EDV-Dokumentation

NCR

ITX COBOL 85

D00-0076120

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

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

ANHANG:

KAPITEL 1: COBOL EINFÜHRUNG

COBOL (Common Business Oriented Language) ist eine dem Englischen nahe Programmiersprache, die hauptsächlich im kaufmännischen Bereich verwendet wird. Als eine dem Englischen nahe Sprache wird es deshalb bezeichnet, weil es seiner freien Form wegen einen Programmierer in die Lage versetzt, in einer Art zu schreiben, die das Endresultat leicht lesbar macht. Der allgemeine Fluß der Logik kann auch von Personen verstanden werden, die nicht wie der Programmierer selbst mit den Details des Problems eng vertraut sind.

Da COBOL eine Programmiersprache ist, muß es übersetzt werden, um als Kommunikation zwischen dem Programmierer und dem Computer zu dienen. Das vom Programmierer geschriebene COBOL-Programm (das Ursprungsprogramm oder Source-Programm) ist Eingabe für ein COBOL Übersetzerprogramm (Compiler-Programm). Das COBOL Compiler-Programm übersetzt dann das SOURCE-Programm in ein Maschinensprache-Programm (das Objekt-Programm). Da die Maschinensprache von Computer zu Computer verschieden ist, hat jeder Computer sein eigenes COBOL Compiler-Programm, um die Übersetzung aus der COBOL-Sprache in das jeweilige Objekt-Programm durchzuführen.

Obwohl jeder Computer über sein eigenes spezielles COBOL Compiler-Programm verfügt, hat ein intensives Bemühen der computerherstellenden Industrie hinsichtlich COBOL zu einem hohen Grad von Kompatibilität geführt, so daß ein COBOL Source-Programm zwischen verschiedenartigen Computern eines Herstellers oder zwischen Computern verschiedener Hersteller ausgetauscht werden kann.

Ein COBOL-Programm ist sowohl ein lesbares Dokument als auch ein leistungsfähiges Computerprogramm. Während des ganzen Erlernens der COBOL-Sprache ist es wichtig, sich dieser beiden Grundfähigkeiten von COBOL bewußt zu sein und deren enge Verbindung zu beachten.

Der Lesbarkeitsfaktor der COBOL-Sprache erleichtert Kommunikation nicht nur zwischen Programmierer und Management, sondern auch zwischen Programmierern, und das mit einem Minimum an zusätzlicher Dokumentation. Der Programmierer kann eine Lösung in Form eines COBOL-Programms durch folgende Leistungen seinerseits erbringen: Kenntnis des Problems, Programmiertechnik, richtige Ausrüstung und Vertrautheit mit den verfügbaren Elementen der COBOL-Sprache.

COBOL ist eine Industriesprache und nicht Eigentum eines Unternehmens, einer Unternehmensgruppe, einer Organisation oder einer Gruppe von Organisationen.

ITX COBOL 85 ist von LPI-COBOL abgeleitet.

KAPITEL 2: DER AUFBAU DER COBOL-SPRACHE

EINLEITUNG

Das kleinste Element der COBOL-Sprache ist das Zeichen. Ein Zeichen kann eine Ziffer, ein Buchstabe des Alphabets, ein Satzzeichen oder ein Sonderzeichen sein. Die Zeichen werden zur Bildung eines COBOL-Wortes benutzt. COBOL-Wörter werden nach speziellen Regeln gebildet.

Die COBOL-Wörter werden zu Anweisungen, Sätzen, Paragraphen und Kapiteln zusammengestellt. Durch falsche Schreibweise oder Satzbildung entstehen Fehler. Es ist daher eine sehr präzise Kenntnis der COBOL-Schreibweise notwendig, um ein arbeitsfähiges Programm schreiben zu können.

ZEICHEN

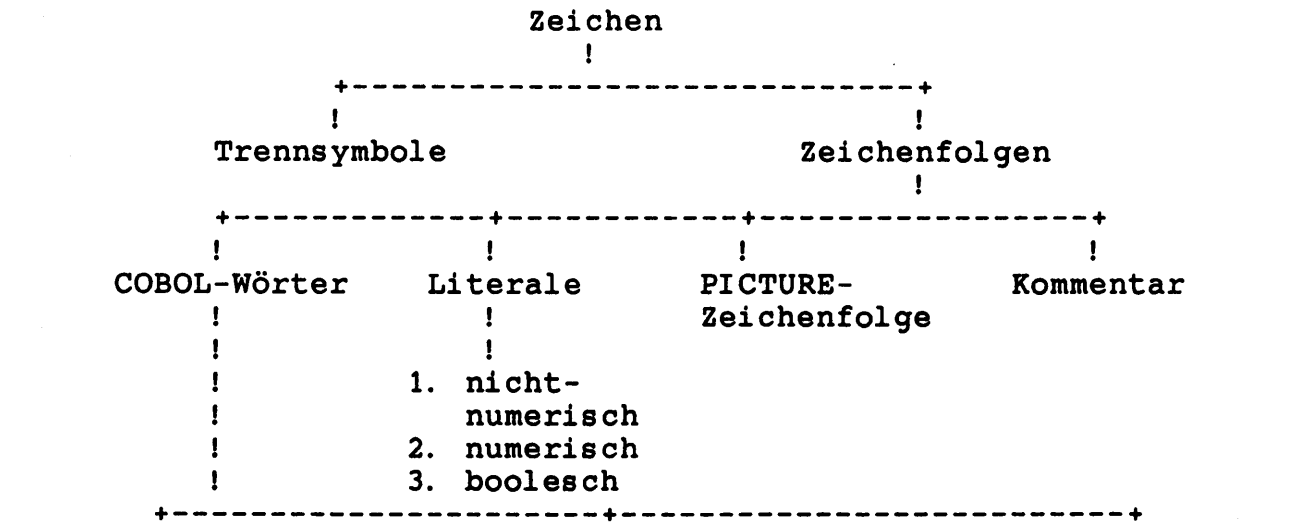
Einundfünfzig Zeichen machen den COBOL-Zeichenvorrat aus:

0 bis 9	Ziffern
A bis Z	Buchstaben
␣	Leerzeichen (Zwischenraum, Space, Blank)
+	Pluszeichen
-	Minuszeichen (oder Bindestrich)
*	Stern
/	Schrägstrich
=	Gleichheitszeichen
$	Währungszeichen
,	Komma (oder Dezimalpunkt)
;	Semikolon
"	Anführungszeichen
(	runde Klammer auf
)	runde Klammer zu
>	Größerzeichen
<	Kleinerzeichen
'	Apostroph

Während die 51 vorgenannten Zeichen den ANSI Norm COBOL-Zeichenvorrat ausmachen, kann der Programmierer auch jedes der im ASCII Zeichenvorrat graphisch dargestellten Zeichen verwenden. (Siehe den ASCII Zeichenvorrat in der nachfolgenden Code-Tabelle). Zum Beispiel ist es beim Schreiben von nicht-numerischen Literalen, Kommentaren oder Kommentarzeilen möglich, diese zusätzlichen Zeichen zu verwenden.

Right Hex / Left Hex		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
0001	1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
0010	2	⌀	!	"	#	$	%	&	'	(	)	*	+	,	—	.	/
0011	3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0100	4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0101	5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
0110	6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0111	7	p	q	r	s	t	u	v	w	x	y	z	{	¦	}	~	DEL

Zeichen des COBOL-Zeichenvorrates werden miteinander verbunden und bilden so entweder ein Trennsymbol oder eine Zeichenfolge. Das Diagramm auf der nächsten Seite stellt die Verbindung von Zeichen, Trennsymbolen und all den verschiedenen Arten von Zeichenfolgen dar.


Zeichen
Trennsymbole
Zeichenfolgen
COBOL-Wörter
Literale
PICTURE-Zeichenfolge
Kommentar
1. nicht-numerisch
2. numerisch
3. boolesch
Programmiererwörter
System-Namen
reservierte Wörter
1. Programm-Name
2. Datei-Name
3. Datensatz-Name
4. Daten-Name bzw. Identifier
5. Prozedur-Name
a) Kapitel-Name (Section)
b) Paragraphen-Name
6. Bedingungs-Name
7. Index-Name
8. Merk-Name (mnemonic)
9. Alphabet-Name
10. Text-Name
11. Stufen-Nummer
12. Segment-Nummer
13. CD-Name
14. Klassen-Name (Class)
15. Library-Name
16. Routinen-Name
17. Symbolic Character
1. Computer-Name
2. Implementor-Name
3. Language-Name
1. Schlüsselwörter
2. Wahlwörter
3. Verbindungen
4. Spezialzähler
5. figurative Konstanten
6. Spezialzeichen-wörter

TRENNSYMBOL (SEPARATOR)

Ein Trennsymbol oder Separator ist eine Folge von einem oder zwei Zeichen, bestehend aus den Satzzeichen, dem Buchstaben B oder dem Buchstaben H. Ein Satzzeichen ist ein Zeichen, das der nachfolgend aufgeführten Reihe von COBOL-Zeichen, die der Satzzeichenvorrat genannt wird, zugehört.

, Komma

; Semikolon

" Anführungszeichen

(runde Klammer auf

) runde Klammer zu

␣ Leerzeichen (Zwischenraumzeichen)

= Gleichheitszeichen

Ein Separator ist in einer Folge mit einem anderen Separator oder mit einer Zeichenfolge verbunden. Die Folge der Zeichen und Trennsymbole bildet den Text eines COBOL Ursprungs-Programms. Unter der Verwendung der als Satzzeichenvorrat definierten Zeichen, des Buchstabens B und des Buchstabens H erfolgt die Bildung der COBOL-Trennsymbole nach den folgenden Regeln:

1. Das Satzzeichen Zwischenraum ist ein Separator. Wo immer ein Zwischenraum als Separator verwendet wird, kann mehr als ein Leerzeichen dafür benutzt werden. Zwischenräume dürfen auch aus mehreren Zwischenraumzeichen bestehen.

2. Die Satzzeichen Komma, Semikolon und Punkt sind Trennsymbole, wenn sie unmittelbar vor einem Leerzeichen stehen. Diese Trennsymbole können nur in einem COBOL Source-Programm erscheinen, wo sie ausdrücklich durch den allgemeinen Aufbau der COBOL-Programmiersprache zugelassen sind.

3. Die Satzzeichen Komma und Semikolon sind immer wahlfrei! Sie sind äquivalent zum Leerzeichen und können jederzeit durch es ersetzt werden.

4. Die Satzzeichen runde Klammer auf und runde Klammer zu sind Trennsymbole, die lediglich paarweise in Erscheinung treten können. Runde Klammer auf und runde Klammer zu werden verwendet, um Subskripte, Indizes, arithmetische Ausdrücke oder Bedingungen zu begrenzen.

5. Die Satzzeichen Anführungszeichen und Apostroph sind Separatoren, die paarweise auftreten. Dabei muß dem linken Anführungszeichen unmittelbar ein Zwischenraum oder eine geöffnete runde Klammer vorausgehen und das rechte Anführungszeichen unmittelbar gefolgt sein von einem der Trennsymbole, wie Leerzeichen, Komma, Semikolon, Punkt oder runde Klammer geschlossen.

6. Zwei aufeinanderfolgende Gleichheitszeichen (==) bilden ein Pseudotext-Begrenzungssymbol. Das Symbol muß paarweise auftreten. Dabei muß dem linken Symbol ein Leerzeichen vorausgehen und dem rechten unmittelbar einer der Separatoren Leerzeichen, Komma, Semikolon oder Punkt folgen.

7. Der Separator Zwischenraum kann wahlweise jedem Separator außer dem Trennsymbol Anführungszeichen rechts unmittelbar vorausgehen. Im Falle des Trennsymbols Anführungszeichen rechts nämlich ist ein vorausgehender Zwischenraum als Teil des nicht-numerischen Literals anzusehen und gilt nicht als Separator.

8. Der Separator Zwischenraum kann wahlweise jedem Separator außer dem Trennsymbol Anführungszeichen links unmittelbar folgen. Im Falle des Trennsymbols Anführungszeichen links ist ein nachfolgender Zwischenraum als Teil des nicht-numerischen Literals anzusehen und gilt nicht als Separator.

Die Regeln, wie sie vorstehend für die Bildung von Trennsymbolen gegeben sind, gelten nicht für die Satzzeichen, wie sie in einem Literal, einer PICTURE-Zeichenfolge, einem Kommentar oder einer Kommentarzeile vorkommen.

ZEICHENFOLGE

Eine Zeichenfolge ist ein Zeichen oder eine Folge von zwei oder mehreren zusammenhängenden Zeichen, die ein COBOL-Wort, ein Literal, eine PICTURE-Zeichenfolge oder einen Kommentar bilden. Eine Zeichenfolge kann in einer Kette lediglich mit einem Trennsymbol verbunden werden. Eine Zeichenfolge ist durch Trennsymbole (Separatoren) begrenzt.

COBOL-WORT

Ein COBOL-Wort ist eine Zeichenfolge von nicht mehr als 30 Zeichen, die entweder ein Programmiererwort, einen System-Namen oder ein reserviertes Wort bildet. Ein COBOL-Wort kann nur einem der drei genannten Typen zugehören, d. h. wenn ein COBOL-Wort ein reserviertes Wort ist, dann kann es weder ein Programmiererwort noch ein System-Name sein.

PROGRAMMIERERWORTE

Ein Programmiererwort ist ein COBOL-Wort, das der Programmierer bildet, um dabei dem Aufbau einer Klausel oder einer Anweisung zu genügen. Ein Programmiererwort ist durch Trennsymbole begrenzt.

Jedes Zeichen eines Programmiererwortes muß aus dem nachfolgend aufgeführten Vorrat von COBOL-Zeichen, genannt der Wortvorrat, ausgewählt werden:

Alphabetische Zeichen A bis Z
Numerische Zeichen 0 bis 9
Bindestrich (-)

Der Bindestrich kann nicht das erste oder letzte Zeichen eines Programmiererwortes sein. Der Bindestrich kann jedoch an jeder anderen Stelle innerhalb des Programmiererwortes verwendet werden.

In der COBOL-Sprache gibt es achtzehn verschiedene Arten von Programmiererwörtern, nämlich Programm-Name, Datei-Name, Datensatz-Name, Daten-Name, Paragraphen-Name, Kapitel-Name, Bedingungs-Name, Index-Name, Merk-Name, Alphabet-Name, Text-Name, Stufen-Nummer, Segment-Nummer, CD-Name, Klassen-Name, Library-Name, Routinen-Name, Symbolic Character.

Programm-Name

Ein Programm-Name ist ein Programmiererwort, das ein COBOL-Ursprungsprogramm, das Objekt-Programm und alle zu einem bestimmten Programm gehörenden Ausdrucke bezeichnet. Ein Programm-Name muß 1 bis 30 Zeichen enthalten. Ein Programm-Name muß zumindest ein alphabetisches Zeichen in einer beliebigen Position enthalten. Jeder Programm-Name muß eindeutig sein.

Datei-Name

Ein Datei-Name ist ein Programmiererwort, das eine Datei bezeichnet, auf die durch das Programm zugegriffen wird. Der Datei-Name muß mindestens ein alphabetisches Zeichen in einer beliebigen Position enthalten. Datei-Namen werden vom Programmierer bestimmt und verwendet. Jeder Datei-Name im Programm muß eindeutig sein.

Datensatz-Name

Ein Datensatz-Name ist ein Programmiererwort, das einen im Programm verwendeten Datensatz bezeichnet. Ein Datensatz-Name muß in einer beliebigen Position zumindest ein alphabetisches Zeichen enthalten. Datensatz-Namen werden vom Programmierer bestimmt und verwendet. Wenn der gleiche Datensatz-Name mehr als einem Datensatz zugeteilt ist, dann muß der Datensatz-Name, wenn darauf in einer Prozeduranweisung Bezug genommen wird, durch Kennzeichnung mit einem Datei-Namen eindeutig gemacht werden.

Daten-Name (Data-Name, Identifier)

Eine Gruppierung zusammenhängender Zeichen, die als eine Dateneinheit behandelt wird, wird ein Datenfeld genannt. Das Programmiererwort, das ein Datenfeld bezeichnet, wird Daten-Name genannt.

Ein Daten-Name muß in irgendeiner Zeichenposition zumindest ein alphabetisches Zeichen enthalten. Daten-Namen werden vom Programmierer bestimmt und verwendet. Wenn der gleiche Daten-Name mehr als einem Datenfeld zugeteilt ist, dann muß der Daten-Name durch Qualifizierung, Indizierung oder Subskribierung eindeutig gemacht werden, wenn darauf in einer Prozeduranweisung Bezug genommen wird. Im allgemeinen COBOL-Format steht der Ausdruck Identifier für einen Daten-Namen, dem wahlweise eine syntaktisch korrekte Kombination von Qualifikatoren, Indizes und Subskripten folgt.

Der Programmierer hat jedes Datenfeld in einem Programm zu benennen und zu definieren, so daß auf dieses während der Durchführung des Programms zugegriffen werden kann. So würde beispielsweise, wenn ein Programm zur Bearbeitung von Gewinnkonten geschrieben wird, eines der Datenfelder in dem Datensatz wahrscheinlich die Kontonummer darstellen. Der Programmierer teilt diesem Datenfeld einen Namen wie z. B. KONTONUMMER zu und verwendet den Identifier KONTONUMMER jedesmal, wenn auf das Feld Bezug genommen wird.

Prozedur-Name

Ein Programm besteht aus einer oder mehreren Prozeduren, in der eine bestimmte Operation oder eine Reihe von Operationen mit Daten durchgeführt wird. So kann beispielsweise durch eine Prozedur ein Konto fortgeschrieben werden. Diese Prozedur könnte daher den Prozedur-Namen KONTOFORTSCHREIBUNG erhalten. Jedesmal, wenn der Programmierer dann die Fortschreibung eines Kontos wünscht, hat er lediglich auf den Prozedur-Namen KONTOFORTSCHREIBUNG Bezug zu nehmen.

Prozedur-Namen werden vom Programmierer bestimmt und verwendet, wenn dies zur Markierung einer Prozedur erforderlich ist. Ein Prozedur-Name enthält eine beliebige Kombination von Zeichen aus dem COBOL-Wortvorrat, wobei die einzige Ausnahme ist, daß der Bindestrich nicht das erste oder letzte Zeichen eines Prozedur-Namens sein kann.

Es gibt zwei Arten von Prozedur-Namen: Paragraphen-Namen und Kapitel-Namen. Ein Paragraph besteht aus null, einem oder mehreren Programmsätzen, die eine gemeinsame Funktion haben, wie beispielsweise die Durchführung einer Berechnung. Der Paragraphen-Name bezeichnet den Beginn des Paragraphen. Wenn der gleiche Paragraphen-Name mehrmals verwendet wird, dann muß der Paragraphen-Name, wenn darauf in einer Prozeduranweisung Bezug genommen wird, durch Kennzeichnung mit einem Kapitel-Namen eindeutig gemacht werden.

Ein Kapitel besteht aus null, einem oder mehreren aufeinander bezogenen Paragraphen. So können beispielsweise 12 zusammenhängende Paragraphen, von denen jeder Prozeduren zur Verarbeitung einer bestimmten Bedingung enthält, in einem Kapitel gruppiert werden. Der Kapitel-Name bezeichnet den Beginn eines Kapitels. Jeder Kapitel-Name in einem Programm muß eindeutig sein.

Bedingungs-Name

Ein Bedingungs-Name ist ein Programmiererwort, das einen Wert eines Datenfeldes oder den Status eines Option-Schalters bezeichnet. Wenn der gleiche Bedingungs-Name mehr als einem Datenfeld zugeordnet ist, dann muß der Bedingungs-Name durch Qualifizierung eindeutig gemacht werden, wenn darauf in einer Prozeduranweisung Bezug genommen wird. Jeder einem Option-Schalter zugeordnete Bedingungs-Name in einem Programm muß in diesem einmalig sein.

Der Programmierer kann einem Datenfeld, dessen Inhalt getestet werden soll, einen oder mehrere Bedingungs-Namen zuordnen. Die Datenbeschreibung des Bedingungs-Namens bestimmt den zu testenden Wert, Wertsatz oder Wertbereich. Das VERKAUFSORT genannte Datenfeld kann zum Beispiel drei Bedingungs-Namen haben: ZENTRALE ist dem Wert 1 zugeordnet; FILIALEN ist dem Wertbereich 2 bis 9 zugeordnet; SB-MARKT ist dem 3, 6, 7 und 8 enthaltenden Wertsatz zugeordnet.

Ein Datenfeld mit einem oder mehreren Bedingungs-Namen wird eine Bedingungsvariable genannt. Folglich ist im vorhergehenden Absatz VERKAUFSORT die Bedingungsvariable mit den drei Bedingungs-Namen ZENTRALE, FILIALE und SB-MARKT.

Index-Name

Ein Index-Name ist ein Programmiererwort, das einen mit einer bestimmten Tabelle verbundenen Index bezeichnet. Ein Index-Name muß an irgendeiner Stelle zumindest ein alphabetisches Zeichen enthalten. Der Programmierer bestimmt und verwendet Index-Namen, wie dies zur Bearbeitung von Tabellen erforderlich ist. Jeder Index-Name in einem Programm muß eindeutig sein.

MERK-NAME (oder mnemonischer Name)

Ein Merk-Name ist ein Programmiererwort, das ein besonderes Gerät (Console) oder einen Schalter bezeichnet. Jeder Merk-Name in einem Programm muß eindeutig sein. Merk-Namen sind im SPECIAL-NAMES-Paragraphen zu definieren.

Alphabet-Name

Ein Alphabet-Name ist ein Programmiererwort, das einen bestimmten Zeichenvorrat und/oder eine bestimmte Sortierfolge bezeichnet. Ein Alphabet-Name muß in irgendeiner seiner Positionen zumindest ein alphabetisches Zeichen enthalten. Jeder Alphabet-Name in einem Programm muß eindeutig sein.

Text-Name

Ein Text-Name ist ein Programmiererwort, das eine Folge von Zeichen und/oder Trennsymbolen in einer COBOL-Bibliothek bezeichnet. Solch eine Folge von Zeichen und/oder Trennsymbolen wird Bibliothekstext genannt. Ein Text-Name enthält 1 bis 30 Zeichen. Ein Text-Name muß in irgendeiner seiner Positionen zumindest ein alphabetisches Zeichen enthalten. Die ersten zehn Zeichen eines jeden Text-Namens müssen eindeutig sein.

Stufen-Nummer

Eine Stufen-Nummer ist ein Programmiererwort, das die Position eines Datenfeldes in der hierarchischen Struktur eines Datensatzes oder die speziellen Merkmale einer Datenerklärung anzeigt. Eine Stufen-Nummer besteht aus ein oder zwei Ziffern.

Segment-Nummer

Eine Segment-Nummer ist ein Programmiererwort, das zu Zwecken der Überlagerung (Segmentation) ein Kapitel (SECTION) im Prozedurteil (PROCEDURE DIVISION) klassifiziert. Eine Segment-Nummer besteht entweder aus einer oder zwei Ziffern.

SYSTEM-NAME

Ein System-Name ist ein COBOL-Wert, das die verwendete Hardware beschreibt. Die drei Arten von System-Namen sind Computer-Name, Implementor-Name und Language-Name. Im COBOL wird der Computer-Name in dem SOURCE-COMPUTER-Paragraphen und in dem OBJECT-COMPUTER-Paragraphen verwendet.

Die Hersteller-Namen CONSOLE, SWITCH-n und LINE-n sind im SPECIAL-NAMES-Paragraphen vorgesehen.

RESERVIERTE WÖRTER

Die Struktur von COBOL bestimmt die Verwendung von bestimmten COBOL-Wörtern, die als reservierte Wörter bezeichnet werden. Ein reserviertes Wort ist durch Trennsymbole begrenzt. Ein Programmierer kann ein reserviertes Wort für ein COBOL-Programm nicht bestimmen. Er muß vielmehr das durch das Format bestimmte Wort verwenden. In einem Programm darf ein reserviertes Wort weder als Programmiererwort noch als System-Name in Erscheinung treten. Eine Liste aller reservierten Wörter ist im Anhang enthalten.

In der COBOL-Sprache gibt es sechs verschiedene Arten von reservierten Wörtern, nämlich Schlüsselwörter, Wahlwörter, Verbindungen, Spezialzähler, figurative Konstanten und Spezialzeichen-Wörter.

Schlüsselwörter

Ein Schlüsselwort informiert den Compiler über die Durchführung eines bestimmten Vorganges. Der Programmierer muß alle Schlüsselwörter, die in einem Format auftreten, schreiben. In den allgemeinen Formaten sind alle Schlüsselwörter in Großbuchstaben geschrieben und unterstrichen. Im nachfolgenden allgemeinen Format sind die Schlüsselwörter beispielsweise READ, INTO und END.

READ Datei-Name RECORD [INTO Identifier] [; AT END]
unbedingte Anweisung

Es gibt drei Arten von Schlüsselwörtern:

- Verben, die einen Vorgang bezeichnen und am Anfang einer COBOL-Anweisung stehen, wie beispielsweise ADD, MOVE, READ und PERFORM.
- Notwendige Wörter, die in Anweisungs- und Beschreibungs-Formaten auftreten, beispielsweise die Wörter INTO und END, wie sie im obigen Format enthalten sind.
- Wörter, die eine spezielle funktionelle Bedeutung haben, wie beispielsweise SECTION und NEGATIVE.

Wahlwörter

Ein Wahlwort kann zur besseren Lesbarkeit in COBOL-Sätzen eingesetzt werden. Das Vorhandensein oder das Fehlen eines solchen Wortes in einem Format beeinträchtigt die Umwandlung des Compilers nicht. In den allgemeinen Formaten sind alle Wahlwörter in Großbuchstaben geschrieben und nicht unterstrichen. In dem allgemeinen Format der obigen READ-Anweisung sind die Wahlwörter RECORD und AT.

Verbindungen

COBOL-Verbindungen sind in drei Arten aufgeteilt, nämlich in qualifizierende, Reihen- und logische Verbindungen.

Die qualifizierenden Verbindungen IN und OF verbinden einen Namen mit einem anderen Namen, um dem ganzen Namen eine Eindeutigkeit zu geben. Der Programmierer kann die Wörter IN und OF austauschbar verwenden. Ein Beispiel dafür ist:

```
ARTNR OF ARTIKEL-SATZ
ARTNR IN ARTIKEL-SATZ
ARTNR OF BEWEG-SATZ
ARTNR IN BEWEG-SATZ
```

Die Reihenverbindungen, das Komma und das Semikolon, verbinden zwei oder mehrere aufeinanderfolgende Operanden, z. B.

```
ADD SUMME-1  SUMME-2  SUMME-3  GIVING TOTAL.
```

Die logischen Verbindungen AND und OR verbinden eine Bedingung mit einer anderen, z. B.

```
IF ZAHLUNG = 10 AND LAST = 35 GO TO BERECHNUNG.
IF NAME1 IS GREATER THAN NAME2 OR NEU-NAME IS GREATER THAN
   NAME3 GO TO ABC.
```

Spezialzähler

Ein Spezialzähler ist ein reserviertes Wort, das einen vom Compiler gebildeten Speicherbereich bezeichnet. Zum Beispiel bezeichnet der Zähler LINAGE-COUNTER den Zeilenzähler, der beim Drucken verwendet werden kann (LINAGE-Klausel).

Figurative Konstante

Werte wie Null und Leerzeichen werden in Programmen sehr häufig verwendet. Daher läßt die COBOL-Sprache ein reserviertes Wort, wie ZERO oder SPACE, immer dann zur Verwendung zu, wenn der Wert des damit verbundenen Zeichens gebraucht wird. Diese reservierten Wörter, die konstanten Werten zugeteilt sind, nennt man figurative Konstanten. Figurative konstante Werte werden vom Compiler gebildet, wenn sie im Ursprungs-Programm angegeben werden. Eine figurative Konstante darf nicht in Anführungszeichen stehen.

Die den Werten von üblicherweise verwendeten Zeichen zugeteilten figurativen Konstanten umfassen ZERO, ZEROS, ZEROES, SPACE, SPACES, HIGH-VALUE, HIGH-VALUES, LOW-VALUE, LOW-VALUES, QUOTE, QUOTES und ALL Literal.

Die figurative Konstante ZERO, ZEROS oder ZEROES stellt den Wert 0 oder ein oder mehrere 0-Zeichen dar. Beispielsweise verursacht MOVE ZEROS TO SALDO, daß der ganze Inhalt des Datenfeldes SALDO mit Nullen aufgefüllt wird.

Die figurative Konstante SPACE oder SPACES stellt ein oder mehrere Leerzeichen (Leerstellen) dar. Zum Beispiel verursacht MOVE SPACES TO ZEILE, daß der gesamte Inhalt des Datenfeldes ZEILE mit Leerzeichen aufgefüllt wird. Diese figurative Konstante wird bei einem Datenfeld verwendet, das alphabetische oder alphanumerische Daten enthält.

Die figurative Konstante HIGH-VALUE oder HIGH-VALUES stellt ein oder mehrere alphanumerische Zeichen mit dem höchsten Wert dar. In der ASCII-Zeichenfolge stellt HIGH-VALUE oder HIGH-VALUES ein oder mehrere Zeichen mit der Bitstruktur 01111111 (Hex 7F) dar. Wenn beispielsweise ein Zeichen ein Speicherbyte umfaßt, dann wird HIGH-VALUE oder HIGH-VALUES durch die Bitstruktur 01111111 dargestellt. Diese figurative Konstante wird bei einem Datenfeld verwendet, das alphanumerische Daten enthält.

Die figurative Konstante LOW-VALUE oder LOW-VALUES stellt ein oder mehrere alphanumerische Zeichen mit dem niedrigsten Wert dar. In der ASCII-Zeichenfolge stellt LOW-VALUE oder LOW-VALUES ein oder mehrere Zeichen mit der Bitstruktur 00000000 (Hex 00) dar. Wenn beispielsweise ein Zeichen ein Speicherbyte umfaßt, dann wird LOW-VALUE oder LOW-VALUES durch die Bit-Struktur 00000000 dargestellt. Diese figurative Konstante wird bei einem Datenfeld verwendet, das alphanumerische Daten enthält.

Die figurative Konstante QUOTE oder QUOTES stellt ein oder mehrere Anführungszeichen (") dar. Das Wort Quote oder Quotes kann ein Anführungszeichen im Ursprungs-Programm zur Aufnahme eines nicht-numerischen Literals nicht ersetzen. QUOTE oder QUOTES wird an den Stellen verwendet, wo das Auftreten eines Anführungszeichens fälschlicherweise den Einschluß eines nicht-numerischen Literals anzeigen würde. Zum Beispiel bringt MOVE QUOTES TO FELD-A Anführungszeichen nach FELD-A. Diese figurative Konstante wird bei einem Datenfeld verwendet, das alphanumerische Daten enthält.

Die figurative Konstante ALL Literal bewirkt eine oder mehrere Wiederholungen des Literals, das ein nicht-numerisches Literal oder eine figurative Konstante sein kann.

Wenn eine figurative Konstante angegeben wird, ist das Wort ALL nicht notwendig, sondern dient nur zur besseren Lesbarkeit.

Nach der Anweisung MOVE ALL "Z" TO FELD-1
enthält das 4-Byte große FELD-1 den Wert ZZZZ.

Nach der Anweisung MOVE ALL "AB" to FELD-2
enthält das 6-Byte große FELD-2 den Wert ABABAB.

Die Anweisung MOVE ALL SPACES TO DRUCKSATZ ist gleich der Anweisung MOVE SPACES TO DRUCKSATZ.

Eine figurative Konstante kann verwendet werden, wo immer ein Literal in der COBOL-Sprache auftritt. Die einzige Einschränkung besteht darin, daß lediglich die figurative Konstante ZERO, ZEROS oder ZEROES zugelassen ist, wenn das Format ein numerisches Literal notwendig macht.

Die Anzahl von Zeichen, die durch eine figurative Konstante dargestellt werden, hängt von der Größe des zugehörigen Datenfeldes ab.

Die figurative Konstante (ALL) Symbolic-Character bewirkt eine oder mehrere Wiederholungen des Zeichens, das als Wert dieses Symbolic-Characters in der SYMBOLIC CHARACTERS Klausel des SPECIAL-NAMES-Paragraphen definiert ist.

Spezialzeichen-Wörter

Ein Spezialzeichen ist ein reserviertes Wort, das ein Vergleichszeichen darstellt. Die Spezialzeichen-Wörter sind folgende:

>	Größerzeichen
<	Kleinerzeichen
=	Gleichheitszeichen

LITERALE

Ein Literal ist eine Zeichenfolge, deren Wert sich bei der Durchführung des Programms nicht ändert. Der Wert eines Literals ist genau der angegebene. Zum Beispiel: ADD 15 TO TOTAL zeigt an, daß der exakte Wert 15 zum Inhalt eines TOTAL genannten Datenfeldes addiert wird. Während der Inhalt des Datenfeldes TOTAL variiert, trifft dies für den Wert des Literals 15 nicht zu.

Ein Literal wird entweder durch einen geordneten Satz von Zeichen, wie 15, oder durch ein reserviertes Wort dargestellt, das auf eine figurative Konstante, wie ZERO, Bezug nimmt. Literale werden vom Programmierer bestimmt und verwendet, wie dies in einem Programm erforderlich ist. Es gibt vier Arten von Literalen: nicht-numerische Literale, numerische Literale, hexadezimale Literale und boolesche Literale.

Nicht-numerische Literale

Ein nicht-numerisches Literal enthält jedes beliebige Zeichen aus dem ASCII-Zeichensatz. Ein nicht-numerisches Literal besteht aus einer in Anführungszeichen (") stehenden Folge aus bis zu 254 Zeichen. Zur Darstellung eines einzigen Anführungszeichens im nicht-numerischen Literal müssen in diesem zwei aufeinanderfolgende Anführungszeichen geschrieben werden.

Alle Zeichen innerhalb der Anführungszeichen werden im Objekt-Programm als ein alphanumerisches Datenfeld mit konstantem Wert gespeichert. Die einzige Ausnahme ist das Auftreten zweier aufeinanderfolgender Anführungszeichen, die als nur ein Anführungszeichen im Objekt-Programm gespeichert werden.

Es folgen Beispiele nicht-numerischer Literale, wovon das letzte Beispiel eine Konstante mit dem Wert JÄHRLICHE "ABC" LISTE bringt.

```
"FEHLER-LISTE"
"65%"
"FORMULAR VOR MONATSANFANG AN VERSANDABT. GEBEN"
"ENDE PROGRAMM XYZ"
"JÄHRLICHE ""ABC"" LISTE"
```

Die figurativen Konstanten SPACE, SPACES, HIGH-VALUE, HIGH-VALUES, LOW-VALUE, LOW-VALUES, QUOTE und QUOTES gehören zur nicht-numerischen Kategorie der Literale.

Numerische Literale

Ein numerisches Literal ist eine Zeichenfolge, die nur die Ziffern 0 bis 9, das Pluszeichen (+), das Minuszeichen (-) und den Dezimal-Separator enthält. Ein numerisches Literal wird nicht von Anführungszeichen, sondern vielmehr von Trennsymbolen eingeschlossen. Alle die Zeichen im numerischen Literal außer dem Dezimal-Separator befinden sich als ein numerisches Datenfeld mit einem konstanten Wert im Objekt-Programm. Numerische Literale werden nach folgenden Regeln gebildet:

1. Ein numerisches Literal muß zumindest eine und darf nicht mehr als 18 Ziffern enthalten.

2. Ein numerisches Literal darf nicht mehr als ein Vorzeichen enthalten. Wenn ein Vorzeichen verwendet wird, muß es als das äußerste linke Zeichen des Literals in Erscheinung treten. Ein Literal ohne Vorzeichen ist als positiv zu betrachten.

3. Ein numerisches Literal darf nicht mehr als einen Dezimal-Separator enthalten. Dieser wird als ein Separator behandelt und kann in jeder beliebigen Position innerhalb des Literals außer an der äußersten rechten Zeichenposition auftreten. Ein numerisches Literal ohne Dezimal-Separator ist als eine Ganzzahl zu betrachten.

Jedes als Teil eines numerischen Literals auftretende Satzzeichen gilt nicht als ein Satzzeichen, sondern vielmehr als ein Symbol in der Spezifikation des numerischen Literals.

Wenn ein Literal den Regeln zur Bildung numerischer Literale entspricht und in Anführungszeichen steht, dann gilt dieses Literal als ein nicht-numerisches Literal und wird vom COBOL-Compiler als solches behandelt.

Nachfolgend sind Beispiele numerischer Literale aufgeführt:

-680
+97.25
102
-.02

Die figurativen Konstanten ZERO, ZEROS und ZEROES gehören zur numerischen Kategorie der Literale.

Hexadezimale Literale

Ein hexadezimales Literal ist eine Zeichenfolge, die nur die Zeichen von 0 bis 9 und die Buchstaben A bis F enthält. Ein hexadezimales Literal besteht aus einer Folge von bis zu 254 Hex-Zeichen, links beginnend mit H" und rechts endend mit dem Separator Anführungszeichen. Ist die Anzahl Hex-Zeichen im hexadezimalen Literal ungerade, so bildet der Compiler eine hexadezimale Null rechts von dem äußersten rechten Hex-Zeichen in der Zeichenfolge. Da ein hexadezimales Literal als zur alphanumerischen Datenkategorie gehörig zu betrachten ist, kann es nur dort verwendet werden, wo ein nicht-numerisches Literal in einem allgemeinen Format zugelassen ist.

Die folgenden Beispiele sind hexadezimale Literale:

H"1C"	ergibt ein Byte mit dem Hexadezimalwert 1C
H"657F"	ergibt zwei Bytes mit dem Hexadezimalwert 657F
H"B1A"	ergibt zwei Bytes mit dem Hexadezimalwert B1A0

Boolesche Literale

Ein boolesches Literal ist eine Zeichenfolge, die nur die booleschen Zeichen 0 und 1 enthält. Ein boolesches Zeichen nimmt eine Bit-Stelle in einem Memory-Byte ein. Ein boolesches Literal besteht aus einer Reihe von booleschen Zeichen, links beginnend mit dem Separator B" und rechts endend mit dem Separator Anführungszeichen. Der Wert eines booleschen Literals wird durch die Reihe von booleschen Zeichen selbst dargestellt, die Begrenzungs-Separatoren sind dabei ausgenommen. Ein boolesches Literal gilt als zur booleschen Datenkategorie gehörig. Beträgt die Länge eines booleschen Literals nicht 8 Zeichen (Bits) oder ein Vielfaches davon, so wird es automatisch links durch 0-Zeichen (Bits) bis auf die nächste durch 8 teilbare Länge erweitert. Die größtmögliche Länge eines booleschen Literals beträgt 18 Bytes oder 144 boolesche Zeichen.

Das folgende sind Beispiele boolescher Literale:

B"01010101"	ergibt ein Byte mit den Bits 01010101
B"111"	ergibt ein Byte mit den Bits 00000111
B"0000111100001111"	ergibt zwei Bytes mit den Bits 0000111100001111
B"101110111"	ergibt zwei Bytes mit den Bits 0000000101110111

PICTURE-ZEICHENFOLGE

Eine PICTURE-Zeichenfolge besteht aus bestimmten Kombinationen von Zeichen des COBOL-Zeichenvorrates. Eine PICTURE-Zeichenfolge beschreibt die allgemeinen Merkmale und Aufbereitungserfordernisse eines Datenfeldes auf Elementarstufe.

Eine PICTURE-Zeichenfolge wird lediglich durch die Trennsymbole Leerzeichen, Komma, Semikolon oder Punkt begrenzt. Jedes Satzzeichen, das als Teil der PICTURE-Zeichenfolge auftritt, ist nicht als Satzzeichen zu betrachten, sondern gilt vielmehr als ein Symbol innerhalb der Spezifikation der PICTURE-Zeichenfolge.

KOMMENTAR

Kommentarzeilen beginnen mit einem Stern (*) in Spalte 7. Ein Sonderfall ist der Schrägstrich in Spalte 7 (/), der den Compiler zu einem Seitenvorschub veranlaßt und einen Kommentar zuläßt.

ANWEISUNGEN

Eine Anweisung ist eine syntaktisch gültige Kombination von Wörtern und Symbolen, die mit einem COBOL-Verb beginnt. Es gibt vier Arten von Anweisungen, nämlich Compiler-Anweisungen, bedingte Anweisungen, unbedingte Anweisungen und Delimited-Scope-Anweisungen.

Eine Compiler-Anweisung besteht aus einem Verb, das den Compiler zur Durchführung einer bestimmten Funktion, wie COPY oder USE, anweist. Dem Verb folgen dann Operanden, die vom Programmierer festgelegt werden. Das folgende sind Beispiele von Compiler-Anweisungen:

```
COPY STAMM-DATEN

USE AFTER STANDARD EXCEPTION PROCEDURE ON RECHN-DATEI
```

Eine bedingte Anweisung bestimmt, daß eine Bedingung auf ihren wahren Wert zu prüfen ist und daß der nachfolgende Vorgang von der Richtigkeit der Aussage abhängt. Bei COBOL sind die bedingten Anweisungen die folgenden:

```
EVALUATE und RETURN
ADD (mit SIZE ERROR-Klausel)
CALL (mit ON OVERFLOW-Klausel)
DELETE (mit INVALID KEY-Klausel)
DIVIDE (mit SIZE ERROR-Klausel)
IF
MULTIPLY (mit SIZE ERROR-Klausel)
READ (mit AT END- oder INVALID KEY-Klausel)
REWRITE (mit INVALID KEY-Klausel)
SEARCH
START (mit INVALID KEY-Klausel)
STRING (mit ON OVERFLOW-Klausel)
SUBTRACT (mit SIZE ERROR-Klausel)
UNSTRING (mit ON OVERFLOW-Klausel)
WRITE (mit INVALID KEY- oder END-OF-PAGE-Klausel)
COMPUTE (mit SIZE ERROR-Klausel)
RECEIVE (mit NO DATA-Klausel)
```

Beispiele bedingter Anweisungen:

```
IF SALDO IS EQUAL TO ZERO MOVE SUMME TO SUMME-NULL.
IF CODE-1 IS GREATER THAN ABC GO TO CODE-3-ROUTINE ELSE GO
TO ROUTINE-C.
ADD MENGE-1, MENGE-2, MENGE-3 GIVING TOTAL ON SIZE ERROR
GO TO UEBERLAUF.
```

Eine unbedingte Anweisung weist das Objekt-Programm zur Durchführung eines bestimmten Vorganges an. (Der zu beachtende Unterschied zwischen einer Compiler-Anweisung und einer unbedingten Anweisung ist folgender: Bei der ersteren ist der bestimmte Vorgang durch den Compiler auszuführen, wohingegen dies bei einer unbedingten Anweisung durch das Objekt-Programm zu geschehen hat.) Eine unbedingte Anweisung kann aus einer Folge unbedingter Anweisungen bestehen, von denen jede möglicherweise von der nächsten durch ein Trennsymbol getrennt ist. Bei COBOL sind die unbedingten Anweisungen die folgenden:

```
ACCEPT
ADD (ohne SIZE ERROR-Klausel)
ALTER
CALL (ohne ON OVERFLOW-Klausel)
CANCEL
CLOSE
COMPUTE
CONTINUE
DELETE (ohne INVALID KEY-Klausel)
DISABLE
DISPLAY
DIVIDE (ohne SIZE ERROR-Klausel)
ENABLE
EXIT
GO
INITIALIZE
INSPECT
MERGE
MOVE
MULTIPLY (ohne SIZE ERROR-Klausel)
OPEN
PERFORM
PURGE
READ (ohne AT END- oder INVALID KEY-Klausel)
RECEIVE (ohne NO DATA)
RELEASE
REWRITE (ohne INVALID KEY-Klausel)
```

SEND
SET
SHIFT
SORT
START (ohne INVALID KEY-Klausel)
STOP
STRING (ohne ON OVERFLOW-Klausel)
SUBTRACT (ohne SIZE ERROR-Klausel)
UNLOCK
UNSTRING (ohne ON OVERFLOW-Klausel)
WRITE (ohne INVALID KEY- oder END-OF-PAGE-Klausel)

Immer, wenn eine unbedingte Anweisung im allgemeinen Format eines COBOL-Verbs auftritt, kann sie eine oder mehrere aufeinanderfolgende unbedingte Anweisungen benennen. Die Folge von aufeinanderfolgenden unbedingten Anweisungen endet mit einem Punkt, einem ELSE in Verbindung mit einer vorausgehenden IF-Anweisung oder einer WHEN-Klausel in Verbindung mit einer vorausgehenden SEARCH-Anweisung.

Ein Satz stellt eine Folge von einer oder mehreren Anweisungen dar. Wie ein Satz in der deutschen Sprache, wird auch der COBOL-Satz durch einen von einer Leerstelle gefolgten Punkt abgeschlossen. Genauso wie es drei Arten von Anweisungen gibt, so gibt es auch drei Arten von Sätzen.

Der unbedingte Satz besteht aus einer oder mehreren unbedinten Anweisungen, die durch einen von einer Leerstelle gefolgten Punkt beendet werden. Ein Komma oder ein Strichpunkt können zwischen Anweisungen, die einen unbedingten Programmsatz bilden, verwendet werden.

Beispiele unbedingter Sätze:

MULTIPLY PREIS BY PROZENT GIVING RABATT.

ADD 400 TO SUMME GO TO PUNKT-8.

ADD 100 TO TOTAL, MOVE TOTAL TO NEU-TOTAL.

Der Compiler-Anweisungs-Satz ist eine eindeutige Anweisung, die durch einen Punkt beendet wird.

Beispiel eines Compiler-Anweisungs-Satzes:

USE AFTER STANDARD EXCEPTION PROCEDURE ON RECHN-DATEI.

Der bedingte Satz ist eine bedingte Anweisung, der wahlweise eine unbedingte Anweisung vorausgeht. Ein Komma oder ein Strichpunkt können zwischen den einen bedingten Satz ausmachenden Anweisungen verwendet werden. Ein bedingter Satz wird durch einen Punkt beendet.

Beispiele bedingter Sätze:

```
IF SALDO = NEU-SALDO GO TO ROUTINE-A ELSE MOVE "ABC"
TO FELD-A.

SUBTRACT 7 FROM MENGE, IF MENGE = 10 GO TO ROUTINE-C.
```

Eine Delimited-Scope-Anweisung ist jede Anweisung, die einen Explicit Scope Terminator enthält. Scope-Terminatoren begrenzen die Ausdehnung (Scope) gewisser Anweisungen in der Procedure Division.

Die Explicit Scope Terminatoren sind:

END-ADD	END-MULTIPLY	END-SEARCH
END-CALL	END-PERFORM	END-START
END-COMPUTE	END-READ	END-STRING
END-DELETE	END-RECEIVE	END-SUBTRACT
END-DIVIDE	END-RETURN	END-UNSTRING
END-EVALUATE	END-REWRITE	END-WRITE
END-IF		

Das Scope von Anweisungen, die ihrerseits in Anweisungen enthalten sind, kann ebenfalls (jedoch implizit) terminiert werden. Die implizite Terminierung schließt den abschließenden Punkt am Ende eines Satzes von Anweisungen mit ein, wodurch das Scope aller vorhergehenden und noch nicht beendeten Anweisungen insgesamt beendet (terminiert) wird. Dasselbe betrifft Aussagen wie ELSE, WHEN, NOT AT END usw. innerhalb einer übergeordneten Anweisung, wenn sie auf eine nicht beendete, untergeordnete Anweisung folgen.

BESCHREIBUNGEN UND KLAUSELN

Eine Beschreibung ist eine Reihe aufeinanderfolgender Klauseln, die durch einen Punkt beendet wird. Das allgemeine Format einer Beschreibung enthält viele Klauseln, von denen der Programmierer jedoch nur die für die Situation benötigten auswählt. Jede Klausel ist ein geordneter Satz von COBOL-Zeichenfolgen, deren Zweck es ist, eine bestimmte Charakteristik des Formats zu spezifizieren. In einer Beschreibung werden die Klauseln durch zumindest ein Leerzeichen getrennt. Zwischen den einzelnen Klauseln kann ein Komma oder ein Semikolon verwendet werden.

Beispiel einer Datenbeschreibung:

```
FD RECHN-DATEI BLOCK CONTAINS 27 RECORDS RECORD CONTAINS
20 CHARACTERS LABEL RECORD IS STANDARD.
```

Eine Datenbeschreibung für ein Datenfeld,
das den konstanten Wert null enthält:

```
02 SEITENNUMMER PIC 999 VALUE IS ZERO.
```

PARAGRAPHEN

Ein Paragraph besteht aus null, einem oder mehreren Sätzen oder Beschreibungen mit einer gemeinsamen Funktion. In der IDENTIFICATION- und ENVIRONMENT DIVISION beginnt jeder Paragraph mit einem reservierten Wort, Paragraphen-Überschrift genannt, auf die ein Punkt und mindestens eine Leerstelle folgt.

In der PROCEDURE DIVISION beginnt jeder Paragraph mit einem Paragraphen-Namen, auf den ein Punkt und eine Leerstelle folgt. Der Paragraphen-Name oder die Paragraphen-Überschrift beginnt in Spalte 8, 9, 10 oder 11 des COBOL-Programmblattes. Ein Paragraph endet mit dem Auftreten des nächsten Paragraphen-Namens, der nächsten Paragraphen-Überschrift, Kapitel-Überschrift, DIVISION-Überschrift oder dem Ende der PROCEDURE DIVISION.

Format für zwei Paragraphen, die den Namen LOESCH-ROUTINE und AENDER-ROUTINE führen:

```
8
LOESCH-ROUTINE. MOVE ZERO TO FLAG-D.
    MOVE KONTO-NR TO DRUCK-KONTO-NR.
    MOVE SALDO TO DRUCK-SALDO; MOVE ACODE TO PCODE.
    WRITE DRUCKSATZ.
AENDER-ROUTINE.
    ADD UEBERW-BETRAG TO SALDO.
    MOVE UEBERW-DATUM TO HAUPT-DATUM.
    IF FAELLIG-DATUM IS EQUAL TO TAGES-DAT GO TO RECHN-ROUTINE.
```

KAPITEL (SECTIONS)

Ein Kapitel besteht aus null, einem oder mehreren aufeinanderfolgenden Paragraphen in der ENVIRONMENT- und PROCEDURE-DIVISION sowie einer Reihe von Datendefinitionen in der DATA DIVISION. In der ENVIRONMENT- und DATA DIVISION wird jedes Kapitel durch eine aus reservierten Wörtern bestehende Kapitel-Überschrift eingeleitet, auf die ein Punkt und eine Leerstelle folgt. In der PROCEDURE DIVISION wird jedes Kapitel durch eine Kapitel-Überschrift eingeleitet, die aus einem von dem reservierten Wort SECTION abgeschlossenen Kapitel-Namen, einer wahlweisen Segment-Nummer, einem Punkt und einer Leerstelle besteht. Die Kapitel-Überschrift beginnt in Spalte 8, 9, 10 oder 11 des COBOL-Programmblattes. Ein Kapitel endet mit dem Auftreten der nächsten Kapitel-Überschrift, der nächsten DIVISION-Überschrift oder dem Ende der PROCEDURE DIVISION.

Format für zwei Kapitel in der PROCEDURE DIVISION:

```
8
HAUPT SECTION.
H1. READ DATEI AT END GO TO ENDE-ROUTINE.
    MOVE KART-SATZ TO MAG-SATZ.
    IF TCODE IS GREATER THAN 5 GO TO FAELL-DAT-A
        ELSE GO TO FAELL-DAT-B.

UNTERROUTINE SECTION.
ENDE-ROUTINE. CLOSE MAG-DATEI, DATEI, FEHL-DATEI.
    STOP RUN.
FEHL-ROUTINE.
    MOVE KART-SATZ TO FEHL-SATZ.
    WRITE FEHLER.
FAELL-DAT-A.
```

TEILE (DIVISIONS)

Ein COBOL-Programm besteht aus vier Haupt-Teilen, von denen jedes aus zugehörigen Kapiteln (SECTIONS) oder Paragraphen besteht, die den Compiler über bestimmte Aspekte des Programms informieren. Zur Umwandlung eines Programmes müssen die DIVISIONS dem Compiler in bestimmter Reihenfolge zugeführt werden und müssen dabei in dem erforderlichen Format geschrieben sein.

Die vier COBOL-TEILE sind:

IDENTIFICATION DIVISION (Identifikationsteil), ENVIRONMENT DIVISION (Maschinenteil), DATA DIVISION (Datenteil) und PROCEDURE DIVISION (Prozedurteil).

Die IDENTIFICATION DIVISION enthält den Programm-Namen und andere dokumentarische Informationen, wie z. B. Autoren-Name, Installations-Name, Programmerstellungs-Datum, Programmumwandlungs-Datum.

Die ENVIRONMENT DIVISION beschreibt den Computer, auf dem das Programm umgewandelt wird (den Source-Computer), und den Computer, auf dem Objekt-Programm auszuführen ist (den Object-Computer). Sie gibt auch Information betreffend die Peripherie für die Ein-/Ausgaben, System-Software-Schalter und Merk-Namen.

Die DATA DIVISION beschreibt die durch das Objekt-Programm zu verarbeitenden Daten.

Die PROCEDURE DIVISION enthält die Anweisungen, die dem Computer schrittweise Instruktionen zur Verarbeitung der Daten im Programm geben.

Der Beginn einer jeden DIVISION ist gekennzeichnet durch eine DIVISION-Überschrift, die aus einer Kombination von reservierten Wörtern besteht, die von einem Punkt und einem Leerzeichen abgeschlossen wird. Im folgenden Beispiel sind die DIVISION-Überschriften IDENTIFICATION DIVISION, ENVIRONMENT DIVISION, DATA DIVISION und PROCEDURE DIVISION zu sehen.

Beispiel eines Cobol-Programmes:

```
8
IDENTIFICATION DIVISION.
PROGRAMM-ID. NEUANL.


ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. NCR-ITX.
OBJECT-COMPUTER. NCR-ITX.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
     SELECT DATEI ASSIGN TO DISC
          ORGANIZATION IS SEQUENTIAL
          ACCESS MODE IS SEQUENTIAL.
     SELECT DATEI1 ASSIGN TO DISC
          ORGANIZATION IS INDEXED
          ACCESS MODE IS SEQUENTIAL
          RECORD KEY IS ST.

DATA DIVISION.
FILE SECTION.
FD   DATEI BLOCK CONTAINS 10 RECORDS
          RECORD CONTAINS 48 CHARACTERS
          LABEL RECORD IS STANDARD
          VALUE OF FILE-ID IS "DATEI".
01   DATSATZ      PIC X(48).

FD   DATEI1 BLOCK CONTAINS 10 RECORDS
          RECORD CONTAINS 48 CHARACTERS
          LABEL RECORD IS STANDARD
          VALUE OF FILE-ID IS "DATEI1".
01   DATSATZ1.
     02   FILLER PIC XXXX.
     02   ST      PIC XXX.
     02   FILLER PIC X(41).

PROCEDURE DIVISION.
BEGINN.
     OPEN INPUT DATEI.  OPEN OUTPUT DATEI1.
LES-EING.
     READ DATEI AT END GO TO SCHLUSS.
     MOVE DATSATZ TO DATSATZ1.
     WRITE DATSATZ1 INVALID KEY GO TO FEHLER-ROUTINE.
     GO TO LES-EING.

FEHLER-ROUTINE.
     DISPLAY "DATEI1 FEHLER".
SCHLUSS.
     CLOSE DATEI, DATEI1.
     STOP RUN.
```

KAPITEL 3: DAS COBOL-PROGRAMMBLATT

Der Programmierer schreibt ein COBOL-Programm auf das NCR COBOL-Programmblatt. Von diesem Blatt weg wird das Programm anschließend mit Hilfe eines Editors erfaßt und auf eine Platte gespeichert.

Spalten 1 bis 6 auf diesem Programmblatt erlauben eine aus sechs Ziffern bestehende Seiten-/Zeilennummer. Diese kann Ordnungs- und Kopierzwecken dienen, ist jedoch nicht obligatorisch.

Bei der Zuordnung teilt der Programmierer die niedrigste Nummer der ersten Ursprungszeile zu. Nachfolgende Ursprungszeilen erhalten dann höhere Folgenummern.

Spalte 8 des COBOL-Programmblatts ist die am weitesten links gelegene Position für den Schreibbeginn der Division-Überschrift, der Kapitel-Überschrift, der Paragraphen-Überschrift und des Paragraphen-Namens. Jede dieser Bezeichnungen kann auch ab Spalte 9, 10 oder 11 geschrieben werden. Die Spalten 8, 9, 10 und 11 werden Zone A genannt.

Spalte 12 des COBOL-Programmblattes ist die am weitesten links gelegene Position für den Schreibbeginn der Eintragungen im Paragraphen. Der erste Satz oder die erste Eintragung eines Paragraphen kann auf derselben Zeile wie die Paragraphen-Überschrift oder der Paragraphen-Name beginnen. Der erste Satz oder die erste Eintragung eines Paragraphen kann auch auf der sich unmittelbar unterhalb der Paragraphen-Überschrift oder dem Paragraphen-Namen befindenden Zeile beginnen. Der Inhalt des Paragraphen wird in jeder Position ab Spalte 12 geschrieben. Die Spalten 12 bis 72 werden Zone B genannt.

Spalte 72 des COBOL-Programmblattesist die am weitestens rechts gelegene Position in einer Ursprungszeile. Wenn in der Spalte 72 ein anderes Zeichen als ein Leerzeichen erscheint, so wird vom Compiler angenommen, daß auf dieses letzte Zeichen in der Ursprungszeile ein Leerzeichen folgt, es sei denn, es tritt ein Bindestrich in Spalte 7 der nächsten Zeile auf. Die Spalten 73 bis 80 können jede Art von Zeichen für Identifikationszwecke enthalten. Der Inhalt der Spalten 73 bis 80 erscheint in der Compiler-Liste, ohne daß er in das Objektprogramm einfließt.

Das nachfolgende Beispiel zeigt die Positionen der COBOL-Ursprungszeilen. Man beachte, daß der Paragraphen-inhalt auf der gleichen Zeile wie die Paragraphen-Überschrift SOURCE-COMPUTER geschrieben ist, ferner daß der Schreibbeginn des Paragrapheninhalts auf der Zeile unterhalb der Paragraphen-Überschrift PROGRAM-ID und auch unter dem Paragraphen-Namen ANFANG liegt.

```
8
IDENTIFICATION DIVISION.
PROGRAM-ID.
    EINKAUF.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. NCR-ITX.
---

PROCEDURE DIVISION.
ANFANG.
    OPEN INPUT EINDAT. MOVE ZERO TO FELD.
    READ EINDAT AT END GO TO SCHLUSS.
    ---
```

ZEILENFORTSETZUNG

Während ein Wort oder ein numerisches Literal nicht auf zwei oder mehr Ursprungszeilen aufgeteilt werden kann, dürfen ein nicht-numerisches, hexadézimales oder boolesches Literal auf zwei oder mehr Ursprungszeilen verteilt sein. Das Zeichen Bindestrich (-) in Spalte 7 zeigt an, daß eine Ursprungszeile, Fortsetzungs-Zeile genannt, die Fortsetzung eines nicht-numerischen, hexadezimalen oder booleschen Literals aus der vorangehenden, die fortgesetzte Zeile genannten Ursprungszeile enthält.

Enthält die fortgesetzte Zeile ein nicht-numerisches, hexa-dezimales oder boolesches Literal ohne ein abschließendes Anführungszeichen, so muß das erste kein Leerzeichen darstellende Zeichen ab Spalte 12 der Fortsetzungs-Zeile ein Anführungszeichen sein. Die Fortsetzung wird mit dem unmit-telbar auf das Anführungszeichen folgenden Zeichen geschrie-ben. Alle Leerzeichen am Ende der fortgesetzten Zeile gelten als Teil des Literals.

Beispiel eines fortgesetzten nicht-numerischen Literals, das den Wert ABCDEFG aufweist:

```
                                                                      7
7                                                                     2
     ADD BETRAG TO SALDO. MOVE SALDO TO WSALDO.      MOVE "ABCD
-       "EFG" TO STATUS-CODE.
```

Beispiel eines fortgesetzten hexadezimalen Literals, das den Wert 140D61781F aufweist:

```
                                                                      7
7                                                                     2
     MOVE NEU-DAT TO STAMM-DAT.  ADD 1 TO ZAEHLA. MOVE H"140D6
-         "1781F" TO COM-CODE.
```

Beispiel eines fortgesetzten booleschen Literals, das den Wert 00011111 aufweist:

```
                                                                      7
7                                                                     2
     SUBTRACT TBETRAG FROM ALTSALDO GIVING NEUSALDO.  MOVE B"O
-             "0011111" TO CODES.
```

KOMMENTAR-ZEILE

Ein Stern (*) in Spalte 7 kennzeichnet die ganze Zeile als Kommentar. Eine Kommentar-Zeile wird auf der Compiler-Liste lediglich dokumentarisch ausgedruckt. Eine Kommentar-Zeile kann als Zeile im Ursprungs-Programm nach der Division-Überschrift IDENTIFICATION DIVISION überall erscheinen. Jede Kombination von Zeichen des dargestellten ASCII-Zeichenvorrats kann ab Spalte 8 der Kommentar-Zeile verwendet werden. Das folgende ist ein Beispiel zur Verwendung des Sterns als Kennzeichnung einer Kommentar-Zeile.

```
7
     ADD MENGE TO SUMME.
* AUFTRAGSKALKULATION.
     MOVE .015 TO PREIS.
```

KOMMENTAR-ZEILE MIT SEITENVORSCHUB

Ein Schrägstrich (/) in Spalte 7 bewirkt eine Seitenaufschaltung der Compiler-Liste. Die das Zeichen Schrägstrich enthaltende Ursprungszeile wird als eine Kommentar-Zeile als erste auf der neuen Seite gedruckt. Beispiel der Verwendung des Zeichens Schrägstrich in Spalte 7 für eine Seitenaufschaltung mit Drucken des Kommentars PREIS-PRUEFUNG auf der neuen Seite:

```
7
     GO TO PRUEF-ENDE.
/PREISPRUEFUNG
 PREIS-PRUEF.
     IF PREI IS LESS THAN ZERO GO TO FALSCHPREIS.
```

LEERZEILE

Eine Leerzeile ist eine von Spalte 7 bis Spalte 72 durchgehend leere Ursprungszeile. Eine Leerzeile kann überall im COBOL Ursprungs-Programm auftreten, außer unmittelbar vor einer Fortsetzungs-Zeile mit dem Bindestrich (-) in Spalte 7. Beispiel einer Leerzeile, um in der Compiler-Liste senkrechten Abstand zu schaffen.

```
7
     SELECT MDATEI ASSIGN TO DISC.

 DATA DIVISION.
```

DATA DIVISION (ÜBERBLICK)

Die Data Division ist die einzige Division, die ein eigenes Format auf dem Programmblatt aufweist. Die Formulierung der Division-Überschrift und der Kapitel-Überschriften beginnt in der üblichen Position, nämlich in Spalte 8, 9, 10 oder 11. Der Inhalt eines jeden Kapitels wird jedoch nach den folgenden Regeln auf dem Programmblatt eingetragen:

1. FD, SD oder CD haben in Spalte 8, 9, 10 oder 11 zu beginnen und auf diese Stufenbezeichnung muß mindestens ein Leerzeichen folgen.

2. Die Formulierung des auf die Stufenbezeichnung folgenden Datei-Namens oder CD-Namens hat in Spalte 12 oder jeder beliebigen Spalte rechts davon zu beginnen. Die Stufenbezeichnung und der darauffolgende Datei-Name oder CD-Name müssen durch mindestens ein Leerzeichen getrennt sein.

3. Die Formulierung der Stufennummer 01 (oder 1) hat in den Spalten 8, 9, 10 oder 11 zu beginnen und auf diese Stufennummer muß mindestens ein Leerzeichen folgen.

4. Die Formulierung der Stufennummer 02 (2) bis 49 sowie 66 und 88 hat in Spalte 8 oder jeder Spalte rechts davon zu beginnen und auf diese Stufennummern muß jeweils mindestens ein Leerzeichen folgen.

5. Die Formulierung der Stufennummer 66, 77 und 88 hat in Spalten 8, 9, 10 oder 11 zu beginnen und auf diese Stufennummer muß mindestens ein Leerzeichen folgen.

6. Die Formulierung einer numerisch höheren Stufennummer kann in der gleichen Position wie die vorausgehende Stufennummer beginnen oder diese numerisch höhere Stufennummer kann von der Spalte 8 ab beliebig nach rechts eingerückt werden.

7. Die Formulierung des auf die Stufennummer folgenden Identifiers oder FILLERs muß in Spalte 12 oder jeder beliebigen Spalte rechts davon beginnen. Es muß jedoch mindestens ein Leerzeichen zwischen der Stufennummer und dem Identifier oder FILLER vorhanden sein.

Von dem Daten-Satz, wie er nachfolgend in einem Diagramm dargestellt ist, zeigen die Beispiele 1, 2 und 3 einige der verschiedenen Formulierungs-Möglichkeiten auf dem COBOL-Programmblatt.

TEILESATZ								
TEILE-NUMMER	HERST-DATUM	STATUS-DATEN				INVENTUR-DATUM		
		IN ARBEIT	MENGE1	MENGE2	MENGE3	IMON	ITAG	IJHR
9 9 9 9	9 9 9 9 9 9	9 9 9	9 9 9	9 9 9	9 9 9	9 9	9 9	9 9

BEISPIEL 1:

```
FD  TEILEDATEI BLOCK CONTAINS 16 RECORDS
        RECORD CONTAINS 32 CHARACTERS
        LABEL RECORD IS STANDARD
        VALUE OF FILE-ID IS "TEILEDATEI".
01  TEILESATZ.
02  TEILENUMMER            PIC 9(4).
02  HERST-DATUM            PIC 9(6).
02  STATUS-DATEN.
03  IN-ARBEIT              PIC 999.
03  MENGE1                 PIC 999.
03  MENGE2                 PIC 999.
03  MENGE3                 PIC 999.
02  INVENTUR-DATUM.
03  IMON                   PIC 99.
03  ITAG                   PIC 99.
03  IJAHR                  PIC 99.
```

BEISPIEL 2:

```
FD  TEILEDATEI BLOCK CONTAINS 16 RECORDS
        RECORD CONTAINS 32 CHARACTERS
        LABEL RECORD IS STANDARD
        VALUE OF FILE-ID IS "TEILEDATEI".
1   TEILESATZ.
2   TEILENUMMER             PIC 9(4).
2   HERST-DATUM             PIC 9(6).
2   STATUS-DATEN.
3   IN-ARBEIT               PIC 999.
3   MENGE1                  PIC 999.
3   MENGE2                  PIC 999.
3   MENGE3                  PIC 999.
2   INVENTUR-DATUM.
3   IMON                    PIC 99.
3   ITAG                    PIC 99.
3   IJAHR                   PIC 99.
```

BEISPIEL 3:

```
FD  TEILEDATEI BLOCK CONTAINS 16 RECORDS
        RECORD CONTAINS 32 CHARACTERS
        LABEL RECORD IS STANDARD
        VALUE OF FILE-ID IS "TEILEDATEI".
01  TEILESATZ.
    02  TEILENUMMER      PIC 9(4).
    02  HERST-DATUM      PIC 9(6).
    02  STATUS-DATEN.
        03  IN-ARBEIT    PIC 999.
        03  MENGE1       PIC 999.
        03  MENGE2       PIC 999.
        03  MENGE3       PIC 999.
    02  INVENTURDATUM.
        03  IMON         PIC 99.
        03  ITAG         PIC 99.
        03  IJAHR        PIC 99.
```

KAPITEL 4: SCHREIBWEISE, ALLGEMEINE FORMATE, PRINZIPIEN, PROGRAMMSTRUKTUR

Obwohl COBOL dem Programmierer die Möglichkeit gibt, sein Programm in einem "Nahezu-Englisch" zu schreiben, ist doch der Aufbau des Programms abhängig von zu beachtenden Regeln. Jede Anweisung oder Klausel erfordert eine Standard-Position für bestimmte Wörter und Satzzeichen. Der Programmierer muß diese Wörter und Satzzeichen in der Anweisung oder Klausel richtig setzen. Der Programmierer muß die Wörter auch richtig buchstabiert schreiben, so daß sie der Compiler in Anweisungen der Maschinensprache übersetzen kann.

Im COBOL wird zur Festlegung der Struktur jeder Anweisung und Klausel ein generelles Format verwendet. Durch dieses ganze Buch werden durchweg generelle Formate aufgezeigt. Das generelle Format ist in numerierte Formate aufgeteilt, wenn mehr als eine Möglichkeit für eine Klausel oder Anweisung erlaubt ist. Klauseln und Zusätze müssen in der vom generellen Format bestimmten Reihenfolge geschrieben werden, es sei denn, eine dem generellen Format zugehörige Regel erlaubt es ausdrücklich, die Reihenfolge zu ändern. Nun zur Erklärung der verschiedenen Schreibweisen in den generellen COBOL-Formaten.

UNTERSTRICHENE, IN GROSSBUCHSTABEN GESCHRIEBENE WÖRTER

Alle Schlüsselwörter im COBOL-Format sind unterstrichen und in Großbuchstaben geschrieben. Bei einer Klausel oder Anweisung mit einem oder mehr als einem Schlüsselwort muß das Schlüsselwort bzw. müssen alle Schlüsselwörter vorhanden und richtig buchstabiert sein. In der folgenden Klausel sind die Schlüsselwörter CURRENCY und IS.

CURRENCY SIGN IS Literal

IN GROSSBUCHSTABEN GESCHRIEBENE WÖRTER

Alle Wahlwörter des COBOL-Formats sind in Großbuchstaben geschrieben, jedoch nicht unterstrichen. Wahlwörter dienen im COBOL-Format zur besseren Lesbarkeit des Textes; sie können weggelassen werden. Wird aber ein Wahlwort benutzt, dann muß es richtig geschrieben werden. In der obigen Klausel ist das Wahlwort SIGN.

IN KLEINBUCHSTABEN GESCHRIEBENE WÖRTER

In Kleinbuchstaben geschriebene Wörter des COBOL-Formats sind generelle Bezeichnungen (Operanden) und stellen Programmiererwörter wie Literale, PICTURE-Zeichenfolgen oder Kommentar-Eintragungen dar. In einer Klausel oder Anweisung mit einem oder mehreren Operanden muß der Operand bzw. müssen alle Operanden vorhanden sein. In den Formaten kann es sich bei einem Operanden um einen Datenfeldnamen, eine Stufen-Nummer, einen Prozedur-Namen oder ein Literal handeln. In der obigen Klausel zeigt der Operand Literal an, daß der Programmierer ein als Währungszeichen zu verwendendes Literal einzusetzen hat, zum Beispiel: CURRENCY SIGN IS "F".

BRACKETS Eckige Klammern

Wenn ein Teil eines allgemeinen Formats in eckigen Klammern steht, kann dieser in gewissen Fällen weggelassen werden. Die vertikale Anordnung von zwei oder mehr Format-Möglichkeiten innerhalb eckiger Klammern zeigt an, daß der Programmierer, wenn der Teil in eckigen Klammern verwendet werden soll, eines dieser Formate auswählen muß. Wahlwörter in eckigen Klammern müssen nicht geschrieben werden. Im folgenden allgemeinen Format zeigen eckige Klammern an, daß Möglichkeiten einer MULTIPLY-Anweisung ROUNDED und ON SIZE sind.

```
MULTIPLY { Identifier-1 } BY { Identifier-2 } GIVING Identifier-3
         { Literal-1    }    { Literal-2    }
    [ROUNDED] [ON SIZE ERROR unbedingte Anweisung]
```

BRACES Geschweifte Klammern

Ist ein Teil des allgemeinen Formats in geschweiften Klammern eingeschlossen, muß der Programmierer eines der vertikal angeordneten Formate innerhalb der geschweiften Klammern auswählen. Wahlwörter in geschweiften Klammern müssen nicht geschrieben werden. Im obigen allgemeinen Format der MULTIPLY-Anweisung zeigen die ersten geschweiften Klammern an, daß entweder Identifier-1 oder Literal-1 vorhanden sein müssen, und die zweiten geschweiften Klammern zeigen an, daß entweder Identifier-2 oder Literal-2 vorhanden sein müssen. Gibt es nur eine Möglichkeit innerhalb der geschweiften Klammern, so ist es die Funktion dieser Klammern, den Teil des Formats, auf den eine nachfolgende Ellipsis zutrifft, zu begrenzen.

ELLIPSIS bzw. Drei-Punkt-Zeichen

Die Ellipsis besteht aus drei aufeinanderfolgenden Punkten, die entweder auf eckige oder geschweifte Klammern folgen. Der Zweck der Ellipsis ist, den Programmierer zu informieren, daß der in den eckigen Klammern oder in den geschweiften Klammern eingeschlossene Formatteil, wenn notwendig, wiederholt werden kann. In dem folgenden allgemeinen Format zeigt die Ellipsis an, daß ein zweiter, dritter, vierter usw. Prozedur-Name in der GO TO DEPENDING-Anweisung verwendet werden kann.

```
GO TO Prozedur-Name-1 [Prozedur-Name-2 ... Prozedur-Name-n]
     DEPENDING ON Identifier.
```

KOMMA oder SEMIKOLON

Die Satzzeichen Komma und Semikolon können verwendet werden. Komma und Semikolon sind auch austauschbar. Steht ein Komma in einem allgemeinen Format, so kann ein Semikolon dieses ersetzen. Steht ein Semikolon in einem allgemeinen Format, so kann ein Komma dieses ersetzen. Unmittelbar der ersten Klausel einer Erklärung oder eines Paragraphen vorausgehend darf weder ein Komma noch ein Semikolon erscheinen. In der PROCEDURE DIVISION kann ein Semikolon oder ein Komma zwischen den Anweisungen verwendet werden.

PUNKT

Ein Punkt, dem ein Leerzeichen folgt, ist zum Abschluß einer Beschreibung, eines Satzes, einer Paragraphen-Überschrift, eines Paragraphen-Namens, einer Kapitel-Überschrift und einer Division-Überschrift notwendig. Paragraphen innerhalb der IDENTIFICATION DIVISION und der PROCEDURE DIVISION müssen mit einem Punkt beendet werden. Eintragungen innerhalb der ENVIRONMENT DIVISION und der DATA DIVISION müssen mit einem Punkt beendet werden.

Noch einige Prinzipien und Begriffe:

ADRESSIERUNG

Befehle können Datenfelder explizit oder implizit adressieren.

EXPLIZIT nennt man die direkte Adressierung eines Feldes durch die Angabe seines Namens im Befehl.

IMPLIZIT nennt man die indirekte Adressierung eines Feldes. Als Beispiel mag hier das Register LINAGE-COUNTER dienen (Zeilenzähler). Mit einem Befehl WRITE ZEILE AFTER 2 LINES wird um 2 Zeilen aufgeschaltet und dann die Zeile gedruckt. Der Befehl bewirkt zugleich, daß der Wert 2 in das Register LINAGE-COUNTER addiert wird, ohne daß dessen Name zu nennen ist.

PROGRAMMABLAUF

COBOL bietet explizit und implizit Möglichkeiten zur Steuerung des Programmablaufs. In der Regel werden die Programm-Befehle immer in der Sequenz abgearbeitet, in der sie geschrieben sind, es sei denn, es erfolgt eine explizite Durchbrechung dieser Regel.

IMPLIZIT ist der serielle Befehlsablauf zu nennen, weil das System von sich aus von Befehl zu Befehl übergeht, ohne daß im Programm hierzu eigens eine Aufforderung nötig ist. IMPLIZIT sind auch solche Abläufe zu nennen, bei denen das System von sich aus Programmschleifen durchläuft oder Verzweigungen durchführt, ohne daß hierzu mittels einer programmierten Verzweigungslogik des Programmierers aufgefordert werden muß. Als Beispiele können die Befehle SORT und PERFORM dienen.

EXPLIZIT ist eine Programmablaufsteuerung zu nennen, wenn ein Implizitablauf durch einen Verzweigungsbefehl oder eine Bedingungsanweisung beeinflußt wird.

ATTRIBUTE

Attribute können EXPLIZIT oder IMPLIZIT vorhanden sein.

Vom Programmierer nicht ausdrücklich angegebene Attribute (das wären explizite Attribute) sind, wenn das System sie von sich aus annimmt, implizite Attribute (auch Defaults genannt).

Zum Beispiel muß die Speicherungsform (Usage) eines Datenfeldes nicht angegeben werden. Verzichtet der Programmierer auf ein explizit angegebenes Attribut COMP oder INDEX oder DISPLAY, wird vom System implizit das Attribut DISPLAY als Default angenommen.

DIE STRUKTUR EINES COBOL-URSPRUNGSPROGRAMMES

Mit Ausnahme von COPY- und REPLACE-Anweisungen sowie dem END-PROGRAM-HEADER werden Befehle, Paragraphen und Sections eines COBOL-Programmes in folgender Reihenfolge in vier DIVISIONS gruppiert:

1. Identification Division
2. Environment Division
3. Data Division
4. Procedure Division

Das Ende eines COBOL-Programmes wird entweder durch einen End-Program-Header angezeigt (falls vorhanden) oder durch das Fehlen weiterer Programmzeilen.

END PROGRAM HEADER

Er zeigt das Ende des namentlich genannten Programmes an.

Format: END PROGRAM programm-name.

Der Programmname muß identisch sein mit einem Programmnamen, der in einem vorhergehenden PROGRAM-ID-Paragraphen angegeben wurde.

Ein Programm kann in einem anderen Programm enthalten sein. Der PROGRAM-ID-Paragraph, der den Namen des enthaltenen Programmes ausweist, muß sich zwischen dem PROGRAM-ID-Paragraphen und dem End-Program-Header des enthaltenden Programmes befinden.

Der End-Program-Header muß in jedem Programm vorhanden sein, das entweder andere Programme enthält oder in einem anderen Programm enthalten ist. Ist das mit einem End-Program-Header abgeschlossene Programm in einem Programm enthalten, muß der darauffolgende Satz entweder eine IDENTIFICATION DIVISION-Zeile eines weiteren Programmes sein oder ein zweiter End-Program-Header, der das enthaltende Programm abschließt.

VERSCHACHTELTE PROGRAMME

Ein COBOL-Programm kann andere COBOL-Programme enthalten. Enthaltene Programme können einige Einrichtungen derjenigen Programme ansprechen, in denen sie enthalten sind.

Ein Programm gilt als in einem anderen Programm direkt enthalten, wenn es nicht gleichzeitig in einem anderen Programm enthalten ist. Ein Programm gilt als indirekt enthalten, wenn eine tiefergehende Verschachtelung vorliegt, wie das folgende Beispiel zeigt:

1. A direkt enthalten in B:

   ```
   Programm-Header für B
   Codierung für B
   Programm-Header für A
   Codierung für A
   End-Program-Header für A
   End-Program-Header für B
   ```

2. A indirekt enthalten in B,
   aber direkt enthalten in C:

   ```
   Programm-Header für B
   Codierung für B
   Programm-Header für C
   Codierung für C
   Programm-Header für A
   Codierung für A
   End-Program-Header für A
   End-Program-Header für C
   End-Program-Header für B
   ```

Es sind fünf Verschachtelungsstufen möglich.

Ein COBOL-Programm, das direkt oder indirekt in einem anderen Programm enthalten ist, wird als ein separates Programm betrachtet, welches zusätzlich gewisse Einrichtungen ansprechen kann, die in dem übergeordneten Programm definiert sind.

Nach erfolgter Compilation sind enthaltene und enthaltende Programme untrennbar.

GENERELLES FORMAT VERSCHACHTELTER PROGRAMME

```
IDENTIFICATION DIVISION
[ ENVIRONMENT DIVISION ]
[ DATA DIVISION ]
[ PROCEDURE DIVISION ]
[ END-PROGRAM-HEADER ]

IDENTIFICATION DIVISION.
PROGRAM-ID. program-name-1 [ IS INITIAL PROGRAM ].
[ ENVIRONMENT DIVISION. environment-division-content ]
[ DATA DIVISION. data-division-content ]
[ PROCEDURE DIVISION. procedure-division-content ]
[ [ nested-source-program ] ...
END PROGRAM program-name-1. ]

IDENTIFICATION DIVISION.
PROGRAM-ID. program-name-2 [ IS { COMMON } PROGRAM ]
                               { INITIAL }
[ ENVIRONMENT DIVISION. environment-division-content ]
[ DATA DIVISION. data-division-content ]
[ PROCEDURE DIVISION. procedure-division-content ]
[ nested-source-program ] ...
END PROGRAM program-name-2.

[ IDENTIFICATION DIVISION.
PROGRAM-ID. program-name-3 [ IS INITIAL PROGRAM ].
[ ENVIRONMENT DIVISION. environment-division-content ]
[ DATA DIVISION. data-division-content ]
[ PROCEDURE DIVISION. procedure-division-content ]
[ nested-source-program ] ...
END PROGRAM program-name-3. ] ...
IDENTIFICATION DIVISION.
PROGRAM-ID. program-name-4 [ IS INITIAL PROGRAM ].
[ ENVIRONMENT DIVISION. environment-division-content ]
[ DATA DIVISION. data-division-content ]
[ PROCEDURE DIVISION. procedure-division-content ]
[ [ nested-source-program ] ...
END PROGRAM program-name-4. ]
```

KAPITEL 5: IDENTIFICATION DIVISION

Der Zweck der IDENTIFICATION DIVISION ist es, das Ursprungs-Programm zu identifizieren. Der Programmierer kann auch das Datum, an dem das Programm geschrieben wurde, den Autoren-Namen und verschiedene andere Informationen aufnehmen.

Die IDENTIFICATION DIVISION ist der erste Teil, der vom COBOL-Compiler eingelesen wird. Er ist der einzige Teil eines COBOL-Programmes, der nicht in Kapitel (SECTIONS) unterteilt ist; er ist stattdessen in die Paragraphen 1 bis 6 unterteilt. Paragraphen-Überschriften machen die Art der in den Paragraphen enthaltenen Information kenntlich. Als Beispiel seien genannt PROGRAM-ID, AUTHOR, INSTALLATION, DATE-WRITTEN, DATE-COMPILED und SECURITY.

Der einzige Paragraph, der in der IDENTIFICATION DIVISION erforderlich ist, ist derjenige, der den Programm-Namen enthält und durch die Paragraphen-Überschrift PROGRAM-ID eingeleitet wird. Die anderen Paragraphen sind wahlweise; es ist Sache des Programmierers, ob er diese Paragraphen in die IDENTIFICATION DIVISION aufnimmt. Nimmt der Programmierer diese Paragraphen jedoch auf, so muß er beim Schreiben die Reihenfolge des unten gezeigten allgemeinen Formats einhalten.

Das folgende allgemeine Format zeigt die Struktur der IDENTIFICATION DIVISION. (Das am weitesten links stehende Zeichen des allgemeinen Formats ist in Spalte 8 des COBOL-Programmblattes geschrieben.)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. programm-name [ IS { COMMON  } PROGRAM ].
                             { INITIAL }

[AUTHOR.                    [kommentar]...]
[INSTALLATION.              [kommentar]...]
[DATE-WRITTEN.              [kommentar]...]
[DATE-COMPILED.             [kommentar]...]
[SECURITY.                  [kommentar]...]
[REMARKS.                   [kommentar]...]
```

Die IDENTIFICATION DIVISION beginnt mit einer Division-Überschrift aus den reservierten Wörtern IDENTIFICATION DIVISION, auf die ein Punkt und ein Leerzeichen folgen.

Der Paragraph PROGRAM-ID, der den Erkennungsnamen des Programms angibt, folgt unmittelbar auf die Division-Überschrift. Wie bereits erwähnt, bezeichnet der Programm-Name das COBOL Ursprungs-Programm, das Objekt-Programm und alle zu einem bestimmten Programm gehörenden Ausdrucke. Ein Programm-Name ist ein Programmiererwort, das lediglich aus den Zeichen des Wortvorrats bestehen darf. Ein Programm-Name muß mindestens ein alphabetisches Zeichen in einer beliebigen Position darin enthalten. Jeder Programm-Name muß eindeutig sein. Es wird empfohlen, die letzten zwei Zeichen des Programm-Namens als vom Programmierer zu kontrollierende Versions-Nummer zu benutzen.

COMMON, falls angegeben, besagt, daß das Programm in einem Programm enthalten ist, jedoch von weiteren Programmen aufgerufen werden kann.

INITIAL, falls angegeben, besagt, daß das Programm und all die Programme, die es enthält, bei deren Aufruf in ihrem ursprünglichen Zustand (von der Progr.-Platte) geladen werden.

Die Paragraphen AUTHOR, INSTALLATION, DATE-WRITTEN, SECURITY und REMARKS sind wahlfrei. Der Programmierer kann diese Paragraphen (obwohl sie heute als veraltet gelten) verwenden, um Informationen festzuhalten. Im allgemeinen Format wird durch Kommentar-Eintrag angezeigt, daß der Programmierer jede beliebige Kombination der Zeichen aus dem graphisch dargestellten ASCII-Zeichenvorrat verwenden kann. Der Kommentar-Eintrag kann auf einer oder mehreren Ursprungszeilen enthalten sein.

Der Paragraph DATE-COMPILED ist wahlfrei und eigentlich veraltet. Während jeder Umwandlung wird der Kommentar-Eintrag dieses Paragraphen durch das System-Datum in der Form YY/MM/DD ersetzt.

Das folgende Beispiel zeigt den Inhalt der IDENTIFICATION DIVISION in einem Lohnbuchhaltungsprogramm.

BEISPIEL 1: Die IDENTIFICATION DIVISION kann alle der folgenden Paragraphen enthalten.

```
8
IDENTIFICATION DIVISION.
PROGRAM-ID.   LOHNLISTE01.
AUTHOR. A. VORNDRAN.
INSTALLATION.   ABTEILUNG OST DER FIRMA XYZ.
DATE WRITTEN.    JANUAR 1990.
DATE COMPILED.
SECURITY. KEINE.
REMARKS.
```

BEISPIEL 2: Letztlich benötigt der COBOL-Compiler nur die folgenden Angaben:

```
8
IDENTIFICATION DIVISION.
PROGRAM-ID.   LOHNLISTE01.
```

KAPITEL 6: ENVIRONMENT DIVISION

EINFÜHRUNG

Die ENVIRONMENT DIVISION muß dem COBOL-Compiler als zweite DIVISION zugeführt werden. Ihr Zweck ist es, den Computer, auf dem das COBOL Ursprungs-Programm compiliert werden soll (Source-Computer) und den Computer, auf dem das Programm laufen soll (Objekt-Computer) zu beschreiben. Sie gibt auch Information betreffend der zu verwendenden Peripherie, System-Software-Schalter und Merk-Namen. Die ENVIRONMENT DIVISION besteht aus zwei Kapiteln, der CONFIGURATION SECTION und der INPUT-OUTPUT SECTION.

Die CONFIGURATION SECTION ist in drei Paragraphen unterteilt. Der SOURCE-COMPUTER Paragraph beschreibt den Source-Computer. Der OBJECT-COMPUTER Paragraph beschreibt den Objekt-Computer. Der SPECIAL-NAMES Paragraph ordnet einem System-Software-Schalter einen Bedingungs-Namen zu, ordnet LINE-n einen Merk-Namen zu, teilt ein Währungssymbol außer $ zu, ändert die Funktion von Komma und Punkt bei der Druckaufbereitung von Dezimalstellen und ordnet einem Zeichensatz und/oder einer Sortierfolge einen Alphabet-Namen zu.

Die INPUT-OUTPUT SECTION beinhaltet die Informationen, die die Übertragung von Daten zwischen externen Speichern und dem Objekt-Programm steuern. Dieses Kapitel ist in zwei Paragraphen unterteilt. Der FILE-CONTROL Paragraph benennt die im Programm verwendeten Dateien. Der I-O-CONTROL Paragraph definiert Steuertechniken, die im Objekt-Programm Verwendung finden sollen.

Das folgende allgemeine Format faßt die Kapitel und Paragraphen der ENVIRONMENT DIVISION zusammen und zeigt ihre Reihenfolge im Source-Programm:

```
[ENVIRONMENT DIVISION.
[CONFIGURATION SECTION.
[SOURCE-COMPUTER.          Source-Computer.]
[OBJECT-COMPUTER.          Object-Computer.]
  [SPECIAL-NAMES.          Special-Names-Eintrag.]]
[INPUT-OUTPUT SECTION.
   FILE-CONTROL.           {File-control-Eintrag}...
  [I-O-CONTROL.            [Input-Output-Control-Eintrag]]
```

Die ENVIRONMENT DIVISION beginnt mit einer Division-Überschrift, bestehend aus den reservierten Wörtern ENVIRONMENT DIVISION, auf die ein Punkt und ein Leerzeichen folgen.

DIE CONFIGURATION SECTION

Die CONFIGURATION SECTION, in der der Programmierer den Source-Computer und den Object-Computer beschreibt, muß ggf. unmittelbar anschließend an die Division-Überschrift geschrieben werden. Dieses Kapitel ist in drei Paragraphen unterteilt:

SOURCE-COMPUTER, OBJECT-COMPUTER und SPECIAL-NAMES

Die CONFIGURATION SECTION beginnt mit einer Kapitel-Überschrift, bestehend aus den reservierten Wörtern CONFIGURATION SECTION, auf die ein Punkt und dann ein Leerzeichen folgen muß.

Die CONFIGURATION SECTION ist WEGZULASSEN in einem Programm, das direkt oder indirekt in einem anderen Programm enthalten ist. Die expliziten bzw. impliziten Bedingungen der CONFIGURATION SECTION eines Programmes, welches andere Programme enthält, GELTEN FÜR JEDES ENTHALTENE PROGRAMM!

DER SOURCE-COMPUTER-PARAGRAPH

Der SOURCE-COMPUTER-Paragraph, der den Computer bezeichnet, auf dem das Programm umzuwandeln ist, wird unmittelbar anschließend an die Kapitel-Überschrift geschrieben. Dieser Paragraph beginnt mit einer Paragraphen-Überschrift, bestehend aus dem reservierten Wort SOURCE-COMPUTER, dem ein Punkt und darauf ein Leerzeichen folgt. Im Anschluß an die Paragraphen-Überschrift schreibt der Programmierer den Computer-Namen, der den Source-Computer bezeichnet. Das Format des SOURCE-COMPUTER-Paragraphen:

[SOURCE-COMPUTER. [NCR-ITX [WITH DEBUGGING MODE].]]

- Der Computer-Name ist beliebig (hier z. B. NCR-ITX)
- Der Computer-Name dient nur zur Dokumentation.
- Die Klausel WITH DEBUGGING MODE bewirkt, wenn angegeben, daß alle Debug-Sections und alle Debug-Zeilen (D in Spalte 7) mitcompiliert werden. Wird die Klausel weggelassen, werden alle Debug-Zeilen und -Sections wie Kommentarzeilen behandelt.

DER OBJECT-COMPUTER-PARAGRAPH

Der OBJECT-COMPUTER-Paragraph, der den Computer bezeichnet, auf dem das Programm auszuführen ist, wird unmittelbar anschließend an den SOURCE-COMPUTER-Paragraphen geschrieben. Der Paragraph beginnt mit der Paragraphen-Überschrift, bestehend aus dem reservierten Wort OBJECT-COMPUTER, dem ein Punkt und ein Leerzeichen folgt. Im Anschluß an die Paragraphen-Überschrift schreibt der Programmierer den Computer-Namen. Das Format des OBJECT-COMPUTER-Paragraphen:

```
[OBJECT-COMPUTER. [NCR-ITX [MEMORY SIZE Zahl {WORDS     }]]
                                             {CHARACTERS}
                                             {MODULES   }
[PROGRAM COLLATING SEQUENCE IS Alphabetname]
[SEGMENT-LIMIT IS Segmentnummer].]]
```

Es kann jeder Computername angegeben werden (hier NCR-ITX).

Alle Klauseln des OBJECT-COMPUTER-Paragraphen beziehen sich auf das Programm, in dem sie explizit oder implizit angegeben sind, sowie auf jedes weitere Programm, welches in diesem enthalten ist.

Die Klausel MEMORY SIZE

Die Klausel MEMORY SIZE dient lediglich der Dokumentation.

Die Klausel PROGRAM COLLATING SEQUENCE

Die Klausel PROGRAM COLLATING SEQUENCE dokumentiert, daß die in ihr angegebene Collating Sequence (= Sortierfolge) verwendet wird, um den Wert in einer Verhältnis-Bedingung oder in einem Vergleich mit Bedingungsnamen zu bestimmen.

Ist die Klausel PROGRAM COLLATING SEQUENCE nicht im COBOL Ursprungs-Programm enthalten, so wird vom Compiler die ASCII-Sortierfolge angenommen (American Standard Code for Information Interchange).

Die Klausel gilt auch für die COBOL-Anweisungen Merge oder SORT, es sei denn, es wird dort eine eigene Collating Sequence angegeben.

Die Klausel SEGMENT-LIMIT

Bei Verwendung dieser Klausel wird sie wie ein Kommentar behandelt.

Hinweis: Die Angabe eines der Object-Computer-Namen NCR-DPS, NCR-IMOS, NCR-MCITS oder NCR-TCS bewirkt einen High-Value-Default 128.

Der SPECIAL-NAMES-PARAGRAPH

Der Paragraph SPECIAL-NAMES gibt dem Programmierer die Möglichkeit, einem System-Software-Schalter einen Bedingungs-Namen zuzuordnen und dem Implementor-Namen LINE-n einen Merk-Namen zuzuordnen. Er ermöglicht es dem Programmierer weiterhin, ein anderes Währungszeichen als das Dollarzeichen zu verwenden, Punkt und Komma für die Dezimalstellen, die in der Druckaufbereitung stehen, auszutauschen und einem Zeichenvorrat und/oder einer Sortierfolge einen Alphabet-Namen zuzuordnen. Der Paragraph SPECIAL-NAMES ist ein Wahl-Paragraph. Bei seiner Anwendung muß er direkt dem Paragraphen OBJECT-COMPUTER folgen.

Der Paragraph SPECIAL-NAMES beginnt mit einer Paragraphen-Überschrift, bestehend aus dem reservierten Wort SPECIAL-NAMES, dem ein Punkt und mindestens ein Leerzeichen folgt. Das allgemeine Format des SPECIAL-NAMES-Paragraphen wird nachstehend dargestellt. Weder ein Komma noch ein Semikolon darf unmittelbar der ersten Klausel des SPECIAL-NAMES-Paragraphen vorausgehen.

SPECIAL-NAMES.

```
[ [ [ { PRINTER  } IS mnemonic-name-1 ] ... ]
      { CONSOLE  }
      { LINE-n   }

[ { SWITCH-1 }
  { SWITCH-2 }
  { SWITCH-3 }
  { SWITCH-4 }
  { SWITCH-5 } [ IS mnemonic-name-2 ] { ON STATUS IS condition-name-1                  } ... ]
  { SWITCH-6 }                        { [ OFF STATUS IS condition-name-2 ]             }
  { SWITCH-7 }                        { OFF STATUS IS condition-name-2                 }
  { SWITCH-8 }                        { [ ON STATUS IS condition-name-1 ]              }

[ ALPHABET alphabet-name-1 IS { ASCII                                                  } ... ]
                              { EXTENDED-H-SET                                         }
                              { KATAKANA                                               }
                              { STANDARD-1                                             }
                              { STANDARD-2                                             }
                              { NATIVE                                                 }
                              { EBCDIC                                                 }
                              { literal-1 [ { THROUGH } literal-2 ]                    }
                                          { THRU                                       }
                                          [ ALSO literal-3 ] ... }

[ SYMBOLIC CHARACTERS { IS  } { ASCII-CONTROL [ HEXADECIMAL ] }
                      { ARE } { HEXADECIMAL [ ASCII-CONTROL ] }

  [ SYMBOLIC CHARACTERS { { [ symbolic-character-1 ] ... { IS  } [ integer-1 ] ... } ...
                        {                               { ARE }                   }
    [ IN alphabet-name-2 ] } ] ...

