiel:

Format	Eingabe	Type	intern
A1	A	INTEGER	A
A3	ABC	"	BC
A4	ABCD	"	CD
A1	A	REAL	A
A7	ABCDEFG	"	DEFG

Fuer die Ausgabe gilt:

Es wird zwischen drei Faellen unterschieden:

1. w=g Es werden g Zeichen aus den Speichereinheiten in das Datenfeld uebertragen.

2. w<g Es werden nur die ersten in den linken Bytes gespeicherten w Zeichen in das Datenfeld uebertragen.

3. w>g Es werden von links beginnend w-g Leerzeichen, gefolgt von g Zeichen in das Datenfeld uebertragen.

Beispiel:

Format	intern	Type	Ausgabe
A1	A1	INTEGER	A
A2	AB	"	AB
A3	ABCD	REAL	ABC
A4	ABCD	"	ABCD
A7	ABCD	"	___ABCD

5.3.7. H-Format

Das H-Format ist fuer die Uebertragung von Zeichenketten (Literalen) zwischen einem externen Speichermedium und dem in der Formatanweisung stehenden Literal zu verwenden.

Syntax: wHc1...cw oder 'c1...cw' 1<=w<=255

Die c1... koennen beliebige Zeichen sein. Wenn in der Form 'c1...cw' ein Apostroph auftritt, dann muss es doppelt geschrieben werden.

Fuer die Eingabe gilt:

Das in der Formatanweisung stehende Literal wird ohne Konvertierung von w Zeichen des Datenfeldes ueberschrieben.

Beispiel:

Format	Eingabe	Format
4H1234	ABCD	4HABCD
'1234'	"	'ABCD'
6H_____	MATRIX	6HMATRIX
'_____'	"	'MATRIX'

Fuer die Ausgabe gilt:

Das in der Formatanweisung stehende Literal der Laenge w wird in das Datenfeld des Satzes uebertragen.

Beispiel:

Format	Ausgabe
1HA	A
'A'	"
8H_STRING_	_STRING_
'_STRING_'	"

5.3.8. X-Format

Das X-Format ist fuer das Uebergehen von Zeichen bzw. fuer das Uebertragen von Leerzeichen zu verwenden.

Syntax: wx 1<=w<=255

Fuer die Eingabe gilt:

Es werden w Zeichen uebergangen.

Beispiel:	Format	Eingabe	interner Wert
	7X,I3	1234567012	012

Fuer die Ausgabe gilt:

In das Datenfeld werden w Leerzeichen uebertragen.

Beispiel:	Format	Ausgabe
	1HA,4X,2HBC	A____BC

5.4. Steuereigenschaften von Formatanweisungen

Die verschiedenen Formate koennen neben ihrer eigentlichen Funktion weitere Eigenschaften haben.

1. Wiederholfaktor

   Die Formate koennen mit einem Wiederholfaktor versehen werden.

   Beispiel:(3F8.3,F9.2) entspricht (F8.3,F8.3,F8.3,F9.2)

2. Gruppenbildung

   Formate koennen durch Klammerung zu Gruppen zusammengefasst werden. Solche Gruppen koennen mit einem Wiederholfaktor versehen werden.

   Beispiel:(2(2I1,E10.3)) entspricht (I1,E10.3,I1,E10.3)

Wenn in einer Gruppe mit Wiederholfaktor ein Literalformat auftritt, dann gilt die Aequivalenz nicht.

In einer geklammerten Formatliste (Gruppe) darf nur eine Formatgruppe enthalten sein.

3. Wiederholung der letzten Formatgruppe

   Wenn in der Formatliste weniger Formate enthalten sind als die E/A-Liste das erfordert, so wird die Abarbeitung mit der am weitesten rechts stehenden Formatgruppe wiederholt unter eventueller Einbeziehung des Wiederholfaktors. Dadurch wird die Verarbeitung des momentanen Datensatzes (Record) beendet und die Verarbeitung eines neuen Datensatzes begonnen.

Beispiel:
```
      DIMENSION A(100)
      READ (3,13) A
        ,
13 FORMAT (5F7.3)
```

Es werden jeweils die ersten fuenf Daten von einem der 20 Datensaetze eingelesen und den A(i) zugeordnet.

```
      WRITE (6,12)E,F,K,L,M,KK,LL,MM,K3,L3,M3
        ,
12 FORMAT (2F9.4,3(3I7))
```

Es werden drei Datensaetze geschrieben.

Datensatz 1: E,F,K,L,M
" 2: KK,LL,MM
" 3: K3,L3,M3

4. Formattrennzeichen

Die Formate muessen durch Komma oder Schraegstrich getrennt sein.

Wird bei der Abarbeitung ein Schraegstrich erkannt, dann wird der Rest eines Eingabesatzes uebersprungen, der Rest eines Ausgabesatzes mit Leerzeichen gefuellt und zu einem neuen Datensatz uebergegangen.

Beispiel:
```
      DIMENSION B(10)
      READ (4,7)B
7 FORMAT (F10.2/F10.2///3F10.2)
```

Die Daten der Datenfelder des 1. und 2. Datensatzes bzw. des 5. Datensatzes werden nach B(1) und B(2) bzw. B(3) bis B(10) uebertragen. Der 3. und 4. Datensatz werden uebergangen.

```
      DIMENSION I(20),A(100)
      WRITE (7,8) I,A
8 FORMAT (10I7/10I7/5OF7.3/5OF7.3)
```

Es werden folgende Datensaetze erzeugt:

Datensatz 1: I(1) ... I(10)
" 2: I(11)... I(20)
" 3: A(1) ... A(50)
" 4: A(51)... A(100)

5. Steuerung des Druckers und der Konsole

Bei formatierter Ein- bzw. Ausgabe ueber Drucker oder Konsole wird das erste Zeichen jedes Ausgabedatensatzes nicht gedruckt, sondern zur Steuerung der ge-

nannten Ausgabegeraete genutzt.
In Abhaengigkeit der Zeichen werden die folgenden Aktionen ausgeloest:

```
     + : keine Aktion
     1 : Einfuegen eines 'Form Feed'
     0 : 2 Zeilen weiterschalten
andere : 1 Zeile        "
Zeichen
```

5.5. Formatfreie Ein- und Ausgabeanweisungen

Die formatfreien Ein- und Ausgabeanweisungen sind speziell fuer die Uebertragung von binaeren Daten zu verwenden. Formatangaben sind also nicht erforderlich.

```
Syntax: READ  (u,ERR=L1,END=L2) V
        WRITE (u,ERR=L1,END=L2) V
```

Bis auf das Fehlen des Hinweises auf eine Formatanweisung entsprechen diese Anweisungen den formatgebundenen Ein- und Ausgabeanweisungen.
Bei der Eingabe ist zu beachten, dass die Anzahl der Variablen nicht grosser sein darf als die Datensatzlaenge.
Bei der Ausgabe ist zu beachten, dass ein logischer Datensatz aus mehr als einem physischen Datensatz bestehen kann.

5.6. Speicherbereichsuebertragungsanweisungen

Diese Anweisungen sind zu verwenden, wenn entsprechend der Formatspezifikation der Inhalt eines Speicherbereiches in einen anderen uebertragen werden soll.

```
Syntax: ENCODE (a,m) V
        DECODE (a,m) V
```

ENCODE wandelt die zu uebertragenden Daten im angegebenen Format in das ASCII-Format um.
DECODE wandelt die zu uebertragenden Daten im ASCII-Format in das angegebene Format um.
a ist der Feldname des Speicherbereiches. Das Feld muss gross genug sein, damit es alle zu verarbeitenden Daten enthalten kann. Ist es zu klein, dann werden bei der Abarbeitung der ENCODE-Anweisung Daten ueberschrieben, die nach dem Feld kommen, und bei der DECODE-Anweisung werden Daten nach dem Feld verarbeitet.
m ist die Marke einer Formatanweisung.
V ist die Ein-/Ausgabeliste.

5.7. Zugriff auf Diskettendateien

Wie bereits in Abschn.5.1. erwaehnt, sind die logischen Einheiten 6...10 standardmaessig den Diskettendateien zugeordnet. Erfolgt durch eine READ- oder WRITE-Anweisung ein Zugriff auf eine solche logische Einheit, dann wird eine

Datei eroeffnet (wenn sie nicht schon eroeffnet war) und bekommt einen Standardnamen, deren wesentlicher Bestandteil die Nummer der logischen Einheit (LUN=logical unit number) ist.

Die Standardnamen koennen sein:

FORT06.DAT,FORT07.DAT,...,FORT10.DAT

Eine Datei kann auch durch die Subroutine OPEN eroeffnet werden. Mit Hilfe dieser Routine kann das Programm einer logischen Einheit einen Dateinamen und ein Laufwerk zuweisen.

Syntax: CALL OPEN (LUN,Dateiname,Laufwerk)

LUN ist die der Datei zugeordnete Nummer einer logischen Enheit. Sie ist eine ganzzahlige Konstante oder Variable zwischen 1 und 10. Dateiname kann bis zu 32 Zeichen lang sein, wobei das letzte Zeichen ein Leerzeichen sein muss. Diese Zeichenfolge ist in Apostrophe einzuschliessen. Laufwerk ist die Nummer des Disketten- Laufwerks, auf welchem die Datei plaziert ist oder plaziert werden soll. Diese Nummer soll ein INTEGER*1-Feld, einfache Variable oder eine Literalkonstante sein und ist in der Form '0','1',...oder '*' zu notieren.
Ein OPEN einer noch nicht vorhandenen Datei generiert eine mit einem Namen versehene Nulldatei. Ist eine Datei vorhanden und nach dem OPEN erfolgt eine:

Eingabe, dann kann auf den Inhalt der Datei zugegriffen werden.
Ausgabe, dann wird die vorhandene Datei zerstoert.

Eine Datei bleibt solange eroeffnet, bis sie durch das nomale Programmende, eine ENDFILE- oder REWIND-Anweisung geschlossen wird.

Syntax: ENDFILE u bzw. REWIND u

Beide Anweisungen schliessen die der logischen Einheit u zugeordnete Datei ab, wobei REWIND u die Datei wieder eroeffnet.

Der Zugriff auf ausgewaehlte 128 Bytes lange Datensaetze (Records) erfolgt folgendermassen:

Die READ- oder WRITE-Anweisung ist durch die Option REC=n zu ergaenzen.

Syntax: READ (u,m,REC=n,ERR=L1,END=L2) V
WRITE(u,m,REC=n,ERR=L1,END=L2) V

n ist die Nummer des Records auf der logischen Einheit u.

Beispiel: I=10
WRITE (6,20,REC=I,ERR=50)X,Y,Z

Der Record 10 wird auf die logische Einheit 6 geschrieben bzw. wenn er schon existiert, dann wird er ueberschrieben.

5.8. Ein- und Ausgabeinterface

E/A-Operationen werden mit Hilfe einer Verteilertabelle (¤LUNTB=logical unit number table) auf die Treiberroutinen der entsprechenden logischen Einheiten verteilt.

Aufbau der Tabelle:

1. max. Anzahl der LUN+1

   Die Anzahl der LUN im Standardlaufzeitpaket betraegt 10, die alle der Konsole entsprechen. Sie koennen vom Anwender neu definiert werden. LUN3 muss der Konsole entsprechen,da es vom Laufzeitpaket fuer Meldungen genutzt wird. Die Anzahl kann erhoeht werden.

2. LUN 2-Byte Treiberroutineadresse
   LUN 2-Byte Treiberroutineadresse
   ,
   ,
   LUN 2-Byte Treiberroutineadresse

Am Anfang der Treiberroutinen ist jeweils eine lokale Adressverteilertabelle fuer die sieben moeglichen Operationen pro Geraet (device).

Die sieben Operationen sind:

1. Formatgebundenes Lesen
2. " Schreiben
3. Formatfreies Lesen
4. " Schreiben
5. Rewind
6. Backspace
7. Endfile

Die Adresse aus ¤LUNTB verweist auf diese Tabelle, und das Laufzeitpaket indiziert die Adresse, um die Adresse der entsprechenden Operationsroutine zu bekommen.

Fuer individuelle E/A-Routinen gelten die folgenden Richtlinien.

1. ¤BF enthaelt die Adresse des Datenpuffers fuer READs und WRITEs.
2. ¤BL enthaelt bei READ die Anzahl der gelesenen Bytes.
3. ¤BL enthaelt bei WRITE die Anzahl der zu schreibenden Bytes.
4. Die Flags werden vor Beendigung der E/A-Operationen wie folgt gesetzt:

a) CY=1, Z=? E/A-Fehler
b) CY=0, Z=0 EOF (Dateiende)
c) CY=0, Z=1 normaler Abschluss

Die Flags werden nach dem Treiberaufruf durch das Laufzeitpaket getestet. Ergibt der Test keine normale Bedingung, dann wird an den in der READ- oder WRITE-Anweisung durch die Option 'ERR=L1' bzw. 'ERR=L2' definierten Marken L1 bzw. L2 fortgesetzt.
Sollten diese Optionen fehlen, kann es zu fatalen Fehlern kommen.

5. Wenn es fuer ein spezielles Geraet nichterlaubte Routinen gibt, dann kann das mit Hilfe der globalen Routine #IOERR mitgeteilt werden. Durch sie wird die nicht fatale Nachricht 'ILLEGAL I/O OPERATION' auf der Konsole mitgeteilt.

Der E/A-Puffer hat eine feste maximale Laenge von 132 Bytes. Wenn der Puffer durch Eingabeoperationen eines Treibers ueberfordert wird, kann es zu undefinierten Fehlern kommen.

6. Funktionen und Unterprogramme

Werden in einem FORTRAN-Programm an verschiedenen Stellen gleiche Programmablaeufe (Prozeduren) benoetigt, dann brauchen diese nur einmal definiert zu werden, wenn das in der folgenden Form erfolgt.

Es wird zwischen vier Typen unterschieden:

Anweisungsfunktionen
Funktionsunterprogramme
Subroutineunterprogramme
Bibliotheksprogramme

Diese Prozeduren werden jeweils unter einem Namen gefuehrt, der aus maximal 6 alphanumerischen Zeichen besteht. Das erste Zeichen muss ein Buchstabe sein.

6.1. Anweisungsfunktionen

Mit Hilfe der Anweisungsfunktion ist es moeglich, Funktionen zu definieren, die aus einer einzigen arithmetischen oder logischen Ergibtanweisung bestehen. Eine Anweisungsfunktion ist in dem Programmkomplex, in dem sie definiert wird, lokal, d.h., sie kann nur dort verwendet werden. Zu gleichnamigen Groessen in anderen Programmkomplexen besteht keine Beziehung. Die Anweisungsfunktion muss zwischen allen Vereinbarungs- und allen ausfuehrbaren Anweisungen stehen.

Syntax: fkname (for.p, for.p,...) = ausdruck

Aufruf: fkname (akt.p, akt.p,...)

fkname ist der Name der Funktion, for.p,... sind die formalen Parameter. Der Inhalt dieser Parameter wird nach dem Aufruf durch die aktuellen Parameter akt.p,... ausgetauscht, d.h., Anzahl und Typ muessen deshalb uebereinstimmen. ausdruck ist ein arithmetischer oder logischer Ausdruck entsprechend der Regeln des Abschn.3..

Bei der Definition ist folgendes zu beachten:

1. Der Typ des Anweisungsfunktionsnamen und der formalen Parameter wird entweder implizit oder in einer Typvereinbarung festgelegt. Wird er in einer Typvereinbarung festgelegt, darf ihm kein Anfangswert zugewiesen werden. Er darf auch nicht als Feld vereinbart werden.

2. Die Parameter koennen Feldelemente sein.

3. Der Ausdruck kann den Aufruf einer vorher definierten anderen Anweisungsfunktion enthalten. Sollte dieser Aufruf einen der formalen Parameter als aktuellen Parameter haben, dann darf dieser dessen Wert nicht aendern.

Beispiel: Definition:

```
FUNC1(A,B,C,D)=((A+B)**C)/D
```

Aufruf:

```
A12=A1-FUNC1(X,Y,Z7,C7)
```

6.2. Funktionsunterprogramme

Eine Funktion, die sich nicht durch eine einzige Anweisung definieren laesst oder die in mehreren Programmkomplexen aufrufbar sein soll, muss als Funktionsunterprogramm definiert werden. Ein solches Unterprogramm ist ein selbstaendiger Programmkomplex. Die im Unterprogramm auftretenden Namen von Variablen, Feldern, Marken und Anweisungsfunktionen haben keine Beziehung zu gleichen Namen in anderen Programmkomplexen.

```
Syntax: t FUNCTION fkname (for.p, for.p,...)
        Folge von Anweisungen
        RETURN
        END

Aufruf: fkname (akt.p, akt.p,...)
```

t ist eine Typbezeichnung fuer fkname, also den Namen der Funktion. Wird t nicht notiert, dann wird der Typ implizit festgelegt. for.p und akt.p haben die gleiche Bedeutung wie bei der Anweisungsfunktion.

Bei der Definition ist folgendes zu beachten:

1. Die Wertuebergabe erfolgt ueber den Funktionsnamen oder ueber die formalen Parameter, wenn ihnen Werte zugewiesen werden. Der Funktionsname muss deshalb wenigstens einmal auf der linken Seite einer Ergibtanweisung stehen oder Element der Eingabeliste einer Eingabeanweisung sein.

2. Es muss wenigstens ein formaler Parameter auftreten. Formale Parameter koennen Variablennamen, Feldnamen, formale Namen von Subroutine- oder anderen Funktionsunterprogrammen sein.

3. Die Namen in der Liste der formalen Parameter duerfen nicht in COMMON-, EQUIVALENCE- oder DATA-Anweisungen des Funktionsunterprogramms vorkommen.

4. Wenn ein aktueller Parameter der Name eines Unterprogramms ist, dann muss dieser Name in einer EXTERNAL-Anweisung notiert werden. (s.Abschn.4.3.8.)

5. Ein Funktionsunterprogramm kann jede Anweisung enthalten, ausser BLOCK DATA-, SUBROUTINE- und andere FUNCTION-Anweisungen. Es darf auch keine Anweisung enthalten, die das gleiche Unterprogramm direkt oder indi-

rekt ueber andere Unterprogrammaufrufe nochmal aufruft.

6. Das logische Ende eines Funktionsunterprogramms ist RETURN. Das physische Ende ist END.

Beispiele: Definition:

```
FUNCTION F(X,Y)
X=X+Y
F=X*Y
RETURN
END
```

Aufruf:

```
C=F(A,B)+A*B
```

Definition: (Parameter ist ein Feld)

```
FUNCTION SUM (BARY,I,J)
DIMENSION BARY (10,20)
SUM=0.0
DO 8 K=1,I
DO 8 M=1,J
8 SUM=SUM+BARY(K,M)
RETURN
END
```

Aufruf:

```
DIMENSION DAT(5,5)
          ,
          ,
S1=TOT1+SUM(DAT,5,5)
```

Definition: (Parameter ist ein UP-Name)

```
FUNCTION PI(X,FUN)
PI=X**2+FUN(X)
RETURN
END
```

Aufruf:

```
EXTERNAL COS,EXP
          ,
          ,
Z=PI(Y,COS)+PI(Y,EXP)
```

6.3. Subroutineunterprogramme

Ein Subroutineunterprogramm liefert in Gegensatz zum Funktionsunterprogramm keinen Wert ueber den Subroutinenamen. Eine Parameterliste ist deshalb nicht unbedingt erforder-

lich.

Syntax: SUBROUTINE sname (for.p, for.p,...)
Folge von Anweisungen
RETURN
END

Aufruf: CALL sname (akt.p, akt.p,...)

sname ist der Name der Subroutine. Er ist zu interpretieren wie eine Marke. Der Name besitzt deshalb keinen Typ. Die optionalen Parameter haben die gleiche Bedeutung wie bei der Anweisungsfunktion.

Bei der Definition sind die gleichen Grundsaetze zu beachten wie bei den Funktionsunterprogrammen, ausser das keine Wertuebergabe ueber den Namen erfolgt und die Parameter optional sind.

Beispiele: Definition: (mit Parametern)

```
      SUBROUTINE SUBR (A,B,C)
      READ (3,7)B
      A=B**C
      RETURN
    7 FORMAT (F9.2)
      END
```

Aufruf:

```
      CALL SUBR (Z9,B7,R1)
```

Definition: (ohne Parameterliste)

```
      SUBROUTINE WECHSEL
      COMMON /TAUSCH/ X,Y
      Z=X
      X=Y
      Y=Z
      RETURN
      END
```

Aufruf:

```
         ,
         ,
      COMMON /TAUSCH/A,B
         ,
         ,
      CALL WECHSEL
         ,
         ,
```

Das Subroutineunterprogramm WECHSEL speichert nach dem Aufruf den Wert des ersten und zweiten Commonlistenelements des Commonblocks TAUSCH um. Somit werden durch den

Aufruf des Subroutineunterprogramms die Werte von A und B vertauscht.

6.4. Bibliotheksprogramme

Die Namen und Aktionen von Bibliotheksprogrammen und die Typen der Parameter sowie des gelieferten Ergebnisses sind durch den FORTRAN-Compiler fest vereinbart und Bestandteil desselben. Die Namen der Bibliotheksprogramme sollten nicht fuer andere Zwecke benutzt werden.

Syntax: name (par., par.,...)

name ist der Name des Bibliotheksprogramms, par,... sind die hinsichtlich Typ und Anzahl vorgeschriebenen Parameter.

Bibliotheksprogramme werden in interne und externe Programme unterteilt.

1. Interne Bibliotheksprogramme

   Der Kode zu ihrer Realisierung wird bei jedem Aufruf durch den Compiler in das ausfuehrbare Programm eingefuegt. Da der Name eines solchen Programms kein Name einer externen Funktion ist, kann er nicht als aktueller Parameter in Unterprogrammaufrufen verwendet werden.

2. Externe Bibliotheksprogramme

   Diese Programme sind vergleichbar mit den Funktionsunterprogrammen. Sie sind Bestandteil der FORTRAN-Standardbibliothek. Der Kode zu ihrer Realisierung wird fuer beliebig viele gleiche Aufrufe nur einmal durch den Compiler in das ausfuehrbare Programm eingefuegt. Da der Name eines solchen Programms der Name einer externen Funktion ist, kann er als aktueller Parameter in Unterprogrammaufrufen verwendet werden, Er muss zuvor in einer EXTERNAL-Anweisung notiert worden sein.

Neben diesen Programmen gibt es in dieser FORTRAN-Variante vier weitere Standardfunktionen.

1. PEEK(a) Ist der Wert des durch a bezeichneten Speicherplatzes.
2. CALL POKE(a1,a2) Der Inhalt des Speicherplatzes a1 wird durch den Inhalt von a2 ersetzt.
3. INP(a) Ist das am Port a anliegende Byte.
4. CALL OUT(a1,a2) An den durch a1 spezifizierten Port wird der Wert von a2 ausgegeben.

In den folgenden Tabellen sind die verfuegbaren Programme und ihre Bedeutung aufgefuehrt.

Die Abkuerzungen in der Tabelle bedeuten:

I : INTEGER
R : REAL
D : DOUBLE PRECISION
Vz : Vorzeichen
p1,... : Parameter

Interne Bibliotheksprogramme

Name	Definition	Parameter Anzahl	Parameter Typ	Typ der Funktion
ABS		1	R	R
IABS	Absolutbetrag	1	I	I
DABS		1	D	D
INT		1	R	I
AINT	f(p)=Vz(p)*	1	R	R
IDINT	(groesste ganze Zahl<=/p/)	1	D	D
MOD	Divisionsrest	2	I	I
AMOD	f(p1,p2)=p1-(p1/p2)*p2	2	R	R
MAXO			I	I
MAX1			R	I
AMAXO	f(p1,p2,...)=Max(p1,p2,...)	>=2	I	R
AMAX1			R	R
DMAX1			D	D
MINO			I	I
MIN1			R	I
AMINO	f(p1,p2,...)=Min(p1,p2,...)	>=2	I	R
AMIN1			R	R
DMIN1			D	D
FLOAT	Konvertierung I —> R	1	I	R
IFIX	Konvertierung R —> I	1	R	I
SNGL	Konvertierung D —> R	1	D	R
DBLE	Konvertierung R —> D	1	R	D
SIGN	Vorzeichenuebertragung f(p1,p2)=VZ(p2)*/p1/	2	R	R
DIM	Schwache Differenz	2	R	R
IDIM	f(p1,p2)=p1-Min(p1,p2)	2	I	I

Externe Bibliotheksprogramme

Name	Definition	Parameter Anzahl	Parameter Typ	Typ der Funktion
EXP DEXP	f(p)=e**p	1 1	R D	R D
ALOG DLOG	f(p)=ln(p)	1 1	R D	R D
ALOG10 DLOG10	f(p)=lg(p)	1 1	R D	R D
SIN DSIN	f(p)=sin(p)	1 1	R D	R D
COS DCOS	f(p)=cos(p)	1 1	R D	R D
TANH	f(p)=tanh(p)	1	R	R
ATAN DATAN	f(p)=arctan(p)	1 1	R D	R D
ATAN2 DATAN2	f(p1,p2)=arctan(p1/p2)	2 2	R D	R D
SQRT DSQRT	f(p)=p**1/2	1 1	R D	R D
DMOD	Divisionsrest f(p1,p2)=p1-(p1/p2)*p2	2	D	D

6.5. BLOCK DATA-Unterprogramme

Ein BLOCK DATA-Unterprogramm hat die Aufgabe, einfachen Variablen und Feldern, die in benannten Commonbloecken liegen, Anfangswerte zuzuweisen. In Hauptprogrammen, Funktions- und Subroutineunterprogrammen ist das nicht moeglich.

Ein BLOCK DATA-Unterprogramm ist ein selbstaendiger Programmkomplex. Ein darin auftretender Commonblock muss mit gleichnamigen Commonbloecken in anderen Programmkomplexen uebereinstimmen. Zu gleichnamigen Groessen anderer Art besteht keine Beziehung.

```
Syntax: BLOCK DATA (name)
        Folge von Vereinbarungsanweisungen
        END
```

name ist optional, fuer Vereinbarungsanweisungen koennen folgende Typen verwendet werden:

EQUIVALENCE
DATA
COMMON
DIMENSION
Typvereinbarungsanseisung

Bei der Definition ist folgendes zu beachten:

1. Wenn irgendeinem Element in einem Commonblock ein Anfangswert zugewiesen werden soll, muss es zuvor in einer COMMON- bzw. EQUIVALENCE-Anweisung aufgelistet sein.

2. Es muss der gesamte Commonblock aufgelistet werden, also auch die Elemente, denen kein Anfangswert zugewiesen werden soll.

3. Es koennen mehrere Commonbloecke vorhanden sein.

4. Es koennen mehrere BLOCK DATA-Unterprogramme existieren

Beispiel:
```
      BLOCK DATA
      LOGICAL A1
      COMMON/BETA/B(3,3)/GAM/C(4)
      COMMON/ALPHA/A1,F,E,D
      DATA B/1.1,2.5,3.8,3*4.96,
     1      2*0.52,1.1/,C/1.2E0.3*4.0/
      DATA A1/.TRUE/,E/-5.6/
```

6.6. Kodeunterprogramme

FORTRAN bietet die Moeglichkeit, auf Unterprogramme, die in anderen Quellsprachen bzw. als maschinenorientierte Unterprogramme geschrieben wurden, zurueckzugreifen. Die problemorientierte Programmiersprache FORTRAN nutzt wegen des allgemeinen Charakters ihrer Anweisungen nicht saemtliche Ausdrucksmoeglichkeitan , die die Befehlsliste des Mikroprozessos U880 bietet. Aus diesem Grunde ist es vorteilhaft, das FORTRAN vorsieht , auf die Maschinensprache des genennten Prozessors in Form von Kodeunterprogrammen Bezug zu nehmen.

Der Aufruf eines Unterprogramms ohne Parameter erzeugt einen CALL-Befehl. Das entsprechende Unterprogramm wird ueber RET verlassen.

Der Aufruf eines Unterprogramms mit Parametern uebergibt zusaetzlich die Parameter in Form von Adressen. Es werden unabhaengig vom Typ immer zwei Bytes belegt. Die Adresse ist diejenige des unteren Bytes des aktuellen Argumentes.

Der Platz der Adressen ist abhaengig von der Anzahl der Parameter.
Fuer bis zu drei Parameter gilt:

```
Registerpaar HL : 1. Parameter
     "       DE : 2.      "
     "       BC : 3.      "
```

Fuer mehr als drei Parameter gilt:

```
Registerpaar HL : 1. Parameter
     "       DE : 2.      "
     "       BC : Anfangsadresse eines Datenblocks,
                  der die Adressen dar Parameter
                  3,4,... enthaelt.
```

Benoetigt das Kodeunterprogramm mehr als drei Parameter und sollen diese in einen lokalen Datenblock uebertragen werden, dann kann das mit Hilfe der Systemroutine ¤AT erfolgen.