  [ CLASS class-name-1 IS { literal-4 [ { THROUGH } literal-5 ] } ... ] ...
                          {           { THRU                  }       }

  [ CURRENCY SIGN IS literal-6 ]

  [ DECIMAL-POINT IS COMMA ] . ]
```

Zur Syntax:

- Werden SWITCH-1 bis SWITCH-8 definiert, kann ein angegebener Mnemonic-Name-2 nur in einer SET-Anweisung angesprochen werden.

- Wird CONSOLE definiert, kann der angegebene Mnemonic-Name-1 nur in den Anweisungen ACCEPT oder DISPLAY verwendet werden.

- Wird PRINTER definiert, kann der angegebene Mnemonic-Name-1 nur in einer DISPLAY-Anweisung verwendet werden.

- Wird LINE-n verwendet, kann der angegebene Mnemonic-Name-1 nur in der Druck-Variante der WRITE-Anweisung verwendet werden.

- Ist ein in der ALPHABET-Klausel definiertes Literal numerisch, muß es eine Ganzzahl ohne Vorzeichen sein zwischen 1 und 256 (inklusive).

- Ist ein in der ALPHABET-Klausel definiertes Literal nicht numerisch und in Verbindung mit THROUGH oder ALSO aufgeführt, darf es nur aus einem Zeichen bestehen.

- Wird die Literal-Variante der ALPHABET-Klausel verwendet, darf jedes Zeichen nur einmal vorkommen.

- Die Wörter THRU und THROUGH sind gleichbedeutend.

- Derselbe Symbolic-Character-1 darf in einer SYMBOLIC CHARACTERS-Klausel nur einmal vorkommen. Das Verhältnis zwischen jedem Symbolic-Character-1 und dessen korrespondierendem Integer-1 wird durch deren Position in der SYMBOLIC CHARACTER-Klausel bestimmt. Zum ersten Symbolic-Character-1 gehört das erste Integer-1, zum zweiten Symbolic-Character-1 gehört das zweite Integer-1 usw.

- Die Anzahl der angegebenen Symbolic-Character-1 muß identisch sein mit der Anzahl der angegebenen Integer-1.

- Die mit Integer-1 benannte Positions-Nr. muß im Default-Zeichensatz vorkommen. Wurde IN angegeben, muß die Positions-Nr. in demjenigen Zeichensatz vorkommen, der durch den Alphabetnamen 2 bestimmt wird.

- Ist ein unter Literal-4 angegebenes Literal numerisch, muß es ohne Vorzeichen sein und einen Wert zwischen 1 und der maximalen Anzahl Zeichen im Default-Zeichensatz haben.

- Ein unter Literal-4 geschriebenes nicht-numerisches Literal in Verbindung mit der THROUGH-Angabe darf nur ein Zeichen umfassen.

- Literal-6 darf keine figurative Konstante sein.

Regeln:

- Alle unter SPECIAL-NAMES gemachten Angaben beziehen sich auch auf Programme, die in diesem Programm enthalten sind. Auch die vergebenen Bedingungsnamen (Condition-Names) sind von diesen verwendbar.

- Verwenden Sie einen der Schalter SWITCH-1 bis SWITCH-8, kann dem jeweiligen OFF- oder ON-Status des betreffenden Schalters ein Bedingungsname gegeben werden, der - abgefragt - direkt den Status des Schalters vermittelt.

- Das Format 3 der Anweisung SET kann den Status eines der Schalter SWITCH-1 bis SWITCH-8 verändern. Die Anweisung SET muß dabei den Merknamen (Mnemonic-Name) des Schalters ansprechen.

- Mittels ALPHABET-Name ist es möglich, einem Zeichensatz oder einer definierten Zeichenfolge einen Namen zu erteilen. Wird in der PROGRAM COLLATING SEQUENCE-Klausel des OBJECT-COMPUTER-Paragraphen oder als COLLATING SEQUENCE einer Anweisung SORT oder MERGE ein Alphabet-Name-1 genannt, legt man dadurch eine bestimmte Zeichenfolge fest. Wird ein Alphabet-Name-1 bei SYMBOLIC CHARACTERS oder als CODE-SET in einer Dateierläuterung genannt, bestimmt man damit einen der Zeichensätze.

  STANDARD-1 bedeutet: ASCII-Code X3.4 - 1977
  STANDARD-2 bedeutet: ISO-Code 7-Bit

  NATIVE ist der Originalcode der ITX-Anlage: 256 Zeichen, von denen die ersten 128 dem ASCII-Code entsprechen, die weiteren 128 Zeichen haben die Hex-Werte 80 bis FF (Bitmuster 10000000 bis 11111111). Dabei ist demnach der LOW-VALUE gleich Hex 00, der HIGH-VALUE gleich Hex FF.

EBCDIC: ANSI-STANDARD-X3.26-1970-Zeichensatz, bei dem LOW-VALUE Hex 00 und HIGH-VALUE Hex FF ist.

KATAKANA: Der japanische Industrie-Code (JISCII).

Wird die Literal-Klausel verwendet, darf deren vergebener Alphabet-Namen nicht als CODE-Set bei einer Dateierläuterung genannt werden! Eine Zeichenfolge wird bei SPECIAL-NAMES unter Einhaltung der folgenden Regeln definiert: Ist das Literal numerisch, bestimmt sein Wert die Positions-Nr. eines Zeichens innerhalb des Original-Zeichensatzes der ITX-Anlage. Diese darf nicht größer als 256 lauten. Ist das Literal nicht numerisch, stellt es ein aktuelles Zeichen innerhalb des Original-Zeichensatz der ITX-Anlage dar. Besteht ein angegebenes nicht-numerisches Literal aus mehr als einem Zeichen, wird jedem Zeichen, beginnend mit dem linken äußeren Zeichen, eine fortlaufend aufsteigende Position innerhalb der zu bildenden Zeichenfolge zugewiesen.

Die Reihenfolge, in der Literale in der ALPHABET-Klausel geschrieben werden, legt deren Positionsnummern in der zu bildenden, freien Zeichenfolge fest. Dabei erhalten Zeichen aus dem NATIVE-Zeichensatz, die nicht in die ALPHABET-Klausel mit aufgenommen werden, die nicht belegten rechten Positionen. Hier behalten sie gleichzeitig auch die Reihenfolge wie im Originalzeichensatz bei.

THROUGH dient der Übernahme einer geschlossenen Gruppe von Zeichen aus dem NATIVE-Zeichensatz in auf- oder absteigender Folge.

ALSO dient der wertmäßigen Gleichstellung zweier oder mehrerer Zeichen in einer neu zu bildenden Zeichenfolge oder in einem gewählten ASCII-, STANDARD-1-, STANDARD-2-, NATIVE- oder EBCDIC-Zeichensatz.

- Die figurative Konstante HIGH VALUE wird durch dasjenige Zeichen dargestellt, das die höchste Position in der Program Collating Sequence einnimmt, es sei denn, diese figurative Konstante wird unter SPECIAL-NAMES als Literal definiert. Haben mehr als ein Zeichen die höchste Position in der Program Collating Sequence, gilt das letzte angegebene als HIGH VALUE.

- Für die figurative Konstante LOW-VALUE gilt die obige Aussage im umgekehrten Sinn. Bei mehreren Zeichen, zusammengefaßt mit ALSO für die niedrigste Position, gilt das erste als figurative Konstante LOW-VALUE.

- Wird in der SYMBOLIC CHARACTERS-Klausel IN nicht angegeben, muß Symbolic-Character-1 das Zeichen repräsentieren, dessen Positionsnummer im Native-Zeichensatz mittels Integer-1 anzugeben ist. Wird IN angegeben, muß als INTEGER-1 die Positionsnummer des Zeichens in dem Zeichensatz angegeben sein, der unter Alphabet-Name-2 angegeben ist.

- Die SYMBOLIC CHARACTERS-Klausel ist zu verwenden zur Angabe der Zeichen, die in nicht-numerischen Literalen eines COBOL-Ursprungsprogrammes auftreten können.

  Die interne Repräsentation des Symbolic-Character-1 ist die interne Repräsentation des entsprechenden angegebenen Zeichens aus dem Native-Zeichensatz.

  Die ASCII-CONTROL-Variante ist zu verwenden, wenn 2- oder 3-Zeichen-Kombinationen, wie in Abb. 5.1 dargestellt, im Programm gehandhabt werden müssen. Sie sind in den Zeichensätzen STANDARD-1 und NATIVE enthalten. Sie können, mit Ausnahme von DEL, als Steuerungszeichen bei der Datenkommunikation verwendet werden.

  Die HEXADEZIMAL-VARIANTE ist zu verwenden zur Definition hexadezimaler Literale im Format Hxx. Für jedes x kann ein Zeichen zwischen 0 und 9 sowie A und F stehen. Mit H"00" würde das erste Zeichen des Native-Zeichensatzes dargestellt, mit H"FF" das letzte, usw.

  Ist die SYMBOLIC CHARACTERS-Klausel nicht angegeben, müssen in einem nicht-numerischen Literal vorkommende Symbole durch die Positionsnummern der Symbole im Native-Zeichensatz repräsentiert sein (0 - 255).

- Die CLASS-Klausel ist zu verwenden, um einem innerhalb der Klausel definierten Zeichensatz einen Namen zu erteilen. Der Class-Name-1 kann nur in einer Class-Bedingung angesprochen werden. Mit Hilfe der Literale in dieser Klausel wird derjenige exklusive Zeichensatz definiert, für den der Class-Name-1 stehen soll.

  Numerische Literale geben die Positions-Nummern der Zeichen im Native-Zeichensatz wieder.

Nicht-numerische Literale stellen direkt die ausgewählten Zeichen aus dem Native-Zeichensatz dar. Besteht ein nicht-numerisches Literal aus mehreren Zeichen, wird jedes in den exklusiven Zeichensatz - identifiziert durch den Class-Name-1 - aufgenommen.

Mit THROUGH können Gruppen von Zeichen aus dem Native-Zeichensatz übernommen werden, und zwar auf- oder absteigend.

Die Klausel CURRENCY SIGN IS Literal bedeutet, daß das angegebene Literal als Währungszeichen in der PICTURE-Klausel benutzt wird. Das Literal darf nur ein Zeichen sein und darf nicht aus folgenden Zeichen bestehen:

den Ziffern 0 bis 9,
den Buchstaben A, B, C, D, P, R, S, V, X, Z,
den Kleinbuchstaben a bis z,
    Leerzeichen (oder Zwischenraumzeichen)
*   Stern
+   Plus-Zeichen
-   Minus-Zeichen (oder Bindestrich)
,   Komma
(runde Klammer auf
)   runde Klammer zu
"   Anführungszeichen
/   Schrägstrich
=   Gleichheitszeichen

Ist die Klausel CURRENCY SIGN IS Literal nicht vorhanden, kann nur das Dollarzeichen als Währungssymbol in der PICTURE-Klausel benutzt werden.

Die Klausel DECIMAL-POINT

Die Klausel DECIMAL-POINT IS COMMA zeigt an, daß in der Zeichenfolge der PICTURE-Kluasel und in numerischen Literalen die Funktion von Komma und Punkt getauscht wurde. Dadurch übernimmt das Komma die Dezimalpunkt-Anzeige. Wird diese Klausel nicht geschrieben, ist der Punkt allein die Dezimalpunkt-Anzeige.

DIE INPUT-OUTPUT-SECTION

Das Kapitel INPUT-OUTPUT dient der Benennung jeder im Programm verwendeten Datei und liefert andere dateibezogene Informationen an den COBOL-Compiler. Wird es verwendet, so muß es direkt der CONFIGURATION SECTION folgen. Die INPUT-OUTPUT SECTION hat zwei Paragraphen: FILE-CONTROL und I-O-CONTROL.

Die INPUT-OUTPUT SECTION beginnt mit einer Kapitel-Überschrift, bestehend aus den reservierten Wörtern INPUT-OUTPUT SECTION, auf die ein Punkt und ein Leerzeichen folgen.

Der Paragraph FILE-CONTROL

Der Paragraph FILE-CONTROL bezeichnet jede im Programm verwendete Datei und enthält Angaben über Datenträger, Dateistruktur und Zugriff. Verwendet der Programmierer die INPUT-OUTPUT SECTION im Programm, dann muß der FILE-CONTROL-Paragraph mit aufgenommen werden.

Der FILE-CONTROL-Paragraph beginnt mit der Paragraphen-Überschrift, enthaltend die reservierten Wörter FILE-CONTROL, von einem Punkt und einem Leerzeichen gefolgt.

Die Datei-Organisationsformen unter COBOL

Sequentielle Dateien

Dateien, in die Datensätze in sequentieller Folge geschrieben werden (WRITE-Anweisung) und aus denen die Datensätze später auch wieder in derselben Folge sequentiell gelesen werden (READ... AT END ...).

Eine Anmerkung zu den Anweisungen SORT und MERGE: Bei Verwendung der Klauseln USING und GIVING können nur sequentielle Dateien bearbeitet werden.

Relative Dateien

Dateien, deren Sätze mittels relativer Positionsnummern identifiziert werden. Die Datei besteht aus einer seriellen Folge von Bereichen, von denen jeder einen Datensatz aufnehmen kann. Ein Datensatz kann per Direktzugriff geschrieben/gelesen werden, indem dem Computer vor dem Zugriff die Positionsnummer des anzupeilenden Bereiches zur Verfügung gestellt wird, in der der Satz geschrieben werden soll bzw. aus dem er gelesen werden soll. Zum Beispiel ist es möglich, daß in einer relativen Datei in den ersten drei Bereichen keine Sätze enthalten oder dort gewesene Sätze gelöscht sind, während im vierten Bereich ein Datensatz liegt.

Auf relative Dateien kann sequentiell, random oder dynamisch zugegriffen werden.

Beim sequentiellen Zugriff werden die Sätze in aufsteigender Folge der relativen Positionen der augenblicklich in der Datei befindlichen Sätze angesprochen.

Beim Random-Zugriff können Datensätze nach dem Belieben des Programmierers ausgewählt werden. Zur Identifizierung des gewünschten Satzes wird dessen Positionsnummer dem Computer vor dem Dateizugriff in einem sogenannten Relative-Key-Feld zur Verfügung gestellt.

Der dynamische Zugriff gestattet Kombinationen von random und sequentiellem Zugriff.

Index-Dateien

Dateien, in denen sich jeder Satz mittels eines eindeutigen Ordnungsbegriffes identifiziert, z. B. einer Kontonummer, eingetragen samt der Adresse des Satzes in ein stets vom Computer aktuell gehaltenes Verzeichnis (Index), über das der Satz jederzeit wiedergefunden werden kann.

Das Feld eines Datensatzes, dessen Name in der RECORD KEY-Klausel des File-Control-Paragraphen angegeben ist, enthält den primären Ordnungsbegriff zur Identifizierung des Satzes. Kein Satz in der Datei darf den gleichen primären Ordnungsbegriff enthalten wie ein anderer Satz in der Datei. Der primäre Ordnungsbegriff ist relevant bei der Aufnahme eines Satzes in die Datei, beim Auffinden eines Satzes zwecks Durchführung von Änderungen im Satzinhalt und zum Löschen eines Satzes. Das Feld eines Satzes, das den primären Ordnungsbegriff enthält, darf inhaltlich nie verändert werden!

Ein Feld eines Satzes, dessen Name in der ALTERNATE RECORD KEY-Klausel des File-Control-Paragraphen angegeben ist, enthält einen weiteren Ordnungsbegriff zur alternativen Identifizierung des Satzes, falls der primäre Ordnungsbegriff einmal nicht zur Hand ist. Duplikate innerhalb verschiedener Sätze in der Datei sind erlaubt und können mit Hilfe des dynamischen Zugriffs gehandhabt werden.

Auf Index-Dateien kann sequentiell, random oder dynamisch zugegriffen werden.

Beim sequentiellen Zugriff werden die Sätze anhand des Indexes nach der aufsteigenden Folge ihrer primären Ordnungsbegriffe aufgesucht.

Beim Random-Zugriff können Sätze nach dem Belieben des Programmierers ausgewählt werden. Zur Identifizierung eines gewünschten Satzes wird der Wert seines Ordnungsbegriffes dem Computer vor dem Dateizugriff in einem sogenannten Record-Key-Feld zur Verfügung gestellt, anhand dessen er die Adresse des Satzes im abzusuchenden Index ermittelt.

Der dynamische Zugriff gestattet Kombinationen von random und sequentiellem Zugriff.

Der FILE-CONTROL Paragraph

Der FILE-CONTROL Paragraph benennt jede im Programm angesprochene Datei und erlaubt die Angabe weiterer dateibezogener Informationen.

```
FILE-CONTROL. {file-control-entry} ...
SELECT [ OPTIONAL ] file-name-1

                 ┌ DISC          ┐
                 │ DISK          │
                 │ PRINT         │
                 │ PRINTER       │
                 │ CASSETTE      │
ASSIGN TO        │ MAGNETIC-TAPE │
                 │ INPUT         │
                 │ INPUT-OUTPUT  │
                 │ OUTPUT        │
                 │ RANDOM        │
                 │ literal-1     │
                 └ data-name-1   ┘

[ RESERVE integer-1 [ AREA  ] ]
                    [ AREAS ]

[ { ORGANIZATION } IS { [ BINARY ]           SEQUENTIAL } ]
  { ORGANISATION }    { [ LINE   ]                      }
                      {                     RELATIVE    }
                      {                     INDEXED     }

[ PADDING CHARACTER IS { data-name-2 } ]
                       { literal-2   }

[ RECORD DELIMITER IS {STANDARD-1} ]

[ ACCESS MODE IS { SEQUENTIAL } [ RELATIVE KEY IS data-name-4 ] ]
                 { RANDOM     }
                 { DYNAMIC    }

[ RECORD KEY IS data-name-5 ]
[ ALTERNATE RECORD KEY IS data-name-6 [WITH DUPLICATES ] ] ...
[ FILE STATUS IS data-name-7 ].
```

Syntax:

- **Alle hier definierten Dateien müssen unterschiedliche Namen und je eine File-Description-Eintragung in der Data Division haben.**

- **Die Bezeichnungen INPUT, INPUT-OUTPUT, OUTPUT und RANDOM, die Dateizugriffs-Namen Literal-1 und Data-Name-1 sowie das Schlüsselwort BINARY sind RM/COBOL-bezogene Erweiterungen. Mit ihrer Anwendung muß die Compiler-Option RMEXT einhergehen.**

- **Bei SORT- oder MERGE-Arbeitsdateien genügen der Dateiname und die ASSIGN-Klausel.**

Der FILE-CONTROL-Eintrag für eine sequentielle Datei

Der FILE-CONTROL-Eintrag benennt eine sequentielle Datei, auf die durch ein Programm zugegriffen wird. Ferner werden dadurch die Medien bezeichnet, die als Datenträger für die sequentielle Datei dienen. Eine sequentielle Datei kann auf einer Magnetplatteneinheit, einer Magnetbandkassetteneinheit, einer Magnetbandeinheit liegen oder eine Druckliste sein.

Für jede sequentielle Datie, auf die durch ein Programm zugegriffen wird, muß ein FILE-CONTROL Eintrag geschrieben werden. Dieser muß mit einer SELECT-Klausel beginnen, die aus dem reservierten Wort SELECT, gefolgt von einem Datei-Namen, besteht.

Ein FILE-CONTROL Eintrag für eine sequentielle Datei enthält eine Pflichtklausel und Wahlklauseln. Diese Klauseln können in jeder beliebigen Reihenfolge nach dem Datei-Namen, der auf das reservierte Wort SELECT folgt, auftreten. Die letzte Klausel des FILE-CONTROL Eintrags muß mit einem Punkt und einem Leerzeichen abgeschlossen werden.

Die Beifügung OPTIONAL

Sie ist möglich bei Dateien, die im Input-, I-O- oder Extend-Modus eröffnet werden. Sie ist erforderlich bei Dateien, die nicht bei jedem Programmlauf vorhanden sind.

Die ASSIGN-Klausel (Sequentielle Datei)

Die ASSIGN-Klausel bestimmt die Art der peripheren Einheit, die der im jeweiligen FILE-CONTROL Eintrag bezeichneten Datei zugeordnet ist.

Der Implementor-Name CASSETTE bezeichnet eine Datei auf einer Kassette, der Implementor-Name MAGNETIC-TAPE eine solche auf einem Magnetband. In diesem Handbuch ist unter dem Ausdruck "Magnetbanddatei" eine Datei zu verstehen, die entweder auf einer Kassette oder einem Magnetband gespeichert ist.

Literal-1 ist ein Datei-Zugrifsname (File Access Name). Er kann bis zu 30 Stellen lang sein und muß in Anführungszeichen stehen.

Data-Name-1 muß der Name eines in der Data Division definierten alphanumerischen Feldes sein (allerdings nicht in der Linkage Section). Das Feld kann auch qualifiziert werden. Sein Inhalt zum Zeitpunkt der Ausführung einer OPEN-Anweisung wird als Dateizugriffsname verwendet.

Die Klausel ORGANIZATION IS SEQUENTIAL (Sequentielle Datei)

Mit der Klausel ORGANIZATION wird die Organisationsform einer Datei angegeben. Die Organisation einer Datei wird zu dem Zeitpunkt festgelegt, wenn die Datei erstellt wird; sie kann danach nicht mehr geändert werden. Lautet die Klausel ORGANIZATION IS SEQUENTIAL, dann ist die Datei so organisiert, daß die Satzfolge lückenlos ist. Diese Satzfolge wird bestimmt durch die Reihenfolge, in der die Sätze bei der Erstellung der Datei geschrieben werden. Wenn einmal festgelegt, ändert sich die Satzfolge nicht mehr, außer daß Sätze am Ende der Datei angefügt werden. Eine Datei dieser Organisationsform nennt man eine sequentielle Datei. Ist die Klausel ORGANIZATION nicht spezifiziert, so wird vom Compiler die Klausel ORGANIZATION IS SEQUENTIAL angenommen.

Für die Kompatibilität mit RMCOBOL-Programmen akzeptiert der COBOL 85-Compiler zwei Arten sequentiell organisierter Dateien: Line und Binary, ignoriert jedoch die jeweilige Angabe und behandelt sie als Kommentar. Über die Hintergründe gibt das amerikanische Originalmanual nähere Auskunft.

Die Klausel ACCESS MODE (Sequentielle Datei)

Auf eine Datei der sequentiellen Organisationsform kann nur sequentiell Zugriff genommen werden. Durch einen sequentiellen Zugriff werden Sätze aus der Datei gelesen oder in diese geschrieben, und zwar in einer fortlaufenden lückenlosen Folge, die durch die Reihenfolge der Sätze in der Datei bestimmt ist. Ist die Klausel ACCESS MODE nicht spezifiziert, so wird vom Compiler die Klausel ACCESS MODE IS SEQUENTIAL angenommen.

Die Klausel FILE STATUS (Sequentielle Datei)

Bei Angabe der Klausel FILE STATUS wird vom Betriebssystem ein Wert in das unter Data-Name-7 definierte Datenfeld übertragen. Es muß in der DATA DIVISION als ein zwei Zeichen umfassendes alphanumerisches Datenfeld definiert werden. Es darf weder in der FILE SECTION noch in der COMMUNICATION SECTION definiert sein. Es darf qualifiziert werden. Besagter Wert wird während der Ausführung einer OPEN-, CLOSE-, READ-, WRITE- oder REWRITE-Anweisung und vor der Ausführung einer USE-Anweisung in das FILE STATUS-Datenfeld übertragen. Der Wert des FILE STATUS-Datenfeldes zeigt im COBOL-Programm den Status der Ein-/Ausgabe-Operation an.

Die Klausel FILE STATUS sollte verwendet werden, wenn der Programmierer im Programm für die Datei DECLARATIVE-Prozeduren vorsieht.

Eine Übersicht über die möglichen Statuswerte finden Sie im Anhang.

Die Klausel PADDING CHARACTER (Sequentielle Datei)

Diese Klausel ist unwirksam und wird vom Compiler wie ein Kommentar behandelt. Aus Gründen der Kompatibilität mit anderen Compilern wurde diese Klausel mit aufgenommen. Weitergehende Auskunft gibt das amerikanische Originalmanual.

Die Klausel RECORD DELIMITER IS STANDARD-1
(Sequentielle Datei)

Sie dient der Angabe, wie die Länge von Sätzen bei einer Datei mit variabler Satzlänge auf Magnetband festgestellt werden kann. Wird diese Klausel weggelassen, ist die Länge eines Satzes dem 2 Bytes langen Satz-Header entnehmbar.

Wird die Klausel angegeben, besagt das, daß zur Feststellung der Länge eines Satzes die Methode dient, welche im ANS-Standard X3. 27-1978 und im Internationalen Standard 1001 1979 festgelegt wurde.

Die Klausel RESERVE (Sequentielle Datei)

Sie wird aus Kompatibilitätsgründen noch geführt, jedoch wie ein Kommentar behandelt.

Der FILE-CONTROL-Eintrag für eine relative Datei

Der FILE-CONTROL Eintrag benennt eine relative Datei, auf die durch ein Programm zugegriffen wird. Eine relative Datei kann nur auf einer Magnetplatte gespeichert sein.

Für jede relative Datei, auf die in einem Programm zugegriffen wird, muß ein FILE-CONTROL Eintrag geschrieben werden. Dieser muß mit der SELECT-Klausel beginnen, die aus dem reservierten Wort SELECT, gefolgt von einem Datei-Namen, besteht.

Ein FILE-CONTROL Eintrag für eine relative Datei enthält zwei Pflichtklauseln und Wahlklauseln. Diese Klauseln können in jeder beliebigen Reihenfolge nach dem Datei-Namen, der auf das reservierte Wort SELECT folgt, auftreten. Die letzte Klausel des FILE-CONTROL Eintrages muß mit einem Punkt und einem Leerzeichen abgeschlossen werden.

Die ASSIGN-Klausel (Relative Datei)

Die ASSIGN-Klausel bestimmt die Art der peripheren Einheit, die der im jeweiligen FILE-CONTROL Eintrag bezeichneten Datei zugeordnet ist. Für eine relative Datei ist eine Magnetplatte notwendig. In NCR COBOL wird die relative Datei durch die Klausel ASSIGN TO DISC einer Magnetplatte zugeordnet. (Es kann auch DISK geschrieben werden.)

Die Klausel ORGANIZATION IS RELATIVE (Relative Datei)

Mit der Klausel ORGANIZATION wird die Organisationsform einer Datei angegeben. Die Organisation einer Datei wird zu dem Zeitpunkt festgelegt, wenn die Datei erstellt wird; sie kann danach nicht mehr geändert werden. Die Klausel ORGANIZATION IS RELATIVE ist im FILE-CONTROL Eintrag für eine relative Datei notwendig. Eine relative Datei ist so organisiert, daß jedem Datensatz eine bestimmte Position in der Datei zugeordnet ist. Eine relative Datensatznummer bezeichnet die Position eines Datensatzes. Dem ersten Satz entspricht die Nummer 1.

Die Klausel ACCESS MODE (Relative Datei)

Die Klausel ACCESS MODE sagt aus, ob eine Datei sequentiell oder im Direktzugriff verarbeitet wird. Auf eine Datei der relativen Organisationsform erfolgt der Zugriff entweder sequentiell oder direkt oder dynamisch.

Erscheint die Klausel ACCESS MODE IS SEQUENTIAL im FILE-CONTROL Eintrag, dann wird auf die relative Datei sequentiell zugegriffen. Sequentieller Zugriff auf eine relative Datei ist dadurch gekennzeichnet, daß dabei vom Dateianfang an Satz für Satz in aufsteigender Reihenfolge gemäß den Satznummern verarbeitet wird. Ist die Klausel RELATIVE KEY spezifiziert, so enthält der Data-Name-4 immer aufs Neue die relative Satznummer des Datensatzes, auf den durch die vorangehende READ- oder WRITE-Anweisung zugegriffen wurde.

Erscheint die Klausel ACCESS MODE IS RANDOM im FILE-CONTROL Eintrag, dann erfolgt der Zugriff auf die relative Datei direkt. Direkter Zugriff auf eine relative Datei ist gekennzeichnet durch Zugriff auf den Datensatz gemäß dem Inhalt des Data-Name-4 in der Klausel RELATIVE KEY.

Erscheint die Klausel ACCESS MODE IS DYNAMIC im FILE-CONTROL Eintrag, dann erfolgt der Zugriff auf eine relative Datei in ein und demselben Programm sequentiell und/oder direkt.

Data-Name-4 in der Klausel RELATIVE KEY muß als ein Ganzzahlfeld ohne Vorzeichen außerhalb der Satzdefinition in der DATA DIVISION definiert sein. Data-Name-4 darf qualifiziert werden. Er wird das relative Satzschlüsselfeld genannt.

Ist die Klausel ACCESS MODE nicht angegeben, so gilt ACCESS MODE IS SEQUENTIAL.

Die Klausel FILE STATUS (Relative Datei)

Bei Eingabe der Klausel FILE STATUS wird vom Betriebssystem ein Wert in das unter Data-Name-7 definierte Datenfeld übertragen. Es muß in der DATA DIVISION als ein zwei Zeichen umfassendes alphanumerisches Datenfeld definiert werden. Es darf qualifiziert werden. Besagter Wert wird während der Ausführung einer OPEN-, CLOSE-, UNLOCK-, READ-, WRITE-, REWRITE-, DELETE- oder START-Anweisung und vor der Ausführrung einer USE-Anweisung in das FILE STATUS-Datenfeld übertragen. Der Wert des FILE STATUS-Datenfeldes zeigt im COBOL-Programm den Status der Ein-/Ausgabe-Operation an.

Die Klausel FILE STATUS sollte verwendet werden, wenn der Programmierer im Programm für die Datei DECLARATIVE-Prozeduren vorsieht.

Eine Übersicht über die möglichen Statuswerte finden Sie im Anhang.

Die Klausel RESERVE (Relative Datei)

Sie wird aus Kompatibilitätsgründen noch geführt, jedoch wie ein Kommentar behandelt.

Der FILE-CONTROL-Eintrag für eine indexierte Datei

Der FILE-CONTROL Eintrag beschreibt eine indexierte Datei, auf die durch ein Programm zugegriffen wird. Eine indexierte Datei kann nur auf einer Magnetplatte gespeichert sein.

Für jede indexierte Datei, auf die durch ein Programm zugegriffen wird, muß ein FILE-CONTROL Eintrag geschrieben werden. Dieser muß mit der SELECT-Klausel beginnen, die aus dem reservierten Wort SELECT, gefolgt von einem Datei-Namen, besteht.

Ein FILE-CONTROL Eintrag für eine indexierte Datei enthält drei Pflichtklauseln und Wahlklauseln. Diese Klauseln können in jeder beliebigen Reihenfolge nach dem Datei-Namen in der SELECT-Klausel auftreten. Die letzte Klausel des FILE-CONTROL Eintrags muß mit einem Punkt und einem Leerzeichen abgeschlossen werden.

Die ASSIGN-Klausel (Indexierte Datei)

Die ASSIGN-Klausel bestimmt die Art der peripheren Einheit, die der im jeweiligen FILE-CONTROL Eintrag bezeichneten Datei zugeordnet ist. Für eine indexierte Datei ist eine Magnetplatte notwendig. In NCR COBOL wird die indexierte Datei durch die Klausel ASSIGN TO DISC einer Magnetplatte zugeordnet (es kann auch DISK geschrieben werden).

Die Klausel ORGANIZATION IS INDEXED (Indexierte Datei)

Mit der Klausel ORGANIZATION wird die Organisationsform einer Datei angegeben. Die Organisation einer Datei wird zu dem Zeitpunkt festgelegt, wenn die Datei erstellt wird; sie kann danach nicht mehr geändert werden. Die Klausel ORGANIZATION IS INDEXED ist im FILE-CONTROL Eintrag für eine indexierte Datei notwendig.

Die Klausel ACCESS MODE (Indexierte Datei)

Die Klausel ACCESS MODE besagt, ob eine Datei sequentiell oder im Direktzugriff verarbeitet wird. Auf eine als Index-Datei organisierte Datei erfolgt der Zugriff entweder sequentiell oder direkt oder dynamisch.

Erscheint die Klausel ACCESS MODE IS SEQUENTIAL im FILE-CONTROL Eintrag, dann wird die indexierte Datei sequentiell bearbeitet. Sequentieller Zugriff auf eine indexierte Datei ist dadurch gekennzeichnet, daß dabei nach der aufsteigenden Folge der Satzschlüssel im Index zugegriffen wird.

Erscheint die Klausel ACCESS MODE IS RANDOM im FILE-CONTROL Eintrag, dann erfolgt der Zugriff auf die indexierte Datei direkt. Direktzugriff auf eine indexierte Datei ist gekennzeichnet durch Zugriff auf den Datensatz gemäß dem Inhalt des Data-Name-5 in der Klausel RECORD KEY.

Erscheint die Klausel ACCESS MODE IS DYNAMIC im FILE-CONTROL Eintrag, dann erfolgt der Zugriff auf die indexierte Datei in ein und demselben Programm sequentiell und/oder direkt.

Ist die Klausel ACCESS MODE nicht angegeben, so gilt ACCESS MODE IS SEQUENTIAL.

Die Klausel RECORD KEY (Indexierte Datei)

Die Klausel RECORD KEY bezeichnet ein Datenfeld (genannt Satzschlüssel) jedes Datensatzes in der indexierten Datei. Ein Index der Datensatzschlüssel, der für jeden Datensatz auch dessen Adresse in der Datei enthält, schafft eine logische Verbindung zu jedem Datensatz der indexierten Datei. Der Wert eines jeden primären Satzschlüsseln in der indexierten Datei muß eindeutig sein und darf vom Programmierer während der Verarbeitung des Datensatzes nicht verändert werden.

Data-Name-5 in der Klausel RECORD KEY muß als ein alphanumerisches Datenfeld innerhalb einer Satzbeschreibung der entsprechenden Datei definiert sein. Es darf qualifiziert werden.

Die Klausel ALTERNATE RECORD KEY (Indexierte Datei)

Die Klausel ALTERNATE RECORD KEY bezeichnet ein Datenfeld (genannt alternativer Satzschlüssel) jedes Datensatzes in der Datei. Über einen alternativen Ordnungsbegriff als Satzschlüssel besteht die Möglichkeit, unter Umgehung eines primären Satzschlüssels einen alternativen Zugriff auf Sätze in einer Indexdatei durchzuführen.

Der Data-Name-6 kann qualifiziert werden.

Das mittels des Data-Name-6 bezeichnete Feld muß innerhalb einer Satzbeschreibung der entsprechenden Indexdatei alphanumerisch und mit fixer Länge definiert sein. Das Data-Name-6-Feld darf auch nicht auf derselben Satzposition wie das Data-Name-5-Feld (primärer Ordnungs-Begriff) oder ein anderes Data-Name-6-Feld beginnen.

Ein Primärschlüssel und bis zu 15 Alternativschlüssel können für eine Indexdatei definiert werden.

Mittels der Wahlklausel WITH DUPLICATES kann ausdrücklich gestattet werden, daß mehrere Sätze in der Datei ein und denselben Inhalt im entsprechenden Alternate-Key-Feld aufweisen dürfen.

Die Klausel FILE STATUS (Indexierte Datei)

Bei Angabe der Klausel FILE STATUS wird vom Betriebssystem ein Wert in das unter Data-Name-7 definierte Datenfeld übertragen. Es muß in der DATA DIVISION als ein zwei Zeichen umfassendes alphanumerisches Datenfeld definiert werden. Es darf qualifiziert werden. Besagter Wert wird während der Ausführung einer OPEN-, CLOSE-, UNLOCK-, READ-, WRITE-, REWRITE-, DELETE- oder START-Anweisung und vor der Ausführung einer USE-Anweisung in das FILE STATUS-Datenfeld übertragen. Der Wert des FILE STATUS-Datenfeldes zeigt im COBOL-Programm den Status der Ein-/Ausgabe-Operation an.

Die Klausel FILE STATUS sollte verwendet werden, wenn der Programmierer im Programm für die Datei DECLARATIVE-Prozeduren vorsieht.

Eine Übersicht über die möglichen Statuswerte finden Sie im Anhang.

Die Klausel RESERVE (indexierte Datei)

Sie wird aus Kompatibilitätsgründen noch geführt, jedoch wie ein Kommentar behandelt.

Der FILE-CONTROL-Eintrag für eine Sortier- oder Mischdatei (Arbeitsdatei)

Eine Sortierdatei ist eine Ansammlung von Sätzen, die von einer SORT-Anweisung innerhalb eines COBOL-Programmes zu sortieren sind. Durch die SORT-Anweisung wird eine Sortierdatei erstellt und benutzt.

Eine Mischdatei ist eine Ansammlung von Sätzen, die von einer MERGE-Anweisung innerhalb eines COBOL-Programmes zu sortieren sind. Durch die MERGE-Anweisung wird eine Mischdatei erstellt und benutzt.

Jede Sortier- oder Mischdatei innerhalb eines Programmes muß nur ein einziges Mal in einem FILE CONTROL-Eintrag definiert sein.

Für jede Sortier- oder Mischdatei, auf die durch ein Programm zugegriffen wird, muß ein FILE CONTROL-Eintrag geschrieben werden. Dieser muß mit der SELECT-Klausel beginnen, die aus dem reservierten Wort SELECT, gefolgt von einem Datei-Namen, besteht.

Ein FILE CONTROL-Eintrag für eine Sortier- oder Mischdatei enthält nur eine Pflichtklausel, und zwar die ASSIGN-Klausel. Der ASSIGN-Klausel des FILE CONTROL-Eintrags für eine Sortier- oder Mischdatei muß ein Punkt und ein Leerzeichen folgen.

Die ASSIGN-Klausel (Sortier- oder Mischdatei)

Die ASSIGN-Klausel bestimmt die Art der peripheren Einheit, die der im jeweiligen FILE CONTROL-Eintrag bezeichneten Datei zugeordnet ist. In NCR COBOL 85 wird eine Sortier- oder Mischdatei durch die Klausel ASSIGN TO DISC einer Magnetplatte zugeordnet.

Der I-O-CONTROL Paragraph

Der I-O-CONTROL Paragraph kennzeichnet die Stellen, an denen bei Bedarf das Objekt-Programm wieder anlaufen kann. In diesem Paragraphen werden auch der Speicherbereich, den sich verschiedene Dateien teilen, sowie die Position von Magnetband-Dateien auf einer Mehrdateien-Spule festgelegt. Dieser Paragraph kann wahlweise benutzt werden. Falls angewendet, muß er mit der Paragraphen-Überschrift beginnen, die den Paragraphen-Namen I-O-CONTROL, gefolgt von einem Punkt und einem Leerzeichen, enthält. Der Paragraph selbst enthält einen Satz mit der RERUN-Klausel, der SAME AREA Klausel und/oder der MULTIPLE FILE TAPE-Klausel. Die Klauseln können in beliebiger Reihenfolge geschrieben werden.

Das allgemeine Format für den I-O-CONTROL Paragraphen in COBOL 85:

```
[I-O-CONTROL.
 [RERUN [ON file-name-1]
  EVERY { [ { [END OF] { REEL  } } OF filename-2 ] ...
        {   { integer-1 RECORDS }
        {   integer-2 CLOCK-UNITS
        {   condition-name-1
  [ SAME { RECORD } AREA FOR file-name-3 [file-name-4] ... ] ...
         { SORT   }
         { SORT-MERGE }
 [ SAME [ RECORD ] AREA FOR file-name-3 [file-name-4] ... ] ...
 [ MULTIPLE FILE TAPE CONTAINS { file-name-5 [ POSITION integer-3 ] } ... ] ... .]
```

Die RERUN-Klausel
(Wird vom ITX COBOL 85-Compiler akzeptiert, aber zur Zeit nur dokumentiert!)

Diese Klausel ist an sich für folgendes gedacht: Wenn das Objekt-Programm große Datenmengen verarbeitet, kann es wünschenswert sein, die Stellen zu benennen, an denen im Falle einer Störung oder eines Bedienungsfehlers während des Programmablaufs ein Wiederanlauf durchgeführt wird. Diese Wiederanlaufpunkte nennt man Wiederholpunkte. Der Punkt, an dem der Status des Computer-Systems bewahrt wird, muß z. B. durch die Verarbeitung von Integer-1 Datensätzen des File-Name-2 bestimmt werden.

ON File-Name 1 besagt, daß die Wiederanlauf-Information (Speicherabzug genannt) auf eine Magnetplatten- oder Magnetbanddatei (genannt Speicherabzugsdatei) geschrieben wird. Die Klausel RERUN ON File-Name-1 in einem Programm macht die Speicherabzugsdatei für das Programm verfügbar, das Speicherabzüge erfordert. Der Programmierer definiert die Speicherabzugsdatei nicht und darf darauf mit COBOL Ein-/Ausgabe-Anweisungen keinen Zugriff nehmen. Die Speicherabzugsdatei steht vollständig unter der Steuerung des Betriebssystems.

Tritt während der Programmausführung eine Störung oder ein Bedienungsfehler auf, so bewerkstelligt die Wiederanlauf-Software den Wiederanlauf eines Programms vorzugsweise vom Wiederholpunkt und nicht vom Programmbeginn an.

Die Klausel SAME AREA

Die Klausel SAME AREA bedeutet, daß zwei oder mehrere Dateien im Laufe ihrer Verarbeitung den gleichen Speicherbereich benutzen. Der Bereich, den sich diese Dateien teilen, enthält alle Speicher-Bereiche, die den benannten Dateien zugeordnet sind. Die Dateien dürfen nicht zur gleichen Zeit eröffnet sein. Jede im Objekt-Programm benutzte Datei muß abgeschlossen werden, bevor eine andere in der Klausel SAME AREA genannte Datei eröffnet werden darf. Es können mehrere SAME AREA Klauseln im Programm auftreten; ein Datei-Name darf jedoch nicht in mehr als einer dieser Klauseln erscheinen. Die in der SAME AREA-Klausel benannten Dateien müssen nicht alle von derselben Organisationsform sein und der Zugriff darauf muß nicht in der gleichen Weise erfolgen.

Die Klausel SAME RECORD AREA

Die Klausel SAME RECORD AREA bestimmt, daß zwei oder mehr Dateien den gleichen Speicherbereich zur Bearbeitung des laufenden Satzes benutzen. Alle Dateien können dabei gleichzeitig eröffnet sein. Die in der SAME RECORD AREA-Klausel benannten Dateien müssen nicht alle von derselben Organisationsform sein und der Zugriff darauf muß nicht in der gleichen Weise erfolgen.

Ein Satz in demselben Datensatz-Bereich wird betrachtet als ein Satz jeder eröffneten Ausgabedatei, deren Datei-Name in der SAME RECORD AREA-Klausel erscheint. Der Satz in demselben Datensatz-Bereich wird auch angesehen als ein Satz der zuletzt gelesenen Eingabedatei, deren Datei-Name in der SAME RECORD AREA-Klausel erscheint. Die SAME RECORD AREA-Klausel verursacht direkte Redefinitionen desselben Datensatz-Bereiches für jede in der Klausel genannte Datei.

Erscheinen in einer SAME RECORD AREA Klausel ein oder mehrere Datei-Namen einer SAME AREA Klausel, so brauchen nicht alle in letzterer enthaltenen Datei-Namen in der SAME RECORD AREA Klausel zu erscheinen. Zusätzliche Datei-Namen, die nicht in dieser SAME AREA Klausel stehen, können aber in besagter SAME RECORD AREA Klausel erscheinen. Die Regel, daß nur eine von den in einer SAME AREA-Klausel genannten Dateien zu jeder gegebenen Zeit eröffnet sein kann, steht über der Regel, nach der alle Dateien, die in einer SAME RECORD AREA Klausel genannt werden, zu jeder gegebenen Zeit eröffnet sein können.

Die Klausel SAME SORT AREA oder SAME SORT MERGE AREA

Die Klausel SAME SORT AREA (oder die Klausel SAME SORT MERGE AREA) bedeutet, daß zwei oder mehrere Dateien im Laufe ihrer Verarbeitung den gleichen Speicherbereich benutzen. Der Bereich, den sich diese Dateien teilen, enthält alle Speicher-Bereiche, die den benannten Dateien zugeordnet sind. In dieser Klausel sind die Wörter SORT und SORT-MERGE äquivalent. Somit wird fortan nur noch auf die Klausel SAME SORT AREA Bezug genommen.

Zumindest einer der Datei-Namen in der Klausel SAME SORT AREA muß eine Sortier- oder eine Mischdatei benennen. Dateien, welche keine Sortier- oder Mischdateien sind, dürfen in der Klausel SAME SORT AREA aber auch genannt werden. Die Dateien, auf die in der Klausel SAME SORT AREA Bezug genommen wird, brauchen nicht alle dieselbe Organisation oder Zugriffsart aufzuweisen.

Jede SORT-Anweisung in einem Programm umfaßt nur eine Sortierdatei. Auch jede MERGE-Anweisung in einem Programm betrifft nur eine Mischdatei. Somit ist zu jeder Zeit nur eine Sortier- oder Mischdatei zur Verarbeitung eröffnet. Die Klausel SAME SORT AREA ermöglicht es zwei oder mehreren Sortier- oder Mischdateien, während der Verarbeitung denselben Speicherbereich einzunehmen. Nachdem eine SORT- oder MERGE-Anweisung durchgeführt wurde, wird der der jeweiligen Sortier- oder Mischdatei zugeordnete Speicherbereich zur Verarbeitung einer anderen, in einer neuen SORT- oder MERGE-Anweisung definierten Sortier- oder Mischdatei verfügbar.

Der Speicherbereich, der einer Nonsort-Datei, also keiner Sortier- oder Mischdatei zugeordnet ist, kann auch von einer Sortier- oder Mischdatei benötigt werden. Die Klausel SAME SORT AREA würde in diesem Fall die Datei-Namen der den gleichen Speicherbereich benutzenden Sortier- oder Mischdatei und der Nonsort-Datei enthalten. Während der Ausführung einer SORT- oder MERGE-Anweisung, deren Sortier- oder Mischdatei in der Klausel SAME SORT AREA definiert ist, darf die Nonsort-Datei nicht eröffnet sein. Die in der SAME AREA-Klausel definierten Nonsort-Dateien benutzen nicht denselben Speicherbereich; damit dies geschieht, müssen in der SAME AREA-Klausel die Nonsort-Dateien angegeben sein.

In einem Programm können mehrere SAME SORT AREA-Klauseln geschrieben werden. Ein Sortier- oder Mischdatei-Name darf jedoch nur in einer SAME SORT AREA-Klausel vorkommen. Erscheint der Datei-Name einer Nonsort-Datei in einer SAME AREA-Klausel und einer oder mehreren SAME SORT AREA-Klauseln, so müssen alle in der SAME AREA-Klausel genannten Dateien in der SAME SORT AREA-Klausel oder den SAME AREA-Klauseln vorkommen.

Die Klausel MULTIPLE FILE TAPE

Die Klausel MULTIPLE FILE TAPE besagt, daß zwei oder mehr Dateien dieselbe Spule eines Magnetbandes gemeinsam benutzen. Unabhängig von der Anzahl der Dateien, die sich auf einer einzelnen Spule befinden können, brauchen nur diejenigen Dateien in der MULTIPLE FILE TAPE-Klausel genannt werden, die im Objekt-Programm benutzt werden.

Die POSITION-Angabe gibt die Position der Datei vom Anfang der Bandspule an. Die erste Datei auf dem Band wird durch die Positionsnummer 1 gekennzeichnet. Die zweite Datei auf dem Band hat die Positionsnummer 2, usw.

Sind alle Namen der auf Magnetband befindlichen Dateien in der MULTIPLI FILE TAPE-Klausel in aufeinanderfolgender Reihenfolge aufgeführt, so braucht die POSITION-Angabe nicht definiert zu sein. Sollte in der Aufzählung aber eine Datei nicht erwähnt sein, so müssen die POSITION-Angaben definiert werden. In einer Spule können beispielsweise ABCFILE, GOODFILE, BESTFILE und XYZFILE in der genannten Reihenfolge aufgeführt sein. Wenn im Programm nur ABCFILE und BESTFILE benutzt werden, hat die MULTIPLE FILE TAPE-Klausel folgendes Format:

```
I-O-CONTROL.
    MULTIPLE FILE TAPE CONTAINS ABCFILE POSITION 1,
        BESTFILE POSITION 3.
```

Werden in einem Programm alle vier Dateien auf ein und derselben Spule angesprochen, dann hat die MULTIPLE FILE TAPE-Klausel folgendes Format:

```
I-O-CONTROL.
    MULTIPLE FILE TAPE CONTAINS ABCFILE, GOODFILE, BESTFILE,
        XYZFILE.
```

Von mehreren Dateien auf einer Bandspule kann immer nur eine eröffnet sein. Sie muß abgeschlossen werden, bevor eine andere eröffnet werden kann.

ENVIRONMENT DIVISION-BEISPIEL

Es folgt ein Beispiel der IDENTIFICATION DIVISION und der ENVIRONMENT DIVISION.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. FLUG-VERARB.
SECURITY.
        DIESES PROGRAMM SOLL AM 5. UND 20. JEDEN MONATS
        LAUFEN. UM DIE FLUGSTUNDEN-DATEI ABZUARBEITEN MUSS
        SWITCH 5 AUF ON GESETZT WERDEN.


ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. NCR-ITX.
OBJECT-COMPUTER. NCR-ITX, PROGRAM COLLATING SEQUENCE
        IS ASCII-CODE-SET.
SPECIAL-NAMES.
    SWITCH-5 ON STATUS IS FLUG-DAT-VERARB,
    ASCII-CODE-SET IS STANDARD-I.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT PLATZ-DATEI ASSIGN TO CASSETTE.
    SELECT FLUG-DATEI ASSIGN TO MAGNETIC-TAPE.
    SELECT STRECKEN-DATEI ASSIGN TO DISC
        ORGANIZATION IS INDEXED
        ACCESS MODE IS RANDOM
        RECORD KEY IS STRECKEN-NR.
    SELECT DRUCK-DATEI ASSIGN TO PRINTER.
```

In diesem Beispiel bezeichnet die CONFIGURATION SECTION den Source-Computer als einen NCR-Computer, der mit ITX arbeitet. Der Object-Computer ist ebenfalls als ein Computer, der mit ITX arbeitet, bezeichnet. Der Zeichenfolge für dieses Programm ist der Alphabet-Name ASCII-CODE-SET zugeteilt.

Im SPECIAL-NAMES-Paragraphen ist der Bedingungs-Name FLUG-DAT-VERARB dem eingeschalteten Zustand des System Software-Schalters Nr. 5 zugeordnet.

Die nächste Zeile dokumentiert, daß der Alphabet-Name ASCII-CODE-SET dem Standardzeichensatz STANDARD-1 zugeordnet ist.

In der INPUT-OUTPUT SECTION werden durch die FILE-CONTROL-Einträge den Dateien periphere Geräte zugeteilt, und zwar:

PLATZ-DATEI wird eine Magnetbandkassetteneinheit zugeteilt;

FLUG-DATEI wird eine Magnetbandeinheit zugeteilt;

STRECKEN-DATEI wird eine Magnetplatteneinheit zugeteilt;

DRUCK-DATEI wird ein Zeilendrucker zugeteilt.

Alle Dateien im Programm FLUG-VERARB haben eine sequentielle Organisationsform, außer der STRECKEN-DATEI, die indexiert organisiert ist.

Auf alle Dateien im Programm FLUG-VERARB wird sequentiell Zugriff genommen, außer auf die STRECKEN-DATEI, auf die Direktzugriff erfolgt. In STRECKEN-DATEI stellt das Datenfeld STRECKEN-NR den Satzschlüssel dar, der die Position jedes Datensatzes innerhalb STRECKEN-DATEI bestimmt (indirekt: über den Index!).

KAPITEL 7: DATA DIVISION

EINLEITUNG

Die DATA DIVISION beschreibt die Daten, die durch das Objekt-Programm verarbeitet werden sollen. Diese Daten fallen in drei Kategorien, nämlich

- in Dateien enthaltene Daten: diese Form von Daten erreicht oder verläßt den internen Speicher von einem bestimmten Medium oder bestimmten Medien aus;

- intern errechnete oder gespeicherte Daten: dieser Datentyp wird im internen Speicher gespeichert.

- durch den Programmierer definierte Konstanten.

Die DATA DIVISION, die als 3. DIVISION in den Compiler eingegeben wird, besteht aus fünf Kapiteln (SECTIONS): der FILE SECTION, der WORKING-STORAGE SECTION, der LINKAGE SECTION, der COMMUNICATION SECTION und der GLOBAL SECTION. Jedes dieser Kapitel ist wahlweise zu schreiben und kann im Programm weggelassen werden, wenn es darin nicht erforderlich ist.

Die FILE SECTION beschreibt u. a. die Daten, die die Eingabe oder Ausgabe des internen Speichers von bzw. zu einem externen Medium (einer Datei) darstellen. Eine Reihe Klauseln, genannt Dateibeschreibung, legen den physischen Aufbau der Datei fest. Eine Reihe Klauseln, genannt Satzbeschreibung, beschreiben den Satzaufbau der Datei. Zwei Beschreibungsformen sind möglich: FD (File-Description), SD (Sort-Merge-File-Description).

Die WORKING-STORAGE SECTION beschreibt die Arbeitsbereiche zur Speicherung von Zwischenergebnissen und anderen, zeitlich begrenzt zu speichernden Daten während des Ablaufs des Objekt-Programms. Diese SECTION beschreibt auch Datenfelder (genannt Konstanten), denen vom Programmierer Werte zugeteilt werden. Die Datenfelder in der WORKING-STORAGE SECTION werden entweder durch eine Bereichbeschreibung oder eine Feldbeschreibung beschrieben.

Die LINKAGE SECTION beschreibt Daten, die von mehreren Programmen abwechselnd bearbeitet werden können. Die LINKAGE SECTION kann lediglich in einem mittels CALL... USING... aufgerufenen Programm stehen. Der Aufbau der LINKAGE SECTION ist derselbe wie der der WORKING-STORAGE SECTION.

Die COMMUNICATION SECTION dient zur Beschreibung des Datenbereichs, der als Schleuse zwischen dem Daten-Übertragungssystem MCS (Message Control System) und dem Programm dient.

Die GLOBAL SECTION beschreibt Datenbereiche und Datenfelder, auf die alle Programme, die im Speicher sind, zugreifen können. Die Datensätze und Datenfelder in der GLOBAL SECTION werden nicht repliziert, wenn die Run Unit gebildet wird. Der Aufbau der GLOBAL SECTION ist derselbe wie der der WORKING-STORAGE SECTION. Die Inhalte der Felder können während des Programmablaufs nicht verändert werden.

Das folgende allgemeine Format zeigt die SECTIONS mit Inhalt in der DATA DIVISION sowie ihre Reihenfolge.

```
DATA DIVISION.

[ FILE SECTION.
  [ file-description-entry {record-description-entry} ... ] ...
  [ sort-file-description-entry {record-description-entry} ... ] ... ]

[ WORKING-STORAGE SECTION.
  [ 77-level-description-entry ]
  [ record-description-entry ] ... ]

[ LINKAGE SECTION.
  [ 77-level-description-entry ]
  [ record-description-entry ] ... ]

[ COMMUNICATION SECTION.
  [communication-description-entry [record-description-entry] ... ] ... ]

[ GLOBAL SECTION.
  [ 77-level-description-entry ]
  [ record-description-entry ] ... ]
```

DIE FILE SECTION

Die FILE SECTION, die den Aufbau der Dateien und ihrer individuellen Datensätze beschreibt, muß vorhanden sein, wenn ein Programm von einem externen Medium, wie Magnetplatten, Magnetbänder oder Drucker usw. Gebrauch macht. Falls verwendet, muß die FILE SECTION unmittelbar auf die Division-Überschrift folgen.

Die FILE SECTION beginnt mit einer Section-Überschrift, die aus den reservierten Wörtern FILE SECTION besteht. Nach der Section-Überschrift formuliert der Programmierer eine Dateibeschreibung und dann eine Satzbeschreibung für jeden Satztyp in der Datei. Für jede Datei, auf die durch das Programm Zugriff genommen wird, ist eine Dateibeschreibung erforderlich, auf die eine oder mehrere Satzbeschreibungen folgen.

Über die Anwender-Dateien hinaus werden in der FILE SECTION auch die SORT- und MERGE-Arbeitsdateien beschrieben.

DIE DATEIBESCHREIBUNG (FILE DESCRIPTION)

Der Zweck der Dateibeschreibung ist es, den physischen Aufbau einer Datei zu beschreiben. Für jede in einem Programm verwendete Datei ist diese Beschreibung notwendig. Die Dateibeschreibung besteht aus der Stufenanzeige (FD), einem Datei-Namen und einer Reihe unabhängiger Klauseln, die durch einen Punkt beendet werden.

Hier das allgemeine Format einer Dateibeschreibung:

```
FD file-name-1
[ IS EXTERNAL ]
[ IS GLOBAL ]
[ BLOCK CONTAINS [ integer-1 TO ] integer-2 { RECORDS / CHARACTERS } ]
[ RECORD { CONTAINS integer-3 CHARACTERS
         / IS VARYING IN SIZE [ [ FROM integer-4 ] [ TO integer-5 ] CHARACTERS ]
           [ DEPENDING ON data-name-1 ]
         / CONTAINS integer-6 TO integer-7 CHARACTERS } ]
[ LABEL { RECORD IS / RECORDS are } { STANDARD / OMITTED } ]
[ VALUE OF { [ FILE-ID IS { data-name-2 / literal-1 } ]
           / LABEL IS literal-1 } ... ]
[ DATA { RECORD IS / RECORDS ARE } {data-name-3} ... ]
[ LINAGE IS { data-name-4 / integer-8 } LINES [ WITH FOOTING AT { data-name-5 / integer-9 } ]
  [ LINES AT TOP { data-name-6 / integer-10 } ] [ LINES AT BOTTOM { data-name-7 / integer-11 } ] ]
[ CODE-SET IS alphabet-name-1 ].
```

Die Stufenanzeige FD bezeichnet den Beginn einer Dateibeschreibung und muß in Spalte 8, 9, 10 oder 11 beginnend geschrieben werden.

Der Datei-Name folgt auf die Stufenanzeige FD und beginnt in Spalte 12 oder in irgendeiner rechts davon liegenden Spalte. Zumindest ein Leerzeichen muß die Stufenbezeichnung FD und den Datei-Namen voneinander trennen. Jeder Datei-Name im Programm muß eindeutig sein. Der Datei-Name in der Dateibeschreibung muß identisch sein mit einem der in den FILE-CONTROL-Einträgen der ENVIRONMENT DIVISION aufgeführten Datei-Namen.

Die auf den Datei-Namen folgenden Klauseln können in jeder beliebigen Reihenfolge auftreten. Sie können auf der gleichen Codierzeile wie die Stufenanzeige FD und der Datei-Name oder auf der nächsten Codierzeile geschrieben werden, wobei die erste Klausel frühestens ab Zeichenposition 12 beginnen darf. Die Dateibeschreibung wird durch einen Punkt nach der letzten Klausel beendet.

Eine genaue Beschreibung der Klauseln innerhalb der Dateibeschreibung ist auf den folgenden Seiten in alphabetischer Reihenfolge gegeben (BLOCK CONTAINS bis VALUE OF).

Die SORT-/MERGE-ARBEITSDATEI-BESCHREIBUNG (SD)

Sie dient der Beschreibung einer Arbeitsdatei, innerhalb derer die COBOL-Anweisungen SORT oder MERGE Datensätze sortieren oder mischen sollen.

Im Vergleich zur FD-Klausel entfallen bei der SD-Klausel eine Reihe von Einträgen.

Das Format der SD-Klausel:

```
SD file-name-1

[ RECORD { CONTAINS integer-1 CHARACTERS                                          }
         { IS VARYING IN SIZE [ [ FROM integer-2 ] [ TO integer-3 ] CHARACTERS ]  }
         {   [ DEPENDING ON data-name-1 ]                                         }
         { CONTAINS integer-4 TO integer-5 CHARACTERS                             } ]

[ DATA { RECORD IS  } {data-name-2} ... ] .
       { RECORDS ARE }
```

Die Einträge RECORD und DATA können weggelassen werden.

Eine oder mehrere Satzbeschreibungen müssen auf die SD-Klausel erfolgen. Jedoch dürfen auf diese Datei keine Ein- oder Ausgabe-Befehle angewendet werden. Dies erfolgt implizit in den Anweisungen SORT oder MERGE.

Im FILE-CONTROL-Paragraphen der ENVIRONMENT DIVISION sind für derartige Dateien lediglich die SELECT- und die ASSIGN-Klausel erlaubt.

Die BLOCK CONTAINS-Klausel

Die BLOCK CONTAINS-Klausel benennt die Anzahl Datensätze oder die Anzahl Zeichen, die einen Datenblock auf einem externen Datenträger bilden. Das Format der BLOCK CONTAINS-Klausel ist:

```
BLOCK CONTAINS [integer-1 TO] integer-2 {RECORDS   }
                                        {CHARACTERS}
```

Beim ITX-Computer können lediglich Dateien auf Magnetbändern und Magnetplatten mehr als einen Datensatz pro Block aufweisen. Drucker-Dateien weisen lediglich einen Datensatz pro Block auf.

Die BLOCK CONTAINS-Klausel ist wahlweise. Wird diese Klausel weggelassen, nimmt der Compiler lediglich einen Datensatz pro Block an.

Bei der Angabe des Wortes RECORDS wird die Blocklänge durch Multiplizieren der Anzahl der Datensätze mit der Anzahl Bytes in der zugeordneten Satzbeschreibung bestimmt. In der Klausel BLOCK CONTAINS 51 RECORDS wird die Blocklänge durch Multiplizieren von 51 mit der Satzlänge errechnet. Somit ist bei einer Satzlänge von 10 Bytes die Blocklänge 510 Bytes.

Bei der Angabe des Wortes CHARACTERS wird die Blocklänge als eine Anzahl von 8-Bit-Zeichen innerhalb des Blocks ausgedrückt. Jedes 8-Bit-Zeichen entspricht einem Byte.

Für eine Datei mit fester Satzlänge wird vom Programmierer nur eine Zahl, die Zahl 2, geschrieben, um die genaue Länge jedes Blocks in der Datei darzustellen. Zum Beispiel: BLOCK CONTAINS 15 RECORDS.

Für Dateien mit variabler Satzlänge benutzt der Programmierer die Zahl-1, um die kleinste, und die Zahl-2, um die größte Blocklänge anzugeben. Zum Beispiel: BLOCK CONTAINS 10 TO 25 RECORDS. Wird die Zahl-1 nicht für eine Datei mit

variabler Länge angegeben, dann gibt die Zahl-2 die größte Blocklänge der Datei an. Bei Angabe des Wortes CHARACTERS für eine Datei mit variabler Länge beinhaltet die Blocklänge auch den variablen Längenindikator (VLI) für jeden Datensatz (2 Bytes).

Beim ITX-Computer beträgt die maximale Blocklänge für Magnetplatten- und Magnetband-Dateien 8192 Bytes, für eine Kassetten-Datei 2048 Bytes und für eine Drucker-Datei 506 Bytes.

Die Klausel CODE-SET

Die CODE-SET-Klausel bezeichnet den zur Darstellung von Daten auf externen Datenträgern verwendeten Zeichenvorrat. Die CODE-SET-Klausel darf nicht für eine Magnetplatten-Datei verwendet werden. Das Format der CODE-SET-Klausel ist:

CODE-SET IS Alphabet-Name-1

Der Alphabet-Name innerhalb der CODE-SET-Klausel muß durch das Vorhandensein der Klausel Alphabet-Name im SPECIAL-NAMES-Paragraphen der ENVIRONMENT DIVISION mit einem Zeichenvorrat zusammenhängen. Die dort verwendete Klausel Alphabet-Name darf jedoch nicht die Literal-Angabe enthalten.

Der Alphabet-Name in der CODE-SET-Klausel legt den Algorithmus fest für die Übersetzung des Zeichencodes auf dem externen Datenträger in den oder vom Native-Zeichenvorrat. Diese Code-Übersetzung geschieht während der Ausführung einer Ein- oder Ausgabeoperation. Alle Datenfelder in der umzuwandelnden bzw. zu übersetzenden Datei müssen explizit oder implizit mit der Klausel USAGE IS DISPLAY beschrieben sein und numerische Datenfelder mit Vorzeichen müssen mit der SIGN-Klausel beschrieben sein, die SEPARATE CHARACTER enthält. Daher sind die zulässigen Datentypen alphanumerisch, dezimal ohne Vorzeichen, ungepackt dezimal mit Vorzeichen, führend dezimal mit Vorzeichen.

Ist die CODE-SET-Klausel für eine Magnetband-Datei nicht angegeben, dann wird der Native-Zeichenvorrat angenommen.

Die DATA RECORDS-Klausel

Die DATA RECORDS-Klausel kann vom Programmierer zur Dokumentation der Namen der Datensätze in der Datei verwendet werden. Hier das allgemeine Format der DATA RECORDS Klausel:

```
DATA  { RECORD  IS }   Identifier-1   [Identifier-2] ...
      { RECORDS ARE }
```

Die EXTERNAL-Klausel

Mit ihr wird festgelegt, daß die Datei oder Datenfelder davon als extern gelten. Das heißt, daß Felder und Feldgruppen eines Datensatzes jedem Programm innerhalb einer Run-Unit, das diesen Satz ebenfalls beschreibt, zur Verfügung stehen.

Format:

```
IS EXTERNAL
```

Syntax: Wenn innerhalb einer Run-Unit zwei oder mehr Programme einen gleichen EXTERNAL-Datensatz beschreiben, muß der Satzname in allen Programmen identisch sein. Auch der Satzaufbau muß identisch sein. Abweichungen vom Satzaufbau sind allerdings per REDEFINITION möglich, die von Programm zu Programm verschieden sein kann. EXTERNAL schließt GLOBAL nicht ein. EXTERNAL darf nur in der FD-Klausel der File Section oder in der Working-Storage Section für Satzbeschreibungen verwendet werden.

Die GLOBAL-Klausel

Mit ihr wird festgelegt, daß der Dateiname oder ein Datenfeldname Globalnamen sind. Ein Globalname kann aus jedem Programm angesprochen werden, welches in demjenigen Programm enthalten ist, das den Globalnamen definiert.

Format:

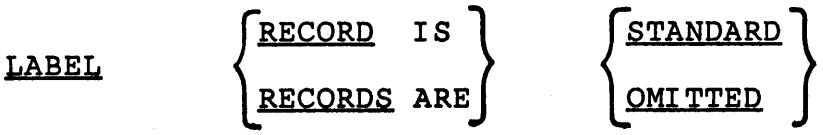
```
IS GLOBAL
```

Syntax: Gleichnamige Felder innerhalb einer Data Division dürfen nicht global sein. Gegebenenfalls sind sie mit unterschiedlichem Namen zu führen. Für Dateien und ihre Satzdefinitionen, welche der Klausel SAME RECORD AREA unterzogen werden, ist die GLOBAL-Klausel nicht statthaft. Wird bei einem Dateinamen GLOBAL angegeben, gelten alle Felder in der Satzdefinition der Datei als global. In Programmen, die ihrerseits direkt oder indirekt in Programmen enthalten sind, welche Global-Datenbereiche enthalten, kann unmittelbar auf diese zugegriffen werden. Das heißt, Globalbereiche brauchen in enthaltenen Programmen nicht gegendefiniert zu werden. Bei redefinierten Bereichen besitzt nur der Stammbereich das Global-Attribut.

Die LABEL RECORDS-Klausel

Die LABEL RECORDS-Klausel gibt Aufschluß, ob zur Datei gehörende Kennsätze vorhanden sind.

Format:



Die Klausel LABEL RECORDS ARE OMITTED sagt aus, daß für die Datei oder das Gerät, dem die Datei zugeordnet ist, keine Kennsätze vorhanden sind. Für Drucker-Dateien bestehen keine Kennsätze. Folglich wird die Klausel LABEL RECORDS ARE OMITTED in die Dateibeschreibung für Drucker-Dateien aufgenommen.

Die Klausel LABEL RECORDS ARE STANDARD besagt, daß für die Datei oder das Gerät, dem die Datei zugeordnet ist, Kennsätze bestehen. Bei ITX-Computern bestehen Kennsätze für Magnetplatten-Dateien. Folglich wird die Klausel LABEL RECORDS ARE STANDARD in die Dateibeschreibung für die Magnetplatten-Datei aufgenommen.

Für Magnetband-Dateien sind Kennsätze wahlweise. Die Klausel LABEL RECORDS ARE STANDARD wird in die Dateibeschreibung einer Kennsätze enthaltenden Magnetband-Datei aufgenommen. Die Klausel LABEL RECORDS ARE OMITTED wird in der Dateibeschreibung einer keine Kennsätze aufweisenden Magnetband-Datei vorgesehen. Wenn eine Magnetband-Datei keine Kennsätze aufweist, dann gilt der erste Satz in der Datei als ein Datensatz.

Wird die Klausel weggelassen, nimmt das System von sich aus LABEL RECORDS ARE STANDARD an.

Die LINAGE-Klausel

Die LINAGE-Klausel bezeichnet die Größe einer durch WRITE-Anweisungen zu druckenden logischen Seite.

Format:

```
LINAGE IS {data-name-1} LINES [WITH FOOTING AT {data-name-2}]
          {integer-1}         {integer-2}

[LINES AT TOP {data-name-3}]
              {integer-3}

[LINES AT BOTTOM {data-name-4}]
                 {integer-4}
```

Syntax: Die Datennamen 1 bis 4 müssen numerisch ganzzahlige Elementarfelder ohne Komma repräsentieren. Integer-1 muß größer als 1 sein. Integer-2 darf nicht größer als Integer-1 sein. Integer-3 und Integer-4 dürfen 0 sein.

Die Größenangaben umfassen:

1. Den Bereich, in dem Zeilen in der logischen Seite bedruckt werden können und/oder ein Zeilentransport erfolgen kann; dieser Bereich wird der Seitenrumpf genannt.

2. Die Anzahl leerer Zeilen auf dem Seitenkopf der logischen Seite.

3. Die Anzahl leerer Zeilen auf dem Seitenfuß der logischen Seite.

4. Die logische Zeilennummer im Seitenrumpf, bei der der Fußzeilen-Bereich beginnt. Die erste Zeile des Seitenrumpfes hat die logische Zeilennummer 1. Der Fußzeilen-Bereich endet mit der letzten Zeile des Seitenrumpfes.

Die Größe einer logischen Seite wird bestimmt durch die Summe der Anzahl leerer Zeilen im Seitenrumpf, im Seitenkopf und im Seitenfuß. Die Größe der logischen Seite braucht nicht notwendigerweise auf die Größe einer physischen Seite bezogen zu sein. Jede logische Seite schließt ohne Leerzeilen an die nächste an.

Der Bereich, in dem Zeilen in der logischen Seite bedruckt und/oder transportiert werden können (Seitenrumpf), ist spezifiziert durch integer-1 oder data-name-1. Der Wert des integer-1 oder der Daten, auf die durch den data-name-1 Bezug genommen wird, muß größer als Null sein. Der Wert des integer-1 oder der Daten, auf die durch den data-name-1 Bezug genommen wird, muß kleiner als 256 sein. Der data-name-1 muß sich auf ein elementares Datenfeld mit numerischen ganzen Zahlen ohne Vorzeichen beziehen.

Die Angabe LINES AT TOP

Die Angabe LINES AT TOP mit integer-3 oder data-name-3 bestimmt die Anzahl leerer Zeilen am Seitenkopf der logischen Seite. Im Seitenkopf der logischen Seite können Zeilen weder bedruckt werden noch kann ein Zeilentransport erfolgen. Der Wert von integer-3 oder der Daten, auf die durch den data-name-3 Bezug genommen wird, kann gleich Null sein. Der data-name-3 muß sich auf ein elementares Datenfeld mit numerischen ganzen Zahlen ohne Vorzeichen beziehen. Ist die Angabe LINES AT TOP nicht aufgeführt, dann gilt die Anzahl der Zeilen am Seitenkopf als gleich Null.

Die Angabe LINES AT BOTTOM

Die Angabe LINES AT BOTTOM mit integer-4 oder dem data-name-4 bestimmt die Anzahl leerer Zeilen am Seitenfuß der logischen Seite. Am Seitenfuß der logischen Seite können leere Zeilen weder bedruckt werden noch kann ein Zeilentransport erfolgen. Der Wert von integer-4 oder der Daten, auf die durch den data-name-4 Bezug genommen wird, kann gleich Null sein. Der data-name-4 muß sich auf ein elementares Datenfeld mit numerischen ganzen Zahlen ohne Vorzeichen beziehen. Ist die Angabe LINES AT BOTTOM nicht aufgeführt, dann gilt die Anzahl leerer Zeilen am Seitenfuß als gleich Null.

FOOTING-Angabe

Die FOOTING-Angabe mit integer-2 oder dem data-name-2 bestimmt die logische Zeilennummer im Seitenrumpf, bei der der Fußzeilen-Bereich beginnt. Die erste Zeile des Seitenrumpfes hat die logische Zeilennummer 1. Die letzte Zeile des Fußzeilen-Bereiches ist dieselbe wie die letzte Zeile des Seitenrumpfes. Der data-name-2 muß sich auf ein elementares Datenfeld mit numerischen ganzen Zahlen ohne Vorzeichen beziehen. Der Wert von integer-2 oder der Daten, auf die durch den data-name-2 Bezug genommen wird, muß größer als Null sein. Der Wert von integer-2 oder der Daten, auf die durch den data-name-2 Bezug genommen wird, darf nicht größer sein als der Wert von integer-1 oder des data-name-1. Folglich kann, wenn der Seitenrumpf 50 Zeilen umfaßt, der Fußzeilen-Bereich nicht bei der logischen Zeilennummer 51 beginnen, sondern er muß bei der logischen Zeilennummer 50, 49, 48 oder einer anderen logischen Zeilennummer, jedenfalls mit einem Wert unter 51, beginnen. Der Fußzeilen-Bereich ist mit Daten, wie Endsummen und Zwischensummen, bedruckbar.

Ohne die FOOTING-Angabe beginnt und endet der Fußzeilen-Bereich auf der letzten Zeile des Seitenrumpfes. Die letzte Zeile des Seitenrumpfes wird von integer-1 oder data-name-1 festgelegt.

Weitere Ausführungen zur Linage-Klausel finden Sie bei der Beschreibung der Anweisungen OPEN und WRITE.

Das Register LINAGE-COUNTER

Beim Vorhandensein einer LINAGE-Klausel in einer Dateibeschreibung stellt die Software dem Programm ein Register, genannt LINAGE-COUNTER, zur Verfügung. Der LINAGE-COUNTER stellt quasi ein Datenfeld mit einer Länge von 2 Bytes dar. Der Inhalt im LINAGE-COUNTER stellt immer die aktuelle logische Zeilennummer dar, auf der der Drucker im Seitenrumpf positioniert ist.

Der LINAGE-COUNTER kann vom Programmierer durch Prozedur-Anweisungen, wie beispielsweise die IF-Anweisung, angesprochen werden. Der Inhalt des LINAGE-COUNTER darf jedoch nicht verändert werden.

Für jede Datei, deren Dateibeschreibung eine LINAGE-Klausel enthält, existiert ein eigener LINAGE-COUNTER. Ist in einem Programm mehr als ein LINAGE-COUNTER vorhanden, dann muß der Programmierer LINAGE-COUNTER mit einem Datei-Namen qualifizieren, wenn er diesen anspricht:

Besteht ein LINAGE-COUNTER z. B. für die OUTFILE und für die REPORTFILE, dann muß bei allen den LINAGE-COUNTER einer dieser Dateien ansprechenden Prozeduren die Bezeichnung LINAGE-COUNTER in OUTFILE oder LINAGE-COUNTER OF REPORTFILE verwendet werden.

Der Inhalt des LINAGE-COUNTER wird während der Ausführung einer OPEN-Anweisung automatisch gesetzt und durch die Ausführung einer WRITE-Anweisung für die zugehörige Datei automatisch verändert. Kurz zusammengefaßt kann die Manipulation des LINAGE-COUNTER folgendermaßen erfolgen:

1. Der LINAGE-COUNTER wird durch die Ausführung einer OPEN OUTPUT-Anweisung automatisch auf den Inhalt 1 gestellt.

2. Der LINAGE-COUNTER wird durch die Ausführung einer die ADVANCING-PAGE Angabe enthaltenden WRITE-Anweisung automatisch auf den Inhalt 1 zurückgestellt.

3. Der LINAGE-COUNTER wird während der Ausführung einer die Angabe ADVANCING data-name/integer enthaltenden WRITE-Anweisung erhöht. Der Inhalt des data-name oder der Wert des integers in der ADVANCING-Klausel bestimmt den Erhöhungsfaktor des LINAGE-COUNTER.

4. Der LINAGE-COUNTER wird automatisch auf den Inhalt 1 zurückgestellt, wenn der Drucker für jede folgende logische Seite auf die erste Zeile des Seitenrumpfes kommt.

5. Der LINAGE-COUNTER wird um 1 erhöht, wenn keine ADVANCING-Klausel in einer WRITE-Anweisung enthalten ist.

Die RECORD-Klausel

Die RECORD-Klausel spezifiziert die Länge der Datensätze in der Datei. Da die Länge jedes Datensatzes in der Satzbeschreibung vollkommen definiert ist, ist die Klausel wahlfrei. Die durch diese Klausel definierte Satzlänge entspricht der Anzahl von Bytes im Datensatz. Zwei Varianten der RECORD-Klausel können in der Dateibeschreibung verwendet werden. Die erste Variante ist die RECORD CONTAINS-Klausel. Die zweite ist die RECORD IS VARYING-Klausel.

Die RECORD CONTAINS-Klausel

Die RECORD CONTAINS-Klausel dient zur Angabe der Länge von Datensätzen fester oder variabler Länge in der Datei.

Format:

RECORD CONTAINS integer-1 [TO integer-2] CHARACTERS

Die hier definierte Datensatzlänge entspricht der Anzahl Bytes im Datensatz.

Bei einer aus Datensätzen fixer Länge bestehenden Datei schreibt der Programmierer zur Darstellung der genauen Länge jedes Datensatzes in der Datei lediglich eine Zahl. Zum Beispiel sagt RECORD CONTAINS 25 CHARACTERS aus, daß jeder Datensatz in der Datei fünfundzwanzig Bytes umfaßt.

Besteht eine Magnetplatten- oder Magnetband-Datei aus Datensätzen variabler Länge, dann verwendet der Programmierer integer-1 zur Darstellung der kleinsten Länge eines Datensatzes und integer-2 zur Darstellung der größten Länge eines solchen, z. B. RECORD CONTAINS 5 TO 20 CHARACTERS. Die in der RECORD CONTAINS-Klausel spezifizierte Länge beinhaltet nicht den Längen-Indikator für variable Länge (VLI). Der COBOL-Compiler fügt am Anfang eines Datensatzes variabler Länge automatisch zwei Bytes für den VLI hinzu.

Die maximale Datensatzlänge einer Drucker-Datei ist 132 Bytes, die einer Magnetband- oder Magnetplatten-Datei ist 8192 Bytes und die einer Kassetten-Datei ist 2048 Bytes.

Die RECORD IS VARYING-Klausel

Die RECORD IS VARYING-Klausel dient ebenfalls zur Angabe des Datenumfangs in einem Datensatz variabler Länge in der Datei.

Format:

```
RECORD IS VARYING IN SIZE [[FROM integer-1] [TO integer-2]
    CHARACTERS] [DEPENDING ON data-name-1]
```

Besteht eine Magnetplatten- oder Magnetband-Datei aus Datensätzen variabler Länge, dann verwendet der Programmierer zur Darstellung der kleinsten Länge eines Datensatzes das integer-1 und zur Darstellung der größten Länge das integer-2. Die anzugebende Länge schließt den variablen Längenindikator (VLI) nicht mit ein. Der COBOL-Compiler fügt am Anfang eines Datensatzes variabler Länge automatisch zwei Bytes für den VLI hinzu.

Die DEPENDING ON-Angabe bezeichnet ein Datenfeld, dessen Inhalt die Anzahl Bytes (minus der für den LVI) eines Datensatzes anzeigt, auf den durch eine READ- oder RETURN-Anweisung zugegriffen wurde, oder der durch eine WRITE-, REWRITE- oder RELEASE-Anweisung zu schreiben ist. Der data-name-1 in der DEPENDING ON-Angabe muß ein ganzzahliges Elementarfeld ohne Vorzeichen in der WORKING-STORAGE SECTION oder der LINKAGE SECTION sein. Vor der Ausführung einer WRITE-, REWRITE- oder RELEASE-Anweisung muß der Programmierer in das data-name-1-Datenfeld die Anzahl der zu schreibenden Bytes (ohne VLI) einsetzen. Nach der Ausführung einer READ-Anweisung enthält das data-name-1-Datenfeld die Anzahl Bytes (ohne VLI) in dem Datensatz, der durch die READ-Anweisung gelesen wird.

Ist die DEPENDING ON-Angabe nicht vorhanden, so wird bei einer WRITE- oder REWRITE-Anweisung die Anzahl Bytes geschrieben, die durch die entsprechende Satzdefinition gegeben ist.

Die VALUE OF-Klausel

Sie dient zur Bezeichnung eines Feldes im Kennsatz einer Datei oder zur Angabe einer Datei-Bezeichnung.

Format:

```
                       ┌                                          ┐
                       │  ┌                 ┌data-name-2┐ ┐        │
                       │  │FILE-ID IS      <              │        │
     VALUE OF         <   └                 └literal-1  ┘ ┘        >
                       │  [LABEL IS literal-2]                     │
                       └                                          ┘
```

Syntax: Data-Name-2 muß in der WORKING-STORAGE-SECTION definiert sein. Er kann qualifiziert werden. Mit FILE-ID wird der Name der Datei angegeben, die bei Ausführung der OPEN-Anweisung gemeint ist. VALUE OF LABEL wird vom Compiler wie ein Kommentar betrachtet. VALUE OF LABEL darf bei einer Datei mit LABEL RECORDS OMITTED nicht angegeben werden.

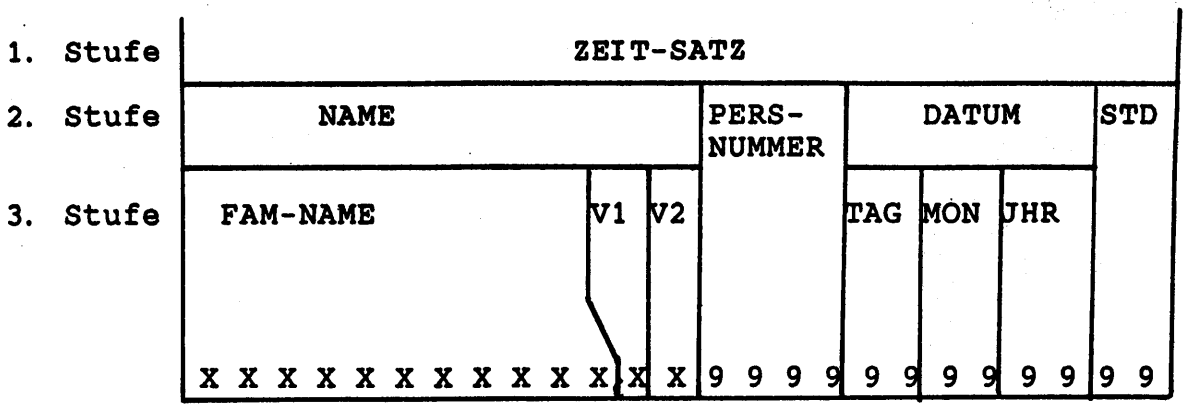
DIE SATZBESCHREIBUNG

Eine Satzbeschreibung stellt eine Reihe von Eintragungen für einen bestimmten Datensatz dar. Jede Beschreibung besteht aus einer Stufennummer, die von einem Identifier und einer Reihe von unabhängigen Klauseln gefolgt wird.

Stufennummern zeigen die Unterteilung eines Datensatzes auf. Nicht weiter unterteilte Bereiche eines Datensatzes werden Elementarfelder genannt. Zur Bezugnahme auf einen Satz von Elementarfeldern werden diese in einer Datengruppe zusammengefaßt.

Ein System von Stufennummern zeigt die Organisation von Datenelementen und Datengruppen. Da Datensätze alle Datenfelder beinhalten, beginnen die Stufennummern für diese bei 1 oder 01; Datenfelder haben höhere Stufennummern. Der Wert der Stufennummern braucht nicht fortlaufend zu sein; der höchste Wert, der zugeteilt werden darf, ist 49. Die folgende Abbildung zeigt den Aufbau eines Datensatzes.

Man sieht, daß Stufennummern 1- oder 2stellig geschrieben werden, ferner, daß bei der Zuteilung der Stufennummer an die nächste Unterteilung diese nicht nur udm 1 erhöht zu werden braucht. Folglich können die Stufennummern 01, 02, 03 oder 01, 03, 05 oder 01, 04, 09 sein. Das einzige Kriterium dabei ist, daß die zugeteilte Stufennummer von höherem numerischen Wert als die vorangehende sein muß.



BEISPIEL 1

```
01  ZEIT-SATZ
    02  NAME
        03  FAM-NAME
        03  V1
        03  V2
    02  PERS-NUMMER
    02  DATUM
        03  TAG
        03  MONAT
        03  JAHR
    02  STD
```

BEISPIEL 2

```
1   ZEIT-SATZ
    3   NAME
        5   FAM-NAME
        5   V1
        5   V2
    3   PERS-NUMMER
    3   DATUM
        5   TAG
        5   MONAT
        5   JAHR
    3   STD
```

Das folgende Beispiel zeigt eine falsche Zuteilung von Stufennummern:

```
01  ZEIT-SATZ
    03  NAME
        05  FAM-NAME
        05  V1
        05  V1
    04  PERS-NUMMER
    04  DATUM
        05  TAG
        05  MONAT
        05  JAHR
    03  STD
```

Falsche Stufennummern
für PERS-NUMMER
und DATUM; 04 sollte 03 sein

Eine Gruppe enthält alle darunter beschriebenen Datengruppen und Elementarfelder, bis eine Stufennummer von einem numerisch geringeren Wert oder eine solche, die gleich dem Wert der Stufennummer dieser Gruppe ist, auftritt. Demnach ist im obigen Beispiel STUNDEN kein Teil der DATUM genannten Gruppe. TAG, MONAT und JAHR sind jedoch ein Teil der DATUM genannten Gruppe, denn sie sind unmittelbar unter DATUM beschrieben und haben eine höhere Stufennummer als DATUM.

In der obigen Darstellung sind FAM-NAME, V1, V2, PERS-NUMMER, TAG, MONAT, JAHR und STD Elementarfelder, denn keines dieser Felder hat weitere Unterteilungen. NAME und DATUM sind Beispiele von Gruppen; ZEIT-SATZ ist im Beispiel der Bereich, der alle Datenfelder beinhaltet, d. h. der Datensatz.

DIE DATENFELDBESCHREIBUNG

Diese liefert Informationen über die Charakteristika eines bestimmten Datenelementes. Diese Eintragung besteht aus einer Stufennummer, einem Identifier oder dem reservierten Wort FILLER und einer Reihe von unabhängigen Klauseln. Das allgemeine Format der Datenfeldbeschreibung ist dieses:

```
level-number [ data-name-1 ]
             [ FILLER      ]  [ REDEFINES data-name-2 ] [ IS EXTERNAL ]
  [ IS GLOBAL ] [ { PIC     } IS character-string ]
                { PICTURE }

[ [ USAGE IS ] { BIT                 } ]
               { BINARY              }
               { COMPUTATIONAL       }
               { COMP                }
               { COMPUTATIONAL-n     }
               { COMP-n              }
               { DISPLAY             }
               { INDEX               }
               { PACKED-DECIMAL      }

[ [ SIGN IS ] { LEADING  } [ SEPARATE CHARACTER ] ]
              { TRAILING }

[ OCCURS { integer-1 TO integer-2 TIMES DEPENDING ON data-name-3 } ]
         { integer-2 TIMES                                      }

[ { ASCENDING  } KEY IS {data-name-4} ... ] ...
  { DESCENDING }
[ INDEXED BY {index-name-1} ... ]

[ { SYNCHRONIZED } [ LEFT  ] ] [ { JUSTIFIED } RIGHT ]
  { SYNC         } [ RIGHT ]   { JUST      }
[BLANK WHEN ZERO]

[ VALUE IS literal-1
  { VALUE IS  } { literal-2 { THROUGH } literal-3 } ... ].
  { VALUES ARE }            { THRU    }
```

Syntax: Die VALUE IS- bzw. VALUES ARE-Angabe ist in der FILE SECTION und in der LINKAGE SECTION nicht erlaubt! EXTERNAL-Angaben sind nur bei der Stufe 01 in der WORKING-STORAGE-SECTION erlaubt. EXTERNAL und REDEFINES sind in derselben Datendefinition nicht erlaubt. GLOBAL ist nur in Datendefinitionen mit Stufen-Nr. 01 erlaubt. THROUGH und THRU sind äquivalent.

Die Stufennummer in einer Datenfeldbeschreibung kann jede 1- oder 2stellige Zahl von 01 - 49 oder 77 sein. Die auf den Data-Name-1 oder auf das Wort FILLER folgenden Klauseln können in jeder beliebigen Reihenfolge auftreten, jedoch mit der Ausnahme, daß, wenn die REDEFINES-Klausel verwendet wird, diese unmittelbar auf den Data-Name-1 folgen muß. Die Datenfeldbeschreibung muß durch einen Punkt nach der letzten Klausel beendet werden.

Die PICTURE-Klausel muß für jedes Elementarfeld in einem Datensatz aufgeführt werden, außer für ein Index-Datenfeld. Die PICTURE-Klausel, die SYNCHRONIZED-Klausel, die JUSTIFIED-Klausel und die BLANK WHEN ZERO-Klausel dürfen für eine ganze Gruppe von Feldern nicht verwendet werden. Die SYNCHRONIZED-Klausel, die JUSTIFIED-Klausel und die BLANK WHEN ZERO-Klausel können für ein Elementarfeld verwendet werden.

Auf den folgenden Seiten wird in alphabetischer Reihenfolge eine genauere Beschreibung der Klauseln gegeben.

DIE BLANK WHEN ZERO-Klausel

Die BLANK WHEN ZERO-Klausel hat eine Aufbereitungsfunktion. Durch sie wird das Datenfeld mit Leerzeichen aufgefüllt, wenn sein Inhalt gleich 0 ist.

Format:

BLANK WHEN ZERO

Der Programmierer benutzt diese Klausel lediglich bei numerischen oder numerisch aufbereiteten Elementarfeldern.

Wenn die BLANK WHEN ZERO-Klausel in der Beschreibung eines Datenfeldes einer numerischen Kategorie auftritt, dann gilt dieses Datenfeld als numerisch aufbereitet.

Die Klausel BLANK WHEN ZERO darf in derselben Beschreibung nicht mit einer PICTURE-Zeichenfolge, die den Schutzstern als ein Nullenunterdrückungszeichen enthält, auftreten.

Die folgenden Beispiele zeigen, wie in einem Datenfeld Daten durch die Klausel BLANK WHEN ZERO beeinflußt werden.

Sendefeld:	PICTURE-Klausel:	Empfangsfeld:
00000	99,999 BLANK WHEN ZERO	leer
00010	99,999 BLANK WHEN ZERO	00,010
00000	99,999	00,000
00000	ZZ,ZZZ	leer
00000	$99,999 BLANK WHEN ZERO	leer
00000	$ZZ,ZZZ BLANK WHEN ZERO	leer
000000	$9,999.99 BLANK WHEN ZERO	leer
000000	$Z,ZZZ.ZZ BLANK WHEN ZERO	leer

Die Data-Name-Klausel

In der Datendefinition spezifiziert der Data-Name-1 den Namen eines Feldes oder einer Gruppe von Feldern. Er darf nicht über 30 Zeichen lang sein.

Er muß das erste auf die Stufennummer folgende Wort sein und beginnt in jeder beliebigen Spalte rechts von der Spalte 11 des COBOL-Programmblattes. Es muß jedoch zumindest ein Leerzeichen die Stufennummer vom Data-Name trennen.

Ein Data-Name kann in einem Programm mehrfach verwendet werden, da er entweder durch einen zugeordneten Datei-Namen oder einen Data-Name einer höheren Stufe qualifiziert sein kann.

Der die Stufennummer 01 aufweisende Data-Name einer Datendefinition wird auch Datensatz-Name genannt. Auch Satznamen können mit dem Namen der zugehörigen Datei qualifiziert werden.

Die FILLER-Klausel

Ist ein Elementarfeld mit FILLER benannt, kann es nicht direkt angesprochen werden.

Da das Wort FILLER nur Elementarfelder betrifft, kann es nicht unterteilt werden. Vom Programmierer kann das Wort FILLER in einem Programm so oft als nötig verwendet werden.

Wird das reservierte Wort FILLER in einer Datenfeldbeschreibung verwendet, muß es dort das erste auf die Stufennummer folgende Wort sein und in jeder beliebigen Spalte rechts von der Spalte 11 des Programmblattes beginnen. Mindestens ein Leerzeichen muß die Stufennummer von dem Wort FILLER trennen.

Die EXTERNAL-Klausel

Mit ihr wird festgelegt, daß ein Datenfeld auch als extern verwendbar gilt, d. h. von anderen Programmen innerhalb der Run-Unit auch angesprochen werden kann, wenn sie ebenfalls dieses Feld unter demselben Namen und mit demselben Aufbau definieren. Redefinition ist dort aber möglich.

Format:

IS EXTERNAL

EXTERNAL darf nur in der FD-Klausel der File-Section oder in der Working-Storage-Section für Satzbeschreibungen verwendet werden. Gleiche Namen für verschiedene Satzbeschreibungen mit EXTERNAL innerhalb desselben Programmes sind nicht erlaubt, da nicht qualifizierbar. Auch die VALUE-Klausel ist nicht erlaubt mit Ausnahme von Bedingungsnamen-Definition (Stufen-Nr. 88) im Zusammenhang mit solchen Datenbereichen. EXTERNAL schließt GLOBAL aus.

Die GLOBAL-Klausel

Mit ihr wird festgelegt, daß ein Daten-Name ein Globalname ist. Ein Globalname kann aus jedem Programm angesprochen werden, welches in demjenigen Programm enthalten ist, das den Globalnamen definiert.

Format:

IS GLOBAL

Syntax: Gleichnamige Felder innerhalb einer Data Division dürfen nicht global sein. Gegebenenfalls sind sie mit unterschiedlichen Namen zu führen. Für Dateien und ihre Satzdefinitionen, welche der Klausel SAME RECORD AREA unterzogen werden, ist die GLOBAL-Klausel nicht statthaft. Wird bei einem Dateinamen GLOBAL angegeben, gelten alle Felder in der Satzdefinition der Datei als global. In Programmen, die ihrerseits direkt oder indirekt in Programmen enthalten sind, welche Global-Datenbereiche enthalten, kann unmittelbar auf diese zugegriffen werden. Das heißt, Globalbereiche brauchen in enthaltenen Programmen nicht gegendefiniert zu werden. Bei redefinierten Bereichen besitzt nur der Stammbereich das Global-Attribut.

Die JUSTIFIED-Klausel

Die Klausel JUSTIFIED definiert die nicht standardmäßige Positionierung von Daten bei der Übertragung in ein Feld. Hier das allgemeine Format dieser Klausel:

$$\left\{\begin{matrix}\underline{\text{JUSTIFIED}}\\ \underline{\text{JUST}}\end{matrix}\right\} \quad \text{RIGHT}$$

Diese Klausel ist nur für Elementarfelder anwendbar, darf jedoch nicht zur Beschreibung eines numerischen, alphanumerisch aufbereiteten, numerisch aufbereiteten oder booleschen Datenfeldes verwendet werden.

In alphanumerischen Datenfeldern werden ansonsten Daten standardmäßig linksbündig ausgerichtet, d. h. die Anordnung der Daten im Empfangsfeld erfolgt von links nach rechts. Verbleibende Speicherstellen werden mit Leerzeichen aufgefüllt; bei nicht ausreichendem Speicherplatz wird nach rechts abgeschnitten. Die Klausel JUSTIFIED gibt an, daß das entsprechende Feld die Daten rechtsbündig empfangen soll, d.h. die Anordnung der Daten im Empfangsfeld erfolgt von rechts nach links.

Wird die Klausel JUSTIFIED für ein Empfangsfeld angegeben, das kleiner ist als das Sendefeld, dann erfolgt rechtsbündige Ausrichtung der Daten im Empfangsfeld; die linken Zeichen werden abgeschnitten.

Ist das Empfangsfeld mit JUSTIFIED-Klausel größer als das Sendefeld, dann erfolgt ebenfalls rechtsbündige Übertragung, die verbleibenden Stellen links im Feld werden mit Leerzeichen aufgefüllt.

Das Wort JUSTIFIED kann auf JUST abgekürzt werden.

Die LEVEL-NUMBER

Ausführungen über die Level-Number finden Sie weiter hinten im Buch unter "STUFEN-NUMMER".

Die OCCURS-Klausel

Die OCCURS-Klausel wird vom Programmierer verwendet, um Tabellen-Posten und andere gleichartige Sätze von Wiederholungsdaten zu definieren. Enthält eine Datenfeldbeschreibung eine OCCURS-Klausel, dann treffen alle Klauseln auf jedes Auftreten des Datenfeldes (ein Tabellen-Posten genannt) zu. Folglich macht es diese Klausel unnötig, wiederholte Definitionen für jedes Datenfeld in der Tabelle zu formulieren.

Format 1:

```
OCCURS Zahl-2 TIMES

[{ASCENDING }                                                    ]
[{DESCENDING} KEY IS Identifier-4 [, Identifier-5] ... ] ...

 [INDEXED BY Index-name-1 [, Index-name-2] ..]
```

Format 2:

```
OCCURS Zahl-1 TO Zahl-2 TIMES DEPENDING ON Identifier-3

[{ASCENDING }                                                    ]
[{DESCENDING} KEY IS Identifier-4 [, Identifier-5] ... ] ...

 [INDEXED BY Index-name-1 [, Index-name-2] ...]
```

Die OCCURS-Klausel ist in einer Datendefinition wahlweise verwendbar; in Datenfeldbeschreibungen mit der Stufennummer 01, 66, 77 oder 88 ist sie nicht erlaubt. Die OCCURS-Klausel ist auch in einem Datenfeld variabler Länge nicht erlaubt. Die Länge eines Datenfeldes ist variabel, wenn die Definition eines untergeordneten Datenfeldes eine OCCURS Klausel mit der DEPENDING Angabe enthält.

Hat der Datenbereich, der die OCCURS-Klausel enthält, zusätzlich den EXTERNAL-Status, dann muß auch ein etwaiger verwendeter Identifier-3 den EXTERNAL-Status bekommen. Dasselbe gilt für den GLOBAL-Status. Die VALUE-Klausel ist im Zusammenhang mit OCCURS erlaubt (bei bis zu 100 OCCURS-Klauseln in einem Programm).

Die OCCURS-Klausel für eine bestimmte Anzahl von Tabellen-Posten

Im vorausgehenden Format 1 legt die Klausel OCCURS Zahl-2 TIMES das Auftreten von Identifier-1 zahlenmäßig genau fest. Identifier-1 ist Gegenstand der Datenfeldbeschreibung, die auch die OCCURS-Klausel enthält.

Die Zahl-2 muß dabei eine positive ganze Zahl sein. Es folgt nun zunächst die Darstellung einer Datenfeldbeschreibung mit einer OCCURS-Klausel, die eine Tabelle mit einer bestimmten Anzahl von Posten beschreibt, und dann die Darstellung der damit beschriebenen Datenstruktur.

```
01  TEILESATZ.
    02  T-NR          PIC 999.
    02  MENGE         PIC 999V9 OCCURS 5 TIMES.
```

T-NR	1. MENGE	2. MENGE	3. MENGE	4. MENGE	5. MENGE
9 9 9	9 9 9 9	9 9 9 9	9 9 9 9	9 9 9 9	9 9 9 9

Das Datenfeld MENGE ist der erste Posten in der Tabelle. Die OCCURS-Klausel definiert die Tabelle als aus fünf MENGE-Posten bestehend. Die Definition der ersten MENGE gilt auch für jede der vier Wiederholungen. Somit bestehen alle fünf Datenfelder aus vier jeweils eine Dezimalstelle beinhaltenden numerischen Zeichen. Die aus fünf MENGE-Posten bestehende Tabelle ist von fester Länge. Folglich ist TEILE-SATZ ein Datenfeld fester Länge.

Identifier-1, der Gegenstand der OCCURS-Klausel ist, macht

immer dann, wenn er in einer anderen Anweisung als in einer SEARCH- oder einer USE FOR DEBUGGING-Anweisung auftritt, eine Index- oder eine Subscript-Angabe erforderlich. Im vorangehenden Beispiel muß bei Angabe von MENGE in einer Prozeduranweisung jedesmal entweder eine Index- oder Subscript-Angabe erfolgen.

Ist Identifier-1 in der Datenfeldbeschreibung der Name einer Datengruppe, dann gelten alle Identifier, die zu dieser Gruppe gehören, auch für jede Wiederholung der Datengruppe. Im folgenden Beispiel bringt das dreimalige Auftreten von LAGER-DATEN jeweils die drei Datenfelder ORT, NR und MENGE.

```
01  LAGER-SATZ.
    02  TEILE-NR     PIC 9999.
    02  LAGER-DATEN OCCURS 3 TIMES.
        03  ORT      PIC 99.
        03  NR       PIC 999.
        03  MENGE    PIC 9(5).
```

TEILE NR	1. LAGER-DATEN			2. LAGER-DATEN			3. LAGER-DATEN		
	ORT	NR	MENGE	ORT	NR	MENGE	ORT	NR	MENGE
9 9 9 9	9 9	999	9 9 9 9 9	9 9	999	9 9 9 9 9	9 9	999	9 9 9 9 9

Das Datenfeld LAGER-DATEN ist ein Posten in der Tabelle. Die OCCURS-Klausel spezifiziert die Tabelle als aus drei LAGER-DATEN-Posten bestehend. Die Definition der drei Unterteilungen von LAGER-DATEN (ORT, NR und MENGE) gilt für alle drei Posten in der Tabelle. Somit bestehen alle drei Datenfelder aus einem zweistelligen ORT, einer dreistelligen NR und einer fünfstelligen MENGE. Die aus drei LAGER-DATEN-Posten bestehende Tabelle ist eine Tabelle von fester Länge. Folglich versteht man unter LAGER-SATZ ein Datenfeld fester Länge.

Der Identifier-1, der Gegenstand der die OCCURS Klausel enthaltenden Datenfeldbeschreibung ist, sowie alle dem Identifier-1 untergeordneten Datenfelder müssen entweder mit einer Index- oder einer Subscript-Angabe versehen werden, wenn sie in einer anderen Anweisung als in einer SEARCH- oder einer USE FOR DEBUGGING-Anweisung auftreten. Im vorangehenden Beispiel muß bei Angabe von LAGER-DATEN, ORT, NR und MENGE in einer Prozeduranweisung jedesmal eine Index- oder Subscript-Angabe erfolgen.

Die OCCURS-Klausel für eine variable Anzahl von Tabellen-Posten

Im Format 2 der OCCURS-Klausel gibt die Klausel OCCURS Zahl-1 TO Zahl-2 TIMES DEPENDING ON Identifier-3 an, daß

Identifier-1 variabel oft auftritt. Identifier-1 ist Gegenstand der OCCURS Klausel. Zahl-1 steht für die geringste Anzahl von Tabellenposten; Zahl-2 steht für die höchste Anzahl von Tabellenposten. Zahl-1 und Zahl-2 müssen positive Zahlen sein. Der Wert von Zahl-1 muß größer als 0 und kleiner als der Wert von Zahl-2 sein.

Der Inhalt des Identifier-3 zeigt die Anzahl der Wiederholungen von Identifier-1 (dem Tabellenposten) an. Identifier-3 muß ein ganzzahliges Feld ohne Vorzeichen sein. Identifier-3 kann innerhalb der OCCURS-Klausel mit einer Qualifikations-Angabe versehen werden. Identifier-3 darf keine Positionen in der Tabelle einnehmen. Der Inhalt des Identifier-3 muß in den Bereich von Zahl-1 bis Zahl-2 fallen.

Das folgende Beispiel zeigt eine Tabelle mit einer variablen Anzahl von Tabellenposten. Das Datenfeld TEILE-NR erscheint in den ersten drei Bytes von ARTIKEL-SATZ. Das Datenfeld MENGE wiederum erscheint ein-, zwei-, drei-, vier- oder fünfmal in ARTIKEL-SATZ. Der Inhalt des Datenfeldes AMP gibt die Anzahl der MENGE-Posten an, die im ARTIKEL-SATZ vorhanden sind. Das Beispiel zeigt im Anschluß an die Satzdefinition Darstellungen der fünf möglichen Strukturen von ARTIKELSATZ.

Das Datenfeld MENGE ist der erste Posten in der Tabelle. Die OCCURS Klausel definiert die Tabelle als aus einem bis fünf MENGE-Posten bestehend. Die Definition der ersten MENGE gilt auch für jede der vier Wiederholungen. Somit bestehen alle fünf Datenfelder aus vier Bytes. Die geringste Anzahl von Tabellenposten in der MENGE-Tabelle ist ein MENGE-Posten, während die höchste mögliche Anzahl fünf MENGE-Posten sind. Somit ist die MENGE-Tabelle eine Tabelle variabler Länge. ARTIKEL-SATZ ist ein Datenfeld variabler Länge, denn eines seiner untergeordneten Felder (MENGE) ist in einer OCCURS - DEPENDING ON - Klausel beschrieben.

Der Inhalt des Datenfeldes AMP zeigt die Anzahl der MENGE-Posten in der Tabelle an. Enthält AMP die Zahl 4, dann erscheint der MENGE-Posten viermal in der Tabelle und der ARTIKEL-SATZ umfaßt somit 20 Zeichen insgesamt.

Enthält AMP die Zahl 1, dann erscheint der MENGE-Posten nur einmal in der Tabelle und der ARTIKEL-SATZ umfaßt in diesem Falle 8 Zeichen.

Einer OCCURS Klausel mit einer DEPENDING-Angabe können in der Datensatzbeschreibung nur untergeordnete Datenfeldbeschreibungen folgen. Somit müssen sich im nachfolgenden Beispiel die zwei anderen Datenfelder vor dem Datenfeld MENGE befinden. Der Identifier-1 macht immer dann, wenn er in einer anderen Anweisung als in einer SEARCH- oder einer USE FOR DEBUGGING-Anweisung auftritt, eine Index- oder eine Subscript-Angabe erforderlich. Im nachfolgenden Beispiel muß bei Angabe von MENGE in einer Prozeduranweisung jedesmal entweder eine Index- oder Subscript-Angabe erfolgen.

```
01  ARTIKEL-SATZ.
    02  TEILE-NR    PIC 999.
    02  AMP         PIC 9.
    02  MENGE       PIC 999V9 OCCURS 1 TO 5 TIMES
          DEPENDING ON AMP.
```

				1			
1. Möglichkeit	TEILE-NR	AMP	MENGE				
	999	9	999V9				
			1	2			
2. Möglichkeit	TEILE-NR	AMP	MENGE	MENGE			
	999	9	999V9	999V9			
			1	2	3		
3. Möglichkeit	TEILE-NR	AMP	MENGE	MENGE	MENGE		
	999	9	999V9	999V9	999V9		
			1	2	3	4	
4. Möglichkeit	TEILE-NR	AMP	MENGE	MENGE	MENGE	MENGE	
	999	9	999V9	999V9	999V9	999V9	
			1	2	3	4	5
5. Möglichkeit	TEILE-NR	AMP	MENGE	MENGE	MENGE	MENGE	MENGE
	999	9	999V9	999V9	999V9	999V9	999V9

Ist der Identifier-1 der Name einer Datengruppe, dann gelten

alle Identifier, die zu dieser Gruppe gehören, auch für jede Wiederholung der Datengruppe. Im folgenden Beispiel enthält ein LAGER-DATEN-Posten wie auch jede seiner Wiederholungen jeweils die drei Datenfelder ORT, NR und MENGE. Das Datenfeld ALP enthält die Anzahl von LAGER-DATEN-Tabellenposten für den L-SATZ.

Das Datenfeld LAGER-DATEN ist der erste Posten in der Tabelle. Die OCCURS Klausel spezifiziert die Tabelle als aus einem bis fünf LAGER-DATEN-Posten bestehend. Die Unterteilung von LAGER-DATEN (ORT, NR und MENGE) gilt für alle Posten in der Tabelle.

```
01  L-SATZ.
    02  TEILE-NR      PIC 9999.
    02  ALP           PIC 99.
    02  LAGER-DATEN OCCURS 1 TO 3 TIMES DEPENDING ON ALP.
        03  ORT   PIC 99.
        03  NR    PIC 999.
        03  MENGE   PIC 9(5).
```

1. Möglichkeit

TEILE-Nr.	ALP	1 LAGER-DATEN		
		ORT	NR	MENGE
5043	01	17	629	00432

2. Möglichkeit

TEILE-Nr.	ALP	1 LAGER-DATEN			2 LAGER-DATEN		
		ORT	NR	MENGE	ORT	NR	MENGE
6745	02	15	042	10004	18	158	01500

3. Möglichkeit

TEILE-Nr.	ALP	1 LAGER-DATEN			2 LAGER-DATEN			3 LAGER-DATEN		
		ORT	NR	MENGE	ORT	NR	MENGE	ORT	NR	MENGE
8045	03	11	035	00157	16	140	00205	18	160	00027

Die geringste Anzahl von Tabellenposten in der LAGER-DATEN-Tabelle ist ein LAGER-DATEN-Posten, während die höchste Anzahl von Tabellenposten drei LAGER-DATEN-Posten sind. Somit ist die LAGER-DATEN-Tabelle eine Tabelle variabler Länge. L-SATZ ist ein Datenfeld variabler Länge.

Der Inhalt des Datenfeldes ALP zeigt die Anzahl der LAGER-DATEN-Posten in der Tabelle an. Enthält ALP die Zahl 2, dann erscheint der LAGER-DATEN-Posten zweimal in der Tabelle. L-SATZ umfaßt somit 26 Zeichen.

Enthält ALP die Zahl 3, dann erscheint der LAGER-DATEN-Posten dreimal in der Tabelle. L-SATZ umfaßt in diesem Fall 36 Zeichen.

Einer Datenfeldbeschreibung mit einer OCCURS-Klausel inklusive DEPENDING-Angabe können nur untergeordnete Datenfeldbeschreibungen folgen. Somit müssen sich im vorangehenden Beispiel die zwei Datenfelder TEILE-NR und ALP vor LAGER-DATEN befinden. Auf die Datenfeldbeschreibung von LAGER-DATEN können die drei Datenfelder ORT, NR und MENGE folgen, da diese Unterteilungen von LAGER-DATEN sind.

Identifier-1 sowie alle dem Identifier-1 untergeordneten Datenfelder machen in einer anderen Anweisung als in einer SEARCH- oder einer USE FOR DEBUGGING-Anweisung eine Index- oder eine Subscript-Angabe erforderlich. Im vorangehenden Beispiel muß bei Angabe von LAGER-DATEN, ORT, NR und MENGE in einer Prozeduranweisung jedesmal entweder eine Index- oder Subscript-Angabe erfolgen.

Die KEY IS-Angabe

KEY IS zeigt an, daß die einer Wiederholung unterliegenden Datenfelder je nach den im Identifier-4, Identifier-5, etc. enthaltenen Werten in aufsteigender oder abfallender Reihenfolge angeordnet sind. Identifier-4 muß den Hauptschlüssel enthalten und jeder nachfolgende Identifier enthält den nächstwichtigen Schlüssel. Der zuletzt aufgeführte Identifier enthält den am wenigsten wichtigen Schlüssel. Das folgende Beispiel beschreibt eine Datenstruktur, in der das dreimalige Auftreten von LAGER-DATEN gemäß dem Hauptschlüssel ORT und dem Sekundär-Schlüssel NR in aufsteigender Reihenfolge erfolgt.

```
01  LAGER-SATZ.
    02  TEILE-NR  PIC 999.
    02  LAGER-DATEN OCCURS 3 TIMES
          ASCENDING KEY IS ORT, NR.
        03  ORT      PIC 99.
        03  NR       PIC 999.
        03  MENGE    PIC 9(5).
```

TEILE-NR	1 LAGER-DATEN			2 LAGER-DATEN			3 LAGER-DATEN		
	ORT	NR	MENGE	ORT	NR	MENGE	ORT	NR	MENGE
5 0 4	1 7	629	0 0 4 3 2	1 8	158	0 1 5 0 0	1 8	160	0 0 0 2 7

In der Angabe KEY IS kann der Identifier-4 gleich dem Identifier-1 sein, der unmittelbar nach der Stufennummer in derselben Datenfeldbeschreibung auftritt. Ist Identifier-4 nicht gleich dem Identifier-1, dann muß er in dem durch Identifier-1 bezeichneten Gruppen-Datenfeld enthalten sein. In der vorangehenden Darstellung zum Beispiel ist ORT (Stufe 3) ein untergeordnetes Datenfeld in LAGER-DATEN (Stufe 2).

Identifier-5 und die nachfolgenden Identifier in der KEY IS-Angabe müssen im Identifier-1 enthalten sein. In der vorangehenden Darstellung ist NR (Stufe 3) ein untergeordnetes Datenfeld in LAGER-DATEN (Stufe 2).

Identifier-4, Identifier-5, etc. in der KEY IS-Angabe der OCCURS Klausel können mit einer Qualifikations-Angabe versehen sein.

Ist Identifier-4 nicht gleich dem Identifier-1, dann darf keiner der in der KEY IS-Angabe genannten Identifier eine OCCURS Klausel enthalten. Ferner darf es zwischen den Datenfeldbeschreibungen von Identifier-4, Identifier-5, etc. und der Datenfeldbeschreibung von Identifier-1 keine mit einer OCCURS Klausel geben.

INDEXED BY

Wird auf den Identifier-1 oder eines seiner untergeordneten Datenfelder in der PROCEDURE DIVISION Bezug genommen, dann muß dies entweder durch eine Index- oder eine Subscript-Angabe geschehen, die es ermöglicht, das Datenfeld in der Tabelle, auf das ein Zugriff erfolgen soll, zu identifizieren.

Die Angabe INDEXED BY in der OCCURS-Klausel ist möglich, wenn auf den Identifier-1 oder ein Feld in diesem Identifier-1 durch einen Tabellen-Index statt durch ein Subskriptfeld Bezug genommen werden soll. (Für den gezielten Zugriff auf bestimmte Tabellenposten muß die Nummer des gewünschten Postens entweder in einem Subskriptfeld oder in einem Tabellen-Index bereitgestellt werden. Ein Subskriptfeld ist in der Data Division als numerisches Feld zu definieren. Ein Tabellen-Index ist in der OCCURS-Klausel mittels INDEXED BY zu definieren. Der Programmierer kann zwischen der Subskript- und der Indexmethode frei wählen. Die Index-Methode hat jedoch zwei Vorteile: Nur mit ihr sind die COBOL-Anweisungen SEARCH und SEARCH ALL durchführbar. Sie ist auch die laufschnellere Methode, weil bei ihr nicht, wie bei der Subskriptmethode, interne dezimalbinäre Umrechnungen erforderlich sind.) Die Angabe INDEXED BY teilt dem in der Datenfeldbeschreibung definierten Tabellen-Posten einen Index zu. Der in INDEXED BY enthaltene Name wird als Index-Name bezeichnet. Der Index-Name wird zur Bezugnahme auf den zugeteilten Index verwendet und ist an anderer Stelle im Programm nicht definiert. Der Index wird mittels der SET-Anweisung manipuliert. Er muß vor einem Zugriff auf die Tabelle die Nummer des gewünschten Postens enthalten.

Aus dem folgenden Beispiel ist ersichtlich, wie der LAGER-DATEN-Tabelle durch die Angabe INDEXED BY ein (LAG-INDEX genannter) Index zugeordnet wird.

```
01  LAGER-SATZ.
    02  TEILE-NR     PIC 9999.
    02  LAGER DATEN OCCURS 3 TIMES INDEXED BY LAG-INDEX.
        03  ORT      PIC 99.
        03  NR       PIC 999.
        03  MENGE    PIC 9(5).
```

TEILE-NR	1. LAGER-DATEN			2. LAGER-DATEN			3. LAGER-DATEN		
	ORT	NR	MENGE	ORT	NR	MENGE	ORT	NR	MENGE
9 9 9 9	9 9	999	9 9 9 9 9	9 9	999	9 9 9 9 9	9 9	999	9 9 9 9 9

Die Datenfeldbeschreibungen ermöglichen durch Index-Angaben den Zugriff auf jeden der drei LAGER-DATEN Posten. In der PROCEDURE DIVISION ist der Index-Name für die Identifizierung des Tabellen-Postens in Klammern eingeschlossen und folgt direkt auf das letzte Zwischenraumzeichen nach dem Identifier für den Tabellen-Posten. Das folgende Beispiel zeigt, wie durch die PROCEDURE DIVISION auf den ersten LAGER-DATEN Posten zugegriffen wird.

```
    SET LAG-INDEX TO 1.
    ADD MENGE (LAG-INDEX) TO TOTAL-MENGE.
    MOVE LAGER-DATEN (LAG-INDEX) TO DRUCKER.
```

Für den Zugriff auf den dritten LAGER-DATEN Posten sind folgende Anweisungen in der PROCEDURE DIVISION nötig:

```
    SET LAG-INDEX TO 3.
    ADD MENGE (LAG-INDEX) TO TOTAL-MENGE.
    MOVE LAGER-DATEN (LAG-INDEX) TO DRUCKER.
```

Der Programmierer würde diese zwei Anweisungen wahrscheinlich in eine Subroutine aufnehmen und die obigen Prozeduren wie nachfolgend ersichtlich abändern.

```
    SET LAG-INDEX TO 1.
    PERFORM ADD-MOVE-ROUTINE.
    SET LAG-INDEX TO 3.
    PERFORM ADD-MOVE-ROUTINE.

ADD-MOVE-ROUTINE.
    ADD MENGE (LAG-INDEX) TO TOTAL-MENGE.
    MOVE LAGER-DATEN (LAG-INDEX) TO DRUCKER.
```

Das vorausgehende Beispiel zeigt den Vorteil der Indexierung, wenn nur eine Routine für die Verarbeitung jedes Tabellen-Postens programmiert wird.

DIE PICTURE-KLAUSEL

Die PICTURE-Klausel beschreibt die Charakteristika und ggf. das Druckaufbereitungsformat eines Elementarfeldes. Mit dieser Klausel beschreibt der Programmierer jede einzelne Position in einem Datenfeld. Diese Klausel ist auf der Stufe eines Elementarfeldes erforderlich, Index-Datenfelder ausgenommen. Bei höheren Stufen als der Elementarstufe darf die PICTURE-Klausel nicht angegeben werden.

Format:

```
{PICTURE}
{PIC    } IS Zeichenfolge
```

Das Wort PICTURE kann auf PIC abgekürzt werden.
Die Zeichenfolge innerhalb der PICTURE-Klausel sollte aus nicht mehr als 30 Zeichen bestehen und ist aus bestimmten zulässigen Zeichenkombinationen aus dem COBOL-Zeichenvorrat aufgebaut. In einer PICTURE-Zeichenfolge dürfen drei Arten von Zeichen auftreten, nämlich Daten-Zeichen, Spezialzeichen und Druckaufbereitungszeichen.

Daten-Zeichen

Daten-Zeichen stellen Daten-Positionen dar, die während des Ablaufs des Objekt-Programmes Werte enthalten. Es gibt vier Daten-Zeichen, nämlich den Buchstaben A, den Buchstaben X, die Ziffer 9 und die Ziffer 1.

Jeder Buchstabe A in der PICTURE-Zeichenfolge stellt eine Zeichenposition in einem Feld dar, die nur ein alphabetisches Zeichen oder ein Leerzeichen enthalten darf. Jeder Buchstabe A zählt im definierten elementaren Datenfeld.

Jeder Buchstabe X in der PICTURE-Zeichenfolge stellt eine Zeichenposition in einem Feld dar, die jedes zulässige Zeichen des Zeichenvorrates enthalten darf. Jeder Buchstabe X zählt im definierten elementaren Datenfeld.

Jede Ziffer 9 stellt eine Zeichenfolge in einem Feld dar, die eine Ziffer, wie 0, 1, 2, 3, 4, 5, 6, 7, 8 oder 9 enthalten darf. Jede Ziffer 9 zählt im definierten elementaren Datenfeld.

Jede Ziffer 1 in der PICTURE-Zeichenfolge stellt eine boolesche Position dar, die eines der booleschen Zeichen 0 oder 1 enthalten darf. Jede Ziffer 1 in der PICTURE-Zeichenfolge nimmt ein Memory-Bit ein. Je acht solche booleschen Stellen zählen als ein Byte im definierten elementaren Datenfeld.

Spezialzeichen

Spezialzeichen zeigen an, ob das Feld ein Vorzeichen und/oder einen Dezimalseparator aufweist. Es gibt drei Spezialzeichen, nämlich den Buchstaben S, den Buchstaben V und den Buchstaben P.

Der Buchstabe S in der PICTURE-Zeichenfolge zeigt das Vorhandensein eines Vorzeichens im Datenfeld an. Der Buchstabe S muß als das äußerste linke Zeichen in der PICTURE-Zeichenfolge geschrieben werden. Der Buchstabe S zählt in einem Datenfeld mit der Klausel USAGE IS DISPLAY nicht, falls die Datenfeldbeschreibung nicht eine SIGN-Klausel mit der SEPARATE CHARACTER-Angabe enthält.

Der Buchstabe V in der PICTURE-Zeichenfolge zeigt die Position des Dezimalseparators an. Der Buchstabe V darf in einer PICTURE-Zeichenfolge nur ein einziges Mal auftreten. Er stellt keine Zeichenposition dar und zählt folglich nicht. Soll das Feld ganze Zahlen ohne Dezimalstellen aufnehmen, ist der Dezimalseparator wegzulassen.

Die folgenden Beispiele machen die Auswirkung des Zeichens V in der PICTURE-Zeichenfolge deutlich.

Daten im Datenfeld:	PICTURE-Klausel:	Wert:
3492	99V99	34.92
169	9V99	1.69
169	99V9	16.9
169	V999	.169
12857	9V9999	1.2857
12857	9999V9	1285.7
12857	99999	12857.

Der Buchstabe P in der PICTURE-Zeichenfolge zeigt eine gedachte Dezimalseparator-Skalenstelle an und gibt die Position eines gedachten Rechenseparators an, wenn sich dieser nicht innerhalb des Datenfeldes befindet. Das Zeichen P kann nur rechts oder links als fortlaufende Reihe von P's innerhalb der PICTURE-Beschreibung erscheinen. Da der Buchstabe P einen Dezimalseparator impliziert, erübrigt sich hier für die Anzeige der Position des Dezimalseparators der Buchstabe V. Der Buchstabe P und das Einfügungszeichen (.) für die Druckaufbereitung dürfen nicht zusammen in ein und derselben PICTURE-Zeichenfolge auftreten.

Der Buchstabe P zählt als Speicherstelle nicht. Skalenstellenzeichen zählen jedoch für die Bestimmung der maximalen Anzahl von Ziffernpositionen (18) in numerischen Datenfeldern oder in numerischen, druckaufbereiteten Datenfeldern.

Daten im Datenfeld:	PICTURE-Klausel:	Wert:
123	999PPP	123000.
479	PP999	.00479
360	999P	3600.

Aufbereitungszeichen

Es gibt zwei Methoden, ein gewünschtes Datenformat in der PICTURE-Klausel durchzuführen. Die eine Methode behandelt Unterdrückungs- und Austauschzeichen in der PICTURE-Zeichenfolge, die andere Methode Einfügungszeichen in der PICTURE-Zeichenfolge.

Unterdrückungs- und Austauschzeichen

Unterdrückungs- und Austauschzeichen in einer PICTURE-Zeichenfolge sind die äußersten linken numerischen Zeichenpositionen, in denen Nullen unterdrückt und ersetzt werden, wenn der Inhalt dieser Zeichenposition 0 ist. Nullenunterdrückung und Ersatz der angezeigten Zeichenpositionen erfolgen, sobald das Objekt-Programm Daten in das Feld bringt. Die beiden Unterdrückungs- und Ersatzzeichen sind der Buchstabe Z und das Symbol * (Stern). Der Buchstabe Z und das Symbol * dürfen nicht zusammen in ein und derselben PICTURE-Zeichenfolge erscheinen.

Jeder Buchstabe Z in der PICTURE-Zeichenfolge stellt eine führende numerische Zeichenposition dar, die durch ein Leerzeichen ersetzt wird, wenn der Inhalt dieser Zeichenposition 0 ist. Werden alle numerischen Zeichenpositionen in der PICTURE-Zeichenfolge durch den Buchstaben Z dargestellt und ist der Wert der Daten 0, dann wird das ganze Datenfeld mit Leerzeichen gefüllt. Jeder Buchstabe Z in der PICTURE-Zeichenfolge zählt in dem definierten elementaren Datenfeld.

Daten im Datenfeld:	PICTURE-Klausel:	Ausgabe:
00000	ZZZZZ	
12385	ZZZZZ	12385
00004	ZZZZZ	4
00184	ZZZZZ	184
00003	ZZZ99	03

Jeder Stern (*) innerhalb der PICTURE-Zeichenfolge stellt eine führende numerische Zeichenposition dar, in die das Symbol * (Stern) plaziert wird, falls der Inhalt dieser Position 0 ist. Wenn alle numerischen Zeichenpositionen in der PICTURE-Zeichenfolge durch das Symbol * dargestellt sind und der Wert der Daten 0 ist, dann wird das gesamte Datenfeld mit Stern-Zeichen versehen, dies jedoch mit Ausgabe des Dezimalseparators. Jeder Stern zählt bei der Länge des definierten elementares Datenfeldes mit.

Daten im Datenfeld:	PICTURE-Klausel:	Ausgabe:
00000	*****	*****
00500	*****	**500
00737	***99	**737
00082	**999	**082
93720	*****	93720

Einfügungszeichen

Die Einfügungszeichen zeigen an, daß bestimmte Zeichen in definierte Positionen mit eingefügt werden, wenn das Objekt-Programm Daten in das Datenfeld überträgt. Die Art des eingefügten Zeichens hängt von dem in die PICTURE-Zeichenfolge geschriebenen Symbol ab. Verwendet der Programmierer Einfügungszeichen für die Druckaufbereitung, so muß er sicher sein, daß das betreffende Datenfeld groß genug ist, um sowohl die Daten als auch die Einfügungszeichen aufnehmen zu können. Widrigenfalls geht der Überhang verloren.

In COBOL gibt es vier Arten von Einfügungen. Diese vier Arten und die zugeordneten Aufbereitungszeichen sind nachfolgend aufgeführt.

Einfügungsart	Aufbereitungszeichen
Einfache Einfügung	Komma (,) Leerzeichen (B) Null (0) Schrägstrich (/) Punkt (.)
Spezial-Einfügung	Punkt (.)
Feste Einfügung	Währungszeichen Plus (+) Minus (-) Credit (CR) Debit (DB)
Gleitende Einfügung	Währungszeichen Plus (+) Minus (-)

Aufbereitung durch feste Einfügung erfolgt dadurch, daß das Einfügungszeichen in dem aufbereiteten Datenfeld die gleiche Zeichenposition wie in der PICTURE-Zeichenfolge einnimmt.

Aufbereitung durch gleitende Einfügung wird in einer PICTURE-Zeichenfolge durch eine Folge von zumindest zwei der gleitenden Einfügungszeichen angezeigt. Diese Folge von gleitenden Einfügungszeichen kann auch nicht-gleitende Einfügungszeichen enthalten. Nicht-gleitende Einfügungszeichen können auch unmittelbar rechts nach der Folge von gleitenden Zeichen erscheinen. In diesem Falle werden die nicht-gleitenden Einfügungszeichen Teil der gleitenden Folge.

Das äußerste linke Zeichen der gleitenden Einfügungsfolge stellt die äußerste linke Grenze des gleitenden Zeichens im Datenfeld dar. Das äußerste rechte Zeichen der gleitenden Einfügungsfolge stellt die äußerste rechte Grenze des gleitenden Zeichens im Datenfeld dar. Das zweite gleitende Zeichen von links stellt die äußerste linke Grenze der numerischen Daten, die in dem Datenfeld gespeichert werden können, dar. Numerische Daten außer 0 können alle die gleitenden Zeichen vom zweiten gleitenden Zeichen an nach rechts ersetzen.

Es gibt zwei Arten der Darstellung eines gleitenden Einfügungszeichens. Bei der einen Art werden beliebige oder alle der führenden numerischen Zeichenpositionen durch das Einfügungszeichen links vom Dezimalseparator dargestellt. Bei der anderen Art werden alle numerischen Zeichenpositionen in der PICTURE-Zeichenfolge durch das Einfügungszeichen dargestellt.

Erscheinen die gleitenden Einfügungszeichen in der PICTURE-Zeichenfolge nur links vom Dezimalpunkt, dann wird ein einzelnes gleitendes Einfügungszeichen in einer der folgenden Positionen plaziert, je nachdem, welche weiter links in der PICTURE-Zeichenfolge steht:

- der Zeichenposition, die dem Dezimalseparator unmittelbar vorangeht;

- der Zeichenposition, die der ersten keine Null darstellenden Ziffer in der Zeichenfolge unmittelbar vorangeht.

Wenn alle numerischen Datenpositionen in der PICTURE-Zeichenfolge durch gleitende Einfügungszeichen dargestellt sind, dann hängt das Resultat der Aufbereitung vom Wert der Daten ab. Ist der Wert der Daten 0, dann erhält das gesamte Datenfeld Leerzeichen. Ist der Wert der Daten nicht 0, dann wird das Resultat der Aufbereitung dasselbe sein, als wenn das Einfügungszeichen lediglich links vom Dezimalseparator steht.

Das Datenfeld, in welches die Daten eingefügt werden sollen, muß groß genug sein, um sowohl die Daten als auch die Einfügungszeichen aufnehmen zu können. Ist das empfangende Datenfeld zu klein, werden die Einfügungszeichen zwar eingefügt, dafür gehen aber Daten, die dann nicht mehr in das Empfangsfeld passen, verloren.

Die angegebene PICTURE-Zeichenfolge für das Empfangsfeld muß die Summe sein aus:

- der Anzahl von Zeichen in dem Sendefeld;

- der Anzahl von nicht-gleitenden Zeichen, die für das Empfangsfeld aufbereitet sind;

- einem Zeichen für das gleitende Einfügungszeichen.

Währungszeichen-Symbol als Einfügungszeichen

Das Währungszeichen-Symbol in der PICTURE-Zeichenfolge stellt eine Zeichenposition dar, in die ein Währungszeichen zu bringen ist. Das Währungszeichen wird durch das $-Symbol oder das in der CURRENCY-SIGN Klausel im SPECIAL-NAMES Paragraphen festgelegte einzelne Zeichen dargestellt. Das Währungszeichen-Symbol wird bei der Datenfeldlänge mitgezählt. Es ist entweder ein festes oder ein gleitendes Einfügungszeichen.

Das feste Währungszeichen-Symbol ist gekennzeichnet durch das Auftreten von nur einem Währungszeichen-Symbol in einer PICTURE-Zeichenfolge. Das feste Währungszeichen muß die äußerste linke Position der Zeichenfolge darstellen, mit einer Ausnahme: Einzig ein + oder - Symbol darf links des festen Währungszeichens stehen.

Gleitende Einfügung mit dem Währungszeichen-Symbol wird in einer PICTURE-Zeichenfolge durch eine Folge von zumindest zwei Währungszeichen-Symbolen angezeigt.

Die folgenden Beispiele zeigen feste und gleitende Währungszeichen-Symbole $.

Daten im Datenfeld:	PICTURE-Klausel:	Ausgabe:
0000	$9999	$0000
1234	$9999	$1234
0005	$9999	$0005
0012	$ZZZZ	$ 12
002	$Z99	$ 02
0007	$$$$$	$7
1269	$$$$$	$1269
0003	$$$99	$03
00000	$****9	$****0
00185	$*****	$**185
3123	$999	$123

Komma-Einfügung

Jedes Komma (,) in der PICTURE-Zeichenfolge stellt eine Zeichenposition dar, in die ein Kommazeichen eingefügt werden soll. Diese Zeichenposition wird bei der Datenfeldlänge mitgezählt. Das Einfügungszeichen Komma darf nie das letzte Zeichen in einer PICTURE-Zeichenfolge sein. Wenn die Null unmittelbar links einer Symbol-Position unterdrückt wurde, dann wird das Komma in diese Position nicht eingefügt. An seiner Stelle wird das Zeichen, welches die Nullen ersetzt, eingefügt. Wenn im SPECIAL-NAMES Paragraphen die Klausel DECIMAL-POINT IS COMMA geschrieben wurde, gelten diese Regeln für das in einer PICTURE-Zeichenfolge auftretende Punktzeichen.

Daten im Datenfeld:	PICTURE-Klausel:	Ausgabe:
00000	99,999	00,000
00000	ZZ,ZZZ	
00125	ZZ,ZZZ	125
01283	ZZ,ZZZ	1,283
01283	$ZZ,ZZZ	$ 1,283
00000	$ZZ,ZZZ	
00000	$**,***	******
00000	$**,*99	$****00

Punkt-Einfügung
Dezimalseparator-Einfügung

Jeder Punkt (.) in der PICTURE-Zeichenfolge stellt die Position des Dezimalseparators dar; gleichzeitig stellt er auch eine Zeichenposition dar, in die ein Punkt eingefügt werden soll. Das Punkt-Zeichen wird bei der Länge eines Datenfeldes mitgezählt. Der Punkt darf nie das letzte Zeichen in der PICTURE-Zeichenfolge sein. Wird das ganze Datenfeld durch Zwischenraumzeichen unterdrückt, wird auch das Zeichen "Punkt" durch ein Zwischraumzeichen ersetzt. Wird dagegen das ganze Datenfeld durch Stern-Zeichen ersetzt, wird der Punkt nicht ersetzt. Enthält eine PICTURE-Zeichenfolge einen Punkt als Einfügungszeichen, so darf sie kein "V"-Zeichen enthalten. Wenn im SPECIAL-NAMES Paragraphen die Klausel DECIMAL-POINT IS COMMA geschrieben wurde, gelten diese Regeln für das in einer PICTURE-Zeichenfolge auftretende Komma.

Daten im Datenfeld:	PICTURE-Klausel:	Ausgabe:
^000135	.999999	.000135
0013^59	9999.99	0013.59
0004^28	99.9999	04.2800
2^8	999.99	002.80
1250	$$,$$$.99	$1,250.00
^1250	$$,$$$.99	$.12
12345	$$,$$$.99	$2,345.00
00000	ZZZZZZ.ZZ	
00000	******.**	******.**
^05	$$.$$	$.05
^00	$$.$$	

Leerstellen-Einfügung durch B-Symbole

Jedes "B" in der PICTURE-Zeichenfolge stellt eine Zeichenposition dar, in die automatisch ein Leerzeichen eingefügt wird, wenn Daten in das Feld übertragen werden. Das "B" wird bei der Datenfeldlänge mitgezählt.

Daten im Datenfeld:	PICTURE-Klausel:	Ausgabe:
A3629	XBXXXX	A 3629
CITYSTATE	XXXXBXXXXX	CITY STATE
FMLASTNAME	XBXBXXXXXXX	F M LASTNAME
123^42	999BV99	123 42

Nullen-Einfügung durch 0-Symbole

Jede "0" in der PICTURE-Zeichenfolge stellt eine Zeichenposition dar, in die die Ziffer "0" durch das Objekt-Programm eingefügt wird, wenn Daten in das Feld übertragen werden. Die Null wird bei der Datenfeldlänge mitgezählt.

Daten im Datenfeld:	PICTURE-Klausel:	Ausgabe
125	999000	125000
347	00999	00347
ABCXYZ	XXXX0XX	ABCX0YZ

Schrägstrich-Einfügung durch /-Symbol

Jedes "/"-Symbol in der PICTURE-Zeichenfolge stellt eine Zeichenposition dar, in die das Zeichen "/" durch das Objekt-Programm eingefügt wird, wenn Daten in das Feld übertragen werden. Der Schrägstrich wird in der Länge des Datenfeldes mitgezählt.

Daten im Datenfeld:	PICTURE-Klausel:	Ausgabe:
102378	99/99/99	10/23/78
781230	99/99/99	78/12/30
0978	Z9/99	9/78
1078	Z9/99	10/78
542001	99/9999	54/2001

Punkt-Einfügung durch .-Symbol

Hier gelten dieselben Regeln wie bei der vorab beschriebenen Einfügung des Schrägstrich-Symbols.

Symbole für das Vorzeichen

Als Vorzeichen gelten die vier Symbole "+", "-", "CR" und "DB". Lediglich eines dieser Vorzeichen darf in ein und derselben PICTURE-Zeichenfolge enthalten sein. Die folgende Tabelle faßt die Wirkung der vier Vorzeichen auf positive und negative Datenfelder zusammen.

	Ergebnis	
Vorzeichen	Datenfeld positiv oder null	Datenfeld negativ
+	+	-
-		-
CR		CR
DB		DB

Minus-Zeichen

Das Minuszeichen (-) in der PICTURE-Zeichenfolge stellt eine Zeichenposition dar, in die durch das Objekt-Programm ein Zeichen eingefügt wird, das die Daten als entweder positiv oder negativ kennzeichnet. Das Minuszeichen erscheint in dem editierten Feld lediglich, wenn die in das Datenfeld übertragenen Daten negativ sind. Sind die in das Datenfeld übertragenen Daten positiv, dann wird ein Leerzeichen in die Position eingefügt.

Das Minuszeichen ist in der Länge des Datenfeldes mitzuzählen. Das Minuszeichen ist entweder ein festes oder ein gleitendes Einfügungszeichen.

Das feste Minuszeichen ist gekennzeichnet durch das Auftreten lediglich eines Minuszeichens in der PICTURE-Zeichenfolge. Das feste Minuszeichen muß sich entweder in der äußersten linken oder der äußersten rechten Zeichenposition der PICTURE-Zeichenfolge befinden.

Das gleitende Minuszeichen wird durch eine folge von zumindest zwei Minuszeichen am linken Ende der PICTURE-Zeichenfolge angezeigt. Das gleitende Minuszeichen bewirkt, daß jede führende 0 in den Daten durch ein Leerzeichen ersetzt wird und daß das der äußersten linken, keine 0 darstellenden Ziffer in den Daten vorausgehende Zeichen entweder ein Leerzeichen oder ein Minuszeichen ist.

Daten im Datenfeld:	PICTURE-Klausel:	Ausgabe:
4500	-9999	4500
-4500	-9999	-4500
3921	9999-	3921
-0921	9999-	0921-
12345	----99	12345
00034	----99	34
-00034	----99	-34
-00034	------	-34
000	----	
-000	----	

Plus-Zeichen

Das Pluszeichen in der PICTURE-Zeichenfolge stellt eine Zeichenposition dar, in die durch das Objekt-Programm ein Zeichen eingefügt wird, das die Daten als entweder positiv oder negativ kennzeichnet. Das Pluszeichen erscheint in dem editierten Feld, wenn die in das Datenfeld übertragenen Daten positiv sind. Sind die in das Datenfeld übertragenen Daten negativ, dann wird ein Minuszeichen (-) in die Position eingefügt.

Das Pluszeichen ist in der Länge des Datenfeldes mitzuzählen. Es ist entweder ein festes oder ein gleitendes Einfügungszeichen.

Das feste Pluszeichen ist gekennzeichnet durch das Auftreten von nur einem Pluszeichen in einer PICTURE-Zeichenfolge. Das feste Pluszeichen muß entweder in der äußersten linken oder äußersten rechten Zeichenposition der PICTURE-Zeichenfolge stehen.

Das gleitende Pluszeichen wird durch eine Folge von wenigstens zwei Pluszeichen am linken Ende der PICTURE-Zeichenfolge angezeigt. Das gleitende Pluszeichen verursacht, daß jede führende 0 in den Daten durch ein Leerzeichen ersetzt wird und daß das der äußersten linken, keine 0 darstellenden Ziffer in den Daten vorausgehende Zeichen entweder ein Pluszeichen oder ein Minuszeichen ist.

Daten im Datenfeld:	PICTURE-Klausel:	Ausgabe:
4500	+9999	+4500
-4500	+9999	-4500
3921	9999+	3921+
-0921	9999+	0921-
12345	++++99	+12345
00034	++++99	+34
-00034	++++99	-34
000	++++	
-000	++++	

Das Symbol CR

Das Credit-Symbol (CR) in der PICTURE-Zeichenfolge umfaßt zwei Zeichenpositionen, in die durch das Objekt-Programm zwei Zeichen eingefügt werden, die die Daten entweder positiv oder negativ kennzeichnen. Sind die in das Feld übertragenen Daten negativ, dann werden die zwei Zeichen "CR" in das editierte Feld eingefügt. Sind die in das Feld übertragenen Zeichen positiv, dann werden anstelle von "CR" zwei Leerzeichen in das editierte Feld eingefügt.

Das Credit-Symbol (CR) muß die beiden Zeichenpositionen am äußersten rechten Ende der PICTURE-Zeichenfolge einnehmen. Das Credit-Symbol "CR" wird bei der Datenfeldlänge als zwei Zeichen mitgezählt.

Daten im Datenfeld:	PICTURE-Klausel:	Ausgabe:
123^45	$999.99CR	$123.45
-123^45	$999.99CR	$123.45CR
-123^45	$***.99BCR	$123.45 CR
-45^6	$$,$$$.99CR	$45.60CR
4820^33	$$,$$$.99CR	$4,820.33
-^00	$$,$$$.99CR	$.00
0^00	$$,$$$.99CR	$.00
0^00	$*,***.99CR	$*****.00

Das Symbol DB

Das Debit-Symbol (DB) in der PICTURE-Zeichenfolge umfaßt zwei Zeichenpositionen, in die durch das Objekt-Programm zwei Zeichen eingefügt werden, die die Daten entweder positiv oder negativ kennzeichnen. Sind die in das Feld übertragenen Daten negativ, dann werden die zwei Zeichen "DB" in das editierte Feld eingefügt. Sind die in das Feld übertragenen Zeichen positiv, dann werden anstelle der beiden Zeichen zwei Leerzeichen in das editierte Feld eingefügt.

Das Debit-Symbol (DB) muß die beiden Zeichenpositionen am äußersten rechten Ende der PICTURE-Zeichenfolge einnehmen. Das Debit-Symbol "DB" wird bei der Datenfeldlänge als zwei Zeichen mitgezählt.

Daten im Datenfeld:	PICTURE-Klausel:	Ausgabe:
123^45	$999.99DB	$123.45
-132^45	$999.99DB	$123.45DB
-123^45	$***.99BDB	$123.45 DB
-45^6	$$,$$$.99DB	$45.60DB
4820^33	$$,$$$.99DB	$4,820.33
-^00	$$,$$$.99DB	$.00
0^00	$$,$$$.99DB	$.00
0^00	$*,***.99DB	$*****.00

Abkürzung für fortlaufende Wiederholungen

Folgt einem der Zeichen A, X, 9, 1, P, Z, *, B, 0, /, -, +, Währungszeichen oder Komma eine in Klammern stehende ganze Zahl, dann zeigt diese die Anzahl der fortlaufenden Wiederholungen des Zeichen an, z. B.:

X(10) entspricht XXXXXXXXXX
9(6)V9(5) entspricht 999999V99999

Von den Zeichen S, V, CR, DB und Punkt darf jedoch nur eines in derselben PICTURE-Zeichenfolge enthalten sein.

Aufgrund der Zeichen in der PICTURE-Zeichenfolge eines Elementarfeldes unterscheidet man sechs Daten-Kategorien, nämlich die alphabetische, numerische, alphanumerische, alphanumerisch editierte (aufbereitete), numerisch editierte (aufbereitete) und boolesche Kategorie.

Ein Feld gehört der alphabetischen Kategorie an, wenn seine PICTURE-Zeichenfolge nur die Zeichen A und B enthält.

Bei einem Feld der numerischen Kategorie enthält dessen PICTURE-Zeichenfolge nur die Zeichen 9, P, S und V.

Ein Feld gehört der alphanumerischen Kategorie an, wenn seine PICTURE-Zeichenfolge eine Kombination des Zeichens X enthält.

Ein Feld gehört der alphanumerisch editierten Kategorie an, wenn seine PICTURE-Zeichenfolge eine Kombination der Zeichen X, B, 0, . oder / enthält. Diese Zeichenfolge muß entweder zumindest ein X kombiniert mit zumindest einem Zeichen B, 0, / oder . enthalten.

Ein Feld gehört der numerisch editierten Kategorie an, wenn seine PICTURE-Zeichenfolge eine Kombination des Daten-Zeichens 9, der Spezialzeichen P und V und zumindest eines der Aufbereitungszeichen B / Z O - + * , . CR DB und des Währungszeichens ist.

Ein Feld gehört der booleschen Kategorie an, wenn seine PICTURE-Zeichenfolge lediglich das Zeichen 1 enthält. Jedes Zeichen 1 in der PICTURE-Zeichenfolge stellt ein Speicher-Bit dar. Jedes achtfache Auftreten des Zeichens 1 stellt ein Speicher-Byte (ein Zeichen mit 8 Bits) dar. Definiert eine Zeichenfolge ein boolesches Feld, dann darf die Anzahl von Bytes, die vom booleschen Feld eingenommen werden, 18 nicht überschreiten.

Picture Zeichenvorrangtabelle

Die folgende Tabelle zeigt die Rangfolge, in der Zeichen in der PICTURE-Zeichenfolge auftreten können. Jedes x in der Tabelle zeigt an, daß das senkrecht über dem x auftretende Zeichen zuerst erscheint, gefolgt von dem Zeichen, das auf der linken Seite der Tabelle aufgeführt ist.

Zum Beispiel kann die Tabelle verwendet werden, um die Richtigkeit einer S9 enthaltenen PICTURE-Zeichenfolge festzustellen. Das erste Zeichen S befindet sich am Kopf der Tabelle. Das zweite Zeichen 9 befindet sich auf der linken Seite der Tabelle. Verfolgt man die obere Reihe und die linke Spalte auf das erste und zweite Zeichen hin, so stellt man anhand des x im Schnittpunkt fest, daß in der PICTURE-Zeichenfolge das Zeichen 9 auf das Zeichen S folgen darf.

In der Tabelle erscheinen sich gegenseitig ausschließende Zeichen in geschweiften Klammern. Das Währungszeichen wird durch cs (Currency Symbol) bezeichnet.

Eine PICTURE-Zeichenfolge muß eine der nachfolgenden Varianten beinhalten:

- Sie muß zumindest eines der Zeichen A, X, Z, 9, * oder 1 enthalten.

- Sie muß zumindest zwei der gleitenden Einfügungszeichen +, - oder das Währungszeichen enthalten.

PICTURE (Zeichenvorrangstabelle)

erstes Zeichen / zweites Zeichen		feste Zeichen									gleitende Zeichen						sonstige Zeichen						
		B	0	/	,	.	{+ -}	{+ -}	{CR DB}	cs	{Z *}	{Z *}	{+ -}	{+ -}	cs	cs	9	A X	S	V	P	P	1
feste Zeichen	B	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x		x		x	
	0	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x		x		x	
	/	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x		x		x	
	,	x	x	x	x	x	x			x	x	x	x	x	x	x	x			x		x	
	.	x	x	x	x		x			x	x		x		x		x						
	{+ -}																						
	{+ -}	x	x	x	x	x					x	x			x	x	x			x	x	x	
	{CR DB}	x	x	x	x	x					x	x			x	x	x			x	x	x	
gleitende Zeichen	cs						x																
	{Z *}	x	x	x	x		x			x	x												
	{Z *}	x	x	x	x	x	x			x	x	x								x		x	
	{+ -}	x	x	x	x					x			x										
	{+ -}	x	x	x	x	x				x			x	x						x		x	
	cs	x	x	x	x		x								x								
	cs	x	x	x	x	x	x								x	x				x		x	
andere Zeichen	9	x	x	x	x	x	x			x	x		x		x		x	x	x	x		x	
	A X	x	x	x													x	x					
	S																						
	V	x	x	x	x		x			x	x		x		x		x		x		x		
	P	x	x	x	x		x			x	x		x		x		x		x		x		
	P						x			x									x	x		x	
	1																						x

CS = Currency Sign, Währungssymbol (Z.B. $)

DIE REDEFINES-KLAUSEL

Die REDEFINES-Klausel gestattet eine unterschiedliche Definition ein und desselben Datenbereichs.

Format:

REDEFINES Identifier-2

Die REDEFINES-Klausel muß unmittelbar auf den Identifier-1 folgen. Der Identifier-1 ist Gegenstand der die REDEFINES-Klausel enthaltenden Datenfeldbeschreibung. Auf die REDEFINES-Klausel können andere Datenbeschreibungsklauseln, wie z. B. die PICTURE-Klausel, folgen. Die Stufennummern des Identifiers-1 und des Identifiers-2 müssen gleich sein.

Die Datendefinition des Identifiers-2 und dessen untergeordnete Datenfeldbeschreibungen müssen der Datenbeschreibung des Identifiers-1 unmittelbar vorangehen. Die Datenbeschreibung des Identifiers-2 definiert den Speicherbereich. Die Datenbeschreibung des Identifiers-1 gibt eine neue Beschreibung (Redefinition) des Speicherbereiches. Die Redefinition beginnt mit dem Identifier-2 und endet vor der nächsten Datendefinition mit derselben oder einer niedrigeren Stufennummer.

Im folgenden Beispiel wird ein aus drei Zeichen bestehendes Datenfeld, das als PROZENT definiert ist, als PROZENT-AUSG redefiniert. Die Redefinition mit einer Dezimalposition ermöglicht es, den Prozentsatz anstatt .259, wie für interne Berechnung verwendet, als 25,9 % zu schreiben.

```
01  SATZ.
    02  KONTO-NUMMER     PIC 9(6).
    02  WERT             PIC 9(5)V99.
    02  PROZENT          PIC V999.
    02  PROZENT-AUSG REDEFINES PROZENT PIC 99V9.
    02  PREIS            PIC 9V999.
```

Der Datenbereich SATZ im Beispiel umfaßt 20 Bytes.

Die REDEFINES-Klausel darf in der FILE SECTION auf der Stufennummer 01 nicht verwendet werden, denn mehr als ein Auftreten einer 01-Stufennummer unter einem FD zeigt automatisch die Redefinition des mit dem FD verbundenen Datensatz-Bereiches im Speicher an. Die REDEFINES-Klausel kann jedoch auf der Stufennummer 01 in der WORKING-STORAGE-SECTION und GLOBAL SECTION verwendet werden.

Bei Redefinitionen von Bereichen mit Stufennummern ungleich 01 muß der Identifier-1 die gleiche Länge wie der Identifier-2 aufweisen. Bei Redefinitionen auf einer Stufennummer 01 in einer WORKING-STORAGE SECTION oder GLOBAL SECTION kann der Identifier-1 einen Speicherbereich mit derselben oder einer kürzeren Länge als der des Identifiers-2 aufweisen.

Nachfolgend wird ein Beispiel eines Datensatzes und seiner zwei Formate in einem Programm gezeigt. Da dieser Datensatz nur einmal gespeichert ist, muß sein zweites Format durch eine REDEFINES-Klausel beschrieben werden.

FORMAT 1:

TEILE-SATZ								
TEILE-NUMMER	HERST-DATUM	MENGEN				INVENTUR-DATUM		
		IN-ARBEIT	MENGE 1	MENGE 2	MENGE 3	INV TAG	INV MON	INV JHR
9 9 9 9 9 9	9 9 9 9 9 9	9 9 9 9	9 9 9	9 9 9	9 9 9	9 9	9 9	9 9

FORMAT 2:

TEILE-SATZ									
TEILE-NUMMER	HERST-DATUM	STATUS					INVENTUR-DATUM		
		STOP-DATUM			BEST GREN-ZE	VORRATS MENGE	INV TAG	INV MON	INV JHR
		TAG	MON	JHR					
9 9 9 9 9 9	9 9 9 9 9 9	9 9	9 9	9 9	9 9 9	9 9 9 9	9 9	9 9	9 9

```
01  TEILE-SATZ.
    02  TEILE-NUMMER          PIC 9(7).
    02  HERST-DATUM           PIC 9(6).
    02  MENGEN.
        03  IN-ARBEIT         PIC 9999.
        03  MENGE1            PIC 999.
        03  MENGE2            PIC 999.
        03  MENGE3            PIC 999.
    02  STATUS REDEFINES MENGEN.
        03  STOP-DATUM.
            04  STOP-TAG      PIC 99.
            04  STOP-MONAT    PIC 99.
            04  STOP-JAHR     PIC 99.
        03  BEST-GRENZE       PIC 999.
        03  VORRATS-MENGE     PIC 9999.
    02  INVENTUR-DATUM.
        03  INV-TAG           PIC 99.
        03  INV-MONAT         PIC 99.
        03  INV-JAHR          PIC 99.
```

Dieses Beispiel zeigt die Reihenfolge der Darstellung der zwei Formate. Zuerst werden die ursprünglichen, den Speicherbereich einnehmenden Felder beschrieben (MENGEN und die Unterteilungen davon); dann kommt die Redefinition (STATUS REDEFINES MENGEN) und schließlich kommen die Unterteilungen von STATUS (STOP-DATUM, TAG, etc.).

Wenn MENGEN eine weitere Redefinition, wie beispielsweise BESTELL-DATEN hätte, würde diese zweite Redefinition unmittelbar auf die erste Redefinition folgen. Alle Redefinitionen eines Bereiches müssen sich auf ihn beziehen.

Vergleichen Sie hierzu das Beispiel auf der folgenden Seite!

```
01  TEILE-SATZ.
    02  TEILE-NUMMER          PIC 9(7).
    02  HERST-DATUM           PIC 9(6).
    02  MENGEN.
        03  IN-ARBEIT         PIC 9999.
        03  MENGE1            PIC 999.
        03  MENGE2            PIC 999.
        03  MENGE3            PIC 999.
    02  STATUS REDEFINES MENGEN.
        03  STOP-DATUM.
            04  STOP-TAG      PIC 99.
            04  STOP-MONAT    PIC 99.
            04  STOP-JAHR     PIC 99.
        03  BEST-GRENZE       PIC 999.
        03  VORRATS-MENGE     PIC 9999.
    02  BESTELL-DATEN REDEFINES MENGEN.
        03  BESTELL-MENGE     PIC 9999.
        03  BESTELL-DATUM     PIC 9(6).
        03  FILLER            PIC 999.
    02  INVENTUR-DATUM.
        03  INV-TAG           PIC 99.
        03  INV-MONAT         PIC 99.
        03  INV-JAHR          PIC 99.
```

Die REDEFINES-Klausel unterliegt folgenden Einschränkungen:

1. Die Datenfeldbeschreibung für Identifier-2 darf keine REDEFINES-Klausel enthalten.

2. Die Datenfeldbeschreibung für Identifier-2 darf keine OCCURS-Klausel enthalten.

3. Die Redefinition darf keine VALUE-Klauseln enthalten.

DIE SIGN-KLAUSEL

Die SIGN-Klauseln bezeichnet die Position und die Darstellungsart des Vorzeichens in einem numerischen Datenfeld, das ungepackte Zeichen, an die das Vorzeichen recht oder links angefügt wird, enthält.

Format:

```
[SIGN IS]   {LEADING}   [SEPARATE CHARACTER]
            {TRAILING}
```

Es gibt zwei Arten von mit Vorzeichen versehenen Usage-Display-Dezimaldaten. Die mit Vorzeichen versehenen Datenarten werden durch eine Datenfeldbeschreibung beschrieben, die eine PICTURE-Zeichenfolge mit dem Buchstaben S und dem Datenzeichen 9, der USAGE IS DISPLAY-Klausel und der SIGN-Klausel enthält. Die Varianten der

SIGN-Klausel ermöglichen es dem Compiler, die Art der mit Vorzeichen versehenen numerischen Daten wie folgt zu bestimmen:

- Numerischer Datentyp mit rechts oder links angefügtem Vorzeichen:

  PICTURE S9 USAGE IS DISPLAY SIGN IS TRAILING SEPARATE CHARACTER

  PICTURE S9 USAGE IS DISPLAY SIGN IS LEADING SEPARATE CHARACTER

- Numerischer Datentyp, bei dem das Vorzeichen aus der rechten oder linken äußeren Ziffer verschlüsselt hervorgeht:

  PICTURE S9 USAGE IS DISPLAY SIGN IS LEADING
  PICTURE S9 USAGE IS DISPLAY SIGN IS TRAILING

Enthält die Datenfeldbeschreibung PICTURE S9 USAGE IS DISPLAY ohne eine SIGN-Klausel, dann nimmt der Compiler den numerischen Datentyp an, bei dem das Vorzeichen aus der rechten äußeren Ziffer verschlüsselt hervorgeht.

Die SIGN-Klausel kann auch für eine Datengruppe angewandt werden, die zumindest eine numerische Datenfeldbeschreibung mit Vorzeichen enthält. Gilt die SIGN-Klausel für eine Datengruppe, so definiert sie die Art der mit Vorzeichen versehenen Dezimaldaten für jede untergeordnete Datenfeldbeschreibung mit PICTURE S9 und USAGE IS DISPLAY.

Ist die CODE-SET Klausel in einer Dateibeschreibung enthalten, dann müssen alle mit Vorzeichen versehenen Datenfeldbeschreibungen , die mit dieser Dateibeschreibung zusammenhängen, mit der Angabe SEPARATE CHARACTER in der SIGN-Klausel beschrieben werden.

STUFENNUMMER (Level-Nr.)

Eine Stufennummer, die den Beginn einer Datenfeldbeschreibung bezeichnet, kann jede beliebige Zahl von 01 bis 49 sowie 66, 77 oder 88 sein. Das einzige Erfordernis dabei ist, daß die Stufennummer 01 immer den Datensatz-Namen bezeichnen muß. Die Werte der Stufennummern 01 bis 09 können als 01, 02, 03, 04, 05, 06, 07, 08, 09 oder als 1, 2, 3, 4, 5, 6, 7, 8, 9 geschrieben werden.

Mehrere einer FD oder SD untergeordnete Satz-Definitionen mit der Stufennummer 01 stellen stillschweigende Redefinitionen des ersten Satzes dar.

Die Stufennummer 01 muß in den Spalten 8, 9, 10 oder 11 des COBOL-Programmblattes beginnen. Die Stufennummern 02 oder 2 bis 49 können irgendwo ab Spalte 8 auf dem Programmblatt geschrieben werden. Eine numerisch höhere Stufennummer wird unter der vorangehenden Stufennummer geschrieben oder kann auch von Spalte 8 weg beliebig nach rechts eingerückt geschrieben werden.

BEISPIEL 1: Untereinander geschriebene Stufennummern

```
01  ZEIT-SATZ.
02  NAME.
03  NACHNAME     PIC X(11).
03  VORNAME1     PIC X.
03  VORNAME2     PIC X.
02  PERS-NUMMER  PIC 9999.
02  DATUM.
03  MONAT        PIC 99.
03  TAG          PIC 99.
03  JAHR         PIC 99.
02  STUNDEN      PIC 99.
```

BEISPIEL 2: Eingerückte Stufennummern

```
01  ZEIT-SATZ.
    02  NAME.
        03  NACHNAME     PIC X(11).
        03  VORNAME1     PIC X.
        03  VORNAME2     PIC X.
    02  PERS-NUMMER      PIC 9999.
    02  DATUM.
        03  MONAT        PIC 99.
        03  TAG          PIC 99.
        03  JAHR         PIC 99.
    02  STUNDEN          PIC 99.
```

Einrücken nach rechts hat keinen Einfluß auf den COBOL-Compiler. Es besteht jedoch der Vorteil, daß die eingerück-

ten Datenfeldbeschreibungen auf dem COBOL-Compiler-Ausdruck die Hierarchie der Daten im Datensatz optisch deutlicher hervortreten lassen.

Stufen-Nr. 77 ist nur in der Working-Storage-, Linkage- und Global-Section erlaubt und kann anstatt der Stufen-Nr. 01 zur Definition eines Feldes verwendet werden, welches allein steht, das heißt keiner Feldgruppe angehört.

Stufen-Nr. 66 dient zur Kennzeichnung von RENAMES-Definitionen (siehe unter RENAMES einige Seiten weiter hinten).

Stufen-Nr. 88 dient zur Kennzeichnung von Definitionen mit Vergabe von Bedingungsnamen:

Die Definition von Bedingungsnamen ermöglicht es, einem speziellen Feldinhalt, einer Reihe von Werten und/oder einem Bereich von Werten einen Namen zuzuordnen, der als Bedingungsname bezeichnet wird. Der Eintrag für Bedingungsnamen kann in allen Sections der DATA DIVISION verwendet werden.

Format:

```
88 Bedingungsname { VALUE IS  }  Literal-1 [ { THROUGH } Literal-2 ]
                  { VALUES ARE }             { THRU    }

    [ Literal-3 [ { THROUGH } Literal-4 ] ] ...
                  { THRU    }
```

Die Stufennummer 88 bezeichnet den Beginn einer Bedingungsnamen-Definition. Sie muß auf Spalte 8 oder ab einer der folgenden Spalten geschrieben werden. Der Bedingungsname folgt auf die Stufennummer 88, beginnend ab Spalte 12 oder einer der folgenden Spalten. Der Bedingungsname muß wenigstens ein Leerzeichen Abstand zur Stufennummer 88 haben.

Die VALUE-Klausel in der Bedingungsnamen-Definition legt fest, welche Werte mit dem Bedingungsnamen verbunden sind. Die Literale in der VALUE-Klausel dürfen numerisch, nicht numerisch oder figurative Konstanten sein. Wenn die Phrase THROUGH verwendet wird, muß das Literal-1 wertmäßig kleiner als das Literal-2, Literal-3 wertmäßig kleiner als das Literal-4 sein, usw.

Die Bedingungsnamen-Definition muß unmittelbar auf die Datenbeschreibung des Feldes folgen, dessen Feldinhalt auf die durch den Bedingungsnamen festgelegten Werte geprüft werden soll. Der zu prüfende Original-Datenbereich wird als Bedingungsvariable bezeichnet. Ein Bedingungsname kann Elementarfeldern oder Gruppenfeldern mit folgenden Einschränkungen zugeordnet werden:

- Ein anderer Bedingungsname
- Eine Stufe 66 Datenbeschreibung
- Ein Gruppenfeld, das Feldbeschreibungen mit der JUSTIFIED, der SYNCHRONIZED- oder einer USAGE-Klausel enthält, die nicht USAGE IS DISPLAY lautet.
- Ein Index-Datenfeld
- Einem Datenfeld, das in der Datenbeschreibung die OCCURS-Klausel mit einer DEPENDING-Angabe enthält
- Einem einer Datengruppe untergeordneten Datenfeld, das in seiner Datenbeschreibung die OCCURS-Klausel mit einer DEPENDING-Angabe enthält.

Die Charakteristik des Bedingungsnamen ist indirekt durch den Typ der Bedingungsvariablen gegeben. Die VALUE-Klausel in einem Bedingungsnamen-Eintrag darf nicht im Widerspruch zu den Datenbeschreibungsklauseln der Bedingungsvariablen stehen. Zur Vermeidung von Konflikten müssen folgende Regeln beachtet werden:

1. Gehört die Bedingungsvariable zum Typ der numerischen Daten, dann müssen die Literale in der VALUE-Klausel numerisch sein. Das Literal darf in der Stellenanzahl das Fassungsvermögen der Bedingungsvariablen nicht überschreiten. Das Literal darf nur ein Vorzeichen enthalten, wenn die PICTURE-Zeichenfolge der Bedingungsvariablen numerische Daten mit Vorzeichen beschreibt.

2. Gehört die Bedingungsvariable zur Kategorie der alphabetischen, alphanumerisch oder numerisch aufbereiteten Daten, dann müssen die Literale in der VALUE-Klausel nichtnumerisch sein. Das Literal darf in der Stellenanzahl das Fassungsvermögen der Bedingungsvariablen nicht überschreiten.

Ein Bedingungsname muß innerhalb des Programmes nicht einmalig sein, er kann mit dem Namen der zugehörigen Bedingungsvariablen qualifiziert werden.

Für eine Bedingungsvariable können mehrere Bedingungsnamen vergeben werden. Die Stufe 88 - Datenbeschreibungseinträge folgen nacheinander unmittelbar auf die Datenbeschreibung der Bedingungsvariablen.

Zur Darstellung einer Bedingungsvariablen und der zugeordneten Bedingungs-Namen ist im folgenden Beispiel ein SATZART genanntes Datenfeld vorgesehen, das jeden beliebigen von mehreren die Art eines Bewegungssatzes anzeigenden Werten enthält.

SATZART	Prozedur-Name
1	NEUANLAGE
2	LOESCHUNG
3	LOESCHUNG
4	LOESCHUNG
5	FELDAENDERUNG
7	FELDAENDERUNG
9	FELDAENDERUNG
10	FELDAENDERUNG
11	FELDAENDERUNG
12	FELDAENDERUNG

Die folgende Datenbeschreibung des Bewegungssatzes zeigt Bedingungs-Namen für das Datenfeld SATZART, welche ZUGANG, ABGANG und AENDERUNG sein können. Das Datenfeld SATZART ist die diesen Bedingungs-Namen zugeordnete Bedingungsvariable.

```
01  BEWEGUNG.
    02  NUMMER  PIC X(5).
    02  SATZART PIC 99.
        88  ZUGANG       VALUE IS 1.
        88  ABGANG       VALUES ARE 2 THRU 4.
        88  AENDERUNG    VALUES ARE 5, 7, 9 THRU 12.
    02  SUMME   PIC 9999V99.
```

Werden die Bedingungsnamen in einer IF-Anweisung benutzt, kann das Feld SATZART auf die Inhalte 1, 2, 3, 4, 5, 7, 9, 10, 11 und 12 abgefragt werden. Im folgenden Beispiel sind beide Möglichkeiten enthalten, mit und ohne Abfrage der Bedingungsnamen.

```
    IF SATZART = 1 GO NEUANLAGE.

    IF ZUGANG   GO NEUANLAGE.

    IF SATZART > 1 AND SATZART < 5 GO LOESCHUNG.

    IF ABGANG   GO LOESCHUNG.

    IF SATZART = 5 OR SATZART = 7 GO FELD-AENDERUNG.
    IF SATZART > 8 AND SATZART < 13 GO FELD-AENDERUNG.

    IF AENDERUNG    GO FELD-AENDERUNG.
```

Der Bedingungsname ZUGANG wird für die Verhältnis-Bedingung SATZART IS EQUAL TO 1 benutzt, um die Steuerung zum Paragraphen NEUANLAGE zu ermöglichen.

Der Bedingungsname ABGANG wird für die Verhältnis-Bedingung SATZART IS GREATER THAN 1 AND SATZART IS LESS THAN 5 benutzt, um die Steuerung zum Paragraphen LOESCHUNG zu ermöglichen.

Die beiden Verhältnis-Bedingungen

```
IF SATZART IS EQUAL TO 5 OR
   SATZART IS EQUAL TO 7
```

und

```
IF SATZART IS GREATER THAN 8 AND
   SATZART IS LESS THAN 13
```

bewirken ggf. dieselbe Steuerung zum Paragraphen FELD-AENDERUNG wie die Abfrage nach dem Bedingungsnamen AENDERUNG.

RENAMES erlaubt das Zusammenfassen von Elementarfeldern zu neuen Feldgruppen, wobei sich die Gruppen überschneiden dürfen. RENAMES kann in allen Sections der DATA DIVISION verwendet werden.

Format:

66 Datenname-1 RENAMES Datenname-2 [{ THROUGH / THRU } Datenname-3].

Die Stufennummer 66 bezeichnet den Beginn einer RENAMES-Definition. Sie muß auf Spalte 8 oder auf einer der folgenden Spalten beginnend geschrieben werden.

Datenname-1 folgt auf die Stufennummer 66 beginnend ab Spalte 12 oder einer der folgenden Spalten. Der Datenname-1 muß wenigstens ein Leerzeichen Abstand zur Stufennummer 66 haben.

Für einen Satz können eine oder mehrere RENAMES-Eintragungen geschrieben werden. Alle RENAMES-Eintragungen für einen Satz müssen unmittelbar nach der Beschreibung des letzten Datenfeldes dieses Satzes folgen.

In der Form "66 Datenname-1 RENAMES Datenname-2" wird für das Feld, das durch Datenname-2 beschrieben wird, der neue Datenname-1 vergeben. Handelt es sich bei Datenname-2 um eine Feldgruppe, so bezeichnet der Datenname-1 ebenfalls eine Feldgruppe. Bezeichnet Datenname-2 ein Elementarfeld, so gilt Datenname-1 ebenfalls als Elementarfeld.

In der Form "66 Datenname-1 RENAMES Datenname-2 THROUGH Datenname-3" bezeichnet Datenname-1 einen Gruppenbegriff, der alle Elementarfelder beginnend mit Datenname-2 und endend mit Datenname-3 einschließt. Handelt es sich bei Datenname-2 und Datenname-3 um Feldgruppen, so bezeichnet Datenname-1 einen Gruppenbegriff, der alle Elementarfelder beginnend mit dem ersten Elementarfeld aus Datenname-2 und endend mit dem letzten Elementarfeld aus Datenname-3 einschließt.

Es folgt ein Beispiel, wie die Felder eines zuvor definierten Satzes durch RENAMES neu gruppiert werden können:

```
01  STAMMSATZ.
    02  KTO-NR            PIC X(6).
    02  UMSATZ-1          PIC 999V99.
    02  UMSATZ-2          PIC 999V99.
    02  UMSATZ-3          PIC 999V99.
    02  GESAMT-UMS        PIC 999V99.
    02  GESAMT-ZAHLG      PIC 9999V99.
    02  ZAHLG-1           PIC 9999V99.
    02  ZAHLG-2           PIC 9999V99.
    66  UMSATZWERT RENAMES UMSATZ-1 THRU GESAMT-UMS.
    66  GESAMTWERTE RENAMES GESAMT-UMS THRU GESAMT-ZAHLG.
    66  ZAHLGWERTE RENAMES GESAMT-ZAHLG THRU ZAHLG-2.
    66  BARZAHLG RENAMES ZAHLG-2.
```

Folgende Regeln gelten für RENAMES:

1. Datenname-2 und Datenname-3 müssen Namen von Elementarfeldern oder Feld-Gruppen des zugehörigen Satzes sein. Sie dürfen nicht identisch sein.

2. Datenname-2 muß in der Satzbeschreibung dem Datennamen-3 vorangehen. Nach irgendeiner Redefinition muß der durch Datennamen-2 beschriebene Bereich vor dem durch Datennamen-3 beschriebenen Bereich beginnen.

3. Datenname-2 und Datenname-3 können qualifiziert werden.

4. Datenname-3 darf keine Unterstufe zu Datenname-2 sein.

5. Eine Stufe 66-Eintragung kann nicht in einer Stufe 66-Eintragung neu benannt werden.

6. Eine Stufe 77, 88 oder 01 kann nicht in einer Stufe 66-Eintragung neu benannt werden.

7. Datenname-1 kann nicht zur Qualifizierung verwendet werden. Datenname-1 wird nur durch die Namen der Stufennummern 01, FD, CD oder SD qualifiziert.

8. Weder Datenname-2 noch Datenname-3 darf eine OCCURS-Klausel enthalten oder Unterstufe eines Datennamens sein, der eine OCCURS-Klausel enthält.

DIE SYNCHRONIZED-KLAUSEL

Die SYNCHRONIZED-Klausel bezeichnet die Angleichung eines elementaren Datenfeldes an die natürlichen Grenzen des Speichers des Computers. Die natürliche Grenze im Speicher des Computers ist ein Wort, das aus zwei Bytes (16 Bits) besteht. Somit gleicht die SYNCHRONIZED-Klausel ein elementares Datenfeld an eine Ganzzahl von Speicherwörtern im Speicher des Computers an.

Format:

```
{SYNCHRONIZED}   [LEFT ]
{SYNC        }   [RIGHT]
```

Das Wort SYNCHRONIZED kann als SYNC abgekürzt werden.

Identifier-1, der Gegenstand der die SYNCHRONIZED-Klausel enthaltenden Datenfeldbeschreibung ist, muß ein elementares Datenfeld sein. Die SYNCHRONIZED-Klausel bewirkt, daß dieses elementare Datenfeld an ganze Speicherwörter angeglichen wird, und zwar derart, daß kein anderes Datenfeld eine der Zeichenpositionen zwischen den äußersten linken und den äußersten rechten Zeichen in einer Ganzzahl von Speicherwörtern einnimmt.

Wird die SYNCHRONIZED LEFT-Klausel angegeben, dann wird das durch den Identifier-1 bezeichnete elementare Datenfeld so im Speicher plaziert, daß das äußerste linke Byte des Datenfeldes an das äußerste linke Byte des nächsten ganzen Speicherwortes angeglichen wird. Enthält das durch Identifier-1 bezeichnete Datenfeld eine ungerade Anzahl von Bytes, dann bleibt das unmittelbar auf das letzte, durch das Datenfeld eingenommene Byte folgende Byte unbenützt. Das Datenfeld MENGE gemäß dem nächsten Beispiel zeigt das Resultat der SYNCHRONIZED LEFT-Klausel.

Wird die SYNCHRONIZED RIGHT-Klausel angegeben, dann wird das durch den Identifier-1 bezeichnete elementare Datenfeld so im Speicher plaziert, daß das äußerste rechte Byte des Datenfeldes an das äußerste rechte Byte einer Ganzzahl von Speicherwörtern angeglichen wird. Enthält das durch Identifier-1 bezeichnete Datenfeld eine ungerade Anzahl von Bytes, dann bleibt das äußerste linke Byte in den ganzen Speicherwörtern unbenützt. Das Datenfeld RCODE im nächsten Beispiel zeigt das Resultat der SYNCHRONIZED RIGHT-Klausel.

```
02  BCODE          PIC X.
02  MENGE          PIC S9(5) COMP-3 SYNC LEFT.
02  RCODE          PIC XXX SYNC RIGHT.
02  SCODE          PIC X.
02  TCODE          PIC 9.
```

WORT		WORT		WORT		WORT		WORT		WORT	
A	x	5 7	4 3	2 -	x	x	B	5	9	B	3
BCODE	nicht benutzt	M E N G E			nicht benutzt	nicht benutzt	R C O D E			SCODE	TCODE

Die SYNCHRONIZED-Klausel ohne das Wort LEFT oder RIGHT entspricht der Klausel SYNCHRONIZED LEFT.

Wenn das einem die SYNCHRONIZED-Klausel enthaltenden Datenfeld vorangehende Datenfeld nicht in dem äußersten rechten Byte eines Speicherwortes endet, dann entsteht ein unbenütztes Byte. Das Datenfeld BCODE in dem vorangehenden Beispiel zeigt ein nicht in dem äußersten rechten Byte eines Speicherwortes endendes Datenfeld. Das auf BCODE folgende Byte wird ein unbenütztes Byte, weil das nächste Datenfeld MENGE ein synchronisiertes Datenfeld ist.

Alle unbenützten Zeichen, die als ein Resultat des Synchronisierens eines elementaren Datenfeldes erzeugt werden, werden mitgezählt

- in der Größe jeder Gruppe, der das elementare Datenfeld zugehört;

- in den Zeichenpositionen, die redefiniert werden, wenn dieses Datenfeld Gegenstand einer REDEFINES-Klausel ist.

Wird die SYNCHRONIZED-Klausel in einer eine OCCURS-Klausel enthaltenden Datenfeldbeschreibung definiert oder wird sie in einer Beschreibung eines Datenfeldes, das einer eine OCCURS-Klausel enthaltenden Datenfeldbeschreibung untergeordnet ist, definiert, dann wird jedes Auftreten des Datenfeldes synchronisiert. Auch werden alle unbenützten Zeichenpositionen für jedes Auftreten des Datenfeldes erzeugt.

DIE USAGE-KLAUSEL

Die USAGE-Klausel bestimmt die Auslegung eines Datenfeldes im Speicher des Computers.

Format:

```
[USAGE IS]  { BIT
              BINARY
              COMPUTATIONAL
              COMP
              COMPUTATIONAL-1
              COMP-1
              COMPUTATIONAL-3
              COMP-3
              COMPUTATIONAL-4
              COMP-4
              COMPUTATIONAL-5
              COMP-5
              COMPUTATIONAL-6
              COMP-6
              DISPLAY
              INDEX
              PACKED-DECIMAL }
```

In der Datendefinition wird die USAGE-Klausel entweder auf Elementarstufe oder Gruppen-Ebene geschrieben. Wird eine USAGE-Klausel auf Gruppen-Ebene angegeben, so gilt diese Klausel für alle untergeordneten Datendefinitionen. Es ist daher nicht notwendig, separate USAGE-Klauseln für die untergeordneten Ebenen zu schreiben. Wird eine USAGE-Klausel auf Gruppen-Ebene angegeben müssen alle der Gruppe untergeordneten Definitionen dieselbe USAGE-Art aufweisen. Wird keine USAGE angegeben, gilt USAGE IS DISPLAY.

COBOL kennt verschiedene Speicherungs-Arten, von denen jede in der DATA DIVISION definiert werden kann. Die PICTURE-, USAGE- und SIGN-Klauseln ermöglichen es dem Compiler zu bestimmen, welche Daten-Art in einer Datenfeldbeschreibung angewandt wird. Die folgende Tabelle faßt die Varianten zusammen, die in der USAGE-Klausel, der SIGN-Klausel und den PICTURE-Datenzeichen möglich sind.

Speicherungsart	Anzahl Bit pro Zeichen	PICTURE		USAGE Klausel	SIGN Klausel
		S	Daten-zeichen		
Alphanumerisch	8	nein	X	wahlweise DISPLAY	nein
Dezimal ohne Vorzeichen	8	nein	9	wahlweise DISPLAY	nein
Ungepackt dezimal mit Vorzeichen rechts oder links angefügt	8	S	9	wahlweise DISPLAY	TRAILING bzw. LEADING SEPARATE
Ungepackt dezimal mit Zonenvorzeichen rechts oder links	8	S	9	wahlweise DISPLAY	LEADING oder wahlweise TRAILING
Gepackt dezimal ohne Vorzeichen	4	nein	9	PACKED DECIMAL oder COMP-1 oder COMP-6 oder COMP	nein
Gepackt dezimal mit Vorzeichen	4	S	9	PACKED DECIMAL oder COMP-3 oder COMP	nein
Binär ohne Vorzeichen	-	nein	9	BINARY oder COMP-5	nein
Binär mit Vorzeichen	-	ja	9	BINARY oder COMP-4	nein
Boolesch	1	nein	1	BIT	nein
Index	-	nein	-	INDEX	nein

Die Usage COMP-1 dient der Kompatibilität zu RM/COS COBOL.

Alphanumerische Zeichen (8-Bit)

Ein 8-Bit alphanumerische Zeichen enthaltendes Datenfeld wird mit dem PICTURE-Datenzeichen X und der Wahlklausel USAGE IS DISPLAY definiert. Diese Datendefinition ordnet für jedes Zeichen in der Zeichenfolge der PICTURE-Klausel 1 Byte (8-Bits) im Speicher zu. Das Datenzeichen X zeigt an, daß das Datenfeld zur alphanumerischen Kategorie gehört.

Im folgenden Beispiel erzeugt jede Datenfeldbeschreibung eine Konstante von 15 alphanumerischen Zeichen, die 15 Bytes des Speichers belegt.

```
77  KOMMENTAR1  PIC X(15) VALUE "NULL BESTELLUNG".
77  KOMMENTAR2  PIC X(15) VALUE "ENDE A-PROGRAMM".
```

Dezimalzahlen ohne Vorzeichen (8-Bit)

Ein Datenfeld, das 8-Bit-Dezimalzahlen ohne Vorzeichen enthält, wird mit dem PICTURE-Datenzeichen 9 und der Wahlklausel USAGE IS DISPLAY definiert. Diese Datendefinition ordnet für jede Ziffer in der PICTURE-Zeichenfolge 1 Byte (8-Bits) im Speicher zu. Das Datenzeichen 9 in der PICTURE-Klausel zeigt an, daß das Datenfeld zur numerischen Kategorie gehört.

Im folgenden Beispiel erzeugt jede Datenfeldbeschreibung eine Konstante von fünf numerischen Zeichen, die fünf Bytes des Speichers belegt.

```
02  MENGEA      PIC 9(5) DISPLAY VALUE IS 14692.
02  MENGEB      PIC 9(5) VALUE IS 329.
```

1 4 6 9 2	0 0 3 2 9
MENGEA	MENGEB

Ungepackte Dezimalzahlen mit Vorzeichen rechts oder links angefügt (8-Bit)

Ein Datenfeld, das 8-Bit-Dezimalzahlen mit rechts oder links angefügten Vorzeichen enthält, wird mit dem PICTURE-Datenzeichen 9, der Wahlklausel USAGE IS DISPLAY, der notwendigen Klausel SIGN IS TRAILING (LEADING) SEPARATE CHARACTER und dem Zeichen "S" an der linken Seite der PICTURE-Zeichenfolge definiert. Diese Datendefinition ordnet für jedes Datenzeichen "9" und für das Zeichen "S" in der Zeichenfolge der PICTURE-Klausel 1 Byte (8 Bits) im Speicher zu. Obwohl das "S" immer auf der linken Seite der PICTURE-Zeichenfolge geschrieben ist, steht dieses Zeichen im Speicher an der äußersten rechten oder linken 8-Bit-Zeichenposition (Byte) des Datenfeldes. Das Datenzeichen 9 in der PICTURE-Klausel zeigt an, daß das Datenfeld zur numerischen Kategorie gehört.

```
03  MENGE1      PIC S9(5) DISPLAY TRAILING SEPARATE
                           VALUE IS -493.
03  MENGE2      PIC S9(5) TRAILING SEPARATE VALUE IS 6000.
```

0 0 4 9 3 -	0 6 0 0 0 +
MENGE1	MENGE2

```
03  MENGE3      PIC S9(5) DISPLAY LEADING SEPARATE
                           VALUE IS -496.
03  MENGE4      PIC S9(5) LEADING SEPARATE VALUE IS 414.
```

- 0 0 4 9 6	+ 0 0 4 1 4
MENGE3	MENGE4

Wird weder TRAILING noch LEADING angegeben, gilt TRAILING als Default.

Im Speicher des Computers weist ein negatives ungepacktes dezimales Datenfeld mit Vorzeichen ein Hexadezimal 2D (das Zeichen -) in dem äußersten rechten oder linken Byte auf; ein positives ungepacktes dezimales Datenfeld weist eine von 2D verschiedene hexadezimale Kombination in den äußersten rechten oder linken vier Bits des äußersten rechten bzw. linken Bytes auf. Ein positives Zeichen, das auf die Durchführung einer COBOL-Anweisung hin erzeugt wurde, ist jedoch ein Hexadezimal 2B (Zeichen +). Zur Darstellung wird im obigen Beispiel das Symbol + in dem äußersten rechten Byte eines positiven Feldes und das Symbol - in dem äußersten rechten Byte eines negativen Feldes gezeigt.

Ungepackte Dezimalzahlen mit Zonenvorzeichen rechts oder links (8-Bit)

Ein Datenfeld, das 8-Bit-Dezimalzahlen enthalten soll, aus deren rechter oder linker äußerer Ziffer das Vorzeichen verschlüsselt hervorgeht, wird mit dem PICTURE-Datenzeichen "9", der Wahlklausel USAGE IS DISPLAY, der Wahlklausel SIGN IS TRAILING (LEADING) und dem Zeichen "S" auf der linken Seite der PICTURE Zeichenfolge definiert. Diese Datendefinition ordnet für jedes Datenzeichen "9" in der Zeichenfolge der PICTURE-Klausel 1 Byte (8-Bits) im Speicher zu. Da das Zeichen "S" mit dem äußersten rechten oder linken Datenzeichen im Datenfeld kombiniert ist, nimmt es kein Byte im Speicher ein. Das Datenzeichen "9" zeigt an, daß das Datenfeld zur numerischen Kategorie gehört.

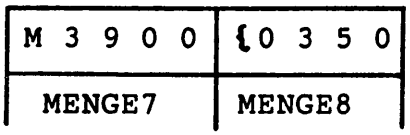
Die folgende Tabelle zeigt die Zeichen, die in COBOL zur Kombination von Vorzeichen und Ziffer in einem dezimalen Datenfeld verwendet werden.

erste/ letzte Ziffer	0	1	2	3	4	5	6	7	8	9
positiv	{(hex7B)	A od. 1	B od. 2	C od. 3	D od. 4	E od. 5	F od. 6	G od. 7	H od. 8	I od. 9
negativ	}(hex7D)	J	K	L	M	N	O	P	Q	R

```
02  MENGE5  PIC S9(5) DISPLAY TRAILING VALUE IS -435.
02  MENGE6  PIC S9(5) VALUE IS 3674.
```

0 0 4 3 N	0 3 6 7 D
MENGE5	MENGE6

```
02  MENGE7  PIC S9(5) DISPLAY LEADING VALUE IS -43900.
02  MENGE8  PIC S9(5) LEADING VALUE IS 350.
```



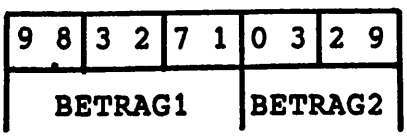
Im vorigen Beispiel enthält die Datenfeldbeschreibung für MENGE6 keine USAGE-Klausel; somit wird die Klausel USAGE IS DISPLAY angenommen. Die Datenfeldbeschreibung für MENGE6 enthält auch keine SIGN-Klausel. Eine fehlende SIGN-Klausel läßt den Compiler die Klausel SIGN IS TRAILING annehmen.

Gepackte Dezimalzahlen ohne Vorzeichen (4-Bit)

Ein Datenfeld, das gepackte 4-Bit-Dezimalzahlen ohne Vorzeichen enthält, wird mit dem PICTURE-Datenzeichen "9" und der Pflichtklausel USAGE IS COMPUTATIONAL-6 definiert. COMPUTATIONAL-6 kann als COMP-6 abgekürzt werden. Diese Datendefinition ordnet für jeweils zwei Ziffern in der PICTURE-Zeichenfolge 1 Byte (8-Bits) im Speicher zu. Das Datenzeichen "9" in der PICTURE-Klausel zeigt an, daß das Datenfeld zur numerischen Kategorie gehört.

Im folgenden Beispiel erzeugt die erste Datenfeldbeschreibung eine Konstante von sechs numerischen Zeichen ohne Vorzeichen, die drei Bytes des Speichers belegt. Die zweite Datenfeldbeschreibung erzeugt eine Konstante von vier numerischen Zeichen ohne Vorzeichen, die zwei Bytes des Speichers belegt.

```
03  BETRAG1  PIC 9(6) COMP-6 VALUE IS 983271.
03  BETRAG2  PIC 999  COMP-6 VALUE IS 329.
```



Bei einem Datenfeld gepackter Dezimalzahlen ohne Vorzeichen mit ungerader Zeichenzahl erweitert der Compiler automatisch das Datenfeld auf eine gerade Anzahl gepackter, dezimaler Zeichen. Folglich sind dem Datenfeld BETRAG2 gemäß dem vorangehenden Beispiel zwei Bytes des Speichers zugeordnet, wobei in der äußersten linken 4-Bit-Position eine Null steht.

Das Wort COMPUTATIONAL-6 bzw. COMP-6 kann durch die Worte COMPUTATIONAL bzwl COMP oder COMPUTATIONAL-1 bzw. COMP-1 oder PACKED DECIMAL ersetzt werden. Die Kombination eines PICTURE-Datenzeichen "9" mit der Klausel USAGE IS COMPUTATIONAL hat ein Datenfeld gepackter 4-Bit-Dezimalzahlen ohne Vorzeichen zur Folge. Die Datenfeldbeschreibungen für BETRAG1 und BETRAG2 könnten daher auch folgendermaßen geschrieben werden:

```
03  BETRAG1  PIC 9(6) COMP VALUE IS 983271.
03  BETRAG2  PIC 999  COMP VALUE IS 329.
```

Gepackte Dezimalzahlen mit Vorzeichen (4-Bit)

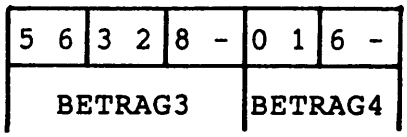
Ein Datenfeld, das gepackte Dezimalzahlen mit Vorzeichen enthält, wird mit dem PICTURE Datenzeichen "9", der Pflichtklausel USAGE IS COMPUTATIONAL-3 und einem Zeichen "S" auf der linken Seite der PICTURE Zeichenfolge definiert. Das Wort COMPUTATIONAL-3 kann auf COMP-3 abgekürzt werden.

Diese Datendefinition bewirkt, daß eine Speicherzuordnung auf der Basis 1 Byte für je zwei Zeichen in der PICTURE-Zeichenfolge erfolgt. Obwohl das "S" immer auf der linken Seite der PICTURE-Zeichenfolge geschrieben wird, steht es im Speicher in der äußersten rechten 4-Bit-Zeichenposition des Datenfeldes. Das Datenzeichen "9" zeigt an, daß das Datenfeld zur numerischen Kategorie gehört.

Im folgenden Beispiel erzeugt die erste Datenfeldbeschreibung eine Konstante von sechs gepackten numerischen Zeichen mit Vorzeichen, die drei Bytes des Speichers belegt. Die zweite Datenfeldbeschreibung erzeugt eine Konstante von vier gepackten numerischen Zeichen mit Vorzeichen, die zwei Bytes des Speichers belegen.

```
03  BETRAG3      PIC S9(5) COMP-3 VALUE IS -56328.
03  BETRAG4      PIC S99   COMP-3 VALUE IS -16.
```

(Speicher-Darstellung siehe nächste Seite!)



Bei einem Datenfeld gepackter Dezimalzahlen mit Vorzeichen und ungerader Zeichenzahl erweitert der Compiler automatisch das Datenfeld auf eine gerade Anzahl gepackter, dezimaler Zeichen mit Vorzeichen. Somit sind dem Datenfeld BETRAG4 gemäß dem vorangehenden Beispiel zwei Bytes des Speichers zugeordnet, wobei in der äußersten linken 4-Bit-Position eine 0 steht.

Das Wort COMPUTATIONAL-3 bzw. COMP-3 kann durch das Wort COMPUTATIONAL bzw. COMP oder die Worte PACKED DECIMAL ersetzt werden. Die Kombination eines PICTURE-Datenzeichens "9", der Klausel USAGE IS COMPUTATIONAL und dem Zeichen "S" auf der linken Seite der PICTURE-Zeichenfolge hat ein Datenfeld gepackter 4-Bit-Dezimalzahlen mit Vorzeichen zur Folge. Die Datenfeldbeschreibungen für BETRAG3 und BETRAG4 könnten daher auch folgendermaßen geschrieben werden:

```
03  BETRAG3     PIC S9(5) COMP VALUE IS -56328.
03  BETRAG4     PIC S99   COMP VALUE IS -16.
```

Im Speicher des Computers weist ein negatives gepacktes dezimales Datenfeld mit Vorzeichen ein Hexadezimal-Zeichen D in seiner äußersten rechten 4-Bit-Zeichenposition auf; ein positives gepacktes dezimales Datenfeld mit Vorzeichen weist ein von D verschiedenes Hexadezimal-Zeichen in seiner äußersten rechten 4-Bit-Zeichenposition auf. Ein positives Zeichen, das auf die Durchführung einer COBOL-Anweisung hin erzeugt wurde, ist jedoch ein Hexadezimal B. Zur Darstellung wurde hier das Symbol + in der äußersten rechten 4-Bit-Zeichenposition eines positiven Datenfeldes und das Symbol - in der äußersten rechten 4-Bit-Zeichenposition eines negativen Feldes gezeigt.

Binärzahlen ohne Vorzeichen

Ein Binärzahlen ohne Vorzeichen enthaltendes Datenfeld wird mit dem PICTURE-Datenzeichen "9" und der Pflichtklausel USAGE IS COMPUTATIONAL-5 definiert. Das Wort COMPUTATIONAL-5 kann auf COMP-5 abgekürzt oder durch das Wort BINARY ersetzt werden. Das Datenzeichen "9" zeigt an, daß das Datenfeld zur numerischen Kategorie gehört.

Nach dieser Datendefinition erfolgt eine Speicherzuordnung für ein binäres Feld ohne Vorzeichen nach der Häufigkeit des Auftretens des Datenzeichens "9" in der PICTURE-Zeichenfolge. Die folgende Tabelle gibt eine Zusammenfassung der einem binären Datenfeld ohne Vorzeichen zugeordneten Speicher-Bytes.

Anzahl 9-Symbole	Anzahl zugeordneter Speicher-Bytes
1 oder 2	1
3 oder 4	2
5 oder 6	3
7, 8 oder 9	4
10, 11 oder 12	5
13 oder 14	6
15 oder 16	7
17 oder 18	8

Im folgenden Beispiel erzeugt die Datenfeldbeschreibung eine Konstante der Binärzahlen, die, weil die PICTURE-Zeichenfolge drei Datenzeichen "9" aufweist, zwei Bytes des Speichers belegt.

```
02  CODE-A      PIC 999  COMP-5  VALUE IS 47.
```

0 0 0 0 0 0 0 0	0 0 1 0 1 1 1 1	= 47 binär

Binärzahlen mit Vorzeichen

Ein Binärzahlen mit Vorzeichen enthaltendes Datenfeld wird mit dem PICTURE-Datenzeichen "9", der Pflichtklausel USAGE IS COMPUTATIONAL-4 und einem Zeichen "S" auf der linken Seite der PICTURE-Zeichenfolge definiert. Das Wort COMPUTATIONAL-4 kann auf COMP-4 abgekürzt oder durch das Wort BINARY ersetzt werden. Das Datenzeichen "9" zeigt an, daß das Datenfeld zur numerischen Kategorie gehört.

Die Speicherzuordnung erfolgt nach folgenden Regeln:

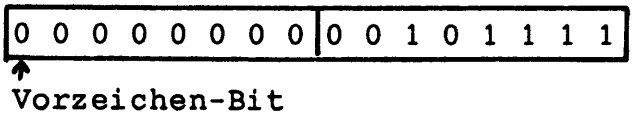
1. Tritt in der PICTURE-Zeichenfolge das Zeichen "9" ein- bis viermal auf, dann werden dem binären Wert zwei Bytes des Speicher zugeordnet.

2. Tritt in der PICTURE-Zeichenfolge das Zeichen "9" fünf- bis neunmal auf, dann werden dem binären Wert vier Bytes des Speichers zugeordnet.

3. Tritt in der PICTURE-Zeichenfolge das Zeichen "9" zehn- bis achtzehnmal auf, dann werden dem binären Wert acht Bytes des Speichers zugeordnet.

Im folgenden Beispiel erzeugt die Datenfeldbeschreibung eine binäre Konstante, die zwei Bytes des Speichers belegt.

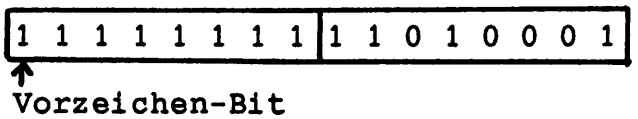
```
02 CODE1 PIC S999 COMP-4 VALUE IS 47.
```



= + 47 als Binärzahl dargestellt

Das äußerste linke Bit des Datenfeldes ist das Vorzeichen-Bit. Ist das Vorzeichen-Bit 0, dann ist das Datenfeld positiv. Ist das Vorzeichen-Bit 1, dann ist das Datenfeld negativ. Um den Wert der negativen Binärzahl zu bestimmen, muß das zweite Komplement des Binärwertes im Datenfeld ermittelt werden. Das folgende Beispiel zeigt die Binärdarstellung eines negativen Wertes 47.

```
02 CODE2 PIC S999 COMP-4 VALUE IS -47.
```



= - 47 als Binärzahl dargestellt

Der obige Inhalt des Datenfeldes CODE2 ergibt sich durch folgende Schritte:

- Ermittlung des ersten Komplements der Binärzahl 47

0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 = 47 als Binärzahl dargestellt

1 1 1 1 1 1 1 1 1 1 0 1 0 0 0 0 = erstes Komplement von 47

(weiter auf der folgenden Seite!)

- Addition einer 1 zum ersten Komplement

```
  1 1 1 1 1 1 1 1  1 1 0 1 0 0 0 0   = erstes Komplement
                                       von 47
+                                  1
  1 1 1 1 1 1 1 1  1 1 0 1 0 0 0 1   = zweites Komplement
                                       von 47
```

Boolesche Daten

Ein boolesche Daten enthaltendes Datenfeld wird mit dem PICTURE-Datenzeichen "1" und der Pflichtklausel USAGE IS BIT definiert. Jedes Datenzeichen "1" in der PICTURE-Zeichenfolge stellt eine boolesche Position dar, die entweder vom booleschen Zeichen "0" oder "1" eingenommen wird.

Die PICTURE-Zeichenfolge eines booleschen Datenfeldes muß eine durch acht teilbare Anzahl 1-Symbole enthalten. Jeweils acht Datenzeichen "1" in der PICTURE-Zeichenfolge ist ein Byte des Speichers zugeordnet. Die maximale Größe eines booleschen Datenfeldes ist 18 Bytes oder 144 Bits. Das Datenzeichen "1" zeigt an, daß das Datenfeld zur booleschen Kategorie gehört.

Im folgenden Beispiel erzeugt die Datenfeldbeschreibung eine Konstante boolescher Daten, die, weil die PICTURE-Zeichenfolge acht Datenzeichen "1" aufweist, ein Byte des Speichers belegt.

```
02  CODE-B      PIC 1(8) BIT VALUE IS B"00010110".
```

```
          0 0 0 1 0 1 1 0    = ASCII Steuerzeichen SYN
```

DIE USAGE IS INDEX-KLAUSEL

Ein mit der Klausel USAGE IS INDEX beschriebenes elementares Datenfeld wird Index-Datenfeld genannt. Ein Index-Datenfeld ist ein Speicherplatz, in dem der Wert eines Index vorübergehend ohne Umwandlung (in Datenform) zwischengespeichert werden kann. Hier das allgemeine Format der USAGE IS INDEX-Klausel:

```
[USAGE IS] INDEX
```

In COBOL besteht ein Index-Datenfeld aus vier Bytes, die auf Wortgrenze synchronisiert sind.

Die folgenden Datenfeldbeschreibungen definieren eine Tabelle mit fünf Tabellenfeldern, auf die mit Hilfe eines

Index, genannt TAB1-IND, zugegriffen wird.

```
01  TAB1.
    02  POSTEN  OCCURS 5 TIMES  INDEXED BY TAB1-IND.
        03  KONTO-NR     PIC XXXXX.
        03  SALDO        PIC 9999V99.
```

Um den Inhalt des Index (TAB1-IND) vorübergehend zu speichern, muß der Programmierer ihn in ein Index-Datenfeld übertragen. Das Index-Datenfeld ist in der WORKING-STORAGE SECTION wie folgt definiert:

```
77  IND-FELD USAGE IS INDEX.
```

Eine SET-Anweisung muß benutzt werden, um den Inhalt des Index (TAB1-IND) in ein Index-Datenfeld (IND-FELD) zu übertragen:

```
    SET IND-FELD TO TAB1-IND.
```

Will der Programmierer den Index (TAB1-IND) auf den Inhalt des Index-Datenfeldes (IND-FELD) zurücksetzen, müßte folgende SET-Anweisung ausgeführt werden:

```
    SET TAB1-IND TO IND-FELD.
```

Wenn eine Datengruppe durch die Klausel USAGE IS INDEX beschrieben ist, sind alle elementaren Datenfelder innerhalb der Gruppe Index-Datenfelder. Die Gruppe selbst gilt nicht als Index-Datenfeld. In den nachfolgenden Datenfeldbeschreibungen sind FELD-A, FELD-B und FELD-C jeweils Index-Datenfelder.

```
01  GRUPPE USAGE IS INDEX.
    02  FELD-A.
    02  FELD-B.
    02  FELD-C.
```

Bei USAGE IS INDEX-Feldern dürfen die Klauseln SYNCHRONIZED, JUSTIFIED, PICTURE, VALUE und BLANK WHEN ZERO nicht angegeben werden.

Auf ein Index-Datenfeld kann nur in einer SET-Anweisung, einer SEARCH-Anweisung, einer Vergleichsbedingung, der USING-Angabe einer PROCEDURE DIVISION-Überschrift oder der USING-Angabe einer CALL-Anweisung Bezug genommen werden. Ein Index-Datenfeld kann Teil einer Gruppe sein, auf die durch eine MOVE-Anweisung oder eine Input-Output Operation in der PROCEDURE DIVISION Bezug genommen wird.

DIE VALUE-KLAUSEL

In der FILE-SECTION ist die VALUE-Klausel nicht vorgesehen. In der LINKAGE SECTION ist die VALUE-Klausel nur bei Stufen-Nr. 88 vorgesehen. Die Klausel VALUE definiert den Anfangsinhalt von Feldern in der WORKING-STORAGE SECTION, der GLOBAL SECTION und der COMMUNICATION SECTION.

Formate:

1. VALUE IS Literal

2. { VALUE IS / VALUES ARE } Literal-2 [{ THROUGH / THRU } Literal-3] ...

Durch die Klausel VALUE nimmt das Datenfeld den angegebenen Inhalt zu Beginn der Ausführung des Objektprogrammes an. Wird die VALUE-Klausel bei einer Datenfeldbeschreibung weggelassen, dann ist der Anfangswert dieses Feldes nicht voraussagbar. Das Literal kann ein numerisches, ein nichtnumerisches Literal, eine figurative Konstante, ein hexadezmales oder ein boolesches Literal sein.

Die VALUE-Klausel darf den anderen Klauseln der Definition nicht widersprechen.

Ist die Kategorie des beschriebenen Datenfeldes numerisch, muß das Literal in der VALUE-Klausel numerisch sein. Der Wert des Literals wird auf die Dezimalseparator-Angabe der PICTURE-Zeichenfolge ausgerichtet. Überzählige Ziffern links und rechts gehen verloren. Ist der Wert kürzer als das Datenfeld, wird das Feld mit Nullen aufgefüllt. Das Literal darf jedoch keinen Wert darstellen, der ein Abschneiden der Ziffern außer Null erforderlich macht. Ein numerisches Literal mit Vorzeichen muß eine numerische PICTURE-Zeichenfolge mit Vorzeichen zugeordnet haben.

Ist die Kategorie des beschriebenen Datenfeldes alphabetisch, alphanumerisch editiert oder numerisch editiert, muß das Literal in der VALUE-Klausel ein nicht-numerisches Literal sein. Der Wert des Literals wird linksbündig ins Datenfeld gebracht. Ist er kürzer als das Feld, wird es rechts mit Leerzeichen aufgefüllt. Das Literal darf nicht länger sein als das Datenfeld.

Ist die Kategorie des beschriebenen Datenfeldes alphanumerisch, muß das Literal in der VALUE-Klausel entweder ein nicht-numerisches oder ein hexadezimales Literal sein. Der Wert des nicht-numerischen Literals wird linksbündig ins Datenfeld gebracht. Ist er kürzer als das Feld, wird es rechts mit Leerzeichen aufgefüllt.

Der Wert des hexadezimalen Literals wird linksbündig ins Datenfeld gebracht. Ist er kürzer als das Feld, wird es rechts mit Nullen aufgefüllt. Die Anzahl von Zeichen im nicht-numerischen oder hexadezimalen Literal darf die in der PICTURE-Klausel angezeigte Länge nicht überschreiten.

Gehört das beschriebene Datenfeld der booleschen Kategorie an, muß das Literal in der VALUE-Klausel ein boolesches Literal sein. Der Wert des booleschen Literals wird linksbündig ins Datenfeld gebracht. Ist er kürzer als das Feld, wird es rechts mit 0-Bits aufgefüllt. Die Anzahl von Zeichen im booleschen Literal darf die in der PICTURE-Klausel angezeigte Länge nicht überschreiten.

VALUE ist bei Datenbereichen variabler Länge nicht erlaubt. Das Format 2 ist nur für Stufen-Nr.-88-Definitionen vorgegesehen (siehe hierzu auch das nächste Kapitel).

Die BLANK WHEN ZERO Klausel und die JUSTIFIED-Klausel werden nicht beachtet.

Die VALUE-Klausel darf weder in einer eine REDEFINES-Klausel enthaltenden Datenfeldbeschreibung noch in einer Beschreibung aufgeführt sein, die einer eine REDEFINES-Klausel enthaltenden untergeordnet ist. Ausnahme: Stufen-Nr.-88-Definitionen.

Wird die VALUE-Klausel in einer Beschreibung der Gruppen-Stufe verwendet, muß das Literal eine figurative Konstante, ein nicht-numerisches Literal oder ein hexadezimales Literal sein. Somit wird dem Gruppenbereich ohne Rücksicht auf die Datenfeldbeschreibung untergeordneter Datenfelder ein Wert zugeordnet. Die VALUE-Klausel darf in den untergeordneten Stufen innerhalb der mit einer VALUE-Klausel definierten Gruppe nicht angegeben sein.

Die VALUE-Klausel darf in der Beschreibung einer Gruppen-Stufe nicht auftreten, wenn Elementarfelder innerhalb der Gruppe Datenfeldbeschreibungen mit der JUSTIFIED-Klausel, der SYNCHRONIZED-Klausel oder einer USAGE Klausel anders als USAGE IS DISPLAY haben.

Ein Beispiel einer VALUE-Klausel für ein numerisches Literal:

```
02  P-FELD      PIC 999V99 VALUE IS 36,7.
```

P-FELD ist initialisiert als 0 3 6^7 0

Hier ein Beispiel einer VALUE-Klausel für eine figurative Konstante:

```
02  ZAHL-1  PIC 9(5) VALUE IS ZERO.
```

ZAHL-1 ist initialisiert als 0 0 0 0 0

Hier ein Beispiel einer VALUE-Klausel für ein alphanumerisches Literal:

```
02  FELD-A  PIC X(8) VALUE IS "BERICHT".
```

FELD-A ist initialisiert als B E R I C H T ϕ

Nun ein Beispiel einer VALUE-Klausel für ein alphanumerisches Literal in einem numerisch editiert aufbereiteten Feld:

```
02  E-FELD  PIC $999,99 VALUE IS "$123,40".
```

E-FELD ist initialisiert als $ 1 2 3 , 4 0

Hier ein Beispiel einer VALUE-Klausel für ein hexadezimales Literal in einem alphanumerischen Feld:

```
03  HEXCODE  PIC XXXX VALUE IS H"3A4F7A3".
```

HEXCODE ist initialisiert als 3A 4F 7A 30

Jetzt ein Beispiel einer VALUE-Klausel für ein boolesches Literal:

```
02  BITCODE  PIC 1(16) BIT VALUE IS B"0111011110001000".
```

BITCODE ist initialisiert als:

0 1 1 1 0 1 1 1 1 0 0 0 1 0 0 0

DIE WORKING-STORAGE SECTION

Die WORKING-STORAGE SECTION beschreibt intern produzierte und verarbeitete Datensätze und unabhängige Datenfelder, welche nicht zu externen Dateien gehören. Diese Section beschreibt auch Datenfelder (Konstanten genannt), denen im Programm vom Programmierer Anfangsinhalte zugeteilt wurden, die sich während der Ausführung des Objekt-Programmes nicht ändern.

Die WORKING-STORAGE SECTION folgt unmittelbar auf die letzte Eintragung in die FILE SECTION, wenn eine FILE SECTION vorhanden ist. Fehlt die FILE SECTION, dann folgt die WORKING-STORAGE SECTION unmittelbar auf die DIVISION-Überschrift für die DATA DIVISION.

Die erste Eintragung in dieser SECTION ist die SECTION-Überschrift. Diese Überschrift setzt sich zusammen aus den reservierten Wörtern WORKING-STORAGE SECTION, auf die ein Punkt und dann ein Leerzeichen folgt.

Auf die SECTION-Überschrift folgen unabhängige WORKING-STORAGE-Datenfelder und WORKING-STORAGE-Satzbeschreibungen. Ein unabhängiges WORKING-STORAGE-Feld ist ein Datenfeld ohne Unterteilungen, das durch die Stufennummer 77 gekennzeichnet ist. Eine WORKING-STORAGE-Satzbeschreibung ist durch die Stufennummer 01 gekennzeichnet. Unabhängige WORKING-STORAGE-Felder und WORKING-STORAGE-Satzbeschreibungen können in der WORKING-STORAGE SECTION in jeder beliebigen Reihenfolge auftreten.

Format:

```
[ WORKING-STORAGE SECTION.

[ 77 Feldbeschreibung   ]  ...  ]
[ Satzbeschreibung      ]       ]
```

Beschreibung strukturunabhängiger Working-Storage-Felder

Felder in der WORKING-STORAGE SECTION, die ohne Beziehung zueinander sind, sind als unabhängige Elementarfelder (oder die Stufennummer 77 aufweisende Eintragungen) klassifiziert und definiert. Jedes Feld wird mit einer eigenen Datenfeldbeschreibung, die mit der speziellen Stufennummer 77 beginnt, definiert. Die Stufennummer 77 beginnt in Spalte 8, 9, 10 oder 11 des Programmblattes.

Die 77-Datenfeldbeschreibung für ein nicht unterteiltes Arbeitsfeld besteht aus einer Stufennummer (77), einem Identifier und der PICTURE-Klausel. Ist das Datenfeld mit der Stufennummer 77 ein Index-Datenfeld, dann wird die PICTURE-Klausel weggelassen und die USAGE IS INDEX-Klausel wird in der Datenfeldbeschreibung verwendet. Andere Klauseln der Datenbeschreibung sind in der die Stufennummer 77 aufweisenden Datenfeldbeschreibung möglich; sie ergänzen, wenn nötig, die Feldbeschreibung.

Die auf Identifier-1 folgenden Klauseln können in jeder beliebigen Reihenfolge auftreten, mit der Ausnahme jedoch, daß die REDEFINES-Klausel, falls verwendet, unmittelbar auf den Identifier-1 folgen muß. Die unabhängige Datenfeldbeschreibung muß durch einen Punkt nach der letzten Klausel beendet werden. Einer Stufen-Nummer 77 kann eine Bedingungsnamen-Beschreibung (Stufen-Nummer 88) folgen.

Anstelle der Stufen-Nr. 77 kann jederzeit auch die Stufen-Nr. 01 verwendet werden.

Working-Storage-Satzbeschreibung

Haben zwei oder mehr Datenfelder in der WORKING-STORAGE SECTION eine Beziehung zueinander, dann werden sie in einem Satz zusammengefaßt. Definitionen für einen WORKING-STORAGE-Satz richten sich nach den Regeln und Formaten der Satzbeschreibung in der FILE SECTION. Satzbeschreibungen beginnen mit einer Stufennummer 01. In der WORKING-STORAGE SECTION zeigt jede Stufennummer 01 einen neuen Satzbereich im Speicher an, wenn es sich nicht um eine REDEFINES-Klausel handelt. Satzbeschreibungen können auch Definitionen mit der Stufennummer 66 bzw. 88 enthalten.

HINWEIS: Pflichtklauseln für Definitionen in der WORKING-STORAGE SECTION sind: Stufen-Nr. 01 oder 77, Datenname (Identifier, Data-Name), PICTURE- oder USAGE IS INDEX-Klausel. Weitere Klauseln können verwendet werden. Siehe hierzu die Beschreibung der Datendefinition in der FILE SECTION.