Fuer die Definition gilt:

```
Registerpaar HL : Adresse des lokalen Datenblocks
Register      A : Anzahl der zu uebertragenden Daten

Kodeunterprogr. : UP: LD (P1),HL
                      LD (P2),DE
                      LD HL,P3
                      LD A,ANZ-2
                      CALL ¤AT
                       ,
                       ,
                  Anwenderanweisungen
                       ,
                       ,
                      RET

                  P1: DS 2
                  P2: DS 2
                  P3: DS 2*(ANZ-2)
```

Die Funktionen von FORTRAN uebergeben ihre Ergebnisse wie folgt:

```
LOGICAL           : Register          A
INTEGER           : Registerpaar      HL
REAL              : Speicherplatz     ¤AC
DOUBLE PRECISION  :      "            DAC

Low-Bytes der     :      "     ¤AC,¤DAC
Mantisse
```

7. Struktur von FORTRAN-Programmen

FORTRAN- Programmkomplexe muessen folgendermassen strukturiert sein:

1. programmart

   Fuer programmart kann eine der folgenden Bezeichnungen stehen:

   a) PROGRAM Fehlt diese Bezeichnung, wird dem Hauptprogramm der Name =MAIN zugeordnet.

   b) FUNCTION fkname (for.p, for.p,...)
   c) SUBROUTINE sname (for.p, for.p,...)
   d) BLOCK DATA (name)

2. Typvereinbarungs-, EXTERNAL-, DIMENSION-Anweisungen
3. COMMON-Anweisungen
4. EQUIVALENCE-Anweisungen
5. DATA-Anweisungen
6. Anweisungsfunktionen
7. Ausfuehrbare Anweisungen
8. END

8. Compilierung von FORTRAN-Programmen

8.1. Kommandostruktur

Nach dem Erscheinen des Promptzeichens (%) des UDOS-Betriebssystems kann der FORTRAN-Compiler durch das Kommando FORTRAN aufgerufen werden; also:

```
%FORTRAN
```

Der FORTRAN-Compiler meldet sich mit *.

Jetzt koennen dem Compiler Angaben fuer die Compilierung eines Quellprogramms gegeben werden.

Die vollstaendige Kommandostruktur lautet:

```
* obj-ger/dateiname.OBJ,list-ger/dateiname.LST
```

Es bedeutet:

obj-ger : Geraet fuer das Speichern des Objektprogramms mit dem Deteinamen dateiname.OBJ.

list-ger : Geraet fuer die Ausgabe des Listings mit dem Dateinamen dateiname.LST.

quell-ger: Geraet fuer das Speichern des Quellprogramms mit dem Dateinamen dateiname.S.

Die Angaben links vom Gleichheitszeichen sind optional. Werden die Geraete nicht oder nur teilweise angegeben, dann werden die entsprechenden Dateien auf allen Geraeten gesucht. Wenn die Erweiterungen der Dateinamen, also OBJ, LST, S nicht angegeben werden, dann werden sie automatisch angefuegt.

Beispiele:

`*=TEST`	Compilierung des Programms TEST.S, Erzeugung des Objektprogramms TEST.OBJ
`*,SYSLIST =TEST`	Compilierung des Programms TEST.S, Drucken des Listings
`*TESTOBJ =TEST`	Compilierung des Programms TEST.S, Erzeugung des Objektprogramms TESTOBJ.OBJ
`*TEST,TEST =TEST`	Compilierung des Programms TEST.S, Erzeugung des Objektprogramms TEST.OBJ, Erzeugung des Listings TEST.LST
`*, =TEST`	Compilierung des Programms TEST.S, Meldung von Fehlern

Die Kommandostruktur kann durch Selektoren beeinflusst werden. Werden die folgenden Selektoren verwendet, dann ist

jeweils ein Minuszeichen (-) voranzustellen.

O : Die Listingadressen werden oktal gedruckt.

H : Die Listingadressen werden hexadezimal gedruckt. (Standart)

N : Der erzeugte Kode wird nicht gelistet.

R : Die Erzeugung eines Objektprogrammes wird erzwungen.

L : Die Erzeugung einer Listingdatei wird erzwungen.

P : Tritt waehrend der Compilierung ein Stack-Ueberlauf (ERROR 134) auf, dann erweitert jedes P den Stack um 100 Bytes.

Beispiele:

*=TEST-L	Compilierung des Programms TEST.S, Erzeugung des Objektprogramms TEST.OBJ, Erzeugung des Listings TEST.LST
*,SYSLIST =TEST-N	Compilierung des Programms TEST.S, Drucken des Listings ohne Objektkode
*=TEST-P-P	Compilierung des Programms TEST.S, Erzeugung des Objektprogramms TEST.OBJ, Stackbereichserweiterung um 200 Bytes

Das Programm FORTRAN wird durch Betaetigen der Tasten 'CTRL' und 'C' verlassen. Das UDOS-Betriebssystem meldet sich mit dem Promptzeichen (%).

8.2. FORTRAN-Programme in EPROMs

Soll der erzeugte Kode in einer Form vorliegen, wie er fuer das Laden von EPROMs erforderlich ist, dann wird das durch die Angabe des Selektors 'M' erreicht.

Der Unterschied zum normalen Kode ist:

1. FORMAT-Anweisungen sind im Programmbereich angeordnet und werden mit einem JMP-Befehl uebersprungen.

2. Parameterbloecke fuer Unterprogrammaufrufe mit mehr als drei Parametern werden zur Laufzeit initialisiert und nicht durch den Lader.

Bei der Programmierung ist folgendes zu beachten:

1. RAM-Speicherplaetze koennen nicht mit DATA-Anweisungen initialisiert werden, da diese Anweisung nur vor der Laufzeit aktiv ist. Variable und Felder koennen dagegen waehrend der Laufzeit durch Ergibt- oder READ-

Anweisungen initialisiert werden.

2. Der Inhalt einer Formatanweisung kann waehrend der Laufzeit nicht veraendert werden (s.Abschn.5.3.7.).

3. Bei Benutzung von E/A-Routinen der Standardbibliothek ist zu beachten, dass Diskettendateien nur auf den logischen Einheiten 6...10 eroeffnet werden. Werden andere logische Einheiten benoetigt, muss die Verteilertabelle (LUNTB) des E/A-Interface mit den entsprechenden Adressen fuer die Treiberroutinen ergaenzt werden (s.Abschn.5.8.).

Die Bibliotheksroutine =INIT setzt den Stackpointer an den Anfang des verfuegbaren Speichers, bevor die Programmausfuehrung beginnt.

Aufruf: LD BC, Rueckkehradreswe
JP =INIT

8.3. FORTRAN-Compiler-Fehlermeldungen

8.3.1. Warnungen

Tritt eine Warnung auf, dann wird die Compilierung mit dem naechsten Objekt der Quellzeile fortgesetzt. Warnungsmeldungen werden ein Prozentzeichen (%) und die Nummer der Quellzeile vorangestellt.

Warnungen:

0 Doppelte Anweisungsmarke
1 Unzulaessiges DO-Ende
2 Blockname=Prozedurname
3 Missbrauch von Feldnamen
4 COMMON-Namen-Behandlung
5 Falsche Anzahl von Indizes
6 Feld mehrfach in einer EQUIVALENCE-Gruppe
7 Mehrfache EQUIVALENCE von COMMON
8 COMMON nach links erweitert
9 Nicht-COMMON-Variable in BLOCK DATA
10 Leere Liste fuer formatfreies WRITE
11 Nicht-ganzzahliger Ausdruck
12 Operandenmodus mit Operator nicht kompatibel
13 Operandanmischung nicht erlaubt
14 Fehlende ganzzahlige Variable
15 Fehlende Marke bei FORMAT
16 Wiederholungsfaktor 0
17 Formatwert 0
18 Zu tiefe Formatschachtelung
19 Marke keinem FORMAT zugeordnet
20 Ungueltige Benutzung einer Marke
21 Anweisung nicht erreichbar
22 Fehlendes Ende eines DO
23 Codeausgabe in BLOCK DATA
24 Nichtdefinierte Marke
25 RETURN in einem Hauptprogramm

26 Statusfehler bei READ
27 Falsche Operandenverwendung
28 Funktion ohne Parameter
29 Ueberlauf bei hexadezimaler Konstante
30 Division durch 0
32 Feldname wird erwartet
33 Falsches Argument bei ENCODE/DECODE

8.3.2. Fatale Fehler

Tritt ein fataler Fehler auf, dann beachtet der Compiler den Rest der logischen Quellzeile einschliesslich der Fortsetzungszeile nicht. Fatalen Fehlern werden ein Fragezeichen (?) und die Nummer der Quellzeile vorangestellt.

Fatale Fehler:

100 Falsche Marke
101 Anweisung nicht deutbar
102 Falscher Anweisungsabschluss
103 Falsche DO-Schachtelung
104 Falsche Datenkonstante
105 Fehlender Name
106 Falscher Prozedurname
107 Falsche DATA-Konstante oder falscher Wiederholfaktor
108 Falsche Anzahl von DATA-Konstanten
109 Falsche ganzzahlige Konstante
110 Ungueltige Marke
111 Kein Variablenname
112 Falscher Operator in logischem Ausdruck
113 Ueberlauf des Datenpuffers
114 Zeichenkette zu lang
115 Falsches Element in E/A-Liste
116 Falsche DO-Schachtelung
117 Name zu lang
118 Falscher Operator
119 Falsche Klammerung
120 Aufeinanderfolgende Operatoren
121 Falsche Indexverwendung
122 Nichzugelassener ganzzahliger Wert
123 Falsche Zeichenkette
124 DO-Bezug rueckwaerts
125 Falscher Anweisungsfunktionsname
126 Falsches Zeichen
127 Anweisung ausserhalb der Programmfolge
128 Fehlender ganzzahliger Wert
129 Falscher logischer Operator
130 Falsche Angabe nach INTEGER, REAL oder LOGICAL
131 Vorzeitiges EOF auf Eingabegeraet
132 Nichtzugelassene gemischte Operation
133 Funktionsaufruf ohne Parameter
134 Stack-Ueberlauf
135 Falsche Anweisung nach logischem IF

9. Binden von FORTRAN-Objektprogrammen

9.1. Kommandostruktur

Nach dem Erscheinen des Promptzeichens (%) des UDOS-Betriebssystems kann der FORTRAN-Binder (Linker) durch das Kommando FLINK aufgerufen werden; also:

```
%FLINK
```

Der FORTRAN-Binder meldet sich mit *.

Danach koennen dem Binder Angaben fuer das Binden der Objektprogramme gegeben werden.

Die vollstaendige Kommandostruktur lautet:

```
*   obj-ger/dateiname1.OBJ-sel1,
    obj-ger/dateiname2.OBJ-sel2,...
```

Es bedeutet:

obj-ger : Geraet, auf welchem das Objektprogramm mit dem Dateinamen dateiname.OBJ gespeichert ist.

-sel : Selektor (optional)

Werden die Geraete nicht oder nur teilweise angegeben, dann werden die entsprechenden Dateien auf allen Geraeten gesucht. Wenn die Erweiterung .OBJ des Dateinamens nicht angegeben wird, dann wird sie automatisch angenommen.

Die Kommandostruktur kann durch Selektoren beeinflusst werden. Werden die folgenden Selektoren verwendet, dann ist jeweils ein Minuszeichen (-) voranzustellen.

R — Bei Erkennen von -R wird der Binder sofort in den Anfangszustand (RESET) versetzt.

E oder E:name — FLINK wird verlassen und in das Betriebssystem zurueckgesprungen. Die FORTRAN-Bibliothek (FORLIB.OBJ) wird durchsucht, um noch existierende undefinierte globale Groessen zu definieren.
Die optionale Form E:name (name ist ein vorher in einem Modul definiertes globales Symbol) benutzt name als Startadresse des Programms.
Mit -E wird ein Programm geladen, welches dann mit IMAGE als ladbares Programm auf einer Diskette gespeichert werden kann.

G oder G:name — Nach Interpretation der aktuellen Kommandozeile wird die Ausfuehrung des Programms gestartet. Wenn das Programm eine FORTRAN-Bibliotheksroutine benoetigt, wird FORLIB.OBJ vor der Programmausfuehrung durchsucht, um eventuell existierende undefinierte globale Symbole zu defi-

nieren.
Die optionale Form G:name entspricht E:name.
Unmittelbar vor der Programmausfuehrung druckt FLINK die folgende Information:

Start-adresse	Adresse des hoechsten verfuegbaren Bytes	Seitenanzahl

[BEGIN EXECUTION]

U Alle undefinierten globalen Symbole werden ausgelistet.

M Alle definierten globalen Symbole und ihre Werte sowie die undefinierten, gefolgt von einem *, werden ausgelistet.

S Der Dateiname wird unmittelbar vor -S durchsucht, um undefinierte globale Symbole zu definieren.

Beispiele:

*-M	Auslisten aller globalen Symbole.
*MAINPROG,SUBPROG,ANWLIB-S	Es werden MAINPROG.OBJ und SUBPROG.OBJ geladen, und ANWLIB.OBJ wird zwecks Definition von undefinierten globalen Symbolen durchsucht.
*-G	Start der Ausfuehrung des Programms.

Der Anfang und das Ende vom Programm und Datenbereich werden ausgedruckt, wenn die Selektoren -U und -M notiert werden.

```
Beispiel: DATA      100    200
          PROGRAM  1000   2000
```

Weitere Selektoren sind:

E Wenn <dateiname>-E angegeben wird, wird das Programm unter dem gewaehlten Namen auf Diskette gerettet, aber nur dann, wenn in der Kommandozeile ein -E oder -G notiert wurde. Wenn es erforderlich ist, wird ein Sprung zum Start des Programms eingefuegt.

P und D -P und -D legen den Anfang fuer das naechste zu ladende Programm fest.

Syntax: -P:adresse oder -D:adresse

Von FLINK werden drei Plaetze fuer einen Sprung zur Startadresse freigelassen.
Wenn -D nicht angegeben wird, werden die Programme wie gewoehnlich geladen, ausser dass eines die Moeglichkeit hat, die Basisadresse festzulegen. Wird -D gegeben , dann werden alle Data- und Commonbereiche beginnend am Datenursprung geladen. Der Programmbereich beginnt am Programmursprung.

```
Beispiel: *-P: 200,F00
          DATA 200 300
          *-E
          *-P: 200 -D:400,F00
          DATA 400 4C0
          PROG 200 280
```

```
Beispiel fuer FLINK: %FLINK
                     *BEISP1,BEISP2-G
                     [304F  30AC  49]
                     [BEGIN EXECUTION]
```

9.2. FORTRAN-Binder-Fehlermeldungen

?NO START ADDRESS	Ein -G ist angegeben, aber kein Hauptprogramm geladen.
?LOADING ERROR	Die letzte Eingabedatei war keine korrekte FLINK-Objektdatei.
?FATAL TABLE COLLISION	Verfuegbarer Speicher zu klein.
?COMMAND ERROR	Nicht deutbares FLINK-Kommando.
?FILE NOT FOUND	Eine Datei aus der Kommandozeile existiert nicht.
%2ND COMMON LARGER /XXXXXX/	Die erste Definition des Commonblocks /XXXXXX/ war nicht die groesste.
%MULT.DEF.GLOBAL YYYYYY	Mehr als eine Definition fuer ein globales Symbol YYYYYY.
?OUT OF MEMORY	Verfuegbarer Speicher zu klein.
?<file> NOT FOUND	Wie ?FILE NOT FOUND, Dateiname wird gedruckt.
%OVERLAYING [PROGRAM] AREA [DATA]	Ein -D oder -P zestoert schon geladene Daten.