DIE LINKAGE SECTION

Sie ist nur in Programmen zulässig und sinnvoll, die von einem anderen Programm mittels einer CALL-Anweisung aufgerufen werden. In ihr werden solche Datenbereiche definiert, die sowohl vom rufenden als auch vom gerufenen Programm angesprochen werden sollen.

Hierzu ist eine weitere Voraussetzung die, daß die CALL-Anweisung im rufenden Programm eine USING-Klausel aufweist, als auch in der PROCEDURE DIVISION-Überschrift des gerufenen Programms eine USING-Klausel vorhanden ist.

Durch die Definitionen in der LINKAGE SECTION wird kein Speicherplatz reserviert, da die Originalfelder sich im rufenden Programm befinden. Durch die Definitionen in der LINKAGE SECTION werden dem gerufenen Programm bei dessen Start lediglich die Adressen der Originalfelder im rufenden Programm zur Verfügung gestellt. Eine Ausnahme hiervon stellen lediglich Tabellen-Indices dar: Diese werden in jedem Programm durch die OCCURS-Klausel separat gebildet.

Ansonsten ist der Aufbau der LINKAGE SECTION derselbe wie bei der WORKING STORAGE SECTION. Die Definition der Daten erfolgt wie in diesem Buch unter dem FILE SECTION-Kapitel beschrieben.

Satznamen und die Namen von unabhängigen Feldern müssen eindeutig sein, denn sie können nicht qualifiziert werden.

Jeder der im gerufenen Programm unter PROCEDURE DIVISION USING... genannte Datenbereich muß in dessen LINKAGE SECTION mit Stufen-Nr. 01 oder ggf. 77 definiert sein.

DIE GLOBAL SECTION

Ein Objekt-Programm kann von mehreren Terminals desselben Computer-Systems gestartet werden, das aber nur einmal im Speicher steht. Datenfelder in einem Objekt-Programm, die nicht verändert werden, können in der GLOBAL SECTION der DATA DIVISION definiert werden und unterliegen dann keiner Replizierung. Datenfelder, die in irgendeinem anderen Bereich der DATA DIVISION definiert sind, werden dagegen mehrmals gespeichert.

Die GLOBAL SECTION beschreibt Sätze und strukturunabhängige Datenfelder, die allen beteiligten Programmen zugänglich sind, wenn ein zweiter Benutzer die Ausführung desselben Objekt-Programmes verlangt. Somit werden GLOBAL-Datenfelder nur einmal im Speicher des Computers reserviert.

Die GLOBAL SECTION muß die letzte SECTION in der DATA DIVISION des COBOL Ursprungs-Programmes darstellen. Die erste Eintragung in der GLOBAL SECTION ist die Section-Überschrift. Diese Überschrift setzt sich zusammen aus den reservierten Wörtern GLOBAL SECTION, gefolgt von einem Punkt und einem Leerzeichen.

Auf die Section-Überschrift folgen unabhängige GLOBAL-Datenfelder und GLOBAL-Satzbeschreibungen. Ein unabhängiges GLOBAL-Datenfeld ist ein Datenfeld ohne Unterteilungen, das an der Stufennummer 77 zu erkennen ist. Eine GLOBAL-Satzbeschreibung wird durch die Stufennummer 01 angezeigt. Unabhängige GLOBAL-Felder und GLOBAL-Sätze können in der GLOBAL SECTION in jeder beliebigen Reihenfolge auftreten.

Format:

```
[GLOBAL SECTION.
 [77 Feldbeschreibung  ]                  ...]
 [Satzbeschreibung     ]
```

Beschreibung strukturunabhängiger Global-Datenfelder

Datenfelder in der GLOBAL SECTION, die keine Beziehung zueinander haben, werden als unabhängige Elementarfelder (oder die Stufennummer 77 aufweisende Eintragungen) klassifiziert und definiert. Jedes Feld wird in einer mit der speziellen Stufennummer 77 beginnenden separaten Datenfeldbeschreibung definiert. Die Stufennummer 77 beginnt in Spalte 8, 9, 10 oder 11 des Programmblattes.

Die Datenfeldbeschreibung für ein nicht unterteiltes Arbeitsfeld besteht aus einer Stufennummer (77), einem Identifier (Datenname) und der PICTURE-Klausel. Ist das Datenfeld mit der Stufennummer 77 ein Index-Datenfeld, dann wird die PICTURE-Klausel weggelassen und die USAGE IS INDEX Klausel wird in der Datenfeldbeschreibung verwendet. Auch andere Klauseln der Datenbeschreibung sind mit der Stufennummer 77 möglich; sie vervollständigen, wenn nötig, die Beschreibung des Feldes. Siehe hierzu die Beschreibung der Datendefinition in der FILE SECTION.

Das Format der Beschreibung unabhängiger GLOBAL-Datenfelder entspricht dem Format der unabhängigen WORKING-STORAGE-Datenfeldbeschreibung.

Die auf Identifier-1 folgenden Klauseln können in jeder beliebigen Reihenfolge auftreten, mit der Ausnahme jedoch, daß die REDEFINES-Klausel, falls verwendet, unmittelbar auf den Identifier-1 folgen muß. Die unabhängige Datenfeldbeschreibung muß durch einen Punkt nach der letzten Klausel beendet werden.

Global-Satzbeschreibung

Haben zwei oder mehr Datenfelder in der GLOBAL SECTION eine Beziehung zueinander, dann werden sie in einem Satz gruppiert. Eintragungen für einen GLOBAL-Satz richten sich nach den Regeln und Formaten der Satzbeschreibung in der FILE SECTION. Satzbeschreibungen beginnen mit einer Stufennummer 01. In der GLOBAL SECTION zeigt jede Stufennummer 01 einen neuen Satzbereich im Speicher an, es sei denn, es handelt sich um eine REDEFINES-Klausel.

VALUE-Klausel in der Global Section

Da Datenfelder in der GLOBAL SECTION im Computer nicht wiederholt werden, muß die VALUE-Klausel in der Datenfeldbeschreibung jedes elementaren GLOBAL-Datenfeldes vorhanden sein; die einzige Ausnahme davon stellt ein Index-Datenfeld dar, dessen Inhalt zur Zeit des Programmstarts nicht voraussagbar ist. Die VALUE-Klausel definiert die im GLOBAL-Datenfeld gespeicherte Konstante.

DIE COMMUNICATION SECTION

Sie dient zur Definition von Datenbereichen im Programm, welche zur Datenübergabe zwischen dem Programm und dem MCS (Message Control System) und umgekehrt vorgesehen sind. Auf die Überschrift der COMMUNICATION SECTION folgt eine "Communication Description"-Eintragung, beginnend mit CD in Spalte 8, danach ein Datenname sowie eine Reihe von weiteren unabhängigen Klauseln.

Für die Datenübernahme von MCS bestimmen diese Klauseln die sogenannten QUEUES, die Sub-QUEUES, Datum und Uhrzeit des Datenempfanges, den Ursprung der übernommenen Daten, deren Länge, Status und Ende-Symbol sowie einen Datenübernahme-Zähler.

Für die Datenabgabe an MCS werden mittels dieser Klauseln definiert: Datenübergabe-Zähler, Längenbemessungs-Feld, Status- und Fehler-Felder sowie die Bestimmungsorte, wohin die Daten gehen sollen.

Beispiel 1: Über 3 Terminals zu erfassende Daten werden von MCS entgegengenommen und an das Programm weitergereicht.

Beispiel 2: Informationen werden vom Programm über MCS an ein bestimmtes von 3 Terminals gesendet.

Der CD-Eintrag

Er dient zur Definition des Datenbereiches, in dem während der Verarbeitung die Daten bereitgestellt werden zum Austausch zwischen dem Programm und MCS.

Format 1

```
CD cd-name-1

                    [ [[SYMBOLIC QUEUE IS data-name-1]]
                      [SYMBOLIC SUB-QUEUE-1 IS data-name-2]
                      [SYMBOLIC SUB-QUEUE-2 IS data-name-3]
                      [SYMBOLIC SUB-QUEUE-3 IS data-name-4]

                      [MESSAGE DATE IS data-name-5]
                      [MESSAGE TIME IS data-name-6]
                      [SYMBOLIC SOURCE IS data-name-7]
FOR [INITIAL] INPUT   [TEXT LENGTH IS data-name-8]
                      [END KEY IS data-name-9]
                      [STATUS KEY IS data-name-10]
                      [MESSAGE COUNT IS data-name-11]]

                     [data-name-1, data-name-2, data-name-3,
                      data-name-4, data-name-5, data-name-6,
                      data-name-7, data-name-8, data-name-9,
                      data-name-10, data-name-11]                ]
```

Format 2

```
CD cd-name-1 FOR OUTPUT
   [DESTINATION COUNT IS data-name-1]
   [TEXT LENGTH IS data-name-2]
   [STATUS KEY IS data-name-3]

   [DESTINATION TABLE OCCURS integer-1 TIMES
     [INDEXED BY  index-name-1  ... ]]

   [ERROR KEY IS data-name-4]
   [SYMBOLIC DESTINATION IS data-name-5].
```

Format 1:

- Innerhalb eines Programmes darf die INITIAL-Klausel nur in einer CD-Eintragung auftreten. INITIAL darf nicht in einem Programm verwendet werden, das in der PROCEDURE DIVISION-Zeile die USING-Klausel enthält.

- Außer der INITIAL-Klausel können die übrigen Wahlklauseln in jeder beliebigen Folge geschrieben werden.

- Werden keine Wahlklauseln angegeben, muß auf die CD-Zeile eine Datendefinition mit der Stufen-Nr. 01 und 11 Elementarfeldern erfolgen. Bei vorhandenen Wahlklauseln kann auf jede eine 01-Stufen-Datendefinition folgen.

- Für jede Input-Communication-Eintragung wird ein 87 Bytes umfassender Bereich benötigt.

Dieser Bereich kann auf einem der folgenden Wege definiert werden:

- Entweder durch Verwendung der Wahlklauseln in jeder beliebigen Folge, insgesamt 11.

- Oder durch Definition von 11 Datenfeldern in Folge, insgesamt 11.

  Für MCS gliedert sich der 87stellige Datenbereich folgendermaßen:

  SYMBOLIC QUEUE
     Diese Klausel definiert mit Data-Name-1 ein 12stelliges alphanumerisches Elementarfeld, das die Positionen 1 bis 12 im Bereich einnimmt.

SYMBOLIC SUB-QUEUE-1
Diese Klausel definiert mit Data-Name-2 ein 12stelliges alphanumerisches Elementarfeld, das die Positionen 13 bis 24 im Bereich einnimmt.

SYMBOLIC SUB-QUEUE-2
Diese Klausel definiert mit Data-Name-3 ein 12stelliges alphanumerisches Elementarfeld, das die Positionen 25 bis 36 im Bereich einnimmt.

SYMBOLIC SUB-QUEUE-3
Diese Klausel definiert mit Data-Name-4 ein 12stelliges alphanumerisches Elementarfeld, das die Positionen 37 bis 48 im Bereich einnimmt.

MESSAGE DATE
Diese Klausel definiert mit Data-Name-5 ein 6stelliges numerisches Feld ohne Vorzeichen, das die Positionen 49 bis 54 im Bereich einnimmt.

MESSAGE TIME
Diese Klausel definiert mit Data-Name-6 ein 8stelliges numerisches Feld ohne Vorzeichen, das die Positionen 55 bis 62 im Bereich einnimmt.

SYMBOLIC SOURCE
Diese Klausel definiert mit Data-Name-7 ein 12stelliges alphanumerisches Feld, das die Positionen 63 bis 74 im Bereich einnimmt.

TEXT LENGTH
Diese Klausel definiert mit Data-Name-8 ein 4stelliges numerisches Feld ohne Vorzeichen, das die Positionen 75 bis 78 im Bereich einnimmt.

END KEY
Diese Klausel definiert mit Data-Name-9 ein einstelliges alphanumerisches Feld, das die Position 79 im Bereich einnimmt.

STATUS KEY
Diese Klausel definiert mit Data-Name-10 ein 2stelliges, alphanumerisches Feld, das die Positionen 80 bis 81 im Bereich einnimmt.

MESSAGE COUNT
Diese Klausel definiert mit Data-Name-11 ein 6stelliges, numerisches Feld ohne Vorzeichen, das die Positionen 82 bis 87 im Bereich einnimmt.

- Bei Verwendung der zweiten Variante zur Definition dieser

Felder korrespondieren sie (in der richtigen Reihenfolge definiert) mit den oben beschriebenen Definitionsklauseln. Dabei müssen die zu vergebenden Feldnamen eindeutig sein, d.h. sie können nicht qualifiziert werden. Innerhalb einer Serie kann jeder Datenname durch das Wort FILLER ersetzt werden.

- Wird im Format überhaupt keine Wahlklausel definiert, muß auf die CD-Angabe eine 01-Datenbeschreibung folgen.

- Auf jede Wahlklausel kann eine 01-Datenbeschreibung folgen.

  Beide Definitionsvarianten ergeben einen Datenbereich, dessen Aufbau in der folgenden Abbildung dargestellt ist:

Stufen-Nr.	Feldaufbau		Kommentar
01	data-name-0.		
02	data-name-1	PICTURE X(12).	SYMBOLIC SUB-QUEUE
02	data-name-2	PICTURE X(12).	SYMBOLIC SUB-QUEUE-1
02	data-name-3	PICTURE X(12).	SYMBOLIC SUB-QUEUE-2
02	data-name-4	PICTURE X(12).	SYMBOLIC SUB-QUEUE-3
02	data-name-5	PICTURE 9(06).	MESSAGE DATE
02	data-name-6	PICTURE 9(06).	MESSAGE TIME
02	data-name-7	PICTURE X(12).	SYMBOLIC SOURCE
02	data-name-8	PICTURE 9(04).	TEXT LENGTH
02	data-name-9	PICTURE X.	END KEY
02	data-name-10	PICTURE XX.	STATUS KEY
02	data-name-11	PICTURE 9(06).	MESSAGE COUNT

Hinweis: Die Kommentar-Angaben dienen nur zur Klarstellung und sind nicht Bestandteil der Definitionen.

- Weitere 01-Satzbeschreibungen können auf eine Input-CD folgen und redefinieren diese und müssen sich ebenfalls über eine Länge von 87 Bytes erstrecken. Nur die erste Redefinition allerdings kann VALUE-Klauseln enthalten. MCS spricht den Satz stets im vorgenannten Format an.

Verwendung von Format 1:

- Die Angaben in den Feldern des Input-Communication-Bereichs stellen die ordnungsgemäße Kommunikation zwischen MCS und dem Programm sicher. Sie dienen der Beschreibung der zu transferierenden Informationen. Sie sind nicht Bestandteil der zu transferierenden Informationen, d.h. sie kommen nicht von einem Terminal.

- Bei Nichtverwendung der Datennamen 2 bis 4 müssen diese Felder Spaces enthalten. Die Felder mit den Datennamen 1 bis 4 enthalten symbolische Namen für Queues, Subqueues, usw. Diese Namen müssen vorab dem MCS mitgeteilt werden.

- Eine Informationseinheit oder ein Teil davon wird mit der Anweisung RECEIVE aus einer Queue entnommen. Näheres hierzu regelt die Communication Description.

  Grobe, mittlere, feine und feinste Entnahmen von Informationen aus den Queues können graduell bestimmt werden durch die Verwendung von Datenname 1 bis hin zu Datenname 4.

  Werden in der Communication Description des Programms weniger Levels angesprochen als die MCS-Queue-Hierarchie umfaßt, liefert MCS die Informationen bzw. Teile davon wunschgerecht.

  Nach Durchführung einer RECEIVE-Anweisung enthalten die mit Datennamen 1 bis Datennamen 4 bezeichneten Felder die symbolischen Namen aller Levels der Queue-Struktur.

- Immer wenn ein Programm von MCS aufgefordert wird, Informationen zu bearbeiten, werden die symbolischen Namen der Queue-Struktur, die diese Aktivität angefordert hat, in diejenigen Datenfelder plaziert, die in der INITIAL-Klausel der Communication Description mit Daten-Name-1 bis Daten-Name-4 lokalisiert werden. Ansonsten werden diese vier Felder mit Spaces initialisiert.

  Es werden die symbolischen Namen plaziert oder Spaces initialisiert, bevor der erste Befehl in der Procedure Division ausgeführt wird. Tritt anschließend irgendwo im Programm eine Anweisung RECEIVE auf, welche dieselben Felder mit Daten-Name-1 bis Daten-Name-4 anspricht, wird die Information, aufgrund derer das Programm von MCS aktiviert wurde, diesem zur Verfügung gestellt. Nur hierbei werden die weiteren Felder der Communication Description aktualisiert.

- Versucht MCS ein Programm ohne INITIAL-Klausel zu aktivie-

ren, sind die Resultate nicht vorhersagbar.

- Daten-Name-5 hat das Format JJMMTT (Jahr, Monat, Tag). Sein Inhalt repräsentiert das Datum, an dem MCS feststellt, daß die Information abgeschlossen ist. MCS aktualisiert Daten-Name-5 nur im Zusammenhang mit einer RECEIVE-Anweisung.

- Daten-Name-6 hat das Format HHMMSSTT (Stunden, Minuten, Sekunden, Hundertstelsekunden). Sein Inhalt repräsentiert die Uhrzeit, bei der MCS feststellt, daß die Information abgeschlossen ist. MCS aktualisiert Daten-Name-6 nur im Zusammenhang mit einer RECEIVE-Anweisung.

- Bei der Ausführung einer RECEIVE-Anweisung überträgt MCS den symbolischen Namen desjenigen Terminals, über das die Information eingegeben wurde, in das Feld mit dem Daten-Namen-7. Kennt MCS den symbolischen Namen des Terminals nicht, wird Daten-Name-7 mit Spaces gefüllt.

- In Daten-Name-8 trägt MCS im Rahmen der RECEIVE-Anweisung die Länge der Information ein. Mehr hierüber finden Sie bei der Beschreibung der RECEIVE-Anweisung.

- Der Inhalt von Daten-Name-9 wird nur als Teil der Ausführung einer RECEIVE-Anweisung gesetzt. Ist eine RECEIVE MESSAGE-Angabe vorhanden und es wurde Gruppen-Ende festgestellt, wird Daten-Name-9 auf 3 gesetzt. Wird bei vorhandener RECEIVE MESSAGE-Angabe das Ende einer Information (End of Message) festgestellt, wird Daten-Name-9 auf 2 gesetzt. Wird nur eine Teil-Information übertragen, wird Daten-Name-9 auf 0 gesetzt.

- Daten-Name-9 wird auch auf 3 gesetzt, wenn die RECEIVE SEGMENT-Angabe gemacht wurde und Gruppen-Ende auftritt. Tritt unter RECEIVE SEGMENT das Ende einer Information auf, wird Daten-Name-9 auf 2 gesetzt. Tritt unter RECEIVE SEGMENT das Ende eines Segments auf, wird Daten-Name-9 auf 1 gesetzt. Bei Übertragung einer Teil-Information unter RECEIVE SEGMENT wird Daten-Name-9 auf 0 gesetzt.

- Treten mehrere der vorab beschriebenen Bedingungen auf, richtet sich der anschließende Inhalt von Daten-Name-9 danach, welche Bedingung in der angegebenen Reihenfolge zuerst aufgetreten ist.

- Daten-Name-10 nimmt den Status einer vorher ausgeführten RECEIVE-, ACCEPT MESSAGE COUNT, ENABLE INPUT oder DISABLE INPUT-Anweisung auf. Eine Tabelle der Stati mit Erläuterungen finden Sie im Anschluß an diese Formatbeschreibung.

- Daten-Name-11 weist die Anzahl der Nachrichten aus, die in einer Queue, Sub-Queue-1, usw. existieren. Daten-Name-11 wird von MCS nur im Rahmen einer ACCEPT COUNT-Anweisung aktualisiert.

Format 2:

- Werden in der Communication Description für Output keine Wahlklauseln verwendet, muß ihr eine 01-Datendefinition folgen.

- Für jede Communication Description für Output ist ein Datenbereich vorgesehen, dessen Länge über alle Felder hinweg sich nach folgender Formel bemißt: 10 plus (13mal Integer-1).

  DESTINATION COUNT
  Diese Angabe präsentiert Daten-Name-1 als Name eines vierstelligen numerischen Feldes ohne Vorzeichen, das die ersten vier Zeichen des Records abdeckt.

  TEXT LENGTH
  Diese Angabe präsentiert Daten-Name-2 als Name eines 4stelligen numerischen Feldes ohne Vorzeichen, das die Positionen 5 - 8 im Record belegt.

  STATUS KEY
  Diese Angabe präsentiert Daten-Name-3 als 2stelliges numerisches Feld, das die Positionen 9 und 10 im Record belegt.

  Positionen 11 - 23
  Die Positionen 11 - 23 und jedes Set von 13 Zeichen im Anschluß daran bilden Tabellenposten des folgenden Formats:

  - ERROR KEY mit Daten-Name-4 (ein einstelliges alphanumerisches Feld)
  - SYMBOLIC DESTINATION mit Daten-Name-5 (ein 12stelliges alphanumerisches Feld)

Stufen-Nr.	Feldaufbau		Kommentar
01	data-name-0.		
02	data-name-1	PICTURE 9(04).	DESTINATION COUNT
02	data-name-2	PICTURE 9(04).	TEXT LENGTH
02	data-name-3	PICTURE XX.	STATUS KEY
02	data-name OCCURS integer-2 TIMES.		DESTINATION TABLE
03	data-name-4	PICTURE X.	ERROR KEY
03	data-name-5	PICTURE X(12).	SYMBOLIC DESTINATION

Hinweis: Die Kommentar-Angaben dienen nur zur Klarstellung und sind nicht Bestandteil der Definitionen.

Die Anwendung der vorab beschriebenen Klauseln ergibt einen Record, dessen Aufbau der obigen Übersicht entspricht.

- Nachfolgende Satzdefinitionen sind statthaft und redefinieren den Record, jedoch darf nur die erste Redefinition VALUE-Angaben enthalten. MCS richtet sich immer nach dem Aufbau der ersten, originalen Definition.

- Die Daten-Namen 1 bis 5 dürfen innerhalb einer Communication Description nur einmal vorkommen.

- Bei Weglassung von DESTINATION TABLE OCCURS wird einmal ERROR KEY und einmal SYMBOLIC DESTINATION angenommen. In diesem Falle ist weder Tabellen-Subscribierung noch -Indexierung statthaft.

- Bei Vorhandensein der Klausel DESTINATION TABLE OCCURS dürfen die Daten-Namen-4 und 5 nur über Subscribierung oder Indexierung angesprochen werden.

- Eine Information wird von diesem Output-Communication-Description-Bereich nicht direkt an das Terminal gesendet, sondern zunächst an MCS.

- Bei Ausführung einer der Anweisungen SEND, ENABLE OUTPUT oder DISABLE OUTPUT entnimmt MCS dem Daten-Namen-1 die Anzahl der Symbolic Destinations (zu beschickende Terminals z. B.). Auf entsprechend viele Posten der Tabelle (vom Anfang an und aufsteigend) greift MCS dann zu.

Bei Ausführung einer SEND-, ENABLE OUTPUT- oder DISABLE OUTPUT-Anweisung wird dann ein ERROR indiziert, wenn der Inhalt von Daten-Name-1 größer ist als der Umfang von 1 bis Integer-1; die Anweisung wird darüber hinaus abgebrochen.

- Die Länge der Information entnimmt MCS dem Daten-Namen-2.

- Jede Symbolic Destination aus dem Daten-Namen-5 der Tabelle (symbolischer Name des Bestimmungsortes - z. B. eines Terminals) ist dem MCS bekannt (angegeben bei der Erstellung des Kommunikations-Netzwerks).

- Das jeweilige Ergebnis einer SEND-, ENABLE OUTPUT- oder DISABLE OUTPUT-Anweisung wird mittels eines Status im Daten-Namen-3 berichtet. Statuserläuterungen finden Sie in der nachfolgenden Tabelle.

- Stellt MCS während der Ausführung einer SEND-, ENABLE OUTPUT- oder DISABLE OUTPUT-Anweisung fest, daß irgendein angegebener Bestimmungsort unbekannt ist, werden die Inhalte von Daten-Name-3 und allen Daten-Namen-4 der Tabelle aktualisiert:

  Eine 1 in Daten-Name-4 besagt, daß der Symbolic Destination-Name im zugehörigen Daten-Namen-5 dem MCS bei der Netzwerkgenerierung nicht bekanntgemacht wurde. Andernfalls enthält Daten-Name-4 eine 0.

Status-Key-Codes und Konditionen

Die nachfolgende Fehlerstatus-Übersicht zeigt die möglichen 2stelligen Statuswerte inklusive einer jeweiligen Erläuterung. Unter "Request" ist die jeweilige COBOL-Kommunikationsanweisung gemeint, mittels derer eine Aktion von TAM (NCR ITX Telecommunications Access Method) verlangt wird.

STATUS	MEANING	SEND	PURGE	ENABLE OUTPUT	DISABLE OUTPUT	ACCEPT COUNT	RECEIVE	ENABLE INPUT	DISABLE INPUT	ENABLE INPUT TERM	DISABLE INPUT TERM
00	**Validation status.** No error is detected. The request is accepted. For intrahost interprogram communication, the request is completed.	X	X	X	X	X	X	X	X	X	X
10	**Validation status.** The destination is disabled. The request is accepted.	X									
11	**Validation status.** The request is rejected. There have been more than the maximum number of SENDs to a disabled destination. For intrahost interprogram monologs and for communication using SNA, the maximum number of SEND requests permitted is zero. If the program previously sent any partial messages or message segments, they are deleted.	X									
12	**Validation status.** The communication link is busy or has not been $TCM connected. TAM has rejected an ENABLE and DISABLE or a SEND request.	X		X	X					X	
15	**Validation status.** The request is rejected. The enable or disable request was not accepted because the inbound or outbound monolog was not in a state to accept the request. For an ENABLE OUTPUT request, the outbound monolog was already in an enabled state, or a disable is required before retrying the enable. For a DISABLE OUTPUT request, the outbound monolog was already in a disabled state. For an ENABLE INPUT TERMINAL request, the inbound monolog was already enabled.			X	X				X	X	
20	**Validation status.** The request is rejected. The source, destination, or queue symbolic name is unknown.	X	X	X	X	X	X	X	X	X	X
21	**Validation status.** The request is rejected. Either there is no space available in the ITX Operating System's memory or a queue is full. For monologs between programs, this may indicate that the destination input queue is full. If this is the reason for the status, any partial messages or message segments previously sent are deleted.	X		X	X		X	X	X	X	X
27	**Input report status.** The input message is longer than the maximum length specified in the EDF. The message is						X				

STATUS	MEANING	SEND	PURGE	ENABLE OUTPUT	DISABLE OUTPUT	ACCEPT COUNT	RECEIVE	ENABLE INPUT	DISABLE INPUT	ENABLE INPUT TERM	DISABLE INPUT TERM
	truncated to the maximum message size for the queue specified in the EDF.										
28	**Input report status.** No input was received before the input timer reached the specified limit. The inbound and outbound monologs are dissolved. The line is disconnected. This status applies to integrated 2780/3780 BSC and 3270 BSCTributary and DLC-Secondary communication.						X				
29	**Unprocessable input report status.** An input has been received for a disabled monolog. The input will be placed on the unprocessable input queue.						X				
2A	**Input report status.** An error occurred during the transmission of input, possibly due to a line drop. For integrated TTY communication, the input is retried by TAM. For other integrated communication, the inbound and outbound monologs are dissolved. With integrated ISO communication, this status indicates a device error. This status applies to integrated communication only.						X				
30	**Validation status.** The content of data-name-1 (DESTINATION COUNT) is invalid. The request is rejected.	X		X	X						
40	**Validation status.** The password is invalid. The request is rejected.			X	X			X	X	X	X
50	**Validation status.** The text length (data-name-2) is greater than the length of the sending field. The request is rejected.	X									
51	**Validation status.** The message or message segment is longer than the maximum length you specified in the EDF. The request is rejected.	X									
52	**Validation status.** This segment would cause an output message to be greater than the maximum size permitted. This segment is not held by TAM; but previously queued segments are held and sent (not on this call) as one message. The request is rejected.	X									
60	**Validation status.** This partial segment has either no sending area specified or a text length equal to zero. The request is rejected.	X									

STATUS	MEANING	SEND	PURGE	ENABLE OUTPUT	DISABLE OUTPUT	ACCEPT COUNT	RECEIVE	ENABLE INPUT	DISABLE INPUT	ENABLE INPUT TERM	DISABLE INPUT TERM
65	**Validation status.** TAM has rejected a SEND request because the maximum number of completed messages to be queued for output while the monolog is enabled has been reached. For example, this status may occur when an application program continues to send messages for a switched line while the connection is being made. The output queue capacity may be exceeded before the connection is made. Recommendation: Increase the NDL value of OUTQSIZE to 100 or more.	X									
90	**Output report status.** TAM rejected a previous SEND request to the remote correspondent. Either a previous ENABLE OUTPUT or SEND request for the same monolog failed or the remote correspondent terminated the monolog. For integrated TTY communication, this status may result from a line break signal received on a previous SEND request. For integrated ISO ASYNC communication, this status may result from receiving a status indicating that the device required operator attention on a previous SEND request. This status is returned on the report queue. When REPORT = IMMEDIATE, the current SEND request was not initiated and no data is returned. NOTE: All reports from 90-9B have to do with the final disposition of a send request. They are normally "received" from a report queue — but do not reflect a receive request.	X					X				
91	**Output report status.** A previous SEND request to the remote destination was terminated because the terminal sent a line break signal during output transmission. Data is returned in the input buffer. The outbound monolog state is changed such that only ENABLEs or DISABLEs are accepted. For example, if the application wants to resume output, it must perform an ENABLE OUTPUT. When REPORT = IMMEDIATE, the current request is terminated and no data is returned.	X					X				
92	**Output report status.** A previous SEND request to the remote destination was terminated because the destination terminal sent a reverse interrupt (RVI) signal during an output operation. Data is returned in the input buffer. The monolog state is not changed. When	X					X				

STATUS	MEANING	SEND	PURGE	ENABLE OUTPUT	DISABLE OUTPUT	ACCEPT COUNT	RECEIVE	ENABLE INPUT	DISABLE INPUT	ENABLE INPUT TERM	DISABLE INPUT TERM
	REPORT = IMMEDIATE, the current request is terminated and no data is returned. This status occurs with integrated BSC communication only.										
93	**Output report status.** A previous SEND request to the remote destination was not initiated. This is caused by the detection during ouput of a VOID input from an ISO ASYNC financial terminal. Data is returned in the input buffer. (This condition is not considered a monolog failure.) The monolog state is not changed. When REPORT = IMMEDIATE, the current request is not initiated and no data is returned. This status applies only to FEP communication with an ISO ASYNC financial terminal.	X					X				
94	**Output report status.** A previous SEND request to the remote destination was terminated because the terminal is in a state which requires operator attention. This can be caused by a printer that is not ready or out of paper. Data is returned in the input buffer. The outbound monolog state is changed such that only ENABLEs or DISABLEs are accepted. For example, if the application wants to resume output, it must perform an ENABLE OUTPUT. When REPORT = IMMEDIATE, the request is terminated and no data is returned. This status occurs with integrated ISO ASYNC communication only.	X					X				
95	**Output report status.** A previous SEND request to the remote destination failed because the device returned a negative acknowledgement (NAK) when selected for output.	X					X				

STATUS	MEANING	SEND	PURGE	ENABLE OUTPUT	DISABLE OUTPUT	ACCEPT COUNT	RECEIVE	ENABLE INPUT	DISABLE INPUT	ENABLE INPUT TERM	DISABLE INPUT TERM
	Data is returned in the input buffer. The inbound and outbound monologs are dissolved. When REPORT = IMMEDIATE, the current SEND request is rejected and no data is returned. This status occurs with integrated ISO ASYNC communication only.										
98	**Output report status.** A previous SEND request to the remote destination was successful. The message was delivered and data is returned in the input buffer. When REPORT = IMMEDIATE no data is returned.	X					X				
99	**Output report status.** A previous SEND request to the remote destination failed. In integrated mode, this status may result from a line drop, an I/O failure or a device error. If the SEND was using the output-and-reply function, this status indicates that the output portion of the function failed. In the FEP mode, this status may result from a failure of transmission or failure of the acknowledgement of the transmission. Data is returned in the input buffer, except in the FEP mode when there is a transmission failure after a previous positive acknowledgement. In that case, data is not returned. For integrated communication, the inbound and outbound monologs are dissolved. For FEP communication, only the outbound monolog is dissolved. When REPORT = IMMEDIATE, no data is returned.	X					X				
9A	**Output report status.** A previous SEND request to the remote destination was unsuccessful because the inactivity time limit expired on a BSC output or a BSC bid for output. Data is returned in the input buffer. The inbound and outbound monologs are dissolved. When REPORT = IMMEDIATE, no data is returned. This status applies only to integrated 2780/3780 BSC and 3270 BSC Trib communication.	X					X				
9B	**Output report status.** A previous SEND request to the remote destination, using output-and-reply, was unsuccessful due to a failure on the reply portion of the function. Data is returned in the input buffer. The inbound and outbound monologs are dissolved. When REPORT = IMMEDIATE, no data is returned. This status occurs with integrated TTY communication only.	X					X				

STATUS	MEANING	SEND	PURGE	ENABLE OUTPUT	DISABLE OUTPUT	ACCEPT COUNT	RECEIVE	ENABLE INPUT	DISABLE INPUT	ENABLE INPUT TERM	DISABLE INPUT TERM
B0	**Input report status.** Integrated ISO ASYNC and DLC only. The remote correspondent scheduled for polling failed to respond to a poll. The result is that the correspondent is polled less frequently, either until it responds (then polled at original frequency), or until it is disabled (then no longer polled). The inbound monolog state stays enabled.						X				
B1	**Input report status.** Integrated ISO AYNC and DLC only. The remote correspondent, previously failing to respond to a poll, now is responding. It will be polled at the original frequency.						X				
B2	**Input report status.** The input transmission from the transmitting BSC station was terminated with an EOT. The BSC input transmission is finished. This status applies only to integrated 2780/3780 BSC and 3270 BSC Trib communication.						X				
B3	**Input report status.** The input message was put on the specified input queue.			X			X				
B4	**Input/output report status.** Integrated DLC-P only. The device is no longer in an information transfer state. DLC devices become logically disconnected (LDS) or report a bad frame. Re-establishment of the information transfer state is attempted.						X				
B5	**Input/output report status.** Integrated DLC-P only. The information transfer state is re-establishment. Suspended monologs are returned to enable state.						X				
CA	**Input/Output report status.** For 2780/3780 BSC communications, an incorrect XID was received and the inbound and outbound monologs were dissolved. For DLC communication, a previous ENABLE INPUT TERMINAL or ENABLE OUTPUT monologs remain disabled. **Immediate output report status.** With DLC integrated communication only, the same conditions as above except that is for the current request.						X				

STATUS	MEANING	SEND	PURGE	ENABLE OUTPUT	DISABLE OUTPUT	ACCEPT COUNT	RECEIVE	ENABLE INPUT	DISABLE INPUT	ENABLE INPUT TERM	DISABLE INPUT TERM
CB	**Input report status.** For 2780/3780 BSC communications, a timeout occurred because no XID or XID response was received. For DLC communication, a previous ENABLE INPUT TERMINAL or ENABLE OUTPUT failed, the monologs remain disabled. Immediate output report status, same conditions as above except that is for the current request.			X			X				
CD	**Validation status.** For BSC communications, the output request for an RVI was rejected.	X									
CE	**Validation status.** The completed BSC message does not have a valid termination character. An end-of-message indicator (EMI) or an end-of-group indicator (EGI) was used as the termination character. The request is rejected and any previous partial segments are not retained by TAM. This status applies only to integrated 2780/3780 BSC and 3270 BSC Trib communication. This status will not be returned when INSERT-REMOVE = YES for this terminal.	X									
CF	**Validation status.** An ENABLE INPUT TERMINAL or ENABLE OUTPUT request to a remote correspondent which is a device on a dial-out line was received and there is no dial-out line available. The request is rejected and the monolog remains disabled.			X						X	
D0	**Input report status.** An inbound monolog between the local correspondent and the remote correspondent, specified in data-name-7 (SYMBOLIC SOURCE) was terminated by the remote correspondent. For integrated communication, this status indicates a line drop or I/O failure. For integrated BSC communication, this status could indicate a station abort (BSC) was detected during input. For FEP communication, this status could indicate that the FEP is down. If the FEP is down, TAM performs recovery. No data is made available. The inbound monolog is dissolved.							X			
D1	**Input report status.** A previous ENABLE INPUT TERMINAL request to the remote correspondent, specified in data-name-7 (SYMBOLIC SOURCE) has failed. For integrated						X				

STATUS	MEANING	SEND	PURGE	ENABLE OUTPUT	DISABLE OUTPUT	ACCEPT COUNT	RECEIVE	ENABLE INPUT	DISABLE INPUT	ENABLE INPUT TERM	DISABLE INPUT TERM
	communication, this status could indicate a failure of an auto-answer or auto- dial I/O. For monologs between programs, this status means that the destination program is not known by TAM to be in the system. No data is made available and the inbound monolog state remains disabled.										
D2	**Input report status.** A previous ENABLE INPUT TERMINAL request to the remote correspondent, specified in data-name-7 (SYMBOLIC SOURCE) was successful. This status is returned only if all control, all, or immediate reports are requested by the report queue option. No data is made available.						X				
D3	**Input report status.** A previous ENABLE INPUT TERMINAL request to the remote correspondent named in data-name-7 (SYMBOLIC SOURCE) has failed due to a device scheduling error. No data is made available. The inbound monolog remains disabled. This status applies to integrated ISO ASYNC and DLC and 3270 BSC Trib communication only.						X				
D4	**Output report status.** A previous ENABLE OUTPUT request to the remote destination was successful. The outbound monolog state is changed to enabled. When REPORT = IMMEDIATE, the current request is successful.			X			X				
D5	**Output report status.** A previous DISABLE OUTPUT request to the remote destination was successful. The outbound monolog state is changed to disabled. When REPORT = IMMEDIATE, the current request is successful.				X		X				
D7	**Output report status.** A previous DISABLE OUTPUT request to the remote destination failed or it was not initiated by TAM. For integrated communication, this may be caused by an I/O failure on the line reset or auto-dial disconnect associated with the DISABLE OUTPUT. A DISABLE OUTPUT is not initiated after a failure of a previous ENABLE OUTPUT or SEND request for the same monolog. No data is made available. The outbound monolog state is changed to disabled. When REPORT = IMMEDIATE the current request failed or was not initiated by TAM.				X		X				

STATUS	MEANING	SEND	PURGE	ENABLE OUTPUT	DISABLE OUTPUT	ACCEPT COUNT	RECEIVE	ENABLE INPUT	DISABLE INPUT	ENABLE INPUT TERM	DISABLE INPUT TERM
D8	**Output report status.** An output monolog to the remote correspondent, specified in data-name-7 (SYMBOLIC SOURCE) was terminated by this remote correspondent. In integrated BSC communication, this status indicates that a STATION ABORT condition was detected for this output monolog. This status is returned on the report queue. No data is made available. For BSC communication both the inbound and outbound monologs are dissolved. The line is disconnected. If the FEP is down, TAM performs recovery.						X				
D9	**Output report status.** A previous ENABLE OUTPUT request to the remote destination failed. For integrated communication, this status could indicate a failure of an auto-answer or auto-dial I/O. No data is made available and the outbound monolog state remains disabled. When REPORT = IMMEDIATE, the current request failed.						X				
DA	**Output report status.** A previous ENABLE OUTPUT request to the remote destination failed. FOR ISO ASYNC this was due to a device scheduling error. For DLC-Secondary the problem was a logical link error or XID error. No data is made available. The outbound monolog state remains disabled. When REPORT = IMMEDIATE, the current request failed. This status applies only to integrated ISO ASYNC, DLC, and 3270 BSC Tributary communication.			X			X				
DB	**Input/output report status** A disconnect request has been received from the remote correspondent. If the inbound monolog is enabled it is placed on the input queue, otherwise it is placed on the output report queue. The application may honor the disconnect request by disabling the monologs, or ignore the request. If ignored, the requesting station can discontinue sending input - even though the inbound monologs remain enabled. For integrated DLC only.						X				
E0	**Validation status.** An ENABLE OUTPUT request to an SNA or X.25 correspondent was rejected because that feature is not available, or a SEND request was rejected because $DPNDM is not active; or, with SNA, an ACTPU did not occur. This status applies to SNA and X.25 only.			X							

STATUS	MEANING	SEND	PURGE	ENABLE OUTPUT	DISABLE OUTPUT	ACCEPT COUNT	RECEIVE	ENABLE INPUT	DISABLE INPUT	ENABLE INPUT TERM	DISABLE INPUT TERM
E1	**Validation status.** A SEND request to a System Network Architecture (SNA) or X.25 correspondent has been rejected because the end indicator was not valid within an SNA network. The only indicators that are valid within an SNA network are an end-of-message indicator (EMI) or an end-of-group indicator (EGI).	X									
E2	**Output report status.** A previous SEND request to the System Network Architecture (SNA) or X.25 remote correspondent, specified in data-name-7 (SYMBOLIC SOURCE) has succeeded or failed according to information within the SNA or X.25 interface header. Data is returned in the input buffer. The monolog states are not changed.						X				
F0	**Validation status.** The request is rejected because TAM is not initialized.	X	X	X	X	X	X	X	X	X	X
F1	**Output report status.** A system error has occurred. The final status of a SEND request to the remote correspondent, specified in data-name-7 (SYMBOLIC SOURCE) is not known. If the requesting program also requests a RECEIVE, data is returned to the input buffer. For integrated communications, this status can be seen only on a global report queue and only due to software (TAM or PIO) error. If you are using integrated communication and you receive this status, inform your NCR software representative. For FEP communication, the FEP went down and could not respond with a SEND acknowledgement.						X				
F2	**Validation status.** The TAM FEP is not available or integrated line recovery is in process. For FEP communication, the FEP is being loaded during the FEP initialization process or after it was down. For integrated ISO ASYNC communication, the line is down and TAM is in the process of notifying the correspondents using devices on the line. When recovery is complete, MCS requests are accepted.	X		X	X			X	X	X	X
F3	**Validation status.** TAM rejected this request because the application program has not previously done any ENABLEs.	X	X		X	X	X		X		X

STATUS	MEANING	SEND	PURGE	ENABLE OUTPUT	DISABLE OUTPUT	ACCEPT COUNT	RECEIVE	ENABLE INPUT	DISABLE INPUT	ENABLE INPUT TERM	DISABLE INPUT TERM
F4	**Validation status.** TAM rejected this request because the application program is not defined in the EDF.	X	X	X	X	X	X	X	X	X	X
F5	**Validation status.** TAM rejected this request because the application program is not defined as replicable in the EDF and the program is already running.			X			X		X		
F6	**Validation status.** The request is rejected. TAM device recovery is in progress. An ENABLE OUTPUT request was received for a monolog with a device which has previously indicated it needs operator attention (status key 94). TAM cannot accept an ENABLE OUTPUT for the device until the outputs which were waiting for initiation to the device are returned to the program. This status applies only to integrated ISO ASYNC communication.			X							
F8	**Validation status.** For DLC-Primary communication only. Meaning: TAM device disable in progress. An Enable Input Terminal or Enable Output request was received for a monolog with a device which is still processing a previous Disable Input, Disable Input Terminal, or Disable Output request. Action: The request is rejected.			X						X	
FF	**Validation status.** A software error has been detected. The error is logged to the system error log file.	X	X	X	X	X	X	X	X	X	X

KAPITEL 8: PROCEDURE DIVISION

EINLEITUNG

Die PROCEDURE DIVISION wird als 4. DIVISION in den COBOL-Compiler eingegeben. Sie enthält die COBOL-Anweisungen für den Computer zur schrittweisen Programm-Ausführung. Diese Anweisungen werden in COBOL "PROCEDURES" (= Prozeduren bzw. Verarbeitungs-Anweisungen) genannt. Die PROCEDURE DIVISION enthält zwei Arten von Verarbeitungs-Anweisungen: DECLARATIVE-Prozeduren und NONDECLARATIVE-Prozeduren.

DECLARATIVE-Prozeduren sind auf einen besonderen Zweck ausgerichtete Anweisungen, die unter der Steuerung des Betriebssystems arbeiten. Wenn das Programm DECLARATIVE-Prozeduren enthält, müssen sie am Anfang der PROCEDURE DIVISION stehen, und zwar muß das Schlüsselwort DECLARATIVES vorausgehen und die Schlüsselwörter END DECLARATIVES folgen.

NONDECLARATIVE-Prozeduren sind logischerweise nachfolgende Anweisungen zur Datenverarbeitung. Sind DECLARATIVES vorhanden, dann müssen sie vor den NONDECLARATIVE-Prozeduren am Anfang der PROCEDURE DIVISION stehen. Die Ausführung des Objekt-Programmes beginnt mit der ersten Anweisung in den NONDECLARATIVE-Prozeduren und setzt sich in logischer Folge fort.

Die PROCEDURE DIVISION beginnt mit einer DIVISION-Überschrift, die sich aus den reservierten Wörtern PROCEDURE DIVISION, der wahlweise verwendbaren USING-Klausel, einem Punkt und einem Leerzeichen zusammensetzt. Eine Darstellung der Funktion und Verwendung der USING-Klausel in der Überschrift der PROCEDURE DIVISION folgt im Kapitel "INTER PROGRAM COMMUNICATION".

Das folgende Format zeigt die Struktur der PROCEDURE DIVISION, wenn diese Kapitel (Sections) enthält.

```
PROCEDURE DIVISION [USING Identifier-1  [Identifier-2]...]
[DECLARATIVES.
{Kapitel-Name SECTION   [Segment-Nummer]. USE-Anweisung.
[ Paragraphen-Name.    [Satz]   ...  ] ... } ...
 END DECLARATIVES.]
{Kapitel-Name  SECTION  [Segment-Nummer].
[ Paragraphen-Name.    [Satz]   ...  ] ... } ...
```

Das folgende Format zeigt die Struktur der PROCEDURE DIVISION, wenn diese keine Kapitel (Sections) enthält.

```
PROCEDURE DIVISION [USING [Identifier-1 [, Identifier-2] ...] .

{Paragraphen-Name.  [Satz]  ...} ...
```

Das Ende der PROCEDURE DIVISION und das physische Ende des COBOL Ursprungs-Programmes ist die physische Position im Ursprungs-Programm, nach der keine weiteren Prozeduren auftreten.

PROZEDUREN

Eine PROZEDUR enthält Anweisungen für den Computer. PROZEDUREN sind in der PROCEDURE DIVISION entweder in Paragraphen oder Kapiteln angeordnet. Programmiererwörter, Prozedur-Namen genannt, werden vom Programmierer bestimmt und verwendet, um einen Paragraphen oder ein Kapitel in der PROCEDURE DIVISION zu benennen. Es gibt zwei Arten von Prozedur-Namen: Kapitel-Namen und Paragraphen-Namen.

Ein Paragraph besteht aus null, einem oder mehreren Befehls-Sätzen. Ein Paragraph beginnt mit einem Paragraphen-Namen. Ein Paragraphen-Name ist ein Programmiererwort, das sich lediglich aus den Zeichen des Wortvorrats zusammensetzt. Ein Paragraphen-Name darf in der Länge 30 Zeichen nicht überschreiten. Ein Paragraphen-Name muß kein alphabetisches Zeichen enthalten, wiederum kann er aber alle numerischen Zeichen enthalten. Ein Paragraphen-Name braucht in einem Programm nicht eindeutig zu sein, denn er kann durch einen Kapitel-Namen qualifiziert sein. Ein Paragraphen-Name beginnt in Spalte 8, 9, 10 oder 11 des Programmblattes und endet mit einem Punkt und zumindest einer Leerstelle.

Der oder die auf den Paragraphen-Namen folgenden Sätze bilden den kompletten Paragraphen. Der erste Satz des Paragraphen kann auf derselben Zeile wie der Paragraphen-Name oder auf der Zeile unmittelbar unter dem Paragraphen-Namen beginnen. Der auf den Zeilen unterhalb des Paragraphen-Namens befindliche Satz beginnt in jeder beliebigen Position ab Spalte 12 des Programmblattes.

Das folgende Beispiel zeigt zwei Paragraphen mit den Paragraphen-Namen LOESCH-ROUTINE und AENDER-ROUTINE.

```
1       8
016010 LOESCH-ROUTINE.    MOVE ZERO TO FLAG-D.
016020      MOVE KONTO-NR TO DRUCK-KONTO-NR.
016030      MOVE SALDO TO DRUCK-SALDO;    MOVE ACODE TO PCODE.
016040      WRITE DRUCKSATZ.
016050 AENDER-ROUTINE.
016060      ADD UEBERW-BETRAG TO SALDO.
016070      MOVE UEBERW-DATUM TO DATUM.
016080      IF FAELLIG-DATUM IS EQUAL TO TAGES-DATUM GO TO
            RECH-ROUTINE.
```

Das Ende eines Paragraphen wird durch das Auftreten eines anderen Paragraphen-Namens, einer neuen Kapitel-Überschrift oder das Ende der PROCEDURE DIVISION bestimmt. In den DECLARATIVE-Prozeduren wird ein Paragraph auch durch die Schlüsselwörter END DECLARATIVES beendet. Im vorhergehenden Beispiel enthält die Zeile mit der Nummer 016040 die letzte Zeile des Paragraphen LOESCH-ROUTINE, da auf der nächsten Zeile der Paragraphen-Name AENDER-ROUTINE erscheint.

Ein Paragraph besteht aus null, einem oder mehreren aufeinanderfolgenden Sätzen, wobei auf jeden dieser Sätze ein Punkt und eine Leerstelle folgt. Im Beispiel enthält der Paragraph LOESCH-ROUTINE vier Sätze.

Ein Satz besteht aus einer oder mehreren COBOL-Anweisungen. Jede Anweisung ist eine syntaktisch gültige Kombination von Wörtern und Symbolen, die mit einem COBOL-Verb beginnt. In dem letzten Beispiel besteht der Satz auf Zeile 016030 aus zwei Anweisungen: MOVE SALDO TO DRUCK-SALDO und MOVE ACODE TO PCODE. Das Verb bezeichnet die Funktion des Computers bei der Ausführung der Anweisung. Das Semikolon (;) dient nur der Übersichtlichkeit, ist jedoch nie erforderlich.

Das Format jeder Anweisung muß mit der für jedes Verb definierten Struktur übereinstimmen. Dieses Format beginnt mit dem Verb, auf das Kombinationen von Schlüsselwörtern, Wahlwörtern, Programmiererwörtern und Literalen folgen.

Ein Kapitel besteht aus null, einem oder mehreren Paragraphen. Ein Kapitel beginnt mit einem Kapitel-Namen, gefolgt vom Wort SECTION, einem Punkt und einer Leerstelle (Kapitel-Überschrift). Eine Kapitel-Überschrift beginnt in Spalte 8, 9, 10 oder 11 des Programmblattes.

Der Kapitel-Name ist ein Programmiererwort, das sich aus den Zeichen des Wortvorrates zusammensetzt. Ein Kapitel-Name darf in der Länge 30 Zeichen nicht überschreiten. Ein Kapitel-Name muß kein alphabetisches Zeichen enthalten, wiederum kann er aber alle numerischen Zeichen enthalten. Ein Kapitel-Name muß in einem Programm eindeutig sein.

Ein Kapitel wird durch das Auftreten einer neuen Kapitel-Überschrift oder das Ende der PROCEDURE DIVISION beendet. In den DECLARATIVE-Prozeduren wird ein Kapitel auch durch die Schlüsselwörter END DECLARATIVES beendet. Im folgenden Beispiel hat die letzte Zeile des Kapitels mit dem Kapitel-Namen HAUPT die Seiten-/Zeilennummer 005050, denn die Kapitel-Überschrift ENDE SECTION erscheint auf der nächsten Zeile.

```
1       8
005010 HAUPT SECTION.
005020 ANFANG.  READ KART-DATEI AT END GO TO ENDE1.
005030      MOVE KART-SATZ TO MAG-SATZ.
005040      IF TCODE IS GREATER THAN 5 GO TO FAELLIG-DAT-A
005050                                ELSE GO TO FAELLIG-DAT-B.
005210 ENDE SECTION.
005220 ENDE1.   CLOSE MAG-DATEI, KART-DATEI, FEHL-DATEI.
005230      STOP RUN.
005240 FEHLER-ROUTINE.
005250      MOVE KART-SATZ TO FEHL-DATEN.
005260      WRITE FEHL-SATZ.
005270 FAELLIG-DAT-A.
```

In einem COBOL-Ursprungsprogramm können Kapitel zu einer größeren Gruppierung, Segment genannt, zusammengefaßt werden. Die auf das Wort SECTION in der Kapitel-Überschrift folgende Segment-Nummer bezeichnet die in einem Segment gruppierten Kapitel. Alle Segmente mit einer Nummer von 00 bis 49 sind ständig im Speicher des Computers. Alle Segmente mit einer Nummer von 50 bis 99 sind extern auf Magnetspeichermedien, wie beispielsweise einer Magnetplatte, gespeichert. Bevor Prozeduren in einem extern gespeicherten Segment ausgeführt werden, wird das Segment von dem externen Speichermedium in den Speicher des Computers geladen. Dadurch werden andere, nicht länger benötigte Prozeduren überlagert. Diese Technik der Programmsegmentation wurde für Computer mit zu kleinem Ladespeicher entwickelt. Sie wird heute kaum noch benötigt.

DECLARATIVES

DECLARATIVES sind ein oder mehrere Kapitel, die am Anfang der PROCEDURE DIVISION geschrieben werden. Jedes DECLARATIVE-Kapitel enthält eine USE-Anweisung, gefolgt von null, einem oder mehreren Paragraphen, die die DECLARATIVE-Prozeduren enthalten, die auszufüllen sind, wenn die in der USE-Anweisung spezifizierte Bedingung gegeben ist. Die Ausführung eines DECLARATIVE-Kapitels erfolgt, wenn eine der folgenden Bedingungen auftritt:

- die Beendigung der durch das NCR ITX-Betriebssystem vorgesehenen Ein-Ausgabe-Fehler-Routine;

- die Erkennung einer AT END-Bedingung, wenn die AT END-Angabe in einer Ein-Ausgabe-Anweisung nicht spezifiziert wurde;

- die Erkennung einer INVALID KEY-Bedingung, wenn INVALID KEY nicht in einer Ein-Ausgabe-Anweisung spezifiziert wurde.

DECLARATIVES beginnen mit dem Schlüsselwort DECLARATIVES, auf das ein Punkt und ein Leerzeichen folgt. Das Wort DECLARATIVES beginnt in Spalte 8, 9, 10 oder 11 des Programmblattes. Auf derselben Zeile darf kein anderer Text erscheinen.

Eine Kapitel-Überschrift wird auf der nächsten Zeile geschrieben. Die Kapitel-Überschrift besteht aus dem Namen des Kapitels (vom Programmierer zugeteilt), gefolgt von einem Leerzeichen, dem Wort SECTION, einer wahlweisen Segment-Nummer, einem Punkt und einem Leerzeichen. Diese Überschrift beginnt in Spalte 8, 9, 10 oder 11 des Programmblattes.

Ein DECLARATIVE-Satz wird unmittelbar nach der Kapitel-Überschrift geschrieben. Ein DECLARATIVE-Satz ist ein Satz mit einer USE-Anweisung. Er spezifiziert die Datei oder Dateien, auf die sich die nachfolgenden Paragraphen beziehen. Die USE-Anweisung muß mit einem Punkt und einem Leerzeichen enden.

Der Rest des Kapitels besteht aus null, einem oder mehreren Paragraphen, die die Prozeduren definieren, welche auszuführen sind, wenn die in der USE-Anweisung spezifizierten Bedingungen gegeben sind.

Zusätzliche DECLARATIVE-Kapitel können geschrieben werden, um andere DECLARATIVE-Prozeduren für andere Dateien, auf die im COBOL-Programm Zugriff genommen wird, zu spezifizieren. Jedes DECLARATIVE-Kapitel hat die Struktur, wie sie im Format gezeigt wurde.

Das Ende der DECLARATIVES wird durch die Schlüsselwörter END DECLARATIVES, abgeschlossen durch einen Punkt und ein Leerzeichen, angezeigt. Die Schlüsselwörter END DECLARATIVES beginnen in Spalte 8, 9, 10 oder 11 des Programmblattes. Kein anderer Text darf auf derselben Zeile erscheinen.

Die USE-Anweisung

Die USE-Anweisung spezifiziert die Bedingungen zum Übergang der Logik auf die Prozeduren in einem DECLARATIVE-Kapitel.

```
USE [GLOBAL] AFTER STANDARD {EXCEPTION} PROCEDURE ON
                            {ERROR    }

    {Datei-Name-1 [, Datei-Name-2] ... }
    {INPUT                             }
    {OUTPUT                            }
    {I-O                               }
    {EXTEND                            }
```

Die USE-Anweisung folgt unmittelbar auf die Kapitel-Überschrift eines DECLARATIVE-Kapitels. Die USE-Anweisung ist die einzige Anweisung in einem Satz; somit muß sie mit einem Punkt und einem Leerzeichen abgeschlossen sein. Auf die USE-Anweisung folgen null, ein oder mehrere Paragraphen, die diejenigen Prozeduren definieren, die auszuführen sind, wenn eine bestimmte Bedingung gegeben ist.

Die USE-Anweisung selbst wird nie ausgeführt. Ihr Zweck ist es lediglich, die zur Ausführung der unmittelbar folgenden Prozeduren notwendige Bedingung zu definieren.

Die mit einer USE-Anweisung verbundenen Prozeduren in dem DECLARATIVE-Kapitel sind auszuführen, wenn eine der folgenden Bedingungen auftritt:

- Die Beendigung der durch das Betriebssystem vorgesehenen Ein-Ausgabe-Fehler-Routine;

- Die Erkennung einer AT END-Bedingung, wenn die AT END-Angabe nicht in einer Ein-Ausgabe-Anweisung spezifiziert wurde;

- die Erkennung einer INVALID KEY-Bedingung, wenn die INVALID KEY-Angabe nicht in einer Ein-Ausgabe-Anweisung spezifiziert wurde.

Das in dem FILE-CONTROL-Eintrag definierte FILE STATUS-Datenfeld muß getestet werden, um die Bedingung zu bestimmen, die die Ausführung der DECLARATIVE-Prozeduren verursachte. Enthält das FILE STATUS-Datenfeld 10, dann ist in der Datei die AT END-Bedingung aufgetreten. Enthält das FILE STATUS-Datenfeld einen Wert von 20 bis 24, dann ist in der Datei die INVALID KEY-Bedingung aufgetreten. Enthält das FILE STATUS-Datenfeld einen Wert von 30 oder noch darüber, dann hat das Betriebssystem seine Ein-Ausgabe-Fehler-Routine für die ERROR Bedingung beendet.

Die Wörter ERROR und EXCEPTION sind gleichbedeutend.

Das Auftreten des reservierten Wortes INPUT in einer USE-Anweisung zeigt an, daß die zugeordneten DECLARATIVE-Prozeduren immer dann ausgeführt werden, wenn die entsprechenden Bedingungen in einer Datei bestehen, die als Eingabedatei eröffnet ist. Somit kann, wenn in einem Programm mehrere Eingabedateien vorgesehen sind, jede davon die Ausführung der zugeordneten DECLARATIVE-Prozeduren verursachen.

Das Auftreten des reservierten Wortes OUTPUT in einer USE-Anweisung zeigt an, daß die zugeordneten DECLARATIVE-Prozeduren immer dann ausgeführt werden, wenn die entsprechenden Bedingungen in einer Datei bestehen, die als Ausgabedatei eröffnet ist. Folglich kann, wenn in einem Programm mehrere Ausgabedateien vorgesehen sind, jede davon die Ausführung der zugeordneten DECLARATIVE-Prozeduren verursachen.

Das Auftreten des reservierten Wortes I-O in einer USE-Anweisung zeigt an, daß die zugeordneten DECLARATIVE-Prozeduren immer dann ausgeführt werden, wenn die entsprechenden Bedingungen in einer Datei bestehen, die als I-O-Datei eröffnet ist. Folglich kann, wenn in einem Programm mehrere I-O-Dateien vorgesehen sind, jede davon die Ausführung der zugeordneten DECLARATIVE-Prozeduren verursachen.

Das Auftreten des reservierten Wortes EXTEND in einer USE-Anweisung zeigt an, daß die zugeordneten DECLARATIVE-Prozeduren immer dann ausgeführt werden, wenn die entsprechenden Bedingungen in einer Datei bestehen, die als EXTEND-Datei eröffnet ist. Somit kann, wenn in einem Programm mehrere EXTEND-Dateien vorgesehen sind, jede davon die Ausführung der zugeordneten DECLARATIVE-Prozeduren verursachen.

Das Auftreten eines oder mehrerer Datei-Namen in einer USE-Anweisung zeigt an, daß die zugeordneten DECLARATIVE-Prozeduren immer dann ausgeführt werden, wenn die entsprechenden Bedingungen in einer der in der USE-Anweisung genannten Dateien bestehen.

Die Dateien, auf die in einer USE-Anweisung indirekt oder ausdrücklich Bezug genommen wird, brauchen nicht alle denselben Aufbau oder Zugriff haben.

Führt eine Ein-Ausgabe-Operation zur Erkennung eines Ein-Ausgabe-Fehlers, einer AT END-Bedingung oder einer INVALID KEY-Bedingung, dann wird die Steuerung automatisch an das entsprechende DECLARATIVE-Kapitel übertragen. Das Ende dieses DECLARATIVE-Kapitels verursacht wiederum eine Übertragung der Steuerung an die Ein-Ausgabe-Anweisung, die zur Ausführung des DECLARATIVE-Kapitels führte.

Aus in einem DECLARATIVE-Kapitel geschriebenen Prozeduren darf nicht auf irgendeine NONDECLARATIVE-Prozedur, d. h. auf Prozeduren, die nach END DECLARATIVES geschrieben sind, verzweigt werden.

Anweisungen in den NONDECLARATIVE-Prozeduren dürfen sich andererseits nicht auf Kapitel-Namen und Paragraphen-Namen in den DECLARATIVES beziehen. Die einzige Ausnahme dabei ist, daß eine PERFORM-Anweisung in den NONDECLARATIVE-Prozeduren sich auf einen Kapitel-Namen oder Paragraphen-Namen in den DECLARATIVES beziehen kann.

Besondere Abfolge-Regeln gelten, wenn Programme in anderen Programmen enthalten sind. Im Rahmen dieser Regeln wird jeweils das erste DECLARATIVE aufgerufen, das sich qualifiziert. Die Auswahlreihenfolge für ein DECLARATIVE ist folgende:

- Zuerst das DECLARATIVE innerhalb des Programmes, das den Ein-Ausgabe-Befehl enthält, welcher den Datei-Status hervorgerufen hat.

- Dann das DECLARATIVE, in dem die GLOBAL-Angabe vorhanden ist und das in demjenigen Programm steht, welches direkt das Programm enthält, welches den Datei-Status hervorgerufen hat.

- Zuletzt dasjenige DECLARATIVE, das im Rahmen ineinander verschachtelter Programme - nach auswärts gehend bis zum äußersten Programm - als erstes auftritt.

Das nachfolgende Beispiel der DECLARATIVE-Prozeduren zeigt erstens die DECLARATIVE-Prozeduren für eine indexierte Datei, genannt IND-DATEI, die im INPUT-Modus eröffnet ist. Das FILE STATUS-Datenfeld für IND-DATEI ist ein zwei Zeichen umfassendes alphanumerisches Datenfeld, geenannt IND-STATUS, das in der WORKING-STORAGE SECTION definiert ist. Erkennt eine der drei Ein-Ausgabe-Anweisungen für IND-DATEI (OPEN, READ oder CLOSE) entweder eine Ein-Ausgabe-Fehler-Bedingung oder eine INVALID KEY-Bedingung, dann geht die Steuerung zum DELCARATIVE-Kapitel, IND-DECL genannt, über. Alle möglichen Werte des FILE STATUS-Datenfeldes für die OPEN-, CLOSE- und READ-Anweisung auf eine indexierte Datei werden in den DECLARATIVE-Prozeduren getestet. Der Programmierer muß auszuführende Prozeduren für jeden der möglichen Werte des FILE STATUS-Datenfeldes schreiben. Der Rücksprung zur Ein-Ausgabe-Anweisung, die die Ausführung der DECLARATIVE-Prozeduren IND-DECL verursachte, wird durch das Ende des IND-DECL-Kapitels erreicht. Das Ende des IND-DECL-Kapitels ist gekennzeichnet durch den Null-Paragraphen mit dem Paragraphen-Namen IND-DECL-ENDE auf der Zeile 005980.

Ein Beispiel zeigt ferner die DECLARATIVE-Prozeduren für eine sequentielle Datei, SEQ-DATEI genannt, die im INPUT-Modus eröffnet ist. Das FILE STATUS-Datenfeld für SEQ-DATEI ist ein zwei Zeichen umfassendes alphanumerisches Datenfeld, SEQ-STATUS genannt, das in der WORKING-STORAGE SECTION definiert ist. Erkennt eine der drei Ein-Ausgabe-Anweisungen für SEQ-DATEI (OPEN, READ oder CLOSE) eine Ein-Ausgabe-Fehler-Bedingung oder eine AT END-Bedingung, dann geht die Steuerung zum DECLARATIVE-Kapitel, SEQ-DECL genannt, über. Der Rücksprung zur Ein-Ausgabe-Anweisung, die die Ausführung der DECLARATIVE-Prozeduren SEQ-DECL verursachte, wird durch das Ende des SEQ-DECL Kapitels erreicht. Das Ende des SEQ-DECL-Kapitels ist gekennzeichnet durch den Null-Paragraphen mit dem Paragraphen-Namen SEQ-DECL-ENDE auf der Ursprungszeile 006580.

```
1       8
002070 FILE-CONTROL.
002080     SELECT IND-DATEI ASSIGN TO DISC
002090                      ORGANIZATION IS INDEXED
002100                      ACCESS MODE IS RANDOM
002110                      RECORD KEY IS IND-SCHL
002120                      FILE STATUS IS IND-STATUS.
002130     SELECT SEQ-DATEI ASSIGN TO DISC
002140                      FILE STATUS IS SEQ-STATUS.

003010 DATA DIVISION.
003020 FILE SECTION.
003030 FD  IND-DATEI BLOCK CONTAINS 25 RECORDS
003040               RECORD CONTAINS 20 CHARACTERS
003050               LABEL RECORD IS STANDARD
003060               VALUE OF FILE-ID IS "IND-DATEI".
003070 01  IND-SATZ.
003080     02  IND-SCHL    PIC X(6).
003090     02  IND-DATEN   PIC X(14).
003100
003110 FD  SEQ-DATEI BLOCK CONTAINS 11 RECORDS
003120               RECORD CONTAINS 46 CHARACTERS
003130               LABEL RECORD IS STANDARD
003140               VALUE OF FILE-ID IS "SEQ-DATEI".
003150 01  SEQ-SATZ.
003160     02  SEQ-KTO-NR  PIC X(9).
003170     02  SEQ-CODE    PIC X(6).
003180     02  SEQ-DATEN   PIC X(31).

004010 WORKING-STORAGE SECTION.
004020 77  IND-STATUS      PIC XX.
004030 77  SEQ-STATUS      PIC XX.
```

```
005010 PROCEDURE DIVISION.
005020 DECLARATIVES.
005030 IND-DECL SECTION.
005040     USE AFTER ERROR PROCEDURE ON IND-DATEI.
005050 IND-DATEI-TEST.
005060     IF IND-STATUS = "23" GO TO KEIN-SATZ.
005070     IF IND-STATUS = "30" GO TO FEHL-30.
                ------------------------------
005180     IF IND-STATUS = "99" GO TO FEHL-99.
005190*        **UNGUELTIGER FILE STATUS WERT**
                ------------------------------
005280     GO TO IND-DECL-ENDE.
005290
005300 KEIN-SATZ. ------------------------------
005460     GO TO IND-DECL-ENDE.
005470
005480 FEHL-30.   ------------------------------
005690     GO TO IND-DECL-ENDE.
005700
005710 FEHL-99.   ------------------------------
005980 IND-DECL-ENDE.

006010 SEQ-DECL SECTION.
006020     USE AFTER STANDARD EXCEPTION PROCEDURE ON SEQ-DATEI.
006030 SEQ-DATEI-STATUS.
006040     IF SEQ-STATUS = "10" GO TO ENDE-SEQ-DATEI.
006050     IF SEQ-STATUS = "30" GO TO SEQ-FEHL-30.
006060*        **UNGUELTIGER FILE STATUS WERT**
006310     GO TO SEQ-DECL-ENDE.
006320
006330 ENDE-SEQ-DATEI. ------------------------------
006450     GO TO SEQ-DECL-ENDE.
006460
006470 SEQ-FEHL-30.
                ------------------------------
006580 SEQ-DECL-ENDE.
006590 END DECLARATIVES.

007010 PROGR-BEGINN SECTION.
007020 BEGINN.
007030     OPEN INPUT IND-DATEI, SEQ-DATEI.
                ------------------------------
007150     READ SEQ-DATEI.
                ------------------------------
007310     READ IND-DATEI.
                ------------------------------
007690     CLOSE SEQ-DATEI IND-DATEI.
```

KAPITEL 9 : PROZEDURANWEISUNGEN

Eine Prozeduranweisung drückt einen Vorgang aus, der während der Ausführung des Objekt-Programmes erfolgen muß. Sie beginnt mit einem Verb und endet mit der Kombination von Wörtern, die die zu bearbeitenden Daten bezeichnen. Der auf das Verb folgende Teil der Anweisung besteht aus Schlüsselwörtern, Wahlwörtern und Operanden.

Jede Prozeduranweisung hat ihr eigenes besonderes Format, das die Typen der erforderlichen Wörter und die Organisation der Anweisung beschreibt. Es ist notwendig, alle in einer Anweisung erforderlichen Wörter darin aufzunehmen und sie dem Compiler in der vorgeschriebenen Form zu unterbreiten. Werden diese beiden Erfordernisse nicht erfüllt, dann kann der Compiler die Anweisungen nicht übersetzen.

Die folgenden Seiten bringen jede COBOL Prozeduranweisung im Detail: die Verwendung, das Format und auch alle Angaben, die vorgesehen werden können, um den Compiler mit zusätzlicher Information zu versorgen. Nachfolgend werden die Prozeduranweisungen für NCR ITX-COBOL nach Verben alphabetisch geordnet aufgeführt.

DIE ACCEPT-ANWEISUNG

Mit der ACCEPT-Anweisung können Daten geringen Umfangs in Speicherfelder des Computers übernommen werden.

Format 1:

```
ACCEPT  Identifier-1 [ LINE { Identifier-2 } ][ POSITION { Identifier-3 } ]
                            { Literal-1    }              { Literal-2    }

SIZE { Identifier-4 } [ UPDATE ]
     { Literal-3    } [ PROMPT ] [Literal-4] [ECHO] [CONVERT] [TAB] [ERASE

[ EOL ] [NO BEEP]
[ EOS ]

[EXACT] [OFF] [ HIGH ] [BLINK] [REVERSE] ... [ON EXCEPTION
              [ LOW  ]

                    Identifier-5 unbedingte Anweisung]
```

Format 2:

```
ACCEPT Identifier FROM { DATE        }
                       { DAY         }
                       { DAY-OF-WEEK }
                       { TIME        }
```

Format 3:

```
ACCEPT Identifier [FROM mnemonic-name ]
```

Zu Format 1:

Bei der Ausführung der ACCEPT-Anweisung überträgt der Computer Daten von einem Bildschirm in das mit Identifier-1 definierte Datenfeld. Diese Übertragung findet statt, wenn der Operator die Eingabe von Daten am Bildschirm entweder durch Eingabe der genauen in der ACCEPT-Anweisung geforderten Anzahl von Zeichen oder durch Drücken der NEWLINE-Taste beendet hat.

Die Übertragung von Daten vom Bildschirm zum Empfangsdatenfeld erfolgt nach den Regeln der MOVE-Anweisung.

Das als Identifier-1 definierte Datenfeld darf nicht USAGE IS INDEX oder USAGE IS BIT aufweisen.

Es folgt nun eine Beschreibung der Wahlangaben der ACCEPT-Anweisung, die in jeder beliebigen Reihenfolge stehen können.

LINE-Angabe

Der Wert von Identifier-2 oder Literal-1 in der LINE-Angabe spezifiziert die Nummer der Zeile, von der die Daten vom Bildschirm des Terminals übernommen werden sollen. Identifier-2 muß ein numerisches Datenfeld sein. Literal-1 muß ein numerisches Literal sein. Der Wert der Zeilennummer reicht von 1 bis 24. Beim Betriebssystem NCR ITX werden die Zeilen 23 und 24 jedoch auch für System-Nachrichten benutzt. Ist die LINE-Angabe in der ACCEPT-Anweisung nicht vorhanden, dann werden die Daten von der nächsttieferen Zeile unterhalb der gerade gültigen Position des Cursors auf dem Bildschirm übernommen. Steht der Cursor gerade auf Zeile 24, dann rollt das Bild auf dem Bildschirm und der Cursor wird auf Zeile 24 positioniert. Das Rollen des Bildes auf dem Bildschirm besteht darin, daß die Daten von Zeile 2 bis 24 auf die Zeilen 1 bis 23 geschoben werden und auf Zeile 24 eine neue Leerzeile entsteht. Die Daten auf Zeile 1 gehen verloren.

Bei Zeilennummer 0 und Cursor-Position 0 wird der Cursor nicht bewegt. Bei Zeilennummer 0 und einer Cursor-Position ungleich 0 wird der Cursor auf der nächsten Zeile an der angegebenen Position positioniert. Ist der Cursor ständig auf Zeile 24, dann rollt das Bild auf dem Bildschirm und der Cursor wird auf Zeile 24 positioniert.

POSITION-Angabe

Der Wert von Identifier-3 oder Literal-2 in der POSITION-Angabe spezifiziert die Position, auf die der Cursor innerhalb der bezeichneten Zeile vor der Übernahme von Daten vom Terminal in den Speicher zu positionieren ist. Dieser Wert ist die Position des ersten Zeichens (links) des Eingabefeldes, in das die Daten durch die ACCEPT-Anweisung in den Speicher eingegeben werden. Identifier-3 muß ein numerisches Literal ohne Vorzeichen sein.

Der Wert der Cursor-Position reicht von 1 bis 80, wobei der Wert 1 den Cursor ganz links auf dem Bildschirm und der Wert 80 diesen ganz rechts auf dem Bildschirm positioniert. Wird in einer ACCEPT-Anweisung die POSITION-Angabe weggelassen, dann wird der Cursor ganz links auf dem Bildschirm positioniert.

Bei Cursor-Positionszahl 0 und Zeilennummer 0 bleibt der Cursor stehen. Ist die Cursor-Positionszahl ungleich 0 und die Zeilennummer 0, wird der Cursor auf der nächsten Zeile an die angezeigte Position gesetzt. Ist der Cursor ständig auf Zeile 24 positioniert, dann rollt das Bild auf dem Bildschirm, und der Cursor wird immer auf Zeile 24 positioniert.

SIZE-Angabe

Der Wert von Identifier-4 oder Literal-3 in der SIZE-Angabe spezifiziert die Anzahl von Bytes, die vom Bildschirm in den Speicher zu übermitteln sind. Identifier-4 muß ein numerisches Datenfeld ohne Vorzeichen sein. Literal-3 muß ein numerisches Literal ohne Vorzeichen sein. Der in der SIZE-Angabe spezifizierte Wert reicht von 1 bis 80. Hat der Operator weniger Zeichen als in der ACCEPT-Anweisung angegeben sind, eingegeben, dann werden die übernommenen Daten im Identifier-1 gemäß den Regeln der MOVE-Anweisung positioniert. Hat der Operator mehr Zeichen als in der ACCEPT-Anweisung angegeben sind, eingegeben, dann werden die zuviel eingegebenen Zeichen rechts abgeschnitten. Ist der Wert des Identifiers-4 oder des Literals-3 0 oder ist keine SIZE-Angabe in der ACCEPT-Anweisung enthalten, dann wird angenommen, daß die Anzahl Bytes in der Datenfeldbeschreibung von Identifier-1 diejenige Anzahl Bytes ist, die vom Bildschirm zu übermitteln ist.

PROMPT-Angabe

Wenn in einer ACCEPT-Anweisung die PROMPT-Angabe steht, werden an der Position des Bildschirms, an der die Dateneingabe erfolgt, Sterne angezeigt. Nach der Ausgabe der Sterne wird der Cursor unter den ersten Stern positioniert. Wenn die PROMPT-Angabe nicht gemacht wird, müssen die Daten "blind" eingegeben werden.

ECHO-Angabe

Das Schlüsselwort ECHO in einer ACCEPT-Anweisung bewirkt, daß der Inhalt des Identifiers-1 auf dem Bildschirm angezeigt wird, nachdem durch die ACCEPT-Anweisung Daten eingegeben wurden. Somit haben die Daten durch die ECHO-Funktion auf dem Bildschirm die gleiche Anzahl von Zeichen, wie sie im Identifier-1 enthalten sind. Ist Identifier-1 alphanumerisch, dann werden die Daten linksbündig angezeigt. Ist Identifier-1 numerisch, dann werden die Daten dezimalstellengerecht angezeigt. Wird ECHO in der ACCEPT-Anweisung weggelassen, dann werden die Daten wie eingegeben auf dem Bildschirm belassen.

CONVERT-Angabe

Das Schlüsselwort CONVERT in einer ACCEPT-Anweisung verursacht die Umwandlung eines eingegebenen numerischen Feldes in ein ungepacktes dezimales Feld mit Vorzeichen. Das ungepackte dezimale Feld mit Vorzeichen wird dann durch die ACCEPT-Anweisung im Identifier-1 gespeichert. Die Umwandlung eines eingegebenen numerischen Feldes wird durch eine von links nach rechts erfolgende byteweise Prüfung erreicht.

Das Vorzeichen des ungepackten dezimalen Feldes wird gleich an das äußerste rechte Zeichen des numerischen Eingabefeldes gesetzt. Ohne Vorzeichen ist das numerische Eingabefeld ein ungepacktes dezimales Feld mit positivem Vorzeichen.

Der Dezimalseparator in der numerischen Eingabe (Punkt oder ggf. Komma) bestimmt die Anzahl Dezimalstellen in einem ungepackten dezimalen Datenfeld. Ist kein Dezimalseparator in dem numerischen Eingabefeld, dann werden für das ungepackte dezimale Feld mit Vorzeichen null Dezimalstellen angenommen.

Alle nicht-numerischen Zeichen in einem numerischen Eingabefeld werden bei Positionierung in einem ungepackten dezimalen Feld mit Vorzeichen elimiert.

Wenn sowohl ECHO als auch CONVERT in derselben ACCEPT-Anweisung auftreten, erfolgt die numerische Umwandlung, bevor, durch die ECHO-Angabe bedingt, der Identifier-1 auf dem Bildschirm angezeigt wird.

ERASE-Angabe

Sie bewirkt das Löschen des kompletten Bildschirms vor Durchführung sämtlicher anderen Leistungen der Anweisung ACCEPT.

EXACT-Angabe

Sie bewirkt, daß die exakte Anzahl Zeichen (wie in SIZE angegeben oder - wenn SIZE weggelassen - entsprechend der Länge des Eingabefeldes Identifier-1) in den Speicher eingeht. Das Mitzählen von Backspace-Anschlägen hierbei ist eine Option im ITX-Sysgen.

Bei Füllung des Identifier-1 braucht nicht die NEW LINE-Taste gedrückt zu werden.

ON EXCEPTION-Angabe

Bei Eingabe eines ungültigen Zeichens wird die hier programmierte unbedingte Anweisung ausgeführt. Dabei steht das ungültige Zeichen (im ASCII-Format) in Identifier-5 zur Verfügung, der als 1-Byte-alphanumerisches Feld definiert sein muß.

Gültige Zeichen sind diejenigen mit den ASCII-Werten Space (hex 20) bis ~ (hex 7E). Alle anderen Zeichen führen zu einer ON EXCEPTION-Situation.

Beim Vorhandensein sowohl der CONVERT- als auch der ON EXCEPTION-Angabe in der ACCEPT-Anweisung verursacht ein Konversionsfehler, daß der Hex-Wert 98 in den Identifier-5 übertragen wird, bevor die unbedingte Anweisung ausgeführt wird.

Beispiel 1:

```
1       8
            DISPLAY "EINGABE DER 5-STELL. KUNDEN-NR -"
                    LINE 12  POSITION 25  ERASE.
            ACCEPT KUNDEN-NR  LINE 12  POSITION 57.
```

Die durch die DISPLAY-Anweisung ausgelösten Vorgänge sind folgende:

- Der Bildschirm wird gelöscht.

- Der Cursor wird auf Position 25 der Zeile 12 gesetzt.

- Die 32 Bytes mit dem Inhalt EINGABE DER 5-STELL. KUNDEN-NR - werden auf Zeile 12 von Position 25 an ausgegeben.

Die durch die ACCEPT-Anweisung ausgelösten Vorgänge sind folgende:

- Der Cursor wird auf Position 57 der Zeile 12 gesetzt.

- Die Steuerung geht über an den Bildschirm, damit die Eingabe durch den Operator erfolgen kann.

- Beginnend an Position 57 in Zeile 12 gibt der Operator 52416 ein und die Steuerung kehrt automatisch zum Programm zurück, da KUNDEN-NR ein fünf Bytes umfassendes Datenfeld ist.

- Die Daten 52416 ab Position 57 bis 61 der Zeile 12 werden in das fünf Bytes umfassende, KUNDEN-NR genannte, numerische Datenfeld positioniert.

Beispiel 2:

```
1       8
            DISPLAY "FLUG-NUMM. EINGEBEN" LINE 8 POSITION 30 ERASE.
            ACCEPT FLUG-NR LINE ZEILE-NR POSITION 39 ECHO.
```

Die durch die DISPLAY-Anweisung ausgelösten Vorgänge sind folgende:

- Der Bildschirm wird gelöscht.
- Der Cursor wird auf Position 30 der Zeile 8 gesetzt.
- Die 19 Bytes mit dem Inhalt FLUG-NUMM. EINGEBEN werden auf Zeile 8, von Position 30 an, ausgegeben.

Die durch die ACCEPT-Anweisung ausgelösten Vorgänge sind folgende:

- Unter der Annahme, daß ZEILE-NR 10 enthält, wird der Cursor auf Position 39 der Zeile 10 gesetzt.
- Die Steuerung geht über an den Bildschirm, damit die Eingabe durch den Operator erfolgen kann.
- Beginnend bei Position 39 der Zeile 10 gibt der Operator die Zahl 121 ein und drückt die NEW-LINE-Taste, um dadurch die Steuerung dem Programm zurückzugeben.
- Die Daten 121, beginnend ab Position 39 der Zeile 10, werden übernommen.
- 121 wird rechtsbündig mit führenden Nullen nach links in das vier Bytes umfassende, FLUG-NR genannte, numerische Datenfeld abgestellt.
- Der Inhalt 0121 in FLUG-NR wird auf dem Bildschirm ab Position 39 der Zeile 10 angezeigt.

Beispiel 3:

```
1       8
            DISPLAY "FLUG-NUMM. EINGEBEN" LINE 8 POSITION 30 ERASE
            ACCEPT FLUG-NR LINE ZEILE-NR POSITION 39 ECHO.
```

Die durch die DISPLAY-Anweisung ausgelösten Vorgänge sind folgende:

- Der Bildschirm wird gelöscht.
- Der Cursor wird auf Position 30 der Zeile 8 gesetzt.
- Die 19 Bytes, mit Inhalt FLUG-NUMM. EINGEBEN, werden auf Zeile 8 von Position 30 an ausgegeben.

Die durch die ACCEPT-Anweisung ausgelösten Vorgänge sind folgende:

- Angenommen, daß ZEILE-NR 11 enthält, wird der Cursor auf Position 39 oder Zeile 11 gesetzt.
- Die Steuerung geht über auf den Bildschirm, damit die Eingabe durch den Operator erfolgen kann.
- Beginnend ab Position 39 der Zeile 11 gibt der Operator die Zahl 1234 ein und die Steuerung kehrt automatisch zum Programm zurück, da FLUG-NR ein vier Bytes umfassendes Datenfeld ist.
- Die Daten 1324 werden in das vier Bytes umfassende, FLUG-NR genannte, numerische Datenfeld gespeichert.
- Die Daten 1234 in FLUG-NR werden auf dem Bildschirm ab Position 39 der Zeile 11 angezeigt.

Beispiel 4:

```
1       8
            DISPLAY "EINGABE-ENDE?". LINE 20 POSITION 1 ERASE.
            ACCEPT ANTWORT LINE 20 POSITION 16 ECHO.
```

Die durch die DISPLAY-Anweisung ausgelösten Vorgänge sind folgende:

- Der Bildschirm wird gelöscht.
- Der Cursor wird auf Position 1 der Zeile 20 gesetzt.
- Die 13 Bytes, mit dem Inhalt EINGABE-ENDE?, werden auf Zeile 20 ab Position 1 ausgegeben.

Die durch die ACCEPT-Anweisung ausgelösten Vorgänge sind folgende:

- Der Cursor wird auf Position 16 der Zeile 20 gesetzt.
- Die Steuerung geht über den Bildschirm, damit die Eingabe durch den Operator erfolgen kann.
- Beginnend ab Position 16 der Zeile 20 gibt der Operator JA ein und die Steuerung kehrt automatisch zum Programm zurück, da ANTWORT ein zwei Bytes umfassendes Datenfeld ist.
- Die Daten JA ab Position 16 der Zeile 20 werden übernommen.
- Die Daten JA werden in das zwei Bytes umfassende, ANTWORT genannte, alphanumerische Datenfeld gespeichert.
- Die Daten JA in ANTWORT werden auf dem Bildschirm ab Position 16 der Zeile 20 angezeigt.

Beispiel 5:

```
1       8
            DISPLAY "FAM.-NAME EING." LINE 1 ERASE.
            ACCEPT NAME LINE 1 POSITION 17 PROMPT.
```

Die durch die DISPLAY-Anweisung ausgelösten Vorgänge sind folgende:

- Der Bildschirm wird gelöscht.
- Der Cursor wird auf Position 1 der Zeile 1 gesetzt.
- Die 15 Bytes mit dem Inhalt FAM.-NAME EING. werden auf Zeile 1 ab Position 1 ausgegeben.

Die durch die ACCEPT-Anweisung ausgelösten Vorgänge sind folgende:

- Der Cursor wird auf Position 17 der Zeile 1 gesetzt.
- Stern-Zeichen (*) werden auf dem Bildschirm von Position 17 bis 44 der Zeile 1 angezeigt.
- Der Cursor wird auf Position 17 der Zeile 1 gesetzt.
- Die Steuerung geht über auf den Bildschirm, damit die Eingabe durch den Operator erfolgen kann.
- Beginnend ab Position 17 der Zeile 1 gibt der Operator MEIER ein und drückt die NEWLINE-Taste, um dadurch die Steuerung dem Programm zurückzugeben.
- Die in Positionen 17 bis 21 der Zeile 1 des Bildschirms befindlichen Daten MEIER werden linksbündig mit Leerstellen nach rechts in das 28 Bytes umfassende, NAME genannte, alphanumerische Feld gespeichert.

Beispiel 6:

```
1       8
            DISPLAY "SALDO-EINGABE" LINE 11 POSITION 34 ERASE.
            ACCEPT SALDO LINE 11 POSITION 49 ECHO CONVERT.
```

Die durch die DISPLAY-Anweisung ausgelösten Vorgänge sind folgende:

- Der Bildschirm wird gelöscht.
- Der Cursor wird an Position 34 der Zeile 11 gesetzt.
- Die 13 Bytes mit dem Inhalt SALDO-EINGABE werden auf Zeile 11 ab Position 34 ausgegeben.

Die durch die ACCEPT-Anweisung ausgelösten Vorgänge sind folgende:

- Der Cursor wird auf Position 49 der Zeile 11 gesetzt.
- Die Steuerung geht über an den Bildschirm, damit die Eingabe durch den Operator erfolgen kann.
- Beginnend ab Position 49 der Zeile 11 gibt der Operator -150.00 ein und die Steuerung kehrt automatisch zum Programm zurück, da SALDO ein sieben Bytes umfassendes Datenfeld ist.
- Die Daten -150.00, beginnend ab Position 49 der Zeile 11, werden vom Bildschirm übernommen.
- Die Daten -150.00 werden in ein ungepacktes Feld mit Vorzeichen 015000- umgewandelt und in das sieben Bytes umfassende, SALDO genannte, numerische Feld gespeichert.
- Die Daten 015000- in SALDO werden, beginnend ab Position 49 der Zeile 11, auf dem Bildschirm angezeigt.

Beispiel 7:

```
1       8
            DISPLAY "LAGER-NUMMER EINGABE" LINE 16 POSITION 10
                                                   ERASE.
            ACCEPT LAG-NR LINE 18 POSITION 10 ECHO.
```

Die durch die DISPLAY-Anweisung ausgelösten Vorgänge sind folgende:

- Der Bildschirm wird gelöscht.
- Der Cursor wird ab Position 10 der Zeile 16 gesetzt.
- Die 20 Bytes mit dem Inhalt LAGER-NUMMER EINGABE werden auf Zeile 16 von Position 10 an ausgegeben.

Die durch die ACCEPT-Anweisung ausgelösten Vorgänge sind folgende:

- Der Cursor wird auf Position 10 der Zeile 18 gesetzt.
- Die Steuerung geht über auf den Bildschirm, damit die Eingabe durch den Operator erfolgen kann.
- Beginnend ab Position 10 der Zeile 18 gibt der Operator AB1 ein und drückt die NEWLINE-Taste, um dadurch die Steuerung dem Programm zurückzugeben.
- Die Daten AB1, beginnend ab Position 10 der Zeile 18, werden vom Bildschirm übernommen.
- Die Daten AB1 werden linksbündig mit Leerstellen nach rechts in dem vier Bytes umfassenden, LAG-NR genannten alphanumerischen Datenfeld gespeichert.
- Die linksbündigen Daten AB1 werden, beginnend ab Position 10 der Zeile 18, auf dem Bildschirm angezeigt.

Zu Format 2:

DATE

DATE in der ACCEPT-Anweisung bewirkt, daß das System-Datum in das durch den Identifier bezeichnete Datenfeld nach den Regeln der MOVE-Anweisung übertragen wird. Der COBOL-Compiler berücksichtigt DATE als ein elementares Datenfeld mit numerischen Ganzzahlen ohne Vorzeichen, mit einer Länge von sechs Ziffern. DATE setzt sich zusammen aus Jahr, dem Monat des Jahres und dem Tag des Monats; folglich wird der 1. Februar 1990 so ausgedruckt: 900201. Der Programmierer macht keine Datenfeldbeschreibung für das Datenfeld DATE. Die Information dafür wird vom System-Datum innerhalb des Betriebssystems gewonnen.

DAY

DAY in der ACCEPT-Anweisung bewirkt, daß das System-Datum umgewandelt und in das durch den Identifier bezeichnete Datenfeld nach den Regeln der MOVE-Anweisung übertragen wird. Der COBOL-Compiler berücksichtigt DAY als ein elementares Datenfeld mit numerischen Ganzzahlen ohne Vorzeichen, in einer Länge von fünf Ziffern. DAY setzt sich zusammen aus dem Jahr und dem Tag des Jahres; folglich lautet der 1. Februar 1990 so: 90032. Der Programmierer macht keine Datenfeldbeschreibung für das Datenfeld DAY. Die Information dafür wird vom System-Datum innerhalb des Betriebssystems gewonnen.

DAY-OF-WEEK

DAY-OF-WEEK in der ACCEPT-Anweisung bewirkt, daß der Wochentag in das vom Identifier bezeichnete Datenfeld nach den Regeln der MOVE-Anweisung übertragen wird. Der Wochentag wird durch nur eine Ziffer dargestellt, z. B. Montag durch 1, Dienstag durch 2, usw.

TIME

TIME in der ACCEPT-Anweisung bewirkt, daß die Uhrzeit in das durch den Identifier bezeichnete Datenfeld nach den Regeln der MOVE-Anweisung übertragen wird. Der COBOL-Compiler berücksichtigt TIME als ein elementares Datenfeld mit numerischen Ganzzahlen ohne Vorzeichen, mit einer Länge von acht Ziffern. TIME setzt sich zusammen aus Stunden, Minuten, Sekunden und Hundertstelsekunden, TIME gibt die verstrichene Zeit nach Mitternacht an, und dies auf einer 24-Stunden-Uhr-Basis. Folglich wird 3.14 Uhr nachmittags ausgedruckt durch 15140000. Der Programmierer macht keine Datenfeldbeschreibung für das Datenfeld TIME. Die Information für das Datenfeld TIME wird von der Uhrzeit innerhalb des Betriebssystems gewonnen.

Beispiel 1:

```
1       8
        WORKING-STORAGE SECTION.
        77  DATUM   PIC 9(6).

        PROCEDURE DIVISION.
            ACCEPT DATUM FROM DATE.
```

DATUM vor Ausführung	**9 0 0 1 1 0**	**10. Januar 1990 anzeigend**
System-Datum	**9 0 0 2 1 7**	**17. Februar 1990 anzeigend**
DATUM nach Ausführung	**9 0 0 2 1 7**	**17. Februar 1990 anzeigend**

Beispiel 2:

```
1       8
        WORKING-STORAGE SECTION.
        77  F-TAG   PIC 9(5).

        PROCEDURE DIVISION.
            ACCEPT F-TAG FROM DAY.
```

F-TAG vor Ausführung	9 0 0 3 0	30. Januar 1990 anzeigend
System-Datum	9 0 0 2 0 1	1. Februar 1990 anzeigend
F-TAG nach Ausführung	9 0 0 3 2	1. Februar 1990 anzeigend

Beispiel 3:

```
1       8
        WORKING-STORAGE SECTION.
        77  TAG  PIC 9.

        PROCEDURE DIVISION.
            ACCEPT TAG FROM DAY-OF-WEEK.
```

An einem Freitag enthielte das Feld TAG nach Ausführung der ACCEPT-Anweisung den Inhalt 5.

Beispiel 4:

```
1       8
        WORKING-STORAGE SECTION.
        01  ZEIT.
            02  STD     PIC 99.
            02  MIN     PIC 99.
            02  SEK     PIC 99.
            02  FILLER  PIC 99.

        PROCEDURE DIVISION.
            ACCEPT ZEIT FROM TIME.
```

ZEIT vor Ausführung	1 0 2 0 0 0 0 0	10.20 Uhr vormittags anzeigend
System-Zeit	1 6 4 5 1 0	16.45 und 10 Sekunden anzeigend
ZEIT nach Ausführung	1 6 4 5 1 0 0 0	16.45 und 10 Sekunden anzeigend

Zu Format 3:

Der Mnemonic Name muß auch im Special-Names-Paragraphen der Environment Division definiert und auf die Konsole bezogen sein.

Fehlt die FROM-Klausel in der ACCEPT-Anweisung, gilt als Default-Eingabegerät ebenfalls die Konsole.

Ist der Umfang der eingegebenen Daten geringer als der Umfang des aufnehmenden Datenfeldes, werden die Daten linksbündig abgestellt.

Ist der Umfang der eingegebenen Daten größer als das aufnehmende Datenfeld, geht der Überhang verloren.

Numerische Datenfelder sind statthaft, da eine Konversion der Daten erfolgt.

Die Anweisung ACCEPT MESSAGE COUNT

Diese Anweisung stellt die Anzahl Nachrichten in einer MCS-Queue zur Verfügung.

Format:

ACCEPT cd-name-1 MESSAGE COUNT

Als cd-name-1 ist der Name einer Input Communication Description (CD) anzugeben.

Es wird das Feld MESSAGE COUNT innerhalb der Input Communication Description aktualisiert. Das heißt, es gibt die Anzahl an Nachrichten wieder, die in einer Queue, Sub-Queue, usw., existieren.

Bei Ausführung dieser Anweisung muß das entsprechende Feld in der Communication Description den Namen der zu befragenden Symbolic Queue beinhalten. Aktualisiert werden Data-Name-10 (STATUS KEY) und Data-Name-11 (MESSAGE COUNT). Weiter vorn im Buch unter dem Stichwort COMMUNICATION SECTION (in der DATA DIVISION) finden Sie mehr hierüber.

DIE ADD-ANWEISUNG

Die ADD-Anweisung addiert zwei oder mehr numerische Operanden und speichert das Ergebnis.

Format 1:

```
ADD {Identifier-1} [Identifier-2] ... TO Identifier-m [ROUNDED] ...
    {Literal-1   } [Literal-2   ]
    [ON SIZE ERROR unbedingte Anweisung 1]
    [NOT ON SIZE ERROR unbedingte Anweisung 2]
    [END-ADD]
```

Format 2:

```
ADD {Identifier-1} {Identifier-2} [Identifier-3] ...
    {Literal-1   } {Literal-2   } [Literal-3   ]
  GIVING Identifier-m [ROUNDED] [Identifier-n [ROUNDED]] ...
    [ON SIZE ERROR     unbedingte Anweisung 1]
  [NOT ON SIZE ERROR   unbedingte Anweisung 2]
  [END-ADD]
```

Bei Format 1 werden die Inhalte der dem Wort TO vorausgehenden Operanden den Inhalten der Identifiers-m, n, ... hinzuaddiert, die "Empfangsdatenfelder" oder "Ergebnisfelder" genannt werden.
Bei Format 2 werden die Inhalte der dem Wort GIVING vorausgehenden Operanden addiert. Das Ergebnis dieser Addition wird dann als der neue Inhalt im Identifier-m, n, ... gespeichert, welche "Empfangsfelder" oder "Ergebnisfelder" genannt werden.
Bei Format 1 muß jeder Identifier ein numerisches Elementarfeld bezeichnen. Bei Format 2 muß jeder Identifier außer den Identifier-m, n, ... ein numerisches Elementarfeld bezeichnen. Die Identifier-m, n, ... bei Format 2 können entweder numerische oder numerisch-editierte Elementarfelder sein. Jedes Literal in der ADD-Anweisung muß ein numerisches Literal sein.

Die Definitionen der Operanden in der ADD-Anweisung brauchen nicht identisch (gepackt oder ungepackt usw.) sein, denn jede notwendige Umwandlung und Dezimalpunktausrichtung erfolgt automatisch während des Rechnens. Die Operanden der ADD-Anweisung können von einem der folgenden Datentypen sein:

Datentyp	maximale Länge
Dezimal ohne Vorzeichen	18 Bytes
Ungepackt dezimal mit Vorzeichen links oder rechts angefügt	19 Bytes einschließlich 8-Bit-Vorzeichen
Ungepackt dezimal mit Zonenvorzeichen links oder rechts	18 Bytes einschließlich Zonen-Vorzeichen
Gepackt dezimal ohne Vorzeichen	9 Bytes
Gepackt dezimal mit Vorzeichen	10 Bytes einschließlich 4-Bit-Vorzeichen
Binär ohne Vorzeichen	8 Bytes

Bei beiden Formaten darf das Ergebnis nicht mehr als achtzehn Ziffern umfassen.

Wenn drei oder mehr Operanden oder Empfangsfelder in der ADD-Anweisung enthalten sind, generiert der COMPILER ein Zwischenspeicher-Feld. Nachdem das Ergebnis durch die ADD-Anweisung in das Zwischenspeicher-Feld abgestellt ist, wird das Ergebnis in die Empfangsfelder übertragen, und zwar in der Reihenfolge, wie sie in der ADD-Anweisung enthalten sind.

```
ADD F-1 F-2 F-3 TO F-4 F-5 F-6
        ist gleich wie
ADD F-1 F-2 F-3 GIVING Zwischensp. Feld
ADD Zwischensp. Feld TO F-4
ADD Zwischensp. Feld TO F-5
ADD Zwischensp. Feld TO F-6
```

ADD ist eine unbedingte Anweisung. Bei Vorhandensein von SIZE ERROR wird ADD jedoch zu einer bedingten Anweisung.

ROUNDED

Weist das Resultat der Addition mehr Dezimalstellen (Positionen rechts vom Dezimalpunkt) als das Ergebnisfeld auf, dann werden beide nach dem Dezimalpunkt ausgerichtet. Ohne die Angabe ROUNDED erfolgt ein Abschneiden. Bei Vorhandensein von ROUNDED erhöht sich der absolute Wert des Ergebnisfeldes immer dann um 1, wenn die erste nicht mehr aufzunehmende Ziffer größer als 4 ist.

SIZE ERROR

Nach Dezimalpunktausrichtung und Rundung wird bei Verwendung von SIZE ERROR eine Prüfung auf eine SIZE ERROR-Bedingung hin vorgenommen. Die SIZE ERROR-Bedingung liegt vor, wenn der Wert des Ergebnisses den höchsten vom Ergebnisfeld aufnehmbaren Wert überschreitet. Lediglich die linke Seite des Empfangsfeldes wird auf Überlauf hin geprüft. Bei Auftreten der SIZE ERROR-Bedingung wird der Wert des Ergebnisses nicht im Ergebnisfeld gespeichert. Bei mehreren Empfangsfeldern wird in allen das Ergebnis gespeichert, die keine SIZE ERROR-Bedingung verursachen. Wird in einem der Empfangsfelder eine SIZE ERROR-Bedingung festgestellt, wird die ADD-Anweisung normal zu Ende geführt und danach die unbedingte Anweisung ausgeführt. Die Schreibweise "unbedingte Anweisung" in der SIZE ERROR Angabe bezieht sich auf eine oder mehrere aufeinanderfolgende unbedingte Anweisungen, deren Folge durch einen Punkt beendet wird.

Ohne Vorliegen einer solchen Überlaufbedingung wird das Ergebnis der Addition in dem Ergebnisfeld gespeichert und die nächste Anweisung nach dem Punkt ausgeführt. Die unbedingte Anweisung in SIZE ERROR wird nicht ausgeführt.

Wird die SIZE ERROR-Klausel nicht verwendet, ist bei Auftreten einer SIZE ERROR-Bedingung der Inhalt des Ergebnisfeldes unvorhersagbar. Die auf die ADD-Anweisung folgende Anweisung wird ausgeführt und der Fehler nicht festgestellt.

Durch die Verwendung von NOT ON SIZE ERROR kann eine andere Ablauflogik gestaltet werden.