Beachte: Liegt der Datenanfang vor dem Programmbereich und wurde der Selektor -D angegeben, dann muss ausreichend Platz fuer die Daten sein. Das trifft besonders fuer den Disktreiber fuer FORTRAN zu, der MEMRY benutzt, um die Diskpuffer und FCBs Speicherbereichen zuzuordnen.

?INTERSECTING [PROGRAM] AREA [DATA]

Programm- und Datenbereich ueberlappen sich, und eine Adresse oder ein Eintrittspunkt liegt in der Ueberlappung.

?START SYMBOL -<name>- UNDEFINED

Nach -E: oder -G: ist das angegebene Symbol nicht definiert.

?ORIGIN [ABOVE] [BELOW] LOADER MEMORY, MOVE ANYWAY (Y OR N)?

Nach einem -E oder -G hat entweder der Daten- oder der Programmbereich einen Anfang oder ein Ende ausserhalb des Ladespeichers.
Betaetigen der Tasten Y <cr>: FLINK verschiebt den Bereich und arbeitet weiter.
Betaetigen einer anderen Taste: FLINK wird verlassen.
Wurde der Selektor -M ausgegeben, wird in beiden Faellen das Speicherbild gerettet.

?CAN'T SAVE OBJECT FILE

Diskettenfehler

9.3. FORTRAN-Laufzeit-Fehlermeldungen

9.3.1. Warnungen

Tritt eine Warnung auf, dann erfolgt die Angabe in der Form: **zz** . Nach 20 Warnungen wird in das Betriebssystem zurueckgesprungen.

Warnungen:

IB Eingabepuffergrenze ueberschritten
TL In Formatanweisung zu viele linke Klammern
OB Ausgabepuffergrenze ueberschritten
DE Ueberlauf des Dezimalexponenten (>99)
IS Integergroesse zu gross
BE Ueberlauf des Binaerexponenten

IN Eingabesatz zu lang
CN Ueberlauf bei Konvertierung REAL ---> INTEGER
SN Argument von Sinus zu gross
A2 Beide Argumente von ATAN2 sind 0
IO Unzulaessige E/A-Operation
BI Puffergroesse waehrend binaerer E/A-Operation ueberschritten
RC Negativer Wiederholfaktor in Formatanweisung

9.3.2. Fatale Fehler

Tritt ein fataler Fehler auf, dann erfolgt die Angabe in der Form: **zz** . Bei Auftreten eines fatalen Fehlers wird sofort in das Betriebssystem zurueckgesprungen.

Fatale Fehler:

ID Unzulaessiges Format in Formatanweisung
FO Formatbreite ist Null
MP Fehlender Punkt im Format
FW Formatbreite zu klein
IT E/A-Uebertragungsfehler
ML Fehlende linke Klammer in Formatanweisung
DZ Division durch Null bei REAL oder INTEGER
LG Unzulaessiges Argument bei LOG (Negativ oder Null)
SQ Unzulaessiges Argument bei SQRT
DT Datentyp stimmt nicht mit Formatfestlegung ueberein.
EF EOF durch READ erreicht

U 8 8 0 - P L Z / S Y S

Benutzerhandbuch

VORWORT

Die vorliegende Anwenderdokumentation gibt eine gekuerzte Beschreibung der hoeheren Programmiersprachen PLZ/SYS.
Eine ausfuehrliche Sprachbeschreibung kann der folgenden Standardliteratur entnommen werden:

- Anwenderdokumentation PLZ/SYS V.3.0 fuer das Betriebssystem UDOS 1526, VEB Robotron-Buchungsmaschinenwerk Karl-Marx-Stadt

Inhaltsverzeichnis Seite

1. Einleitung

PLZ/SYS ist eine hoehere Programmiersprache fuer Mikrocomputer, die vor allem zur Entwicklung von Systemprogrammen (Betriebssysteme, Dateisysteme, Compiler) entwickelt wurde. Ausserdem koennen auch Anwenderprogramme damit formuliert werden.
Im folgenden Text wird die PLZ/SYS-Programmiersprache Version 3.0 speziell fuer Mikrorechnersysteme auf der Basis des Mikroprozessors U880 beschrieben.

2. Sprachbeschreibung PLZ/SYS

2.1 Zeichenvorrat und Struktur

.Der Zeichenvorrat besteht aus folgenden Grundsymbolen:

- Buchstaben : A - Z , a - z
- Ziffern : 0 - 9
- Hexadezimalziffern : 0 - 9, A - F, a - f
- Sonderzeichen : + | - | * | / | =
  < | > | <= | >= | <>
  . | % | ^ | := | += | -=
  # | [|] | (|)
- Wortsymbole : ABS ; AND ; ARRAY
  BYTE ; CASE ; CONSTANT ; DEC ; DO
  ELSE; END ; ENTRY ; EXIT ; EXTERNAL
  FI; FROM; GLOBAL; IF; INC; INTEGER
  INTERNAL; LOCAL; MOD; MODULE; NIL
  NOT; OD; OR; PROCEDURE; RECORD
  REPEAT;RETURNS;SIZEOF;SHORT INTEGER
  THEN; TYPE; WORD; XOR; ANDIF; ORIF

  auch in Kleinbuchstaben schreibbar!
- Begrenzungszeichen : , | ; | : | Leerzeichen | Tabulator | Formularvorschub | Zeilenschaltung |

. Kommentar steht zwischen Ausrufezeichen und ist auch ueber mehrere Zeilen moeglich. Er kann dort stehen, wo ein Begrenzungszeichen stehen darf.
. Der Programmtext besteht aus :
  Deklarationen und
  Anweisungen
  Die Anweisungen bestehen aus Elementarausdruecken (Zeichenkette, Wortsymbol, Name oder Zahl), die durch beliebig viele Trennzeichen (Begrenzungszeichen oder Sonderzeichen) voneinander getrennt werden.

Besonderheit von PLZ : Zwischen 2 Anweisungen muss kein besonderes Zeichen stehen.

Verboten sind: Leerzeichen innerhalb eines Namens, Aneinanderreihung von Namen, Wortsymbolen oder Zahlen ohne Leerzeichen

2.2. Namen, Zahlen und Zeichenketten

2.2.1. Namen

Sie dienen zur Bezeichnung von Konstanten, Variablen, Schleifen, Typen und Prozeduren. Das 1. Zeichen muss ein Buchstabe sein, ansonsten kann neben Buchstaben und Zahlen auch der Unterstrich benutzt werden, z.B. HEX_ZAHL1.
Namen duerfen im jeweiligen Gueltigkeitsbereich nur einmal deklariert werden und muessen immer die gleiche Schreibweise haben.

2.2.2. Zahlen

Zahlen koennen als :

Dezimalzahlen z.B.: 12, 4000 oder
Hexadezimalzahlen z.B.: %12, %1A2B, %1b2b

geschrieben werden.

2.2.3. Zeichenketten

Zeichenketten werden in Apostrophe eingeschlossen, z.B. 'EOF'.
Nichtdruckbare Zeichen in Zeichenketten sind als zweistellige Hexadezimalzahlen anzugeben, z.B. %1B (Escape).

Sondersymbole fuer haeufig in Zeichenketten gebrauchte Steuerzeichen (auch in Kleinbuchstaben schreibbar!):

%L Zeilenschaltung (line feed),
%T Tabulator,
%R Wagenruecklauf (return),
%P Formularvorschub (page),
%% einmal das Zeichen %
%Q Apostroph

Beispiel: 'Erste Zeile%R zweite Zeile%r'

2.3. Konstanten

Eine Konstante ist eine Zahl, Zeichenkette oder Ausdruck, der nur Konstanten enthaelt. Ein Konstantenausdruck darf nur die Operationen +, -, NOT, als Vorzeichenoperatoren und +, -, *, /, MOD, AND, OR, XOR als Verknuepfungsoperatoren enthalten.

Beispiel:
```
. %10           : Zahl
  'KONSTANTE'   : Zeichenkette
  %10*5         : Ausdruck
```

Konstantendefinition:
Damit wird ein Name als Synonym fuer einen konstanten Wert eingefuehrt. Der Wert , der dem Konstantennamen zugewiesen wird, muss sich aus einem konstanten Ausdruck berechnen lassen, d.h., allen darin vorkommenden Namen muss vorher ein Wert zugewiesen worden sein. Konstanten sind im gesamten Modul gueltig, aber nicht darueber hinaus.

Beispiel:
```
CONSTANT
        SATZLAENGE    := 64
        PUFFERLAENGE  := 4*SATZLAENGE
        SEMIKOLON     := ';'
```

2.4. Datentypen

Ein Datentyp bestimmt, welche Werte eine Variable annehmen kann und welche Operationen mit ihr ausgefuehrt werden koennen.

Eine Typendefinition verbindet einen Namen mit einem Typ.

Beispiel:
```
TYPE
    ZEICHEN          BYTE
    SONDERZEICHEN    ZEICHEN
```

Der Grundtyp der beiden Typen ZEICHEN und SONDERZEICHEN ist Byte. Das bestimmt die Operationen, die mit Variablen beider Typen ausgefuehrt werden koennen.

Der Typ kann auch direkt durch eine Variablendeklaration festgelegt werden (s. Abschn. 2.5.).

2.4.1. Einfache Typen

WORD	- nicht negative ganze Zahl von 0 bis 65535 (16 Bit)
BYTE	- nicht negative ganze Zahl von 0 bis 255, kann auch Einzelzeichen darstellen (8 Bit)
INTEGER	- ganze Zahl von -32768 bis 32767 (16 Bit)
SHORT_INTEGER	- ganze Zahl von -128 bis 127 (8 Bit)
Zeiger (^)	- eine maschinenabhaengige Groesse, die eine Speicheradresse darstellt

Beispiel:
```
TYPE
     PBYTE    ^BYTE
     PWORD    ^WORD
```

2.4.2. Strukturierte Typen

ARRAY - Ein Feldtyp besteht aus einer festen Anzahl von Komponenten, die alle vom gleichen Typ sind. Jedes Element eines Feldes ist durch einen Index bezeichnet, der vom Typ WORD oder BYTE sein darf. Die Indexnummerierung beginnt bei 0. Die anzugebende Elementenzahl muss ein konstanter Ausdruck sein.
Feldelemente koennen beliebige strukturierte Typen sein, d.h., Felder aus Feldern sind moeglich.

Beispiel:
```
TYPE
     PUFFER     ARRAY [128 BYTE]
     MATRIX     ARRAY [100 100 BYTE]
!MATRIX IST EIN ZWEIDIMENSIONALES FELD MIT 10000 BYTE!
```

Wenn eine Zeichenkette zur Initialisierung eines Feldes variabler Laenge benutzt wird, dann kann sie in Form mehrerer kuerzerer Zeichenketten geschrieben werden.
Beide Anweisungen sind aequivalent:

```
A     ARRAY [* BYTE]:='ABCD%REFGH'
A     ARRAY [* BYTE]:='ABCD%R'
                      'EFGH'
```

RECORD - Ein Datensatztyp besteht aus einer festen Anzahl von Komponenten, die verschiedenen Typs sein koennen und nicht durch einen Index, sondern durch einen Komponentennamen ausgewaehlt werden.

Beispiel:
```
TYPE
PATIENT RECORD [ALTER,GROESSE,GEWICHT BYTE
                GESCHLECHT            BYTE
                GEBURTSDATUM RECORD [TAG, MONAT, JAHR BYTE]
                ZIMMER                WORD]
ZEIKETT RECORD [LAENGE BYTE, ZEICHEN ARRAY [50 BYTE]]
```

Feldnamen von RECORD-Typen sind lokal bezueglich des RECORD-Typs, in dem sie auftreten. Lokal bedeutet, dass der gleiche Name in anderer Bedeutung in weiteren RECORD-Typen auftreten darf.

2.4.3. Zeigertypen

Allgemeine Form der Zeigerdefinition:

```
Zeigertypenname     ^Typ
```

Zu einer Variablen, Record oder Array kann man nicht nur direkt ueber den Namen zugreifen, sondern auch ueber einen Zeigerwert, der als Zeiger auf den Typ der Variablen deklariert wurde.
Einen Zeigerwert kann man interpretieren als Adresse einer

Variablen.

2.4.3.1. Zeiger auf einen einfachen Typ

Ein Zeiger auf BYTE, INTEGER, SHORT_INTEGER, andere Zeiger oder auf selbstdefinierte einfache Typen kann zum Zugriff auf jede Variable des entsprechenden Typs benutzt werden, auch wenn diese Variable ein Element einer Struktur (z.B. ARRAY) ist.
Beispiel:

```
CONSTANT
     LEN:=%08
TYPE
     ZEIGB      ^BYTE
     ZEICHEN    BYTE
     ZEIGC      ^ZEICHEN
     ZEIGB2     ^^BYTE
     A ARRAY    [10 BYTE]
     B ARRAY    [10 BYTE]
     BUF ARRAY  [LEN A]       !Feld von 8 mal 10 Bytes !
     BUF_PTR    ^BUF
```

ZEIGB kann als Zeiger auf jede Variable des Typs BYTE benutzt werden, z.B. auf ein Element des Feldes A.
ZEIGC kann dagegen nur als Zeiger auf Variable vom Typ ZEICHEN benutzt werden.
ZEIGB2 kann auf alle Variablen zeigen, die selbst als Zeiger auf BYTE vereinbart sind , nicht aber z.B. auf Variablen des Typs ZEIGC.

2.4.3.2. Zeiger auf einen strukturierten Typ (ARRAY, RECORD)

Dieser Zeiger wird durch Benutzung des Typennamens der Struktur deklariert, eine Typdefinition ist also unbedingt notwendig. Der Zeiger weist dann auf die Basisadresse (den ersten Speicherplatz) der Struktur, er kann auch als Zeiger auf einzelne Elemente benutzt werden. Man kann mit seiner Hilfe auf die einzelnen Elemente zugreifen.
Fortsetzung des letzten Beispiels:

```
TYPE
     ZEIGA      ^A
     R RECORD   [F1, F2 BYTE, F3 ZEICHEN]
     S RECORD   [F1, F2 BYTE, F3 ZEICHEN]
     ZEIGER     ^R

INTERNAL
     .
     .
     .
ZEIGER.F2 := 150
```

ZEIGA zeigt nur auf Felder des Typs A (nicht auf solche des Typs B).
ZEIGER kann nur auf den Datensatz des Typs R zeigen.

2.4.4. Typkompatibilitaet

Der Compiler prueft, ob die in Ausdruecken, Zuweisungen und Prozedurparametern auftretenden Variablen miteinander vertraeglich sind und zeigt andernfalls Fehler an.

2.4.4.1. Variablen einfacher Typen (ausser Zeiger)

Zwei Variablen einfacher Typen (ausser Zeiger) sind nur dann miteinander vertraeglich,wenn sie in den Variablendeklarationen mit uebereinstimmenden Typennamen deklariert wurden.