Beispiel 1:

```
           ADD FELD-A TO FELD-B.

FELD-A vor Ausführung     3 9 2 +                    392 +
                                                     045 +
                                                     437 +

FELD-B vor Ausführung     0 4 5 +
FELD-A nach Ausführung    3 9 2 +   unverändert
FELD-B nach Ausführung    4 3 7 +   mit Additionsergebnis
```

Beispiel 2:

```
           ADD 5,4 TO FELD-C

Literal 5,4 vor Ausführung      5,4                  5,4
                                                    16,0
                                                    21,4

FELD-C vor Ausführung          16^0
Literal 5,4 nach Ausführung     5,4   unverändert
FELD-C nach Ausführung         21^4   mit Additionsergebnis
```

Beispiel 3:

```
           ADD MENGE-1 MENGE-2 GIVING GESAMT-MENGE.

MENGE-1 vor Ausführung          149 +              149,0 +
                                                    35 7 +
                                                   184,7 +

MENGE-2 vor Ausführung           35^7 +
GESAMT-MENGE vor Ausführung     569^9 +
MENGE-1 nach Ausführung         149 +    unverändert
MENGE-2 nach Ausführung          35^7 +  unverändert
GESAMT-MENGE nach Ausführung    184^7 +  mit Additionsergebnis
```

Beispiel 4:

```
ADD MENGE-1 MENGE-2 GIVING GESAMT-MENGE ROUNDED.
```

MENGE-1 vor Ausführung	1 4 9 +	$149_{\wedge}0$ +
		$35_{\wedge}7$ +
		$184_{\wedge}7$ +
MENGE-2 vor Ausführung	3 $5_{\wedge}7$ +	
GESAMT-MENGE vor Ausführung	8 3 2 +	
MENGE-1 nach Ausführung	1 4 9 +	unverändert
MENGE-2 nach Ausführung	3 $5_{\wedge}7$ +	unverändert
GESAMT-MENGE nach Ausführung	1 8 5 +	mit aufgerundetem Additionsergebnis

Beispiel 5:

```
ADD -58,2 TO SUMME.
```

Literal-58,2 vor Ausführung	5 8,2 -	00152,36 +
		58,20 -
		00094,16 +
SUMME vor Ausführung	0 0 1 5 $2_{\wedge}3$ 6 +	
Literal-58,2 nach Ausführung	5 8,2 -	unverändert
SUMME nach Ausführung	0 0 0 9 $4_{\wedge}1$ 6 +	mit Additionsergebnis

Beispiel 6:

```
ADD FELD-D FELD-E GIVING FELD-F ROUNDED ON SIZE ERROR
                                GO TO ABC.
```

FELD-D vor Ausführung	3 5 0	350,0
		1580,3
FELD-E vor Ausführung	1 5 8 $0_{\wedge}3$	1930,3
FELD-F vor Ausführung	9 9 6	
FELD-D nach Ausführung	3 5 0	unverändert
FELD-E nach Ausführung	1 5 8 $0_{\wedge}3$	unverändert
FELD-F nach Ausführung	9 9 6	unverändert, da das abgerundete Ergebnis von 1930 die Kapazität des Ergebnisfeldes (FELD-F) überschreitet

Der Prozedurname ABC wird nach der ADD-Anweisung ausgeführt, weil eine ON SIZE ERROR-Bedingung auftrat.

Beispiel 7:

```
          ADD VERK-1 VERK-2 VERK-3 TO VERK-SUMME.

VERK-1 vor Ausführung          0 1 2^0 0 +          012,00 +
                                                    005,93 +
VERK-2 vor Ausführung          0 0 5^9 3 +          010,05 +
                                                    020,00 +
VERK-3 vor Ausführung          0 1 0^0 5 +          047,98 +
VERK-SUMME vor Ausführung      0 2 0^0 0 +
VERK-1 nach Ausführung         0 1 2^0 0 +  unverändert
VERK-2 nach Ausführung         0 0 5^9 3 +  unverändert
VERK-3 nach Ausführung         0 1 0^0 5 +  unverändert
VERK-SUMME nach Ausführung     0 4 7^9 8 +  mit Additionsergebnis
```

Beispiel 8:

```
          ADD VERK-1 VERK-2 VERK3 GIVING VERK-SUMME.

VERK-1 vor Ausführung          0 1 2^0 0 +               012,00
                                                         005,93
VERK-2 vor Ausführung          0 0 5^9 3 +               010,05
                                                         027,98
VERK-3 vor Ausführung          0 1 0^0 5 +
VERK-SUMME vor Ausführung      0 2 0^0 0 +
VERK-1 nach Ausführung         0 1 2^0 0 +  unverändert
VERK-2 nach Ausführung         0 0 5^9 3 +  unverändert
VERK-3 nach Ausführung         0 1 0^0 5 +  unverändert
VERK-SUMME nach Ausführung     0 2 7^9 8 +  mit Additionsergebnis
```

Beispiel 9:

```
          ADD  100 MENGE  TO  SUMME-1 SUMME-2.

Literal 100 vor Ausführung     100
MENGE       vor Ausführung      75
SUMME-1     vor Ausführung     375
SUMME-2     vor Ausführung    1035

Literal 100 nach Ausführung    100   unverändert
MENGE       nach Ausführung     75   unverändert
SUMME-1     nach Ausführung    550   mit Ergebnis von 100+75+ 375
SUMME-2     nach Ausführung   1210   mit Ergebnis von 100+75+1035
```

Beispiel 10:

```
          ADD 100  MENGE   GIVING   SUMME-3  SUMME-4.

Literal 100 vor Ausführung     100
MENGE       vor Ausführung      75
SUMME-3     vor Ausführung     300
SUMME-4     vor Ausführung     005

Literal 100 nach Ausführung    100   unverändert
MENGE       nach Ausführung     75   unverändert
SUMME-3     nach Ausführung    175   mit Ergebnis von 100+75
SUMME-4     nach Ausführung    175   mit Ergebnis von 100+75
```

DIE ADD CORRESPONDING-ANWEISUNG

Bei der CORRESPONDING-Variante der ADD-Anweisung werden Felder gleichen Namens aus zwei Gruppen summiert und in den entsprechenden Feldern der Empfangsgruppe gespeichert.

Format:

```
ADD [CORRESPONDING | CORR] Identifier-1 TO Identifier-2 [ROUNDED]
    [ON SIZE ERROR unbedingte Anweisung 1]
    [NOT ON SIZE ERROR unbedingte Anweisung 2]
    [END-ADD]
```

Folgende Bedingungen müssen erfüllt sein, damit Datenfelder bei einer ADD CORRESPONDING-Anweisung summiert werden.

- FILLERs in Identifier-1 und Identifier-2 gelten nicht als korrespondierende Namen.
- Ein Datenfeld in Identifier-1 und Identifier-2 hat einen identischen Datennamen und identische Qualifikatoren mit Ausnahme des Identifier-1 und Identifier-2.
- Beide Datenfelder müssen Elementarfelder sein.
- Felder innerhalb beider Gruppen, die eine RENAMES-, REDEFINES-, OCCURS- oder USAGE IS INDEX-Klausel enthalten, werden ignoriert.

Der Feldinhalt der Elementarfelder in Identifier-1 wird zum Inhalt der namentlich korrespondierenden Datenfelder in Identifier-2 addiert. Das Ergebnis wird in den korrespondierenden Datenfeldern in Identifier-2 gespeichert. Identifier-1 und Identifier-2 müssen eine Feldgruppe bezeichnen. Identifier-1 und Identifier-2 dürfen nicht auf Stufennummer 66, 77 und 88 definiert sein. Identifier-1 und Identifier-2 dürfen eine REDEFINES oder OCCURS-Klausel enthalten, sie können auch Unterstufen eines Datennamens sein, der eine OCCURS oder REDEFINES-Klausel enthält.

Die Definitionen der korrespondierenden zu addierenden Felder brauchen nicht identisch (gepackt oder ungepackt usw.) sein. Jede Konvertierung von Datentypen und Stellenangleichung erfolgt durch den COBOL-Compiler analog zur einfachen ADD-Anweisung. Die Operanden der ADD CORRESPONDING-Anweisung können von einem der folgenden Datentypen sein:

Datentyp	maximale Länge
Dezimal ohne Vorzeichen	18 Bytes
Ungepackt dezimal mit Vorzeichen links oder rechts angefügt	19 Bytes einschließlich 8-Bit-Vorzeichen
Ungepackt dezimal mit Zonenvorzeichen links oder rechts	18 Bytes einschließlich Zonen-Vorzeichen
Gepackt dezimal ohne Vorzeichen	9 Bytes
Gepackt dezimal mit Vorzeichen	10 Bytes einschließlich 4-Bit-Vorzeichen
Binär ohne Vorzeichen	8 Bytes

In einer ADD CORRESPONDING-Anweisung werden die Längen der Ergebnisse für jedes Paar korrespondierender Datenfelder getrennt ermittelt. Diese Länge ist eine hypothetische Größe, die sich aus den angeglichenen Stellenzahlen der korrespondierenden Felder ergibt. Die zusammengesetzte Länge darf 18 Stellen nicht überschreiten.

Die ADD CORRESPONDING-Anweisung ist eine unbedingte Anweisung, soweit sie keine SIZE ERROR-Angabe enthält.

ROUNDED

Weist das Resultat der Addition mehr Dezimalstellen (Positionen rechts vom Dezimalpunkt) als das Ergebnisfeld auf, dann werden beide nach dem Dezimalpunkt ausgerichtet. Ohne die Angabe ROUNDED erfolgt ein Abschneiden. Bei Vorhandensein von ROUNDED erhöht sich der absolute Wert des Ergebnisfeldes immer dann um 1, wenn die erste nicht mehr aufzunehmende Ziffer größer als 4 ist.

SIZE ERROR

Nach Dezimalpunktausrichtung und Rundung wird bei Verwendung von SIZE ERROR eine Prüfung auf eine SIZE ERROR-Bedingung hin vorgenommen. Die SIZE ERROR-Bedingung liegt vor, wenn der Wert des Ergebnisses den höchsten vom Ergebnisfeld aufnehmbaren Wert überschreitet. Lediglich die linke Seite des Empfangsfeldes wird auf Überlauf hin geprüft. Bei Auftreten der SIZE ERROR-Bedingung wird der Wert des Ergebnisses nicht im Ergebnisfeld gespeichert. Bei mehreren Empfangsfeldern wird in allen das Ergebnis gespeichert, die keine SIZE ERROR-Bedingung verursachen. Wird in einem der Empfangsfelder eine SIZE ERROR-Bedingung festgestellt, wird die ADD CORRESPONDING-Anweisung normal zu Ende geführt und danach die unbedingte Anweisung ausgeführt. Die Schreibweise "unbedingte Anweisung" in der SIZE ERROR-Angabe bezieht sich auf eine oder mehrere aufeinanderfolgende unbedingte Anweisungen, deren Folge durch einen Punkt beendet wird.

Ohne Vorliegen einer solchen Bedingung wird das Ergebnis der Addition in den Ergebnisfeldern gespeichert und die nächste Anweisung ausgeführt. Die unbedingte Anweisung in SIZE ERROR wird nicht ausgeführt.

Ohne spezifizierte SIZE ERROR-Angabe ist bei Auftreten einer SIZE ERROR-Bedingung der Inhalt des Ergebnisfeldes unvorhersagbar. Die auf die ADD CORRESPONDING-Anweisung folgende Anweisung wird ausgeführt und der Fehler nicht festgestellt.

Beispiel:

```
ADD CORRESPONDING SUMMEN TO GESAMTSUMMEN.
```

```
01 LAGERUMSATZ
   02 LAGER-NR
   02 SUMMEN
      03 BETRAG1
      03 BETRAG2
      03 BETRAG3
      03 BETRAG4
      03 BETRAG5
```

```
01 TOTALSATZ
   02 LAGER-NR
   02 GESAMTSUMMEN
      03 TOTALSUMME
      03 BETRAG4
      03 BETRAG3
      03 BETRAG2
      03 BETRAG1
```

Das Ergebnis der ADD CORRESPONDING-Anweisung entspricht dem Ergebnis folgender Einzelanweisungen:

```
ADD BETRAG1 IN SUMMEN TO BETRAG1 IN GESAMTSUMMEN.
ADD BETRAG2 IN SUMMEN TO BETRAG2 IN GESAMTSUMMEN.
ADD BETRAG3 IN SUMMEN TO BETRAG3 IN GESAMTSUMMEN.
ADD BETRAG4 IN SUMMEN TO BETRAG4 IN GESAMTSUMMEN.
```

Das Feld BETRAG5 wird nicht addiert, da kein korrespondierendes Datenfeld vorhanden ist.

DIE ALTER-ANWEISUNG

Die ALTER-Anweisung verändert eine zuvor durch Prozedur-anweisungen festgelegte Folge von Operationen.

Format:

```
ALTER  Prozedur-Name-1  TO [PROCEED TO]  Prozedur-Name-2
     [Prozedur-Name-3  TO [PROCEED TO]  Prozedur-Name-4] ...
```

Während des Programmablaufes verändert die ALTER-Anweisung die GO TO-Anweisungen in den mit Prozedur-Name-1, Prozedur-Name-3 usw. bezeichneten Paragraphen und ersetzt den jeweiligen dort angegebenen Prozedurnamen durch den Prozedur-Namen-2, Prozedur-Namen-4 usw.

Jeder Prozedur-Name-1, Prozedur-Name-3 usw. ist der Name eines Paragraphen, der nur einen Satz enthält, welcher aus einer GO TO-Anweisung ohne DEPENDING-Angabe besteht.

Jeder Prozedur-Name-2, Prozedur-Name-4 usw. ist der Name eines Paragraphen oder Kapitels (Section) in der PROCEDURE DIVISION.

Auf eine GO TO-Anweisung in einem Kapitel, dessen Segment-Nummer größer oder gleich 50 ist, darf nicht durch eine ALTER-Anweisung in einem Kapitel mit einer anderen Segment-Nummer Bezug genommen werden.

Beispiel:

```
       AENDERN.     GO TO VORBEREITUNG.
```

Der GO TO-Satz gemäß dem vorangehenden Beispiel ist nach seiner ersten Ausführung so abzuändern, daß er bei seiner zweiten Ausführung einen Sprung zu der Prozedur ABSCHLUSS bewirkt.

```
                    ALTER AENDERN TO PROCEED TO ABSCHLUSS.
```

Die vorangehende ALTER-Anweisung verändert die GO TO-Anweisung. Somit wird nach Ausführung dieser ALTER-Anweisung die GO TO-Anweisung bei AENDERN folgende:

```
       AENDERN.     GO TO ABSCHLUSS.
```

DIE CALL-ANWEISUNG

und

DIE CANCEL-ANWEISUNG

Siehe hierzu das separate Kapitel "UNABHÄNGIG COMPILIERTE MODULE" (INTER PROGRAM COMMUNICATION).

DIE CLOSE-ANWEISUNG

Die CLOSE-Anweisung beendet die Verarbeitung einer oder mehrerer Dateien, und zwar inklusive wahlweises Rückspulen bei Bändern bzw. Freigabe. Diese Freigabe kann auch dazu verwendet werden, um die Verarbeitung von einzelnen Spulen mit wahlweisem Rückspulen abzuschließen.

Format 1 (Sequentielle Dateien):

```
CLOSE { Datei-Name-1 [ REEL ]                [ FOR REMOVAL ] } ...
                     [ UNIT ]
                       WITH   { NO REWIND }
                              { LOCK      }
```

Format 2 (Relative und Indexdateien):

```
CLOSE { Datei-Name-1 [ WITH LOCK ] } ...
```

Die Dateien, auf die in der CLOSE-Anweisung Bezug genommen wird, brauchen nicht dieselbe Struktur und Zugriffstechnik aufzuweisen.

Eine CLOSE-Anweisung läßt sich nur für eine vorher mit OPEN eröffnete Datei ausführen. Die erfolgreiche Ausführung einer CLOSE-Anweisung beendet die Verarbeitung der durch den Datei-Namen bezeichneten Datei. Jede Datei, auf die in einem Programm Zugriff genommen wird, muß durch eine CLOSE-Anweisung vor Ende des Programmes abgeschlossen werden. Eine Datei kann überall im Ursprungsprogramm abgeschlossen werden.

Erstreckt sich eine sequentielle Datei über mehrere Magnetbandspulen, kann die Anweisung CLOSE Datei-Name REEL/UNIT verwendet werden. Diese Anweisung schließt die laufende Spule in der Datei ab. Sie eröffnet dann die nächste Spule, so daß der nächste Datensatz, auf den zugegriffen wird, der erste dieser Spule ist. In COBOL sind die Wörter REEL und UNIT gleichbedeutend.

Enthält die CLOSE-Anweisung WITH NO REWIND, wird das Band nicht zurückgespult.

Die WITH LOCK-Angabe stellt sicher, daß die Datei während der Ausführung dieser Ablaufeinheit nicht mehr eröffnet werden kann. Im Betriebssystem bewirkt WITH LOCK, daß die Zuordnung der Datei aufgehoben wird.

Nach der Ausführung einer CLOSE-Anweisung ohne REEL/UNIT-Angabe ist der dem Datei-Namen zugeordnete Satzbereich nicht mehr verfügbar.

Ist die FILE STATUS-Klausel in dem FILE CONTROL-Eintrag (SELECT) für eine Datei spezifiziert, dann wird dieser Datei ein FILE STATUS-Datenfeld zugeordnet. Nach Ausführung der CLOSE-Anweisung wird in das FILE STATUS-Datenfeld für jede Datei, welcher ein solches zugeordnet ist, der Wert 00 übertragen. Alle auf eine Datei anwendbaren DECLARATIVE-Prozeduren werden ausgeführt, nachdem das FILE STATUS-Datenfeld seinen Status der für diese Datei ausgeführten CLOSE-Anweisung empfängt. Die FILE STATUS-Werte stehen in der Tabelle im Anhang.

Ist bei der Ausführung einer STOP RUN-Anweisung eine Datei noch nicht abgeschlossen, wird eine File-Management-Warnung ausgegeben und die Datei automatisch geschlossen.

Eine CANCEL-Anweisung schließt alle noch offenen Dateien des zu cancelnden Programmes.

Bei vorhandener OPTIONAL-Klausel im FILE CONTROL Paragraphen wird die Standard-Dateiendeprozedur für eine nicht vorhandene Datei natürlich nicht ausgeführt.

DIE COMPUTE-ANWEISUNG

Die COMPUTE-Anweisung weist einem Datenfeld den Wert eines arithmetischen oder booleschen Ausdrucks zu.

Format 1:

COMPUTE Identifier-1 [ROUNDED] ,Identifier-2 [ROUNDED]] ...
= arithmetischer Ausdruck [ON SIZE ERROR unbedingte Anweisung]
[NOT ON SIZE ERROR unbedingte Anweisung]
[END-COMPUTE]

Unter einen arithmetischen Ausdruck fallen:

- Ein Identifier eines numerischen Datenfeldes
- Ein numerisches Literal
- Durch binäre arithmetische Operatoren getrennte Identifier numerischer Datenfelder und numerische Literale
- Zwei durch einen binären arithmetischen Operator wie +, -, *, / und ** getrennte arithmetische Ausdrücke
- Ein arithmetischer Ausdruck in runden Klammern

Einem arithmetischen Ausdruck kann ein monadischer arithmetischer Operator, wie + und -, vorangehen.

In der COMPUTE-Anweisung links vom Gleichhzeitszeichen erscheinende Identifier müssen sich entweder auf ein numerisches Datenfeld oder ein numerisch editiertes Datenfeld beziehen. Identifier-1, Identifier-2, usw. sind die Ergebnisfelder in der COMPUTE-Anweisung.

Ist der arithmetische Ausdruck in einer COMPUTE-Anweisung entweder ein Identifier oder ein numerisches Literal, so wird es durch die COMPUTE-Anweisung ermöglicht, die Werte von Identifier-1, Identifier-2, usw., dem des einzelnen Identifiers oder Literals gleichzusetzen.

Ist der arithmetische Ausdruck in einer COMPUTE-Anweisung eine Kombination von Identifiern, numerischen Literalen und arithmetischen Operatoren in runden Klammern, so kann durch diese COMPUTE-Anweisung der Wert einer Kombination von arithmetischen Operationen festgestellt werden. Die COMPUTE-Anweisung gestattet es dem Programmierer, arithmetische Operationen zu kombinieren, und dies ohne die Einschränkungen auf die Zusammensetzung von Operanden und/oder Empfangsfeldern, wie sie durch die arithmetischen Anweisungen ADD, DIVIDE, MULTIPLY und SUBTRACT bestehen. Die nachfolgende Tabelle ist eine Zusammenfassung der fünf Mehroperanden und der zwei monadischen Operatoren, die zur Darstellung eines arithmetischen Ausdruckes benutzt werden können. Allen arithmetischen Operatoren muß ein Leerzeichen vorausgehen und auch nachfolgen.

Arithmetischer Operator		Bedeutung	COBOL-Beispiel	Arithmetisches Beispiel
Mehr-operanden	+	Addition	AMT1+AMT2	2+4=6
	-	Subtraktion	BAL-AMT	7-5=2
	*	Multiplikation	AMT*4	16x4=64
	/	Division	BAL/4	32/4=8
	**	Potenzierung	RATE**3	5x5x5=125
Monadisch	+	Multiplikationseffekt mit +1	+QUAN	+1x+14=14
	-	Multiplikationseffekt mit -1	-QUAN	-1x+16=-16

Die Bewertung der Potenzierung in einem arithmetischen Ausdruck geschieht mit folgenden Einschränkungen:

- Ergibt die Bewertung sowohl eine positive als auch eine negative Realzahl, so gilt die positive Zahl als Ergebniswert.
- Ist keine Realzahl als Bewertungsergebnis vorhanden, so besteht die SIZE ERROR-Bedingung.

- **Ist der auf eine Potenz zu erhebende Wert eines Ausdruckes Null, dann muß der Exponent einen Wert größer als Null haben; ansonsten ist die SIZE ERROR-Bedingung gegeben.**

Runde Klammern können in arithmetischen Ausdrücken verwendet werden, um die Reihenfolge, in der Elemente zu bewerten sind, anzuzeigen. Es müssen gleich viele linke wie rechte runde Klammern in einem arithmetischen Ausdruck vorhanden sein. Ausdrücke in runden Klammern werden zuerst bewertet. In einer Schachtelung von runden Klammern wird die Bewertung vom kürzesten in Klammern befindlichen Satz zum längsten solchen Satz fortschreitend durchgeführt. Sind keine runden Klammern vorgesehen oder sind die Ausdrücke in runden Klammern von gleicher Länge, dann werden die Sätze nach folgender hierarchischen Reihenfolge bewertet:

1. Monadisches Plus und Monadisches Minus

2. Potenzierung

3. Multiplikation und Division

4. Addition und Subtraktion

Wenn die Reihenfolge der Ausführung der Bewertung nicht durch runde Klammern festgelegt ist und die Hierarchie der arithmetischen Operationen dieselbe ist, dann werden die Operationen innerhalb des arithmetischen Ausdruckes von links nach rechts fortschreitend durchgeführt.

Ein arithmetischer Ausdruck beginnt entweder mit einer linken runden Klammer, einem Pluszeichen, einem Minuszeichen, einem Identifier oder einem Literal. Ein arithmetischer Ausdruck endet entweder mit einer rechten runden Klammer, einem Identifier oder einem Literal.

Arithmetische Operatoren, Identifier, Literale und runde Klammern können in einem der in der nachfolgenden Tabelle zusammengefaßten arithmetischen Ausdrücke kombiniert sein. Ein "Ja" zeigt an, daß das erste Symbol vom zweiten Symbol gefolgt werden darf. Ein "Nein" zeigt an, daß das erste Symbol nicht vom zweiten Symbol gefolgt werden darf.

Erstes Symbol	Zweites Symbol				
	Identifier oder Literal	Mehrfach vorkommender	Monadischer Operator	(	)
Identifier oder Literal	Nein	Ja	Nein	Nein	Ja
Mehrfach vorkommender Operator	Ja	Nein	Ja	Ja	Nein
Monadischer Operator	Ja	Nein	Nein	Ja	Nein
Linke runde Klammer	Ja	Nein	Ja	Ja	Nein
Rechte runde Klammer	Nein	Ja	Nein	Nein	Ja

Ist mehr als ein Identifier links vom Gleichheitszeichen angegeben, so wird der Wert des arithmetischen Ausdruckes in der COMPUTE-Anweisung zuerst errechnet. Daraufhin wird der errechnete Wert als der neue Wert eines jeden Identifiers links vom Gleichheitszeichen gespeichert.

Die Datenbeschreibungen der Identifier und Literale in dem arithmetischen Ausdruck einer COMPUTE-Anweisung brauchen nicht dieselben zu sein, denn der Compiler sorgt für jede notwendige Umwandlung und Dezimalpunktausrichtung während des Rechnens. Bei NCR-ITX-COBOL können die Operanden der COMPUTE-Anweisung von einem der folgenden Datentypen sein:

Datentyp	maximale Länge
Dezimal ohne Vorzeichen	18 Bytes
Ungepackt dezimal mit Vorzeichen rechts angefügt	19 Bytes einschließlich 8-Bit-Zeichen
Ungepackt dezimal mit Vorzeichen links angefügt	19 Bytes einschließlich 8-Bit-Zeichen
Ungepackt dezimal mit Zonenvorzeichen rechts	18 Bytes einschließlich Zonen-Zeichen
Ungepackt dezimal mit Zonenvorzeichen links	18 Bytes einschließlich Zonen-Zeichen
Gepackt dezimal ohne Vorzeichen	9 Bytes
Gepackt dezimal mit Vorzeichen	10 Bytes einschließlich 4-Bit-Zeichen
Binär mit Vorzeichen	8 Bytes einschließlich Vorzeichen-Bit

COMPUTE ist eine unbedingte Anweisung; es wird jedoch zu einer bedingten Anweisung, wenn die SIZE ERROR-Angabe vorliegt.

ROUNDED

Weist das Resultat mehr Dezimalstellen (Positionen rechts vom Dezimalpunkt) als das Ergebnisfeld auf, dann werden beide nach dem Dezimalpunkt ausgerichtet. Ohne die Angabe ROUNDED erfolgt ein Abschneiden. Bei Vorhandensein von ROUNDED erhöht sich der absolute Wert des entsprechenden Ergebnisfeldes immer dann um 1, wenn die abzuschneidende Stelle größer als 4 ist. Werden die Ganzzahlpositionen in dem Ergebnisfeld durch das Zeichen P in der PICTURE-Zeichenfolge für dieses Ergebnisfeld dargestellt, dann erfolgt ein Runden oder Abschneiden relativ zur äußersten rechten Ganzzahlposition, der ein Speicherplatz zugeordnet ist.

SIZE ERROR

Nach Dezimalpunktausrichtung und Rundung wird, wenn SIZE ERROR spezifiziert ist, eine Prüfung auf eine SIZE ERROR-Bedingung hin vorgenommen. Die SIZE ERROR-Bedingung liegt vor, wenn der Wert des Ergebnisses den höchsten vom Ergebnisfeld aufnehmbaren Wert überschreitet. Lediglich die linke Seite des Empfangsdatenfeldes wird auf Überlauf hin geprüft. Bei Auftreten der SIZE ERROR-Bedingung wird der Wert des Ergebnisses nicht im Ergebnisfeld gespeichert. Bei einer COMPUTE-Anweisung mit mehr als einem Ergebnisfeld erhalten alle außer den eine SIZE ERROR-Bedingung aufweisenden Ergebnisfeldern einen Wert. Liegt bei einem oder mehreren Ergebnisfeldern eine SIZE ERROR-Bedingung vor, so erfolgt die Ausführung der unbedingten Anweisung in der SIZE ERROR-Angabe, nachdem die COMPUTE-Anweisung vollständig ausgeführt ist. Die Schreibweise "unbedingte Anweisung" in der SIZE ERROR-Angabe bezieht sich auf eine oder mehrere aufeinanderfolgende unbedingte Anweisungen, deren Folge durch einen Punkt beendet wird.

Bei SIZE ERROR, jedoch ohne Vorliegen einer solchen Bedingung bei einem der Ergebnisfelder, wird das Rechnerergebnis in jedem Ergebnisfeld gespeichert und die nächste Anweisung ausgeführt. Eine unbedingte Anweisung in SIZE ERROR wird nicht ausgeführt.

Ohne spezifizierte SIZE ERROR-Angabe ist bei Auftreten einer SIZE ERROR-Bedingung bei einem oder mehreren der Ergebnisfelder der Wert des betroffenen Ergebnisfeldes unvorhersagbar. Die auf die COMPUTE-Anweisung folgende Anweisung wird ausgeführt und der Fehler wird nicht festgestellt.

Beispiel 1:

```
COMPUTE JAHRE = MONATE / 12.
```

	vor Ausführung	nach Ausführung	
MONATE	1003	1003	unverändert
JAHRE	019	083	abgeschnitten

$$\text{JAHRE} = \frac{\text{MONATE}}{12} = \frac{1003}{12} = 083$$

Beispiel 2:

```
COMPUTE JAHRE ROUNDED = MONATE / 12
```

	vor Ausführung	nach Ausführung	
MONATE	1003	1003	unverändert
JAHRE	019	084	aufgerundet

$$JAHRE = \frac{MONATE}{12} = \frac{1003}{12} = 084$$

Beispiel 3:

```
COMPUTE ENDSUMME = ZEITSPANNE * (1 + RATE)
```

Dies entspricht ENDSUMME = ZEITSPANNE x (1 + RATE)

	vor Ausführung	nach Ausführung
RATE	5	5
ZEITSPANNE	10	10
ENDSUMME	4297	0060

ENDSUMME = 10 x (1 + 5) = 10 x 6 = 60

Beispiel 4:

```
COMPUTE X-ABSTAND = MAGNET / (LAENGE - 2).
```

Dies entspricht x - Abstand = $\frac{\text{MAGNET}}{\text{LAENGE-2}}$

	vor Ausführung	nach Ausführung	
LAENGE	7	7	
MAGNET	14	14	
x-ABSTAND	29	02	abgeschnitten

$$\text{x-ABSTAND} = \frac{\text{MAGNET}}{\text{LAENGE-2}} = \frac{14}{7-2} = \frac{14}{5} = 2$$

Beispiel 5:

```
COMPUTE X-ABSTAND = MAGNET / LAENGE - 2.
```

Dies entspricht X-ABSTAND = $\frac{\text{MAGNET}}{\text{LAENGE}} - 2$

	vor Ausführung	nach Ausführung
MAGNET	14	14
LAENGE	7	7
x-ABSTAND	29	00

$$\text{x-ABSTAND} = \frac{\text{MAGNET}}{\text{LAENGE}} - 2 = \frac{14}{7} - 2 = 2 - 2 = 0$$

Beispiel 6:

```
COMPUTE NEUE-SUM = (- 4 + SUM1 - SUM2) / 2.
```

Dies entspricht $NEUE\text{-}SUM = \frac{(-1 \times 4) + SUM1 - SUM2}{2}$

	vor Ausführung	nach Ausführung
SUM1	100	100
SUM2	20	20
NEUE-SUM	375	38

$$NEUE\text{-}SUM = \frac{(-1 \times 4) + 100 - 20}{2} = \frac{-4 + 80}{2} = \frac{76}{2} = 38$$

Beispiel 7:

```
COMPUTE SUM1, SUM2 ROUNDED = 2.7 * (WERT1 + WERT2 + WERT3).
```

Dies entspricht SUM1 = 2.7 x (WERT1 + WERT2 + WERT3)
und SUM2 = 2.7 x (WERT1 + WERT2 + WERT3)

	vor Ausführung	nach Ausführung	
WERT1	10	10	
WERT2	21	21	
WERT3	67	67	
SUM1	402	264	abgeschnitten
SUM2	921	265	aufgerundet

SUM1 = 2.7 x (10+21+67) = 2.7 x 98 = 264.6 oder 264 abgeschnitten
SUM2 = 2.7 x (10+21+67) = 2.7 x 98 = 264.6 oder 265 aufgerundet

Beispiel 8:

```
COMPUTE NEU-ANZ = (ANZ + 10) ** 3.
                                    3
Dies entspricht NEU-ANZ = (ANZ + 10)

                vor                 nach
                Ausführung          Ausführung

ANZ               04                  04
NEU-ANZ         0000                2924
                   3     3
NEU-ANZ = (4 + 10)   = 14   = 2924
```

Format 2:

COMPUTE Identifier = boolescher Ausdruck

Unter einen booleschen Ausdruck fallen:

1. Identifiers von booleschen Datenfeldern und booleschen Literalen, die mit booleschen Operatoren verbunden sind.

2. Zwei boolesche Ausdrücke, die durch einen booleschen Operator, wie AND, OR und EXOR, verbunden sind.

3. Einzelne boolesche Ausdrücke in Klammern.

Der in der COMPUTE-Anweisung links vom Gleichheitszeichen erscheinende Identifier muß sich auf ein boolesches Datenfeld beziehen. Dieser Identifier ist das Ergebnisfeld in der COMPUTE-Anweisung.

Durch die COMPUTE-Anweisung wird es ermöglicht, den Wert eines Identifiers dem eines booleschen Ausdruckes gleichzusetzen. Es gibt vier boolesche Operatoren, die zum Schreiben eines Ausdruckes verwendet werden können. Vor und nach einem Operator muß eine Leerstelle stehen. Die folgende Tabelle zeigt die booleschen Operatoren in den verschiedenen möglichen Kombinationen.

Boolescher Operator	Bedeutung	Erstes Bit	Zweites Bit	Ergebniswert
AND	Boolesche Konjunktion	0	0	0
		0	1	0
		1	0	0
		1	1	1
OR	Boolesch, mit inklusivem ODER	0	0	0
		0	1	1
		1	0	1
		1	1	1
EXOR	Boolesch, mit exklusivem ODER	0	0	0
		0	1	1
		1	0	1
		1	1	0
Monadisch NOT	Boolesche Negation	0	-	1
		0	1	-

Runde Klammern können in booleschen Ausdrücken verwendet werden, um die Reihenfolge, in der Elemente zu bewerten sind, anzuzeigen. Es müssen gleich viele linke wie rechte runde Klammern in einem booleschen Ausdruck vorhanden sein. Ausdrücke in runden Klammern werden zuerst bewertet. In einer Schachtelung von runden Klammern wird die Bewertung vom kürzesten in Klammern befindlichen Satz zum längsten solchen Satz fortschreitend durchgeführt. Sind keine runde Klammern vorgesehen oder sind die Ausdrücke in runden Klammern von gleicher Länge, dann werden die Sätze nach folgender hierarchischen Reihenfolge bewertet:

1. Negation (NOT)

2. Boolesche Konjunktion mit dem booleschen Operator AND

3. Boolesche Disjunktion mit dem booleschen Operator OR oder dem booleschen Operator EXOR

Ein boolescher Ausdruck beginnt entweder mit einer linken runden Klammer, einem Identifier eines booleschen Datenfeldes oder mit einem booleschen Literal. Ein boolescher Ausdruck endet entweder mit einer rechten runden Klammer, einem Identifier eines booleschen Datenfeldes oder mit einem booleschen Literal.

Boolesche Operatoren, Identifiers, boolesche Literale und runde Klammern können in einem der in der nachfolgenden Tabelle zusammengefaßten booleschen Ausdrücke kombiniert sein. Ein "Ja" zeigt an, daß dem ersten Symbol das zweite Symbol folgen darf. Ein "Nein" zeigt an, daß dem ersten Symbol nicht das zweite Symbol folgen darf.

	Zweites Symbol				
Erstes Symbol	Identifier oder Literal	AND, OR, EXOR	linke Klammer (	rechte Klammer)	NOT
Identifier oder Literal	Nein	Ja	Nein	Ja	Nein
AND, OR, EXOR	Ja	Nein	Ja	Nein	Ja
linke Klammer (	Ja	Nein	Ja	Nein	Ja
rechte Klammer)	Nein	Ja	Nein	Ja	Nein
NOT	Ja	Nein	Ja	Nein	Nein

Wenn die Operanden einer Konjunktion oder Disjunktion (mit inklusivem oder exklusivem ODER) gleicher Länge sind, erfolgt die Operation bei gleichartigen oder verschiedenen booleschen Zeichen in entsprechenden booleschen Zeichenpositionen (Bits) beginnend vom linken Ende und sich zum rechten Ende fortsetzend. Wenn die Operanden von ungleicher Länge sind, dann erfolgt die Operation so, als ob der kurze Operand an seinem rechten Ende durch eine ausreichende Anzahl boolescher Nullen auf die Länge des längeren Operanden ergänzt wäre und so die Operanden gleich lang wären.

Beispiel 1:

```
COMPUTE B1CODE = BCODE AND B"00110011".
```

	vor Ausführung	nach Ausführung
BCODE	00001111	00001111
boolesches Literal	00110011	00110011
B1CODE	00000000	00000011

Beispiel 2:

```
COMPUTE B2CODE = BCODE OR B"00110011".
```

	vor Ausführung	nach Ausführung
BCODE	00001111	00001111
boolesches Literal	00110011	00110011
B2CODE	11111111	00111111

Beispiel 3:

```
COMPUTE B3CODE = BCODE EXOR B"00110011".
```

	vor Ausführung	nach Ausführung
BCODE	00001111	00001111
boolesches Literal	00110011	00110011
B3CODE	10101010	00111100

Beispiel 4:

```
COMPUTE B4CODE = B5CODE OR (B6CODE AND B7CODE).
```

	vor Ausführung	nach Ausführung
B6CODE	00001111	00001111
B7CODE	00110011	00110011
B6CODE und B7CODE		00000011
B5CODE	10101010	10101010
B4CODE	00000000	10101011

DIE CONTINUE-ANWEISUNG

Format:

CONTINUE

Diese Anweisung ist eine "No Operation"-Anweisung. Sie hat keine Auswirkung auf die Ausführung des Programmes.

DIE COPY-ANWEISUNG

Sie dient während einer Compilation zur Übernahme von Programmzeilen aus einer COBOL-Bibliothek (Library) oder einer individuellen Datei in das COBOL-Ursprungsprogramm.

Format 1:

```
COPY   {Text-Name-1}   [[{OF}  {Library-Name-1}]]
       {Literal-1  }   [{IN}  {Literal-2   }]

[REPLACING   {{==Pseudo-Text-1==}   BY   {==Pseudo-Text-2==}}   ...]
             {{Identifier-1     }        {Identifier-2     }}
             {{Literal-3        }        {Literal-4        }}
             {{Wort-1           }        {Wort-2           }}
```

Format 2:

```
COPY Test-Name-1   [Zeilen-Nr.-1   [[{THROUGH}   {Zeilen-Nr.-2}]]
                                    {THRU   }   {END         }]
```

Syntax:

- Die Begrenzung für Pseudo-Text sind zwei Gleichheitszeichen (==).
- Steht während der Compilation mehr als eine COBOL-Bibliothek zur Verfügung, muß der Text-Name-1 mit dem Library-Namen derjenigen Bibliothek qualifiziert werden, die den mit Text-Name-1 bezeichneten Text enthält.
- Pseudo-Text-2 kann Null sein.
- Längere Zeichenfolgen innerhalb Pseudo-Text-1 und Pseudo-Text-2 können über Fortsetzungszeilen gehen, allerdings müssen beide Zeichen eines Pseudo-Text-Delimiters innerhalb einer Zeile beieinander stehen.

- Wort-1 und Wort-2 können jedes einzelne COBOL-Wort sein.

- Eine COPY-Anweisung kann im COBOL-Programm überall stehen, wo eine COBOL-Zeichenfolge oder ein Separator stehen kann, ausgenommen in einer COPY-Anweisung selbst.

- Literal-1 und Literal-2 müssen nicht-numerische Literale sein.

- Im zweiten Format bedeutet Zeilen-Nr.-1 die erste zu kopierende Zeile. Wird sie im zu kopierenden Text nicht gefunden, startet der Kopiervorgang bei der ersten höheren Zeilen-Nr.

- Im zweiten Format bedeutet Zeilen-Nr.-2 die letzte zu kopierende Zeile. END bedeutet kopieren bis zum Ende des Textes. Bei Weglassung sowohl einer Zeilen-Nr.-2 als auch von END wird nur eine Zeile kopiert.

Wirkungsweise des Befehls:

- Der COPY-Befehl selbst wird durch den hereinkopierten Text ersetzt.

- Fehlt die REPLACING-Klausel, wird der Text unverändert übernommen.

- Bei vorhandener REPLACING-Klausel wird der Library-Text kopiert, wobei jede auftretende Zeichenfolge, die mit Pseudo-Text-1, Identifier-1, Literal-1 oder Wort-1 übereinstimmt, im Library-Text durch den entsprechenden Pseudo-Text-2, Identifier-2, Literal-4 oder Wort-2 ersetzt wird.

DIE DELETE-ANWEISUNG (relative Datei - sequentieller Zugriff)

Bei sequentiellem Zugriff löscht die DELETE-Anweisung einen Datensatz von einer auf Magnetplatte befindlichen relativen Datei.

Format:

DELETE Datei-Name RECORD

Der Datei-Name muß einem der in einer Dateibeschreibung in einem FILE-CONTROL-Eintrag auftretenden Datei-Name entsprechen. Der FILE-CONTROL-Eintrag für den Datei-Namen muß die Klausel ORGANIZATION IS RELATIVE enthalten. Der FILE-CONTROL-Eintrag muß indirekt oder ausdrücklich die Klausel ACCESS MODE IS SEQUENTIAL aufweisen.

Ehe die DELETE-Anweisung ausgeführt wird, muß die relative Datei mit einer OPEN I-O Anweisung eröffnet worden sein.

Wenn auf die relative Datei sequentiell zugegriffen wird, muß die Ein-Ausgabe-Anweisung vor der Ausführung der DELETE-Anweisung eine korrekt ausgeführte READ-Anweisung gewesen sein. Bei Ausführung der DELETE-Anweisung wird der Datensatz, auf den durch diese READ-Anweisung zugegrifen wurde, von der Datei gelöscht.

Die Ausführung der DELETE-Anweisung hat keinen Einfluß auf den Inhalt des Satz-Bereiches.

Nach erfolgreicher Ausführung der DELETE-Anweisung ist der Datensatz logisch aus der Datei gelöscht und es kann darauf nicht mehr zugegriffen werden.

Beispiel:

```
002010 FILE-CONTROL.
002020     SELECT DATEI-A ASSIGN TO DISC
002030                    ORGANIZATION IS RELATIVE
002040                    ACCESS MODE IS SEQUENTIAL.

003010 DATA DIVISION.
003020 FILE SECTION.
003030 FD DATEI-A BLOCK CONTAINS 51 RECORDS
003040            RECORD CONTAINS 10 CHARACTERS
003050            LABEL RECORD IS STANDARD.
003070 01  SATZ-A     PIC X(10).

005010 PROCEDURE DIVISION.
005020 BEGINN.
005030     OPEN I-O DATEI-A.

005100     READ DATEI-A AT END GO TO ENDE.

005200     DELETE DATEI-A.

006130 ENDE.
006140     CLOSE DATEI-A.
```

Im obigen Beispiel ist eine auf Magnetplatte gespeicherte Datei, auf die der Zugriff sequentiell erfolgt, relativ aufgebaut. Sie hat den Namen DATEI-A. Damit ein Datensatz von DATEI-A gelöscht werden kann, muß die Datei zuerst mit der OPEN I-O Anweisung auf der Ursprungszeile 005030 im I-O Modus eröffnet werden. Die READ-Anweisung auf der Ursprungszeile 005100 greift sequentiell auf DATEI-A zu, um dadurch in SATZ-A den nächsten Datensatz der DATEI-A verfügbar zu machen. Wenn der Datensatz, auf den durch READ Zugriff genommen wurde, gelöscht werden soll, wird eine DELETE-Anweisung ausgeführt. Die DELETE-Anweisung auf der Ursprungszeile 005200 löscht den vorher durch die READ-Anweisung gelesenen Datensatz. Die CLOSE-Anweisung auf Ursprungszeile 006140 beendet die Bearbeitung der DATEI-A.

DIE DELETE-ANWEISUNG (relative Datei - Direktzugriff)

Bei direktem Zugriff löscht die DELETE-Anweisung einen bestimmten Datensatz von einer auf Magnetplatte gespeicherten relativen Datei.

Format:

DELETE Datei-Name RECORD [INVALID KEY unbedingte Anweisung 1]
[NOT INVALID KEY unbedingte Anweisung 2]
[END-DELETE]

Der in der DELETE-Anweisung auftretende Datei-Name muß in einem FILE-CONTROL-Eintrag auftreten. Der FILE-CONTROL-Eintrag für den Datei-Namen muß die Klausel ORGANIZATION IS RELATIVE und die Klausel ACCESS MODE IS RANDOM oder ACCESS MODE IS DYNAMIC enthalten. Ehe die DELETE-Anweisung ausgeführt wird, muß die Datei mit einer OPEN I-O Anweisung eröffnet worden sein.

Wenn auf eine relative Datei direkt zugegriffen wird, löscht die DELETE-Anweisung den durch das Satzschlüsselfeld identifizierten Datensatz von der Datei. Das Satzschlüsselfeld muß durch die RELATIVE KEY-Angabe in der ACCESS MODE-Klausel bezeichnet werden. Der Programmierer muß die relative Satznummer des zu löschenden Datensatzes vor Ausführung der DELETE-Anweisung in das Satzschlüsselfeld übertragen. Eine relative Datensatznummer 3 bezeichnet den dritten Datensatz in einer relativen Datei; eine relative Datensatznummer 7 bezeichnet den siebenten Datensatz in einer relativen Datei.

Die Ausführung der DELETE-Anweisung hat keinen Einfluß auf den Inhalt des Satz-Bereiches.

Nach erfolgreicher Ausführung der DELETE-Anweisung ist der Datensatz logisch aus der Datei gelöscht und es kann nicht mehr darauf zugegriffen werden.

INVALID KEY

Die INVALID KEY-Angabe muß für eine DELETE-Anweisung spezifiziert werden, die auf eine relative Datei mit direktem oder dynamischem Zugriff Bezug nimmt, die keine zugeordnete USE-Anweisung in den Declaratives enthält.

Wenn eine relative Datei, auf die direkt zugegriffen wird, keinen gültigen Datensatz in der spezifizierten Datensatzposition enthält, denn tritt eine INVALID KEY-Bedingung auf.

Bei einer INVALID KEY-Bedingung ist die DELETE-Anweisung nicht ausgeführt worden und es wird kein Einfluß auf die Datei genommen. Eine INVALID KEY-Bedingung verursacht folgende Vorgänge:

1. Wurde die FILE STATUS-Klausel in dem FILE CONTROL-Eintrag für die relative Datei verwendet, wird ein Wert 23 in das FILE STATUS-Datenfeld übertragen, um anzuzeigen, daß kein Datensatz gefunden wurde.

2. Ist INVALID KEY in der DELETE-Anweisung vorhanden, dann wird die Steuerung auf die unbedingte Anweisung übertragen. Keine der für diese Datei spezifizierten DECLARATIVE-Prozeduren wird ausgeführt. Eine vorhandene NOT INVALID KEY Klausel wird ignoriert.

3. Ist INVALID KEY nicht in der DELETE-Anweisung, dann werden für diese Datei DECLARATIVE-Prozeduren ausgeführt. Eine vorhandene NOT INVALID KEY Klausel wird ignoriert.

Tritt keine INVALID KEY-Bedingung auf, wird die INVALID KEY Klausel ignoriert, falls sie vorhanden ist in der Anweisung. Das Statusfeld wird mit dem Status beschickt und folgende Aktionen werden ausgeführt:

1. Liegt ein Ausnahmestatus vor (nicht 00), der nicht ein INVALID KEY-Status ist, wird in die einschlägige Declarative-Prozedur verzweigt.

2. Ist der Status 00 und die NOT INVALID KEY Klausel wurde in der Anweisung nicht verwendet, wird an das Ende der Anweisung gesprungen.

3. Ist der Status 00 und die NOT INVALID KEY Klausel wurde in der Anweisung verwendet, wird die unbedingte Anweisung 2 ausgeführt.

Die END-DELETE Klausel, die funktionell nichts bewirkt, schließt die DELETE-Anweisung ab. Sie ist nicht erforderlich.

Beispiel:

```
002010 FILE-CONTROL.
002020     SELECT DATEI-B ASSIGN TO DISC
002030                    ORGANIZATION IS RELATIVE
002040                    ACCESS MODE IS RANDOM
002050                    RELATIVE KEY IS SCHL-B.

003010 DATE DIVISION.
003020 FILE SECTION.
003030 FD  DATEI-B      BLOCK CONTAINS 51 RECORDS
003040                 RECORD CONTAINS 10 CHARACTERS
003050                 LABEL RECORD IS STANDARD
003070 01  SATZ-B      PIC X(10).

004010 WORKING-STORAGE SECTION.
004020 77  SCHL-B      PIC 999.

005010 PROCEDURE DIVISION.
005020 BEGINN.
005030     OPEN I-O DATEI-B.

005100     MOVE 88 TO SCHL-B.
005110     DELETE DATEI-B INVALID KEY GO TO FEHLER-B.

006090     CLOSE DATEI-B.
```

In diesem Beispiel geht es um eine relative Datei mit Direktzugriff. Das Satzschlüsselfeld für die relative Datei ist SCHL-B. Damit ein Datensatz gelöscht werden kann, muß die Datei zuerst mit der OPEN I-O-Anweisung auf der Ursprungszeile 005030 im I-O Modus eröffnet werden. Der Programmierer bringt die Ganzzahl 88 in das Satzschlüsselfeld SCHL-B, um den 88sten Datensatz zu löschen. Die DELETE-Anweisung auf der Ursprungszeile 005110 löscht dann den 88sten Datensatz in der DATEI-B. Ist kein gültiger Datensatz in der 88sten Datensatzposition der DATEI-B, dann führt die DELETE-Anweisung in INVALID KEY die GO TO FEHLER-B-Anweisung aus. Die CLOSE-Anweisung auf der Ursprungszeile 006090 beendet die Bearbeitung der DATEI-B.

DIE DELETE-ANWEISUNG (indexierte Datei - sequentieller Zugriff)

Bei sequentiellem Zugriff löscht die DELETE-Anweisung einen Datensatz aus einer indexierten Datei.

Format:

DELETE Datei-Name RECORD

Der FILE-CONTROL-Eintrag für den Datei-Namen muß die Klausel ORGANIZATION IS INDEXED enthalten. Der FILE-CONTROL-Eintrag muß indirekt oder ausdrücklich die Klausel ACCESS MODE IS SEQUENTIAL aufweisen.

Die indexierte Datei kann Datensätze fester oder variabler Länge enthalten. Besteht die indexierte Datei aus Datensätzen variabler Länge, dann dürfen die RECORD CONTAINS-Klausel und die Satzbeschreibungen nicht die von dem variable-Längen-Indikator (VLI) eingenommenen zwei Bytes beinhalten. Der COBOL-Compiler setzt automatisch zwei Bytes für den VLI an den Anfang eines Datensatzes mit variabler Länge.

Bevor die DELETE-Anweisung ausgeführt wird, muß die indexierte Datei mit einer OPEN I-O-Anweisung eröffnet sein.

Wenn auf die indexierte Datei sequentiell zugegriffen wird, muß die letzte Anweisung vor der DELETE-Anweisung eine korrekt ausgeführte READ-Anweisung gewesen sein. Bei Ausführung der DELETE-Anweisung wird der Datensatz, auf den durch diese READ-Anweisung zugegriffen wurde, aus der Datei gelöscht.

Die Ausführung der DELETE-Anweisung hat keinen Einfluß auf den Inhalt des Satz-Bereiches.

Nach erfolgreicher Ausführung der DELETE-Anweisung ist der Datensatz aus der Datei gelöscht und es kann nicht mehr darauf zugegriffen werden. Der Satzschlüssel des gelöschten Datensatzes wird physisch aus dem Index gelöscht.

Beispiel:

```
002010 FILE-CONTROL.
002020     SELECT DATEI-C ASSIGN TO DISC
002030     ORGANIZATION IS INDEXED
002040     ACCESS MODE IS SEQUENTIAL, RECORD KEY IS C-NR.

003010 DATA DIVISION.
003020 FILE SECTION.
003030 FD  DATEI-C      BLOCK CONTAINS 20 RECORDS
003040                  RECORD CONTAINS 25 CHARACTERS
003050                  LABEL RECORD IS STANDARD.
003070 01  SATZ-C.
003080     02  C-NR     PIC X(6).
003090     02  FILLER   PIC X(19).

005010 PROCEDURE DIVISION.
005020 BEGINN. OPEN I-O DATEI-C.

005100     READ DATEI-C AT END GO TO ENDE.

005200     DELETE DATEI-C.

005410 ENDE.
005420     CLOSE DATEI-C.
```

In diesem Beispiel geht es um eine indexierte Datei mit sequentiellem Zugriff. Die Datei hat den Namen DATEI-C. C-NR spezifiziert das Satzschlüsselfeld.

Damit ein Datensatz von DATEI-C gelöscht werden kann, muß die Datei zuerst im I-O-Modus eröffnet werden (mit der OPEN I-O-Anweisung auf der Ursprungszeile 005020). Die READ-Anweisung auf der Ursprungszeile 005100 greift sequentiell auf DATEI-C zu, um dadurch in SATZ-C den nächsten Datensatz der DATEI-C verfügbar zu machen. Wenn der Datensatz, auf den durch die READ-Anweisung zugegriffen wurde, gelöscht werden soll, wird eine DELETE-Anweisung ausgeführt. Die DELETE-Anweisung auf der Ursprungszeile 005200 löscht den vorher durch die READ-Anweisung auf der Ursprungszeile 005100 von der DATEI-C gelesenen Datensatz. Die CLOSE-Anweisung auf Ursprungszeile 005420 beendet die Bearbeitung der DATEI-C.

DIE DELETE-ANWEISUNG (indexierte Datei - direkter Zugriff)

Bei direktem Zugriff löscht die DELETE-Anweisung einen bestimmten Datensatz aus einer indexierten Datei.

Format:

```
DELETE Datei-Name RECORD [INVALID KEY unbedingte Anweisung 1]
                         [NOT INVALID KEY unbedingte Anweisung 2]
                         [END-DELETE]
```

Der FILE-CONTROL-Eintrag für den Datei-Namen muß die Klausel ORGANIZATION IS INDEXED und die Klausel ACCESS MODE IS RANDOM oder ACCESS MODE IS DYNAMIC enthalten.

Die indexierte Datei kann entweder Datensätze fester oder variabler Länge enthalten. Besteht die indexierte Datei aus Datensätzen variabler Länge, dann dürfen die RECORD CONTAINS-Klausel und die Satzbeschreibungen nicht die vom VLI eingenommenen zwei Bytes beinhalten. Der COBOL-Compiler setzt automatisch zwei Bytes für den VLI an den Anfang eines Datensatzes mit variabler Länge.

Bevor die DELETE-Anweisung ausgeführt wird, muß die Datei mit einer OPEN I-O-Anweisung für den I-O-Modus eröffnet sein.

Wenn auf die indexierte Datei direkt zugegriffen wird, dann löscht die DELETE-Anweisung den durch das Satzschlüsselfeld bestimmten Datensatz aus der Datei. Die RECORD KEY-Klausel muß im FILE-CONTROL-Eintrag zur Ausführung zweier Funktionen vorhanden sein. Ihre erste Funktion ist es, den Satzschlüssel in jedem Datensatz der indexierten Datei zu bestimmen. Ihre zweite Funktion ist es, das Satzschlüsselfeld im Speicher zu bestimmen. Vor der Ausführung der DELETE-Anweisung muß der Programmierer den Wert des Satzschlüssels in das Satzschlüsselfeld übertragen.

Die Ausführung der DELETE-Anweisung hat keinen Einfluß auf den Inhalt des Satz-Bereiches.

Nach erfolgreicher Ausführung der DELETE-Anweisung ist der Datensatz aus der Datei gelöscht und es kann nicht mehr darauf zugegriffen werden.

Der Satzschlüssel des gelöschten Datensatzes wird physisch aus dem Index gelöscht.

INVALID KEY

INVALID KEY muß für eine DELETE-Anweisung spezifiziert werden, die auf eine indexierte Datei mit direktem oder dynamischem Zugriff Bezug nimmt und keine zugeordnete USE-Anweisung in den Declaratives aufweist. Wenn eine Datei mit Direktzugriff nicht den durch das Satzschlüsselfeld spezifizierten Datensatz enthält, entsteht eine INVALID KEY-Bedingung.

Wenn diese Bedingung auftritt, wird die DELETE-Anweisung nicht ausgeführt und es wird kein Einfluß auf die Datei genommen. Eine INVALID KEY-Bedingung verursacht folgende Vorgänge:

1. Wurde die FILE STATUS-Klausel in dem FILE-CONTROL-Eintrag für die indexierte Datei verwendet, wird ein Wert 23 in das FILE STATUS-Datenfeld übertragen, um anzuzeigen, daß kein Datensatz gefunden wurde.

2. Ist INVALID KEY in der DELETE-Anweisung vorhanden, dann wird die Steuerung auf die unbedingte Anweisung übertragen. Keine der für diese Datei spezifizierten DECLARATIVE-Prozeduren wird ausgeführt. Eine vorhandene NOT INVALID KEY-Klausel wird ignoriert.

3. Ist die INVALID KEY in der DELETE-Anweisung nicht vorhanden, dann werden für diese Datei DECLARATIVE-Prozeduren ausgeführt. Eine vorhandene NOT INVALID KEY-Klausel wird ignoriert.

Tritt keine INVALID KEY-Bedingung auf, wird die INVALID KEY-Klausel ignoriert, falls sie vorhanden ist in der Anweisung. Ds Statusfeld wird mit dem Status beschickt und folgende Aktionen werden ausgeführt:

1. Liegt ein Ausnahmestatus vor (nicht 00), der nicht ein INVALID KEY-Status ist, wird in die einschlägige Declarative-Prozedur verzweigt.

2. Ist der Status 00 und die NOT INVALID KEY-Klausel wurde in der Anweisung nicht verwendet, wird an das Ende der Anweisung gesprungen.

3. Ist der Status 00 und die NOT INVALID KEY-Klausel wurde in der Anweisung verwendet, wird die unbedingte Anweisung 2 ausgeführt.

Beispiel:

```
002010 FILE-CONTROL.
002020     SELECT DATEI-D ASSIGN TO DISC
002030                    ORGANIZATION IS INDEXED
002040                    ACCESS MODE IS RANDOM
002050                    RECORD KEY IS D-NR.

003010 DATA DIVISION.
003020 FILE SECTION.
003030 FD  DATEI-D      BLOCK CONTAINS 51 RECORDS
003040                 RECORD CONTAINS 10 CHARACTERS
003050                 LABEL RECORD IS STANDARD.
003070 01  SATZ-D.
003080     02  D-NR     PIC X(8).
003090     02  FILLER   PIC XX.

005010 PROCEDURE DIVISION.
005020 BEGINN.
005030     OPEN I-O DATEI-D.

005100     MOVE KD-NR TO D-NR.
005110     DELETE DATEI-D INVALID KEY GO TO FEHLER-D.

006140     CLOSE DATEI-D.
```

Im vorangehenden Beispiel handelt es sich um eine indexierte Datei mit Direktzugriff. D-NR spezifiziert die Position des Satzschlüsselfeldes in SATZ-D. Damit ein Datensatz aus DATEI-D gelöscht werden kann, muß die Datei zuerst im I-O-Modus eröffnet werden (mit der OPEN I-O-Anweisung auf der Ursprungszeile 005030). Der Programmierer überträgt den im Datenfeld KD-NR enthaltenen Wert in das Satzschlüsselfeld D-NR. Die DELETE-Anweisung auf der Ursprungszeile 005110 löscht dann denjenigen Datensatz mit einem Satzschlüssel, der dem im Satzschlüsselfeld D-NR enthaltenen Wert entspricht. Wird in DATEI-D kein Datensatz mit einem Satzschlüssel, der im Wert gleich dem von D-NR ist, gefunden, führt die DELETE-Anweisung die GO TO FEHLER-D-Anweisung in der INVALID KEY-Klausel aus. Die CLOSE-Anweisung auf Ursprungszeile 006140 beendet die Bearbeitung der DATEI-D.

Die DISABLE-Anweisung

Sie dient zur Mitteilung an das Message Control System (MCS), daß der Datentransfer eingestellt werden soll zwischen bestimmten Output-Queues und Ausgabe-Adressaten oder zwischen bestimmten Eingabe-Quellen und Input-Queues für Input.

Format:

```
DISABLE  {INPUT  [TERMINAL]}  cd-name-1
         {OUTPUT           }
     [WITH KEY  {Identifier-1}]
                {Literal-1   }
```

- Bei Verwendung der INPUT-Klausel muß cd-name-1 eine Input-CD-Eintragung benennen.
- Bei Verwendung der OUTPUT-Klausel muß cd-name-1 eine Output-CD-Eintragung benennen.
- Literal-1 oder Identifier-1 müssen alphanumerisch sein.

Die DISABLE-Anweisung dient zur logischen Entkopplung zwischen dem MCS und den im Befehl genannten Dateneingabe-Quellen oder Datenausgabe-Adressaten. Existiert diese Entkoppelung bereits oder muß sie anderweitig außerhalb des Programmes bewerkstelligt werden, ist die DISABLE-Anweisung in diesem Programm nicht erforderlich. Der Datentransfer zwischen den COBOL-Programmen und MCS wird von der DISABLE-Anweisung nicht berührt.

Wird in der DISABLE INPUT-Anweisung das Wahlwort TERMINAL mitverwendet, wird der logische Pfad zwischen der Dateneingabequelle und allen Queues und Subqueues deaktiviert. Nur der Inhalt des durch Datenname-7 (SYMBOLIC SOURCE) adressierten Feldes innerhalb des CD-Name-1-Bereichs ist von Bedeutung.

Wird in der DISABLE INPUT-Anweisung das Wahlwort TERMINAL nicht verwendet, werden alle logischen Pfade deaktiviert zwischen allen Queues und Sub-Queues und allen ihnen zugewiesenen Terminals. Betroffen sind die von Datenname-1 (Symbolic Queue) bis inklusive Datenname-4 (Symbolic Subqueue-3) innerhalb des CD-Name-1-Bereiches beschriebenen Queues.

Wird die OUTPUT-Variante der Anweisung verwendet, wird der logische Pfad für einen Ausgabe-Adressaten oder werden die logischen Pfade für alle Ausgabe-Adressaten (beschrieben mittels Datenname-5 (SYMBOLIC DESTINATION) innerhalb des CD-Name-1-Bereiches) deaktiviert.

Literal-1 oder Identifier-1 werden mit einem Paßwort verglichen, das in der Network Definition Language (NDL) vergeben wurde. Die DISABLE-Anweisung wird mit angenommen, wenn Literal-1 oder Identifier-1 mit dem Paßwort übereinstimmen. Ist dies nicht der Fall, wird ein Wert übertragen in das Feld STATUS KEY innerhalb des mit CD-Name-1 bezeichneten Bereiches. MCS kann 1- bis 10stellige Paßwörter handhaben.

MCS stellt sicher, daß die DISABLE-Anweisung die logische Diskonnektion zum frühesten Zeitpunkt ausführt, zu dem die Dateneingabe-Quelle oder der Datenausgabe-Adressat inaktiv ist. Die DISABLE-Anweisung wird niemals einen im Gange befindlichen Datenübertragungsvorgang ohne Abwarten seiner Beendigung abbrechen.

DIE DISPLAY-ANWEISUNG

Die DISPLAY-Anweisung bewirkt, daß ein oder mehrere Datenfelder auf den Bildschirm oder Drucker übertragen werden.

Format 1:

```
DISPLAY  {Identifier-1}  ...
         {Literal-1   }
[UPON mnemonischer Name 1]
[WITH NO ADVANCING]
```

Format 2:

```
DISPLAY  {[Identifier-1]}  [ LINE  {Identifier-2}]
         {[Literal-1   ]}          {Literal-2   }
         [UNDERLINE]
[ POSITION  {Identifier-3}]  [ SIZE  {Identifier-4}]
            {Literal-3   }           {Literal-4   }
         [BEEP]
[ ERASE  [{EOL}]]  [{HIGH}]  [BLINK]
         [{EOS}]   {LOW }
         [REVERSE]  ...
```

Allgemeines:

- **Der Mnemonische Name kann in der SPECIAL-Names-Klausel in der Environment Division dem Bildschirm (Konsole) oder dem Drucker zugewiesen werden.**
- **Ein Literal kann auch jede figurative Konstante außer ALL literal sein.**
- **Ein numerisches Literal darf kein Vorzeichen haben.**
- **Wahlwörter können in beliebiger Reihenfolge geschrieben werden.**

Zu Format 1:

Der Inhalt eines jeden Operanden wird auf dem Bildschirm oder auf dem Drucker ausgegeben in der Reihenfolge, wie die Operanden in der Anweisung genannt sind.

Fehlt die UPON-Klausel, ist der Bildschirm Default.

Die WITH NO ADVANCING-Klausel bewirkt, daß vor der Datenausgabe kein Zeilenvorschub erfolgt. Stattdessen werden die Daten rechts im Anschluß an die letzte Datenausgabe angefügt bzw. erfolgt, wenn dort kein Platz mehr ist (und das Ausgabegerät dies zuläßt), ein Überschreiben der Zeile.

Zu Format 2:

Die DISPLAY-Anweisung überträgt den Inhalt von Identifier-1 (oder Literal-1) usw. (...) auf dem Bildschirm. Jedem Operanden kann eine LINE-, POSITION-, eine SIZE- und eine ERASE-Angabe beigefügt sein.

Identifier-1 muß entweder ein alphanumerisches oder ein numerisches Datenfeld sein. Ist der Identifier-1 ein numerisches Datenfeld, dann muß die Speicherungsart desselben entweder dezimal ohne Vorzeichen oder ungepackt dezimal mit Vorzeichen sein. Literal-1 kann jede beliebige figurative Konstante sein. Ohne Vorhandensein von SIZE führt die figurative Konstante nur zu einer ein Byte großen Ausgabe auf dem Bildschirm. Ist SIZE spezifiziert, dann wird die Ausgabe der figurativen Konstante auf dem Bildschirm so oft wiederholt, wie dies durch den Wert in SIZE bestimmt ist.

Enthält die DISPLAY-Anweisung mehr als einen Operanden, werden die Werte der Operanden in der Reihenfolge übertragen, in der die Operanden in der DISPLAY-Anweisung spezifiziert sind.

Es folgt eine Beschreibung der Wahl-Angaben LINE, POSITION, SIZE und ERASE:

LINE

Der Wert von Identifier-2 oder Literal-2 spezifiziert die Zeilennummer, auf der die Daten auf dem Bildschirm ausgegeben werden sollen. Identifier-2 muß ein numerisches Datenfeld ohne Vorzeichen sein, Literal-2 ein numerisches Literal ohne Vorzeichen. Der Wert der Zeilennummer reicht von 1 bis 24. Ausgaben auf Zeile 23 und 24 sollten jedoch möglichst gering gehalten werden, da diese Zeilen jederzeit von Systemmeldungen belegt werden können.

Wird für einen Operanden kein LINE angegeben, erfolgt die Ausgabe auf der nächsttieferen Zeile. War der Cursor vorher auf Zeile 24, schiebt sich das ganze Bild um eine Zeile nach oben und auf Zeile 24 entsteht dabei eine neue Zeile. Die oberste Zeile geht damit verloren.

Bei keiner LINE-Angabe (oder Zeilennummer 0) und Cursor-Position 0 findet keine Cursor-Bewegung statt. Bei Zeilennummer 0 und einer Cursor-Position ungleich 0 wird der Cursor auf die nächste Zeile an der angezeigten Position gestellt. Ist der Cursor laufend auf Zeile 24 positioniert, dann rollt das Bild auf dem Bildschirm. Zeile 24 ist dann immer die laufende Zeile.

POSITION

Der Wert von Identifier-3 oder Literal-3 in POSITION spezifiziert die Nummer der Position, auf die der Cursor innerhalb der bestimmten Zeile vor der Anzeige von Daten auf dem Bildschirm zu positionieren ist. Dies ist die äußerste linke Position eines Bildschirm-Feldes, auf das Daten mit der DISPLAY-Anweisung ausgegeben werden. Identifier-3 muß ein numerisches Datenfeld ohne Vorzeichen sein, Literal-3 ein numerisches Literal ebenfalls ohne Vorzeichen.

Der Wert der Cursor-Positionen reicht von 1 bis 80 bzw. bei bestimmten Terminals bis 132, wobei der Wert 1 den Cursor ganz links auf dem Bildschirm und der Wert 80 diesen ganz rechts auf dem Bildschirm positioniert. Wird für einen Operanden keine POSITION-Angabe spezifiziert, dann wird der Cursor ganz links auf dem Bildschirm positioniert.

Bei Cursor-Position 0 und Zeilennummer 0 ist keine Cursor-Bewegung gegeben. Ist die Cursor-Positionszahl ungleich 0 und die Zeilennummer 0, wird der Cursor auf der nächsten Zeile an die entsprechende Position gestellt. Ist der Cursor laufend auf Zeile 24, dann rollt das Bild auf dem Bildschirm, und Zeile 24 ist die laufende Zeile.

SIZE

Der Wert von Identifier4 oder Literal4 in der SIZE-Angabe spezifiziert die Anzahl von Bytes, die zur Anzeige auf den Bildschirm zu übermitteln sind. Identifier-4 muß ein numerisches Datenfeld ohne Vorzeichen sein, Literal-4 ein numerisches Literial ebenfalls ohne Vorzeichen. Ist für einen Operanden kein SIZE spezifiziert, dann wird angenommen, daß die Anzahl von Bytes, die Identifier-1 oder Literal-1 einnehmen, diejenige Anzahl von Bytes ist, die dem Bildschirm zu übermitteln ist.

Ist die Anzahl der anzuzeigenden Bytes größer als die Anzahl Positionen, die sich aus der Cursor-Position zum rechten Ende der Zeile hin ergeben, wird der Überhang auf der nächsten Zeile ausgegeben.

ERASE

Das Schlüsselwort ERASE bei einem Operanden löst aus, daß der ganze Bildschirm gelöscht wird, bevor der Inhalt dieses Operanden auf dem Bildschirm angezeigt wird. Ist ERASE nicht angegeben, dann wird der Bildschirm vor der Anzeige dieses Operanden nicht gelöscht.

Beispiel 1:

```
005040      DISPLAY "PROGRAMM-BEGINN" LINE 12 ERASE.
```

Die durch die DISPLAY-Anweisung ausgelösten Vorgänge sind folgende:

- Der Bildschirm wird gelöscht.
- Der Cursor wird auf Position 1 der Zeile 12 gesetzt.
- Der Text PROGRAMM-BEGINN wird auf Zeile 12 ab Position 1 ausgegeben.

Beispiel 2:

```
004010 WORKING-STORAGE SECTION.
004020 77  FEHLER-ZAHL      PIC 9999.
004030 77  FEHLER-ANZEIGE   PIC Z.ZZ9.

006010 PROCEDURE DIVISION.
006410      MOVE FEHLER-ZAHL TO FEHLER-ANZEIGE.
006420      DISPLAY FEHLER-ANZEIGE LINE 10 POSITION 35 ERASE
006430              " FEHLER" LINE 10 POSITION 40.
```

Die durch diese Prozedur-Anweisungen ausgelösten Vorgänge sind folgende:

- Eine Fehler-Anzahl (z. B. 16) wird von FEHLER-ZAHL in das edierte Feld FEHLER-ANZEIGE übertragen.

- Der Bildschirm wird gelöscht.

- Der Cursor wird auf Position 35 von Zeile 10 positioniert.

- Die 5 Bytes, enthaltend 16, werden auf Zeile 10 ab Position 35 ausgegeben.

- Der Cursor wird auf Position 40 von Zeile 10 gesetzt.

- Die 7 Bytes mit dem INHALT FEHLER werden auf Zeile 10 ab Position 40 ausgegeben.

Beispiel 3:

```
004020 WORKING-STORAGE SECTION.
004070 01  PREIS-TABELLE.
004080      02  FILLER  PIC X(11)   VALUE IS "B0100055015".
004090      02  FILLER  PIC X(11)   VALUE IS "B0500045053".
004100      02  FILLER  PIC X(11)   VALUE IS "B1000032053".
004110      02  FILLER  PIC X(11)   VALUE IS "B9999027053".
004120      02  FILLER  PIC X(11)   VALUE IS "R0050060015".
004130      02  FILLER  PIC X(11)   VALUE IS "R0100058032".
004140      02  FILLER  PIC X(11)   VALUE IS "R0200051032".
004150      02  FILLER  PIC X(11)   VALUE IS "R0500054732".
004160      02  FILLER  PIC X(11)   VALUE IS "R9999040032".
004170*
```

```
004180 01  PREIS-TABELLE1  REDEFINES PREIS-TABELLE.
004190     02  PREIS-POSTEN OCCURS 9 TIMES INDEXED BY TAB-IND.
004200         03  T-CODE       PIC X.
004210         03  T-MENGE      PIC 9999.
004220         03  PREIS        PIC V999.
004230         03  PROZENT      PIC V999.
004240
004290 77  FABRIK-PREIS         PIC V999.
004300 77  FABRIK-PROZENT  REDEFINES FABRIK-PREIS  PIC 99V9.
004310
004350 77  ZEILEN-NR            PIC 99.
004360
004370 01  TAB-ZEILE.
004380     02  FILLER           PIC XXXX VALUE IS SPACE.
004390     02  ELEMENT-NR       PIC 9.
004400     02  FILLER           PIC X(12) VALUE IS SPACE.
004410     02  C-CODE           PIC X.
004420     02  FILLER           PIC X(14) VALUE IS SPACE.
004430     02  C-MENGE          PIC 9999.
004440     02  FILLER           PIC X(13) VALUE IS SPACE.
004450     02  C-PREIS          PIC $,999.
004460     02  FILLER           PIC X(13) VALUE IS SPACE.
004470     02  C-PROZENT        PIC 99,9.
004480     02  FILLER           PIC X    VALUE IS "%".
004490     02  FILLER           PIC X(8) VALUE IS SPACE.

005010 PROCEDURE DIVISION.
005050 TAB-ANZEIGE.
005060     DISPLAY "LFD NR  SERVICE CODE   VERBRAUCHSMENGE"
005061               LINE 2 POSITION 2 ERASE
005062             "   FRACHTKOSTEN   RABATTSATZ"
005070               LINE 2 POSITION 43.
005080     MOVE 1 TO ELEMENT-NR.
005090     SET TAB-IND TO 1.
005100     MOVE 3 TO ZEILEN-NR.
005110 BILDSCH-AUSGABE.
005120     MOVE T-CODE (TAB-IND) TO C-CODE.
005130     MOVE T-MENGE (TAB-IND) TO C-MENGE.
005140     MOVE PREIS (TAB-IND) TO C-PREIS.
005150     MOVE PROZENT (TAB-IND) TO FABRIK-PREIS.
005160     MOVE FABRIK-PROZENT TO C-PROZENT.
005170     DISPLAY TAB-ZEILE LINE ZEILEN-NR.
005180     IF ELEMENT-NR IS LESS THAN 9
005190             ADD 1 TO ELEMENT-NR
005200             ADD 1 TO ZEILEN-NR
005210             SET TAB-IND UP BY 1
005220             GO TO BILDSCH-AUSGABE.
```

Die durch die DISPLAY-Anweisung ausgelösten Vorgänge auf den Ursprungszeilen 005060 und 005070 sind folgende:

- Der Bildschirm wird gelöscht.
- Der Cursor wird auf Position 2 der Zeile 2 gesetzt.
- Das 70 Bytes umfassende Literal in der DISPLAY-Anweisung wird auf Zeile 2 ab Position 2 ausgegeben.

Die durch die DISPLAY-Anweisung auf der Ursprungszeile 005170 ausgelösten Vorgänge, die ja nach der ersten Anzeige und Löschung erfolgen, sind infolge der Prozeduren auf Ursprungszeilen 005120 bis 005160 folgende:

- Der Cursor wird auf Position 1 der Zeile 3 gesetzt.
- Die durch den Identifier TAB-ZEILE definierten 80 Bytes werden auf Zeile 3 ab Position 1 ausgegeben.

Da ELEMENT-NR 1 entspricht und weniger als 9 ist, werden ELEMENT-NR, ZEILEN-NR und TAB-IND jeweils um 1 erhöht, so daß auf das nächste Tabellenelement zugegriffen werden kann, wenn die Prozeduren auf Ursprungszeilen 005120 bis 005170 wiederholt werden. Somit werden durch die Wiederholung der Anweisungen in Ursprungszeilen 005120 bis 005170 das zweite bis neunte Element der Wert-Tabelle auf den Zeilen 4 bis 11 des Bildschirmes angezeigt. Nachdem diese Prozeduren neunmal ausgeführt wurden, zeigt der Bildschirm folgendes Bild:

LFD NR	SERVICE CODE	VERBRAUCHSMENGE	FRACHTKOSTEN	RABATTSATZ
1	B	0100	$,055	01,5%
2	B	0500	$,045	05,3%
3	B	1000	$,032	05,3%
4	B	9999	$,027	05,3%
5	R	0050	$,060	01,5%
6	R	0100	$,058	03,2%
7	R	0200	$,051	03,2%
8	R	0500	$,047	03,2%
9	R	9999	$,040	03,2%

Weitere Möglichkeiten bei der Ausgabe von Daten auf dem Bildschirm sind mit Hilfe der Parameter UNDERLINE, BEEP, EOL, EOS, HIGH, LOW, BLINK UND REVERSE gegeben:

UNDERLINE	zum Unterstreichen der Ausgabe
BEEP	schaltet den Signalgeber bei der Ausgabe ein
EOL	für das Löschen (erase) bis zum Zeilenende
EOS	für das Löschen (erase) bis zum Bildschirmende
HIGH	für normale Intensität der Ausgabe
LOW	für halbe Intensität der Ausgabe
BLINK	für das Blinken der Ausgabe
REVERSE	um die Ausgabe durch Farbumkehrung hervorzuheben

Beispiel:
```
DISPLAY "FEHLER" LINE 20
            REVERSE ERASE EOS
        FEHL-NR LINE 20 POSITION 8
            BEEP UNDERLINE
        "ABBRUCH" LINE 20 POSITION 11
            BLINK.
```

Video-Attribute

Zusätzlich zu dem bisher Beschriebenen gibt es die Möglichkeit, DISPLAY-Ausgaben auf dem Bildschirm zu modifizieren. Diese Möglichkeiten sind z. B.: die halbe Intensität, das Blinken, das Umkehren der Anzeige und das Unterstreichen von Bildschirm-Ausgaben.

Eine oder mehrere der zusätzlichen Möglichkeiten werden dadurch erreicht, daß ein 6stelliger hexadezimaler Steuerblock, der eine Zeichen-Position auf dem Bildschirm belegt, vor das zu sendende Datenfeld oder Literal übertragen wird.

Die Funktion des hexadezimalen Steuerblocks bleibt für alle nachfolgenden Zeichen so lange erhalten, bis ein anderer hexadezimaler Steuerblock auf dem Bildschirm erkannt wird.

Die folgende Tabelle zeigt die 6stelligen hexadezimalen Steuerblöcke und ihre Funktion für Bildschirme:

Hexadezimaler Steuerblock	Funktion
1B3040	Normale Anzeige
1B3041	Halbe Intensität
1B3042	Blinken
1B3043	Blinken, halbe Intensität
1B3050	Umkehrung der Anzeige
1B3051	Umkehrung, halbe Intensität
1B3052	Umkehrung, Blinken
1B3053	Umkehrung, Blinken, halbe Intensität
1B3060	Unterstreichung
1B3061	Unterstreichung, halbe Intensität
1B3062	Unterstreichung, Blinken
1B3063	Unterstreichung, Blinken, halbe Intensität
1B3070	Unterstreichung, Umkehrung
1B3071	Unterstreichung, Umkehrung, halbe Intensität
1B3072	Unterstreichung, Umkehrung, Blinken
1B3073	Unterstreichung, Umkehrung, Blinken, halbe Intensität

Alle 16 hexadezimalen Steuerblöcke können in einem COBOL Ursprungs-Programm mit folgenden Dateneintragungen in der WORKING-STORAGE SECTION oder der GLOBAL SECTION eingebaut werden:

```
WORKING-STORAGE SECTION.
01  7900-CODES.
    02  NORMAL                  PIC XXX VALUE H"1B3040".
    02  HALBE                   PIC XXX VALUE H"1B3041".
    02  BLINK                   PIC XXX VALUE H"1B3042".
    02  BLINK-HALBE             PIC XXX VALUE H"1B3043".
    02  UMK                     PIC XXX VALUE H"1B3050".
    02  UMK-HALBE               PIC XXX VALUE H"1B3051".
    02  UMK-BLINK               PIC XXX VALUE H"1B3052".
    02  UMK-BLINK-HALBE         PIC XXX VALUE H"1B3053".
    02  UNTER                   PIC XXX VALUE H"1B3060".
    02  UNTER-HALBE             PIC XXX VALUE H"1B3061".
    02  UNTER-BLINK             PIC XXX VALUE H"1B3062".
    02  UNTER-BLINK-HALBE       PIC XXX VALUE H"1B3063".
    02  UNTER-UMK               PIC XXX VALUE H"1B3070".
    02  UNTER-UMK-HALBE         PIC XXX VALUE H"1B3071".
    02  UNTER-UMK-BLINK         PIC XXX VALUE H"1B3072".
    02  UNTER-UMK-BLINK-HALBE   PIC XXX VALUE H"1B3073".
```

Die folgenden Beispiele zeigen die Möglichkeiten zur Ausgabe von Daten auf dem Bildschirm in halber Intensität, Blinken, Umkehrung und Unterstreichung:

Beispiel 1:

```
PROCEDURE DIVISION.
NACHRICHT.

    DISPLAY NORMAL                  LINE 1                ERASE.
    DISPLAY BLINK                   LINE 4.
    DISPLAY "FEHLER"                LINE 4  POSITION 2.
    DISPLAY NORMAL                  LINE 4  POSITION 8.
    DISPLAY "IN UEBERTRAGUNGS-NUMMER" LINE 4    POSITION 9.
    DISPLAY BLINK                   LINE 4  POSITION 32.
    DISPLAY TNO                     LINE 4  POSITION 33.
    DISPLAY NORMAL                  LINE 4  POSITION 39.
```

Die Ergebnisse dieser DISPLAY-Anweisungen sind folgende:

- Der Bildschirm wird gelöscht.
- Zeile 1, Position 1, enthält den Hex-Code für normale Bildschirm-Ausgabe.
- Zeile 4, Position 1, enthält den Hex-Code für Blinken.
- Zeile 4, Position 2 bis 7, enthält das blinkende Literal FEHLER.
- Zeile 4, Position 8, enthält den Hex-Code für normale Bildschirm-Ausgabe.
- Zeile 4, Position 9 bis 31, enthält das Literal IN UEBERTRAGUNGS-NUMMER, das in normaler Bildschirm-Ausgabe erscheint.
- Zeile 4, Position 32, enthält den Hex-Code für Blinken.
- Zeile 4, Position 33 bis 38, enthält die 6stellige Übertragungsnummer, die blinkt und aus dem Feld TNO stammt.
- Zeile 4, Position 39, enthält den Hex-Code für normale Bildschirm-Ausgabe.

Beispiel 2:

```
WORKING-STORAGE SECTION.
01  ARB80    PIC X(80).

PROCEDURE DIVISION.
ANZEIGE.
    MOVE SPACES TO ARB80.
    STRING  BLINK "WARNUNG" NORMAL "DATEI-UEBERLAUF"
            UNTER "5.000"   NORMAL "SAETZE"
    DELIMITED BY SIZE   INTO ARB80.
    DISPLAY ARB80 LINE 8 POSITION 1 ERASE.
```

Die Ergebnisse dieser DISPLAY-Anweisungen sind folgende:

- Zeile 8, Position 1, enthält den Hex-Code für Blinken.
- Zeile 8, Position 2 bis 8, enthält das blinkende Literal WARNUNG.
- Zeile 8, Position 9, enthält den Hex-Code für normale Bildschirm-Ausgabe.
- Zeile 8, Position 10 bis 24, enthält das Literal DATEI-UEBERLAUF in normaler Bildschirm-Ausgabe.
- Zeile 8, Position 25, enthält den Hex-Code für Unterstreichung.
- Zeile 8, Position 26 bis 30, enthält das Literal 5.000, das unterstrichen ist.
- Zeile 8, Position 31, enthält den Hex-Code für normale Bildschirm-Ausgabe.
- Zeile 8, Position 32 bis 37, enthält das Literal SAETZE in normaler Bildschirm-Ausgabe.

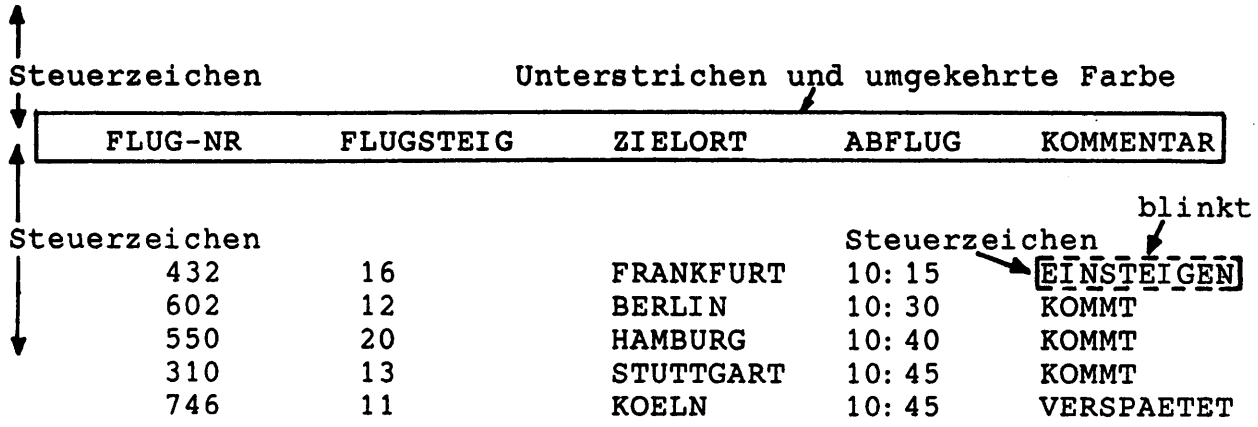
Beispiel 3:

```
WORKING-STORAGE SECTION.
01  FLUG-TAB.
    02  FLUG-POSTEN OCCURS 5 TIMES INDEXED BY F-IND.
        03  FLUG-NR          PIC XXX.
        03  TFLUGSTEIG       PIC XX.
        03  TZIELORT         PIC X(12).
        03  TABFLUG          PIC X(5).
        03  TKOMMENTAR       PIC X(10).
01  LINE-NR PIC 99.

PROCEDURE DIVISION.
ANZEIGE.
    DISPLAY NORMAL                    LINE 1  ERASE.
    DISPLAY UNTER-UMK                 LINE 6.     DISPLAY
    "FLUG-NR   FLUGSTEIG   ZIELORT    ABFLUG    KOMME
-   "NTAR       "                  LINE 6  POSITION 10.
    DISPLAY NORMAL                        LINE 7.
    MOVE 9 TO LINE-NR.
    SET F-IND TO 1.
AUFBAU.
    DISPLAY TFLUG-NR (F-IND) LINE LINE-NR   POSITION 13
            TFLUGSTEIG (F-IND) LINE LINE-NR POSITION 26
            TZIELORT (F-IND) LINE LINE-NR   POSITION 35
            TABFLUG (F-IND)  LINE LINE-NR   POSITION 53.
    IF TKOMMENTAR (F-IND) = "EINSTEIGEN"
       DISPLAY BLINK LINE LINE-NR  POSITION 64.
    DISPLAY TKOMMENTAR (F-IND) LINE LINE-NR POSITION 65.
    ADD 1 TO LINE-NR.
    DISPLAY NORMAL LINE LINE-NR.
    IF F-IND = 5    GO  ANZ-ENDE.
    SET F-IND UP BY 1.
                    GO  AUFBAU.

ANZ-ENDE.
```

Die Ergebnisse dieser Anweisungen sind folgende:

- Der Bildschirm wird gelöscht.
- Zeile 1, Position 1, enthält den Hex-Code für normale Bildschirm-Ausgabe.
- Zeile 6, Position 1, enthält den Hex-Code für Umkehrung der Ausgabe und Unterstreichung.

- Zeile 6, Position 10 bis 79, enthält Überschriften, die in umgekehrter Schrift und unterstrichen ausgegeben werden.
- Zeile 7, Position 1, enthält den Hex-Code für normale Bildschirm-Ausgabe.
- Zeile 9, Position 13 bis 57, enthält die 1. Datenzeile in normaler Bildschirm-Ausgabe.
- Enthält das anzugebende Feld KOMMENTAR den Inhalt EINSTEIGEN, dann enthält die Zeile 9, Position 64, den Hex-Code für Blinken und in Position 65 bis 74 wird das Literal EINSTEIGEN blinkend ausgegeben.
- Wenn das Feld KOMMENTAR nicht den Inhalt EINSTEIGEN enthält, wird der Inhalt von Feld KOMMENTAR von Position 65 bis 74 in normaler Bildschirm-Ausgabe ausgegeben.
- Zeile 10, Position 1, enthält den Hex-Code für normale Bildschirm-Ausgabe.
- Die letzten vier Aktionen werden nun für jede der Datenzeilen wiederholt, die auf dem Bildschirm angezeigt werden.



DIE DIVIDE-ANWEISUNG

Die DIVIDE-Anweisung dient zum Dividieren. Der Quotient und der Rest werden in ein oder mehrere Datenfelder gespeichert.

Format 1:

```
DIVIDE {Identifier-1} BY {Identifier-2}
       {Literal-1   }    {Literal-2   }
       GIVING Identifier-3 [ROUNDED]  ...
      [ON SIZE ERROR unbedingte Anweisung 1]
      [NOT ON SIZE ERROR unbedingte Anweisung 2]
      [END-DIVIDE]
```

Format 2:

```
DIVIDE {Identifier-1} BY {Identifier-2}
       {Literal-1   }    {Literal-2   }
       GIVING Identifier-3 [ROUNDED]
       REMAINDER Identifier-4
      [ON SIZE ERROR unbedingte Anweisung 1]
      [NOT ON SIZE ERROR unbedingte Anweisung 2]
      [END-DIVIDE]
```

Bei Format 1 wird der Wert von Identifier-1 oder Literal-1 durch den Wert von Identifier-2 oder Literal-2 dividiert. Der Quotient wird in Identifier-3, Identifier-4, usw., also in den Ergebnisfeldern, gespeichert.

Bei Format 2 wird der Wert von Identifier-1 oder Literal-1 durch den Wert von Identifier-2 oder Literal-2 dividiert. Der Quotient wird in Identifier-3, dem Ergebnisfeld, gespeichert. Der Rest wird im Feld Identifier-4 abgestellt.

Identifier-1 und Identifier-2 müssen numerische Elementarfelder bezeichnen. Jedes Literal in der DIVIDE-Anweisung muß numerisch sein. Identifier-3, Identifier-4, usw., können entweder ein numerisches oder ein numerisch-editiertes Elementarfeld bezeichnen.

Die Datenbeschreibungen der Operanden in der DIVIDE-Anweisung brauchen nicht identisch sein, denn jede notwendige Umwandlung und Dezimalkommaausrichtung erfolgt automatisch während des Rechenvorganges.

DIVIDE ist eine unbedingte Anweisung. Bei Vorhandensein von (NOT) SIZE ERROR wird DIVIDE jedoch zu einer bedingten Anweisung.

ROUNDED

Die DIVIDE-Anweisung führt die Division bis zur Größe des Ergebnisfeldes aus. Folglich erfolgt ohne Vorhandensein der ROUNDED-Angabe ein Abschneiden des Divisionsergebnisses, wenn dieses im Ergebnisfeld gespeichert wird. Ist ROUNDED spezifiziert, dann erfolgt die Division um eine Stelle mehr. Ist die Überhangziffer des Divisionsergebnisses größer als 4, dann erhöht sich der absolute Wert des Divisionsergebnisses um 1. Werden die Ganzzahlpositionen in dem Ergebnisfeld durch P in der PICTURE-Zeichenfolge dargestellt, dann erfolgt das Runden oder Abschneiden relativ zur äußersten rechten Ganzzahlposition, welcher Speicherplatz zugeordnet ist.

REMAINDER

Die REMAINDER-Angabe wird in der DIVIDE-Anweisung spezifiziert, wenn das Verbleiben eines Restes aus dem Divisionsvorgang gewünscht wird. In COBOL wird der Divisionsrest definiert als das Ergebnis der Subtraktion des Produktes des Quotienten (Identifier-3) und des Divisors vom Dividenden. Ist Identifier-3 als ein numerisch editiertes Datenfeld definiert, ist der zum Berechnen des Divisionsrestes verwendete Quotient ein Zwischendatenfeld, das der Compiler zur Speicherung des uneditierten Quotienten erzeugt. Sind ROUNDED und REMAINDER spezifiziert, dann ist der zur Berechnung des Divisionsrestes verwendete Quotient ein Zwischendatenfeld, das den Quotienten der DIVIDE-Anweisung nicht in abgerundeter, sondern in abgeschnittener Form enthält. Der Divisionsrest ist eine positive Ganzzahl, die nach entsprechender Dezimalkommaausrichtung und nach Abschneiden im Identifier-4 steht.

SIZE ERROR

Nach Dezimalkommaausrichtung und Runden wird, wenn SIZE ERROR spezifiziert ist, eine Prüfung auf eine SIZE ERROR-Bedingung hin vorgenommen. Die SIZE ERROR-Bedingung liegt vor, wenn der Wert des Quotienten den höchsten von Identifier-4 aufnehmbaren Wert überschreitet. Nur der linke Rand des Empfangsdatenfeldes wird auf Überlauf geprüft. Tritt beim Quotienten die SIZE ERROR-Bedingung auf, dann werden die Werte des Quotienten und des Divisionsrestes nicht gespeichert. Tritt die SIZE ERROR-Bedingung beim Divisionsrest auf, dann wird der Wert des Quotienten in Identifier-3 gespeichert, während Identifier-4 unverändert bleibt. Wenn die DIVIDE-Anweisung mehr als ein Empfangsfeld hat und in einem dieser Felder wird eine SIZE ERROR-Bedingung festgestellt, dann wird, nachdem die DIVIDE-Anweisung ausgeführt wurde, das Programm bei der unbedingten Anweisung fortgesetzt. Die Werte der Empfangsfelder, die die SIZE ERROR-Bedingung erfüllen, bleiben unverändert erhalten. Die Schreibweise "unbedingte Anweisung" in der SIZE ERROR-Angabe bezeichnet einen oder mehrere aufeinanderfolgende unbedingte Anweisungen, deren Folge durch einen Punkt beendet wird.

Ist bei spezifizierter SIZE ERROR-Angabe die SIZE ERROR-Bedingung nicht gegeben, werden die Werte des Quotienten und des Divisionsrestes gespeichert und die nächste Anweisung wird ausgeführt. Die unbedingte Anweisung in SIZE ERROR wird nicht ausgeführt.

Ohne spezifizierte SIZE ERROR-Angabe sind bei Auftreten einer SIZE ERROR-Bedingung die Werte des Quotienten und/oder Divisionsrestes unvorhersagbar. Die nächste Anweisung nach der DIVIDE-Anweisung wird ausgeführt und der Fehler nicht festgestellt.

Bei Verwendung des Wahlwortes NOT zusätzlich in der SIZE ERROR-Klausel wird bei erfolgreicher Division die unbedingte Anweisung durchgeführt, dagegen im Falle eines SIZE ERRORs die Divisions-Anweisung beendet und die nachfolgende Anweisung ausgeführt.

Beispiel 1:

```
    DIVIDE MENGE BY 4 GIVING ERGEBNIS.

MENGE vorher              0 3 9 +
ERGEBNIS vorher           1 4^4 +
MENGE nachher             0 3 9 + unverändert
ERGEBNIS nachher          0 9^7 + enthält das Ergebnis
```

Beispiel 2:

```
    DIVIDE MENGE BY 4 GIVING ERGEBNIS ROUNDED.

MENGE vorher              0 3 9 +
ERGEBNIS vorher           1 4^4 +
MENGE nachher             0 3 9 + unverändert
ERGEBNIS nachher          0 9^8 + enthält das aufgerundete
                                  Ergebnis
```

Beispiel 3:

```
    DIVIDE MENGE-A BY 4 GIVING ERGEBNIS REMAINDER REST.

MENGE-A vorher              3 9
ERGEBNIS vorher           0 9 2^1
REST vorher                 3 2
MENGE-A nachher             3 9 unverändert
ERGEBNIS nachher          0 0 9^7 enthält das Ergebnis
REST nachher                0 2 enthält den Rest
```

Die ENABLE-Anweisung

Sie eröffnet den Datentransfer zwischen definierten Output-Queues und Datenausgabe-Adressaten oder zwischen definierten Dateneingabe-Quellen und Input-Queues für Input.

Format:

```
ENABLE {INPUT [TERMINAL]}  cd-name-1
       {OUTPUT          }

       [WITH KEY {Identifier-1}]
                 {Literal-1   }]
```

- Die INPUT-Variante erfordert, daß der CD-Name-1 eine im Programm definierte Input-Communication-Description (CD) bezeichnet.

- Die OUTPUT-Variante erfordert, daß der CD-Name-1 eine im Programm definierte Output-Communication-Description (CD) bezeichnet.

- Literal-1 oder Identifier-1 müssen alphanumerisch sein.

Die ENABLE-Anweisung stellt eine logische Verbindung her zwischen dem MCS und den definierten Eingabe- oder Ausgabe-Medien. Existiert diese logische Verbindung bereits oder wird sie anderweitig außerhalb dieses Programmes getätigt, ist die ENABLE-Anweisung in diesem Programm nicht erforderlich. Der logische Pfad für den Datentransfer zwischen COBOL-Programmen und dem MCS wird von der ENABLE-Anweisung nicht berührt.

Wird bei der INPUT-Variante das zusätzliche Wahlwort TERMINAL angegeben, wird der logische Pfad aktiviert zwischen dem Eingabe-Medium und allen zur Verfügung stehenden Queues und Sub-Queues, die bereits "enabled" sind. Für MCS ist dabei nur der Inhalt von Datenname-7 (SYMBOLIC SOURCE) des durch CD-Name-1 bezeichneten Datenbereichs von Bedeutung.

Wird bei der INPUT-Variante das Wahlwort TERMINAL weggelassen, wird der logische Pfad aktiviert zwischen allen Eingabe-Medien und allen zur Verfügung stehenden Queues und Sub-Queues, die bereits "enabled" sind. Für MCS ist dabei nur der Inhalt von Datenname-7 (SYMBOLIC SOURCE) des durch CD-Name-1 bezeichneten Datenbereichs von Bedeutung.

Wird bei der INPUT-Variante das Wahlwort TERMINAL weggelassen, wird der logische Pfad aktiviert zwischen allen Eingabe-Medien und den ihnen zugeordneten Queues und Sub-Queues, definiert im durch CD-Name-1 bezeichneten Datenbereich mittels Datenname 1 (SYMBOLIC QUEUE) bis Datenname 4 (SYMBOLIC SUB-QUEUE-3).

Bei Verwendung der OUTPUT-Variante wird der logische Pfad für das Ausgabe-Medium oder die logischen Pfade für alle Ausgabe-Medien aktiviert, beschrieben im Datenfeld 5 (SYMBOLIC DESTINATION) des mittels CD-Name-1 bezeichneten Bereichs.

Literal-1 oder Identifier-1 werden mit einem Paßwort verglichen, das mittels NDL-Anweisungen definiert wurde. Die ENABLE-Anweisung wird lediglich dann ausgeführt, wenn Literal-1 oder Identifier-1 sich mit dem Paßwort decken. Ist dies nicht der Fall, wird im Feld STATUS KEY des mit CD-Name-1 bezeichneten Bereichs ein Status gesetzt. Wurde in der NDL kein Paßwort definiert, erfolgt keine Paßwortkontrolle und es wird kein Status ausgegeben.

MCS bearbeitet Paßwörter in einer Länge von 1 bis 10 Stellen.

Die ENTER-Anweisung

Diese Anweisung wird lediglich hier noch aufgeführt, um darauf hinzuweisen, daß sie veraltet ist. Der NCR COBOL 85 Compiler behandelt sie nur noch wie einen Kommentar.

Format:

ENTER Sprachen-Name [Routinen-Name].

Hinweis: Um zu Routinen in einer anderen Sprache überzuwechseln, verwenden Sie die CALL-Anweisung!

Die EVALUATE-Anweisung

Die EVALUATE-Anweisung ermöglicht eine Mehrfachverzweigungs- bzw. Mehrfachverknüpfungsstruktur. (Ähnliche Strukturen gibt es in anderen Sprachen). Hierbei besteht die Möglichkeit, mehrere Bedingungen überprüfen zu lassen. Der weitere Ablauf des Programmes hängt von den Ergebnissen dieser Überprüfungen bzw. Auswertungen (Evaluation) ab.

Format:

```
EVALUATE { Identifier-1 }  [ ALSO { Identifier-2 } ] ...
         { Literal-1    }         { Literal-2    }
         { Ausdruck-1   }         { Ausdruck-2   }
         { TRUE         }         { TRUE         }
         { FALSE        }         { FALSE        }

  {{ WHEN
  {  ANY
  {  Bedingung-1
  {  TRUE
  {  FALSE
  {  [NOT] {{ Identifier-3 }  [{ THROUGH } { Identifier-4 }]}}
  {        {{ Literal-3    }  [{ THRU    } { Literal-4    }]}}
  {        {{ Arithmetischer Ausdr.-1 }  [{ THROUGH } { Arithmet. Ausdr.-2 }]}}
  }

[ALSO
  {  ANY
  {  Bedingung-2
  {  TRUE
  {  FALSE
  {  [NOT] {{ Identifier-5 }  [{ THROUGH } Identifier-6 ]}}...}...
  {        {{ Literal-5    }  [{ THRU    } Literal-6    ]}}
  {        {{ Arithm. Ausdr.-3 }  [{ THROUGH } Arithm. Ausdr.-4 ]}}
  }
  unbedingte Anweisung-1} ...
  [WHEN OTHER unbedingte Anweisung-2]
  [END-EVALUATE]
```

Syntax-Regeln:

Die Operanden bzw. die Werte TRUE oder FALSE vor der ersten WHEN-Klausel werden als "Selection Subjects" betrachtet, in ihrer Summe als "Set of Selection Subjects".

Die Operanden bzw. die Worte TRUE, FALSE und ANY innerhalb einer WHEN-Klausel werden als "Selection Objects" betrachtet, in ihrer Summe innerhalb einer WHEN-Klausel als "SET OF SELECTION OBJECTS".