Beispiel:

```
TYPE
     ZEICHEN    BYTE
INTERNAL
     A,B        BYTE
     C          ZEICHEN
     D          BYTE
     E          WORD
```

A,B und D sind miteinander vertraeglich.
C und E sind mit keiner der anderen Variablen vertraeglich.

2.4.4.2. Variablen mit strukturierten Typen (ARRAY, RECORD)

Zwei Variablen mit strukturierten Typen haben nur dann kompatible Typen, wenn sie entweder in der gleichen Deklarationsliste auftauchen oder in ihren Deklarationen der gleiche, selbst definierte Typname verwendet wird.

Beispiel:

```
TYPE
     PUFFER     ARRAY  [128 BYTE]
     DATUM      RECORD [TAG, MONAT, JAHR BYTE]
INTERNAL
     A,B        PUFFER
     C,D        ARRAY  [128 BYTE]
     E          PUFFER
     G          DATUM
     K          RECORD [TAG, MONAT, JAHR BYTE]
```

A, B und E sind miteinander vertraeglich, ebenso C mit D.
G und K sind inkompatibel mit allen Variablen.

2.4.4.3. Zeigervariable

Zwei Zeigervariable sind miteinander vertraeglich, wenn die Deklarationen der Objekte, auf die sie zeigen, miteinander vertraeglich sind.

Typen ist es moeglich, statt der Elementeanzahl das Zeichen * anzugeben. Die Elementeanzahl wird dann von der Anzahl der Initialisierungselemente bestimmt. Dafuer kann dann eine in Klammern eingeschlossene Konstantenliste oder eine Zeichenkette mit 8-Bit-Feldelementen angegeben werden.
Beispiel:

```
INTERNAL
     A     ARRAY [* WORD] := [%FFFF,%8000,%F01B]
     B     ARRAY [* BYTE] := ['Y','N','A','Q']
     M     ARRAY [* BYTE] := 'YOUR NAME?'
     CH    ARRAY [* BYTE] := 'Bei sehr langen Zeichenketten'
                             'ist auch eine Unterbrechung'
                             'moeglich.'
```

Negative Initialwerte werden in Klammern gesetzt:
Beispiel:

```
     A     ARRAY [* INTEGER] := [0,[-1],2,[-3]]
```

Feldtypen koennen auch durch Typendefinitionen eingefuehrt werden. In den Variablendeklarationen tritt dann nur noch der festgelegte Typenname auf:
Beispiel:

```
TYPE
     PUFFER     ARRAY [128 BYTE]
     MATRIX     ARRAY [8 8 BYTE]
INTERNAL
     ZEICHENPUFFER   PUFFER
     SCHACHBRETT     MATRIX := 0
```

Zu Feldelementen wird durch Angabe des Variablennamens zugegriffen. Diesem folgt die Indexliste, wobei die Indizes konstante Ausdruecke oder Variablen mit den Basistypen WORD oder BYTE sein koennen:

```
     Feldname [Feldindex1 Feldindex2 ...]
```

Beispiel:

```
     ZEICHENPUFFER [10]
     SCHACHBRETT   [ZEILE SPALTE]
```

2.5.2.2. Datensatzvariablen und Elementebezeichner

Datensatzvariablen sind vom Typ RECORD.

Die Form der Deklaration ist:

```
          Datensatzname  RECORD     [Elemente-Deklaration]
oder      Datensatzname  Datensatztyp
```

Beispiel:

```
TYPE
     VEKTOR     RECORD     [UNIT BYTE, ADR ^BYTE, LEN WORD]
INTERNAL
     S          RECORD [X BYTE,Y,Z WORD] := [?,0,...]
     VEK1       VEKTOR := [CONOUT,?,?,100]
```

"?" kann fuer nicht zu initialisierende Werte stehen.

Elementebezeichner:
Damit wird ein Element eines Datensatzes ausgewaehlt:
Datensatzkennzeichnung.Elementebezeichner

Die Datensatzkennzeichnung kann sein:
- ein Datensatzname
- eine Zeigervariable
- eine Feldvariable

Beispiel:
```
TYPE
     PATIENT RECORD [NAME ARRAY [64 BYTE]
                     ALTER, GROESSE,  GEWICHT BYTE
                     GESCHLECHT                BYTE
                     GEBURTSDAT RECORD [TAG,MO,JAHR BYTE]]

INTERNAL
     STATION ARRAY  [100 PATIENT]
     I       BYTE
```

Die Auswahl eines Datensatzelementes erfolgt z.B.:

```
STATION[I].GEBURTSDAT.TAG
STATION[I].ALTER
```

2.5.3. Zeigerbezogene Variablen

Zeigerbezogene Variablen sind solche, auf die ein Zeiger verweist. Der Basistyp einer Zeigervariablen ist der Typ Zeiger (^).
Der definierte Typ eines Zeigers setzt sich aus dem Verweisniveau (d.h. der Anzahl der ^) und dem Typ zusammen, auf den der Zeiger verweist.
Ein Zeiger auf eine Komponente einer Struktur wird als Zeiger auf den Typ der Komponente deklariert, er kann also auch auf Variablen entsprechender Typen ausserhalb eines Feldes oder Datensatzes zeigen.
Ein Zeiger, der als Verweis auf ein ganzes Feld oder einen Datensatz deklariert ist, kann dagegen nur auf den Anfang der entsprechenden Struktur zeigen. Mit Zeigern sind nur folgende Operationen moeglich:

- Test auf Gleichheit oder Groessenrelation;
- der Zeiger-Operator ^, der die Variable bereitstellt, auf die gezeigt wird;
- der Adressoperator #, der die Adresse einer Variablen (also einen Zeiger) bereitstellt;
- die Operationen INC und DEC, die einen Zeiger auf die naechsthoehere bzw. naechstniedrigere Position des Typs setzen, auf den gezeigt wird. Die Operation ist nicht immer identisch mit einem Veraendern der Speicheradresse um 1! Es werden so viele Speicherplaetze uebersprungen, wie von der Struktur benoetigt werden, auf die gezeigt wird.

Beispiel:

```
TYPE
     PB          ^BYTE
     PTR_PB      ^^BYTE
INTERNAL
     B     BYTE := 5
           .
           .
     PB       := #B
     PTR_PB   := #PB
```

Dann sind:

Wert von B	ist 5
Wert von PB	ist #B
Wert von PB^	ist 5
Wert von PTR_PB	ist #PB
Wert von PTR_PB^	ist #B
Wert von PTR_PB^^	ist 5

Ein Zeiger auf ein Feld kann zum Verweis auf ein Feldelement benutzt werden:

Beispiel:

```
TYPE
     TABELLE    ARRAY [100 WORD]
INTERNAL
     ZEIGTAB    ^TABELLE
           .
           .
     ZEIGTAB := #TABELLE
```

2.5.4. Gueltigkeitsbereich von Variablen

Variablen und Prozeduren koennen verschiedene Gueltigkeitsbereiche haben. Sie werden folgendermassen festgelegt:

- EXTERNAL
  Die hier aufgezaehlten Variablen und Prozeduren sind in einem anderen Modul als GLOBAL vereinbart worden und koennen auch in diesem Modul benutzt werden.
- INTERNAL
  Variablen und Prozeduren gelten nur in diesem Modul.
- GLOBAL
  Variablen und Prozeduren gelten in diesem Modul und koennen auch von anderen Modulen aus erreicht werden (ueber EXTERNAL)
- LOCAL
  Variablen gelten nur in der zu LOCAL gehoerenden Prozedur.
- CONSTANT (Konstantendefinition)
  Die Konstanten sind im gesamten Modul gueltig.

Neue Namen werden deklariert:
- in Variablen-, Typ- oder Konstantendefinitionen des Moduls,
- als Datensatz (RECORD)-Elemente,
- als formale Eingabeparameter oder Rueckgabeparameter von Prozedurdeklarationen,

- in der LOCAL-Liste einer Prozedur.
Auf Elementenamen von Datensaetzen kann auch ausserhalb der Datensatzdefinition zurueckgegriffen werden.
Prozedurnamen gelten immer im gesamten Modul, formale Parameter dagegen nur innerhalb des Prozedurkoerpers.

2.6. Ausdruecke

Ausdruecke sind Rechenregeln zum Vergleich von Variablenwerten oder zur Erzeugung neuer Werte durch Anwendung von Operatoren. Sie bestehen aus Operanden, Operatoren und Prozeduraufrufen,die genau einen Wert zurueckgeben.
In PLZ ist maximal ein Verknuepfungsoperator in je einem Ausdruck und noch hoechstens ein Vorzeichenoperator fuer jeden Operanden zugelassen. Verschachtelte Konstruktionen mit Klammerung sind nicht zugelassen, sie koennen ueber Zwischenschritte ausgedrueckt werden.

2.6.1. Operatoren

Es duerfen nur Ausdruecke eines Typs miteinander verknuepft werden.

2.6.1.1. Verknuepfungsoperatoren

- Arithmetische Operatoren:

  Sie sind definiert fuer Variablen der Basistypen WORD, BYTE, INTEGER, SHORT_INTEGER (arithmetische Standardtypen)
  Der Typ des Ergebnisses ist identisch mit dem der Operanden.

*	Multiplikation
/	Division
MOD	Moduldivision (Restdivision)
+	Addition
-	Subtraktion

- Vergleichsoperatoren:

  Sie sind definiert fuer Variablen der Basistypen WORD, BYTE, INTEGER, SHORT_INTEGER und Zeiger.Die Vergleiche werden vorzeichenbehaftet ausgefuehrt, wenn der Variablentyp ein Vorzeichen hat:

=	gleich
<>	ungleich
<	kleiner als
>	groesser als
<=	kleiner oder gleich
>=	groesser oder gleich

  Vergleichsoperatoren duerfen nur im Bedingungsausdruck einer IF-Anweisung verwendet werden, da keine Daten er-

zeugt werden.

- Logische Operatoren

Sie sind fuer die arithmetischen Standardtypen zugelassen. Das Ergebnis ist identisch mit dem Typ der Operanden:

```
AND  logisches UND
OR   logisches ODER
XOR  logische ANTIVALENZ
```

2.6.1.2. Vorzeichenoperatoren (Unaeroperatoren)

Der Typ der Resultate ist identisch mit dem definierten Typ des Operanden.

Operator	Operation	Grundtyp des Operanden
+	plus	arithmetische Typen
-	minus	arithmetische Typen
ABS	Absolutwert	
NOT	log.Komplement	arithmetische Typen
INC	Zeigerwert + Laenge des Basistyps, auf den gezeigt wird	Zeiger
DEC	Zeigerwert - Laenge des Basistyps, auf den gezeigt wird	Zeiger
SIZEOF	Anzahl der Feldelemente, das Ergebnis ist ein konstanter Wert!	ARRAY, RECORD
#	Adresse einer Variablen	alle Typen

Der SIZEOF-Operator kann auf Feldnamen oder Feldtypennamen angewendet werden.

Beispiel:

```
TYPE
     MATRIX     ARRAY [8 4 WORD]
INTERNAL
     CLARA      ARRAY [* BYTE] := 'NETTE'
                .
                .
                .
     SIZEOF     MATRIX          (Wert ist 64)
     SIZEOF     CLARA           (Wert ist 5)
```

Der Adressenoperator # kann auf Variablen jedes Typs angewendet werden. Er erzeugt einen Zeiger auf den entsprechenden Variablentyp. Wenn der Operator auf einen Feldnamen angewendet wird, entsteht ein Zeiger auf den Feldtyp. Wenn dagegen nach dem Feldnamen noch ein Index angegeben wird, erzeugt man einen Zeiger auf den Typ der Feldelemente. Entsprechendes gilt bei Datensaetzen. Der Adressoperator

kann als inverse Operation zum Zeigeroperator aufgefasst werden. So ist beispielsweise :

#ZEIGERA^ identisch mit ZEIGERA.

2.6.2. Prozeduraufruf

Form:

Prozedurname (Liste der aktuellen Parameter)

Ein Prozeduraufruf bewirkt die Ausfuehrung einer vorher definierten Prozedur mit den im Aufruf angegebenen aktuellen Parametern. Die Zuordnung der aktuellen Parameter erfolgt entsprechend der Reihenfolge in der Parameterliste. Es wird grundsaetzlich nur der Wert der Parameter uebergeben.

Jeder aktuelle Parameter kann sein:

- eine einfache Variable,
- ein konstanter Ausdruck,
- eine Adressvariable.

Der Typ und die Anzahl der aktuellen Parameter muessen mit den jeweiligen formalen Parametern uebereinstimmen. Da ein Parameter auch ein Zeiger auf eine Variable sein kann, ist auf diese Weise auch die Uebergabe der Adresse einer Variablen moeglich; das entspricht einer Parameteruebergabe ueber deren Adressen (Namen).
Wenn ein Prozeduraufruf Bestandteil eines Ausdrucks ist, muss die Prozedur genau einen Parameter vom richtigen Typ zurueckgeben.

Beispiel:

```
INTERNAL
    P    PROCEDURE (EIN1 BYTE, EIN2 ^BYTE)
         RETURNS   (WERT BYTE)
         ENTRY
              .
              .
              .
         END
MASKE    Y    Z    BYTE
ZEIGERZ                ^BYTE
```

Einige zulaessige Ausdruecke sind:

```
Y + P (Z ZEIGERZ)
P (Y #Z) AND MASKE
```

2.7. Anweisungen

2.7.1. Einfache Anweisungen

Einfache Anweisungen sind:

- Ergibtanweisungen,
- Typenumwandlung,
- Prozeduranweisung (Prozeduraufruf),
- Rueckkehranweisung (RETURN-Anweisung),
- Schleifensteuerungsanweisung.

Sie enthalten keine untergeordnete Anweisungsarten.

2.7.1.1. Ergibtanweisung

Form:

```
Variable := Ausdruck
```

Durch eine Ergibtanweisung wird einer Variablen der Wert eines Ausdrucks zugewiesen. Dabei muessen Variable und Ausdruck vom gleichen Typ sein.

Beispiel:

```
FLAECHE := LAENGE * BREITE
ZEIGER  := INC ZEIGER
KODE    := KODE1 AND MASKE
ADRESSE := #ZEICHEN
```

2.7.1.2. Typumwandlung

Damit wird der Wert eines Ausdrucks eines bestimmten Typs einer Variablen eines anderen Typs zugewiesen.
Umwandlungen zwischen strukturierten Typen (ARRAY und RECORD) und einfachen Typen sind nicht erlaubt!

Die dabei moeglichen Faelle werden entsprechend der folgenden Tabelle behandelt:

Typ der Variable	Typ des Ausdrucks BYTE	WORD	SHORT_INTEGER	INTEGER	Zeiger
BYTE	=	v	=	v	v
WORD	r	=	r	=	w
SHORT_INTEGER	=	v	=	v	v
INTEGER	=	=	s	=	w
Zeiger	r	r	r	r	=

= keine Aenderung
r Argument kommt rechtsbuendig in ein Feld
s Das Vorzeichen des Arguments bleibt erhalten
v Verkuerzung, es werden nur die 8 niederwertigen Bits weiterverwendet
w Verkuerzung, es werden nur die 16 niederwertigen Bits weiterverwendet

PLZ bewahrt so viele Bits des Wertes wie moeglich.

Beispiel:

```
INTERNAL
     ZAEHLER          INTEGER
     KLEINEZAHL       SHORT_INTEGER
     INDEX            WORD
```

Anweisungen damit sind:

```
     ZAEHLER := INTEGER KLEINEZAHL
     INDEX   := WORD ZAEHLER
```

2.7.1.3. Prozeduranweisung

Form:

Namen der Ergebnisvariablen := Prozeduraufruf

Die Prozeduranweisung stellt eine spezielle Form der Wertzuweisung dar, wobei auf der linken Seite des Ergibt-Zeichens die Namen der aktuellen Rueckgabeparameter stehen (wenn vorhanden). Die Anzahl der Rueckgabeparameter muss mit der Anzahl der Parameter in der RETURN-Liste der Prozedurdefinition uebereinstimmen.

Beispiele:

```
     GROSSZAHL          := MAX (X,I)
     INITIALISIERUNG
     TAG, MONAT, JAHR   := DATUMSRECH
     DRUCKEZEICHEN ('A')
```

2.7.1.4. Rueckkehranweisung

Form:

RETURN

Die Rueckkehranweisung bewirkt das Verlassen eines Prozedurkoerpers und die Weiterfuehrung des Programms, das die Prozedur aufgerufen hat. Unmittelbar vor dem END der Prozedur wird die Rueckkehr in jedem Fall angenommen, d.h., die Ruekkehranweisung ist nur zum Verlassen einer Prozedur an einer anderen Stelle notwendig.

2.7.1.5. Schleifensteueranweisung

Es gibt zwei Arten von Shleifensteueranweisungen:

- Austritts-(EXIT)-Anweisung

  Sie bewirkt ein Verlassen des innersten DO-OD-Blockes, der die Austrittsanweisung enthaelt. Das Programm wird mit der Anweisung, die auf das betreffende OD folgt, fortgesetzt.

Form:

```
                EXIT
        oder    EXIT FROM Markenname
```

- Wiederhol-(REPEAT)-Anweisung

Sie bewirkt eine Fortsetzung des Programms mit der ersten Anweisung des DO-OD Blockes, der die Wiederholanweisung enthaelt. Das ist die erste Anweisung nach DO.
Form:

```
                REPEAT
        oder    REPEAT FROM Markenname
```

Zusaetzlich kann in der Schleifensteueranweisung der Name einer Marke angegeben werden. Die Anweisungen beziehen sich dann nicht auf den innersten DO-OD Block, sondern auf den, der mit der entsprechenden Marke vor DO versehen ist.

2.7.2. Strukturierte Anweisung

Unter strukturierten Anweisungen werden solche verstanden, die andere Anweisungen enthalten, die bedingt (Bedingungsanweisung) oder wiederholt (Schleifenanweisung) ausgefuehrt werden. Eine Verbundanweisung ist in PLZ nicht erforderlich, da die strukturierten Anweisungen bereits die Kennzeichnung enthalten, auf welche Anweisungen sie sich beziehen.

2.7.2.1. Bedingungsanweisungen

Die bedingte Ausfuehrung der Anweisungen gestattet Wenn-Dann-Anweisungen und Fallauswahlanweisungen.
Die bedingten Konstruktionen 'ANDIF' und 'ORIF' sind erlaubt.

2.7.2.1.1. Wenn-Dann-Anweisung

Form:

```
IF Bedingungsausdruck THEN Anweisungen FI    oder
IF Bedingungsausdruck THEN Anweisungen ELSE Anweisungen FI
```

Der Bedingungsausdruck ist die einzige Konstruktion, in der die Vergleichsoperatoren (s. Abschn. 2.6.1.1.) auftreten duerfen, da dabei kein explizierter Datenwert erzeugt wird.

Beispiel:

```
IF a[i] > a[j] THEN
   x := a[i]  a[i] := a[j]  a[j] := x
FI
IF i  <= 5 THEN prozess (1) ELSE
   IF i <= 8 THEN prozess (2) ELSE
      IF i <= 13 THEN prozess (3) ELSE prozess (4) FI
   FI
FI
```

Die Anweisung bewirkt die Ausfuehrung der Anweisungen zwischen THEN und ELSE, wenn ein Bedingungsausdruck wahr ist (<>0).
Wenn der Bedingungsausdruck falsch ist (=0), dann werden die Anweisungen zwischen ELSE und FI ausgefuehrt; der ELSE-Fall kann auch entfallen.

2.7.2.1.2. Fallauswahlanweisung

Form:

```
IF Verteilerausdruck
    CASE Vergleichskonstanten THEN Anweisungen
    CASE ...
      .
      .
      .
    ELSE Anweisungen                    !kann entfallen!
FI
```

Die Fallauswahlanweisung stellt eine Erweiterung der Wenn-Dann-Anweisung dar. Sie erlaubt die Ausfuehrung verschiedener Anweisungsfolgen in Abhaengigkeit vom Wert eines Variablenausdrucks. Es wird die Anweisungsfolge ausgefuehrt, bei der eine der davor angegebenen Vergleichskonstanten mit dem Wert des Verteilerausdrucks uebereinstimmt. Falls dies fuer keine der angegebenen Vergleichskonstanten zutrifft, wird die Anweisungsfolge nach ELSE ausgefuehrt.Wenn ELSE nicht benutzt wurde, dann wird das Programm nach dem abschliessenden FI der Fallanweisung fortgesetzt.

Beispiel:

```
IF P1
   CASE '=' THEN AUSG(P2,P3)
   CASE 'C' THEN AUSGN(#'(%R')
   CASE ',' THEN AUSG(P4,P5)                ELSE
                 AUSGN(#'ERROR 5%R')
FI
```

2.7.2.2. Schleifenanweisung

Form:

```
Markenname:   DO   Anweisungen   OD        oder
              DO   Anweisungen   OD
```

In PLZ gibt es anstatt vieler verschiedener Schleifenarten eine einheitliche Schleifenanweisung. Die Anweisungsfolge zwischen DO und OD wird solange wiederholt, bis dies durch die Schleifensteueranweisung EXIT oder durch RETURN unterbrochen wird.
Die REPEAT-Anweisung bewirkt den vorzeitigen Sprung zum Schleifenanfang. Die Schleifensteueranweisungen koennen in Bedingungsanweisungen auftreten. Auf diese Weise lassen sich gleiche Wirkungen erzielen wie mit dem FOR, WHILE,

UNTIL-Konstruktionen anderer Programmiersprachen. Es ist zu beachten, dass DO-OD und IF-FI-Bloecke keinen Einfluss auf die Gueltigkeitsbereiche von Variablen haben.

Beispiele:

```
      !Geschachtelte  Laufanweisung mit Austritt aus dem !
      !innersten Niveau!
i := 1
m1: DO
      IF i > maxi THEN EXIT FI
         j := 1
    m2: DO
          IF j > maxj THEN EXIT FI
          IF a [i,j] = such THEN EXIT FROM m1 FI
             j := j + 1

        OD
        i := i + 1  !Ziel, wenn j > maxj!
    OD
!Ziel,wenn i > maxi  oder  a[i,j] = such  und  j <= maxj!

!WHILE beding DO...-Schleife mit Pruefung am Anfang!

DO
  IF NOT beding THEN EXIT FI
      .
      .
      .
OD

!DO...UNTIL beding...-Schleife mit Pruefung am Ende!

DO
     .
     .
     .
   IF beding THEN EXIT FI
OD
```

2.8. Prozedurdeklaration

Eine Prozedurdeklaration definiert einen ausfuehrbaren Teil eines Programms und gibt ihm einen Namen, unter dem er aufgerufen werden kann.

allgemeine Form:

```
Prozedurname     PROCEDURE (Liste der formalen Uebergabepa-
                           ramter)
                 RETURNS   (Liste der formalen Rueckgabepa-
                           rameter)
                 LOCAL     (Variablendeklaration ohne Ini-
                           tialisierung, nur innerhalb der
                           Prozedur gueltig!)
                 ENTRY
                       Anweisungen
                 END   Prozedurname
```

RETURNS- und LOCAL-Teil sowie die Uebergabeparameterliste koennen entfallen!
Die Parameter koennen alle einfache Typen haben, sie duerfen aber keine Felder oder Datensaetze sein! Trotzdem koennen Prozeduren nichtlokale Felder und Datensaetze bearbeiten, indem Zeiger auf diese Strukturen als Parameter uebergeben werden. Eine Prozedurdeklaration kann auch lokale Variable beinhalten, die nur innerhalb der Prozedur gueltig sind.
Wenn die Prozedur genau einen Rueckgabeparameter liefert, darf sie in Ausdruecken aufgerufen werden, ansonsten nur durch eine Prozeduranweisung.
Achtung! Alle Prozeduren muessen vor ihrer Benutzung deklariert werden. Diese Bedingung bereitet Probleme, wenn Prozeduren sich gegenseitig aufrufen (A ruft B auf, B wiederum A). Die Prozeduren sind dann verschiedenen Moduln zuzuordnen und ueber GLOBAL/EXTERNAL-Deklarationen zu verbinden.

Beispiel: (s. Abschn. 5)

2.9. Programme und Moduln

Ein PLZ-Programm besteht aus einer Menge von Moduln, die erst durch ein Bindeprogramm (LINK, PLINK) verbunden werden.
Sowohl Variable als auch Prozeduren, die in einem Modul als GLOBAL deklariert sind, koennen in anderen Moduln benutzt werden, wenn sie dort als EXTERNAL eingefuehrt werden.

ACHTUNG: Ausfuehrbare Anweisungen duerfen nur in Prozeduren auftreten! Das sogenannte "Hauptprogramm" ist ebenfalls als GLOBAL-Prozedur zu formulieren, deren Name ist dann dem Linker als Eintrittspunkt (E=...) mitzuteilen.

Beispiel: (s. Abschn. 5)

3. Bedienung

3.1. Compiler PLZSYS

- Quellprogramme werden i.allg. mit dem UDOS-Texteditor erstellt.
- Der Name der Quelldatei muss mit .S oder .s enden.
- Der Compiler erzeugt eine Listendatei, deren Name auf .L endet. Diese enthaelt die Quelltextzeilen mit einer fortlaufenden Numerierung. Durch Aufwaertspfeile wird auf wahrscheinlich fehlerhafte Stellen der Quelltextzeilen hingewiesen, die eindeutig den darunter angegebenen Fehlernummern zugewiesen werden koennen.
- Weiterhin wird eine Datei erzeugt, die einen Zwischenkode enthaelt und deren Namen auf .Z endet. Diese kann vom Interpreter ZINTERP, vom Kodegenerator PLZCG oder vom PLZ-Linker PLINK weiterverarbeitet werden.

3.1.1. Compileraufruf

%PLZSYS Dateiname[.S] Optionen (wahlweise)

".S" im Dateinamen kann weggelassen werden.
Wenn mehrere Optionen auftreten, sind diese durch Leerzeichen oder TAB zu trennen!

Moegliche Optionen:

L=Dateiname	- die Listendatei erhaelt einen anderen Namen; wenn Listendatei nicht benoetigt, dann: L=¤NULL
O=Dateiname	- die Objektdatei erhaelt einen anderen Namen; wenn Objektdatei (Zwischenkode) nicht benoetigt, dann: O=¤NULL
P	- sofortige Ausgabe der Listen auf dem Geraet SYSLIST (Grundzuweisung: Bildschirm)
D=Zeichenkette	- Ueberschrift fuer Listen, z.B. Daten, Version (max. 18 Zeichen; keine Leerzeichen, TAB, Semikolon benutzen!)
ND,NC	- ohne die Angabe von ND und NC werden Symbolinformationen erzeugt, die bei Anwendung des Testhilfsprogramms PDT noetig sind. Werden die Optionen angegeben, verkuerzen sich der Z-Code-Umfang um 18-25 % und die Uebersetzungszeit um ca. 15 %. ND bedeutet keine Erzeugung von Adress- bzw. Symbolinformationen lokaler Variablen, waehrend NC keine Informationen ueber Konstantenzuweisungen erzeugt.
E	- Es wird eine Datei *.E erzeugt, die bei aufgetretenen Uebersetzungsfehlern ausschliesslich Fehlermitteilungen sowie die zugehoerige Quellzeile enthaelt.

Der Compiler benutzt folgende logische Geraete:

2 (CONOUT)	- Mitteilung zur Konsole
3 (SYSLST)	- Listenausgabe, wenn Option P
4	- Quelldatei
5	- Listendatei
6	- Objektdatei (Zwischenkode)
7	- Arbeitsdatei (wird nach Compilerlauf automatisch geloescht)

3.1.2. Besondere Compilereinschraenkungen

(1) Es wird der Standard-ASCII-Zeichensatz verwendet. Gross- und Kleinschreibung sind zugelassen, werden aber als verschiedene Zeichen gewertet; die gemischte Schreibweise von Schluesselwoertern ist nicht zulaessig, z.B. GLOBAL und global sind moeglich, nicht aber Global! Hex.-Zahlen und Kettensonderzeichen koennen beliebig geschrieben werden.

Beispiel:

```
%AbC        '1.Zeile %R 2. Zeile %r'
```

(2) Quelltextzeilen mit mehr als 124 Zeichen werden verarbeitet, aber in der Liste verkuerzt. Kommentare und aufgeteilte Zeichenketten koennen ueber mehrere Zeilen gehen, die abschliessenden Zeichen ! bzw. ' duerfen aber nicht vergessen werden.

(3) Zeichenkettenlaengen muessen im Bereich 1-255 liegen.

(4) CONSTANTs werden intern als 16-Bit-Groessen dargestellt, jeder Operand in einem konstanten Ausdruck wird wie der Typ WORD behandelt. Das heisst, 4/2 ist gleich 2, aber 4/-2 ist gleich 0, weil -2 als eine sehr grosse positive Zahl dargestellt wird. Bei der Errechnung von konstanten Ausdruecken wird das Auftreten von Ueberlaeufen ignoriert. Wegen der Darstellung als 16-Bit-Wert werden bei einer Zeichenkette , die als Konstante benutzt wird, nur die ersten 2 Zeichen ausgewertet, d.h., 'AB' = 'ABCD'. Die Reihenfolge der Bytes bei Abspeicherung eines Wortes ist maschinenabhaengig (bei U880 erst niederwertiger Teil, dann hoeherwertiger). Programme, bei denen diese Reihenfolge eine Rolle spielt, sind nicht ohne weiteres auf anderen Maschinen lauffaehig!

(5) Der Compiler benutzt den Stack bei der Bearbeitung verschachtelter Konstruktionen. Bei komplizierter Programmschachtelung kann es zweckmaessig sein, mit Hilfe des UDOS-Kommandos SET (siehe UDOS-Handbuch) den Stack des Compilers PLZSYS zu vergroessern.
Beispiel: SET STACK_SIZE OF Dateiname TO ...