Die Worte THROUGH und THRU sind gleichbedeutend.

Die beiden Operanden, die durch THROUGH miteinander verbunden sind, müssen der gleichen Klasse angehören. Die so miteinander verbundenen Operanden bilden ein einzelnes Selection Object.

Die Anzahl der Selection Objects innerhalb jedes Set of Selection Objects muß gleich der Anzahl der Selection Subjects sein.

Jedes Selection Object innerhalb eines Set of Selection Objects muß mit dem Selection Subject korrespondieren, das die gleiche reihenfolgemäßige Position innerhalb das Set of Selection Subjects hat. Hierzu gelten nachstehende Regeln:

- Identifiers, Literale oder arithmetische Ausdrücke innerhalb eines Selection Object müssen für Vergleiche mit den korrespondierenden Operanden im Set of Selection Subject gültige Operanden sein.

- Bedingung-1, Bedingung-2 oder die Worte TRUE oder FALSE, die als Selection Object angegeben worden sind, müssen mit einem Bedingungs-Ausdruck oder den Worten TRUE oder FALSE im Set of Selection Subjects korrespondieren.

- Das Wort ANY darf mit einem Selection Subject eines beliebigen Typs korrespondieren.

Funktionsweise:

Die EVALUATE-Anweisung arbeitet so, daß alle Selection Subjects und Selection Objects verglichen werden, die einen numerischen oder nichtnumerischen Wert, eine Reihe numerischer oder nichtnumerischer Werte oder einen TRUE-Wert darstellen. Unter diesen Werten ist folgendes zu verstehen:

- Jedes Selection Subject, das durch Identifier-1 bzw. Identifier-2 verkörpert wird, und jedes Selection Object, das durch Identifier-3 bzw. Identifier-5 verkörpert wird (ohne die NOT- oder THROUGH-Klauseln), hat den Wert und die Klasse des mittels des Identifiers bezeichneten Datenfeldes inne.

- Ein Selection Subject, das durch Literal-1 bzw. Literal-2 verkörpert wird, und ein Selection Object, das durch Literal-3 bzw. Literal-5 verkörpert wird (ohne die NOT- oder THROUGH-Klauseln), hat den Wert und die Klasse des Literals. Handelt es sich bei den Literalen 3 und 5 um die figurative Konstante ZERO, wird ihm die Klasse des korrespondierenden Selection Subject zugewiesen.

- Ein Selection Subject, in dem Ausdruck-1 und Ausdruck-2 arithmetische Ausdrücke darstellen, und ein Selection Object (ohne NOT- oder THROUGH-Klausel), in dem der arithmet. Ausdruck-1 und der arithmetische Ausdruck-3 verwendet werden, werden als numerisch aufgefaßt.

- Ein Selection Subject, in dem Ausdruck-1 und Ausdruck-2 Bedingungsausdrucke darstellen, und ein Selection Object, in dem Bedingung-1/Bedingung-2 verwendet werden, verkörpern einen TRUTH-Wert.

- Ein Selection Subject oder ein Selection Object, verkörpert durch eines der Wörter TRUE oder FALSE, gilt als TRUTH-Wert.

- Ein Selection Object, verkörpert durch das Wort ANY, wird keiner Überprüfung (Evaluation) unterzogen.

- THROUGH bei einem Selection Object (ohne NOT) bewirkt, daß eine Kette von Werten alle zulässigen Werte des Selection Subjects umfaßt, die entsprechend den Vergleichsregeln größer oder gleich dem ersten Operanden und kleiner oder gleich dem zweiten Operanden sind.

- NOT bei einem Selection Object bewirkt eine Negierung der Werte, die dem Selection Object ohne die NOT-Klausel adäquat wären.

Unter den vorab beschriebenen Voraussetzungen arbeitet EVALUATE folgendermaßen:

- Jedes Selection Object innerhalb des Set of Selection Objects für die erste WHEN-Klausel wird mit demjenigen Selection Subject verglichen, das an der gleichen Stelle (innerhalb seines Set of Selection Subjects) steht wie das Selection Object. Eine der folgenden Bedingungen muß erfüllt sein, damit der Vergleich erfüllt ist: Wenn die zu vergleichenden Datenbereiche numerischen oder nichtnumerischen Inhalt haben oder aus einer Reihe numerischer oder nichtnumerischer Werte bestehen, ist der Vergleich erfüllt, wenn der Wert (oder einer aus einer Reihe von Werten) des Selection Object entsprechend den Vergleichsregeln gleich dem Wert des Selection Subjects ist. Sind den zu vergleichenden Komponenten TRUTH-Werte zugeordnet, ist ein Vergleich dann erfüllt, wenn die verglichenen Komponenten identische TRUTH-Werte aufweisen. Wird das Selection Object durch das Wort ANY gebildet, gilt der Vergleich immer als erfüllt.

- Sind die vorangehenden Vergleiche für alle Selection Objects innerhalb eines Set of Selection Objects erfüllt, gilt diejenige WHEN-Klausel, die dieses Set of Selection Objects enthält, als die, in welcher der Vergleich mit dem Set of Selection Subjects erfüllt ist.

- Widrigenfalls gilt der Vergleich als nicht erfüllt und

- die Prozedur wird mit etwaigen weiteren WHEN-Klauseln in der Reihenfolge ihres Auftretens wiederholt, bis entweder eine WHEN-Klausel auftritt, die mit dem Set of Selection Subjects übereinstimmt oder bis alle Sets of Selection Objects ohne Erfolg verglichen worden sind.

Nach Abschluß der Vergleichsoperation geht EVALUATE in seine letzte Phase:

- Bei erfülltem Vergleich wird die unbedingte Anweisung 1 derjenigen WHEN-Klausel ausgeführt, in der der Vergleich als erfüllt festgestellt wurde.

- Bei nicht erfülltem Vergleich wird, falls eine WHEN OTHER-Klausel verwendet wurde, die unbedingte Anweisung 2 ausgeführt. Ohne WHEN OTHER-Klausel wird die auf die EVALUATE-Anweisung folgende Anweisung ausgeführt.

DIE EXIT-ANWEISUNG

Die EXIT-Anweisung bestimmt einen gemeinsamen Endpunkt für eine Reihe von Prozeduren, die durch eine PERFORM-Anweisung angesprochen werden.

Format:

EXIT

Der EXIT-Anweisung muß ein Paragraphen-Name vorausgehen und sie muß die einzige Anweisung im Paragraph sein.

Im folgenden Beispiel stellt die EXIT-Anweisung einen gemeinsamen Endpunkt für eine Folge von Paragraphen dar, auf die durch eine PERFORM-Anweisung Bezug genommen wird.

Die PERFORM-Anweisung im Paragraph PRUEFUNG führt die von ABFRAGE bis ABFRAGE-EXIT reichende Folge von Prozeduren aus. In diesen Prozeduren führt die Prüfung des CODE-1 zu den Verzweigungen PRUEF-A, PRUEF-B, PRUEF-C und ABFRAGE-EXIT.

Entspricht der CODE-1 12, dann wird zu PRUEF-B verzweigt. Am Ende von PRUEF-B erfolgt die Rückkehr zur auf die auslösende PERFORM-Anweisung folgenden Anweisung (SATZ-A). Dies wird durch die Anweisung GO TO ABFRAGE-EXIT erreicht, in der ABFRAGE-EXIT der Paragraphen-Name der EXIT-Anweisung ist, die der auslösenden PERFORM-Anweisung zugeordnet ist.

Entspricht der CODE-1 15, dann wird zu PRUEF-C verzweigt. Am Ende von PRUEF-C erfolgt die Rückkehr zur auf die auslösende PERFORM-Anweisung folgenden Anweisung (SATZ-A). Da der unmittelbar auf das Ende von PRUEF-C folgende Paragraph der ABFRAGE-EXIT-Paragraph ist, ist es nicht nötig, die Anweisung GO TO ABFRAGE-EXIT zu verwenden, um die Rückkehr zur auslösenden PERFORM-Anweisung zu bewirken.

Beispiel:

```
PRUEFUNG.    PERFORM ABFRAGE THRU ABFRAGE-EXIT.
SATZ-A.
             ----------
ABFRAGE.
    IF CODE-1 = 12 GO TO PRUEF-B.
    IF CODE-1 = 15 GO TO PRUEF-C.
    IF CODE-1 = 16 GO TO ABFRAGE-EXIT.
PRUEF-A.
             ----------

    GO TO ABFRAGE-EXIT.
PRUEF-B.
             ----------

    GO TO ABFRAGE-EXIT.
PRUEF-C.
             ----------

ABFRAGE-EXIT.    EXIT.
UNTERPROG-A.
             ----------
```

Entspricht der CODE-1 16, dann erfolgt eine unmittelbare Rückkehr auf die Anweisung (SATZ-A), die auf die auslösende PERFORM-Anweisung folgt. Die Rückkehr wird durch die Anweisung GO TO erzielt, in der ABFRAGE-EXIT der Paragraphen-Name der EXIT-Anweisung ist, die der auslösenden PERFORM-Anweisung zugeordnet ist.

Entspricht der CODE-1 weder 12, 15 noch 16, dann wird zu PRUEF-A weitergegangen. Am Ende von PRUEF-A erfolgt die Rückkehr zur auf die auslösende PERFORM-Anweisung folgenden Anweisung (SATZ-A). Die Rückkehr wird durch die Anweisung GO TO ABFRAGE-EXIT erzielt, in der ABFRAGE-EXIT der Paragraphen-Name der EXIT-Anweisung ist, die der auslösenden PERFORM-Anweisung zugeordnet ist.

Null-Paragraph anstatt EXIT:

Da ein Paragraph aus einem Paragraphen-Namen ohne weitere Sätze bestehen kann, darf der Programmierer einen solchen Null-Paragraphen verwenden, um einen gemeinsamen Endpunkt für eine Reihe von Prozeduren, wie sie in einer PERFORM-Anweisung vorkommen, zu schaffen. Im folgenden Beispiel ist der Paragraphen-Name ABFR-ENDE der eines Null-Paragraphen mit einem gemeinsamen Endpunkt für die ausgeführten Prozeduren ABFRAGE bis ABFR-ENDE.

Bei Ausführung der PERFORM-Anweisung wird der Null-Paragraph für eine Rückkehr zur nächsten auf die PERFORM-Anweisung folgenden, ausführbaren Anweisung bestimmt. Ist der Null-Paragraph ausgeführt, dann kehrt die Steuerung zur nächsten auf die PERFORM-Anweisung folgenden, ausführbaren Anweisung zurück und der Null-Paragraph ist abgearbeitet. Kommt es jedoch nicht zur Ausführung des Null-Paragraphen, dann verbleibt der Null-Paragraph gesetzt für eine Rückkehr zur nächsten auf die PERFORM-Anweisung folgenden, ausführbaren Anweisung.

Beispiel:

```
PRUEFUNG.    PERFORM ABFRAGE THRU ABFR-ENDE.
SATZ-A.
             ----------

ABFRAGE.
    IF CODE-1 = 12 GO TO PRUEF-B.
    IF CODE-1 = 15 GO TO PRUEF-C.
    IF CODE-1 = 16 GO TO ABFR-ENDE.
PRUEF-A.
             ----------

    GO TO ABFR-ENDE.
PRUEF-B.
             ----------

    GO TO ABFR-ENDE.
PRUEF-C.
             ----------

ABFR-ENDE.
UNTERPROG-A.
```

Die EXIT PROGRAM-Anweisung

Diese Anweisung markiert das logische Ende eines mittels CALL aufgerufenen Programmes.

Format:

EXIT PROGRAM

Anwendungsbeispiele für diese Anweisung finden Sie im Kapitel "Inter-Program-Communication" in diesem Buch.

DIE GO TO-ANWEISUNG

Die GO TO-Anweisung dient zum Springen von einem Teil der PROCEDURE DIVISION zu einem anderen.

Format 1:

GO TO [Prozedur-Name]

Format 2:

GO TO Prozedur-Name [, Prozedur-Name-2] ...[, Prozedur-Name-n]
DEPENDING ON Identifier.

Wird Format 1 der GO TO-Anweisung ausgeführt, erfolgt ein Sprung zu Prozedur-Name.

Wird Format 2 der GO TO-Anweisung ausgeführt, erfolgt ein Sprung zu einer der angegebenen Prozeduren, d. h. Prozedur-Name-1, Prozedur-Name-2, usw., abhängig vom ganzzahligen Wert 1, 2 ... n, der im Identifier enthalten ist. Ist der Wert im Identifier ein anderer als der der positiven Ganzzahl ohne Vorzeichen 1, 2 ...n, hat die GO TO-Anweisung keine Wirkung und die unmittelbar auf die GO TO-Anweisung folgende Anweisung wird ausgeführt. Der Identifier muß der Name eines numerischen Elementarfeldes ohne Dezimalstellen sein.

Ein Prozedur-Name, wie er in beiden Formaten verwendet wird, kann der Name eines Paragraphen oder eines Kapitels (SECTION) in der PROCEDURE DIVISION des Programmes sein.

Beispiel:

```
GO TO EINZAHLUNG AUSZAHLUNG ZINSEN DEPENDING ON ARB-CODE.
```

Die obige GO TO-Anweisung ermöglicht einen Sprung zu der Prozedur EINZAHLUNG, AUSZAHLUNG oder ZINSEN, abhängig vom Inhalt des Feldes ARB-CODE. Die folgenden Anweisungen sind gleichbedeutend mit der obigen GO TO DEPENDING ON-Anweisung.

```
IF ARB-CODE = 1 GO TO EINZAHLUNG.
IF ARB-CODE = 2 GO TO AUSZAHLUNG.
IF ARB-CODE = 3 GO TO ZINSEN.
```

DIE IF-ANWEISUNG

Die IF-Anweisung prüft eine Bedingung. Die nachfolgende Verarbeitung des Objekt-Programmes hängt davon ab, ob die Bedingung erfüllt oder nicht erfüllt ist.

Format:

```
IF Bedingung  THEN  {Anweisung-1 ...}
                    {NEXT SENTENCE }

  {{ELSE     }                          [END-IF]}
  {{OTHERWISE} Anweisung-2  ...         [      ]}
  {
  {  ELSE NEXT SENTENCE                         }
  {                                             }
  {  END-IF                                     }
```

Syntax:

- Anweisung-1 und Anweisung-2 stehen für unbedingte oder bedingte Anweisungen. Ihnen können wiederum bedingte Anweisungen folgen.
- Die Klausel ELSE NEXT SENTENCE kann entfallen, wenn sie unmittelbar vor dem Punkt stehen würde, der die IF-Anweisung beendet.
- Wird die END-IF-Klausel verwendet, darf die NEXT SENTENCE-Klausel nicht verwendet werden.

Weitere Hinweise:

Eine IF-Anweisung kann auf folgende Arten beendet werden:

- Eine END-IF-Klausel auf derselben Ebene der Verschachtelung
- Einen Punkt mit anschließend mindestens einem Space
- Bei Verschachtelung durch eine ELSE-Klausel mit einer anschließenden IF-Anweisung einer höheren Stufe.

Die Ausführung einer IF-Anweisung erfolgt auf folgende Weise:

- Bei erfüllter Bedingung wird Anweisung-1 ausgeführt, falls eine vorhanden ist. Handelt es sich bei Anweisung-1 um eine Sprunganweisung in eine andere Prozedur oder um eine erneute bedingte Anweisung, hängt der weitere Ablauf von diesen ab. Enthält Anweisung-1 keine Sprunganweisung in eine andere Prozedur und keine Bedingungsanweisung, wird die ELSE-Klausel, falls vorhanden, ignoriert, und der nächste Satz gelangt zur Ausführung.

- Bei erfüllter Bedingung wird, falls anstelle von Anweisung-1 die Klausel NEXT SENTENCE steht, die ELSE-Klausel, falls vorhanden, ignoriert, und der nächste Satz gelangt zur Ausführung.

- Bei nicht erfüllter Bedingung wird Anweisung-1 oder NEXT SENTENCE ignoriert und (falls programmiert) Anweisung-2 gelangt zur Ausführung. Handelt es sich bei Anweisung-2 um eine Sprunganweisung in eine andere Prozedur oder um eine erneute bedingte Anweisung, hängt der weitere Ablauf von diesen ab. Enthält Anweisung-2 keine Sprung-Anweisung und keine Bedingungs-Anweisung, gelangt der nächste Satz zur Ausführung. Fehlt die Klausel ELSE Anweisung-2, gelangt der nächste Satz zur Ausführung. Dasselbe geschieht beim Vorhandensein der Klausel ELSE NEXT SENTENCE.

Anweisung-1 und/oder Anweisung-2 können ihrerseits eine IF-Anweisung enthalten. In diesem Falle spricht man von einer verschachtelten IF-Anweisung.

IF-Anweisungen innerhalb von IF-Anweisungen werden als gleichgestellte IF-, ELSE- und END-IF-Kombinationen aufgefaßt, die von links nach rechts abgearbeitet werden. Daher wird jedes sich ergebende ELSE oder END-IF als zu dem unmittelbar vorhergehenden IF zugehörig betrachtet, das keinem ELSE oder END-IF gleichgestellt wurde.

In Beispiel 1 werden die Anweisungen ADD 7 TO BETRAG und MOVE BETRAG TO ARB-BETRAG ausgeführt, wenn COD gleich 20 ist. Die Steuerung geht dann auf den dem IF-Satz folgenden Satz über. In diesem Beispiel wird DIVIDE SALDO BY 2 GIVING SALDO-NEU als nächste Anweisung ausgeführt.

Beispiel 1:

```
IF COD  IS EQUAL TO 20 ADD 7 TO BETRAG MOVE BETRAG TO
    ARB-BETRAG ELSE SUBTRACT 10 FROM BETRAG MOVE BETRAG TO
    ZUSATZ-BETR.
DIVIDE SALDO BY 2 GIVING SALDO-NEU.
```

Ist die Bedingung nicht erfüllt, dann werden die unmittelbar auf das Wort ELSE folgenden Anweisungen (Anweisung-2) ausgeführt. Die Steuerung geht danach auf den Satz nach dem IF-Satz über. In Beispiel 1 werden die Anweisungen SUBTRACT 10 FROM BETRAG und MOVE BETRAG TO ZUSATZ-BETR ausgeführt, wenn COD nicht gleich 20 ist. In diesem Beispiel wird DIVIDE SALDO BY 2 GIVING SALDO-NEU als nächste Anweisung ausgeführt.

Wird für eine nicht erfüllte Bedingung keine Aktion angezeigt, dann wird Anweisung-2 durch die reservierten Wörter NEXT SENTENCE ersetzt. Diese bedeutet, daß die Steuerung dann direkt auf den Satz nach dem IF-Satz übergeht. Im folgenden Beispiel 2 wird die Anweisung DIVIDE SALDO BY 2 GIVING SALDO-NEU ausgeführt, wenn COD nicht gleich 33 ist. Ist COD gleich 33, dann werden die Anweisungen MOVE 66 TO CODE-NEU und MOVE BETRAG TO NEU-BETRAG vor der Anweisung DIVIDE SALDO BY 2 GIVING SALDO-NEU ausgeführt.

Beispiel 2:

```
IF COD  IS EQUAL TO 33 MOVE 66 TO CODE-NEU MOVE BETRAG TO
    NEU-BETRAG ELSE NEXT SENTENCE.
DIVIDE SALDO BY 2 GIVING SALDO-NEU.
```

Die IF-Anweisung kann ohne ELSE NEXT SENTENCE erfolgen, wenn die nicht erfüllte Bedingung keine Aktionen außer dem Übergang der Steuerung auf den Satz, der auf den IF-Satz folgt, erfordert. Somit könnte die IF-Anweisung gemäß Beispiel 2 ohne die Angabe ELSE NEXT SENTENCE, wie in Beispiel 3 dargestellt, geschrieben werden. Ist COD nicht gleich 33, dann geht die Steuerung auf den nächsten Satz DIVIDE SALDO BY 2 GIVING SALDO-NEU über.

Beispiel 3:

```
IF COD  IS EQUAL TO 33 MOVE 66 TO CODE-NEU MOVE BETRAG TO
                  NEU-BETRAG.
DIVIDE SALDO BY 2 GIVING SALDO-NEU.
```

Wird für eine erfüllte Bedingung keine Aktion verlangt, dann wird Anweisung-1 durch die reservierten Wörter NEXT SENTENCE ersetzt. Dies bedeutet, daß die Steuerung dann direkt auf den Satz nach dem IF-Satz übergeht. Im folgenden Beispiel 4 wird die unbedingte Anweisung MOVE NEU-BETRAG TO ARB-BETRAG ausgeführt, wenn COD gleich 50 ist. Ist COD nicht gleich 50, dann wird die unbedingte Anweisung ADD 100 TO NEU-BETRAG vor der Anweisung MOVE NEU-BETRAG TO ARB-BETRAG ausgeführt.

Beispiel 4:

```
IF COD IS EQUAL TO 50 NEXT SENTENCE ELSE ADD 100 TO
    NEU-BETRAG.
MOVE NEU-BETRAG TO ARB-BETRAG.
```

Im Format der IF-Anweisung können Anweisung-1 und Anweisung-2 sein:

- eine oder mehrere unbedingte Anweisungen;
- eine IF-Anweisung;
- eine oder mehrere unbedingte Anweisungen, gefolgt von einer IF-Anweisung;
- eine IF-Anweisung, gefolgt von einer IF-Anweisung.

Im Format der IF-Anweisung ist das Wort Bedingung ein beliebiger bedingter Ausdruck. Ein bedingter Ausdruck ist entweder eine einfache oder eine Mehrfach-Bedingung. Es gibt verschiedene Arten von einfachen Bedingungen: Vergleichs-Bedingung, Klassen-Bedingung, Vorzeichen-Bedingung, Schalter-Zustands-Bedingung und Bedingungs-Namen-Bedingung. Es gibt Mehrfach-Bedingungen: die kombinierte Bedingung und die verneinende einfache Bedingung.

Die Vergleichs-Bedingung

Diese Bedingung vergleicht zwei Operanden, von denen jeder entweder ein Daten-Name oder ein Literal sein kann.

Format:

```
{Identifier-1}                                {Identifier-2}
{Literal-1   } Vergleichsoperator             {Literal-2   }
```

Der erste Operand, bestehend aus Identifier-1 oder Literal-1, ist Subjekt der Vergleichs-Bedingung. Der zweite Operand, bestehend aus Identifier-2 oder Literal-2, ist Objekt der Vergleichs-Bedingung. Sie müssen nicht die gleiche Anzahl von Dezimalstellen haben, da die Dezimalpunktausrichtung automatisch erfolgt.

Der Vergleichsoperator gibt die in einer Vergleichs-Bedingung enthaltene Art des Verhältnisses an. Aus der folgenden Übersicht wird die Bedeutung jedes Vergleichsoperators klar:

Vergleichs-Operator	Vergleich
IS GREATER THAN IS >	größer als
IS NOT GREATER THAN IS NOT >	nicht größer als
IS LESS THAN IS <	kleiner als
IS NOT LESS THAN IS NOT <	nicht kleiner als
IS EQUAL TO IS =	gleich
IS NOT EQUAL TO IS NOT =	nicht gleich
IS GREATER THAN OR EQUAL TO IS > =	größer als oder gleich
IS LESS THAN OR EQUAL TO IS < =	kleiner als oder gleich

Eine Vergleichs-Bedingung ist erfüllt, wenn ein angegebenes Verhältnis zwischen den Operanden besteht. Zwei numerische Operanden können ungeachtet des in den einzelnen USAGE-Klauseln angegebenen Formats verglichen werden. In allen anderen Vergleichen müssen die Operanden jedoch die gleichen USAGE-Klauseln haben. Ist einer der Operanden eine Datengruppe, dann gelten für den Vergleich die nicht-numerischen Vergleichsregeln.

Numerische Operanden in einer Vergleichs-Bedingung

Ein numerischer Operand ist ein numerisches Literal oder ein numerisches Datenfeld. Ein numerisches Datenfeld hat eine Beschreibung, die seinen Inhalt auf einen Wert beschränkt, der von aus den Ziffern 0 bis 9 ausgewählten Zeichen dargestellt wird; falls mit einem Vorzeichen versehen, kann das Feld entweder ein positives oder ein negatives Vorzeichen enthalten.

Ein numerisches Datenfeld hat eine PICTURE-Zeichenfolge, die lediglich die Symbole 9, P, S und V umfaßt. Zwei numerische Operanden können ungeachtet des in den einzelnen USAGE-Klauseln angegebenen Formats verglichen werden.

Der Vergleich von numerischen Operanden bestimmt, daß im Hinblick auf die algebraischen Werte der Operanden einer davon kleiner als der andere, gleich dem anderen oder größer als der andere ist. Die Länge der Operanden, die durch die Anzahl der Ziffern ausgedrückt wird, ist nicht entscheidend. Vor dem Vergleich werden die zwei Operanden nach ihren Dezimalseparatoren ausgerichtet. Weisen die Operanden ungleiche Längen auf, läuft der Vergleich, wie wenn der kürzere mit führenden Nullen auf die Länge des längeren ergänzt wäre.

Im Vergleich von numerischen Operanden wird Null ungeachtet des Vorzeichens als eindeutiger Wert angesehen. Numerische Operanden ohne Vorzeichen werden im Vergleich als positiv angesehen.

Beispiele des Vergleichs numerischer Operanden:

Operand 1	Operand 2	Ergebnis des Vergleichs
125,50	12,4010	125.50 ist größer als 12.4010
151	51,25	151 ist größer als 51.25
1000,0	1000,1	1000.0 ist kleiner als 1000.1
01,0-	00,0+	- 1.0 ist kleiner als + 0.0
00-	00+	0 ist gleich 0
34	0128	34 ist kleiner als 0128
025	25	025 ist gleich 25

Nicht-numerische Operanden in einer Vergleichs-Bedingung

Ein nicht-numerischer Operand ist ein nicht-numerisches Literal oder ein nicht-numerisches Datenfeld. Ein nicht-numerisches Datenfeld kann aus jeder Kombination von Zeichen des ASCII-Zeichenvorrats bestehen. Ein Vergleich von nicht-numerischen Operanden bestimmt, daß gemäß der für dieses Programm durch die Klausel PROGRAM COLLATING SEQUENCE im OBJECT-COMPUTER-Paragraphen der ENVIRONMENT DIVISION festgelegten Sortierfolge einer der Operanden kleiner als der andere, gleich dem anderen oder größer als der andere ist. Als Standard gilt der ASCII-Code.

Beim Vergleich von nicht-numerischen Operanden wird die Gesamtzahl der Zeichen im Operanden als die Operandengröße angesehen. Nach linksbündiger Ausrichtung sind die zwei nicht-numerischen Operanden entweder von gleicher oder ungleicher Größe.

Sind die zwei nicht-numerischen Operanden in einer Vergleichs-Bedingung von gleicher Größe, dann werden Zeichen in gleichen Zeichenpositionen der zwei Operanden verglichen, und zwar von der höchsten Stelle (links) bis zur niedrigsten Stelle (rechts). Erweisen sich die zu vergleichenden Zeichenpaare vom ersten bis zum letzten Paar als gleich, so gelten die Operanden als gleich, sobald der Vergleich der niedrigsten Stelle (rechts) erfolgt ist. Das erste Paar ungleicher Zeichen, auf das gestoßen wird, wird verglichen, um die relative Position dieser Zeichen in der Sortierfolge zu bestimmen. Als der größere Operand gilt derjenige, der das in der Sortierfolge höher eingestufte Zeichen enthält.

Sind die zwei nicht-numerischen Operanden in einer Vergleichs-Bedingung von ungleicher Größe, dann erfolgt der Vergleich der Zeichen von links nach rechts. Der Vergleich läuft, bis auf ein Paar ungleicher Zeichen gestoßen wird oder bis bei einem der Operanden keine weiteren Zeichen mehr vorhanden sind. Ist das Ende des kürzeren Operanden erreicht, so gilt dieser als dem längeren Operanden nachstehend, falls nicht die übrigen Zeichen im längeren Operanden Leerstellen sind. Ist dies der Fall, dann gelten die zwei Operanden als gleich.

Ein nicht-numerischer Operand und ein numerischer Operand können nur verglichen werden, wenn sie dieselbe USAGE-Klausel aufweisen. Der numerische Operand muß entweder ein ganzzahliges Datenfeld oder ein ganzzahliges Literal sein. Der Vergleich dieser zwei Operanden verläuft dann in gleicher Weise wie der Vergleich von zwei nicht-numerischen Operanden, der in den vorangehenden Absätzen beschrieben ist.

Beispiele des Vergleichs nicht-numerischer Operanden:

Operand 1	Operand 2	Ergebnis des Vergleichs
AB954	AB954	AB954 ist gleich AB954
A540	9450	A450 ist größer als 9450
950	95J	950 ist kleiner als 95J
ABCD	STUV	ABCD ist kleiner als STUV
MNOPQ	ABC	MNOPQ ist größer als ABC
ABC	MNOPQ	ABC ist kleiner als MNOPQ
JBHNALCARN	JBHNALCAN	JBHNALCARN ist größer als JBHNALCAN
JKHN$$AL	JKHN$$AM	JKHN$$AL ist kleiner als JKHN$$AM
DEF	DEFG	DEF ist kleiner als DEFG
DEF	DEF$	DEF ist gleich DEF$

Boolesche Operanden in einer Vergleichs-Bedingung

Ein boolescher Operand ist ein boolesches Literal oder ein boolesches Datenfeld. Ein boolesches Datenfeld hat eine Beschreibung derart, daß sie aus dem booleschen Zeichenvorrat 0 und 1 bestehen kann. Ein boolesches Datenfeld hat eine PICTURE-Zeichenfolge, die lediglich aus Datenzeichen 1 besteht.

Der Vergleich von booleschen Operanden bestimmt, daß im Hinblick auf die booleschen Werte der Operanden einer davon kleiner als der andere, gleich dem anderen oder größer als der andere ist. Beide der booleschen Operanden werden linksbündig ausgerichtet, bevor sie in der Vergleichs-Bedingung verglichen werden. Nach linksbündiger Ausrichtung sind die zwei booleschen Operanden entweder von gleicher oder ungleicher Größe.

Sind die zwei booleschen Operanden von gleicher Größe, dann werden die booleschen Zeichen in gleichen Zeichenpositionen der zwei Operanden verglichen, und zwar von der höchsten Stelle (links) bis zur niedrigsten Stelle (rechts). Erweisen sich die zu vergleichenden booleschen Zeichenpaare vom ersten bis zum letzten als gleich, so gelten die Operanden als gleich, sobald der Vergleich der niedrigsten Stelle (rechts) erfolgt ist.

Sind die zwei booleschen Operanden in einer Vergleichs-Bedingung von ungleicher Größe, dann erfolgt der Vergleich der booleschen Zeichen von links nach rechts. Der Vergleich läuft, bis auf ein Paar ungleicher boolescher Zeichen gestoßen wird oder bis bei einem der Operanden keine weiteren booleschen Zeichen mehr vorhanden sind. Ist das Ende des kürzeren Operanden erreicht, so gilt dieser als dem längeren Operanden nachstehend, falls nicht die übrigen booleschen Zeichen im längeren Operanden alle 0 sind. Ist dies der Fall, dann gelten die zwei Operanden als gleich.

Es folgen Beispiele des Vergleichs boolescher Operanden; dabei sind diejenigen booleschen Zeichen unterstrichen, durch die eine Ungleichheit erkannt wird:

Operand 1	Operand 2	Ergebnis des Vergleichs
01010101	01010101	01010101 = 01010101
00001111	00000000	00001111 > 00000000
00001111	00101111	00001111 < 00101111
0000111101010101	0000111101010101	0000111101010101 = 0000111101010101
0001000000010100	0001000000011100	0001000000010100 < 0001000000011100
0111111100000000	0011111100000000	0111111100000000 > 0011111100000000
00001111	0000111110000000	00001111 < 0000111110000000
00001111	0000111100000000	00001111 = 0000111100000000
0000111110000000	00001111	0000111110000000 > 00001111

Vergleichs-Bedingung bei Index oder Index-Datenfeld

Die Vergleichs-Bedingung kann dazu verwendet werden, um einen Vergleich, bei dem einer oder beide Operanden aus einem Index oder einem Index-Datenfeld bestehen, durchzuführen. Beim Vergleich eines Index-Datenfeldes mit einem Index (oder einem anderen Index-Datenfeld) werden die Aktualwerte ohne Umwandlung in eine Tabellenpostennummer verglichen. Vergleicht man zwei Indexe, so ist das Ergebnis dasselbe, wie wenn man entsprechende Tabellenpostennummern vergleicht. Beim Vergleich eines Indexes mit einem Datenfeld (nicht einem Index-Datenfeld) wird die Tabellenpostennummer des Indexes mit dem Datenfeld verglichen. Die Tabellenpostennummer des Indexes wird auch verwendet, wenn ein Index mit einem Literal verglichen wird. Ein Vergleich betreffend ein Index oder ein Index-Datenfeld ist unzulässig, wenn die Operanden nicht mit den genannten Vergleichsstrukturen übereinstimmen. Die folgende Tabelle faßt die in einer Vergleichs-Bedingung zulässigen Kombinationen von Index und Index-Datenfeld zusammen.

Operand 1	Operand 2	Vergleich von
Index-Datenfeld	Index	Aktualwerten
Index-Datenfeld	Index-Datenfeld	Aktualwerten
Index	Index	Tabellenpostennummer
Index	Daten-Name, der kein Index-Datenfeld ist	Tabellenpostennummer
Index	Literal	Tabellenpostennummer

Klassen-Bedingung

Die Klassen-Bedingung bestimmt, ob der Operand numerisch oder alphanumerisch ist, nur aus Groß- (Upper) oder nur aus Kleinbuchstaben (Lower) besteht oder aus einer Zeichenfolge, die unter SPECIAL-NAMES mittels einer CLASS-Namens definiert wurde. Ein numerischer Operand besteht aus den Zeichen 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 mit oder ohne Vorzeichen. Ein alphabetischer Operand besteht aus den Buchstaben A bis Z und dem Leerzeichen. Die Datenbeschreibung des Operanden muß die Klausel USAGE IS DISPLAY enthalten.

Format:

```
Identifier IS [NOT] { NUMERIC          }
                    { ALPHABETIC       }
                    { ALPHABETIC-LOWER }
                    { ALPHABETIC-UPPER }
                    { CLASS-Name       }
```

Beim ALPHABETIC-Test (Test eines alphabetischen Operanden) muß der Identifier eine PICTURE-Zeichenfolge enthalten, die aus dem Datenzeichen X oder A besteht. Der Inhalt des zu prüfenden Datenfeldes wird nur dann als alphabetisch anerkannt, wenn er aus einer Kombination der Buchstaben A bis Z und/oder a bis z und Leerzeichen besteht.

Beim NUMERIC-Test (Test eines numerischen Operanden) darf der Identifier kein alphabetisches Datenfeld mit einer lediglich aus den Symbolen A und B bestehenden PICTURE-Zeichenfolge sein. Der Identifier darf auch keine Datengruppe sein, die aus Datenbeschreibungen und ein Vorzeichen enthaltenden Datenelementen besteht. Der Identifier muß eine PICTURE-Zeichenfolge enthalten, die aus dem Datenzeichen X oder 9 und der wahlweisen Klausel USAGE IS COMPUTATIONAL oder DISPLAY besteht.

Weist der auf einen numerischen Zustand hin getestete Identifier die SIGN-Klausel nicht auf, dann gilt das Datenfeld nur dann als numerisch, wenn sein Inhalt numerisch ohne Vorzeichen ist. Weist der auf einen numerischen Zustand hin getestete Identifier die SIGN-Klausel auf, dann gilt das Datenfeld nur dann als numerisch, wenn sein Inhalt numerisch und ein gültiges Vorzeichen vorhanden ist.

Es folgen Beispiele der Klassen-Bedingung:

Beispiel 1:

```
02  TCODE   PIC AA.

    IF TCODE IS ALPHABETIC MOVE HAUPT TO HAUPT-NEU.
```

Diese Klassen-Bedingung testet jedes Zeichen im TCODE dahingehend, ob es ein Leerzeichen oder ein Buchstabe (A bis Z bzw. a bis z) ist.

Beispiel 2:

```
02  KTO-NR      PIC XXXX.

    IF KTO-NR IS NUMERIC GO TO OK.
```

Diese Klassen-Bedingung testet jedes Zeichen in KTO-NR auf die Werte 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Beispiel 3:

```
02  MENGE-1     PIC S999 DISPLAY TRAILING SEPARATE.

    IF MENGE-1 IS NUMERIC GO TO VORGANG-1.
```

Diese Klassen-Bedingung testet jedes der drei äußersten linken Zeichen in MENGE-1 auf die Werte 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 hin. Das äußerste rechte Zeichen von MENGE-1 wird auf ein gültiges Vorzeichen + oder - hin geprüft.

Beispiel 4:

```
02  MENGE-3     PIC S999 DISPLAY TRAILING.

    IF MENGE-3 IS NUMERIC GO TO GUELTIG.
```

Diese Klassen-Bedingung testet jedes der beiden äußersten linken Zeichen auf die Werte 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 hin. Das äußerst rechte Zeichen wird auf ein mit einer Ziffer kombiniertes gültiges Vorzeichen hin geprüft; die gültigen Zeichen in dieser Position sind A bis R, und .

Die Beispiele 3 und 4 gelten auch für die Angabe LEADING SEPARATE bzw. LEADING, nur daß dann das äußerste linke Zeichen auf ein gültiges Vorzeichen bzw. mit einer Ziffer kombiniertes Vorzeichen geprüft wird.

Vorzeichen-Bedingung

Die Vorzeichen-Bedingung bestimmt, ob der algebraische Wert eines numerischen Operanden oder eines arithmetischen Ausdrucks größer, kleiner oder gleich Null ist. Format für die Vorzeichen-Bedingung:

```
{Identifier,          }     IS   [NOT]     {POSITIVE}
{arithmetischer-      }                    {NEGATIVE}
{Ausdruck             }                    {ZERO    }
```

Da durch den Identifier bezeichnete Datenfeld in der Vorzeichen-Bedingung muß ein numerisches Elementarfeld sein. Der algebraische Wert dieses Datenfeldes wird auf einen Wert, der größer ist als Null (positiv), einen Wert, der kleiner ist als Null (negativ) oder einen Wert, der gleich Null ist, getestet.

Es folgen Beispiele für die Vorzeichen-Bedingung:

Beispiel 1:

```
02  BETRAG        PIC S9999V99 DISPLAY TRAILING SEPARATE.

    IF BETRAG IS NOT ZERO MOVE NEU-BETRAG TO ALT-BETRAG.
```

Diese Vorzeichen-Bedingung prüft das Datenfeld BETRAG auf einen Wert ungleich Null.

Beispiel 2:

```
02  MENGE         PIC S9999 DISPLAY TRAILING SEPARATE.

    IF MENGE IS POSITIVE GO TO BERECHNUNG.
```

Diese Vorzeichen-Bedingung testet das Datenfeld MENGE auf einen Wert größer als Null (positiv).

Beispiel 3:

```
02  SALDO         PIC S9(6)V99 DISPLAY TRAILING SEPARATE.

    IF SALDO IS NEGATIVE GO TO UEBERZOGEN.
```

Diese Vorzeichen-Bedingung testet das Datenfeld SALDO auf einen Wert kleiner als Null (negativ).

Schalter-Zustands-Bedingung

Diese Bedingung testet den An- oder Aus-Zustand eines System-Software-Schalters. Der Bedingungs-Name und der ihm zugeordnete An- oder Aus-Wert müssen im SPECIAL-NAMES-Paragraphen der ENVIRONMENT DIVISION benannt werden.

Der Vergleich ist erfüllt, wenn der Schalter den dem Bedingungs-Namen entsprechenden Wert innehat. In Beispiel 1 ist dem Bedingungs-Namen DRUCKEN im SPECIAL-NAMES-Paragraphen der ENVIRONMENT DIVISION der An-Zustand des Schalters 3 zugeordnet. Der An-Zustand vom Schalter 3 wird dann in der PROCEDURE DIVISION mit einer IF-Anweisung getestet. Die Bedingung ist erfüllt, wenn Schalter 3 im An-Zustand (ON) ist; die Bedingung ist nicht erfüllt, wenn Schalter 3 im Aus-Zustand (OFF) ist.

Beispiel:

```
SPECIAL-NAMES.
    SWITCH-3 ON STATUS IS DRUCKEN.

PROCEDURE DIVISION.
    IF DRUCKEN GO TO DRUCK ELSE GO TO
                    KEIN-DRUCK.
```

Ist Schalter 3 im An-Zustand, dann führt die IF-Anweisung zur Ausführung der im Prozedur-Namen DRUCK festgelegten Prozeduren. Ist Schalter 3 im Aus-Zustand, dann führt die IF-Anweisung zur Ausführung der im Prozedur-Namen KEIN-DRUCK festgelegten Prozeduren.

Bedingungsnamen-Bedingung

Ein Bedingungsname ist ein Benutzer-Wort, das einem Wert, mehreren Werten oder einem Bereich von Werten zugeordnet wird.

In der DATA DIVISION folgt die Festlegung der Bedingungsnamen unmittelbar auf die Definition der Bedingungsvariablen.

Bedingungsnamen werden festgelegt in der Stufen-Nr. 88.

Die Bedingungsnamen-Bedingung frägt den Inhalt der Bedingungsvariablen dahingehend ab, ob ein Wert enthalten ist, der dem Bedingungsnamen zugeordnet wurde.

Der Vergleich gilt als erfüllt, wenn der Inhalt der Bedingungsvariablen gleich einem Wert ist, der dem Bedingungsnamen zugeordnet wurde; andernfalls ist die Bedingung nicht erfüllt.

Im folgenden Beispiel kann das Feld SATZART einen dieser Werte enthalten, um die Bewegungsart auszuweisen:

Wert von SATZART	Bewegungsart
1	NEUANLAGE
2	LOESCHUNG
3	LOESCHUNG
4	LOESCHUNG
5	FELD-AENDERUNG
7	FELD-AENDERUNG
9	FELD-AENDERUNG
10	FELD-AENDERUNG
11	FELD-AENDERUNG
12	FELD-AENDERUNG

Die Definition des BEWEGUNG-Satzes enthält die Bedingungsvariable SATZART und die Bedingungsnamen ZUGANG, ABGANG und AENDERUNG:

```
01  BEWEGUNG.
    02  NUMMER  PIC X(5).
    02  SATZART PIC 99.
        88  ZUGANG       VALUE IS 1.
        88  ABGANG       VALUES ARE 2 THRU 4.
        88  AENDERUNG    VALUES ARE 5, 7, 9 THRU 12.
    02  SUMME   PIC 9999V99.
```

Werden die Bedingungsnamen in einer IF-Anweisung benutzt, kann das Feld SATZART auf die Inhalte 1, 2, 3, 4, 5, 7, 9, 10, 11 und 12 abgefragt werden.

Im folgenden Beispiel sind beide Möglichkeiten enthalten, mit und ohne Abfrage der Bedingungsnamen.

```
    IF SATZART = 1 GO NEUANLAGE.
    IF ZUGANG   GO NEUANLAGE.

    IF SATZART > 1 AND SATZART < 5 GO LOESCHUNG.
    IF ABGANG   GO LOESCHUNG.

    IF SATZART = 5 OR SATZART = 7 GO FELD-AENDERUNG.
    IF SATZART > 8 AND SATZART < 13 GO FELD-AENDERUNG.

    IF AENDERUNG    GO FELD-AENDERUNG.
```

Der Bedingungsname ZUGANG wird für die Verhältnis-Bedingung SATZART IS EQUAL TO 1 benutzt, um die Steuerung zum Paragraphen NEUANLAGE zu ermöglichen.

Der Bedingungsname ABGANG wird für die Verhältnis-Bedingung SATZART IS GREATER THAN 1 AND SATZART IS LESS THAN 5 benutzt, um die Steuerung zum Paragraphen LOESCHUNG zu ermöglichen.

Die beiden Verhältnis-Bedingungen

```
    IF SATZART IS EQUAL TO 5 OR
       SATZART IS EQUAL TO 7
```

und

```
    IF SATZART IS GREATER THAN 8 AND
       SATZART IS LESS THAN 13
```

erreichen dieselbe Steuerung zum Paragraphen FELD-AENDERUNG wie die Abfrage nach dem Bedingungsnamen AENDERUNG.

Verneinte einfache Bedingung

Eine einfache Bedingung wird verneint, wenn der logische Operator NOT der einfachen Bedingung vorausgestellt wird. Dadurch wird das entgegengesetzte Ergebnis der Bedingung erreicht.

Beispiel 1 zeigt eine verneinte Wahlschalterbedingung. Bei den SPECIAL-NAMES wird dem Wahlschalter 4, falls gesetzt, der Bedingungsname DRUCKER zugeordnet.

Beispiel 1:

```
SPECIAL-NAMES.
    SWITCH-4 ON STATUS IS DRUCKER.

PROCEDURE DIVISION.
    IF DRUCKER GO TO DRUCKAUSGABE
        ELSE GO TO PLATTENAUSGABE.

    IF NOT DRUCKER GO TO PLATTENAUSGABE
        ELSE GO TO DRUCKAUSGABE.
```

Die erste IF-Anweisung zeigt eine einfache Bedingungsabfrage.

- Wenn die Bedingung erfüllt ist (Wahlschalter 4 ist an), wird zu DRUCKAUSGABE verzweigt.

- Wenn die Bedingung nicht erfüllt ist (Wahlschalter 4 ist aus), wird zu PLATTENAUSGABE verzweigt.

Die zweite IF-Anweisung von Beispiel 1 zeigt eine verneinte einfache Bedingung:

- Wenn die verneinte Bedingung erfüllt ist (Wahlschalter 4 ist aus), wird zu PLATTENAUSGABE verzweigt.

- Wenn die verneinte Bedingung nicht erfüllt ist (Wahlschalter 4 ist an), wird zu DRUCKAUSGABE verzweigt.

Eine verneinte Bedingung ist nur dann erfüllt, wenn die Bedingung nicht erfüllt ist.
Eine verneinte Bedingung ist nur dann nicht erfüllt, wenn die Bedingung erfüllt ist.
Es ändert sich nichts an den Ausgängen Ja und Nein, wenn die verneinte Bedingung in Klammern steht.

Mehrfach-Bedingung

Eine Mehrfach-Bedingung besteht aus zwei oder mehr Bedingungen, die durch die logischen Operatoren AND und/oder OR (auch Konnektoren genannt) verbunden sind. Format:

Bedingung-1 $\left\{\begin{matrix}\underline{\text{AND}}\\ \underline{\text{OR}}\end{matrix}\right\}$ Bedingung-2 ...

Unter dem Wort "Bedingung" im Format ist zu verstehen:
- eine einfache Bedingung
- eine verneinte einfache Bedingung
- eine Mehrfach-Bedingung
- eine verneinte Mehrfach-Bedingung, wobei dem logischen Konnektor NOT eine Mehrfach-Bedingung in Klammer folgt
- eine Kombination der vorgenannten Bedingungen

Klammern können verwendet werden, um die Reihenfolge der Auflösung bei einer die logischen Operatoren AND, OR und NOT enthaltenden Mehrfach-Bedingung festzulegen. Die folgende Tabelle zeigt die Möglichkeiten auf, unter denen Bedingungen und logische Operatoren kombiniert und in Klammern gesetzt werden können. Die Klammern müssen immer paarweise verwendet werden, so daß für jede linke Klammer eine entsprechende rechte Klammer vergeben wird.

zu prüfendes Element	Position in der verknüpften Bedingung		In der Reihenfolge von links nach rechts	
	vorangehend	nachfolgend	dem Element darf, falls es nicht das erste ist, unmittelbar vorangehen	dem Element darf, falls es nicht das letzte ist, unmittelbar nachfolgen
einfache Bedingung	Ja	Ja	AND, OR, NOT, (	AND, OR,)
AND, OR	Nein	Nein	einfache Bedingung,)	einfache Bedingung, NOT, (
NOT	Ja	Nein	AND, OR, (	einfache Bedingung, (
(	Ja	Nein	AND, OR, NOT, (	einfache Bedingung, NOT, (
)	Nein	Ja	einfache Bedingung,)	AND, OR,)

Klammern können verwendet werden, um die Reihenfolge festzulegen, in der einzelne Bedingungen in der Mehrfach-Bedingung abzufragen sind. In Klammern stehende Bedingungen werden zuerst abgefragt. Im Falle einer Verschachtelung von eingeklammerten Bedingungen verläuft die Auflösung von der innersten zur äußersten Klammer. Werden Klammern nicht verwendet oder sind die in Klammern stehenden Bedingungen gleichrangig, dann gilt die folgende hierarchische Reihenfolge in der Abfrage:

1. Erfüllte einfache Bedingungen in der Reihenfolge:

   - Vergleichs-Bedingungen
   - Klassen-Bedingungen
   - Bedingungsnamen-Bedingungen
   - Schalter-Zustands-Bedingungen
   - Vorzeichen-Bedingungen

2. Erfüllte verneinte einfache Bedingungen

3. Erfüllte Mehrfach-Bedingungen in der Reihenfolge:

   - Logische Konnektoren AND
   - Logische Konnektoren OR

4. Erfüllte verneinte Mehrfach-Bedingungen

Ist die Reihenfolge der Abfrage nicht voll durch Klammern festgelegt, so werden aufeinanderfolgende, der gleichen hierarchischen Stufe zugehörende Operationen innerhalb der Mehrfach-Bedingungen von links nach rechts abgefragt. Aus der folgenden Tabelle ist ersichtlich, in welchem Fall eine Mehrfach-Bedingung mit den logischen Konnektoren AND, OR und NOT als erfüllt gelten kann.

Wenn Bedingung A =	Wenn Bedingung B =
wahr	wahr
wahr	falsch
falsch	wahr
falsch	falsch

dann ist A AND B

wahr
falsch
falsch
falsch

dann ist A OR B

wahr
wahr
wahr
falsch

dann ist NOT A

falsch
falsch
wahr
wahr

Beispiel 1:

Dieses Beispiel zeigt eine Mehrfach-Bedingung, die aus zwei durch den logischen Konnektor AND verbundenen Vergleichs-Bedingungen besteht. Eine Mehrfach-Bedingung, in der nur der logische Konnektor AND vorgesehen ist, ist dann erfüllt, wenn jede der Bedingungen erfüllt ist.

```
IF COD  IS EQUAL TO 50 AND BETRAG IS GREATER THAN 100
        ADD BETRAG TO SUMME ELSE SUBTRACT BETRAG FROM
        SUMME.
MOVE SALDO TO SALDO-NEU.
```

Beispiel 2:

Dieses Beispiel zeigt eine Mehrfach-Bedingung, die aus zwei durch den logischen Konnektor OR verbundenen Vergleichs-Bedingungen besteht. Eine Mehrfach-Bedingung, in der nur der logische Konnektor OR vorgesehen ist, ist dann erfüllt, wenn zumindest eine der Bedingungen erfüllt ist.

```
IF COD  IS EQUAL TO 70 OR BETRAG IS LESS THAN 500
        ADD BETRAG TO ARB-SUMME
        ELSE SUBTRACT BETRAG FROM ARB-SUMME.
MULTIPLY PREIS BY 2 GIVING NEU-PREIS.
```

Beispiel 3:

Dieses Beispiel zeigt eine Mehrfach-Bedingung, die aus drei durch die logischen Konnektoren AND und OR verbundenen Vergleichs-Bedingungen besteht. Bei einer Mehrfach-Bedingung, die die logischen Konnektoren AND und OR aufweist, werden zuerst die mit dem logischen Konnektor AND verbundenen Bedingungen und dann die mit dem logischen Konnektor OR verbundenen Bedingungen abgefragt.

```
IF COD  = 9 AND BETRAG IS GREATER THAN 10 OR SALDO IS
     LESS THAN ZERO ADD 700 TO UEBERSCHUSS ELSE
     SUBTRACT 300 FROM FAELL-BETRAG.
MULTIPLY SALDO BY 10 GIVING SALDO-NEU.
```

Die Addition findet nur statt, wenn entweder der Saldo kleiner als Null ist oder wenn COD den Inhalt 9 aufweist und zugleich der Betrag größer als 10 ist.

Beispiel 4:

Dieses Beispiel zeigt eine Mehrfach-Bedingung, die aus drei durch die logischen Konnektoren AND und OR verbundenen Vergleichs-Bedingungen besteht. Bei einer Mehrfach-Bedingung, die die logischen Konnektoren AND und OR aufweist, werden zuerst die mit dem logischen Konnektor AND verbundenen Bedingungen abgefragt. Dieses Beispiel zeigt, welche Auswirkungen es auf die Logik hat, wenn das OR in Beispiel 3 in ein AND und das AND dort in ein OR abgeändert wird.

```
IF COD  = 9 OR BETRAG IS GREATER THAN 10 AND SALDO IS
    LESS THAN ZERO ADD 700 TO UEBERSCHUSS ELSE
    SUBTRACT 300 FROM FAELL-BETRAG.
MULTIPLY SALDO BY 10 GIVING SALDO-NEU.
```

Die Addition findet nur statt, wenn entweder COD den Inhalt 9 aufweist oder wenn der Betrag größer als 10 und zugleich der Saldo kleiner als Null ist.

Beispiel 5:

Dieses Beispiel zeigt eine Mehrfach-Bedingung, die aus drei durch die logischen Konnektoren AND und OR verbundenen Vergleichs-Bedingungen besteht. In dieser Mehrfach-Bedingung ist die Reihenfolge der Abfrage durch Klammern bestimmt. Die in Klammern stehenden Bedingungen werden zuerst abgefragt.

```
IF COD  = 9 AND (BETRAG IS GREATER THAN 10 OR SALDO
    IS LESS THAN ZERO) ADD 700 TO UEBERSCHUSS ELSE
    SUBTRACT 300 FROM FAELL-BETRAG.
MULTIPLY SALDO BY 10 GIVING SALDO-NEU.
```

Die Addition findet nur dann statt, wenn

a) COD = 9 und der Betrag > 10 ist
   oder wenn
b) COD = 9 und der Saldo < 0 ist.

Beispiel 6:

Dieses Beispiel zeigt eine Mehrfach-Bedingung, bestehend aus einer einfachen verneinten Bedingung und einer einfachen Verhältnis-Bedingung, die durch den logischen Konnektor AND verbunden ist. Die Bedingung ist erfüllt, wenn SWITCH-6 aus und TOTAL-A größer als 100 ist.

```
SPECIAL-NAMES.
    SWITCH-6 ON STATUS IS SW-6-AN.

PROCEDURE DIVISION.
    IF NOT SW-6-AN AND TOTAL-A IS GREATER THAN 100
    GO TO R1. ADD TOTAL-A TO GTOTAL-A.
```

Beispiel 7:

Dieses Beispiel zeigt eine Mehrfach-Bedingung, bestehend aus einer verneinten Mehrfach-Bedingung, die dem logischen Konnektor NOT folgt und in Klammern eingeschlossen ist. Die Gesamt-Bedingung ist erfüllt, wenn die Mehrfach-Bedingung in der Klammer nicht erfüllt ist.

```
IF NOT (COD IS EQUAL TO 50 AND BETRAG IS GREATER
    THAN 100) ADD BETRAG TO SUMME ELSE SUBTRACT
    BETRAG FROM SUMME.
MOVE SALDO TO SALDO-NEU.
```

Addiert wird also nur, wenn die innerhalb der Klammern gemachten Angaben unzutreffend sind.

Verschachtelter Bedingungs-Satz

Ein verschachtelter Bedingungs-Satz ist eine IF-Anweisung mit zusätzlichen IF-Anweisungen in entweder dem Ja-Zweig, dem Nein-Zweig oder beiden. IF-Anweisungen in IF-Anweisungen sind paarweise, von links nach rechts fortschreitende IF- und ELSE-Kombinationen. Somit gehört jedes ELSE dem unmittelbar vorangehenden IF zu, das nicht schon sein ELSE hat. Das nachfolgende Beispiel zeigt zwei verschachtelte IF-Anweisungen. Eine IF- und ELSE-Kombination ist durch dieselbe, jeweils unter IF und ELSE gesetzte, in einem Kreis dargestellte, Nummer angezeigt.

Beispiel:

```
IF KTO-NR = MKTO-NR IF COD  = 15 ADD BETRAG1 TO SALDO
 2                   1
        ELSE SUBTRACT BETRAG1 FROM SALDO
        1
        ELSE MOVE "B" TO CODE-B.
        2
ADD FAELL-BETRAG TO MAHN-BETR.
```

Sind die Inhalte der Felder KTO-NR und MKTO-NR einander ungleich, wird der Buchstabe B in das Feld CODE-B übertragen und anschließend FAELL-BETRAG zu MAHN-BETR addiert.

Sind die Inhalte der Felder KTO-NR und MKTO-NR einander gleich, kommt es darauf an, ob COD den Inhalt 15 aufweist. Weist COD den Inhalt 15 auf, wird BETRAG1 auf SALDO addiert. Weist COD einen anderen Inhalt als 15 auf, wird BETRAG1 von SALDO subtrahiert. Anschließend wird in beiden Fällen FAELL-BETRAG zu MAHN-BETR addiert, usw.

Die INITIALIZE-Anweisung

Diese Anweisung wird verwendet, um ausgewählten Datenfeldern Inhalte zu geben, wie zum Beispiel numerischen Feldern den Inhalt Null, alphanumerischen Feldern Spaces, usw.

Format:

```
INITIALIZE  Identifier-1 ...
[ REPLACING { [ALPHABETIC              ] }  DATA BY { [Identifier-2] }... ]
            { [ALPHANUMERIC            ] }          { [Literal-1   ] }
            { [NUMERIC                 ] }
            { [ALPHANUMERIC-EDITED     ] }
            { [NUMERIC-EDITED          ] }
```

Syntax:

- Identifier-2 und Literal-2 repräsentieren den zu vergebenden Wert. Identifier-1 repräsentiert das Feld, das den Wert als Inhalt bekommen soll.
- Die unter REPLACING angegebene Kategorie muß den Regeln der MOVE-Anweisung in bezug auf den zu übertragenden Feldinhalt aus Identifier-2 oder Literal-1 entsprechen.
- Die gleiche Kategorie kann in einer REPLACING-Klausel (innerhalb der gleichen INITIALIZE-Anweisung) nicht wiederholt werden.
- In der Datendefinition von Identifier-1 oder Feldern, die Identifier-1 untergeordnet sind, darf keine DEPENDING-Angabe der OCCURS-Klausel vorkommen.
- Index-Datenfelder (USAGE IS INDEX) dürfen der INITIALIZE-Anweisung nicht unterzogen werden.
- Die Datendefinition für Identifier-1 darf keine RENAMES-Klausel aufweisen.

Wirkung des Befehls:

- Das auf das Wort REPLACING folgende Schlüsselwort korrespondiert mit der COBOL-Datenkategorie.

- Egal, ob Identifier-1 ein Elementarfeld oder eine Gruppe von Feldern darstellt, werden alle Operationen ausgeführt wie eine Serie von MOVE-Anweisungen unter Beachtung der dort gültigen Regeln.

- Wird REPLACING verwendet, gelten folgende Regeln:

  - Stellt Identifier-1 eine Feldgruppe dar, werden nur solche Elementarfelder der Initialisierung unterzogen, die der unter REPLACING angegebenen Kategorie zugehören.

  - Stellt Identifier-1 ein Elementarfeld dar, wird es nur initialisiert, wenn es der unter REPLACING angegebenen Kategorie angehört.

  - Index-Datenfelder und elementare FILLER-Felder bleiben von der INITIALIZE-Anweisung unberührt.

  - Jedes Feld, das hierarchisch einem Identifier des Empfangsbereiches angehört und die REDEFINES-Klausel aufweist, oder jedes Feld, das wiederum einem solchen Feld hierarchisch untersteht, wird von der Initialisierung ausgeschlossen. Jedoch darf ein Identifier des Empfangsbereiches selbst eine REDEFINES-Klausel enthalten oder einem Feld mit einer REDEFINES-Klausel untergeordnet sein.

  - Wird im Befehl die REPLACING-Klausel nicht verwendet, werden alphabetische, alphanumerische und alphanumerisch editierte Felder mit Spaces (Blanks) gefüllt. Boolesche, numerische und numerisch editierte Felder werden mit Nullen gefüllt.

Ein Beispiel finden Sie auf der folgenden Seite.

Beispiel:

```
01  SATZ.
    05  AFELD  PIC X(5).       Inhalt: "ABCDE"
    05  NGRUPPE.
      10  NFLD1  PIC 9.         Inhalt: 7
      10  NFLD2  PIC 999.       Inhalt: 537
```

Nach Durchführung der Anweisung

INITIALIZE SATZ

haben die Felder folgende neuen Inhalte:

```
AFELD:     5 Spaces (Blanks)
NFLD1:     0
NFLD2:     000
```

DIE INSPECT-ANWEISUNG (TALLYING-Variante)

Die erste Variante der INSPECT-Anweisung zählt das jeweilige Auftreten eines einzelnen Zeichens oder von Zeichengruppen in einem Datenfeld.

Format:

```
INSPECT Identifier-1 TALLYING  {Identifier-2}
FOR {{ALL       } {Identifier-3} [{BEFORE} INITIAL {Identifier-4} ...]}.
    {{LEADING   } {Literal-1   } [{AFTER }         {Literal-2   }    ]}
    {{CHARACTERS}                }
```

Die TALLYING-Variante der INSPECT-Anweisung führt ein einzelnes Zeichen oder eine Zeichengruppe betreffende Prüfung des im Identifier-1 bezeichneten Datenfeldes aus. Während der Prüfung des Identifiers-1 wird jedes dort ermittelte Auftreten der Daten gemäß Literal-1 bzw. Identifier-3 im Identifier-2 gezählt. Die Ausführung der INSPECT TALLYING-Anweisung schließt eine Abgrenzung durch die BEFORE INITIAL- oder AFTER INITIAL-Klausel, ferner den Vergleichszyklus und den Zählvorgang (TALLYING Process) ein.

Identifier-1 bezeichnet das Datenfeld, dessen Inhalt geprüft werden soll, um das Auftreten von einzelnen Zeichen oder von Zeichengruppen zu ermitteln. Identifier-1 muß entweder eine Datengruppe oder ein Elementarfeld bezeichnen, das direkt oder indirekt mit der Klausel USAGE IS DISPLAY beschrieben ist.

Identifier-2 bezeichnet das Datenfeld, dessen Inhalt für jedes Zeichen oder jede Zeichengruppe im Identifier-1 um 1 erhöht wird, welche(s) den Bedingungen der Prüfoperation entspricht. Identifier-2 muß ein numerisches Elementarfeld sein. Der Programmierer muß im Identifier-2 eine Null vorsehen, wenn die im Identifier-2 enthaltene Zählung beim Start jeder Ausführung der INSPECT-Anweisung bei Null beginnen soll.

Identifier-3 bzw. Literal-1 bezeichnet das oder die Zeichen, für die der Vergleich durch die INSPECT-Anweisung stattfindet. Identifier-3 muß ein alphabetisches, alphanumerisches oder numerisches Elementarfeld bezeichnen, das direkt oder indirekt mit der Klausel USAGE IS DISPLAY beschrieben ist. Literal-1 muß ein nicht-numerisches Literal sein. Literal-1 kann jede figurative Konstante außer dem Literal ALL sein. Ist Literal-1 eine figurative Konstante, dann bezeichnet diese ein implizites 1-Byte-Datenfeld.

Identifier-4 bzw. Literal-2 bezeichnet das oder die Zeichen zur Abgrenzung für die BEFORE INITIAL- oder AFTER INITIAL-Klausel in der INSPECT-Anweisung. Identifier-4 muß ein alphabetisches, alphanumerisches oder numerisches Elementarfeld bezeichnen, das direkt oder indirekt mit der Klausel USAGE IS DISPLAY beschrieben ist. Literal-2 muß ein nicht-numerisches Literal sein. Literal-2 kann jede figurative Konstante außer dem Literal ALL sein. Ist Literal-2 eine figurative Konstante, dann bezeichnet diese ein implizites 1-Zeichen-Datenfeld.

Ist Identifier-1, Identifier-3 oder Identifier-4 als ein alphanumerisches Datenfeld beschrieben, dann wird durch die INSPECT-Anweisung der Inhalt dieses Datenfeldes als eine Zeichenfolge betrachtet. Ist Identifier-1, Identifier-3 oder Identifier-4 als ein editiertes alphanumerisches, ein editiertes numerisches oder ein numerisches Datenfeld ohne Vorzeichen beschrieben, dann wird durch die INSPECT-Anweisung der Inhalt des numerischen Datenfeldes mit Vorzeichen in ein zu diesem Zwecke behelfsweise gebildetes numerisches Datenfeld ohne Vorzeichen übertragen; dieses wird dann von der INSPECT-Anweisung als ein alphanumerisches Datenfeld betrachtet.

INSPECT Vergleichszyklus

Der Vergleichszyklus der Anweisung INSPECT ermittelt die Häufigkeit des Auftretens von Identifier-3 bzw. Literal-1 und zählt dieses im Identifier-2. Der Vergleichszyklus beginnt mit der äußersten linken Zeichenposition des Identifiers-1 und schreitet von links nach rechts zur äußersten rechten Zeichenposition des Identifiers-1 fort. Die äußerste linke Abgrenzung der Prüfoperation kann durch die AFTER INITIAL-Klausel geändert werden. Die äußerste rechte Abgrenzung der Prüfoperation kann durch BEFORE INITIAL-Klausel geändert werden. Weist eine INSPECT-Anweisung weder die AFTER INITIAL-Klausel noch die BEFORE INITIAL-Klausel auf, dann gilt die Prüfoperation für alle Zeichen im Identifier-1. Die innerhalb der Abgrenzungen befindlichen Daten werden nachfolgend mit Prüffeld bezeichnet.

Die Operanden in der TALLYING-Angabe werden in der Reihenfolge berücksichtigt, in der sie in der INSPECT-Anweisung von links nach rechts aufgeführt sind.

Das erste Literal-1 bzw. der erste Identifier-3 wird mit einer gleichen Anzahl aufeinanderfolgender Zeichen verglichen, und zwar beginnend mit der äußersten linken Zeichenposition im Prüffeld. Literal-1 bzw. Identifier-3 gelten mit dem oder den Zeichen im Prüffeld nur dann als identisch, wenn auch in einer Zeichengruppe die einzelnen Zeichen jeweils übereinstimmen mit denen einer entsprechenden Zeichengruppe im zu prüfenden Feld. Sobald eine Übereinstimmung auftritt, findet der Zählvorgang statt. Die Zeichenposition im Prüffeld, die unmittelbar rechts von der äußersten rechten Zeichenposition der gefundenen Zeichenfolge liegt, gilt nun als die äußerste linke Zeichenposition des Prüffeldes und der Vergleichszyklus beginnt erneut mit dem ersten Literal-1 bzw. Identifier-3.

Wird keine Übereinstimmung mit dem ersten Literal-1 bzw. Identifier-3 festgestellt, so wird der Vergleichszyklus mit jedem vorliegenden nachfolgenden Literal-1 bzw. Identifier-3 wiederholt, bis es zu einer Übereinstimmung kommt oder bis kein nachfolgendes Literal-1 bzw. kein nachfolgendes Identifier-3 mehr vorliegt. Ohne nachfolgendes Literal-1 bzw. nachfolgenden Identifier-3 gilt die Zeichenposition im Prüffeld, die unmittelbar rechts von der äußersten linken, im letzten Vergleichszyklus berücksichtigten Zeichenposition liegt, als die äußerste linke Zeichenposition und der Vergleichszyklus beginnt erneut mit dem ersten Literal-1 bzw. Identifier-3.

Der Vergleichszyklus wird fortgesetzt, bis die äußerste rechte Zeichenposition des Prüffeldes verglichen worden ist oder als äußerste linke Zeichenposition gilt. Ist dies der Fall, dann ist die INSPECT-Anweisung beendet und die nächste Anweisung wird ausgeführt.

CHARACTERS

Ist die CHARACTERS-Klausel definiert, so führt ein 1-Zeichen Operand die Prüfung für alle Zeichen im Prüffeld durch. Dieses Zeichen läßt sich mit jedem Zeichen im Prüffeld vergleichen. Somit erfolgt auch für jedes der im laufenden Vergleichszyklus verglichenen Zeichen im Prüffeld eine Zählung.

Zählvorgang (Tallying Process)

Ist ALL angegeben, dann wird der Inhalt von Identifier-2 um 1 erhöht bei jedem Auftreten von Daten gemäß Literal-1 oder Identifier-3 im Prüffeld.

Ist LEADING angegeben, dann wird der Inhalt von Identifier-2 um 1 erhöht bei jedem fortlaufenden Auftreten von Daten gemäß Literal-1 bzw. Identifier-3 im Prüffeld. Das erste gefundene Zeichen muß an der Stelle stehen, wo der Vergleich mit dem Prüffeld begann.

Ist CHARACTERS angegeben, dann wird der Inhalt von Identifier-2 für jedes beliebige Zeichen im Prüffeld um 1 erhöht. Hier wird praktisch die Länge eines Feldes geprüft.

BEFORE INITIAL

Die Klausel BEFORE INITIAL verändert die äußerste rechte Abgrenzung der Prüfoperation. Ist BEFORE INITIAL angegeben, dann wird der Vergleichszyklus von der äußersten linken Zeichenposition des Identifier-1 an bis zum ersten Auftreten der Daten gemäß Identifier-4 bzw. Literal-2 ausgeführt. Die Position dieses ersten Auftretens der Daten gemäß Identifier-4 bzw. Literal-2 wird in INSPECT festgestellt, bevor der erste Vergleich ausgeführt wird.

Treten im Identifier-1 keine Daten gemäß Literal-2 bzw. Identifier-4 auf, dann werden alle Zeichen im Identifier-1 geprüft.

AFTER INITIAL

Die Klausel AFTER INITIAL verändert die äußerste linke Abgrenzung der Prüfoperation. Ist AFTER INITIAL angegeben, dann wird der Vergleichszyklus beginnend mit der Zeichenposition unmittelbar rechts von den erstmals auftretenden Daten gemäß Literal-2 bzw. Identifier-4 durchgeführt. Der Vergleich wird mit der äußersten rechten Zeichenposition von Identifier-1 beendet. Die Position des ersten Auftretens der Daten gemäß Identifier-4 bzw. Literal-2 wird von INSPECT festgestellt, bevor der erste Vergleichszyklus ausgeführt wird.

Treten im Identifier-1 keine Daten gemäß Literal-2 bzw. Identifier-4 auf, dann werden die Zeichen im Identifier-1 nicht geprüft.

Beispiel 1:

```
MOVE ZERO TO ZAHL-1.
INSPECT FELD-1 TALLYING ZAHL-1 FOR ALL "0".
```

Daten in FELD-1 vor dem INSPECT-Vorgang	Inhalt des Zählers ZAHL-1 nach dem INSPECT-Vorgang
0 0 3 4 5	0 0 2
0 0 1 0 5 7	0 0 3
1 0 2 0 3	0 0 2
0 0 0 0 0 0 0 0	0 0 8
1 9 3 0 2	0 0 1

Beispiel 2:

```
MOVE ZERO TO ZAHL-2.
INSPECT FELD-2 TALLYING ZAHL-2 FOR LEADING SPACE.
```

Daten in FELD-2 vor dem INSPECT-Vorgang	Inhalt des Zählers ZAHL-2 nach dem INSPECT-Vorgang
∅ ∅ ∅ 4 5	0 3
∅ A B	0 1
C D ∅ ∅ ∅	0 0
D ∅ E ∅	0 0
∅ ∅ D W ∅	0 2

Beispiel 3:

```
MOVE ZERO TO ZAHL-3.
INSPECT FELD-3 TALLYING ZAHL-3 FOR CHARACTERS.
```

Daten in FELD-3 vor dem INSPECT-Vorgang	Inhalt des Zählers ZAHL-3 nach dem INSPECT-Vorgang
1 4 5 0 2	0 5
A B	0 2
2 H U ⌀	0 4

Beispiel 4:

```
MOVE ZERO TO ZAHL-4.
INSPECT FELD-4 TALLYING ZAHL-4 FOR CHARACTERS BEFORE
    INITIAL "2".
```

Daten in FELD-4 vor dem INSPECT-Vorgang	Inhalt des Zählers ZAHL-4 nach dem INSPECT-Vorgang
1 3 2 0 5	0 2
⌀ ⌀ A 2 D 2	0 3
D K G A A	0 5
4 9 1 2 2 2	0 3

Beispiel 5:

```
MOVE ZERO TO ZAHL-5.
INSPECT FELD-5 TALLYING ZAHL-5 FOR ALL "A" AFTER
    INITIAL "C".
```

Daten in FELD-5 vor dem INSPECT-Vorgang	Inhalt des Zählers ZAHL-5 nach dem INSPECT-Vorgang
A B C A A	2
C A C A C A	3
B H K Q T	0
⌀ ⌀ ⌀ D C P A	1

Beispiel 6:

```
MOVE ZERO TO ZAHL-6.
INSPECT FELD-6 TALLYING ZAHL-6 FOR ALL "ABC"
```

Daten in FELD-6 vor dem INSPECT-Vorgang	Inhalt des Zählers ZAHL-6 nach dem INSPECT-Vorgang
A B C B C D E F	0 0 1
W Y Z C B A B C A	0 0 1
A B C A B C A B C	0 0 3
C B A B C B A B C	0 0 2
A A B A C B A	0 0 0

Beispiel 7:

```
MOVE ZERO TO ZAHL-7.
INSPECT FELD-7 TALLYING ZAHL-7 FOR LEADING "14".
```

Daten in FELD-7 vor dem INSPECT-Vorgang	Inhalt des Zählers ZAHL-7 nach dem INSPECT-Vorgang
1 4 1 4 4 1 4	0 2
4 1 0 1 4 0 1 4	0 0
1 4 0 0 3 1	0 1
0 1 4 1 4 1 4	0 0
1 4 1 4 1 4 1 4	0 4

Beispiel 8:

```
MOVE ZERO TO ZAEHLER-R, ZAEHLER-M.
INSPECT FELD-8 TALLYING ZAEHLER-R FOR LEADING "R" BEFORE
    INITIAL "M", ZAEHLER-M FOR ALL "M".
```

Daten in FELD-8 vor dem INSPECT-Vorgang	Inhalt von ZÄHLER-R nach dem INSPECT-Vorgang	Inhalt von ZÄHLER-M nach dem INSPECT-Vorgang
R M R M S V	0 1	0 2
R R M M A B	0 2	0 2
M M M R J	0 0	0 3
R L R M X X	0 1	0 1

Beispiel 9:

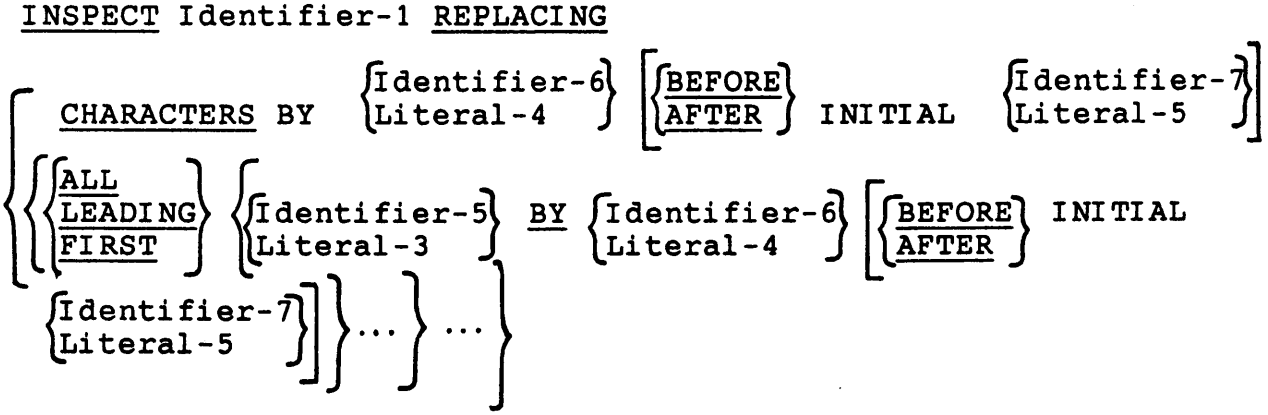
```
MOVE ZERO TO ZAHL-9.
INSPECT FELD-9 TALLYING ZAHL-9 FOR ALL H"10".
```

Daten in FELD-9 vor dem INSPECT-Vorgang	Inhalt des Zählers ZAHL-9 nach dem INSPECT-Vorgang
10 1F 10 1D OF	2
7F 7F 10 11 14 00	1
10 10 10	3
77 7A 7B 7D	0
01 10 01 1E	1

Die INSPECT-Anweisung (REPLACING-Variante)

Die zweite Variante der INSPECT-Anweisung ersetzt ein einzelnes Zeichen oder Zeichengruppe in einem Datenfeld.

Format:



Die REPLACING-Variante der INSPECT-Anweisung führt eine Prüfung des im Identifier-1 bezeichneten Datenfeldes nach Einzelzeichen oder Zeichengruppen aus. Während der Prüfung des Identifiers-1 werden alle Daten gemäß Literal-3 bzw. Identifier-5 durch Zeichen im Literal-4 bzw. Identifier-6 ersetzt. Die Ausführung der INSPECT REPLACING-Anweisung schließt die Abgrenzungen durch die BEFORE INITIAL- oder AFTER INITIAL-Klausel sowie den Vergleichszyklus und das Austauschen (REPLACING Process) ein.

Identifier-1 bezeichnet das Datenfeld, dessen Inhalt geprüft werden soll, um das Auftreten von einzelnen Zeichen oder Zeichengruppen zu ermitteln. Identifier-1 muß entweder eine Datengruppe oder ein Elementarfeld bezeichnen, das indirekt oder ausdrücklich mit der Klausel USAGE IS DISPLAY beschrieben ist.

Identifier-5 bzw. Literal-3 bezeichnet das oder die Zeichen, für die der Vergleich durch die INSPECT-Anweisung stattfindet. Identifier-5 muß ein alphabetisches, alphanumerisches oder numerisches Elementarfeld bezeichnen, das indirekt oder ausdrücklich mit der Klausel USAGE IS DISPLAY beschrieben ist. Literal-3 muß ein nicht-numerisches Literal sein. Literal-3 kann jede figurative Konstante außer dem Literal ALL sein. Ist Literal-3 eine figurative Konstante, dann kommt diese einem 1-Zeichen-Datenfeld gleich.

Identifier-6 bzw. Literal-4 bezeichnet das oder die Zeichen, welches bzw. welche jedes durch Identifier-5 bzw. Literal-3 bezeichnete Auftreten eines oder mehrerer Zeichen ersetzt. Identifier-6 muß ein alphabetisches, alphanumerisches oder numerisches Elementarfeld sein, das indirekt oder ausdrücklich mit der Klausel USAGE IS DISPLAY beschrieben ist. Literal-4 muß ein nicht-numerisches Literal sein. Literal-4 kann jede figurative Konstante außer dem Literal ALL sein. Die Länge von Identifier-6 bzw. Literal-4 muß der von Identifier-5 bzw. Literal-3 entsprechen. Ist Literal-3 eine figurative Konstante, dann muß Identifier-6 bzw. Literal-4 in der Länge ein Zeichen umfassen. Ist Literal-4 eine figurative Konstante, dann entspricht diese in der Länge der von Identifier-5 bzw. Literal-3. Ist die CHARACTERS-Klausel definiert, dann müssen Identifier-6 bzw. Literal-4 in der Länge 1 Zeichen (d. h. keine Zeichengruppe) umfassen.

Identifier-7 bzw. Literal-5 bezeichnet das oder die Zeichen, mit denen die Abgrenzung für die BEFORE INITIAL- oder AFTER INITIAL-Klausel in der INSPECT-Anweisung geschaffen wird. Identifier-7 muß ein alphabetisches, alphanumerisches oder numerisches Elementarfeld bezeichnen, das indirekt oder ausdrücklich mit der Klausel USAGE IS DISPLAY beschrieben ist. Literal-5 muß ein nicht-numerisches Literal sein. Literal-5 kann jede figurative Konstante außer dem Literal ALL sein. Ist Literal-5 eine figurative Konstante, dann bezeichnet diese ein implizites 1-Zeichen-Datenfeld. Ist die CHARACTERS-Klausel definiert, dann dürfen Identifier-7 bzw. Literal-5 nicht mehr als 1 Zeichen umfassen.

Ist Identifier-1, Identifier-5, Identifier-6 oder Identifier-7 als ein alphanumerisches Datenfeld beschrieben, wird von der INSPECT-Anweisung der Inhalt dieses Datenfeldes als Zeichenfolge betrachtet. Ist Identifier-1, Identifier-5, Identifier-6 oder Identifier-7 als ein editiertes alphanumerisches, ein editiertes numerisches oder ein numerisches Datenfeld ohne Vorzeichen beschrieben, wird von der INSPECT-Anweisung der Inhalt dieses Datenfeldes als ein alphanumerisches Datenfeld betrachtet. Ist Identifier-1, Identifier-5, Identifier-6 oder Identifier-7 als ein numerisches Datenfeld mit Vorzeichen beschrieben, dann wird durch die INSPECT-Anweisung der Inhalt des numerischen Datenfeldes mit Vorzeichen in ein für diesen Zweck vorübergehend gebildetes numerisches Datenfeld ohne Vorzeichen übertragen; das sich ergebende numerische Datenfeld ohne Vorzeichen wird dann von der INSPECT-Anweisung als ein alphanumerisches Datenfeld betrachtet.

INSPECT Vergleichszyklus

Der Vergleichszyklus der INSPECT-Anweisung ermittelt die Anzahl des Auftretens des durch Identifier-6 bzw. Literal-4 zu ersetzenden Identifiers-5 bzw. Literals-3. Der Vergleichszyklus beginnt mit der äußersten linken Zeichenposition des Identifiers-1 und schreitet von links nach rechts zur äußersten rechten Zeichenposition des Identifiers-1 fort. Die äußerste linke Abgrenzung der Prüfoperation kann durch die AFTER INITIAL-Klausel geändert werden. Die äußerste rechte Abgrenzung der Prüfoperation kann durch die BEFORE INITIAL-Klausel geändert werden. Weist eine INSPECT-Anweisung weder die AFTER INITIAL- noch BEFORE INITIAL-Klausel auf, dann gilt die Prüfoperation für den gesamten Inhalt des Identifiers-1. Die innerhalb der Abgrenzung befindlichen Daten werden nachfolgend mit Prüffeld bezeichnet.

Die Operanden in der REPLACING-Variante werden in der Reihenfolge berücksichtigt, in der sie in der INSPECT-Anweisung von links nach rechts aufgeführt sind.

Das erste Literal-3 bzw. der erste Identifier-5 wird mit einer gleichen Anzahl aufeinanderfolgender Zeichen im Prüffeld verglichen, und zwar beginnend mit der äußersten linken Zeichenposition im Prüffeld. Sobald eine Übereinstimmung vorliegt, findet der Austauschvorgang statt. Die Zeichenposition im Prüffeld, die unmittelbar rechts von der verglichenen äußersten rechten Zeichenposition liegt, gilt nun als die äußerste linke Zeichenposition des Prüffeldes und der Vergleichszyklus geht weiter mit dem ersten Literal-3 bzw. Identifier-5.

Besteht keine Übereinstimmung mit dem ersten Literal-3 bzw. Identifier-5, so wird der Vergleichszyklus mit jedem nachfolgenden Literal-3 bzw. Identifier-5 falls angegeben wiederholt, bis eine Übereinstimmung festgestellt wird oder bis kein nachfolgendes Literal-3 bzw. kein nachfolgender Identifier-5 mehr vorliegt. Ohne nachfolgendes Literal-3 bzw. nachfolgenden Identifier-5 gilt die Zeichenposition im Prüffeld, die unmittelbar rechts von der äußersten linken, im letzten Vergleichszyklus berücksichtigten Zeichenposition liegt, als die nunmehr äußerste linke Zeichenposition und der Vergleichszyklus beginnt erneut mit dem ersten Literal-3 bzw. Identifier-5.

Der Vergleichszyklus wird fortgesetzt, bis die äußerste rechte Zeichenposition des Prüffeldes verglichen worden ist oder als äußerste linke Zeichenposition gilt. Ist dies der Fall, dann ist die INSPECT-Anweisung beendet und die nächste Anweisung wird ausgeführt.

CHARACTERS

Ist die CHARACTERS-Klausel definiert, wird durchgehend zeichenweise geprüft und jedes Zeichen durch den Identifier-6 oder Literal-4 ersetzt.

Austauschvorgang (Replacing Process)

Ist ALL angegeben, dann wird jedes Zeichen gemäß Literal-3 bzw. Identifier-5 im Prüffeld durch die Zeichen im Literal-4 bzw. Identifier-6 ersetzt. ALL gilt für jede nachfolgende BY-Angabe bis zum Auftreten einer LEADING-Angabe, einer FIRST-Angabe oder einer anderen ALL-Angabe.

Ist LEADING angegeben, dann werden nur die führenden Zeichen gemäß Literal-3 bzw. Identifier-5 im Prüffeld durch die Zeichen im Literal-4 bzw. Identifier-6 ersetzt. Wo der erste Vergleichszyklus mit dem Prüffeld begann, ist das äußerste linke Vergleichszeichen. LEADING gilt für jede nachfolgende BY-Angabe bis zum Auftreten einer ALL-Angabe, einer FIRST-Angabe oder einer anderen LEADING-Angabe.

Ist FIRST angegeben, dann wird nur das erste Zeichen gemäß Literal-3 bzw. Identifier-5 im Prüffeld durch die Zeichen im Literal-4 bzw. Identifier-6 ersetzt. FIRST gilt für jede nachfolgende BY-Angabe bis zum Auftreten einer LEADING-Angabe, einer ALL-Angabe oder einer anderen FIRST-Angabe.

Ist CHARACTERS angegeben, dann wird jedes Zeichen im Prüffeld durch die Zeichen im Literal-4 bzw. Identifier-6 ersetzt.

BEFORE INITIAL

Die BEFORE INITIAL-Klausel verändert die äußerste rechte Abgrenzung der Prüfoperation. Ist BEFORE INITIAL angegeben, dann wird der Vergleichszyklus von der äußersten linken Zeichenposition des Identifiers-1 an bis zum ersten Auftreten der Daten gemäß Identifier-7 bzw. Literal-5 im Identifier-1 ausgeführt. Die Position dieses ersten Auftretens der Daten gemäß Identifier-6 bzw. Literal-5 wird von INSPECT festgestellt, bevor der erste Vergleichszyklus ausgeführt wird.

Treten im Identifier-1 keine Daten gemäß Literal-5 bzw. Identifier-7 auf, dann sind alle Zeichen im Identifier-1 vom Vergleichszyklus betroffen.

AFTER INITIAL

Die AFTER INITIAL-Klausel verändert die äußerste linke Abgrenzung der Prüfoperation. Ist AFTER INITIAL angegeben, dann wird der Vergleichszyklus durchgeführt, beginnend mit der Zeichenposition, die unmittelbar rechts von der äußersten rechten Zeichenposition der auftretenden Daten gemäß Literal bzw. Identifier-7 im Identifier-1 steht. Der Vergleichszyklus ist mit der äußersten rechten Zeichenposition von Identifier-1 beendet. Die Position des ersten Auftretens der Daten gemäß Identifier-7 bzw. Literal-5 wird durch INSPECT festgestellt, bevor der Vergleichszyklus ausgeführt wird.

Treten im Identifier-1 keine Daten gemäß Literal-5 bzw. Identifier-7 auf, dann findet kein Vergleichszyklus statt.

Beispiel 1:

```
INSPECT FELD-A REPLACING ALL "B" BY "F".
```

Daten in FELD-A vor dem INSPECT-Vorgang	Daten in FELD-A nach dem INSPECT-Vorgang
B B B A B C 1 2 B 9 B 0 0 0 0 0 0 2 8 B B B	F F F A F C 1 2 F 9 F 0 0 0 0 0 0 2 8 F F F

Beispiel 2:

```
INSPECT FELD-B REPLACING LEADING "Ø" BY "0".
```

Daten in FELD-B vor dem INSPECT-Vorgang	Daten in FELD-B nach dem INSPECT-Vorgang
Ø Ø Ø Ø 3 9 0	0 0 0 0 3 9 0
A Ø	A
Ø 9 2 Ø D	0 9 2 D
	0 0 0 0

Beispiel 3:

```
INSPECT FELD-C REPLACING FIRST "Z" BY "Y".
```

Daten in FELD-C vor dem INSPECT-Vorgang	Daten in FELD-C nach dem INSPECT-Vorgang
A Z 7	A Y 7
Z Z Z Z	Y Z Z Z
A B C D Z Z	A B C D Y Z
0 1 2 Z 3 Z	0 1 2 Y 3 Z

Beispiel 4:

```
INSPECT FELD-D REPLACING CHARACTERS BY "*".
```

Daten in FELD-D vor dem INSPECT-Vorgang	Daten in FELD-D nach dem INSPECT-Vorgang
0 1 2 3 4	* * * * *
A B	* *
A 1 0 *	* * * *
9 2 0 1	* * * *

Beispiel 5:

```
INSPECT FELD-E REPLACING ALL "1" BY "0" BEFORE INITIAL "9".
```

Daten in FELD-E vor dem INSPECT-Vorgang	Daten in FELD-E nach dem INSPECT-Vorgang
1 1 1 9 9	0 0 0 9 9
1 9 1 8 1 4	0 9 1 8 1 4
9 1 0 2	9 1 0 2
1 1 1 1 1 1 1	0 0 0 0 0 0 0
1 2 1 A B 9 3 1	0 2 0 A B 9 3 1

Beispiel 6:

```
INSPECT FELD-F REPLACING LEADING "A" BY "Z" AFTER INITIAL
    "H".
```

Daten in FELD-F vor dem INSPECT-Vorgang	Daten in FELD-F nach dem INSPECT-Vorgang
A A H A A	A A H Z Z
A B H H A	A B H H A
H A H A	H Z H A
A A A	A A A
1 2 9 H K	1 2 9 H K

Beispiel 7:

```
INSPECT FELD-G REPLACING ALL "AB" BY "XY".
```

Daten in FELD-G vor dem INSPECT-Vorgang	Daten in FELD-G nach dem INSPECT-Vorgang
B A B C A B B A B	B X Y C X Y B X Y
A A B B A A A A B	A X Y B A A A X Y
1 2 F G P R S	1 2 F G P R S
A B A B A B A B	X Y X Y X Y X Y

Beispiel 8:

```
INSPECT FELD-H REPLACING LEADING "12" BY "98".
```

Daten in FELD-H vor dem INSPECT-Vorgang	Daten in FELD-H nach dem INSPECT-Vorgang
1 2 1 2 1 2 1 2 1 2	9 8 9 8 9 8 9 8 9 8
0 1 2 1 2 1 2 1	0 1 2 1 2 1 2 1
1 2 1 2 2 1 2 1 2	9 8 9 8 2 1 2 1 2
1 2 0 1 2 1 2 1 2 4	9 8 0 1 2 1 2 1 2 4

Beispiel 9:

```
INSPECT FELD-I REPLACING ALL "B" BY "S",
    "X" BY "W", "Z" BY "P".
```

Daten in FELD-I vor dem INSPECT-Vorgang	Daten in FELD-I nach dem INSPECT-Vorgang
B B X X Z Z A	S S W W P P A
B B X X Z Z A A B B	S S W W P P A A S S
A B B X P Z S	A S S W P P S
B X X A Z B A B A B	S W W A P S A S A S

Beispiel 10:

```
INSPECT FELD-J REPLACING ALL "B" BY "S" AFTER INITIAL "A",
    "X" BY "W" AFTER INITIAL "A",
    "Z" BY "P" AFTER INITIAL "A".
```

Daten in FELD-J vor dem INSPECT-Vorgang	Daten in FELD-J nach dem INSPECT-Vorgang
B B X X Z Z A	B B X X Z Z A
B B X X Z Z A A B B	B B X X Z Z A A S S
A B B X P Z S	A S S W P P S
B X X A Z B A B A B	B X X A P S A S A S

Beispiel 11:

```
INSPECT FELD-K REPLACING ALL "B" BY "S", "X" BY "W",
    "Z" BY "P" AFTER INITIAL "A".
```

Daten in FELD-K vor dem INSPECT-Vorgang	Daten in FELD-K nach dem INSPECT-Vorgang
B B X X Z Z A	S S W W Z Z A
B B X X Z Z A A B B	S S W W Z Z A A S S
A B B X P Z S	A S S W P P S
B X X A Z B A B A B	B W W A P S A S A S

Beispiel 12:

```
INSPECT FELD-L REPLACING ALL H"7F" BY H"30"
```

Daten in FELD-L vor dem INSPECT-Vorgang	Daten in FELD-L nach dem INSPECT-Vorgang
7F 7F 10 20 00 7F	30 30 10 20 00 30
22 11 33 77 7F	22 11 33 77 30
00 00 00 30	00 00 00 30
10 7F	10 30
7F 11	30 11

Die INSPECT-Anweisung (TALLYING- und REPLACING-Variante)

Die dritte Variante der INSPECT-Anweisung zählt und ersetzt das jeweilige Auftreten eines einzelnen Zeichens oder von Zeichengruppen in einem Datenfeld.

Format:

```
INSPECT Identifier-1 TALLYING

[ {Identifier-2} FOR [ {ALL       } {Identifier-3} [ {BEFORE} INITIAL
                     [ {LEADING   } {Literal-1   } [ {AFTER } ]
                     [ {CHARACTERS}                ]
  [ {Identifier-4} ] ] ... ] ...
  {Literal-2   }

REPLACING

[ CHARACTERS BY {Identifier-6} [ {BEFORE} INITIAL {Identifier-7} ]
               {Literal-4   }   {AFTER }         {Literal-5   }

  {ALL       } {Identifier-5} BY {Identifier-6} [ {BEFORE} INITIAL
  {LEADING   } {Literal-3   }    {Literal-4   } [ {AFTER } ]
  {FIRST     }
  [ {Identifier-7} ] ] ... ] ... ]
  {Literal-5   }
```

Die TALLYING- und REPLACING-Variante der INSPECT-Anweisung führt eine Prüfung des im Identifier-1 bezeichneten Datenfeldes nach Einzelzeichen oder Zeichengruppen aus. Während der ersten Prüfung von Identifier-1 wird jedes dort ermittelte Auftreten der Daten gemäß Literal-1 bzw. Identifier-3 im Identifier-2 gezählt. Nach abgeschlossenem Zählvorgang verursacht eine zweite Prüfung von Identifier-1, daß alle ermittelten Daten gemäß Literal-3 bzw. Identifier-5 durch Zeichen im Literal-4 bzw. Identifier-6 ersetzt werden.

Die TALLYING- und REPLACING-Variante der INSPECT-Anweisung wird so ausgeführt, als ob der Programmierer eine INSPECT TALLYING-Anweisung unmittelbar gefolgt von einer INSPECT REPLACING-Anweisung geschrieben hätte. Beide Anweisungen bezeichnen denselben Identifier-1.

Beispiel 1:

```
MOVE ZERO TO ZAHL.
INSPECT FELD TALLYING ZAHL FOR LEADING "B"
    REPLACING ALL "B" BY "R".
```

Daten in FELD vor dem INSPECT-Vorgang	Daten in FELD nach dem INSPECT-Vorgang	Inhalt des Zählers ZAHL nach dem INSPECT-Vorgang
B B B B A B C D	R R R R A R C D	0 4
B B B B B	R R R R R	0 5
R A P B B	R A P R R	0 0
B 1 2 B	R 1 2 R	0 1

Beispiel 2:

```
MOVE ZERO TO ZAHL.
INSPECT FELD TALLYING ZAHL FOR ALL " " BEFORE INITIAL
    "A" REPLACING ALL " " BY "0" BEFORE INITIAL "A".
```

Daten in FELD vor dem INSPECT-Vorgang	Daten in FELD nach dem INSPECT-Vorgang	Inhalt des Zählers ZAHL nach dem INSPECT-Vorgang
␣ ␣ ␣ A B A	0 0 0 A B A	0 3
␣ 1 2 3 ␣	0 1 2 3 0	0 2
␣ A A A	0 A A A	0 1
B C ␣ ␣ K ␣ A ␣	B C 0 0 K 0 A ␣	0 3

Beispiel 3:

```
MOVE ZERO TO ZAHL.
INSPECT FELD TALLYING ZAHL FOR CHARACTERS BEFORE
    INITIAL "A" REPLACING CHARACTERS BY "Z" AFTER INITIAL
    "B".
```

Daten in FELD vor dem INSPECT-Vorgang	Daten in FELD nach dem INSPECT-Vorgang	Inhalt des Zählers ZAHL nach dem INSPECT-Vorgang
1 2 3 A B C	1 2 3 A B Z	0 3
A K B J K	A K B Z Z	0 0
0 0 1 2 3	0 0 1 2 3	0 5
A A A	A A A	0 0

Beispiel 4:

```
MOVE ZERO TO ZAHL.
INSPECT FELD TALLYING ZAHL FOR LEADING H"1F"
    REPLACING LEADING H"1F" BY H"00".
```

Daten in FELD vor dem INSPECT-Vorgang	Daten in FELD nach dem INSPECT-Vorgang	Inhalt des Zählers ZAHL nach dem INSPECT-Vorgang
1F 1F 18	00 00 18	02
1F 66 54 1F	00 66 54 1F	01
1F 1F 1F	00 00 00	03
2F 1F 1F 1F 1F	2F 1F 1F 1F 1F	00
1F 1F 1F 1E 1F	00 00 00 1E 1F	03

Die INSPECT-Anweisung (CONVERTING-Variante)

Diese Anweisung dient der Konvertierung von Daten in Identifier-1. Sie wirkt wie eine Serie von INSPECT REPLACING-Anweisungen mit der ALL-Klausel.

Format:

```
INSPECT Identifier-1

          {Identifier-2}        {Identifier-3}
CONVERTING{Literal-1   } TO     {Literal-2   }

[{BEFORE}                {Identifier-4}  ]
[{AFTER } INITIAL        {Literal-3   }  ]  ...
```

Beispiel:

```
INSPECT AFELD CONVERTING "ABCD" TO "XYZX"
        AFTER QUOTE BEFORE "!".

AFELD vorher:   AC"AEBDFBCD!AB"D
AFELD nachher:  AC"XEYXFYZX!AB"D
```

DIE MOVE-ANWEISUNG

Die MOVE-Anweisung überträgt Daten von einem Datenbereich in einen oder mehrere andere Bereiche, je nach Aufbereitungsregeln.

Format:

```
MOVE {Identifier-1} TO Identifier-2 [, Identifier-3] ...
     {Literal     }
```

Die Daten, die durch Literal oder Identifier-1 gekennzeichnet sind, werden zuerst nach Identifier-2, anschließend nach Identifier-3 usw. übertragen. Identifier-1 und Literal sind Sendefelder; Identifier-2 und Identifier-3 sind Empfangsfelder.

Bei jedem Übertrag, bei dem Sende- und Empfangsfeld Elementarfelder sind, handelt es sich um einen Elementar-Übertrag. Jedes Elementarfeld gehört zu einer der folgenden Kategorien: alphabetisch, numerisch, alphanumerisch, alphanumerisch aufbereitet, numerisch aufbereitet, boolesch.

Die folgende Tabelle zeigt die erlaubten Kombinationen von Sende- und Empfangsfeldern und die dazugehörigen Übertragungsregeln (Siehe nächste Seite):

Sendefeld \ Empfangsfeld	alphabetisch	- alphanumerisch - alphanumerisch editiert	numerisch - ganzzahlig - m. Dez. St. - editiert	boolesch
alphabetisch	LINKS *	LINKS *	verboten	verboten
alphanumerisch			DEZ	BOOLE
alphanumerisch-editiert			verboten	verboten
numerisch-ganzzahlig	verboten	LINKS * 1)	DEZ	
numerisch mit Dezimal-Stellen		verboten		
numerisch-editiert		LINKS *	verboten	
boolesch				BOOLE

DEZ = Dezimale Angleichung nach den Regeln des numerischen Elementarübertrags.

LINKS = Links-Angleichung nach den Regeln des alphanumerischen Übertrags.

BOOLE = Links-Angleichung nach den Regeln des booleschen Übertrags.

1) = Das Vorzeichen aus dem Sendefeld wird nicht übertragen.

* = Enthält die Definition des Empfangsfeldes die Klausel JUSTIFIED, werden die Daten rechtsstatt linksbündig übertragen.

Nachfolgend sind die nicht erlaubten Elementar-Überträge aufgeführt:

- Ein numerisch editiertes, ein alphanumerisch editiertes bzw. ein alphabetisches Datenfeld darf nicht in ein numerisches, ein numerisch editiertes bzw. ein boolesches Datenfeld übertragen werden.

- Die figurative Konstante SPACE darf nicht in ein numerisches, ein numerisch editiertes bzw. ein boolesches Datenfeld übertragen werden.

- Ein numerisches Literal, ein boolesches Literal, die figurative Konstante ZERO, ein numerisches Datenfeld, ein numerisch editiertes Datenfeld bzw. ein boolesches Datenfeld darf nicht in ein alphabetisches Datenfeld übertragen werden.

- Ein numerisches Literal mit Dezimalstellen bzw. ein numerisches Datenfeld mit Dezimalstellen darf nicht in ein alphanumerisches bzw. alphanumerisch editiertes Datenfeld übertragen werden.

- Ein numerisches Datenfeld, ein numerisch editiertes Datenfeld, ein numerisches Literal oder eine andere figurative Konstante als ZERO darf nicht in ein boolesches Datenfeld übertragen werden.

- Ein boolesches Datenfeld darf nicht in ein numerisches Datenfeld bzw. ein numerisch editiertes Datenfeld übertragen werden.

Jede notwendige Umwandlung von Daten von einer Form interner Darstellung in eine andere findet während eines erlaubten Elementar-Übertrages statt. Jede für das Empfangsfeld spezifizierte Aufbereitung findet ebenfalls während eines erlaubten Elementar-Übertrages statt.

Ist das Empfangsfeld ein alphanumerisch editiertes bzw. ein alphanumerisches Datenfeld, werden die empfangenen Daten linksbündig ausgerichtet. Rechts des Empfangsfeldes erfolgt ein Auffüllen mit Leerstellen oder ein Abschneiden. Ist das Sendefeld ein numerisches Datenfeld mit Vorzeichen, wird das Vorzeichen nicht übertragen und die Länge des Sendefeldes ist dann Aktuallänge minus ein Zeichen. Aufbereitung erfolgt, wenn das Empfangsfeld ein alphanumerisch editiertes Datenfeld ist.