(6) Eine einzelne Prozedur darf nicht laenger als 1000 Byte (Zwischenkode) sein!

(7) Nachdem der Compiler einen Fehler bemerkt hat, sucht er eine Stelle, an der die Uebersetzung fortgesetzt werden kann. In Deklarationen (ausser Prozeduren) ist das das Auftreten der naechsten Deklarationsklassenbezeichnung (GLOBAL, INTERNAL ...), in einer Prozedur das naechste Schluesselwort, das den Beginn einer neuen Anweisung anzeigt (d.h., IF, DO, EXIT, REPEAT, RETURN, END). Die uebersprungenen Programmteile werden nicht geprueft. Daher kann es vorkommen, dass nicht alle Fehler gleich beim ersten Compilerlauf angezeigt werden. Es sind mitunter mehrere Compilerlaeufe notwendig, bis alle Fehler erkannt und schrittweise beseitigt worden sind. Da durch die Struktur der PLZ-Programme Programmierfehler haeufig zu Folgefehlern fuehren, ist es zweckmaessig, bei der Fehlerbeseitigung mit dem ersten Fehler zu beginnen.

3.1.3. Fehlermeldungen

Fehlerkategorien:

1	- 9	Warnungen
10	- 19	Fehler,die waehrend der lexikal. Analyse gefunden werden
20	- 199	Syntaktische oder semantische Fehler
200	- 255	Tabellenueberlaeufe, Compilergrenzen

Fehler	Erklaerung
1	Name nicht vom nachfolgenden Ausdruck getrennt
2	Feld ohne Elemente
3	Keine Bereiche in Datensatzdeklarationen
4	Falscher Prozedurname
5	Falscher Modulname
10	Dezimalzahl zu gross
11	Falsches Operationszeichen
12	Falsches Zeichenkettensonderzeichen
13	Falsche Hexadezimalziffer
14	Zeichenkette mit Laenge 0
15	Falsches Zeichen
16	Hexadezimalzahl zu gross
	Schleifenfehler
20	Unpassendes 'OD'
21	'OD' erwartet
22	Ungueltige REPEAT-Anweisung
23	Ungueltige EXIT-Anweisung
24	Ungueltige 'FROM'-Marke
	'IF'-Anweisungsfehler
30	Unpassendes 'FI'
31	'FI' erwartet
32	'THEN' oder 'CASE' erwartet
	Erwartete Symbole
40	')' erwartet
41	'(' erwartet
42	']' erwartet
43	'[' erwartet
44	':=' erwartet
45	'^' erwartet
	Nicht definierte Namen
50	Nicht definierter Bezeichner
51	Nicht definierter Prozedurname
	Deklarationsfehler
60	Typbezeichner erwartet
61	Ungueltige Moduldeklaration
62	Ungueltige Deklarationsklasse
63	Ungueltige Verwendung der ARRAY [* ...]-Deklaration
64	Nichtinitialisierte ARRAY [* ...]-Deklaration
65	Ungueltiger Dimensionswert

66 Ungueltiger Feldkomponententyp
67 Ungueltige Datensatzdeklaration
68 Ungueltiger Typ bei Zeigerdeklaration

Prozedur-Deklarationsfehler
70 Ungueltige Prozedurdeklaration
71 'ENTRY' erwartet
72 Prozedurname nach 'END' erwartet
73 Formaler Parametername erwartet

Initialisierungsfehler
80 Ungueltiger Initialisierungswert
81 Zu viele Initialisierungselemente fuer deklarierte Variablen
82 Ungueltige Initialisierung
83 ARRAY [* ...] wurde durch Nicht-Zeichenkette initialisiert

Spezielle Fehler
90 Ungueltige Anweisung
97 Mehrfach deklarierter Datensatz-Bereichsname
99 Mehrfach deklarierter Bezeichner

Ungueltige Variablen
100 Ungueltige Variable
101 Ungueltiger Operand fuer '#' oder 'SIZEOF'
102 Ungueltiger Bereichsname
103 Einbeziehung einer Nicht-Array-Variablen
104 Ungueltige Anwendung von '.'
105 Ungueltige Anwendung von '^'

Fehler in Ausdruecken
110 Ungueltiger arithmetischer Ausdruck
111 Ungueltiger bedingter Ausdruck
112 Ungueltiger konstanter Ausdruck
113 Ungueltiger Auswahlausdruck
114 Ungueltiger Indexausdruck
115 Ungueltiger Ausdruck bei der Zuweisung

Fehler bei Konstantengroessen
120 Konstante zu gross fuer 8 Bit
121 Konstante zu gross fuer 16 Bit
122 Konstantenfeldindex ausserhalb der Grenzen

Fehler im Prozeduraufruf
130 Ungueltige Funktion im Ausdruck
131 Ungueltiger Prozeduraufruf
132 Prozeduraufruf mit mehreren Ausgabeparametern erwartet
133 Zu wenige Ausgabeparameter
134 Zu viele Ausgabeparameter
135 Zu wenige Eingabeparameter
136 Zu viele Eingabeparameter

Typinkompatibilitaeten
140 Zeichenketteninitialisierung fuer ARRAY [*...] mit Basistyp der Komponenten ungleich 8 Bit
141 Typinkompatibilitaet bei Initialisierung

150 Typinkompatibilitaet im arithmetischen Ausdruck
151 Ungueltiger Operandentyp fuer Unaeroperator
152 Ungueltiger Operandentyp fuer Binaeroperator
153 Nicht zuweisbarer Typ
154 Ungueltiger Indextyp
156 Inkompatibilitaet des Parametertyps
157 Ungueltiger aktueller Parameter
158 Typinkompatibilitaet des Return-Parameters
159 Returnwert muss Adresse sein
160 Typinkompatibilitaet in der Zuweisung
161 Ungueltiger Operandentyp fuer Relationsoperator
162 Typinkompatibilitaet in bedingtem Ausdruck
163 Ungueltige Typumwandlung

Dateifehler
198 EOF erwartet
199 Unerwartetes EOF in Quelle gefunden

Implementierungseinschraenkungen
230 Zeichenkette oder Bezeichner zu lang
231 Ueberlauf der Compiler-Symboltabelle
232 Prozedur zu gross (max. 1000 Byte im Zwischenkode)
233 Linke Seite einer Zuweisung zu kompliziert
235 Ueberlauf des Compilerstacks
237 Statischer Datenbereichsueberlauf
238 Ueberlauf des Programmbereichs

Anmerkung: Fehlernummern > 240 koennen auftreten; wenn das Programm ansonsten richtig ist, weist das auf einen Fehler des Compilers hin. Es sollte dann versucht werden, die betreffenden Anweisungen anders zu formulieren.

3.2. Kodegenerator PLZCG

Der PLZ/SYS-Compiler erzeugt noch keinen Maschinenkode, sondern einen Zwischenkode.
Dieser Zwischenkode wird vom Programm Kodegenerator in verschieblichen Objektkode, wie ihn auch der Assembler erzeugt, umgesetzt.
Der Zweck des Zwischenkodes besteht im wesentlichen darin, dass beim Uebergang auf einen anderen Prozessor nicht ein voellig neuer Compiler, sondern nur ein neuer Kodegenerator notwendig wird, wodurch der aufwendige Analyseteil des Compilers weiterverwendet werden kann.

3.2.1. Kodegeneratoraufruf

```
%PLZCG Dateiname.Z [ND]
```

Der Dateiname muss die Endung .Z haben, wie sie vom Compiler auch erzeugt wird. Es wird eine Datei mit den verschieblichen Objektdaten erzeugt, deren Name auf .OBJ endet.

Der Bedienlauf kann mit Hilfe einer Kommandodatei verein-

facht werden, wie z.B.:

```
EDIT  #1.S      PLZSYS #1.S      PLZCG  #1.Z
```

Diese Datei mit #1-Dateiname als Parameter kann aufgerufen werden mit:

```
%D0  PLZ TEST    TEST: Name der zu erstellenden Datei
```

Der Kodegenerator benutzt als logische Ein-Ausgabe-Geraete:

2 - Mitteilung zum Bediener
4 - Zwischenkode-Eingabedatei
5 - OBJ-Kode-Ausgabedatei
6 - Notizdatei

3.2.2. Fehlermitteilungen des Kodegenerators

Es koennen Mitteilungen in zwei Formen auftreten:

(1) UDOS error Hex.zahl - UDOS-Systemfehler oder E/A-Fehler
(2) code generator error Hex.zahl - Kodegeneratorfehler

Fehlermitteilungen des Kodegenerators:

Fehler	Erklaerung
2	Ueberlauf der externen Namentabelle
3	Fehlender externer Name
4	Unzulaessiger Zwischenkode-Operator
5	Ueberlauf der Rueckwaertssprungtabelle
6	Fehlender PC-Wert bei Vorwaertssprung
7	Fehlender PC-Wert bei Rueckwaertssprung
9	Ueberlauf des 8-Bit-Stapelspeichers
10	Ueberlauf des 16-Bit-Stapelspeichers
11	Ueberlauf der Fallentscheidungstabelle (CASE)
12	Unzulaessiger Typ bei Multiplikation
13	Unzulaessiger Typ bei Division
14	Unzulaessiger Typ bei MOD-Operation
15	Nicht implementierter Zwischenkode-Operator
16	Ueberlauf der Vorwaertssprungtabelle
17	Fehlender Prozedurname
18	Ueberlauf der GLOBAL-Namenstabelle
20	Fehlerhaftes Verschiebefeld
21	Ueberlauf des Namensbereichs
25	Prozedurueberlauf

3.3 Interpreter ZINTERP

Es existiert ein verschieblicher Objektmodul mit dem Namen ZINTERP.OBJ, der mit dem Objekt- oder Zwischenkode-Moduln des Anwenders gebunden werden kann (mit PLINK) und dann eine interpretative Abarbeitung des Zwischenkodes bewirkt.

Fuer die Steuerungsuebergabe zwischen Maschinenkode und Zwischenkode wird die Marke "Zintrp" als GLOBAL benutzt, der Nutzer sollte diesen Namen deshalb nicht verwenden.
Der Vorteil des Interpreters besteht darin, dass er den Zwischenkode verarbeitet, der kuerzer ist als der U880-Maschinenkode. Ein Programm benoetigt dann weniger Programmspeicherplatz. Ein Teil der Einsparungen wird durch das mit zu bindende Interpreterprogramm wieder zunichte gemacht, so dass ein Einsatz erst ab einer bestimmten Programmgroesse (etwa 5 KByte) sinnvoll ist. Weiterhin ist zu beachten, dass interpretierte Programme etwa 10 bis 100 mal langsamer ausgefuehrt werden als uebersetzte Maschinenprogramme.

3.4. Binder PLINK

Fuer PLZ gibt es ein spezielles Bindeprogramm PLINK, das aehnlich arbeitet wie das UDOS-LINK (LINK kann fuer PLZ-Programme ebenfalls benutzt werden).

Aufruf:

```
%PLINK Dateiname (Optionen)
```

Es sind mehrere Dateinamen angebbar!

Beispiel:

```
%PLINK  u=5000  LIST.Z  ZINTERP  SIMLBIO.Z  PLZ.IO  STREAM
        (E=MAIN)
```

Anwendungsbeispiel: (s. Abschn. 5)

Unterschiede zu LINK:

- Es werden neben verschieblichen Objektdateien (.OBJ) auch Zwischenkode-Dateien direkt verarbeitet, deren Namen muessen dann auf .Z enden.
- Es gelten die gleichen Optionen wie beim UDOS-LINKer Da alle PLZ/SYS-Prozeduren die temporaeren Daten im Stack abspeichern, wird die normale Staokgroesse (80H) nicht ausreichen; mit Option ST=n kann ein groesserer Stack fuer den Programmlauf unter dem Betriebssystem UDOS zugewiesen werden.
- Die SYmbol-Option erzeugt eine Symboltabelle, die nur die GLOBALs und INTERNALs enthaelt, die in Maschinenkode-Moduln benutzt werden.
- Bei Angabe der Option 'D' in der Kommandozeile wird eine Datei '.SYM' mit Symbolinformationen fuer den Testprozess mittels PDT erzeugt.
- Die ENTRY-Option wird zur Angabe einer Startadresse fuer die Programmausfuehrung unter UDOS benutzt; das kann eine hexadezimale Adresse oder der Name einer als GLOBAL deklarierten Prozedur sein.
- Wenn ein Programm zur Ausfuehrung aufgerufen wird (der Programmname wird wie ein UDOS-Kommandoname eingegeben), dann uebergibt UDOS einen Parameter. Dies ist ein Zeiger auf das Trennzeichen nach dem Programmnamen in der UDOS-

Kommandozeile; der Zeiger kann ignoriert oder aber zur Gewinnung weiterer Eingabeinformationen fuer das Programm benutzt werden, indem die Kommandozeile ausgehend von diesem Zeiger durch das Programm abgesucht wird.
- Um die Verbindung zwischen Maschinenkode- und Zwischenkode-Moduln fuer die interpretative Abarbeitung zu ermoeglichen, wird am Ende des gebundenen Programms zusaetzlicher Speicherplatz freigehalten. Fuer jede einzelne Zwischenkode-Prozedur, auf die durch Maschinenkodemoduln zugegriffen wird, werden 7 Byte reserviert, um dem Interpreter die Uebergabe an die Zwischenkode-Prozedur zu ermoeglichen. Dieser zusaetzliche Speicherplatz ist in der .MAP-Datei unter der Ueberschrift "LINKAGE INFO" mit Adresse und Bytezahl aufgefuehrt.
- PLINK benutzt die gleichen logischen E/A-Geraete wie LINK.

4. Kommunikation mit PLZ-Programmen

In der Programmiersprache PLZ gibt es normalerweise keine E/A-Standardprozeduren oder Anweisungen, da diese Programme haeufig ohne oder mit sehr einfachen Betriebssystemen ablaufen sollen. Es ist dann zweckmaessiger, die unmittelbaren (physischen) E/A-Prozeduren in Assembler bzw. PLZ/ASM zu formulieren und die Zusammenarbeit mit dem Hauptprogramm entsprechend zu organisieren.
Die hier beschriebenen E/A-Prozeduren dienen der bequemen E/A-Moeglichkeit im Betriebssystem UDOS, d.h., sie sind anwendbar bei Programmen, die ausschliesslich unter UDOS laufen sowie fuer Test-Ein/Ausgaben in allen Programmen.