Ist das Empfangsfeld ein alphabetisches Datenfeld, werden die empfangenen Daten linksbündig ausgerichtet. Rechts des Empfangsfeldes erfolgt ein Auffüllen mit Leerstellen oder ein Abschneiden.

Ist das Empfangsfeld ein numerisches bzw. numerisch editiertes Datenfeld, werden die Daten auf den Dezimalseparator angeglichen und der Rest des Feldes mit Nullen aufgefüllt, außer dort, wo wegen Aufbereitung keine Nullen stehen können. Weist das Empfangsfeld ein Vorzeichen auf, so wird dieses in das Vorzeichen des Sendefeldes umgewandelt. Die Umwandlung der Darstellung des Vorzeichens erfolgt immer dann, wenn es nötig ist. Ist das Sendefeld ohne Vorzeichen, dann wird das Empfangsfeld ein positives Vorzeichen (Pluszeichen) generiert. Ist das Empfangsfeld ohne Vorzeichen, wird der Absolutwert des Sendefeldes übertragen und kein Vorzeichen wird für das Empfangsfeld generiert. Ist das Sendefeld ein alphanumerisches Datenfeld, dann erfolgt die Übertragung der Daten als wäre das Sendefeld ein numerisches ganzzahliges Datenfeld ohne Vorzeichen. Aufbereitung erfolgt, wenn das Empfangsfeld ein numerisch editiertes Datenfeld ist.

Ist das Empfangsfeld ein boolesches Datenfeld, werden die empfangenen Daten dort linksbündig gespeichert. Überzählige Stellen rechts im Empfangsfeld werden mit Null-Bits aufgefüllt. Überzählige Stellen rechts im Sendefeld werden während der Übertragungen auf ein kleineres Empfangsfeld abgeschnitten.

Sind das Sende- und Empfangsfeld nicht beide Elementarfelder, erfolgt jede Übertragung so, als wäre sie von einem alphanumerischen Elementarfeld in ein anderes alphanumerisches Elementarfeld geschehen. Es gibt jedoch keine Datenumwandlung von einer Form interner Darstellung in eine andere. Bei einer solchen wird das Empfangsfeld von links nach rechts gefüllt ohne Berücksichtigung der Elementar- bzw. Gruppenfelder, wie sie entweder in Sende- oder Empfangsbereichen enthalten sind.

Ein Index-Datenfeld darf nicht als ein Operand einer MOVE-Anweisung auftreten.

Jedes Subskript bzw. jeder Index-Name, die dem Identifier-2 oder Identifier-3 zugeordnet sind, wird ausgewertet unmittelbar bevor die Daten zum entsprechenden Datenfeld übertragen werden. Jedes Subscript bzw. jeder Index-Name, das bzw. der dem Identifier-1 zugeordnet ist, wird nur einmal ausgewertet, und zwar unmittelbar bevor die Daten zum ersten der Empfangs-Operanden übertragen werden.

Beispiel 1:

```
MOVE ZERO TO FELD-A FELD-B.
```

	Picture	vor Ausführung	nach Ausführung
FELD-A	999	123	000
FELD-B	9999V99	0014$_{\wedge}$58	0000$_{\wedge}$00

Beispiel 2:

```
MOVE FELD-1 TO FELD-2.
```

	Picture	vor Ausführung	nach Ausführung
FELD-1	XXX	ABC	ABC
FELD-2	XXXX	XYWK	ABC$\not\emptyset$

Beispiel 3:

```
MOVE FELD-3 TO FELD-4.
```

	Picture	vor Ausführung	nach Ausführung
FELD-3	XXX	ABC	ABC
FELD-4	XX	XY	AB

Beispiel 4:

MOVE "123" TO FELD-5.

	Picture	vor Ausführung	nach Ausführung
FELD-5	XXXX	ABCD	123Ø

Beispiel 5:

MOVE KONTO-NR TO PR-KTO-NR.

	Picture	vor Ausführung	nach Ausführung
KONTO-NR	XXXX	A123	A123
PR-KTO-NR	XBXXX	AØCDE	AØ123

Beispiel 6:

MOVE 125,7 TO D-MARK.

	Picture	vor Ausführung	nach Ausführung
D-MARK	9999V99	$1234_{\wedge}66$	$0125_{\wedge}70$

Beispiel 7:

MOVE -15 TO CODE-1.

	Picture	vor Ausführung	nach Ausführung
CODE-1	S999 TRAILING SEPARATE	123-	015-

Beispiel 8:

MOVE SUMME TO PR-SUMME.

	Picture	vor Ausführung	nach Ausführung
SUMME	9999V99	1258^39	1259^39
PR-SUMME	$9,999.99	$ 3,333.33	$1,258.39

Beispiel 9:

MOVE SU-1 TO PR-SU-1.

	Picture	vor Ausführung	nach Ausführung
SU-1	9999V999	1258^397	1258^397
PR-SU-1	$Z,ZZZ.99	$ØØØØØ.30	$1,258.39

Beispiel 10:

MOVE SU-2 TO PR-SU-2.

	Picture	vor Ausführung	nach Ausführung
SU-2	9999V99	0000^03	0000^03
PR-SU-2	$$,$$$.99	$23.19	ØØØØØ$.03

Beispiel 11:

MOVE NO-SUM TO PA-SUM.

	Picture	vor Ausführung	nach Ausführung
NO-SUM (8-Bit)	S999 TRAILING SEPARATE	128+	128+
PA-SUM (4-Bit)	S999 COMP	475-	128+

Beispiel 12:

MOVE PA-SUM TO NO-SUM.

	Picture	vor Ausführung	nach Ausführung
PA-SUM (4-Bit)	S999 COMP	294-	294-
NO-SUM (8-Bit)	S999 TRAILING SEPARATE	338-	294-

Beispiel 13:

MOVE SUMME-1 TO AUS-SUMME.

	Picture	vor Ausführung	nach Ausführung
SUMME-1	999PPP	238	238
AUS-SUMME	9(6)	129074	238000

Beispiel 14:

MOVE TOTAL-1 TO TOTAL-2.

	Picture	vor Ausführung	nach Ausführung
TOTAL-1	9(5)V99	$14794_{\wedge}23$	$14794_{\wedge}23$
TOTAL-2	999PP	438	147

Beispiel 15:

MOVE ORT TO P-ORT.

	Picture	vor Ausführung	nach Ausführung
ORT	X(8)	AUGSBURG	AUGSBURG
P-ORT	X(12) JUST RIGHT	BRAUNSCHWEIG	␣␣␣␣AUGSBURG

Beispiel 16:

MOVE SUMME (LAGER) TO LAGER W-SUMME (LAGER).

	Picture	vor Ausführung	nach Übertrag in LAGER	nach Übertrag in W-SUMME (LAGER)
LAGER	99	03	05	05
SUMME (3)	99	05	05	05
W-SUMME (5)	99	10	10	05

Beispiel 17:

MOVE KA-MENGE TO MA-MENGE.

	Picture	vor Ausführung	nach Ausführung
KA-MENGE	S999 TRAILING	12M	12M
MA-MENGE	S999 TRAILING SEPARATE	539-	124-

Beispiel 18:

MOVE H"17" TO HEX-WERT-1.

	Picture	vor Ausführung	nach Ausführung
HEX-WERT-1	X	05	17

Beispiel 19:

MOVE H"1A097C" TO HEX-WERT-2.

	Picture	vor Ausführung	nach Ausführung
HEX-WERT-2	XXX	7F6A5E	1A097C

Beispiel 20:

MOVE H"1C" TO HEX-WERT-3.

	Picture	vor Ausführung	nach Ausführung
HEX-WERT-3	XXX	601B0E	1C ∅ ∅

Beispiel 21:

MOVE B"01110111" TO B-WERT-1.

	Picture	vor Ausführung	nach Ausführung
B-WERT-1	1(8)	01010101	01110111

Beispiel 22:

MOVE B"01010101" TO B-WERT-2.

	Picture	vor Ausführung	nach Ausführung
B-WERT-2	1(16)	0111011111111111	0101010100000000

Beispiel 23:

MOVE B"1111000010101010" TO B-WERT-3.

	Picture	vor Ausführung	nach Ausführung
B-WERT-3	1(8)	0101010101	11110000

Beispiel 24:

MOVE B-WERT-4 TO B-WERT-5.

	Picture	vor Ausführung	nach Ausführung
B-WERT-4	1(8)	10101111	10101111
B-WERT-5	1(8)	00000001	10101111

DIE MOVE-ANWEISUNG (JCL-CODE Variante)

Mit dieser MOVE-Anweisung wird ein numerischer Wert in ein (JCL-Code genanntes) Datenfeld des Betriebssystems übertragen.

Format:

```
MOVE { Identifier } TO JCL-CODE
     { Literal    }
```

Der Effekt dieser MOVE-Anweisung besteht darin, einen numerischen Wert in das JCL-CODE-Datenfeld zu übertragen, um dadurch eine Verbindung zum Betriebssystem herzustellen. Nach Ausführung des COBOL Objekt-Programmes prüft das Betriebssystem den Inhalt des JCL-CODE-Datenfeldes durch den System-Befehl IF. Der Wert des JCL-CODE-Datenfeldes muß ein numerischer Wert von 0 bis 7 sein.

Für das Datenfeld JCL-CODE nimmt der Programmierer keine Datenfeldbeschreibung vor.

Beispiel 1:

```
MOVE 3 TO JCL-CODE.
```

```
Literal 3 vor Ausführung:   3
JCL-CODE vor Ausführung:    5
Literal-3 nach Ausführung:  3
JCL-CODE nach Ausführung:   3
```

Beispiel 2:

```
MOVE ITX-CODE-1 TO JCL-CODE.
```

```
ITX-CODE-1 vor Ausführung:  1
JCL-CODE vor Ausführung:    4
ITX-CODE-1 nach Ausführung: 1
JCL-CODE nach Ausführung:   1
```

DIE MOVE CORRESPONDING-ANWEISUNG

Bei der CORRESPONDING-Variante der MOVE-Anweisung werden Felder gleichen Namens von einer Gruppe in eine andere Gruppe nach den entsprechenden Aufbereitungsregeln übertragen.

Format:

```
MOVE  {CORRESPONDING}  Identifier-1 TO Identifier-2
      {CORR         }
```

Datenfelder aus dem Gruppenfeld, das durch Identifier-1 bezeichnet wird, werden in korrespondierende Datenfelder der Gruppe, die durch Identifier-2 bezeichnet wird, übertragen. Es kann nur eine Gruppe als Empfangsbereich angegeben werden. Das Ergebnis einer MOVE CORRESPONDING-Anweisung entspricht einzelnen Übertragungen durch separate MOVE-Anweisungen.

Identifier-1 und Identifier-2 müssen jeweils eine Gruppe bezeichnen. Identifier-1 und Identifier-2 dürfen nicht auf Stufennummer 66, 77 und 88 definiert sein. Identifier-1 und Identifier-2 dürfen eine REDEFINES- oder OCCURS-Klausel enthalten; sie können auch Unterstufen eines Datennamens sein, der eine OCCURS- oder REDEFINES-Klausel enthält.

Bei einem Paar korrespondierender Datenfelder ist das Feld aus Identifier-1 das Sendefeld, das Feld aus Identifier-2 ist das Empfangsfeld. Für die Übertragung von Sende- nach Empfangsfeld gelten die gleichen Regeln wie beim MOVE für ein einzelnes Feld (siehe weiter vorn).

Beispiel 1:

```
MOVE CORRESPONDING EINGABE TO AUSGABE.
```

```
01  EINGABE
    02  NAME
        03  VORNAME
        03  FAMILIENNAME
    02  VERSICHERUNG
    02  ARBEITSZEIT
        03  NORMALSTD
        03  ÜBERSTD
    02  STDLOHN
    02  PERSDATEN
        03  ALTER
        03  GESCHLECHT
        03  AUSBILDUNG
        03  ANSCHRIFT
            04  STRASSE
            04  ORT
                05  PLZ
                05  ORTSNAME
```

```
01  AUSGABE
    02  NAME
        03  VORNAME
        03  FAMILIENNAME
    02  PERSDATEN
        03  ANSCHRIFT
            05  STRASSE
            05  ORT
        03  NORMALSTD
        03  ÜBERSTD
        03  URLAUB
        03  ALTER
        03  AUSBILDUNG
        03  GESCHLECHT
    02  STDLOHN
    02  VERSICHERUNG
```

Die korrespondierenden Daten-Felder in EINGABE und AUSGABE müssen folgende Bedingungen erfüllen, damit sie übertragen werden.

1. Ein Datenfeld mit dem Namen FILLER wird nicht übertragen.

2. Ein Datenfeld muß den gleichen Datennamen und die gleichen Qualifikationen in EINGABE und AUSGABE (aber nicht inklusive EINGABE oder AUSGABE) haben.

3. Eines der Datenfeld muß ein Elementar-Feld sein.

4. Keines der Datenfelder darf eine RENAMES, REDEFINES, OCCURS oder USAGE IS INDEX Klausel enthalten.

Da im Beispiel 1 keine RENAMES, REDEFINES, OCCURS oder USAGE IS INDEX Klausel enthalten ist sowie kein Datenfeld den reservierten Namen FILLER aufweist, ist die 1. Bedingung und die 4. Bedingung erfüllt.

Eines der Datenfelder, die von EINGABE in AUSGABE übertragen werden, ist VORNAME. VORNAME ist in EINGABE und AUSGABE ein Elementar-Feld und damit ist die 3. Bedingung erfüllt.

Die 2. Bedingung ist erfüllt, da VORNAME dieselben Qualifikationen (NAME) in EINGABE und AUSGABE hat, aber nicht einschließlich EINGABE und AUSGABE.

Die folgende Übersicht zeigt die Datenhierarchie für VORNAME in EINGABE und AUSGABE:

Hierarchie Stufe	in EINGABE	in AUSGABE
1	EINGABE	AUSGABE
2	NAME	NAME
3	VORNAME	VORNAME

Das Datenfeld NORMALSTD wird nicht von EINGABE in AUSGABE übertragen, da die 2. Bedingung nicht erfüllt ist. In EINGABE ist die Datenhierarchie von NORMALSTD anders als in AUSGABE, wie die folgende Übersicht zeigt:

Hierarchie Stufe	in EINGABE	in AUSGABE
1	EINGABE	AUSGABE
2	ARBEITSZEIT	PERSDATEN
3	NORMALSTD	NORMALSTD

Das Datenfeld STRASSE wird ebenfalls von EINGABE in AUSGABE übertragen, da es in beiden ein Elementarfeld ist und somit die 3. Bedingung erfüllt ist. Die Hierarchie der Daten ist in EINGABE und AUSGABE folgende: STRASSE IN ANSCHRIFT in PERSDATEN. Damit ist auch die 2. Bedingung über dieselbe Qualifikation erfüllt.

ORT ist ein anderes Datenfeld, das von EINGABE in AUSGABE übertragen wird. Die 3. Bedingung ist erfüllt, nachdem ORT in EINGABE zwar ein Gruppenfeld, dafür aber in AUSGABE ein Elementarfeld ist.

Die 2. Bedingung ist auch erfüllt, wie die folgende Übersicht zeigt:

Hierarchie Stufe	in EINGABE	in AUSGABE
1	EINGABE	AUSGABE
2	PERSDATEN	PERSDATEN
3	ANSCHRIFT	ANSCHRIFT
4	ORT	ORT

Das Ergebnis der MOVE CORRESPONDING-Anweisung in Beispiel 1 entspricht dem Ergebnis folgender Einzelanweisungen:

```
MOVE VORNAME IN NAME IN EINGABE
   TO VORNAME IN NAME IN AUSGABE.
MOVE FAMILIENNAME IN NAME IN EINGABE
   TO FAMILIENNAME IN NAME IN AUSGABE.
MOVE VERSICHERUNG IN EINGABE
   TO VERSICHERUNG IN AUSGABE.
MOVE STDLOHN IN EINGABE
   TO STDLOHN IN AUSGABE.
MOVE ALTER IN PERSDATEN IN EINGABE
   TO ALTER IN PERSDATEN IN AUSGABE.
MOVE GESCHLECHT IN PERSDATEN IN EINGABE
   TO GESCHLECHT IN PERSDATEN IN AUSGABE.
MOVE AUSBILDUNG IN PERSDATEN IN EINGABE
   TO AUSBILDUNG IN PERSDATEN IN AUSGABE.
MOVE STRASSE IN ANSCHRIFT IN PERSDATEN IN EINGABE
   TO STRASSE IN ANSCHRIFT IN PERSDATEN IN AUSGABE.
MOVE ORT IN ANSCHRIFT IN PERSDATEN IN EINGABE
   TO ORT IN ANSCHRIFT IN PERSDATEN IN AUSGABE.
```

In der MOVE CORRESPONDING-Anweisung müssen Identifier-1 und Identifier-2 nicht unbedingt mit Stufen-Nr. 01 definiert sein, wie das Beispiel 2 zeigt:

Beispiel 2:

```
MOVE CORR PERSDATEN IN EINGABE TO PERSDATEN IN
    AUSGABE.
```

Das Ergebnis der MOVE CORRESPONDING-Anweisung in Beispiel 2 entspricht dem Ergebnis folgender Einzelanweisungen:

```
MOVE ALTER IN PERSDATEN IN EINGABE
   TO ALTER IN PERSDATEN IN AUSGABE.
MOVE GESCHLECHT IN PERSDATEN IN EINGABE
   TO GESCHLECHT IN PERSDATEN IN AUSGABE.
MOVE AUSBILDUNG IN PERSDATEN IN EINGABE
   TO AUSBILDUNG IN PERSDATEN IN AUSGABE.
MOVE STRASSE IN ANSCHRIFT IN PERSDATEN IN EINGABE
   TO STRASSE IN ANSCHRIFT IN PERSDATEN IN AUSGABE.
MOVE ORT IN ANSCHRIFT IN PERSDATEN IN EINGABE
   TO ORT IN ANSCHRIFT IN PERSDATEN IN AUSGABE.
```

DIE MULTIPLY-ANWEISUNG

Durch die MULTIPLY-Anweisung wird ein numerisches Datenfeld mit einem anderen multipliziert und das Ergebnis gespeichert. Das allgemeine Format der MULTIPLY-Anweisung kann wie folgt aussehen.

Format 1:

```
MULTIPLY {Identifier-1} BY Identifier-2 [ROUNDED]
         {Literal-1   }

    [Identifier-3 [ROUNDED]] ...
[ON SIZE ERROR unbedingte Anweisung-1]
[NOT ON SIZE ERROR unbedingte Anweisung-2]
[END-MULTIPLY]
```

Format 2:

```
MULTIPLY {Identifier-1} BY {Identifier-2} GIVING Identifier-3 [ROUNDED]
         {Literal-1   }    {Literal-2   }

   [,Identifier-4 [ROUNDED]] ...
[ON SIZE ERROR unbedingte Anweisung-1]
[NOT ON SIZE ERROR unbedingte Anweisung-2]
[END-MULTIPLY]
```

Bei Format 1 wird der Inhalt von Identifier-1 oder Literal-1 mit den Inhalten von Identifier-2, Identifier-3 usw. multipliziert. Das Ergebnis der Multiplikation wird in den als Empfangsfelder oder Ergebnisfelder bezeichneten Identifier-2, Identifier-3 usw. gespeichert.

Bei Format 2 wird der Inhalt von Identifier-1 oder Literal-1 mit dem Inhalt von Identifier-2 oder Literal-2 multipliziert. Das Ergebnis der Multiplikation wird in den als Empfangsfelder oder Ergebnisfelder bezeichneten Identifier-3, Identifier-4 usw. gespeichert.

In Format 1 muß jeder Identifier ein numerisches Elementarfeld bezeichnen. In Format 2 muß jeder Identifier, mit Ausnahme des Identifiers, die auf das Wort GIVING folgen, ein numerisches Elementarfeld bezeichnen. Identifier-3, Identifier-4 usw. in Format 2 können entweder ein numerisches oder ein numerisch editiertes Elementarfeld bezeichnen. Jedes Literal in der MULTIPLY-Anweisung muß ein numerisches Literal sein.

Die Datenbeschreibung der Operanden in der MULTIPLY-Anweisung brauchen nicht dieselben zu sein, denn jede notwendige Umwandlung und Ausrichtung nach den Dezimalseparatoren während der Rechnung erfolgt automatisch. Die Operanden der MULTIPLY-Anweisung können von folgendem Datentyp sein:

Datentyp	maximale Länge
Dezimal ohne Vorzeichen	18 Bytes
Ungepackt dezimal mit Vorzeichen links oder rechts angefügt	19 Bytes einschließlich 8-Bit-Zeichen
Ungepackt dezimal mit Zonenvorzeichen rechts oder links	18 Bytes einschließlich Zonen-Zeichen
Gepackt dezimal ohne Vorzeichen	9 Bytes
Gepackt dezimal mit Vorzeichen	10 Bytes einschließlich 4-Bit-Zeichen
Binär ohne Vorzeichen	8 Bytes

MULTIPLY ist eine unbedingte Anweisung. Ist SIZE ERROR oder NOT SIZE ERROR in der MULTIPLY-Anweisung enthalten, dann wird sie zu einer bedingten Anweisung.

ROUNDED

Weist das Ergebnis der Multiplikation mehr Dezimalstellen (Stellen rechts vom Dezimalseparator) auf als das Ergebnisfeld, werden beide auf den Dezimalseparator ausgerichtet. Bei fehlender Angabe von ROUNDED wird der Dezimalstellen-Überhang abgeschnitten. Ist ROUNDED definiert, wird das Ergebnis um 1 erhöht, wenn die erste Ziffer des Überhanges > 4 ist. Wurde das Ergebnisfeld durch ein P in der PICTURE-Zeichenfolge ganzzahlig dargestellt, erfolgt Rundung oder Abschneiden für die ganze rechts befindliche Stelle, für die noch eine Speicherung erfolgen soll.

SIZE ERROR

Ist bei einer MULTIPLY-Anweisung SIZE ERROR angegeben, dann wird nach der Ausrichtung auf den Dezimalseparator und nach Rundung des Dezimalstellen-Überhanges eine Prüfung auf Überlauf des Ergebnisfeldes, d. h. auf eine SIZE ERROR-Bedingung vorgenommen. Eine Überlauf-Bedingung tritt dann auf, wenn der Wert des Ergebnisses den Wert übersteigt, den das Empfangsfeld maximal aufnehmen kann. Eine Überlauf-Prüfung erfolgt nur für den linken Rand des Empfangsfeldes. Würde das Ergebnisfeld überlaufen, wird das Ergebnis nicht im Empfangsfeld gespeichert. Wenn die MULTIPLY-Anweisung mehrere Empfangsfelder hat, wird das Ergebnis nur in die Empfangsfelder übertragen, die keine SIZE ERROR-Bedingung haben. Wenn ein oder mehrere Empfangsfelder die SIZE ERROR-Bedingung erfüllen, wird das Programm nach Ausführung der MULTIPLY-Anweisung mit der unbedingten Anweisung fortgesetzt. Die Schreibweise "unbedingte Anweisung" in der SIZE ERROR-Angabe bezieht sich auf eine oder mehrere aufeinanderfolgende, unbedingte Anweisungen, deren Folge durch einen Punkt beendet ist.

Ist die SIZE ERROR-Bedingung für das Ergebnisfeld nicht gegeben, jedoch SIZE ERROR angegeben, wird das Ergebnis der Multiplikation in dem Ergebnisfeld gespeichert und die nächste Anweisung ausgeführt; die unbedingte Anweisung in der SIZE ERROR-Angabe wird nicht ausgeführt.

Ist SIZE ERROR nicht angegeben und eine SIZE ERROR-Bedingung tritt auf, dann ist der Wert der Ergebnisfelder nicht vorhersagbar. Die auf die MULTIPLY-Anweisung folgende Anweisung wird ausgeführt und der Überlauf nicht festgestellt.

NOT ON SIZE ERROR

Hier gilt das für SIZE ERROR Gesagte unter umgekehrten Vorzeichen. D. h., daß die unbedingte(n) Ausweisung(en) dann ausgeführt werden, wenn kein Längenfehler aufgetreten und die Multiplikation erfolgreich verlaufen ist. Würde ein Längenfehler auftreten, unterbliebe die Ausführung der unbedingten Anweisung(en) und das Programm würde ab dem Beginn des nächsten COBOL-Satzes (nach dem Punkt) weiterarbeiten.

Beispiel 1:

```
MULTIPLY PREIS BY MENGE GIVING SUMME.

PREIS   vorher          300 +
MENGE   vorher          020 +
SUMME   vorher        09930 +
PREIS   nachher         300 +   unverändert
MENGE   nachher         020 +   unverändert
SUMME   nachher       06000 +   enthält das Ergebnis der
                                Multiplikation
```

Beispiel 2:

```
MULTIPLY 5,99 BY MENGE GIVING RABATT ROUNDED.

Literal 5,99  vorher      599 +
MENGE         vorher      050 +
RABATT        vorher      371 +
Literal 5,99  nachher     599 +   unverändert
MENGE         nachher     050 +   unverändert
RABATT        nachher     300 +   enthält das aufgerundete
                                  Ergebnis der Multiplikation
```

Beispiel 3:

```
MULTIPLY 5,99 BY MENGE GIVING MENGE-A ON SIZE ERROR GO TO
    FEHLER.
```

Literal 5,99	vorher	5^99 +	
MENGE	vorher	050 +	
MENGE-A	vorher	37^1 +	
Literal 5,99	nachher	5^99 +	unverändert
MENGE	nachher	050 +	unverändert
MENGE-A	nachher	37^1 +	unverändert, da das Ergebnis der Multiplikation die Größe des Empfangsfeldes (MENGE-A) übersteigt.

Da eine SIZE ERROR-Bedingung gegeben ist, werden nach der MULTIPLY-Anweisung die Anweisungen des Paragraphen FEHLER ausgeführt.

Beispiel 4:

```
MULTIPLY 10 BY MENGE-1.
```

MENGE-1	vorher	00043 +	
MENGE-1	nachher	00430 +	enthält das Ergebnis der Multiplikation

Beispiel 5:

```
MULTIPLY 10 BY MENGE-2, MENGE-3.
```

MENGE-2	vorher	0305^01	
MENGE-3	vorher	0400	
MENGE-2	nachher	3050^10	enthält Ergebnis von 10 x 0305,01
MENGE-3	nachher	4000	enthält Ergebnis von 10 x 0400

Beispiel 6:

```
MULTIPLY MENGE BY PROZ-SATZ GIVING ERG-1, ERG-2.

MENGE          vorher     04507
PROZ-SATZ      vorher      ^006
ERG-1          vorher     98^20
ERG-2          vorher      105
MENGE          nachher    04507    unverändert
PROZ-SATZ      nachher     ^006    unverändert
ERG-1          nachher    27^04    Ergebnis der Multiplikation
ERG-2          nachher     027     Ergebnis der Multiplikation
```

DIE OPEN-ANWEISUNG

Mit der OPEN-Anweisung werden Dateien eröffnet, Kennsätze geprüft und/oder geschrieben und andere Input-Output Operationen durchgeführt.

Format 1 (Sequentielle Dateien):

```
OPEN { INPUT    {Dateiname-1 [WITH NO REWIND]} [WITH LOCK] ...
     { OUTPUT   {Dateiname-2 [WITH NO REWIND]} [WITH LOCK] ...
     { I-O      {Dateiname-3 [WITH LOCK]} ...                  } ...
     { EXTEND   {Dateiname-4 [WITH LOCK]} ...
```

Format 2 (Relative und Index-Dateien):

```
OPEN { INPUT    {Dateiname-1} ...
     { OUTPUT   {Dateiname-2} ...  } ...
     { I-O      {Dateiname-3} ...
```

Jeder Datei-Name in der OPEN-Anweisung muß einem der Datei-Namen einer Dateibeschreibung und einem der Datei-Namen in einem FILE-CONTROL-Eintrag entsprechen. Die in der OPEN-Anweisung angesprochenen Dateien können verschiedene Strukturen (ORGANIZATION) und Zugriffsarten (ACCESS) aufweisen. Die OPEN-Anweisung gilt für alle Dateien.

Die korrekte Ausführung einer OPEN-Anweisung bestimmt die Verfügbarkeit der Datei und führt dazu, daß die Datei einen OPEN-Modus einnimmt. Durch die korrekte Ausführung einer OPEN-Anweisung steht der entsprechende Satz-Bereich für die Datei dem Programm zur Verfügung. Jedoch wird durch die OPEN-Anweisung der erste Datensatz einer Datei nicht gelesen bzw. nicht freigegeben.

Vor der Ausführung jeder erlaubten Prozedur-Anweisung für die Datei muß eine OPEN-Anweisung korrekt ausgeführt werden. Eine Datei wird im INPUT-Modus, im OUTPUT-Modus, im I-O-Modus oder im EXTEND-Modus eröffnet. Aus der Tabelle sind die erlaubten Prozedur-Anweisungen für jeden auf entweder eine sequentielle Datei, eine relative Datei oder eine indexierte Datei angewandten OPEN-Modus ersichtlich.

Die INPUT-Angabe bewirkt, daß eine sequentielle, relative oder indexierte Datei im INPUT-Modus eröffnet wird. Die INPUT-Angabe bedeutet das Vorhandensein der Datei. Besteht die Datei, ohne jedoch Datensätze zu enthalten, dann ergibt die erste für die Datei ausgeführte READ- oder READ NEXT-Anweisung eine AT END-Bedingung. Die READ-, READ NEXT- und START-Anweisungen sind die zulässigen Prozedur-Anweisungen für eine Eingabedatei je nach ihrer Zugriffsart.

Die OUTPUT-Angabe bewirkt, daß eine sequentielle, relative oder indexierte Datei im OUTPUT-Modus eröffnet wird. Die OUTPUT-Angabe bedeutet, daß die Datei zu Beginn der Ausführung des Objekt-Programmes nicht besteht. Somit muß die Datei durch die Ausführung des Objekt-Programmes erstellt werden. Die erfolgreiche Ausführung einer OPEN OUTPUT-Anweisung bewirkt, daß die Datei ohne irgendwelche Datensätze besteht. Die WRITE-Anweisung ist die erlaubte Prozedur-Anweisung für eine Ausgabedatei je nach ihrer Zugriffsart.

Die I-O-Angabe bewirkt, daß eine sequentielle, relative oder indexierte Datei im I-O-Modus eröffnet wird. Die I-O-Angabe kann nur für Magnetplatten-Dateien verwendet werden. Da die I-O-Angabe das Vorhandensein der Datei bedeutet, kann die I-O-Angabe nicht verwendet werden, wenn die Datei neu erstellt wird. Besteht die Datei, ohne jedoch Datensätze zu enthalten, dann ergibt die erste READ- oder READ NEXT-Anweisung, die für die Datei ausgeführt wird, eine AT END-Bedingung. Die READ-, READ NEXT-, WRITE-, REWRITE-, START- und DELETE-Anweisungen sind die zulässigen Prozedur-Anweisungen für eine Eingabe-Ausgabe-Datei je nach ihrer Zugriffsart.

Die EXTEND-Angabe bewirkt, daß eine sequentielle Datei auf Platte oder Magnetband in dem EXTEND-Modus eröffnet wird. Ist eine sequentielle Datei auf Platte oder Magnetband bereits erstellt worden, dann positioniert die OPEN EXTEND-Anweisung die neu erstellte Datei hinter dem letzten Datensatz der schon bestehenden Datei; eine solche EXTEND-Datei wird im Betriebssystem mit dem wahlweisen Parameter OLD im ASSIGN-Systembefehl zugeordnet. Ist eine sequentielle Datei nicht schon früher erstellt worden, dann positioniert die OPEN EXTEND-Anweisung die Datei derart, daß die erste für die Datei durchgeführte WRITE-Anweisung bewirkt, daß ein Datensatz am Anfang der Datei geschrieben wird; eine solche EXTEND-Datei wird im Betriebssystem mit dem erforderlichen Parameter NEW im ASSIGN-Systembefehl zugeordnet. Nachfolgende WRITE-Anweisungen, die sich auf die Datei beziehen, fügen der Datei Datensätze an, als ob diese mit der OPEN-

OUTPUT-Anweisung eröffnet worden wäre. Die WRITE-Anweisung ist die zulässige Prozedur-Anweisung für eine Datei, die im EXTEND-Modus eröffnet wird. Die EXTEND-Angabe ist nicht zulässig bei Magnetband-Ausgabe, wenn mehrere Dateien auf einer Spule liegen.

		OPEN-Modus → / Anweisung ↓	ORGANIZATION											
			SEQUENTIAL				RELATIVE				INDEXED			
			INPUT	OUTPUT	I-O	EXTEND	INPUT	OUTPUT	I-O	EXTEND	INPUT	OUTPUT	I-O	EXTEND
ACCESS	SEQUENTIAL	READ	X		X		X		X		X		X	
		READ NEXT												
		WRITE		X		X		X				X		
		REWRITE			X				X				X	
		START					X		X		X		X	
		DELETE							X				X	
	RANDOM	READ					X		X		X		X	
		READ NEXT												
		WRITE						X	X			X	X	
		REWRITE							X				X	
		START												
		DELETE							X				X	
	DYNAMIC	READ					X		X		X		X	
		READ NEXT					X		X		X		X	
		WRITE						X	X			X	X	
		REWRITE							X				X	
		START					X		X		X		X	
		DELETE							X				X	

Eine Datei kann im gleichen Programm mit der INPUT-, OUTPUT-, EXTEND- und I-O-Angabe eröffnet werden. Nach der erstmaligen Ausführung der OPEN-Anweisung für eine Datei muß jeder nachfolgenden Ausführung der OPEN-Anweisung für diese Datei die Ausführung einer CLOSE-Anweisung ohne die REEL-, UNIT- oder LOCK-Angabe für diese Datei vorausgehen.

Die Ausführung einer OPEN-Anweisung für eine Magnetbanddatei auf einer Spule bewirkt, daß die Magnetbanddatei an ihrem Anfang positioniert wird. Ist die NO REWIND-Angabe für eine auf einer Spule befindliche Magnetbanddatei spezifiziert, dann bewirkt die Ausführung der OPEN-Anweisung nicht, daß die Datei zurückgesetzt wird; die Datei muß bereits vor der Ausführung der OPEN-Anweisung an ihrem Anfang positioniert sein.

Die Ausführung einer OPEN-OUTPUT-Anweisung mit einer zugeordneten LINAGE-Klausel für eine Druck-Datei bewirkt den Papiervorschub auf die erste Zeile der nächsten logischen Seite; der LINAGE-COUNTER wird auf den Wert 1 gestellt. Die Ausführung einer OPEN OUTPUT-Anweisung ohne eine zugeordnete LINAGE-Klausel für eine Druck-Datei bewirkt, daß das Papier am physischen Anfang der nächsten Seite positioniert wird. Die Ausführung einer OPEN EXTEND-Anweisung für eine Druckdatei bewirkt keine Papierbewegung, vielmehr bleibt das Papier in der Stellung, die es gerade einnimmt.

Ist die FILE STATUS-Klausel im FILE-CONTROL-Eintrag für eine Datei spezifiziert, dann wird dieser Datei ein FILE STATUS-Datenfeld zugeordnet. Nach Ausführung der OPEN-Anweisung wird für jede Datei, der ein FILE STATUS-Datenfeld zugeordnet ist, ein Wert in das FILE STATUS-Datenfeld übertragen. Alle DECLARATIVE-Prozeduren, die auf eine Datei anwendbar sind, werden ausgeführt, nachdem das FILE STATUS-Datenfeld seinen Wert empfängt, der dem Stand der OPEN-Anweisung entspricht. Die möglichen FILE STATUS-Werte stehen in der Tabelle im Anhang.

DIE PERFORM-ANWEISUNG

Die PERFORM-Anweisung gestattet das Verzweigen aus der normalen Befehlsfolge heraus, damit eine oder mehrere Prozeduren ausgeführt werden können, mit anschließender Rückkehr hinter die PERFORM-Anweisung in der normalen Befehlsfolge.

Format:

```
PERFORM   Prozedur-Name-1 [ {THROUGH} Prozedur-Name-2 ]
                          {THRU    }
   [unbedingte Anweisung END-PERFORM]
```

Jeder Prozedur-Name in der PERFORM-Anweisung muß ein Kapitel- oder Paragraphen-Name in der PROCEDURE DIVISION sein.

Ist das Format PERFORM Prozedur-Name-1 spezifiziert, bewirkt die PERFORM-Anweisung eine unmittelbare Steuerungsübertragung zur ersten (Prozedur-Name-1 genannten) Prozeduranweisung. Die Ausführung der PERFORM-Anweisung bewirkt auch, daß eine indirekte Steuerungsübertragung zur Anweisung hinter der PERFORM-Anweisung erfolgt: Wenn Prozedur-Name-1 ein Paragraphen-Name ist, so kehrt die Steuerung nach Ausführung der letzten Anweisung innerhalb des Paragraphen zur nächsten Anweisung hinter der PERFORM-Anweisung zurück. Ist Prozedur-Name-1 ein Kapitel-Name, so kehrt die Steuerung nach Ausführung der letzten Anweisung des letzten Paragraphen des Kapitels zur nächsten Anweisung der PERFORM-Anweisung zurück.

Auf den nächsten Seiten finden Sie Beispiele:

Beispiel 1:

```
A1. MULTIPLY SUMME BY 300 GIVING TOTAL-SUMME.
    PERFORM BERECHNUNG.
    ADD 100 TO TOTAL.
        -------------------

BERECHNUNG.
    ADD 10 TO TOTAL.
    MOVE TOTAL TO TOTAL-NEU.
    SUBTRACT TOTAL FROM TOTAL-ALT.
UEBERW-DATUM.   IF VORG-DAT IS EQUAL TO TAGES-DAT MOVE
                TAGES-DAT TO AUSGABE-DAT.
        -------------------
```

In Beispiel 1 werden durch die Anweisung PERFORM BERECHNUNG die drei im Paragraphen BERECHNUNG enthaltenen Anweisungen ausgeführt. Die Anweisungen werden somit in folgender Reihenfolge ausgeführt:

```
MULTIPLY SUMME BY 300 GIVING TOTAL-SUMME.
ADD 10 TO TOTAL.
MOVE TOTAL TO TOTAL-NEU.
SUBTRACT TOTAL FROM TOTAL-ALT.
ADD 100 TO TOTAL.
```

Ist das Format PERFORM Prozedur-Name-1 THRU Prozedur-Name-2 angegeben, bewirkt die PERFORM-Anweisung eine unmittelbare Steuerungsübertragung zur ersten, Prozedur-Name-1 genannten Prozeduranweisung. Nach Übergang der Steuerung auf Prozedur-Name-1 wird eine Reihenfolge von Operationen ausgeführt, beginnend bei Prozedur-Name-1 und endend mit der letzten Anweisung der mit Prozedur-Name-2 bezeichneten Prozedur. Wenn Prozedur-Name-2 ein Paragraphen-Name ist, so kehrt die Steuerung nach Ausführung der letzten Anweisung innerhalb des Paragraphen zur nächsten Anweisung hinter der PERFORM-Anweisung zurück. Ist Prozedur-Name-2 ein Kapitel-Name, so kehrt die Steuerung nach Ausführung der letzten Anweisung des letzten Paragraphen des Kapitels zur nächsten Anweisung hinter der PERFORM-Anweisung zurück. Die Wörter THRU und THROUGH sind gleichbedeutend.

Beispiel 2:

```
ABC.   MOVE KTO-NR-NEU TO KTO-NR.
       PERFORM UEBERTRAG THRU BERECHNUNG.
       WRITE AUSG-SATZ.
           -----------------

VERGLEICH.
       IF CODE-1 = 1 GO TO UNTERPROG-1.
       IF CODE-1 = 2 GO TO UNTERPROG-2.
       IF CODE-1 = 3 GO TO UNTERPROG-3.
UEBERTRAG.   MOVE SALDO TO SALDO-NEU.
       MOVE CODE-1 TO CODE-NEU.
NEUSATZ.     MOVE 2 TO CODE-NEU-L.
       MOVE DATUM TO DATUM-NEU.
BERECHNUNG. ADD SUMME TO SALDO.
       SUBTRACT 35 FROM LIMIT-1.
AUSGABE.     WRITE NEU-SATZ FROM BER-ALT.
       ADD 1 TO ZAEHLER-1.
```

In Beispiel 2 werden die Anweisungen in folgender Reihenfolge ausgeführt:

```
MOVE KTO-NR-NEU TO KTO-NR.
MOVE SALDO TO SALDO-NEU.
MOVE CODE-1 TO CODE-NEU.
MOVE 2 TO CODE-NEU-L.
MOVE DATUM TO DATUM-NEU.
ADD SUMME TO SALDO.
SUBTRACT 35 FROM LIMIT-1.
WRITE AUSG-SATZ.
```

Ein in einer PERFORM-Anweisung benannter Prozedur-Name kann ein in den DECLARATIVE-Prozeduren eingeschlossener Prozedur-Name sein. Sind sowohl Prozedur-Name-1 als auch Prozedur-Name-2 spezifiziert und ist jeder davon der Name einer Prozedur in dem DECLARATIVE-Kapitel des Programmes, dann müssen beide Prozedur-Namen in demselben DECLARATIVE-Kapitel liegen.

Die Prozeduren, auf die durch irgendeine PERFORM-Anweisung Bezug genommen wird, werden bei jeder Ausführung der PERFORM-Anweisung einmal ausgeführt. Die Steuerung geht anschließend zur nächsten Anweisung hinter der PERFORM-Anweisung über.

Geht die Steuerung anders als durch eine PERFORM-Anweisung an eine Serie von Prozeduren über (z. B. mittels einer GO TO-Anweisung), dann geht es nach der letzten Anweisung dieser Prozeduren weiter, als ob keine PERFORM-Anweisung gegeben wäre. Beispielsweise schließt die Anweisung GO TO UEBERTRAG die Ausführung der folgenden Reihenfolge von Anweisungen ein:

```
MOVE SALDO TO SALDO-NEU.
MOVE CODE-1 TO CODE-NEU.
MOVE 2 TO CODE-NEU-L.
MOVE DATUM TO DATUM-NEU.
ADD SUMME TO SALDO.
SUBTRACT 35 FROM LIMIT-1.
WRITE NEU-SATZ FROM BER-ALT.
ADD 1 TO ZAEHLER-1.
```

Ein drittes Beispiel finden Sie auf der nächsten Seite:

Umfaßt eine Serie von Anweisungen, die durch eine PERFORM-Anweisung aktiviert wird, eine weitere PERFORM-Anweisung, dann muß die Serie von Anweisungen, die von dieser neuen PERFORM-Anweisung aufgerufen wird, entweder vollständig eingeschlossen oder vollständig ausgeschlossen von der durch die erste PERFORM-Anweisung angesprochenen logischen Reihenfolge sein. D. h.: Solche PERFORM-Anweisungen, die zur selben Zeit aktiv sind, dürfen keinen gemeinsamen Ausgang haben.

In Beispiel 3 befindet sich PARAGRAPH-E vollständig außerhalb der durchgeführten Folge PARAGRAPH-C THRU PARAGRAPH-D. Dies ist in Ordnung:

Beispiel 3:

```
PARAGRAPH-A.  PERFORM PARAGRAPH-C THRU PARAGRAPH-D.
PARAGRAPH-B.  MOVE 100 TO SUMME NEU.
  ........................
PARAGRAPH-C.
  ........................
  ........................
     PERFORM PARAGRAPH-E.
  ........................
  ........................
PARAGRAPH-D.
  ........................
PARAGRAPH-E.
  ........................
PARAGRAPH-F.
  ....:...................
```

Ein viertes Beispiel finden Sie auf der nächsten Seite:

In Beispiel 4 befindet sich P-3 vollständig innerhalb der durchgeführten Folge P-2 THRU P-5. Dies ist ein Beispiel vollständigen Einschlusses und ebenfalls in Ordnung:

Beispiel 4:

```
P-1.    PERFORM P-2 THRU P-5.
        ADD 4000, SUMME GIVING SUMME-NEU.
        ........................
P-2.
        ........................
P-3.
        ........................
P-4.
        ........................
        PERFORM P-3.
P-5.
        ........................
P-6.
        ........................
```

Auf der nächsten Seite finden Sie ein fünftes Beispiel:

Die einer PERFORM-Anweisung zugeordnete Folge von Prozeduren kann die einer anderen PERFORM-Anweisung zugeordnete Folge von Prozeduren überlappen oder überschneiden, vorausgesetzt, daß keine der beiden Folgen die der jeweils anderen Folge zugeordnete PERFORM-Anweisung enthält. Solche PERFORM-Anweisungen dürfen nicht zur selben Zeit, sondern nur nacheinander aktiv sein.

Beispiel 5 zeigt zwei solche sich in der Ausführung überlappende Folgen, eine Programmierweise, die so in Ordnung geht.

Beispiel 5:

```
P-TEST.   PERFORM TEST-1 THRU NULL-DIFF.
          ADD 1 TO MENGE-4.
          ........................

P-SALDO.  PERFORM SALDO-UEB THRU SALDO-TEST.
          MULTIPLY SUMME BY PROZ GIVING RABATT.
          ........................
TEST-1.
          ........................
ADDITION.
          ........................
SALDO-UEB.
          ........................
NULL-DIFF.
          ........................
UEBERTRAG.
          ........................
SALDO-TEST.
          ........................
END-SUMMEN.
          ........................
```

Auf der nächsten Seite finden Sie ein sechstes Beispiel:

GO TO-Anweisungen können zwischen Prozedur-Name-1 und dem Ende von Prozedur-Name-2 erfolgen. Eine oder mehrere verwendete GO TO-Anweisungen können zu zwei oder mehreren Wegen verzweigen, die jedoch wieder in das ausgeführte Unterprogramm zurückkommen müssen, damit eine Rückkehr der Steuerung zur nächsten Anweisung hinter der PERFORM-Anweisung erfolgen kann. In diesem Fall muß unter Umständen Prozedur-Name-2 in der PERFORM-Anweisung der Name eines (entweder einen EXIT-Satz oder keine Sätze enthaltenden) Paragraphen sein. Ein Paragraph, der keine Sätze enthält, wird Null-Paragraph genannt. Damit eine Rückkehr der Steuerung zur nächsten Anweisung hinter der PERFORM-Anweisung erfolgen kann, endet jeder der zwei oder mehr eingeschlagenen Wege wiederum mit einer GO TO-Anweisung zum Paragraphen-Namen des EXIT-Satzes oder zum Null-Paragraphen.

In Beispiel 6 liefert die EXIT-Anweisung im Paragraphen ABFRAGE-EXIT einen gemeinsamen Endpunkt für eine Folge von Paragraphen, die durch die PERFORM-Anweisung aufgerufen werden. Bei der Ausführung der PERFORM-Anweisung dient die EXIT-Anweisung für eine Rückkehr zur nächsten Anweisung hinter der PERFORM-Anweisung (die ADD-Anweisung).

Beispiel 6:

```
PRUEFUNG.    PERFORM ABFRAGE THRU ABFRAGE-EXIT.
    ADD MENGE-1  MENGE-2  GIVING TOTAL-MENGE.

ABFRAGE.
    IF CODE-1 = 12 GO TO PRUEF-B.
    IF CODE-1 = 15 GO TO PRUEF-C.
    IF CODE-1 = 16 GO TO ABFRAGE-EXIT
PRUEF-A.    ----------------------------
            ----------------------------
            ----------------------------
    GO TO ABFRAGE-EXIT.
PRUEF-B.    ----------------------------
            ----------------------------
            ----------------------------
    GO TO ABFRAGE-EXIT.
PRUEF-C.    ----------------------------
            ----------------------------
            ----------------------------
ABFRAGE-EXIT.    EXIT.
UNTERPROG-A. ---------------------------
```

In Beispiel 7 liefert der Null-Paragraph ABFR-ENDE einen gemeinsamen Endpunkt für eine Folge von Paragraphen, die durch die PERFORM-Anweisung aktiviert werden. Bei Ausführung der PERFORM-Anweisung wird der Null-Paragraph für eine Rückkehr zur nächsten Anweisung hinter der PERFORM-Anweisung (die ADD-Anweisung) verwendet. Wird der Null-Paragraph ausgeführt, kehrt die Steuerung zur nächsten Anweisung hinter der PERFORM-Anweisung zurück.

Beispiel 7:

```
PRUEFUNG.    PERFORM ABFRAGE THRU ABFR-ENDE.
    ADD MENGE-1  MENGE-2  GIVING  TOTAL-MENGE.
             -------------------------

ABFRAGE.
    IF CODE-1 = 12 GO TO PRUEF-B.
    IF CODE-1 = 15 GO TO PRUEF-C.
    IF CODE-1 = 16 GO TO ABFR-ENDE.
PRUEF-A.
    MOVE     -------------------------
             -------------------------
    GO TO ABFR-ENDE.
PRUEF-B.
    ADD      -------------------------
             -------------------------
    GO TO ABFR-ENDE.
PRUEF-C.
    SUBTRACT ------------------------
             -------------------------
ABFR-ENDE.
LESEN.
    READ ASATZ AT END CLOSE XDATEI STOP RUN.
```

Eine PERFORM-Anweisung, die in einem eine Segment-Nummer unter 50 aufweisenden Kapitel auftritt, kann zusätzlich zu jedem DECLARATIVE-Kapitel, dessen Ausführung ansteht, lediglich einschließen:

- Kapitel und/oder Paragraphen, die vollständig in einem oder mehreren, eine Segment-Nummer unter 50 aufweisenden Segmenten enthalten sind.

- Kapitel und/oder Paragraphen, die vollständig in einem einzelnen, eine Segment-Nummer über 49 aufweisenden Segment enthalten sind.

Eine PERFORM-Anweisung, die in einem eine Segment-Nummer über 49 aufweisenden Kapitel auftritt, kann zusätzlich zu jedem DECLARATIVE-Kapitel, dessen Ausführung ansteht, lediglich einschließen:

- Kapitel und/oder Paragraphen, die vollständig in einem oder mehreren, eine Segment-Nummer unter 50 aufweisenden Segmenten enthalten sind.

- Kapitel und/oder Paragraphen, die vollständig in dem gleichen unabhängigen Segment enthalten sind, in dem sich die PERFORM-Anweisung befindet.

Worauf zu achten ist:

1. Wird im Zuge der Ausführung eines PERFORM A THRU B das Ende der Prozedur B logisch nicht erreicht oder übergangen, bleibt die gespeicherte Rückkehradresse in der einschlägigen Software-Tabelle erhalten und kann von dort aus ggf. unkontrolliert wirksam werden.

2. Wird in eine Logik, die am Ende einen EXIT- oder Null-Paragraphen aufweist, nicht mittels Perform, sondern mit einem GO TO verzweigt, wird der EXIT- bzw. Null-Paragraph schlichtweg überlaufen und die nachfolgenden Prozeduren gelangen zur Ausführung! (Dies kann allerdings ggf. beabsichtigt sein.)

DIE PERFORM-ANWEISUNG (TIMES-Variante)

Die TIMES-Variante der PERFORM-Anweisung erlaubt die Abzweigung aus der normalen Befehlsfolge, damit zwischendurch eine oder mehrere Prozeduren wiederholt durchgeführt werden können.

Format:

```
PERFORM Proz.-Name-1 [ {THROUGH} Proz.-Name-2 ] {Identifier} TIMES
                     {THRU    }                 {Ganzzahl  }
   [unbedingte Anweisung END-PERFORM]
```

Jeder Prozedur-Name in der PERFORM-Anweisung muß der Name eines Kapitels oder eines Paragraphen in der PROCEDURE DIVISION sein.

Bei der TIMES-Variante der PERFORM-Anweisung werden die Prozeduren so oft wiederholt, wie dies durch die Ganzzahl oder durch den Inhalt des Identifiers festgelegt ist. Der Identifier stellt ein in der DATA DIVISION beschriebenes numerisches Elementarfeld dar. Dieses numerische Elementarfeld muß eine Ganzzahl beinhalten. Der Ausdruck "Ganzzahl" bezeichnet ein numerisches Literal, das eine Ganzzahl, d.h. ohne Vorzeichen sein muß und dessen Wert nicht Null sein darf.

Enthält der Identifier einen Null-Wert oder einen negativen Wert, dann geht die Steuerung zur nächsten Anweisung hinter der PERFORM-Anweisung über.

Beispiel 1:

```
PERFORM ADDITION-ROUTINE 3 TIMES.
```

Diese PERFORM-Anweisung führt die Prozedur ADDITION-ROUTINE dreimal aus.

Beispiel 2:

```
PERFORM ABC THRU EFG 30 TIMES.
```

Diese PERFORM-Anweisung führt die mit ABC beginnenden und mit EFG endenden Prozeduren dreißigmal aus.

DIE PERFORM-ANWEISUNG (UNTIL-Variante)

Die UNTIL-Variante der PERFORM-Anweisung erlaubt die Abzweigung aus der normalen Befehlsfolge, damit zwischendurch eine oder mehrere Prozeduren so oft ausgeführt werden können, bis (UNTIL) eine angegebene Bedingung erfüllt ist.

Format:

```
PERFORM  Prozedur-Name-1 [{THROUGH} Prozedur-Name-2]
                         {THRU   }
[WITH TEST {BEFORE}] UNTIL Bedingung
           {AFTER }
  [unbedingte Anweisung END-PERFORM]
```

Jeder Prozedur-Name in der PERFORM-Anweisung muß der Name eines Kapitels oder eines Paragraphen in der PROCEDURE DIVISION sein.

Bei der UNTIL-Variante der PERFORM-Anweisung werden die Prozeduren so oft ausgeführt, bis die in der UNTIL-Angabe festgelegte Bedingung erfüllt ist: eine einfache Bedingung oder eine komplexe Bedingung (Vergleichs-Bedingung, Klassen-Bedingung, Vorzeichen-Bedingung, Schalter-Zustands-Bedingung, Bedingungsnamen-Bedingung, Mehrfach-Bedingung, verneinte Bedingung).

Ist die Bedingung erfüllt, geht die Steuerung zur nächsten Anweisung hinter der PERFORM-Anweisung über. Ist die Bedingung nicht erfüllt, werden die in der PERFORM-Anweisung spezifizierten Prozeduren ausgeführt. Ist die in der UNTIL-Angabe festgelegte Bedingung bereits bei Beginn der PERFORM-Anweisung erfüllt, werden die spezifizierten Prozeduren nicht ausgeführt und die Steuerung geht direkt zur nächsten Anweisung hinter der PERFORM-Anweisung über.

Beispiel 1:

```
PERFORM EDIT-ROUTINE UNTIL TCODE = 5.
```

Die PERFORM-Anweisung führt die Prozedur EDIT-ROUTINE so oft aus, bis der Inhalt im Feld TCODE gleich 5 ist. Dabei wird diese Bedingung stets zu Beginn von EDIT-ROUTINE geprüft, als wäre TEST BEFORE angegeben worden (= Default).

Beispiel 2:

```
PERFORM EDIT-ROUTINE UNTIL ARB-NR IS EQUAL TO HAUPT-NR.
```

Die PERFORM-Anweisung führt die Prozedur EDIT-ROUTINE so oft aus, bis die Inhalte der Felder ARB-NR und HAUPT-NR gleich sind. Dabei wird diese Bedingung stets zu Beginn von EDIT-ROUTINE geprüft, als wäre TEST BEFORE angegeben worden (=Default).

Die PERFORM-Anweisung (VARYING...AFTER...-Variante)

Diese Anweisung gestattet zählerkontrollierte, hierarchische Schleifendurchläufe bis zu 13 Stufen.

Format:

```
PERFORM  Prozedur-Name-1 [ {THROUGH} Prozedur-Name-2 ]
                         {THRU   }

[ WITH TEST {BEFORE} ]
            {AFTER  }

  VARYING {Identifier-2} FROM {Identifier-3}
          {Index-Name-1}      {Index-Name-2}
                              {Literal-1   }

  BY {Identifier-4} UNTIL Bedingung-1
     {Literal-2   }

[ AFTER {Identifier-5} FROM {Identifier-6}
        {Index-Name-3}      {Index-Name-4}
                            {Literal-3   }

  BY {Identifier-7} UNTIL Bedingung-2 ] ...
     {Literal-4   }

[ unbedingte Anweisung END-PERFORM ]
```

Bei Nichtverwendung von Prozedur-Name-1 ist auch die AFTER-Klausel nicht gestattet. Ansonsten können bis zu 12 AFTER-Klauseln in einer PERFORM-Anweisung angewendet werden.

Wird weder TEST BEFORE noch TEST AFTER angegeben, gilt als Default TEST BEFORE, das heißt, daß die jeweilige Überprüfung der Bedingung immer zu Beginn der mehrfach aufzurufenden Logik erfolgt.

Anschauliche Beispiele für die Anwendung dieser Anweisung anhand von Tabellenbearbeitungen finden Sie in den folgenden Kapiteln!

DIE PERFORM-VARYING-ANWEISUNG
(Variante zur Behandlung eines Tabellen-Indizes bzw. -Identifiers

Die VARYING-Angabe in einer PERFORM-Anweisung gestattet es, den Inhalt eines Indizes oder eines Datenfeldes während der mehrfachen Ausführung einer oder mehrerer Prozeduren fortlaufend zu erhöhen oder zu vermindern und somit die Anzahl der Durchläufe zu kontrollieren.

Format:

```
PERFORM Prozedure-Name-1 [ { THROUGH } Prozedure-Name-2 ]
                         { THRU    }

[ WITH TEST { BEFORE } ]
            { AFTER  }

VARYING { Identifier-1 } FROM { Identifier-2 } BY { Identifier-3 }
        { Index-Name-1 }      { Index-Name-2 }    { Literal-2    }
                              { Literal-1    }

UNTIL Bedingung 1
```

Jeder Prozedur-Name in der PERFORM-Anweisung muß der Name eines Kapitels oder eines Paragraphen in der PROCEDURE DIVISION sein. Jeder Identifier in der PERFORM-Anweisung muß ein in der DATA DIVISION beschriebenes numerisches Elementarfeld sein. Jedes Literal muß ein numerisches Literal sein. Bedingung-1 kann irgendein bedingter Ausdruck sein.

Die VARYING-Angabe spezifiziert Index-Name-1 oder Identifier-1 als Index oder das Datenfeld, dessen Inhalt während der Ausführung der PERFORM-Anweisung erhöht oder vermindert wird.

Die FROM-Angabe spezifiziert den Inhalt, auf den der Index oder das Datenfeld bei Beginn der PERFORM-Anweisung zu setzen ist. Der Wert von Literal-1, der Inhalt von Index-Name-2 oder der Inhalt von Identifier-2 stellt den in Index-Name-1 oder Identifier-1 zu Beginn anzusetzenden Zählerstand dar.

Die BY-Angabe spezifiziert den Wert, um den Index-Name-1 oder Identifier-1 bei jeder weiteren Ausführung der PERFORM-Anweisung erhöht werden soll. Der Wert von Literal-2 oder der Inhalt von Identifier-3 spezifiziert dabei den Wert, um den erhöht werden soll. Ist Literal-2 oder der Inhalt von Identifier-3 positiv, dann wird Index-Name-1 oder Identifier-1 erhöht. Ist Literal-2 oder der Inhalt von Identifier-3 negativ, dann wird Index-Name-1 oder Identifier-1 vermindert. Der Wert von Literal-2 oder Identifier-3 darf nicht Null sein.

Die UNTIL-Angabe spezifiziert die Bedingung, die erfüllt sein muß, damit die Ausführung der Prozeduren beendet wird. Ist die in der UNTIL-Angabe spezifizierte Bedingung erfüllt, dann geht die Steuerung zur nächsten Anweisung nach der PERFORM-Anweisung über. Ist die in der UNTIL-Angabe spezifizierte Bedingung nicht erfüllt, werden die in der PERFORM-Anweisung festgelegten Prozeduren mit entsprechender Erhöhung von Index-Name-1 oder Identifier-1 ausgeführt.

Ist Index-Name-1 in der VARYING-Angabe der PERFORM-Anweisung spezifiziert, dann gelten folgende Regeln:

- Der Identifier oder das Literal in der FROM-Angabe müssen einen positiven ganzzahligen Wert aufweisen.

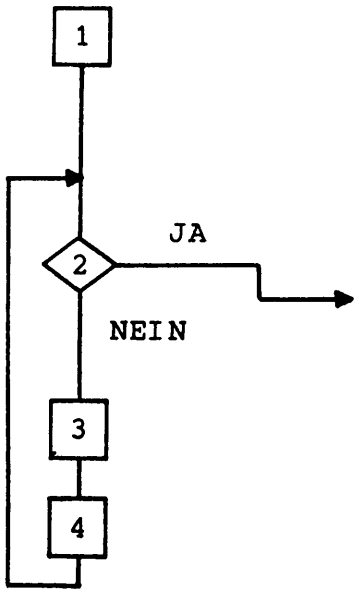
- Der Wert des Identifiers oder des Literals in der BY-Angabe darf nicht Null sein.

Ist Index-Name-2 in der FROM-Angabe der PERFORM-Anweisung spezifiziert, dann gelten folgende Regeln:

- Identifier-1 in der VARYING-Angabe muß ein ganzzahliges Datenfeld sein.

- Der Wert des Identifiers oder des Literals in der BY-Angabe darf nicht Null sein.

Das nachfolgende Diagramm zeigt die während der Ausführung einer PERFORM-VARYING-Anweisung mit einem Index oder Datenfeld erfolgenden Schritte (dabei wird als Default - im Befehl nicht angegeben - TEST BEFORE praktiziert):



Inhalt von Identifier-2 (oder Index-Name-2 oder Literal-1) in Index-Name-1 (oder Identifier-1) einsetzen (= Index oder Datenfeld mit FROM-Wert initialisieren).

Bedingung-1 erfüllt?
Zum nächsten Befehl nach der PERFORM-Anweisung.

Prozedur-1 THRU Prozedur-2 durchführen.

Index-Name-1 (oder Identifier-1) um den Wert von Literal-2 (oder Identifier-3) erhöhen (= Index oder Datenfeld um den BY-Wert erhöhen).

Bei der Ausführung einer PERFORM-VARYING-Anweisung mit nur einem Index oder Subskript wird zuerst der Inhalt von Identifier-2 (oder Index-Name-2 oder Literal-1) in Index-Name-1 (oder Identifier-1) übertragen. Erscheint Index-Name-1 in der PERFORM-VARYING-Anweisung, dann wird dieser Index nach den Regeln der SET-Anweisung initialisiert. Erscheint Index-Name-2 in der FROM-Angabe und Identifier-1 in der PERFORM-VARYING-Anweisung, dann wird Identifier-1 nach den Regeln der SET-Anweisung initialisiert. Das Initialisieren von Index-Name-1 oder Identifier-1 wird bei Beginn der PERFORM-VARYING-Anweisung nur einmal ausgeführt.

Der durch die PERFORM-VARYING-Anweisung ausgelöste zweite Schritt ist vorgesehen, um herauszufinden, ob Bedingung-1 erfüllt ist. Ist Bedingung-1 erfüllt, wird die Steuerung zur nächsten Anweisung nach der PERFORM-Anweisung übertragen. Ist Bedingung-1 nicht erfüllt, werden die durch die PERFORM-VARYING-Anweisung spezifizierten Prozeduren ausgeführt.

Nachdem die durch Prozedur-Name-1 THROUGH Prozedur-Name-2 spezifizierten Prozeduren ausgeführt wurden, wird der Wert von Index-Name-1 (oder Identifier-1) um den Inhalt von Literal-2 (oder Identifier-3) erhöht. Erscheint Index-Name-1 in der PERFORM-VARYING-Anweisung, dann wird dieser Index nach den Regeln der SET-Anweisung erhöht.

Nach Erhöhung von Index-Name-1 oder Identifier-1 geht der Ausführungszyklus der PERFORM-VARYING-Anweisung wieder zum zweiten Schritt zurück, wo geprüft wird, ob Bedingung-1 erfüllt ist. Ist Bedingung-1 erfüllt, wird die Steuerung zur nächsten Anweisung nach dem PERFORM übertragen. Ist Bedingung-1 nicht erfüllt, wird die durch Prozedur-Name-1 THROUGH Prozedur-Name-2 spezifizierte Befehlsfolge erneut ausgeführt, was wiederum die Erhöhung von Index-Name-1 oder Identifier-1 nach sich zieht. Somit wird die PERFORM-VARYING-Anweisung die Ausführung der Prozeduren so lange fortsetzen und den Inhalt des Indexes oder Datenfeldes parallel dazu erhöhen, bis Bedingung-1 erfüllt ist.

Während der Ausführung der Prozeduren wird jede Änderung der VARYING-Variablen (Identifier-1 oder Index-Name-1), der FROM-Variablen (Identifier-2 oder Index-Name-2) oder der BY-Variablen (Identifier-3) berücksichtigt und hat ihre Auswirkung auf den internen Ablauf der PERFORM-VARYING-Anweisung.

Die PERFORM-VARYING-Anweisung mit einem Index oder Datenfeld kann zum sequentiellen Zugriff auf die Datenfelder in einer 1-dimensionalen Tabelle verwendet werden. Beispiel 2 zeigt eine PERFORM-VARYING-Anweisung mit einem Datenfeld (einem Subscript), durch das der Zugriff auf eine 1-dimensionale Tabelle erfolgt. Beispiel 1 zeigt eine PERFORM-VARYING-Anweisung mit einem Index, durch den der Zugriff auf eine 1-dimensionale Tabelle erfolgt.

Beispiel 1:

```
FILE SECTION.
FD  DATEI1 BLOCK CONTAINS 15 RECORDS
        RECORD CONTAINS 34 CHARACTERS
        LABEL RECORD IS STANDARD
        VALUE OF FILE-ID IS "DATEI1".
01  SATZ1.
    02  ABTLG-NR  PIC XXXX.
    02  TAGES-UMSATZ  PIC 9999V99 OCCURS 5 TIMES INDEXED BY
                                              TAG-INDEX.

PROCEDURE DIVISION.
    PERFORM KUMULATION VARYING TAG-INDEX FROM 1 BY 1
            UNTIL TAG-INDEX IS GREATER THAN 5.

KUMULATION.
    ADD TAGES-UMSATZ (TAG-INDEX) TO GESAMT-UMSATZ.
```

Im Beispiel 1 ist TAG-INDEX ein Index, der benutzt wird zum Zugriff auf eine 1-dimensionale Tabelle. Die PERFORM-VARYING-Anweisung addiert den jeweiligen TAGES-UMSATZ in das Datenfeld GESAMT-UMSATZ. Die folgende Übersicht enthält die fünf ADD-Anweisungen, die in diesem Beispiel von der PERFORM-VARYING-Anweisung durchgeführt werden:

ADD-Anweisung	Verarbeiteter Tabellenposten Tag
ADD TAGES-UMSATZ (1) TO GESAMT-UMSATZ.	1
ADD TAGES-UMSATZ (2) TO GESAMT-UMSATZ.	2
ADD TAGES-UMSATZ (3) TO GESAMT-UMSATZ.	3
ADD TAGES-UMSATZ (4) TO GESAMT-UMSATZ.	4
ADD TAGES-UMSATZ (5) TO GESAMT-UMSATZ.	5

Beispiel 2:

```
FILE SECTION.
FD  DATEI1 BLOCK CONTAINS 15 RECORDS
        RECORD CONTAINS 34 CHARACTERS
        LABEL RECORD IS STANDARD
        VALUE OF FILE-ID IS "DATEI1".
01  SATZ1.
    02  ABTLG-NR  PIC XXXX.
    02  TAGES-UMSATZ  PIC 9999V99 OCCURS 5 TIMES.

WORKING-STORAGE SECTION.
77  TAG-SUB PIC 9.

PROCEDURE DIVISION.
    PERFORM KUMULATION VARYING TAG-SUB FROM 1 BY 1
            UNTIL TAG-SUB IS GREATER THAN 5.

KUMULATION.
    ADD TAGES-UMSATZ (TAG-SUB) TO GESAMT-UMSATZ.
```

Im Beispiel 2 ist TAG-SUB ein Subscript, das benutzt wird zum Zugriff auf eine 1-dimensionale Tabelle. Die PERFORM-VARYING-Anweisung addiert den jeweiligen TAGES-UMSATZ in das Datenfeld GESAMT-UMSATZ. Die Übersicht oben enthält die fünf ADD-Anweisungen, die in diesem Beispiel von der PERFROM-VARYING-Anweisung ausgeführt werden.

DIE PERFORM-VARYING-ANWEISUNG
(Variante zur Behandlung zweier Tabellen-Indizes bzw. -Identifiers)

Die VARYING-Angabe mit zwei Bedingungen in einer PERFORM-Anweisung ermöglicht, daß der Inhalt zweier Indizes oder zweier Datenfelder während der mehrfachen Ausführung einer oder mehrerer Prozeduren fortlaufend erhöht oder vermindert wird und somit die Anzahl der Durchläufe kontrollierbar ist.

Format:

```
PERFORM  Prozedure-Name-1 [ { THROUGH } Prozedure-Name-2 ]
                           { THRU    }
[ WITH TEST { BEFORE } ]
            { AFTER  }
VARYING { Identifier-1 } FROM { Identifier-2 } BY { Identifier-3 }
        { Index-Name-1 }      { Index-Name-2 }    { Literal-2    }
                              { Literal-1    }
     UNTIL  Bedingung-1
AFTER   { Identifier-4 } FROM { Identifier-5 } BY { Identifier-6 }
        { Index-Name-3 }      { Index-Name-4 }    { Literal-4    }
                              { Literal-3    }
     UNTIL  Bedingung-2
```

Jeder Prozedur-Name in der PERFORM-Anweisung muß der Name eines Kapitels oder eines Paragraphen in der PROCEDURE DIVISION sein. Jeder Identifier in der PERFORM-Anweisung muß ein in der DATA DIVISION beschriebenes numerisches Elementarfeld sein. Jedes Literal muß ein numerisches Literal sein. Bedingung-1 und Bedingung-2 können beliebig bedingte Ausdrücke sein.

Die VARYING-Angabe spezifiziert Index-Name-1 als den Index auf der höheren Stufe bzw. Identifier-1 als das Datenfeld auf der höheren Stufe, deren Inhalte während der Ausführung der PERFORM-Anweisung erhöht oder vermindert werden.

Die AFTER-Angabe spezifiziert Index-Name-3 als den Index auf der niedrigeren Stufe bzw. Identifier-4 als das Datenfeld auf der niedrigeren Stufe, deren Inhalte während der Ausführung der PERFORM-Anweisung erhöht oder vermindert werden.

Bei der Ausführung einer PERFORM-VARYING-Anweisung mit einer AFTER-Angabe wird der in der AFTER-Angabe genannte Index der niedrigeren Stufe bzw. das dort genannte Datenfeld der niedrigeren Stufe erhöht und es wird ermittelt, ob Bedingung-2 erfüllt ist. Daraufhin wird der in der VARYING-Angabe enthaltene Index bzw. das Datenfeld auf der höheren Stufe erhöht und es wird geprüft, ob Bedingung-1 erfüllt ist. Der vollständige logische Ablauf dieser Variante der PERFORM-Anweisung wird auf den nächsten Seiten gezeigt.

Die FROM-Angaben spezifizieren die Werte, auf die die Indizes oder Datenfelder zu Beginn der PERFORM-Anweisung zu setzen sind. Der Wert von Literal-1 (oder Literal-3), der Inhalt von Index-Name-2 (oder Index-Name-4) oder der Inhalt von Identifier-2 (oder Identifier-5) zeigen den im Index (Index-Name-1 oder Index-Name-3) oder Datenfeld (Identifier-1 oder Identifier-4) zu setzenden Grundwert an.

Die BY-Angaben spezifizieren die Werte, um die die Indizes oder Datenfelder während der Ausführung der PERFORM-Anweisung zu erhöhen sind. Der Wert von Literal-2 (oder Literal-4) oder der Inhalt von Identifier-3 (oder Identifier-6) spezifizieren den Wert, um den der zugeordnete Index (Index-Name-1 oder Index-Name-3) oder das zugeordnete Datenfeld (Identifier-1 oder Identifier-4) erhöht werden soll. Ist Literal-2 (oder Literal-4) oder der Inhalt von Identifier-3 (oder Identifier-6) positiv, dann wird Index-Name-1 (oder Index-Name-3) oder Identifier-1 (oder Identifier-4) erhöht. Ist Literal-2 (oder Literal-4) oder der Inhalt von Identifier-3 (oder Identifier-6) negativ, dann wird der zugeordnete Index oder das zugeordnete Datenfeld vermindert. Der Wert von Literal-2 (oder Literal-4) oder Identifier-3 (oder Identifier-6) darf nicht Null sein.

Die UNTIL-Angabe spezifiziert die Bedingung, die erfüllt sein muß, damit das Verändern des Indexes oder Datenfeldes beendet wird. In einer PERFORM-VARYING-Anweisung mit zwei Bedingungen steht Bedingung-2 im Zusammenhang mit der Erhöhung des Indexes (Index-Name-3) oder des Datenfeldes der niedrigeren Stufe (Identifier-4). Bedingung-1 steht im Zusammenhang mit der Erhöhung des Indexes (Index-Name-1) oder des Datenfeldes der höheren Stufe (Identifier-1). Das Diagramm zeigt die Position von Bedingung-1 und Bedingung-2 im internen Ablauf der PERFORM-VARYING-Anweisung.

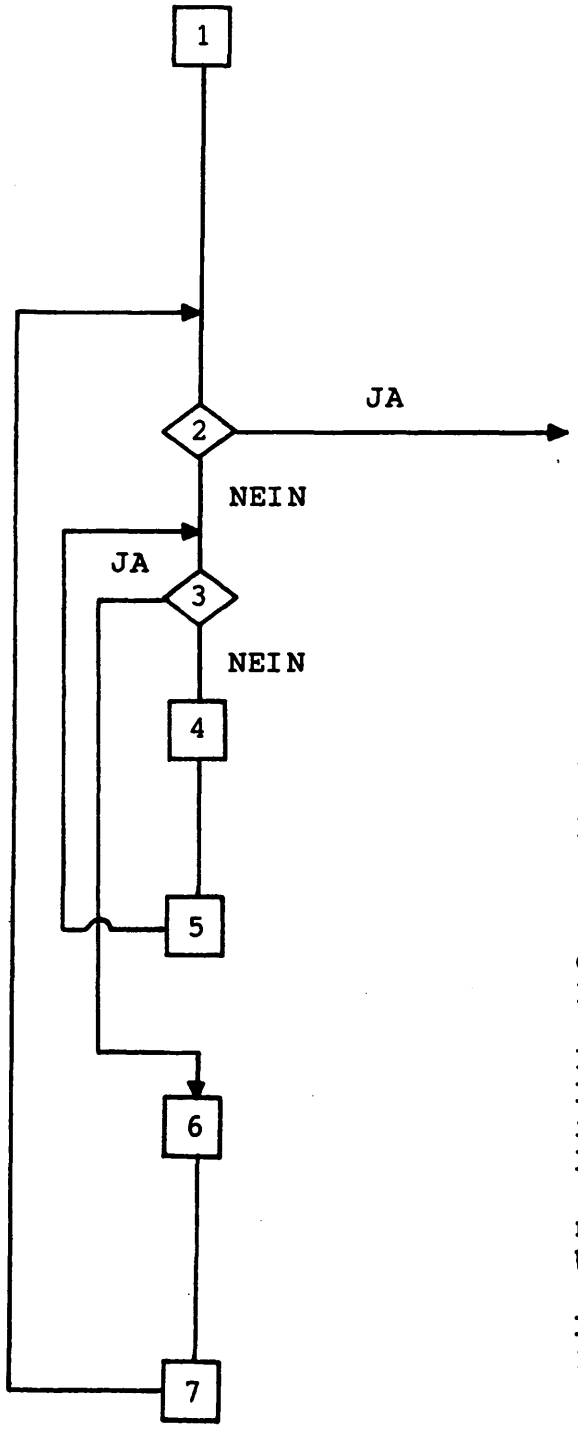
Ist Index-Name-1 in der VARYING-Angabe bzw. Index-Name-3 in der AFTER-Angabe der PERFORM-Anweisung spezifiziert, dann gelten folgende Regeln:

- Der Identifier oder das Literal in der jeweiligen FROM-Angabe müssen einen positiven ganzzahligen Wert aufweisen.
- Der Wert des Identifiers oder des Literals in der jeweiligen BY-Angabe darf nicht Null sein.

Ist Index-Name-2 (oder Index-Name-4) in der jeweiligen FROM-Angabe der PERFORM-Anweisung spezifiziert, dann gelten folgende Regeln:

- Der Identifier-1 in der VARYING-Angabe bzw. der Identifier-4 in der AFTER-Angabe müssen ein ganzzahliges Datenfeld sein.
- Der Wert des Identifiers oder des Literals in der jeweiligen BY-Angabe darf nicht Null sein.

Das Diagramm auf der nächsten Seite zeigt die während der Ausführung einer PERFORM-VARYING-Anweisung mit zwei Indizes oder zwei Datenfeldern erfolgenden Schritte (dabei wird als Default - im Befehl nicht angegeben - TEST BEFORE praktiziert).



Inhalt von Identifier-2 (oder Index-Name-2 oder Literal-1) in Index-Name-1 (oder Identifier-1) einsetzen; laufenden Wert von Identifier-5 (oder Index-Name-4 oder Literal-3) in Index-Name-3 (oder Identifier-4) einsetzen (= Indizes (oder Subskripte) mit FROM-Wert initialisieren).

Bedingung-1 erfüllt?
Zum nächsten Befehl nach der PERFORM-Anweisung.

Bedingung-2 erfüllt?

Prozedur-1 THRU Prozedur-2 durchführen.

Index-Name-3 (oder Identifier-4) um den Wert von Literal-4 (oder Identifier-6) erhöhen (= Index oder Subskript auf der niedrigeren Stufe um den BY-Wert erhöhen).

Inhalt von Identifier-5 (oder Index-Name-4 oder Literal-3) in Index-Name-3 (oder Identifier-4) einsetzen (= Index oder Subskript der niedrigeren Stufe mit FROM-Wert neu initialisieren).

Index-Name-1 (oder Identifier-1) um den Wert von Literal-2 (oder Identifier-3) erhöhen (= Index oder Subskript der höheren Stufe um den BY-Wert erhöhen).

Bei der Ausführung einer PERFORM-VARYING-Anweisung mit zwei Indizes oder zwei Subskriptfeldern besteht die erste vorgenommene Aktion darin, den Wert von Identifier-2 (oder Index-Name-2 oder Literal-1) in Index-Name-1 (oder Identifier-1) einzusetzen. Diese erste Aktion schließt auch ein, daß der Wert von Identifier-5 (oder Index-Name-4 oder Literal-3) in Index-Name-3 (oder Identifier-4) eingesetzt wird. Erscheint Index-Name-1 oder Index-Name-3 in der PERFORM-VARYING-Anweisung, dann wird dieser Index nach den Regeln der SET-Anweisung initialisiert. Erscheint Index-Name-2 (oder Index-Name-4) in der FROM-Angabe und Identifier-1 (oder Identifier-4) in der PERFORM-VARYING-Anweisung, dann wird Identifier-1 (oder Identifier-4) nach den Regeln der SET-Anweisung initialisiert. Das Initialisieren wird bei Beginn der PERFORM-VARYING-Anweisung jeweils einmal ausgeführt.

Der durch die PERFORM-VARYING-Anweisung ausgelöste zweite Schritt besteht darin, herauszufinden, ob Bedingung-1 erfüllt ist. Ist Bedingung-1 erfüllt, wird die Steuerung an die nächste Anweisung hinter der PERFORM-Anweisung übertragen. Ist Bedingung-1 nicht erfüllt, dann geht, wie das Diagramm zeigt, der logische Ablauf zur dritten Aktion weiter.

Die durch die PERFORM-VARYING-Anweisung ausgelöste dritte Aktion besteht darin, herauszufinden, ob Bedingung-2 erfüllt ist. Ist Bedingung-2 nicht erfüllt, werden die durch die PERFORM-VARYING-Anweisung spezifizierten Prozeduren ausgeführt. Ist Bedingung-2 erfüllt, geht, wie das Diagramm zeigt, der logische Ablauf zur sechsten Aktion der PERFORM-VARYING-Anweisung weiter.

Die vierte Aktion, die durchgeführt wird, wenn Bedingung-2 nicht erfüllt ist, besteht in der Ausführung der Prozeduren. Nach Ausführung dieser Prozeduren wird der Wert von Index-Name-3 (oder Identifier-4) um den Wert von Literal-4 (oder Identifier-6) erhöht. Erscheint Index-Name-3 in der PERFORM-VARYING-Anweisung, dann wird dieser Index nach den Regeln der SET-Anweisung erhöht. Nach Erhöhung von Index-Name-3 (oder Identifier-4) erfolgt, um herauszufinden, ob Bedingung-2 erfüllt ist, ein Rücksprung.

Ist Bedingung-2 erfüllt, geht der logische Ablauf weiter zur sechsten Aktion, wo der Wert von Identifier-5 (oder Index-Name-4 oder Literal-3) in Index-Name-3 (oder Identifier-4) eingesetzt wird. Somit wird, um Zugriff zum ersten Posten der nächsten Tabellendimension zu ermöglichen, der Index oder das Subskript auf der niedrigeren Stufe erneut initialisiert.

Die siebte durch die PERFORM-VARYING-Anweisung ausgelöste Aktion besteht darin, Index-Name-1 (oder Identifier-1) um den Wert von Literal-2 (oder Identifier-3) zu erhöhen. Erscheint Index-Name-1 in der PERFORM-VARYING-Anweisung, dann wird dieser Index nach den Regeln der SET-Anweisung erhöht. Somit erfolgt die Erhöhung des Indexes oder des Subskripts der höheren Stufe. Nach der Erhöhung von Index-Name-1 oder Identifier-1 erfolgt ein Rücksprung zum zweiten Schritt, um herauszufinden, ob Bedingung-1 erfüllt ist.

Der logische Ablauf der PERFORM-VARYING-Anweisung mit zwei Indizes oder zwei Subskripts geht so vor sich, daß der Index oder das Subskript auf der niedrigeren Stufe variiert wird, bis Bedingung-2 erfüllt ist. Ist Bedingung-2 erfüllt, wird der Index oder das Subskript der höheren Stufe so lange variiert, bis Bedingung-1 erfüllt ist.

Während der Ausführung der Prozeduren wird jede Änderung der VARYING-Variablen (Identifier-1 oder Index-Name-1), der AFTER-Variablen (Identifier-4 oder Index-Name-3), der FROM-Variablen (Identifier-2, Index-Name-2, Identifier-5 oder Index-Name-4) oder der BY-Variablen (Identifier-3 oder Identifier-6) berücksichtigt und hat somit ihre Auswirkung auf den internen Ablauf der PERFORM-VARYING-Anweisung.

Die PERFORM-VARYING-Anweisung mit zwei Indizes oder zwei Subskripts kann zum sequentiellen Zugriff auf die Datenfelder einer 2-dimensionalen Tabelle verwendet werden. Das folgende Beispiel zeigt eine PERFORM-VARYING-Anweisung mit zwei Indizes, durch die Zugriff auf eine 2-dimensionale Tabelle erfolgt. Das nachfolgende Beispiel 2 zeigt eine PERFORM-VARYING-Anweisung mit zwei Datenfeldern (Subscripten), durch die Zugriff auf eine zwei-dimensionale Tabelle erfolgt.

Beispiel 1:

```
FILE SECTION.
FD DATEI2 BLOCK CONTAINS 3 RECORDS
       RECORD CONTAINS 136 CHARACTERS
       LABEL RECORD IS STANDARD
       VALUE OF FILE-ID IS "DATEI2".
01  SATZ2.
    02  ABTLG  OCCURS 4 TIMES INDEXED BY ABT-INDEX.
        03  ABTLG-NR  PIC XXXX.
        03  TAGES-UMSATZ  PIC 9999V99 OCCURS 5 TIMES
                INDEXED BY TAG-INDEX.
```

```
PROCEDURE DIVISION.
    PERFORM KUMULATION VARYING ABT-INDEX FROM 1 BY 1
            UNTIL ABT-INDEX IS GREATER THAN 4
        AFTER TAG-INDEX FROM 1 BY 1
            UNTIL TAG-INDEX IS GREATER THAN 5.

KUMULATION.
    ADD TAGES-UMSAWTZ (ABT-INDEX, TAG-INDEX) TO GESAMT-UMSATZ.
```

In diesem Beispiel ist ABT-INDEX der Index auf der höheren Stufe, der zum Zugriff auf den mit dem Daten-Namen ABTLG bezeichneten Tabellenposten der ersten Dimension benutzt wird. TAG-INDEX ist der Index auf der niedrigeren Stufe, der zum Zugriff auf den mit dem Daten-Namen TAGES-UMSATZ bezeichneten Tabellenposten der zweiten Dimension benutzt wird. Die PERFORM-VARYING-Anweisung bewirkt, daß die Tages-Umsätze der einzelnen Abteilungen in das Datenfeld GESAMT-UMSATZ addiert werden. Die folgende Übersicht enthält die zwanzig ADD-Anweisungen, die die PERFORM-VARYING-Anweisung intern durchführt:

ADD-Anweisung	Verarbeitender Tabellenposten	
	Abteilung	Tag
ADD TAGES-UMSATZ (1, 1) TO GESAMT-UMSATZ	1	1
ADD TAGES-UMSATZ (1, 2) TO GESAMT-UMSATZ	1	2
ADD TAGES-UMSATZ (1, 3) TO GESAMT-UMSATZ	1	3
ADD TAGES-UMSATZ (1, 4) TO GESAMT-UMSATZ	1	4
ADD TAGES-UMSATZ (1, 5) TO GESAMT-UMSATZ	1	5
ADD TAGES-UMSATZ (2, 1) TO GESAMT-UMSATZ	2	1
ADD TAGES-UMSATZ (2, 2) TO GESAMT-UMSATZ	2	2
ADD TAGES-UMSATZ (2, 3) TO GESAMT-UMSATZ	2	3
ADD TAGES-UMSATZ (2, 4) TO GESAMT-UMSATZ	2	4
ADD TAGES-UMSATZ (2, 5) TO GESAMT-UMSATZ	2	5
ADD TAGES-UMSATZ (3, 1) TO GESAMT-UMSATZ	3	1
ADD TAGES-UMSATZ (3, 2) TO GESAMT-UMSATZ	3	2
ADD TAGES-UMSATZ (3, 3) TO GESAMT-UMSATZ	3	3
ADD TAGES-UMSATZ (3, 4) TO GESAMT-UMSATZ	3	4
ADD TAGES-UMSATZ (3, 5) TO GESAMT-UMSATZ	3	5
ADD TAGES-UMSATZ (4, 1) TO GESAMT-UMSATZ	4	1
ADD TAGES-UMSATZ (4, 2) TO GESAMT-UMSATZ	4	2
ADD TAGES-UMSATZ (4, 3) TO GESAMT-UMSATZ	4	3
ADD TAGES-UMSATZ (4, 4) TO GESAMT-UMSATZ	4	4
ADD TAGES-UMSATZ (4, 5) TO GESAMT-UMSATZ	4	5

Beispiel 2:

```
FILE SECTION.
    DATEI2  BLOCK CONTAINS 3 RECORDS
    RECORD CONTAINS 136 CHARACTERS
        LABEL RECORD IS STANDARD
        VALUE OF FILE-ID IS "DATEI2".
01  SATZ2.
    02  ABTLG OCCURS 4 TIMES.
        03  ABTLG-NR PIC XXXX.
        03  TAGES-UMSATZ PIC 9999V99 OCCURS 5 TIMES.

WORKING-STORAGE SECTION.
77  TAG-SUB PIC 9.
77  ABT-SUB PIC 9.

PROCEDURE DIVISION.
    PERFORM KUMULATION VARYING ABT-SUB FROM 1 BY 1
            UNTIL ABT-SUB IS GREATER THAN 4
         AFTER TAG-SUB FROM 1 BY 1
            UNTIL TAG-SUB IS GREATER THAN 5.

KUMULATION.
    ADD TAGES-UMSATZ (ABT-SUB, TAG-SUB) TO GESAMT-UMSATZ.
```

In diesem Beispiel ist ABT-SUB das Subscript auf der höheren Stufe der ersten Dimension, das zum Zugriff auf den mit ABTLG bezeichneten Tabellenposten benutzt wird. TAG-SUB ist das Subscript auf der niedrigeren Stufe der zweiten Dimension, das zum Zugriff auf den mit TAGES-UMSATZ bezeichneten Tabellenposten benutzt wird. Die PERFORM-VARYING-Anweisung bewirkt, daß die Tages-Umsätze der einzelnen Abteilungen in das Datenfeld GESAMT-UMSATZ addiert werden. Die vorangehende Übersicht enthält die zwanzig ADD-Anweisungen, die die PERFORM-VARYING-Anweisung in diesem Beispiel intern durchführt.

DIE PERFORM-VARYING-ANWEISUNG (Variante zur Behandlung dreier Tabellen-Indizes bzw. Identifiers)

Mit der PERFORM VARYING AFTER-Anweisung ist es möglich, daß der Wert dreier Indizes oder dreier Datenfelder während der mehrfachen Ausführung einer oder mehrerer Prozeduren fortlaufend erhöht oder vermindert wird und somit die Anzahl der Durchläufe kontrollierbar ist.

Format:

```
PERFORM  Prozedur-Name-1 [ { THROUGH } Prozedur-Name-2 ]
                         { THRU    }
[ WITH TEST { BEFORE } ]
            { AFTER  }

VARYING { Identifier-1 } FROM { Identifier-2 } BY { Identifier-3 }
        { Index-Name-1 }      { Index-Name-2 }    { Literal-2   }
                              { Literal-1    }

    UNTIL   Bedingung-1

AFTER { Identifier-4 } FROM { Identifier-5 } BY { Identifier-6 }
      { Index-Name-3 }      { Index-Name-4 }    { Literal-4   }
                            { Literal-3    }

    UNTIL   Bedingung-2

AFTER { Identifier-7 } FROM { Identifier-8 } BY { Identifier-9 }
      { Index-Name-5 }      { Index-Name-6 }    { Literal-6   }
                            { Literal-5    }

    UNTIL   Bedingung-3
```

Jeder Prozedur-Name in der PERFORM-Anweisung muß der Name eines Kapitels oder eines Paragraphen in der PROCEDURE DIVISION sein. Jeder Identifier in der PERFORM-Anweisung muß ein in der DATA DIVISION beschriebenes numerisches Elementarfeld sein. Jedes Literal muß ein numerisches Literal sein. Bedingung-1, Bedingung-2 und Bedingung-3 können beliebige bedingte Ausdrücke sein.

Die VARYING-Anweisung spezifiziert Index-Name-1 als den Index auf der höchsten Stufe bzw. Identifier-1 als das Datenfeld auf der höchsten Stufe, deren Inhalte während der Ausführung der PERFORM-Anweisung erhöht oder vermindert werden.

Die erste AFTER-Angabe spezifiziert Index-Name-3 als den Index auf der mittleren Stufe bzw. Identifier-4 als das Datenfeld auf der mittleren Stufe, deren Inhalte während der Ausführung der PERFORM-Anweisung erhöht oder vermindert werden.

Die zweite AFTER-Angabe spezifiziert Index-Name-5 als den Index auf der niedrigsten Stufe bzw. Identifier-7 als das Datenfeld auf der niedrigsten Stufe, deren Inhalte während der Ausführung der PERFORM-Anweisung erhöht oder vermindert werden.

Bei der Ausführung einer sowohl die VARYING-Angabe als auch die zwei AFTER-Angaben enthaltenden PERFORM-VARYING-Anweisung wird der in der zweiten AFTER-Angabe genannte Index bzw. das dort genannte Datenfeld der niedrigsten Stufe erhöht und es wird zuerst ermittelt, ob Bedingung-3 erfüllt ist. Daraufhin wird der Index bzw. das Datenfeld auf der mittleren Stufe in der ersten AFTER-Angabe erhöht und es wird ermittelt, ob Bedingung-2 erfüllt ist. Schließlich wird der in der VARYING-Angabe enthaltene Index bzw. das Datenfeld auf der höchsten Stufe erhöht und es wird ermittelt, ob Bedingung-1 erfüllt ist. Der vollständige logische Ablauf dieser Variante der PERFORM-VARYING-Anweisung ist auf den nächsten Seiten gezeigt.

Die FROM-Angaben spezifizieren die Werte, auf die die Indizes oder Datenfelder zu Beginn der PERFORM-Anweisung zu setzen sind. Der Wert von Literal-1, der Inhalt von Index-Name-2 oder der Inhalt von Identifier-2 zeigen den in Index-Namen-1 oder Identifier-1 zu setzenden Grundwert an. Der Wert von Literal-3, der Inhalt von Index-Name-4 oder der Inhalt von Identifier-5 zeigt den in Index-Namen-3 oder Identifier-4 zu setzenden Grundwert an. Der Wert von Literal-5,1 der Inhalt von Index-Name-6 oder der Inhalt von Identifier-8 zeigt den in Index-Namen-5 oder Identifier-7 zu setzenden Grundwert an.

Die BY-Angaben spezifizieren die Werte, um die die Indizes oder Datenfelder während der Ausführung der PERFORM-Anweisung zu erhöhen sind. Der Wert von Literal-2 oder der Inhalt von Identifier-3 spezifizieren den Wert, um den Index-Name-1 oder Identifier-1 erhöht werden soll. Der Wert von Literal-4 oder der Inhalt von Identifier-6 spezifiziert den Wert, um den Index-Name-3 oder Identifier-4 erhöht werden soll. Der Wert von Literal-6 oder der Inhalt von Identifier-9 spezifiziert den Wert, um den Index-Name-5 oder Identifier-7 erhöht werden soll. Ist Literal-2 (oder Literal-4 oder Literal-6) oder der Inhalt von Identifier-3

(oder Identifier-6 oder Identifier-9) positiv, dann wird Index-Name-1 (oder Index-Name-3 oder Index-Name-5) oder Identifier-1 (oder Identifier-4 oder Identifier-7) erhöht. Ist Literal-2 (oder Literal-4 oder Literal-6) oder der Inahlt von Identifier-3 (oder Identifier-6 oder Identifier-9) negativ, dann wird der zugeordnete Index oder das zugeordnete Datenfeld vermindert. Der Wert von Literal-2 (oder Literal-4 oder Literal-6) oder Identifier-3 (oder Identifier-6 oder Identifier-9) darf nicht Null sein.

Die UNTIL-Angabe spezifiziert die Bedingung, die erfüllt sein muß, damit das Verändern des dieser Bedingung zugeordneten Indexes oder Datenfeldes beendet werden kann. In einer PERFORM-VARYING-Anweisung mit drei Bedingungen steht Bedingung-3 im Zusammenhang mit der Erhöhung des Indexes (Index-Name-5) oder des Datenfeldes (Identifier-7) der niedrigsten Stufe. Bedingung-2 steht im Zusammenhang mit der Erhöhung des Indexes (Index-Name-3) oder des Datenfeldes (Identifier-4) der mittleren Stufe. Bedingung-1 steht im Zusammenhang mit der Erhöhung des Indexes (Index-Name-1) oder des Datenfeldes (Identifier-1) der höchsten Stufe. Das Diagramm zeigt die Position von Bedingung-1, Bedingung-2 und Bedingung-3 im internen Ablauf der PERFORM-VARYING-Anweisung.

Ist Index-Name-1 in der VARYING-Angabe bzw. Index-Name-3 oder Index-Name-5 in den AFTER-Angaben der PERFORM-Anweisung spezifiziert, dann gelten folgende Regeln:

- Der Identifier oder das Literal in der jeweiligen FROM-Angabe müssen einen positiven ganzzahligen Wert aufweisen.

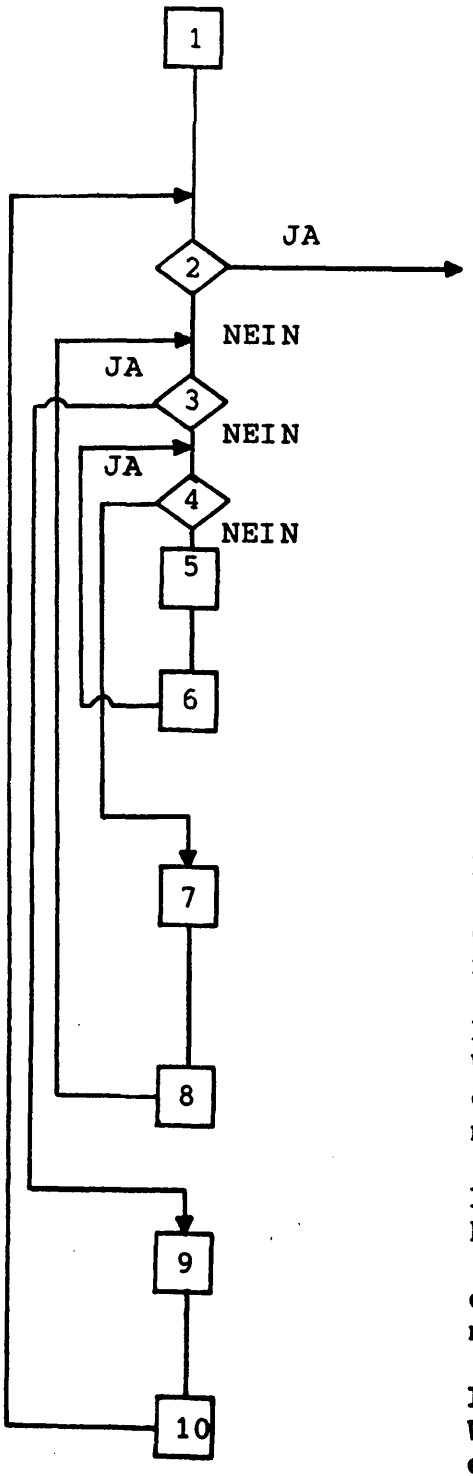
- Der Wert des Identifiers oder des Literals in der jeweiligen BY-Angabe darf nicht Null sein.

Ist Index-Name-2 (oder Index-Name-4 oder Index-Name-6) in der jeweiligen FROM-Angabe der PERFORM-Anweisung spezifiziert, dann gelten folgende Regeln:

- Der Identifier-1 in der VARYING-Angabe bzw. der Identifier-4 oder der Identifier-7 in der jeweiligen AFTER-Angabe müssen ein ganzzahliges Datenfeld sein.

- Der Wert des Identifiers oder des Literals in der jeweiligen BY-Angabe darf nicht Null sein.

Das Diagramm auf der nächsten Seite zeigt die während einer PERFORM-VARYING-Anweisung mit drei Indizes oder drei Datenfeldern erfolgenden Schritte (Default ist - wenn im Befehl nicht angegeben - TEST BEFORE):



Identifier-2 (oder Index-Name-2 oder Literal-1) in Index-Name-1 (oder Identifier-1) einsetzen; Identifier-5 (oder Index-Name-4 oder Literal-3) in Index-Name-3 (oder Identifier-4) einsetzen; Identifier-8 (oder Index-Name-6 oder Literal-5) in Index-Name-5 (oder Identifier-7) einsetzen (= Indizes (oder Subskripts) mit FROM-Wert initialis.)

Bedingung-1 erfüllt?
Zum nächsten Befehl nach der PERFORM-Anweisung.

Bedingung-2 erfüllt?

Bedingung-3 erfüllt?

Prozedur-1 THRU Prozedur-2 durchführen.

Index-Name-5 (oder Identifier-7) um den Wert von Literal-6 (oder Identifier-9) erhöhen (= Index oder Subskript auf der niedrigsten Stufe um den BY-Wert erhöhen).

Inhalt von Identifier-8 (oder Index-Name-6 oder Literal-5) in Index-Name-5 (oder Identifier-7) einsetzen (= Index oder Subskript auf der niedrigsten Stufe mit FROM-Wert neu initialisieren).

Index-Name-3 (oder Identifier-4) um den Wert von Literal-4 (oder Identifier-6) erhöhen (= Index oder Datenfeld auf der mittleren Stufe um den BY-Wert erhöhen).

Inhalt von Identifier-5 (oder Index-Name-4 oder Literal-3) in Index-Name-3 (oder Identifier-4) einsetzen (= Index oder Datenfeld auf der mittleren Stufe mit FROM-Wert neu initialisieren).

Index-Name-1 (oder Identifier-1) um den Wert von Literal-2 (oder Identifier-3) erhöhen (= Index oder Datenfeld auf der höchsten Stufe um den BY-Wert erhöhen).

Bei der Ausführung einer PERFORM-VARYING-Anweisung mit drei Indizes oder drei Datenfeldern (Subskripts) besteht die erste vorgenommene Aktion darin, den Wert von Identifier-2 (oder Index-Name-2 oder Literal-1) in Index-Name-1 (oder Identifier-1) einzusetzen. Diese erste Aktion schließt auch ein, daß der Wert von Identifier-5 (oder Index-Name-4 oder Literal-3) in Index-Name-3 (oder Identifier-4) eingesetzt wird und daß der Wert von Identifier-8 (oder Index-Name-6 oder Literal-5) in Index-Name-5 (oder Identifier-7) eingesetzt wird. Erscheint Index-Name-1, Index-Name-3 oder Index-Name-5 in der PERFORM-VARYING-Anweisung, dann wird dieser Index nach den Regeln der SET-Anweisung initialisiert. Erscheint Index-Name-2 (oder Index-Name-4 oder Index-Name-6) in der FROM-Angabe und Identifier-1 (oder Identifier-4 oder Identifier-7) in der PERFORM-VARYING-Anweisung, dann wird Identifier-1 (oder Identifier-4 oder Identifier-7) nach den Regeln der SET-Anweisung initialisiert. Diese Aktion des Initialisierens wird bei Beginn der PERFORM-VARYING-Anweisung jeweils einmal ausgeführt.

Die durch die PERFORM-VARYING-Anweisung ausgelöste zweite Aktion besteht darin, herauszufinden, ob Bedingung-1 erfüllt ist. Ist Bedingung-1 erfüllt, wird die Steuerung an die nächste Anweisung nach der PERFORM-Anweisung übertragen. Ist Bedingung-1 nicht erfüllt, dann geht, wie das Diagramm zeigt, der logische Ablauf zur dritten Aktion weiter.

Die dritte Aktion besteht darin, herauszufinden, ob Bedingung-2 erfüllt ist. Ist Bedingung-2 erfüllt, dann geht, wie das Diagramm zeigt, der logische Ablauf zur neunten Aktion der PERFORM-VARYING-Anweisung weiter. Ist Bedingung-2 nicht erfüllt, dann geht, wie das Diagramm zeigt, der logische Ablauf zur vierten Aktion weiter.

Die vierte Aktion besteht darin, herauszufinden, ob Bedingung-3 erfüllt ist. Ist Bedingung-3 erfüllt, dann geht, wie das Diagramm zeigt, der logische Ablauf zur siebten Aktion der PERFORM-VARYING-Anweisung weiter. Ist Bedingung-3 nicht erfüllt, dann werden die spezifizierten Prozeduren ausgeführt.