Die notwendigen Prozeduren sind in den Moduln PLZ.IO.OBJ und STREAM.OBJ enthalten. Diese beiden Moduln muessen an die Anwenderprogramme zusaetzlich mit gebunden werden. Jede der unten aufgefuehrten Prozeduren ist in PLZ.IO.OBJ als GLOBAL definiert, wobei vollstaendige Gross- oder Kleinschreibung zulaessig sind (z.B.: OPEN oder open).
Die Ein/Ausgabe beruht auf einer reihenweise Ein/Ausgabe (STREAM) fuer Informationsdateien. Eine Datei wird als Bytefolge beliebiger Laenge aufgefasst, von der ein oder mehrere Bytes von der aktuellen Position des Dateizeigers (file cursor) gelesen oder dorthin geschrieben werden koennen.
Jeder aktivierten Datei ist ein Pufferbereich zugeordnet, damit sich der Nutzer nicht darum zu kuemmern braucht, dass bestimmte physische Geraete bei der Datenuebertragung an eine feste Zeichenanzahl gebunden sind (z.B. Rekordlaenge). Dieser Speicherbereich wird vom UDOS-Speicherverwalter reserviert, wenn eine Datei eroeffnet wird, sowie wieder aufgehoben, wenn sie geschlossen wird. Die Pufferlaenge ist abhaengig von der Laenge der physischen Aufzeichnungen in einer Datei.
Die reihenweise Ein/Ausgabe (STREAM) ist geeignet fuer alle logischen E/A-Geraete wie z.B. Geraetetreiber fuer die Disketten, Lochbandgeraete, Drucker, Bildschirmgeraete; deshalb koennen die folgenden Prozeduren geraeteunabhaengig

fuer alle logischen E/A-Geraete benutzt werden.

Die Grundfunktionen der Module PLZ.IO und STREAM sind:

(1) Die automatische Datenpufferung, um einen Bytetransport von und zu Dateien mit einem sequentiellen oder wahlfreien Zugriff zu ermoeglichen.
(2) Die Parameteruebergabe zwischen PLZ-Programmen und UDOS zu organisieren.

Die Prozeduren sind allgemein gehalten, um einen Einsatz auch mit anderen Betriebssystemen zu erleichtern. Fuer den Zugriff zu systemabhaengigen Informationen kann der Nutzer eigene Prozeduren definieren.

Die wichtigsten E/A-Prozeduren sind:

```
OPEN PROCEDURE (UNIT BYTE, FILENAME ^BYTE, FLAG BYTE)
     RETURNS   (RCODE BYTE)
```

- oeffnet auf dem logischen Geraet UNIT die Datei, auf deren Namen FILENAME zeigt. Dieser Name sollte eine Folge von ASCII-Zeichen, gefolgt von einem Trennzeichen (CR, TAB, SPACE, o.a.) sein.
Wenn der Zeiger auf ein CR oder Semikolon zeigt oder der FILENAME = 0 ist, wird eine Notizdatei eroeffnet.
UDOS-Request: ASSIGN REQUEST = %02
OPEN REQUEST = %04

In FLAG wird der gewuenschte OPEN-Typ fuer die Datei angegeben (s.auch NDOS-Aufruf "OPEN" im UDOS-Systemhandbuch der P8000-Dokumentation). Dabei gilt:

FLAG

0 = Open fuer Eingabe
Existiert die Datei bereits, wird sie aktiviert; existiert sie nicht, wird ein Fehler "FILE NOT FOUND" (%C7) gemeldet.
(entspricht OPEN-Typ 0 mit gesetztem Bit 3)
1 = Open fuer Ausgabe
Existiert die Datei bereits, wird sie aktiviert und ihr bisheriger Inhalt geloescht; existiert sie nicht, wird sie erzeugt.
(entspricht OPEN-Typ 1 mit gesetztem Bit 3)
2 = Open fuer Erstellung einer neuen Datei
Existiert die Datei bereits, wird ein Fehler DUPLICATE FILE (%D0) gemeldet; existiert sie noch nicht, wird sie erzeugt.
(entspricht OPEN-Typ 2 mit gesetztem Bit 3)
3 = Open fuer "update" (auf den neuesten Stand bringen)
Existiert die Datei bereits, dann wird sie aktiviert; existiert sie noch nicht, wird sie erzeugt.
(entspricht OPEN-Typ 4 mit gesetztem Bit 3)

Andere Werte von FLAG ergeben einen Fehler "INVALID REQUEST" (%C1). Der Dateizeiger wird immer auf das

erste Byte der Datei gesetzt. Ein Fehler, der bei OPEN auftritt, wird im Fertigstellungscode RCODE zurueckgemeldet, und die Datei wird nicht aktiviert. RCODE = %4A bedeutet, dass zu wenig Speicherplatz vorhanden ist, um einen E/A - Puffer anzulegen; die Datei wurde nicht eroeffnet.

```
CLOSE PROCEDURE (UNIT BYTE)
      RETURNS   (RCODE BYTE)
```

- Schliessen des logischen Geraets UNIT.
- Loeschen des E/A-Puffers fuer das Geraet UNIT.

UDOS-Request: %06

```
GETSEQ PROCEDURE (UNIT BYTE, BUFPTR ^BYTE, NUMBYTES WORD)
       RETURNS   (RETBYTES WORD, RCODE BYTE)
```

- Liest eine Anzahl von Bytes (NUMBYTES) vom Geraet UNIT auf die Adresse BUFPTR. Die Anzahl der wirklich gelesenen Bytes wird in RETBYTES zurueckgemeldet.

UDOS-Request: %0A

```
PUTSEQ PROCEDURE (UNIT BYTE, BUFPTR ^BYTE, NUMBYTES WORD)
       RETURNS   (RETBYTES WORD, RCODE BYTE)
```

- Schreibt eine Anzahl Bytes (NUMBYTES) zum logischen Geraet UNIT vom Speicherplatz, auf den BUFPTR zeigt. Die Anzahl der geschriebenen Bytes wird in RETBYTES zurueckgemeldet.

UDOS-Request: %0E

```
SEEK PROCEDURE (UNIT BYTE, POSHIGH WORD, POSLOW WORD,
                FLAG BYTE)
     RETURNS   (RCODE BYTE)
```

- Setzt den Dateizeiger des logischen Geraetes UNIT wie folgt:

FLAG

0 : Der Dateizeiger ist der 32-bit-Wert von POSHIGH und POSLOW, d.h., ein positiver Wert von 0 bis 4.294.967.295

1 : Der Dateizeiger ergibt sich aus dem aktuellen Dateizeiger plus dem 32-Bit-Wert von POSHIGH und POSLOW (vorzeichenbehafteter Wert von -2.147.483.648 bis 2.147.483.647)

- Andere Werte von FLAG ergeben einen Fehler "INVALID REQUEST" (%C1) in RCODE.

Der Dateizeiger kann auf das dem letzten Byte der Datei folgende gesetzt werden, z.B., um Bytes mit PUTSEQ anfue-

gen zu koennen; ein GETSEQ von dieser Position ergibt einen Fehler "End of File".

UDOS-Request: %36

```
TRUNC PROCEDURE (UNIT BYTE)
      RETURNS   (RCODE BYTE)
```

- Setzt das Dateiende (End of File) auf die gerade gueltige Dateizeigerposition des logischen Geraets UNIT, alle darauffolgenden Bytes der Datei werden dabei geloescht (abgeschnitten). Wenn der Zeiger vor dem ersten Byte der Datei stand, werden alle Bytes der Datei geloescht.

UDOS-Request: %32

```
DELETE PROCEDURE (UNIT BYTE, FILENAME ^BYTE)
       RETURNS   (RCODE BYTE)
```

FILENAME <> 0 :Die Datei, auf die FILENAME zeigt und die auf dem Geraet UNIT existiert , wird aus dem Inhaltsverzeichnis des Geraetes geloescht (vorausgesetzt, der Dateiname besteht aus einer Folge von ASCII-Zeichen gefolgt von einem Trennzeichen, und ein ASSIGN Request wurde zum Geraet UNIT durchgefuehrt).

FILENAME = 0 :Die Datei , die mit dem Geraet UNIT verbunden ist, wird aus dem Inhaltsverzeichnis geloescht.
Das Geraet UNIT wird nach dem Loeschen geschlossen und die E/A-Puffer werden freigegeben (vorausgesetzt, dass das Geraet UNIT eroeffnet ist (OPEN-Request wurde durchgefuehrt) und dass kein ASSIGN-Request ausgefuehrt wurde).

UDOS-Request : %02, %06, %1A

5. Beispiel

Das folgende Beispiel soll die einzelnen Arbeitsgaenge beim Programmieren eines PLZ-Programmes zeigen.
Aufgabenstellung: Ausgabe einer UDOS-Datei von der Diskette auf den Bildschirm
Die UDOS-Datei soll DAT heissen.

(1) Erstellung der Quellprogrammdatei mit dem UDOS-Texteditor

```
%EDIT PROG.S
```

Es ist folgende Quellprogrammdatei einzugeben:

```
PROG MODULE
!!
TYPE
     PBYTE      ^BYTE
EXTERNAL
     OPEN       PROCEDURE (UNIT BYTE,FILENAME PBYTE,FLAG BYTE)
                RETURNS   (RCODE BYTE)
     CLOSE      PROCEDURE (UNIT BYTE)
                RETURNS   (RCODE BYTE)
     GETSEQ     PROCEDURE (UNIT  BYTE,BUFPTR   PBYTE,
                NUMBYTES WORD)
                RETURNS (RETBYTES WORD,RCODE BYTE)
     PUTSEQ     PROCEDURE (UNIT  BYTE,BUFPTR  PBYTE,
                NUMBYTES WORD)
                RETURNS (RETBYTES WORD,RCODE BYTE)
CONSTANT
     INPUT      := 0        !Oeffnen der Datei fuer Eingabe!
     EOF        := %C9      !Kode fuer Dateiende!
     NOTFOUND   := %C7      !Kode fuer Datei nicht gefunden!
     ERRBIT     := %40      !Maske fuer UDOS-Fehlerbit!
     CONOUT     := 2        !Geraet Bildschirmausgabe!
     INFILE     := 4        !Geraet fuer Eingabedatei!
!!
INTERNAL
     RCODE      BYTE        !RETURN-Kode!
!!
GETCH PROCEDURE RETURNS (CH BYTE)
!Eingabe 1 Byte in Puffer!
LOCAL LENGTH WORD
ENTRY
     LENGTH,RCODE := GETSEQ (INFILE, #CH, 1)
END GETCH
!!
PUTCH PROCEDURE (CH BYTE)
!Ausschrift eines Byte auf dem Bildschirm!
LOCAL LENGTH WORD
ENTRY
     LENGTH,RCODE := PUTSEQ (CONOUT, #CH, 1)
END PUTCH
!!
PUTSTR PROCEDURE (PTR PBYTE)
!Ausgabe einer Kette von Bytes, beendet durch CR!
ENTRY
     DO
          PUTCH(PTR^)
          IF PTR^  = '%R' THEN EXIT FI
          PTR      := INC PTR
     OD
END PUTSTR
!!
```

```
!!
GLOBAL
!!
LIST PROCEDURE (FILENAMEPTR PBYTE)              !Hauptprogramm!
!Textausgabe auf dem Bildschirm!
LOCAL CH BYTE
ENTRY
     RCODE := OPEN(INFILE, FILENAMEPTR, INPUT)
     IF RCODE = NOTFOUND THEN
        PUTSTR(#'FILE NOT FOUND%R')     ELSE

                  !Kopiere Bytes von Datei zum Bildschirm!
        DO
          CH := GETCH
          IF RCODE = EOF ORIF RCODE AND ERRBIT <> 0 THEN
             EXIT                       !Abbruch bei Dateiende!
          FI                            !oder Fehler!
          PUTCH(CH)
        OD
        RCODE := CLOSE(INFILE)
     FI
END LIST
!!
END PROG
```

(2) Uebersetzen der Quellprogrammdatei mit dem Compiler:

```
%PLZSYS PROG.S
```

Es entsteht die Listendatei PROG.L sowie die Zwischenkodedatei PROG.Z.

(3) Umsetzen der Zwischenkodedatei mit dem Kodegenerator:

```
%PLZCG PROG.Z
```

Es entsteht die Objektkodedatei PROG.OBJ

(4) Binden der einzelnen Programmodule:

```
%PLINK =5000 PROG.OBJ PLZ.IO STREAM (N=TEST E=LIST)
```

Dabei bedeuten:

=5000	ab Adresse 5000 wird das gelinkte Programm geladen
PROG.OBJ	Objektmodul des Programmbeispiels
PLZ.IO	Objektmodul mit Ein-/Ausgabeprozeduren
STREAM	Objektmodul fuer alle logischen E/A-Geraete
N=TEST	Name des gelinkten Programmes ist TEST
E=LIST	Entry-Option: Startadresse fuer die Programmausfuehrung ist die Prozedur LIST (muss unter GLOBAL deklariert sein).

(5) Ausgabe der Datei DAT erfolgt mit

```
%TEST DAT
```

auf den Bildschirm. Der beim Aufruf des Programms TEST durch UDOS uebergebene Parameter wird zur Uebergabe des Namens der auszugebenden Datei benutzt. Die Zeichen werden byteweise zum Bildschirm transportiert, bis der "End of File"-Rueckkehrkode auftritt.

Soll das Programm mit dem Testhilfsprogramm PDT getestet werden, dann faellt der Arbeitsgang (3) weg, und die einzelnen Programmodule werden folgendermassen gebunden:

```
%PLINK n=5000 ZINTERP PROG.Z PLZ.IO STREAM (D SY N=TEST
      E=LIST)
```

Dabei bedeuten:

ZINTERP - Zwischenkode-Interpreter; notwendig, wenn Programm mit PDT getestet werden soll
D, SY - Optionen zur Erzeugung einer .SYM-Datei

Hinweise des Lesers zu diesem Dokumentationsband

Wir sind staendig bemueht, unsere Unterlagen auf einem qualitativ hochwertigen Stand zu halten. Sollten Sie deshalb Hinweise zur Verbesserung dieses Dokumentationsbandes bzw. zur Beseitigung von Fehlern haben, so bitten wir Sie, diesen Fragebogen auszufuellen und an uns zurueckzusenden.

Titel des Dokumentationsbandes: UDOS-Software Programmiersprachen

Ihr Name / Tel.-Nr.:

Name und Anschrift des Betriebes:

Genuegt diese Dokumentation Ihren Anspruechen? ja / nein
Falls nein, warum nicht?

Was wuerde diese Dokumentation verbessern?

Sonstige Hinweise:

Fehler innerhalb dieser Dokumentation:

Unsere Anschrift: Kombinat VEB ELEKTRO-APPARATE-WERKE
BERLIN-TREPTOW "FRIEDRICH EBERT"
Abteilung Basissoftware
Hoffmannstrasse 15-26
BERLIN
1193

KOMBINAT VEB
ELEKTRO-APPARATE-WERKE
BERLIN-TREPTOW »FRIEDRICH EBERT«
Hoffmannstraße 15-26, Berlin, DDR-1193
⚡ 011 2263 eaw 011 2264 eaw

Die Angaben über technische Daten entsprechen dem bei Redaktionsschluß vorliegenden Stand. Änderungen im Sinne der technischen Weiterentwicklung behalten wir uns vor.

Ausgabe August 1986