Die fünfte Aktion, die durchgeführt wird, wenn Bedingung-3 nicht erfüllt ist, besteht in der Ausführung der Prozeduren Prozedur-Name-1 THRU Prozedur-Name-2. Nach Ausführung dieser Prozeduren wird der Inhalt von Index-Name-5 (oder Identifier-7) um den Wert von Literal-6 (oder Identifier-9) erhöht. Erscheint Index-Name-5 in der PERFORM-VARYING-Anweisung, wird dieser Index nach den Regeln der SET-Anweisung

erhöht. Nach Erhöhung von Index-Name-5 (oder Identifier-7) erfolgt, um herauszufinden, ob Bedingung-3 erfüllt ist, ein Rücksprung zum vierten Schritt.

Ist Bedingung-3 erfüllt, geht der logische Ablauf weiter bei der siebten Aktion, wo der Wert von Identifier-8 (oder Index-Name-6 oder Literal-5) in Index-Name-5 (oder Identifier-7) eingesetzt wird. Somit wird wegen des Zugriffs auf den ersten Posten der nächsten Tabellendimension der Index oder das Datenfeld Subskript auf der niedrigsten Stufe erneut initialisiert.

Die achte Aktion besteht darin, Index-Name-3 (oder Identifier-4) um den Wert von Literal-4 (oder Identifier-6) zu erhöhen. Erscheint Index-Name-3 in der PERFORM-VARYING-Anweisung, dann wird dieser Index nach den Regeln der SET-Anweisung erhöht. Somit erfolgt die Erhöhung des Indexes oder des Subskript-Datenfeldes auf der mittleren Stufe. Nach der Erhöhung von Index-Name-3 oder Identifier-4 erfolgt, um herauszufinden, ob Bedingung-2 erfüllt ist, ein Rücksprung zum vierten Schritt.

Ist Bedingung-2 erfüllt, geht der logische Ablauf weiter zur neunten Aktion, wo der Inhalt von Identifier-5 (oder Index-Name-4 oder Literal-3) in Index-Name-3 (oder Identifier-4) eingesetzt wird. Somit wird wegen des Zugriffs auf den ersten Posten der nächsten Tabellendimension der Index oder das Subskript-Datenfeld auf der höchsten Stufe erneut initialisiert.

Die zehnte Aktion besteht darin, Index-Name-1 (oder Identifier-1) um den Wert von Literal-2 (oder Identifier-3) zu erhöhen. Erscheint Index-Name-1 in der PERFORM-VARYING-Anweisung, dann wird dieser Index nach den Regeln der SET-Anweisung erhöht. Somit erfolgt die Erhöhung des Indexes oder des Subskript-Datenfeldes auf der höchsten Stufe. Nach der Erhöhung von Index-Name-1 oder Identifier-1 erfolgt ein Rücksprung zum zweiten Schritt, um herauszufinden, ob Bedingung-1 erfüllt ist.

Der logische Ablauf der PERFORM-VARYING-Anweisung mit drei Indizes oder drei Identifiern geht so vor sich, daß der Index oder das Subskriptfeld auf der niedrigsten Stufe variiert wird, bis Bedingung-3 erfüllt ist. Ist Bedingung-3 erfüllt, dann wird der Index oder das Subskriptfeld auf der mittleren Stufe so lange variiert, bis Bedingung-2 erfüllt ist. Ist Bedingung-2 erfüllt, dann wird der Index oder das Subskriptfeld auf der höchsten Stufe so lange variiert, bis Bedingung-1 erfüllt ist.

Während der Ausführung der Prozeduren wird jede Änderung der VARYING-Variablen (Identifier-1 oder Index-Name-1), der AFTER-Variablen (Identifier-4, Index-Name-3, Identifier-7 oder Index-Name-5), der FROM-Variablen (Identifier-2, Index-Name-2, Identifier-5, Index-Name-4, Identifier-8 oder Index-Name-6) oder der BY-Variablen (Identifier-3, Identifier-6 oder Identifier-9) berücksichtigt und hat somit ihre Auswirkung auf den internen Ablauf der PERFORM-VARYING-Anweisung.

Die PERFORM-VARYING-Anweisung mit drei Indizes oder drei Subskriptfeldern kann zum sequentiellen Zugriff auf die Datenfelder in einer 3-dimensionalen Tabelle verwendet werden. Das folgende Beispiel 1 zeigt eine PERFORM-VARYING-Anweisung mit drei Indizes, durch die Zugriff auf eine 3-dimensionale Tabelle erfolgt. Beispiel 2 zeigt eine PERFORM-VARYING-Anweisung mit drei Datenfeldern, durch die Zugriff auf dieselbe 3-dimensionale Tabelle erfolgt.

Beispiele folgen auf der nächsten Seite:

Beispiel 1:

```
FILE SECTION.
FD  DATEI3  BLOCK CONTAINS 1 RECORDS
        RECORD CONTAINS 284 CHARACTERS
        LABEL RECORD IS STANDARD
        VALUE OF FILE-ID IS "DATEI3".
01  SATZ3.
    02  ARTIKEL-NR  PIC X(6).
    02  FIRMA OCCURS 2 TIMES INDEXED BY FIRMA-INDEX.
        03  FIRMA-NR  PIC XXX.
        03  ABTLG  OCCURS 4 TIMES
                   INDEXED BY ABT-INDEX.
            04  ABTLG-NR    PIC XXXX.
            04  TAGES-UMSATZ PIC 9999V99 OCCURS 5 TIMES
                   INDEXED BY TAG-INDEX.

PROCEDURE DIVISION.
    PERFORM KUMULATION VARYING FIRMA-INDEX FROM 1 BY 1
            UNTIL FIRMA-INDEX IS GREATER THAN 2
         AFTER ABT-INDEX FROM 1 BY 1
            UNTIL ABT-INDEX IS GREATER THAN 4
         AFTER TAG-INDEX FROM 1 BY 1
            UNTIL TAG-INDEX IS GREATER THAN 5.

KUMULATION.
    ADD TAGES-UMSATZ (FIRMA-INDEX, ABT-INDEX, TAG-INDEX) TO
            GESAMT-UMSATZ.
```

In diesem Beispiel ist FIRMA-INDEX der Index auf der höchsten Stufe, der zum Zugriff auf den mit FIRMA bezeichneten Tabellenposten der ersten Dimension benutzt wird. ABT-INDEX ist der Index auf der mittleren Stufe, der zum Zugriff auf den mit ABTLG bezeichneten Tabellenposten der zweiten Dimension benutzt wird. TAG-INDEX ist der Index auf der niedrigsten Stufe, der zum Zugriff auf den mit TAGESUMSATZ bezeichneten Tabellenposten der dritten Dimension benutzt wird. Die PERFORM-VARYING-Anweisung bewirkt, daß die Tages-Umsätze der einzelnen Abteilungen der beiden Geschäfte in das Datenfeld GESAMT-UMSATZ addiert werden. Die folgende Übersicht enthält die vierzig ADD-Anweisungen, die die PERFORM-VARYING-Anweisung intern durchführt.

ADD-Anweisung	Verarbeiteter Tabellenposten		
	Geschäft	Abteilung	Tag
ADD TAGES-UMSATZ (1, 1, 1) TO GESAMT-UMSATZ	1	1	1
ADD TAGES-UMSATZ (1, 1, 2) TO GESAMT-UMSATZ	1	1	2
ADD TAGES-UMSATZ (1, 1, 3) TO GESAMT-UMSATZ	1	1	3
ADD TAGES-UMSATZ (1, 1, 4) TO GESAMT-UMSATZ	1	1	4
ADD TAGES-UMSATZ (1, 1, 5) TO GESAMT-UMSATZ	1	1	5
ADD TAGES-UMSATZ (1, 2, 1) TO GESAMT-UMSATZ	1	2	1
ADD TAGES-UMSATZ (1, 2, 2) TO GESAMT-UMSATZ	1	2	2
ADD TAGES-UMSATZ (1, 2, 3) TO GESAMT-UMSATZ	1	2	3
ADD TAGES-UMSATZ (1, 2, 4) TO GESAMT-UMSATZ	1	2	4
ADD TAGES-UMSATZ (1, 2, 5) TO GESAMT-UMSATZ	1	2	5
ADD TAGES-UMSATZ (1, 3, 1) TO GESAMT-UMSATZ	1	3	1
ADD TAGES-UMSATZ (1, 3, 2) TO GESAMT-UMSATZ	1	3	2
ADD TAGES-UMSATZ (1, 3, 3) TO GESAMT-UMSATZ	1	3	3
ADD TAGES-UMSATZ (1, 3, 4) TO GESAMT-UMSATZ	1	3	4
ADD TAGES-UMSATZ (1, 3, 5) TO GESAMT-UMSATZ	1	3	5
ADD TAGES-UMSATZ (1, 4, 1) TO GESAMT-UMSATZ	1	4	1
ADD TAGES-UMSATZ (1, 4, 2) TO GESAMT-UMSATZ	1	4	2
ADD TAGES-UMSATZ (1, 4, 3) TO GESAMT-UMSATZ	1	4	3
ADD TAGES-UMSATZ (1, 4, 4) TO GESAMT-UMSATZ	1	4	4
ADD TAGES-UMSATZ (1, 4, 5) TO GESAMT-UMSATZ	1	4	5
ADD TAGES-UMSATZ (2, 1, 1) TO GESAMT-UMSATZ	2	1	1
ADD TAGES-UMSATZ (2, 1, 2) TO GESAMT-UMSATZ	2	1	2
ADD TAGES-UMSATZ (2, 1, 3) TO GESAMT-UMSATZ	2	1	3
ADD TAGES-UMSATZ (2, 1, 4) TO GESAMT-UMSATZ	2	1	4
ADD TAGES-UMSATZ (2, 1, 5) TO GESAMT-UMSATZ	2	1	5
ADD TAGES-UMSATZ (2, 2, 1) TO GESAMT-UMSATZ	2	2	1
ADD TAGES-UMSATZ (2, 2, 2) TO GESAMT-UMSATZ	2	2	2
ADD TAGES-UMSATZ (2, 2, 3) TO GESAMT-UMSATZ	2	2	3
ADD TAGES-UMSATZ (2, 2, 4) TO GESAMT-UMSATZ	2	2	4
ADD TAGES-UMSATZ (2, 2, 5) TO GESAMT-UMSATZ	2	2	5
ADD TAGES-UMSATZ (2, 3, 1) TO GESAMT-UMSATZ	2	3	1
ADD TAGES-UMSATZ (2, 3, 2) TO GESAMT-UMSATZ	2	3	2
ADD TAGES-UMSATZ (2, 3, 3) TO GESAMT-UMSATZ	2	3	3
ADD TAGES-UMSATZ (2, 3, 4) TO GESAMT-UMSATZ	2	3	4
ADD TAGES-UMSATZ (2, 3, 5) TO GESAMT-UMSATZ	2	3	5
ADD TAGES-UMSATZ (2, 4, 1) TO GESAMT-UMSATZ	2	4	1
ADD TAGES-UMSATZ (2, 4, 2) TO GESAMT-UMSATZ	2	4	2
ADD TAGES-UMSATZ (2, 4, 3) TO GESAMT-UMSATZ	2	4	3
ADD TAGES-UMSATZ (2, 4, 4) TO GESAMT-UMSATZ	2	4	4
ADD TAGES-UMSATZ (2, 4, 5) TO GESAMT-UMSATZ	2	4	5

Beispiel 2:

```
FILE SECTION.
FD  DATEI3  BLOCK CONTAINS 1 RECORDS
    RECORD CONTAINS 284 CHARACTERS
        LABEL RECORD IS STANDARD
        VALUE OF FILE-ID IS "DATEI3".
01  SATZ3.
    02  ARTIKEL-NR  PIC X(6).
    02  FIRMA OCCURS 2 TIMES.
        03  FIRMA-NR  PIC XXX.
        03  ABTLG  OCCURS 4 TIMES.
            04  ABTLG-NR  PIC XXXX.
            04  TAGES-UMSATZ PIC 9999V99 OCCURS 5 TIMES.

WORKING-STORAGE SECTION.
77  TAG-SUB   PIC 9.
77  ABT-SUB   PIC 9.
77  FIRMA-SUB PIC 9.

PROCEDURE DIVISION.
    PERFORM KUMULATION VARYING FIRMA-SUB FROM 1 BY 1
            UNTIL FIRMA-SUB IS GREATER THAN 2
         AFTER ABT-SUB FROM 1 BY 1
            UNTIL ABT-SUB IS GREATER THAN 4
         AFTER TAG-SUB FROM 1 BY 1
            UNTIL TAG-SUB IS GREATER THAN 5.

KUMULATION.
    ADD TAGES-UMSATZ (FIRMA-SUB, ABT-SUB, TAG-SUB) TO
            GESAMT-UMSATZ.
```

In diesem Beispiel ist FIRMA-SUB das Subskript auf der höchsten Stufe, das zum Zugriff auf den mit FIRMA bezeichneten Tabellenposten der ersten Dimension wird. ABT-SUB ist das Subskript auf der mittleren Stufe, das zum Zugriff auf den mit ABTLG bezeichneten Tabellenposten der zweiten Dimension benutzt wird. TAG-SUB ist das Subskript auf der niedrigsten Stufe, das zum Zugriff auf den mit TAGES-UMSATZ bezeichneten Tabellenposten der dritten Dimension benutzt wird. Die PERFORM-VARYING-Anweisung bewirkt, daß die TAGES-UMSÄTZE der einzelnen Abteilungen der beiden Firmen in das Datenfeld GESAMT-UMSATZ addiert werden.

Die PURGE-Anweisung

Diese Anweisung dient der Rückgängigmachung bzw. Herausnahme einer unkompletten Message aus dem Message Control System (MCS), die von einer oder mehreren SEND-Anweisungen dorthin gesandt wurde.

Format:

PURGE CD-Name-1

CD-Name-1 muß eine Output-Communication-Description-Eintragung benennen (CD).

MCS eliminiert jede unkomplette Message, die im Wartezustand zur Übertragung an denjenigen Empfänger steht, der in der Communication Description unter CD-Name-1 angegeben ist.

Dabei werden von der PURGE-Anweisung solche Messages nicht berührt, die an ihrem Ende ein EMI (End of message indicator) oder ein EGI (End of group indicator) aufweisen.

Die Inhalte des STATUS-KEY-Feldes und des ERROR-KEY-Feldes des mittels CD-Name-1 adressierten Bereiches werden von MCS aktualisiert.

DIE READ-ANWEISUNG (Sequentielle Datei)

Die READ-Anweisung macht den nächsten Satz einer sequentiellen Datei verfügbar.

Format:

```
READ  Datei-Name RECORD [WITH NO LOCK] [INTO Identifier]
      [AT END  unbedingte Anweisung 1]
      [NOT AT END unbedingte Anweisung 2]
      [END-READ]
```

Der in der READ-Anweisung gegebene Datei-Name muß einem in einer Dateibeschreibung auftretenden Datei-Namen und einem in einem FILE-CONTROL-Eintrag auftretenden Datei-Namen entsprechen. Der FILE-CONTROL-Eintrag für den Datei-Namen muß indirekt oder ausdrücklich die Klausel ORGANIZATION IS SEQUENTIAL und die Klausel ACCESS MODE IS SEQUENTIAL aufweisen.

Eine sequentielle Datei kann Sätze fester oder variabler Satzlänge aufweisen; eine Ausnahme besteht, wenn die INTO-Angabe vorliegt. Die INTO-Angabe nämlich beschränkt die sequentielle Datei auf Sätze fester Länge. Besteht eine sequentielle Datei aus Sätzen variabler Länge, dann dürfen die RECORD CONTAINS-Klausel und die Satzbeschreibungen nicht die von dem Variable-Längen-Indikator (VLI) eingenommenen Bytes umfassen.

Vor Ausführung der READ-Anweisung muß die sequentielle Datei durch eine OPEN INPUT- oder OPEN I-O-Anweisung eröffnet worden sein.

Die erste Ausführung der READ-Anweisung betrifft den ersten Satz der sequentiellen Datei. Nachfolgende Ausführungen von READ-Anweisungen für dieselbe Datei haben Zugriff zum sequentiell nächsten Satz der Datei. Nach Ausführung der READ-Anweisung befindet sich der Satz in dem der Datei zugeordneten Satz-Bereich und ist zur Verarbeitung verfügbar.

Wenn die Sätze einer Datei mit mehr als einer Satzdefinition definiert sind, nehmen diese denselben Speicherbereich ein. Jede weitere Satzbeschreibung stellt somit eine indirekte Redefinition des Satzbereiches dar. Die Inhalte aller Datenfelder, die über den Bereich des laufenden Datensatzes, auf den durch die READ-Anweisung Zugriff erfolgt, hinausgehen, sind bei Beendigung der Ausführung der READ-Anweisung unbestimmt.

Wird während der Ausführung der READ-Anweisung das Ende einer Magnetbandspule festgestellt, dann veranlaßt das Betriebssystem automatisch den Spulen-Wechsel. Sobald die neue Spule betriebsbereit ist, wird sie eröffnet und der dortige erste Datensatz gelesen.

WITH NO LOCK-Angabe

Diese Angabe gestattet parallele Programmzugriffe auf den zu lesenden Block. Fehlt diese Angabe in der READ-Anweisung, wird der Block für gleichzeitige andere Operationen gesperrt.

INTO-Angabe

Ist INTO angegeben, wird der gelesene Datensatz aus dem Satzbereich in den mittels Identifier definierten Bereich kopiert. Diese Übertragung wird nach den für die MOVE-Anweisung festgelegten Regeln erreicht. Bei unkorrekter Ausführung der READ-Anweisung erfolgt die Übertragung jedoch nicht. Jedes dem Identifier zugeordnete Subskript oder jeder Index kommt zur Wirkung, nachdem der Satz gelesen wurde und wenn er in den Datenbereich übertragen wird.

Der Satz, der gelesen wird, ist anschließend sowohl im Satzbereich als auch in dem dem Identifier zugeordneten Datenbereich verfügbar. Der dem Identifier zugeordnete Datenbereich und der dem Datei-Namen zugeordnete Satzbereich dürfen nicht denselben Speicherbereich bilden.

Die INTO-Angabe darf nicht verwendet werden, wenn die sequentielle Datei Sätze variabler Länge enthält.

AT END-Angabe

AT END muß für eine READ-Anweisung angegeben sein, die auf eine sequentielle Datei Bezug nimmt, für die in den DECLARATIVES keine USE-Anweisung enthalten ist.

Wenn zur Zeit der Ausführung einer READ-Anweisung kein weiterer Datensatz mehr in der Datei ist, wird die AT END-Bedingung wirksam und die Ausführung der READ-Anweisung gilt als erfolglos. Der Inhalt des zugeordneten Satz-Bereiches ist unbestimmt.

Wenn die AT END-Bedingung auftritt, werden die folgenden Aktionen in der angegebenen Reihenfolge durchgeführt:

1. Ist die FILE STATUS-Klausel im FILE-CONTROL-Eintrag für diese sequentielle Datei spezifiziert, wird der Wert 10 in das FILE STATUS-Feld übertragen, um die AT END-Bedingung anzuzeigen.

2. Ist die AT END-Angabe in der READ-Anweisung enthalten, wird die Steuerung auf die unbedingte Anweisung übertragen. Keine der etwaigen für diese Datei spezifizierten DECLARATIVE-Prozeduren wird ausgeführt.

3. Ist die AT END-Angabe nicht in der READ-Anweisung enthalten, werden für diese Datei DECLARATIVE-Prozeduren ausgeführt.

Wurde die AT END-Bedingung festgestellt, darf eine erneute READ-Anweisung für diese Datei nicht erfolgen, ohne daß ihr eine korrekte CLOSE-Anweisung mit anschließender korrekter OPEN-Anweisung vorangeht.

NOT AT END-Angabe

Ihre Verwendung ist fakultativ. Ist diese Angabe in der READ-Anweisung nicht vorhanden, gelangt der nächste COBOL-Satz zur Ausführung.

Beispiel 1:

```
FILE-CONTROL.
    SELECT DATEI-F ASSIGN TO MAGNETIC-TAPE
        ORGANIZATION IS SEQUENTIAL
        ACCESS MODE IS SEQUENTIAL.

DATA DIVISION.
FILE SECTION.
FD  DATEI-F BLOCK CONTAINS 70 RECORDS
            RECORD CONTAINS 10 CHARACTERS
            LABEL RECORD IS STANDARD
            VALUE OF FILE-ID IS "DATEI-F".
01  SATZ-F  PIC X(10).

PROCEDURE DIVISION.
ANFANG.
    OPEN INPUT DATEI-F.
R.  READ DATEI-F AT END GO TO ENDE.
        ------------------------
    GO TO R.
ENDE.
    CLOSE DATEI-F WITH LOCK.
```

Im Beispiel 1 geht es um eine Magnetband-Datei. Sie hat den Namen DATEI-F. Damit Daten aus DATEI-F gelesen werden können, muß sie zuerst im INPUT-Modus eröffnet werden (OPEN INPUT-Anweisung). Die READ-Anweisung greift sequentiell auf DATEI-F zu, um jeweils den nächsten Datensatz im Bereich SATZ-F zur Verfügung zu stellen.

Wird das Dateiende erreicht, führt die READ-Anweisung AT END GO TO ENDE durch. Die CLOSE-Anweisung beendet die Bearbeitung der DATEI-F mit Rückspulen des Magnetbandes und bringt dann die Datei in einen gesperrten Zustand, um dadurch eine weitere Verarbeitung der DATEI-F während dieses Programmes auszuschließen.

Beispiel 2:

```
FILE-CONTROL.
    SELECT DATEI-G  ASSIGN TO DISC.
DATA DIVISION.
FILE SECTION.
FD  DATEI-G BLOCK CONTAINS 51 RECORDS
            RECORD CONTAINS 10 CHARACTERS
            LABEL RECORD IS STANDARD
            VALUE OF FILE-ID IS "DATEI-G".
01  SATZ-G  PIC X(10).
WORKING-STORAGE SECTION.
01  ARB-SATZ.
    02  ARB-FELD1   PIC 9(5).
    02  ARB-FELD2   PIC X(5).

PROCEDURE DIVISION.
ANFANG.
    OPEN INPUT DATEI-G.
R.  READ DATEI-G INTO ARB-SATZ AT END GO TO ENDE.
       -------------------------
    GO TO R.
ENDE.
    CLOSE DATEI-G.
```

In diesem Beispiel geht es um eine sequentielle Magnetplatten-Datei mit sequentiellem Zugriff. Die Datei hat den Namen DATEI-G. Enthält der FILE-CONTROL-Eintrag keine ORGANIZATION-Klausel, wird vom Compiler die sequentielle Datei-Struktur angenommen. Enthält der FILE-CONTROL-Eintrag keine ACCESS MODE-Klausel, wird vom Compiler die sequentielle Zugriffsart angenommen.

Damit Daten aus DATEI-G gelesen werden können, muß die Datei zuerst im INPUT-Modus eröffnet werden (OPEN INPUT-Anweisung). Die READ-Anweisung greift sequentiell auf DATEI-G zu, um dadurch den nächsten Datensatz zur Verarbeitung verfügbar zu machen. Nach Ausführung der READ-Anweisung ist der Datensatz aus DATEI-G sowohl in SATZ-G als auch in ARB-SATZ vorhanden, weil die INTO-Angabe in der READ-Anweisung enthalten ist.

Sobald die READ-Anweisung feststellt, daß in DATEI-G keine weiteren Sätze zur Verfügung stehen, wird AT END GO TO ENDE ausgeführt. Die CLOSE-Anweisung beendet die Bearbeitung der DATEI-G.

Beispiel 3:

```
FILE-CONTROL.
    SELECT DATEI-H ASSIGN TO DISC
        ORGANIZATION IS SEQUENTIAL
        ACCESS MODE IS SEQUENTIAL.

DATA DIVISION.
FILE SECTION.
FD  DATEI-H BLOCK CONTAINS 20 RECORDS
            RECORD CONTAINS 25 CHARACTERS
            LABEL RECORD IS STANDARD
            VALUE OF FILE-ID IS "DATEI128AA".
01  SATZ-H  PIC X(25).

PROCEDURE DIVISION.
ANFANG.
    OPEN I-O DATEI-H.
R.  READ DATEI-H AT END GO TO ENDE.
        ------------------------
    REWRITE SATZ-H.
        ------------------------
    GO TO R.
ENDE.
    CLOSE DATEI-H.
```

Im obigen Beispiel geht es um eine sequentielle Magnetplatten-Datei mit sequentiellem Zugriff. Die Datei hat den Namen DATEI-H. Da Datensätze sowohl aus der DATEI-H zu lesen als auch in diese zurückzuschreiben sind, muß die Datei zuerst im I-O-Modus eröffnet werden (OPEN I-O-Anweisung). Die READ-Anweisung greift sequentiell auf DATEI-H zu, um dadurch in SATZ-H den nächsten Datensatz zur Verarbeitung verfügbar zu machen.

Wird der Inhalt des zuletzt gelesenen Datensatzes während der Verarbeitung verändert, schreibt die REWRITE-Anweisung den Inhalt aus SATZ-H auf DATEI-H in die Satz-Position, auf die durch die letzte READ-Operation zugegriffen wurde.

Wenn die READ-Anweisung feststellt, daß in DATEI-H keine weiteren Sätze zur Verfügung stehen, dann wird AT END GO TO ENDE ausgeführt. Die CLOSE-Anweisung beendet die Bearbeitung der DATEI-H.

DIE READ-ANWEISUNG (Relative Datei - Sequentieller Zugriff)

Bei sequentiellem Zugriff macht die READ-Anweisung den nächsten Satz einer relativen Datei verfügbar.

Format:

READ Datei-Name RECORD [WITH NO LOCK] [INTO Identifier]
[AT END unbedingte Anweisung 2]
[END-READ]

Der in der READ-Anweisung gegebene Datei-Name muß einem in einer Dateibeschreibung auftretenden Datei-Namen und einem in einem FILE-CONTROL-Eintrag auftretenden Datei-Namen entsprechen. Der FILE-CONTROL-Eintrag muß die Klausel ORGANIZATION IS RELATIVE aufweisen. Der FILE-CONTROL-Eintrag muß zusätzlich indirekt oder ausdrücklich die Klausel ACCESS MODE IS SEQUENTIAL aufweisen.

Vor Ausführung der READ-Anweisung muß die relative Datei entweder durch eine OPEN INPUT- oder durch eine OPEN I-O-Anweisung eröffnet worden sein.

Die erste Ausführung der READ-Anweisung trifft auf den ersten aktiven Datensatz. Wurden keine Daten in eine Satz-Position geschrieben oder wurden Daten daraus gelöscht, so wird diese übergangen. Durch nachfolgende Ausführungen der READ-Anweisung für dieselbe relative Datei wird auf den nächsten Daten enthaltenden Satz der Datei zugegriffen. Nach Ausführung der READ-Anweisung befindet sich der zur Verarbeitung verfügbare Satz in dem dem Datei-Namen zugeordneten Satz-Bereich.

Der FILE-CONTROL-Eintrag kann in der Klausel ACCESS MODE IS SEQUENTIAL wahlweise die Angabe RELATIVE KEY enthalten. Ist dies der Fall, wird die relative Nummer des Datensatzes, auf den durch die READ-Anweisung sequentiell Zugriff genommen wird, bei erfolgreicher Beendigung der READ-Anweisung im relativen Satzschlüsselfeld zur Verfügung gestellt.

WITH NO LOCK-Angabe

Diese Angabe gestattet parallele Programmzugriffe auf den zu lesenden Block. Fehlt diese Angabe in der READ-Anweisung, wird der Block für gleichzeitige andere Operationen gesperrt.

INTO-Angabe

Ist INTO angegeben, wird der gelesene Datensatz aus dem Satz-Bereich zusätzlich in den mittels Identifier definierten Bereich kopiert. Diese Übertragung wird nach den für die MOVE-Anweisung festgelegten Regeln erreicht. Bei unkorrekter Ausführung der READ-Anweisung erfolgt die Übertragung jedoch nicht. Jedes dem Identifier zugeordnete Subskript oder jeder Index kommt zur Wirkung, nachdem der Satz gelesen wurde und wenn er in den Datenbereich übertragen wird.

Der Satz, der gelesen wird, ist anschließend im Satz-Bereich als auch in dem dem Identifier zugeordneten Datenbereich verfügbar. Der dem Identifier zugeordnete Datenbereich und der dem Datei-Namen zugeordnete Satz-Bereich dürfen nicht denselben Speicherbereich bilden.

Die INTO-Angabe darf nicht verwendet werden, wenn die relative Datei Sätze variabler Länge enthält.

AT END-Angabe

AT END muß für eine READ-Anweisung angegeben sein, die auf eine relative Datei Bezug nimmt, für die in den DECLARATIVES keine USE-Anweisung enthalten ist.

Wenn zur Zeit der Ausführung einer READ-Anweisung kein weiterer Datensatz mehr in der Datei ist, ist die AT END-Bedingung erfüllt und die Ausführung der READ-Anweisung gilt als erfolglos. Nach erfolgloser Ausführung einer READ-Anweisung ist der Inhalt des zugeordneten Satz-Bereiches unbestimmt.

Wenn die AT END-Bedingung auftritt, werden folgende Aktionen in der angegebenen Reihenfolge durchgeführt:

1. Ist die FILE STATUS-Klausel im FILE-CONTROL-Eintrag für diese relative Datei spezifiziert, wird der Wert 10 in das FILE-STATUS-Feld übertragen, um die AT END-Bedingung anzuzeigen.

2. Ist die AT END-Angabe in der READ-Anweisung enthalten, wird die Steuerung auf die unbedingte Anweisung übertragen. Keine der etwaigen für diese Datei spezifizierten DECLARATIVE-Prozeduren wird ausgeführt.

3. Ist die AT END-Angabe nicht in der READ-Anweisung enthalten, werden für diese Datei DECLARATIVE-Prozeduren ausgeführt.

Wurde die AT-END-Bedingung festgestellt, darf eine erneute READ-Anweisung für diese Datei nicht erfolgen, ohne daß ihr eine korrekte CLOSE-Anweisung mit anschließender korrekter OPEN-Anweisung vorangeht.

NOT AT END-Angabe

Ihre Verwendung ist fakultativ. Ist diese Angabe in der READ-Anweisung nicht vorhanden, gelangt der nächste COBOL-Satz zur Ausführung.

FILE STATUS-Datenfeld

Ist die FILE STATUS-Klausel im FILE-CONTROL-Eintrag für die relative Datei spezifiziert, wird dieser Datei ein FILE STATUS-Feld zugeordnet. Nach Ausführung der READ-Anweisung wird ein Wert in das FILE STATUS-Feld übertragen. Alle auf die relative Datei anwendbaren DECLARATIVE-Prozeduren werden ausgeführt, nachdem das FILE STATUS-Feld seinen Status empfangen hat. Die möglichen Werte für das FILE STATUS-Feld stehen im Anhang.

Beispiele folgen auf den nächsten Seiten:

Beispiel 1:

```
FILE-CONTROL.
    SELECT DATEI-I ASSIGN TO DISC
        ORGANIZATION IS RELATIVE
        ACCESS MODE IS SEQUENTIAL.

DATA DIVISION.
FILE SECTION.
FD  DATEI-I BLOCK CONTAINS 51 RECORDS
            RECORD CONTAINS 10 CHARACTERS
            LABEL RECORD IS STANDARD
            VALUE OF FILE-ID IS "DATEI-I".
01  SATZ-I  PIC X(10).

PROCEDURE DIVISION.
ANFANG.
    OPEN INPUT DATEI-I.

R1. READ DATEI-I AT END GOT TO ENDE.
     ---------------------------
    GO TO R1.

ENDE.  CLOSE DATEI-I.
```

Im obigen Beispiel geht es um eine relative Magnetplatten-Datei mit sequentiellem Zugriff. Die Datei hat den Namen DATEI-I. Damit Daten aus DATEI-I gelesen werden können, muß sie zuerst mit der OPEN INPUT-Anweisung eröffnet werden.

Die READ-Anweisung greift sequentiell auf DATEI-I zu, um in SATZ-I den nächsten gültigen Datensatz der DATEI-I verfügbar zu machen. Jede nicht belegte Satzposition der relativen Datei wird dabei von der READ-Anweisung ignoriert.

Wenn die READ-Anweisung feststellt, daß in DATEI-I keine weiteren Sätze zur Verfügung stehen, dann wird AT END GO TO ENDE ausgeführt. Die CLOSE-Anweisung beendet die Bearbeitung der DATEI-I.

Beispiel 2:

```
FILE-CONTROL.
     SELECT DATEI-J ASSIGN TO DISC
       ORGANIZATION IS RELATIVE
       ACCESS MODE IS SEQUENTIAL RELATIVE KEY IS SCHL-J.
DATA DIVISION.
FILE SECTION.
FD   DATEI-J   BLOCK CONTAINS 17 RECORDS
               RECORD CONTAINS 30 CHARACTERS
               LABEL RECORD IS STANDARD.
01   SATZ-J    PIC X(30).
WORKING-STORAGE SECTION.
01   SCHL-J  PIC 9999.
PROCEDURE DIVISION.
A.
     OPEN I-O DATEI-J.
B.   READ DATEI-J AT END GO TO ENDE.
       -------------------------
     REWRITE SATZ-J.
       -------------------------
     GO TO B.
ENDE.
     CLOSE DATEI-J.
```

Im obigen Beispiel geht es um eine relative Magnetplatten-Datei mit sequentiellem Zugriff. Sie hat den Namen DATEI-J. Da Datensätze sowohl aus der DATEI-J zu lesen als auch in diese zurückzuschreiben sind, muß sie zuerst im I-O-Modus eröffnet werden (OPEN I-O-Anweisung).

Die READ-Anweisung greift sequentiell auf DATEI-J zu, um in SATZ-J den nächsten gültigen Datensatz von DATEI-J verfügbar zu machen. Jede nicht belegte Satzposition der relativen Datei wird dabei von der READ-Anweisung ignoriert. Nach der korrekten Ausführung der READ-Anweisung enthält das Satzschlüsselfeld SCHL-J die relative Nummer des Datensatzes, auf den durch die READ-Anweisung zugegriffen wurde.

Wurde der Inhalt des laufenden Datensatzes verändert, schreibt die REWRITE-Anweisung den Inhalt von SATZ-J in die Satz-Position zurück, auf die durch die letzte READ-Anweisung zugegriffen wurde.

Wenn die READ-Anweisung feststellt, daß in DATEI-J keine weiteren Sätze zur Verfügung stehen, wird AT END GO TO ENDE ausgeführt. Die CLOSE-Anweisung beendet die Bearbeitung der DATEI-J.

DIE READ-ANWEISUNG (Relative Datei - Direktzugriff/ Dynamischer Zugriff)

Bei direktem Zugriff macht die READ-Anweisung einen bestimmten Satz einer relativen Datei verfügbar.

Format 1 (Direktzugriff):

```
READ  Datei-Name RECORD [WITH NO LOCK] [INTO Identifier]
      [INVALID KEY unbedingte Anweisung 1]
      [NOT INVALID KEY unbedingte Anweisung 2]
      [END-READ]
```

Beim dynamischen Zugriff werden ab einem bestimmten Satz in der Datei weitere Sätze sequentiell gelesen:

Format 2 (Dynamischer Zugriff):

```
READ  Datei-Name NEXT RECORD
      [WITH NO LOCK] [INTO Identifier]
      [AT END unbedingte Anweisung 1]
      [NOT AT END unbedingte Anweisung 2]
      [END-READ]
```

Der in der READ-Anweisung angegebene Datei-Name muß einem in einer Dateibeschreibung auftretenden Datei-Namen und einem in einem FILE-CONTROL-Eintrag auftretenden Datei-Namen entsprechen. Der FILE-CONTROL-Eintrag muß die Klausel ORGANIZATION IS RELATIVE und die Klausel ACCESS MODE IS RANDOM oder DYNAMIC aufweisen.

Vor Ausführung der READ-Anweisung muß die relative Datei entweder im INPUT- oder I-O-Modus eröffnet worden sein.

Beim Zugriff auf die Datei macht die READ-Anweisung den durch das Satzschlüsselfeld bezeichneten Satz im Satz-Bereich verfügbar. Das Satzschlüsselfeld muß in der RELATIVE KEY-Klausel angegeben sein. Der Programmierer muß die relative Nummer des einzulesenden Satzes vor Ausführung der READ-Anweisung in das Satzschlüsselfeld bringen. Nach Ausführung der READ-Anweisung ist der gelesene Satz im Satz-Bereich verfügbar.

WITH NO LOCK-Angabe

Diese Angabe gestattet parallele Programmzugriiffe auf den zu lesenden Block. Fehlt diese Angabe in der READ-Anweisung, wird der Block für gleichzeitige andere Operationen gesperrt.

INTO-Angabe

Ist INTO angegeben, wird der gelesene Datensatz aus dem Satz-Bereich zusätzlich in den mittels Identifier definierten Bereich kopiert. Diese Übertragung wird nach den für die MOVE-Anweisung festgelegten Regeln erreicht. Bei unkorrekter Ausführung der READ-Anweisung erfolgt die Übertragung jedoch nicht. Jedes dem Identifier zugeordnete Subskript oder jeder Index kommt zur Wirkung, nachdem der Satz gelesen wurde und wenn er in den Datenbereich übertragen wird.

Der Satz, der gelesen wird, ist anschließend sowohl im Satz-Bereich als auch in dem dem Identifier zugeordneten Datenbereich verfügbar. Der dem Identifier zugeordnete Datenbereich und der dem Datei-Namen zugeordnete Satz-Bereich dürfen nicht denselben Speicherbereich bilden.

Die INTO-Angabe darf nicht verwendet werden, wenn die relative Datei Sätze variabler Länge enthält.

INVALID KEY-Angabe

Die INVALID KEY-Angabe muß für eine READ-Anweisung spezifiziert werden, die auf eine relative Datei mit direktem oder dynamischem Zugriff Bezug nimmt und für die keine USE-Anweisung in den DECLARATIVES existiert.

Weist die relative Datei, auf die direkt zugegriffen wird, keinen gültigen Datensatz in der durch das Satzschlüsselfeld spezifizierten Datensatzposition auf, liegt eine INVALID KEY-Bedingung vor und die Ausführung der READ-Anweisung gilt als erfolglos. Danach ist der Inhalt des Satz-Bereiches unbestimmt.

Liegt die INVALID KEY-Bedingung vor, werden folgende Aktionen in der angegebenen Reihenfolge durchgeführt:

1. Ist die FILE STATUS-Klausel im FILE-CONTROL-Eintrag für diese relative Datei spezifiziert, wird der Wert 23 in das FILE STATUS-Feld übertragen, um anzuzeigen, daß kein Datensatz in der durch das Satzschlüsselfeld spezifizierten Datensatzposition gefunden wurde.

2. Ist die INVALID KEY-Angabe in der READ-Anweisung vorhanden, wird die Steuerung auf die INVALID KEY-Anweisung übertragen. Keine der für diese Datei spezifizierten DECLARATIVE-Prozeduren wird ausgeführt.

3. Ist die INVALID KEY-Angabe in der READ-Anweisung nicht vorhanden, werden für diese Datei DECLARATIVE-Prozeduren ausgeführt.

Beispiele folgen auf den nächsten Seiten:

Beispiel 1:

```
FILE-CONTROL.
    SELECT DATEI-K  ASSIGN TO DISC
            ORGANIZATION IS RELATIVE
            ACCESS MODE IS RANDOM, RELATIVE KEY IS SCHL-K.
DATA DIVISION.
FILE SECTION.
FD  DATEI-K BLOCK CONTAINS 10 RECORDS
            RECORD CONTAINS 51 CHARACTERS
            LABEL RECORD IS STANDARD
            VALUE OF FILE-ID IS "DATEI-K".
01  SATZ-K  PIC X(5).
WORKING-STORAGE SECTION.
01  SCHL-K  PIC 99999.

PROCEDURE DIVISION.
ANFANG.
    OPEN INPUT DATEI-K.
        ------------------------
    MOVE 50 TO SCHL-K.
    READ DATEI-K  INVALID KEY GO TO KEIN-SATZ.
        ------------------------
    CLOSE DATEI-K.
```

Im obigen Beispiel handelt es sich um eine relative Magnetplatten-Datei mit Direktzugriff. Sie hat den Namen DATEI-K. Damit Daten aus DATEI-K gelesen werden können, muß die Datei zuerst im INPUT-Modus eröffnet werden (mit der OPEN INPUT-Anweisung).

Der Programmierer bringt die Ganzzahl 50 in das Satzschlüsselfeld SCHL-K, um die relative Nummer des Datensatzes, auf den Zugriff genommen werden soll, zu spezifizieren. Die anschließende READ-Anweisung macht den 50sten Datensatz aus DATEI-K zur Verarbeitung verfügbar. Nachdem die READ-Anweisung korrekt ausgeführt wurde, befindet sich der 50ste Datensatz aus DATEI-K in SATZ-K. Weist die 50ste Satzposition in DATEI-K keinen Datensatz auf, führt die Software GO TO KEIN-SATZ aus. Die CLOSE-Anweisung beendet die Bearbeitung der DATEI-K.

Beispiel 2:

```
FILE-CONTROL.
    SELECT DATEI-L ASSIGN TO DISC
        ORGANIZATION IS RELATIVE
        ACCESS MODE IS RANDOM, RELATIVE KEY IS SCHL-L.
DATA DIVISION.
FILE SECTION.
FD  DATEI-L BLOCK CONTAINS 25 RECORDS
            RECORD CONTAINS 20 CHARACTERS
            LABEL RECORD IS STANDARD.
01  SATZ-L  PIC X(20).
WORKING-STORAGE SECTION.
01  SCHL-L  PIC 9999.

PROCEDURE DIVISION.
ANFANG.
    OPEN I-O DATEI-L.
        -------------------------
    MOVE 20 TO SCHL-L.
    READ DATEI-L INVALID KEY GO TO FEHLER-1.
        -------------------------
    REWRITE SATZ-L INVALID KEY GO TO FEHLER-2.
        -------------------------
    CLOSE DATEI-L.
```

Im obigen Beispiel handelt es sich um eine relative Magnetplatten-Datei mit Direktzugriff. Sie hat den Namen DATEI-L. Da Datensätze sowohl zu lesen als auch zurückzuschreiben sind, muß die Datei im I-O-Modus eröffnet werden.

Der Programmierer bringt die Ganzzahl 20 in das Satzschlüsselfeld SCHL-L, um die relative Nummer des Datensatzes, auf den Zugriff genommen werden soll, zu spezifizieren. Die anschließende READ-Anweisung macht den 20sten Datensatz aus DATEI-L zur Verarbeitung verfügbar. Nachdem die READ-Anweisung korrekt ausgeführt wurde, befindet sich der 20ste Datensatz aus DATEI-L in SATZ-L. Weist die 20ste Satzposition in DATEI-L keinen Datensatz auf, führt die Software GO TO FEHLER-1 aus.

Die REWRITE-Anweisung schreibt den Inhalt von SATZ-L nach dessen Bearbeitung in die Satzposition 20 der Datei zurück. Dabei könnte beispielsweise dann "invalid key" auftreten, wenn zwischen READ und REWRITE der Inhalt des Satzschlüsselfeldes SCHL-L unzulässigerweise verändert wurde und somit ein fremder Satz in der Datei überschrieben würde.

Beispiel 3:

```
FILE-CONTROL.
    SELECT DATEI-M  ASSIGN TO DISC
        ORGANIZATION IS RELATIVE
        ACCESS MODE IS DYNAMIC
        RELATIVE KEY IS SCHL-M.
DATA DIVISION.
FILE SECTION.
FD  DATEI-M BLOCK CONTAINS 25 RECORDS
            RECORD CONTAINS 20 CHARACTERS
            LABEL RECORD IS STANDARD.
01  SATZ-M  PIC X(20).
WOKRING-STORAGE SECTION.
01  SCHL-M  PIC 9999.

PROCEDURE DIVISION.
ANFANG.
    OPEN INPUT  DATEI-M.
         ---------------------
    MOVE 3 TO SCHL-M.
    START DATEI-M INVALID KEY GO TO FEHLER-2.
R1. READ DATEI-M NEXT RECORD AT END GO TO ENDE.
         ---------------------
    GO TO R1.
    CLOSE DATEI-M.
```

Im obigen Beispiel handelt es sich um eine relative Magnetplatten-Datei mit dynamischem Zugriff.

Der Programmierer bringt die Ganzzahl 3 in das Satzschlüsselfeld SCHL-M, um die relative Nummer des ersten einer Gruppe von Datensätzen in DATEI-M zu spezifizieren. Die START-Anweisung veranlaßt, daß DATEI-M auf die dritte Datensatzposition vom Beginn der Datei an gesehen positioniert wird. Weist die dritte Datensatzposition in DATEI-M keinen Datensatz auf, wird GO TO FEHLER-2 ausgeführt.

Nach korrekter Ausführung der START-Anweisung macht die READ-NEXT-Anweisung den dritten Datensatz aus DATEI-M zur Verarbeitung verfügbar. Nachdem die READ NEXT-Anweisung korrekt ausgeführt wurde, befindet sich der dritte Datensatz aus DATEI-M in SATZ-M und kann verarbeitet werden.

Durch weitere Ausführungen der READ NEXT-Anweisung wird sequentiell auf die nächsten gültigen Datensätze in DATEI-M zugegriffen. Jede leere Satzposition der relativen Datei wird dabei von der READ NEXT-Anweisung übergangen. Somit verursacht die zweite Ausführung der READ NEXT-Anweisung den Zugriff auf den die relative Nummer 4 aufweisenden Datensatz. Die dritte Ausführung der READ NEXT-Anweisung greift auf den fünften Datensatz in der DATEI-M zu, usw.

Wenn die READ NEXT-Anweisung feststellt, daß in DATEI-M keine weiteren Datensätze mehr zur Verfügung stehen, wird AT END GO TO ENDE ausgeführt.

DIE READ-ANWEISUNG (Indexierte Datei - Sequentieller Zugriff)

Bei sequentiellem Zugriff macht die READ-Anweisung den nächsten Satz einer indexierten Datei verfügbar.

Format:

```
READ  Datei-Name  RECORD [WITH NO LOCK] [INTO Identifier]
      [AT END  unbedingte Anweisung 1]
      [NOT AT END unbedingte Anweisung 2]
      [END-READ]
```

Der in der READ-Anweisung angegebene Datei-Name muß einem in einer Dateibeschreibung auftretenden Datei-Namen und einem in einem FILE-CONTROL-Eintrag auftretenden Datei-Namen entsprechen. Der FILE-CONTROL-Eintrag muß die Klausel ORGANIZATION IS INDEXED aufweisen. Der FILE-CONTROL-Eintrag muß zusätzlich indirekt oder ausdrücklich die Klausel ACCESS MODE IS SEQUENTIAL aufweisen.

Die indexierte Datei kann Datensätze fester oder variabler Länge enthalten; eine Ausnahme besteht, wenn die INTO-Angabe vorliegt. Die INTO-Angabe nämlich beschränkt die indexierte Datei auf Sätze fester Satzlänge. Besteht die indexierte Datei aus Sätzen variabler Länge, dürfen die RECORD CONTAINS-Klausel und die Satzbeschreibungen nicht die vom Variable-Längen-Indikator (VLI) eingenommenen Bytes umfassen.

Vor Ausführung der READ-Anweisung muß die Datei entweder durch eine OPEN INPUT- oder durch eine OPEN I-O-Anweisung eröffnet worden sein.

Die erste Ausführung der READ-Anweisung trifft auf den ersten Datensatz der indexierten Datei gemäß der aufsteigenden Reihenfolge der Satzschlüssel, die der indexierten Datei zugeordnet sind. Der Satzschlüssel muß im FILE-CONTROL-Eintrag durch die RECORD KEY-Klausel benannt sein. Weitere Ausführungen der READ-Anweisungen für dieselbe indexierte Datei greifen auf die sequentiell nächsten Datensätze der Datei zu gemäß der aufsteigenden Reihenfolge der Satzschlüssel.

Wenn die Sätze einer Datei mit mehr als einer Satzdefinition beschrieben sind, nehmen diese automatisch denselben Speicherbereich ein. Jede Satzdefinition stellt somit eine stillschweigende Redefinition des Satz-Bereiches dar.

WITH NO LOCK-Angabe

Diese Angabe gestattet parallele Programmzugriffe auf den zu lesenden Block. Fehlt diese Angabe in der READ-Anweisung, wird der Block für gleichzeitige andere Operationen gesperrt.

INTO-Angabe

Ist INTO angegeben, wird der gelesene Datensatz aus dem Satz-Bereich zusätzlich in den mittels Identifier definierten Bereich kopiert. Diese Übertragung wird nach den für die MOVE-Anweisung festgelegten Regeln erreicht. Bei unkorrekter Ausführung der READ-Anweisung erfolgt die Übertragung jedoch nicht. Jedes dem Identifier zugeordnete Subskript oder jeder Index kommt zur Wirkung, nachdem der Satz gelesen wurde und wenn er in den Datenbereich übertragen wird.

Der Satz, der gelesen wird, ist anschließend im Satz-Bereich als auch in dem dem Identifier zugeordneten Datenbereich verfügbar. Der dem Identifier zugeordnete Datenbereich und der dem Datei-Namen zugeordnete Satz-Bereich dürfen nicht denselben Speicherbereich bilden.

Die INTO-Angabe darf nicht verwendet werden, wenn die indexierte Datei Sätze variabler Länge enthält.

AT END-Angabe

AT END muß für eine READ-Anweisung angegeben sein, die auf eine indexierte Datei Bezug nimmt, für die in den DECLARATIVES keine USE-Anweisung enthalten ist.

Wenn zur Zeit der Ausführung einer READ-Anweisung kein weiterer Datensatz mehr in der Datei ist, ist die AT END-Bedingung gegeben und die Ausführung der READ-Anweisung gilt als erfolglos. Danach ist der Inhalt des zugeordneten Satz-Bereiches unbestimmt.

Wenn die AT END-Bedingung auftritt, werden die folgenden Aktionen in der angegebenen Reihenfolge durchgeführt:

1. Ist die FILE STATUS-Klausel im FILE-CONTROL-Eintrag für diese indexierte Datei spezifiziert, wird der Wert 10 in das FILE STATUS-Feld übertragen, um die AT END-Bedingung anzuzeigen.

2. Ist die AT END-Angabe in der READ-Anweisung enthalten, wird die Steuerung auf die unbedingte Anweisung übertragen. Keine der etwaigen für diese Datei spezifizierten DECLARATIVE-Prozeduren wird ausgeführt.

3. Ist die AT END-Angabe nicht in der READ-Anweisung enthalten, werden für diese Datei DECLARATIVE-Prozeduren ausgeführt.

Wurde die AT END-Bedingung festgestellt, darf eine erneute READ-Anweisung für diese Datei nicht erfolgen, ohne daß ihr eine korrekte CLOSE-Anweisung mit anschließender korrekter OPEN-Anweisung vorangeht.

NOT AT END-Angabe

Ihre Verwendung ist fakultativ. Ist diese Angabe in der READ-Anweisung nicht vorhanden, gelangt der nächste COBOL-Satz zur Ausführung.

Beispiele folgen auf den nächsten Seiten:

Beispiel 1:

```
FILE-CONTROL.
    SELECT DATEI-N ASSIGN TO DISC
    ORGANIZATION IS INDEXED
    ACCESS MODE IS SEQUENTIAL
    RECORD KEY IS KTO-N.
DATA DIVISION.
FILE SECTION.
FD  DATEI-N BLOCK CONTAINS 20 RECORDS
            RECORD CONTAINS 25 CHARACTERS
            LABEL RECORD IS STANDARD.
01  SATZ-N.
    02  KTO-N    PIC X(5).
    02  FILLER   PIC X(20).

PROCEDURE DIVISION.
ANFANG.
    OPEN INPUT DATEI-N.
R.  READ DATEI-N AT END GO TO ENDE.
        ------------------------
    GO TO R.
ENDE.
    CLOSE DATEI-N.
```

Im obigen Beispiel handelt es sich um eine indexierte Datei mit sequentiellem Zugriff. Sie hat den Namen DATEI-N. KTO-N spezifiziert die jeweilige Position des in SATZ-N befindlichen Satzschlüsselfeldes. Damit Daten aus DATEI-N gelesen werden können, muß die Datei zuerst eröffnet werden mit der OPEN INPUT-Anweisung. Die READ-Anweisung greift sequentiell auf DATEI-N zu, um in SATZ-N den nächsten Datensatz aus der DATEI-N gemäß der aufsteigenden Reihenfolge der Satzschlüssel verfügbar zu machen.

Wenn die READ-Anweisung feststellt, daß in DATEI-N keine weiteren Datensätze zum Zugriff zur Verfügung stehen, dann wird AT END GO TO ENDE ausgeführt. Die CLOSE-Anweisung beendet die Bearbeitung der DATEI-N.

Beispiel 2:

```
FILE-CONTROL.
    SELECT DATEI-O  ASSIGN TO DISC
           ORGANIZATION IS INDEXED.
           ACCESS MODE IS SEQUENTIAL
           RECORD KEY IS FELD-4.
DATA DIVISION.
FILE SECTION.
FD  DATEI-O BLOCK CONTAINS 20 RECORDS
           RECORD CONTAINS 25 CHARACTERS
           LABEL RECORD IS STANDARD.
01  SATZ-O.
    02  FILLER  PIC X(10).
    02  FELD-4  PIC X(8).
    02  FILLER  PIC X(7).

PROCEDURE DIVISION.
ANFANG.
    OPEN I-O DATEI-O.
RE. READ DATEI-O AT END GO TO ENDE.
        ------------------------
    REWRITE SATZ-O  INVALID KEY GO TO FEHLER.
        ------------------------
    GO TO RE.
ENDE.
    CLOSE DATEI-O.
```

Hier handelt es sich um eine indexierte Datei mit sequentiellem Zugriff. FELD-4 spezifiziert das Satzschlüsselfeld. Nachdem Datensätze sowohl aus der Datei zu lesen als auch in sie zurückzuschreiben sind, muß die Datei mit OPEN I-O eröffnet werden. READ greift sequentiell auf DATEI-O zu, um in SATZ-O den nächsten Datensatz aus DATEI-O gemäß der aufsteigenden Folge der Satzschlüssel verfügbar zu machen.

Wird der Inhalt des eingelesenen Datensatzes verändert, schreibt die REWRITE-Anweisung den neuen Inhalt in die Satzposition der Datei zurück, aus der der Satz vorher mit READ gelesen wurde. Dabei durfte der Inhalt des Schlüsselfeldes zwischen READ und REWRITE nicht verändert worden sein. Widrigenfalls wird in das Unterprogramm FEHLER verzweigt.

Wenn die READ-Anweisung feststellt, daß in DATEI-O keine weiteren Datensätze zur Verfügung stehen, wird AT END GO TO ENDE ausgeführt.

DIE READ-ANWEISUNG (Indexierte Datei - Direktzugriff/ Dynamischer Zugriff)

Bei direktem Zugriff macht die READ-Anweisung einen bestimmten Satz aus einer indexierten Datei verfügbar.

Format 1 (Direktzugriff):

```
READ  Datei-Name  RECORD [WITH NO LOCK]
      [KEY IS Identifier 2]
      [INVALID KEY unbedingte Anweisung 1]
      [NOT INVALID KEY unbedingte Anweisung 2]
      [END-READ]
```

Beim dynamischen Zugriff werden ab einem bestimmten Satz in der Datei weitere Sätze sequentiell gelesen, und zwar in der Sequenz, wie sie sich aus den im Index einsortierten Schlüsselbegriffen ergibt.

Format 2 (Dynamischer Zugriff):

```
READ Datei-Name NEXT RECORD
     [WITH NO LOCK] [INTO Identifier]
     [AT END unbedingte Anweisung 1]
     [NOT AT END unbedingte Anweisung 2]
     [END-READ]
```

Der in der READ-Anweisung angegebene Datei-Name muß einem in einer Dateibeschreibung auftretenden Datei-Namen und einem in einem FILE-CONTROL-Eintrag auftretenden Datei-Namen entsprechen. Der FILE-CONTROL-Eintrag muß die Klauseln ORGANIZATION IS INDEXED und ACCESS MODE IS RANDOM oder DYNAMIC aufweisen.

Die indexierte Datei kann entweder Datensätze fester oder variabler Länge enthalten; eine Ausnahme besteht, wenn die INTO-Angabe vorliegt. Die INTO-Angabe nämlich beschränkt die indexierte Datei auf Sätze fester Länge. Besteht die indexierte Datei aus Sätzen variabler Länge, dann dürfen die RECORD CONTAINS Klausel und die Satzbeschreibungen nicht die von dem Variable-Längen-Indikator (VLI) eingenommenen Bytes umfassen.

Vor der Ausführung der READ-Anweisung muß die indexierte Datei durch eine OPEN INPUT- oder eine OPEN I-O-Anweisung eröffnet worden sein.

Die Ausführung des Direktzugriffs durch die READ-Anweisung veranlaßt, daß der Wert des Satzschlüsselfeldes im Speicher mit den im Index enthaltenen Werten verglichen wird. Dieser Vergleich läuft so lange, bis der Datensatz mit einem Satzschlüssel gefunden ist, der dem Wert des Satzschlüsselfeldes im Speicher entspricht. Der Datensatz, der den entsprechenden Satzschlüssel enthält, wird dann aus der Datei in den Satzbereich des Programmes eingelesen.

WITH NO LOCK-Angabe

Diese Angabe gestattet parallele Programmzugriffe auf den zu lesenden Block. Fehlt diese Angabe in der READ-Anweisung, wird der Block für gleichzeitige andere Operationen gesperrt.

INTO-Angabe

Ist INTO angegeben, wird der gelesene Datensatz aus dem Satz-Bereich zusätzlich in den mittels Identifier definierten Bereich kopiert. Diese Übertragung wird nach den für die MOVE-Anweisung festgelegten Regeln erreicht. Bei unkorrekter Ausführung der READ-Anweisung erfolgt die Übertragung jedoch nicht. Jedes dem Identifier zugeordnete Subskript oder jeder Index kommt zur Wirkung, nachdem der Satz gelesen wurde und wenn er in den Datenbereich übertragen wird.

Der Satz, der gelesen wird, ist anschließend sowohl im Satz-Bereich als auch in dem dem Identifier zugeordneten Datenbereich verfügbar. Der dem Identifier zugeordnete Datenbereich und der dem Datei-Namen zugeordnete Satz-Bereich dürfen nicht denselben Speicherbereich bilden.

Die INTO-Angabe darf nicht verwendet werden, wenn die indexierte Datei Sätze variabler Länge enthält.

KEY IS-Angabe

Soll auf einen bestimmten Daten-Satz anhand eines alternativen Schlüssels zugegriffen werden (definiert im FILE-CONTROL-Eintrag), weil der primäre Satzschlüssel nicht bekannt ist, ist dies mit Hilfe der KEY IS-Klausel zum Ausdruck zu bringen. Sehen Sie hierzu das Beispiel 1!

INVALID KEY-Angabe

Die INVALID KEY-Angabe muß für eine READ-Anweisung spezifiziert werden, die auf eine indexierte Datei mit direktem oder dynamischem Zugriff Bezug nimmt und für die keine USE-Anweisung in den DECLARATIVES existiert.

Wenn kein Datensatz in der indexierten Datei festgestellt werden kann, der einen Satzschlüssel aufweist, der dem Wert des Satzschlüsselfeldes im Speicher entspricht, tritt eine INVALID KEY-Bedingung ein und die Ausführung der READ-Anweisung gilt als erfolglos. Danach ist der Inhalt des Satz-Bereiches unbestimmt.

Wenn die INVALID KEY-Bedingung auftritt, werden die folgenden Aktionen in der angegebenen Reihenfolge durchgeführt:

1. Ist die FILE STATUS-Klausel im FILE-CONTROL-Eintrag für diese indexierte Datei spezifiziert, wird ein Wert 23 in das FILE STATUS-Feld übertragen, um anzuzeigen, daß kein Datensatz gefunden wurde.

2. Ist die INVALID KEY-Angabe in der READ-Anweisung vorhanden, wird die Steuerung auf die INVALID KEY-Anweisung übertragen. Keine der für diese Datei spezifizierten DECLARATIVE-Prozeduren wird ausgeführt.

3. Ist die INVALID KEY-Angabe in der READ-Anweisung nicht vorhanden, werden für diese Datei DECLARATIVE-Prozeduren ausgeführt.

Beispiele folgen auf den nächsten Seiten:

Beispiel 1:

```
FILE-CONTROL.
    SELECT DATEI-P  ASSIGN TO DISC
       ORGANIZATION IS INDEXED
       ACCESS MODE IS RANDOM
       RECORD KEY IS SCHL-P
       ALTERNATE RECORD KEY IS SCHL-PA WITH DUPLICATES.
DATA DIVISION.
FILE SECTION.
FD  DATEI-P BLOCK CONTAINS 42 RECORDS
            RECORD CONTAINS 12 CHARACTERS
            LABEL RECORD IS STANDARD.
01  SATZ-P.
    02  SCHL-P  PIC XXXX.
    02  SCHL-PA PIC X(3).
    02  FILLER  PIC X(5).
PROCEDURE DIVISION.
ANFANG.
    OPEN INPUT DATEI-P.
         ----------------------
    MOVE SCHL-T  TO  SCHL-P.
    READ DATEI-P INVALID KEY GO TO FEHLER.
         ----------------------
    MOVE SCHL-TA TO SCHL-PA.
    READ DATEI-P KEY IS SCHL-PA INVALID KEY GO TO FEHLER.
         ----------------------
    CLOSE DATEI-P.
```

Im obigen Beispiel handelt es sich um eine indexierte Datei mit Direktzugriff. Die Datei hat den Namen DATEI-P. SCHL-P spezifiziert die Position des primären Satzschlüsselfeldes. Damit Daten aus DATEI-P gelesen werden können, muß die Datei zuerst mit der OPEN INPUT-Anweisung eröffnet werden.

Der Programmierer bringt den im Datenfeld SCHL-T enthaltenen Wert in das Satzschlüsselfeld SCHL-P. Die READ-Anweisung macht dann den Datensatz der DATEI-P, der einen dem Inhalt von SCHL-P entsprechenden Satzschlüssel aufweist, zur Verarbeitung verfügbar. Nach korrekter Ausführung der READ-Anweisung befindet sich der Datensatz mit dem gewünschten Satzschlüssel im (SATZ-P genannten) Satz-Bereich. Wird in DATEI-P kein Datensatz gefunden, der einen entsprechenden Satzschlüssel hat, führt READ die Anweisung GO TO FEHLER aus.

Im zweiten Teil des Beispiels wird ein Direktzugriff anhand eines alternativen Schlüssels gezeigt. Vor dem Zugriff auf den Satz in der Datei muß der alternative Suchbegriff im für diesen Zweck vorgesehenen Feld SCHL-PA zur Verfügung gestellt werden. Dies geschieht mit MOVE SCHL-TA TO SCHL-PA. In der READ-Anweisung wird dann dafür gesorgt, daß dieser Suchbegriff mit den diesbezüglichen alternativen Schlüsselbegriffen im Index der Datei so lange verglichen wird, bis ein "Treffer" erzielt wird (wird der Index vergeblich abgesucht, nimmt die READ-Anweisung den INVALID KEY-Ausgang). Nun wird der gefundene Satz aus der Datei in den Bereich SATZ-P hereingelesen und steht zur Bearbeitung zur Verfügung. Sollten mehrere Sätze mit dem gleichen alternativen Schlüssel in der Datei existieren (Duplicates), wird hierbei der erste zur Verfügung gestellt. Interessiert man sich für diesen nicht, kann mit einer READ-NEXT-Schleife ein dynamisches Lesen der weiteren diesbezüglichen Sätze erreicht werden.

Beispiel 2:

```
FILE-CONTROL.
    SELECT DATEI-Q  ASSIGN TO DISC
           ORGANIZATION IS INDEXED
           ACCESS MODE IS RANDOM
           RECORD KEY IS NR-Q.

DATA DIVISION.
FILE SECTION.
FD  DATEI-Q BLOCK CONTAINS 51 RECORDS
           RECORD CONTAINS 10 CHARACTERS
           LABEL RECORD IS STANDARD.
01  SATZ-Q.
    02  NR-Q  PIC XXXX.
    02  FILLER  PIC X(6).

PROCEDURE DIVISION.
ANFANG.
    OPEN I-O DATEI-Q.
         ---------------------
    MOVE NR-D TO NR-Q.
    READ DATEI-Q INVALID KEY GO TO KEIN-SATZ.
         ---------------------
    REWRITE SATZ-Q INVALID KEY GO TO KEIN-SATZ.
         ---------------------
    CLOSE DATEI-Q.
```

Im obigen Beispiel handelt es sich um eine indexierte Datei mit Direktzugriff. Die Datei hat den Namen DATEI-Q. NR-Q spezifiziert die Position des Satzschlüsselfeldes. Da Datensätze sowohl aus der DATEI-Q zu lesen als auch in diese zurückzuschreiben sind, muß die Datei zuerst mit der OPEN I-O-Anweisung eröffnet werden.

Der Programmierer bringt den im Datenfeld NR-D enthaltenen Wert in das Satzschlüsselfeld NR-Q. Die READ-Anweisung macht dann den Datensatz der DATEI-Q, der einen dem Inhalt von NR-Q entsprechenden Satzschlüssel aufweist, zur Verarbeitung verfügbar. Nach korrekter Ausführung der READ-Anweisung befindet sich der Datensatz mit dem gewünschten Satzschlüssel im (SATZ-Q genannten) Satz-Bereich. Wird in DATEI-Q kein Datensatz gefunden, der einen entsprechenden Satzschlüssel hat, führt die READ-Anweisung GO TO KEIN-SATZ aus. Ansonsten kann der Satz jetzt bearbeitet werden, wobei das Schlüsselfeld nicht verändert werden darf.

Die REWRITE-Anweisung schreibt anschließend den Inhalt des Datensatzes in die Datei zurück.

Die INVALID KEY-Bedingung muß dabei abgefragt werden (dies verlangt der Compiler, damit sichergestellt ist, daß zwischen READ und REWRITE das primäre Satzschlüsselfeld nicht verändert wurde).

Beispiel 3:

```
FILE-CONTROL.
    SELECT DATEI-R  ASSIGN TO DISC
           ORGANIZATION IS INDEXED
           ACCESS MODE IS DYNAMIC
           RECORD KEY IS SCHL-R.

DATA DIVISION.
FILE SECTION
FD  DATEI-R BLOCK CONTAINS 25 RECORDS
           RECORD CONTAINS 20 CHARACTERS
           LABEL RECORD IS STANDARD.
01  SATZ-R.
    02  SCHL-R       PIC X(5).
    02  FILLER       PIC X(15).

PROCEDURE DIVISION.
ANFANG.
    OPEN INPUT DATEI-R.
         ----------------------
    MOVE "MOOO8" TO SCHL-R.
    START DATEI-R INVALID KEY GO TO FEHLER.
R1. READ DATEI-R NEXT RECORD AT END GO TO ENDE.
         ----------------------
    GO TO R1.
         ----------------------
    CLOSE DATEI-R.
```

Im obigen Beispiel handelt es sich um eine indexierte Datei mit dynamischen Zugriff.

Der Programmierer bringt "MOOO8" in das Satzschlüsselfeld SCHL-R. Die START-Anweisung veranlaßt, daß DATEI-R bei demjenigen Datensatz positioniert wird, dessen Satzschlüssel dem Inhalt von SCHL-R, nämlich MOOO8, entspricht. Wird in DATEI-R kein Datensatz gefunden, der einen Satzschlüssel gleich MOOO8 hat, führt die START-Anweisung GO TO FEHLER aus.

Nach korrekter Ausführung der START-Anweisung macht die READ NEXT-Anweisung den einen Satzschlüssel MOOO8 aufweisenden Datensatz in der DATEI-R zur Verarbeitung verfügbar. Nach korrekter Ausführung der READ NEXT-Anweisung befindet sich dieser Datensatz in SATZ-R und kann bearbeitet werden.

Durch nachfolgende Ausführungen der READ NEXT-Anweisung wird auf die (in Index-Reihenfolge) sequentiell nächsten Datensätze der Datei zugegriffen.

Wenn die READ NEXT-Anweisung feststellt, daß in DATEI-R keine weiteren Datensätze zur Verfügung stehen, wird AT END GO TO ENDE ausgeführt.

DIE RECEIVE-ANWEISUNG

Diese Anweisung dient dazu, eine Message (Information), ein Message-Segment oder einen Teil davon (sowie Aussagen über die Qualität dieser Informationen) aus einer durch MCS (Message Control System) verwalteten Queue dem Programm zur Verfügung zu stellen. Für den Fall, daß die Queue keine solchen Informationen aufweist, besitzt die RECEIVE-Anweisung eine NO DATA-Abzweigung.

Format:

```
RECEIVE CD-Name { READ-ONLY } INTO Identifier
                { MESSAGE   }
                { SEGMENT   }
  [NO DATA unbedingte Anweisung 1]
  [WITH DATA unbedingte Anweisung 2]
  [END-RECEIVE]
```

Unter CD-Name ist eine vorab im Programm definierte Input-Communication-Description (CD) anzugeben.

Die Inhalte von Daten-Name-1 (SYMBOLIC QUEUE) bis inklusive Daten-Name-4 (SYMBOLIC SUB-QUEUE-3) des bei CD-Name angegebenen Bereiches beschreiben die Struktur der Queue, aus der die Informationen bezogen werden sollen. Mehr hierüber finden Sie weiter vorn im Buch.

Die aus der Queue bezogenen Informationen werden in den als Identifier im Format anzugebenden Datenbereich übertragen, und zwar linksbündig ohne Auffüllung mit Spaces.

Die READ-ONLY-Klausel bewirkt, daß die bezogene Information nur kopiert wird, also weiterhin in der Queue verbleibt.

Wenn während der Ausführung einer RECEIVE-Anweisung MCS Informationen in den Identifier überträgt, geht die Steuerung auf die nächste Anweisung über, ungeachtet, ob die NO DATA-Klausel in der RECEIVE-Anweisung enthalten ist oder nicht.

Wenn während der Ausführung einer RECEIVE-Anweisung MCS keine Informationen in den Identifier überträgt, kommt es auf folgendes an:

- Bei programmierter NO DATA-Klausel wird die RECEIVE-Anweisung beendet unter Signalisierung des Abschlusses des Vorgangs, und die unbedingte Anweisung der NO DATA-Klausel wird ausgeführt.

- Bei fehlender NO DATA-Klausel wird der Programmlauf so lange unterbrochen, bis Daten im Identifier-1 angekommen sind.

- Sind eine oder mehrere im CD-Bereich definierte Queues oder Sub-Queues MCS nicht bekannt, geht die Steuerung auf die nächste Anweisung über, egal ob die NO DATA-Klausel in der RECEIVE-Anweisung vorkommt oder nicht.

Wenn MCS während der Ausführung einer RECEIVE-Anweisung Informationen in den Identifier überträgt und in der Anweisung die WITH DATA-Klausel programmiert wurde, geht die Steuerung auf die unbedingte Anweisung 2 über. Stellt dabei die unbedingte Anweisung 2 keine Abzweigung dar, geht die Steuerung nach der Beendigung von Anweisung 2 an das Ende der RECEIVE-Anweisung über.

Die in der Input-CD definierten Felder werden durch MCS bei jeder Ausführung der RECEIVE-Anweisung entsprechend aktualisiert.

Eine einmalige Ausführung einer RECEIVE-Anweisung stellt niemals mehr als eine einzelne Message (bei angegebener MESSAGE-Klausel) oder ein einzelnes Message-Segment (bei angegebener SEGMENT-Klausel) in Identifier-1 zur Verfügung. Andererseits gibt MCS keinen Teil einer Message an das Objekt-Programm frei, bevor nicht die gesamte Message in der Input-Queue komplett ist. Daran ändert auch eine in der RECEIVE-Anweisung angegebene SEGMENT-Klausel nichts.

Bei verwendeter MESSAGE-Klausel werden Segment-Ende-Indikatoren ignoriert und folgende Regeln gelten für die Datenübertragung:

- Paßt eine Message exakt in den mit Identifier bezeichneten Datenbereich, wird sie dort abgestellt.

- Ist eine Message kürzer als der mit Identifier bezeichnete Datenbereich, wird sie dort linksbündig ohne Space-Auffüllung rechts abgestellt.

- Ist eine Message länger als der mit Identifier bezeichnete Datenbereich, wird ihr erster Teil dort abgestellt und kann verarbeitet werden. Der Überhang der Message kann anschließend ebenfalls in den mit Identifier bezeichneten Datenbereich geholt werden, und zwar mit weiteren RECEIVE-Anweisungen, die sich auf dieselbe Queue, Sub-Queue, usw. beziehen. In Anbetracht dieser Regeln wird ein solcher Überhang jeweils wie eine neue Message behandelt.

Bei verwendeter SEGMENT-Klausel gelten für die Datenübertragung folgende Regeln:

- Paßt ein Segment exakt in den mit Identifier bezeichneten Datenbereich, wird es dort abgestellt.

- Ist ein Segment kürzer als der mit Identifier bezeichnete Datenbereich, wird es dort linksbündig ohne Space-Auffüllung rechts abgestellt.

- Ist ein Segment länger als der mit Identifier bezeichnete Datenbereich, wird dessen erster Teil dort abgestellt und kann verarbeitet werden. Der Überhang des Segments kann anschließend ebenfalls in den mit Identifier bezeichneten Datenbereich geholt werden, und zwar mit weiteren RECEIVE-Anweisungen, die sich auf dieselbe Queue, Sub-Queue, usw. beziehen. In Anbetracht dieser Regeln wird ein solcher Überhang jeweils wie ein neues Segment behandelt.

- Wird der auf diese Weise durch die RECEIVE-Anweisung beschaffte Text durch einen End-of-Message- oder durch einen End-of-Group-Indikator abgeschlossen, wird so verfahren, als handle es sich um einen End-of-Segment-Indikator, und der Text wird als Message-Segment behandelt.

DIE RELEASE-ANWEISUNG

Diese Anweisung dient in der Dateneingabephase einer Sortierung zur Übernahme der Daten in die Arbeitsdatei (Workfile). Nähere Ausführungen zur RELEASE-Anweisung finden Sie in einem eigenen Kapitel zum Thema Sortierung weiter hinten im Buch.

DIE REPLACE-ANWEISUNG

Diese Anweisung dient zum Austauschen von Ursprungsprogramm-Texten durch andere Texte während der Compilation. Die neuen Texte werden als voll gültige Bestandteile des Programmes in die Compilation einbezogen.

Format 1:

```
REPLACE  {==Pseudotext-1==
      BY  ==Pseudotext-2==} ...
```

Format 2:

```
REPLACE OFF
```

Die Pseudotexte werden durch je zwei Gleichheitszeichen am Anfang und am Ende eines Textes begrenzt.

Die REPLACE-Anweisung kann an jeder Stelle des Ursprungs-programmes zwischen den Spalten 8 und 72 (inklusive) stehen. Ihr muß ein Punkt vorausgehen, es sei denn, sie ist die erste Anweisung im Programm.

Die REPLACE-Anweisung ist mit einem Punkt abzuschließen.

Pseudotext-1 muß aus einem oder mehreren Text-Wörtern bestehen.

Pseudotext-2 kann aus null, einem oder mehreren Text-Wörtern bestehen.

Pseudotext-1 und Pseudotext-2 dürfen Fortsetzungszeilen bilden.

Ein Wort innerhalb eines Pseudotextes darf aus bis zu 322 Zeichen bestehen. Pseudotext 1 darf nicht nur aus einem Komma oder einem Semikolon bestehen.

In Format 1 der REPLACE-Anweisung stellt Pseudotext 1 den im Programm zu ersetzenden Text und Pseudotext 2 den neuen Text dar, der an die Stelle des bisherigen kommen soll.

Mit Format 2 der REPLACE-Anweisung wird zum Ausdruck gebracht, daß ab dieser Stelle des Programmes kein Textaustausch (replacing) mehr erfolgen soll. Das bedeutet, daß ein Textaustausch innerhalb des Programmes nur stattfindet ab einer REPLACE-Anweisung des Formats 1 bis hin zu einer weiteren REPLACE-Anweisung (Formate 1 und 2) oder bis das Ende des Programmes erreicht ist.

In einem Ursprungsprogramm enthaltene REPLACE-Anweisungen werden erst im Anschluß an etwaige vorhandene COPY-Anweisungen ausgeführt.

Beim Vergleich von Pseudotext 1 mit dem Ursprungsprogramm werden (sowohl in Pseudotext 1 als auch im Ursprungsprogramm) Kommata, Semikolons oder Leerstellen als eine einzelne Leerstelle betrachtet.

Kommentarzeilen und Leerzeilen werden von der REPLACE-Anweisung gänzlich ignoriert und übergangen, und zwar sowohl im Pseudotext 1 als auch im Ursprungsprogramm.

Debug-Zeilen werden dergestalt berücksichtigt, als wenn das D in Spalte 7 nicht vorhanden wäre.

Mit Ausnahme der Anweisungen COPY und REPLACE selbst erfolgt eine Syntax-Prüfung des Ursprungsprogrammes erst nach deren beendigter Durchführung.

DIE RETURN-ANWEISUNG

Diese Anweisung dient in der Datenausgabephase einer Sortierung zur Entnahme der sortierten Daten aus der Arbeitsdatei (Workfile). Nähere Ausführungen zur RETURN-Anweisung finden Sie in einem eigenen Kapitel zum Thema Sortierung weiter hinten im Buch.

DIE REWRITE-ANWEISUNG (Sequentielle Datei)

Die REWRITE-Anweisung schreibt einen Datensatz auf eine sequentielle Magnetplatten-Datei zurück.

Format:

REWRITE Satz-Name [FROM Identifier]

Der Satz-Name ist der Name eines Datensatzes in der FILE SECTION der DATA DIVISION. In der REWRITE-Anweisung kann der Satz-Name durch einen Datei-Namen qualifiziert werden.

Der in der REWRITE-Anweisung angegebene Satz-Name muß einem der Satz-Namen der Datei entsprechen. Der FILE-CONTROL-Eintrag der Datei muß indirekt oder ausdrücklich die Klausel ORGANIZATION IS SEQUENTIAL und die Klausel ACCESS MODE IS SEQUENTIAL aufweisen.

Eine sequentielle Datei kann Datensätze fester oder variabler Länge enthalten. Besteht die sequentielle Datei aus Sätzen variabler Länge, dürfen die RECORD CONTAINS-Klausel und die Satzbeschreibungen nicht die von dem Variable-Längen-Indikator (VLI) eingenommenen Bytes beinhalten.

Vor der Ausführung der REWRITE-Anweisung muß die Datei durch eine OPEN I-O-Anweisung eröffnet worden sein.

Die letzte für die Datei ausgeführte INPUT-OUTPUT-Anweisung vor Ausführung der REWRITE-Anweisung muß eine korrekt ausgeführte READ-Anweisung gewesen sein. Die REWRITE-Anweisung bewirkt das Zurückschreiben des Datensatzes, auf den durch die READ-Anweisung zugegriffen wurde.

FROM-Angabe

Ist FROM angegeben, findet vor dem Zurückschreiben eine Datenübertragung aus dem durch Identifier bezeichneten Bereich in den durch Satz-Name bezeichneten Bereich statt. Diese Übertragung erfolgt nach den für die MOVE-Anweisung bestimmten Regeln. Satz-Name und Identifier dürfen sich nicht auf denselben Speicherbereich beziehen.

Beispiel:

```
FILE-CONTROL.
    SELECT DATEI-H ASSIGN TO DISC
           ORGANIZATION IS SEQUENTIAL
           ACCESS MODE IS SEQUENTIAL.
DATA DIVISION.
FILE SECTION.
FD  DATEI-H BLOCK CONTAINS 20 RECORDS
           RECORD CONTAINS 25 CHARACTERS
           LABEL RECORD IS STANDARD.
01  SATZ-H  PIC X(25).

PROCEDURE DIVISION.
ANFANG.
    OPEN I-O DATEI-H.
R.  READ DATEI-H AT END GO TO ENDE.
       --------------
    REWRITE SATZ-H.
    GO TO R.
ENDE.
    CLOSE DATEI-H.
```

Im obigen Beispiel handelt es sich um eine sequentielle Datei mit dem Namen DATEI-H. Da Datensätze sowohl aus der DATEI-H zu lesen als auch in diese zurückzuschreiben sind, muß die Datei zuerst im I-O-Modus eröffnet werden. Die READ-Anweisung greift sequentiell auf DATEI-H zu, um dadurch einen Datensatz zur Verarbeitung verfügbar zu machen.

Wird der Inhalt eines Datensatzes während der Verarbeitung verändert, schreibt die REWRITE-Anweisung den Inhalt des SATZ-H genannten Satz-Bereiches in die Satz-Position von DATEI-H, auf die durch die letzte Ausführung der READ-Anweisung zugegriffen wurde.

Wenn bei der READ-Anweisung festgestellt wird, daß in DATEI-H keine weiteren Sätze stehen, dann wird AT END GO TO ENDE ausgeführt. Die CLOSE-Anweisung beendet die Bearbeitung der DATEI-H.

DIE REWRITE-ANWEISUNG (Relative Datei - Sequentieller Zugriff)

Bei sequentiellem Zugriff schreibt die REWRITE-Anweisung einen Datensatz auf eine relative Datei zurück.

Format:

REWRITE Satz-Name [FROM Identifier]

Der Satz-Name ist der Name eines Datensatzes in der FILE SECTION der DATA DIVISION. In der REWRITE-Anweisung kann der Satz-Name durch einen Datei-Namen qualifiziert werden.

Der in der REWRITE-Anweisung angegebene Satz-Name muß einem der Satz-Namen der Datei entsprechen. Der FILE-CONTROL-Eintrag der Datei muß die Klausel ORGANIZATION IS RELATIVE und indirekt oder ausdrücklich die Klausel ACCESS MODE IS SEQUENTIAL aufweisen.

Vor Ausführung der REWRITE-Anweisung muß die relative Datei durch eine OPEN-I-O-Anweisung eröffnet worden sein und die letzte für die Datei ausgeführte INPUT-OUTPUT-Anweisung vor Ausführung der REWRITE-Anweisung muß eine korrekt ausgeführte READ-Anweisung gewesen sein. Die REWRITE-Anweisung bewirkt das Zurückschreiben desjenigen Datensatzes, auf den durch die READ-Anweisung zugegriffen wurde.

FROM-Angabe

Ist FROM angegeben, findet vor dem Zurückschreiben eine Datenübertragung aus dem durch Identifier bezeichneten Bereich in den durch Satz-Name bezeichneten Bereich statt. Diese Übertragung erfolgt nach den für die MOVE-Anweisung bestimmten Regeln. Satz-Name und Identifier dürfen sich nicht auf denselben Speicherbereich beziehen.

Beispiel:

```
FILE-CONTROL.
    SELECT DATEI-J  ASSIGN TO DISC
           ORGANIZATION IS RELATIVE
           ACCESS MODE IS SEQUENTIAL
           RELATIVE KEY IS SCHL-J.
DATA DIVISION.
FILE SECTION.
FD  DATEI-J BLOCK CONTAINS 17 RECORDS
           RECORD CONTAINS 30 CHARACTERS
           LABEL RECORD IS STANDARD.
01  SATZ-J  PIC X(30).
WORKING-STORAGE SECTION.
01  SCHL-J  PIC 999.
PROCEDURE DIVISION.
A.
    OPEN I-O DATEI-J.
B.  READ DATEI-J AT END GO TO ENDE.
       --------------
    REWRITE SATZ-J.
    GO TO B.
```

Im obigen Beispiel handelt es sich um eine relative Datei mit sequentiellem Zugriff. Sie hat den Namen DATEI-J. Da Datensätze sowohl aus der DATEI-J zu lesen als auch in diese zurückzuschreiben sind, muß die Datei zuerst im I-O-Modus eröffnet werden.

Die READ-Anweisung greift sequentiell auf DATEI-J zu, um dadurch den jeweils nächsten Datensatz von DATEI-J verfügbar zu machen. Jeder nicht belegte Record-Slot der relativen Datei wird von der READ-Anweisung ignoriert. Nach der korrekten Ausführung der READ-Anweisung enthält das Satzschlüsselfeld SCHL-J die relative Nummer des Datensatzes, auf den durch die READ-Anweisung zugegriffen wurde.

Wird der Inhalt des Datensatzes verändert, schreibt die REWRITE-Anweisung den Inhalt des Satz-Bereiches zurück in die Satz-Position der Datei, auf die durch die letzte Ausführung der READ-Anweisung zugegriffen wurde.

Wenn die READ-Anweisung feststellt, daß in DATEI-J keine weiteren Sätze stehen, wird AT END GO TO ENDE ausgeführt.

DIE REWRITE-ANWEISUNG (Relative Datei - Direktzugriff/ Dynamischer Zugriff

Hierbei schreibt die REWRITE-Anweisung einen vorher gelesenen Satz auf seinen Platz in der relativen Datei zurück.

Format:

```
REWRITE  Satz-Name  [FROM Identifier]
         [INVALID KEY  unbedingte Anweisung 1]
         [NOT INVALID KEY  unbedingte Anweisung 2]
         [END-REWRITE]
```

Der Satz-Name ist der Name eines Datensatzes in der FILE SECTION der DATA DIVISION. In der REWRITE-Anweisung kann der Satz-Name durch einen Datei-Namen qualifiziert werden.

Der in der REWRITE-Anweisung angegebene Satz-Name muß einem der Satz-Namen der Datei entsprechen. Der FILE-CONTROL-Eintrag der Datei muß die Klauseln ORGANIZATION IS RELATIVE und ACCESS MODE IS RANDOM oder DYNAMIC aufweisen.

Vor Ausführung der REWRITE-Anweisung muß die relative Datei durch eine OPEN I-O-Anweisung eröffnet worden sein und es muß eine READ-Anweisung ausgeführt worden sein.

Ist FROM angegeben, findet vor dem Zurückschreiben eine Datenübertragung aus dem durch Identifier bezeichneten Bereich in den durch Satz-Name bezeichneten Bereich statt. Diese Übertragung erfolgt nach den für die MOVE-Anweisung bestimmten Regeln. Satz-Name und Identifier dürfen sich nicht auf denselben Speicherbereich beziehen.

INVALID KEY-Angabe

Die INVALID KEY-Angabe muß für eine REWRITE-Anweisung spezifiziert werden, die auf eine relative Datei mit direktem oder dynamischem Zugriff Bezug nimmt, für die keine USE-Anweisung in den DECLARATIVES vorgesehen wurde.

Weist die relative Datei, auf die ein Satz zurückgeschrieben werden soll, keinen gültigen Datensatz in der durch das relative Satzschlüsselfeld spezifizierten Position auf, tritt eine INVALID KEY-Bedingung auf und die Ausführung der REWRITE-Anweisung gilt als erfolglos.

Wenn die INVALID KEY-Bedingung auftritt, werden folgende Aktionen in der festgelegten Reihenfolge durchgeführt:

1. Ist die FILE STATUS-Klausel im FILE-CONTROL-Eintrag für diese Datei spezifiziert, dann wird der Wert 23 in das FILE STATUS-Datenfeld übertragen, um anzuzeigen, daß kein Datensatz in der durch das relative Satzschlüsselfeld spezifizierten Datensatzposition gefunden wurde.

2. Ist die INVALID KEY-Angabe in der REWRITE-Anweisung vorhanden, wird die Steuerung auf die dortige unbedingte Anweisung übertragen. Keine der für diese Datei spezifizierten DECLARATIVE-Prozeduren wird ausgeführt.

3. Ist die INVALID KEY-Angabe in der REWRITE-Anweisung nicht vorhanden, werden für diese Datei DECLARATIVE-Prozeduren ausgeführt.

Ein Beispiel finden Sie auf der nächsten Seite:

Beispiel:

```
FILE-CONTROL.
    SELECT DATEI-L  ASSIGN TO DISC
            ORGANIZATION IS RELATIVE
            ACCESS MODE IS RANDOM, RELATIVE KEY IS SCHL-L.

DATA DIVISION.
FILE SECTION.
FD  DATEI-L BLOCK CONTAINS 25 RECORDS
            RECORD CONTAINS 20 CHARACTERS
            LABEL RECORD IS STANDARD.
01  SATZ-L  PIC X(20).

WORKING-STORAGE SECTION.
01  SCHL-L  PIC 9999.

PROCEDURE DIVISION.
ANFANG.
    OPEN I-O DATEI-L.
     -------------
    MOVE 20 TO SCHL-L.
    READ DATEI-L INVALID KEY GO TO FEHLER-1.
     -------------
    REWRITE SATZ-L INVALID KEY GO TO FEHLER-2.
```

Im obigen Beispiel handelt es sich um eine relative Datei mit Direktzugriff. Die Datei hat den Namen DATEI-L. Da Datensätze sowohl aus DATEI-L zu lesen als auch in diese zurückzuschreiben sind, muß sie mit der OPEN I-O-Anweisung eröffnet werden.

Der Programmierer bringt die Zahl 20 in das Satzschlüsselfeld SCHL-L, um die relative Satznummer des Datensatzes, auf den Zugriff genommen wird, zu spezifizieren. Die READ-Anweisung macht den 20sten Datensatz der DATEI-L zur Verarbeitung verfügbar. Nach der korrekten Ausführung der READ-Anweisung befindet sich der 20ste Datensatz der DATEI-L in SATZ-L. Weist die 20ste Datensatzposition der DATEI-L keinen Datensatz auf, führt die READ-Anweisung in ihrer INVALID KEY-Angabe die Anweisung GO TO FEHLER-1 aus.

Hinweis: Bei einer relativen Datei mit dynamischem Zugriff (ACCESS MODE DYNAMIC) kann ein mittels READ NEXT eingelesener Satz ebenfalls mit REWRITE zurückgeschrieben werden.

Die REWRITE-Anweisung schreibt den Inhalt von SATZ-L auf die Satzposition mit der relativen Nummer 20 zurück.

Die INVALID KEY-Klausel in der REWRITE-Anweisung dient dem Zweck, sicherzustellen, daß nicht zwischen READ und REWRITE das relative Satzschlüsselfeld durch den Programmierer verändert wird und somit unter Umständen ein ganz anderer Satz beim Zurückschreiben in die Datei überschrieben wird.

DIE REWRITE-ANWEISUNG (Indexierte Datei - Sequentieller Zugriff)

Bei sequentiellem Zugriff schreibt die REWRITE-Anweisung einen Datensatz auf eine indexierte Datei zurück.

Format:

```
REWRITE  Satz-Name  [FROM Identifier]
```

Der Satz-Name ist der Name eines Datensatzes in der FILE SECTION der DATA DIVISION. In der REWRITE-Anweisung kann der Satz-Name durch einen Datei-Namen qualifiziert werden.

Der in der REWRITE-Anweisung angegebene Satz-Name muß einem der Satz-Namen der Datei entsprechen. Der FILE-CONTROL-Eintrag der Datei muß die Klausel ORGANIZATION IS INDEXED und indirekt oder ausdrücklich die Klausel ACCESS MODE IS SEQUENTIAL aufweisen.

Die indexierte Datei kann entweder Datensätze fester oder variabler Länge enthalten. Besteht die indexierte Datei aus Sätzen variabler Länge, dürfen die RECORD CONTAINS-Klausel und die Satzbeschreibungen nicht die von dem Variable-Längen-Indikator (VLI) eingenommenen Bytes beinhalten.

Vor Ausführung der REWRITE-Anweisung muß die indexierte Datei durch eine OPEN I-O-Anweisung eröffnet worden sein und die letzte für die Datei ausgeführte INPUT-OUTPUT-Anweisung vor Ausführung der REWRITE-Anweisung muß eine korrekt ausgeführte READ-Anweisung gewesen sein. Die REWRITE-Anweisung bewirkt das Zurückschreiben desjenigen Datensatzes, auf den durch die READ-Anweisung zugegriffen wurde. Bei der Bearbeitung des Datensatzes (zwischen READ und REWRITE) muß das primäre Satzschlüsselfeld unangetastet bleiben.

FROM-Angabe

Ist FROM angegeben, findet vor dem Zurückschreiben eine Datenübertragung aus dem durch Identifier bezeichneten Bereich in den durch Satz-Name bezeichneten Bereich statt. Diese Übertragung erfolgt nach den für die MOVE-Anweisung bestimmten Regeln. Satz-Name und Identifier dürfen sich nicht auf denselben Speicherbereich beziehen.

Ein Beispiel finden Sie auf der nächsten Seite:

Beispiel:

```
FILE-CONTROL.
    SELECT DATEI-O  ASSIGN TO DISC
           ORGANIZATION IS INDEXED
           ACCESS MODE IS SEQUENTIAL
           RECORD KEY IS FELD-4.
DATA DIVISION.
FILE SECTION.
FD  DATEI-O BLOCK CONTAINS 20 RECORDS
           RECORD CONTAINS 25 CHARACTERS
           LABEL RECORD IS STANDARD.
01  SATZ-O.
    02  FILLER  PIC X(10).
    02  FELD-4  PIC X(8).
    02  FILLER  PIC X(7).

PROCEDURE DIVISION.
ANFANG.
    OPEN I-O DATEI-O.
RE. READ DATEI-O AT END GO TO ENDE.
       ---------------
    REWRITE SATZ-O.
    GO TO RE.
```

Im obigen Beispiel handelt es sich um eine indexierte Datei mit sequentiellem Zugriff. Die Datei hat den Namen DATEI-O. FELD-4 spezifiziert die Position des Satzschlüsselfeldes im Datensatz. Da Datensätze sowohl aus DATEI-O zu lesen als auch in diese zurückzuschreiben sind, muß die Datei zuerst im I-O-Modus eröffnet werden. Die READ-Anweisung greift sequentiell auf DATEI-O zu, um den nächsten Datensatz aus DATEI-O verfügbar zu machen gemäß der aufsteigenden Reihenfolge der Satzschlüssel im Index.

Wird der Inhalt des gelesenen Datensatzes verändert, schreibt die REWRITE-Anweisung den Inhalt von SATZ-O in die Datensatzposition der Datei zurück, auf die durch die letzte Ausführung der READ-Anweisung zugegriffen wurde.

DIE REWRITE-ANWEISUNG (Indexierte Datei - Direktzugriff/ Dynamischer Zugriff)

Hierbei schreibt die REWRITE-Anweisung einen vorher gelesenen Datensatz auf seinen Platz in der indexierten Datei zurück.

Format:

```
REWRITE  Satz-Name  [FROM Identifier]
         [INVALID KEY  unbedingte Anweisung 1]
         [NOT INVALID KEY  unbedingte Anweisung 2]
         [END-REWRITE]
```

Der Satz-Name ist der Name eines Datensatzes in der FILE SECTION der DATA DIVISION. In der REWRITE-Anweisung kann der Satz-Name durch einen Datei-Namen qualifiziert werden.

Der in der REWRITE-Anweisung angegebene Satz-Name muß einem der Satz-Namen der Datei entsprechen. Der FILE-CONTROL-Eintrag der Datei muß die Klauseln ORGANIZATION IS INDEXED und ACCESS MODE IS RANDOM oder DYNAMIC aufweisen.

Die indexierte Datei kann Datensätze fester oder variabler Länge enthalten. Besteht die indexierte Datei aus Sätzen variabler Länge, dann dürfen die RECORD CONTAINS-Klausel und die Satzbeschreibungen nicht die von dem Variable-Längen-Indikator (VLI) eingenommenen Bytes beinhalten.

Vor Ausführung der REWRITE-Anweisung muß die Datei durch eine OPEN I-O-Anweisung eröffnet worden sein und es muß eine READ-Anweisung ausgeführt worden sein.

FROM-Angabe

Ist FROM angegeben, findet vor dem Zurückschreiben eine Datenübertragung aus dem durch Identifier bezeichneten Bereich in den durch Satz-Name bezeichneten Bereich statt. Diese Übertragung erfolgt nach den für die MOVE-Anweisung bestimmten Regeln. Satz-Name und Identifier dürfen sich nicht auf denselben Speicherbereich beziehen.

INVALID KEY-Angabe

Die INVALID KEY-Angabe muß für eine REWRITE-Anweisung spezifiziert werden, die auf eine indexierte Datei mit direktem oder dynamischem Zugriff Bezug nimmt, für die keine zugeordnete USE-Anweisung in den DECLARATIVES vorgesehen wurde.

Bei der Bearbeitung des Datensatzes (zwischen READ und REWRITE) muß das primäre Satzschlüsselfeld unangetastet bleiben.

Andernfalls tritt die INVALID KEY-Bedingung auf und es werden die folgenden Aktionen in der festgelegten Reihenfolge durchgeführt:

1. Ist die FILE STATUS-Klausel im FILE-CONTROL-Eintrag für diese Datei spezifiziert, dann wird der Wert 23 in das FILE STATUS-Datenfeld übertragen, um anzuzeigen, daß kein Datensatz gefunden wurde.

2. Ist die INVALID KEY-Angabe in der REWRITE-Anweisung vorhanden, wird die Steuerung auf die dortige unbedingte Anweisung übertragen. Keine der für diese Datei spezifizierten DECLARATIVE-Prozeduren wird ausgeführt.

3. Ist die INVALID KEY-Angabe in der REWRITE-Anweisung nicht vorhanden, werden für diese Datei DECLARATIVE-Prozeduren ausgeführt.

Auf den nächsten Seiten finden Sie ein Beispiel:

Beispiel:

```
FILE-CONTROL.
    SELECT DATEI-Q ASSIGN TO DISC
           ORGANIZATION IS INDEXED
           ACCESS MODE IS DYNAMIC
           RECORD KEY IS NR-Q.
DATA DIVISION.
FILE SECTION.
FD  DATEI-Q BLOCK CONTAINS 51 RECORDS
           RECORD CONTAINS 10 CHARACTERS
           LABEL RECORD IS STANDARD.
01  SATZ-Q.
    02  NR-Q     PIC XXXX.
    02  FILLER   PIC X(6).

PROCEDURE DIVISION.
ANFANG.
    OPEN I-O DATEI-Q.
       -----------
    MOVE NR-D TO NR-Q.
    READ DATEI-Q INVALID KEY GO TO KEIN-SATZ.
       -----------
    REWRITE SATZ-Q  INVALID KEY GO TO FEHLER.
       -----------
    MOVE NR-D TO NR-Q.
    START DATEI-Q KEY NOT < NR-Q
          INVALID KEY GO TO KEIN-SATZ.
SCHLEIFE.
    READ DATEI-Q NEXT AT END GO TO ENDE.
       ----------
    REWRITE SATZ-Q INVALID KEY GO TO FEHLER.
    GO TO SCHLEIFE.
```

Im obigen Beispiel handelt es sich um eine indexierte Datei mit Direktzugriff und dynamischem Zugriff. Die Datei hat den Namen DATEI-Q. NR-Q spezifiziert die Position des Satzschlüsselfeldes. Nachdem Datensätze sowohl aus DATEI-Q zu lesen als auch in diese zurückzuschreiben sind, muß die Datei zuerst mit der OPEN I-O-Anweisung eröffnet werden.

Die erste Anweisung MOVE NR-D TO NR-Q stellt den Schlüsselbegriff zur Verfügung für den unmittelbar folgenden Direkt-(Random-)Zugriff auf die Datei mittels READ... INVALID KEY... Durch die READ-Anweisung erfolgt ein Absuchen des Datei-Indexes anhand des Schlüsselbegriffs. Wird im Index ein identischer Schlüsselbegriff gefunden, kann anhand der

beigefügten Adresse der Satz random aus der Datei gelesen und anschließend bearbeitet werden (ansonsten handelt es sich um einen falschen Schlüsselbegriff und der Logikablauf beschreitet den INVALID KEY-Ausgang der READ-Anweisung). Nach der Bearbeitung des Satzes stellt die REWRITE-Anweisung den Satz wieder an seinen Platz in der Datei zurück. Der INVALID KEY-Ausgang der REWRITE-Anweisung ist obligatorisch und dient folgendem Zweck: Es muß stets sichergestellt sein, daß zwischen einem READ und einem REWRITE das primäre Satzschlüsselfeld nicht verändert wird! (Bei alternativen Satzschlüsselfeldern ist eine Veränderung jedoch statthaft.)

Die zweite Anweisung MOVE NR-D TO NR-Q stellt den Anfangs-Schlüsselbegriff zur Verfügung für den unmittelbar darauf folgenden dynamischen Zugriff auf die Datei. Die START-Anweisung dient zum Absuchen des Datei-Indexes nach einem Begriff, der mit dem in NR-Q vorgegebenen Schlüsselbegriff identisch ist oder ihm am nächsten kommt (trifft beides nicht zu, wird der INVALID KEY-Ausgang der START-Anweisung beschritten) und es wird im Falle der Fündigkeit an dieser Stelle des Indexes eine Markierung gesetzt.

Die nachfolgende READ-NEXT-Anweisung stellt dann denjenigen Satz aus der Datei zur Verfügung, dessen Schlüsselbegriff und Adresse sich aus der markierten Stelle im Index ergeben. Der gelesene Satz kann bearbeitet und danach mit der REWRITE-Anweisung wieder in die Datei zurückgestellt werden, wobei der INVALID KEY-Ausgang der REWRITE-Anweisung zur Absicherung dient, daß der Programmierer zwischen READ und REWRITE nicht das primäre Satzschlüsselfeld verändert.

Im Zuge der Programm-Schleife liest die READ-NEXT-Anweisung anschließend einen weiteren Satz aus der Datei, dessen Adresse wiederum aus dem nächsten Index-Eintrag entnommen wird, usw.

DIE SEARCH-ANWEISUNG

Die SEARCH-Anweisung sucht eine Tabelle auf einen Tabellenposten hin sequentiell ab, der eine gegebene Bedingung erfüllt, und stellt die Nummer des ggf. gefundenen Postens im Indexfeld zur Verfügung.

Format:

```
SEARCH Identifier-1  [VARYING {Identifier-2 }]
                              {Index-Name-1 }
     [AT END  unbedingte Anweisung-1]
      WHEN Bedingung-1 {unbedingte Anweisung-2}
                       {NEXT SENTENCE         }
     [WHEN Bedingung-2 {unbedingte Anweisung-3}]
                       {NEXT SENTENCE         }  ...
     [END-SEARCH]
```

Identifier-1 muß den Namen des Tabellenpostens ausweisen. Die Datendefinition von Identifier-1 muß eine OCCURS-Klausel mit einer INDEXED BY-Angabe enthalten. Auf Identifier-1 darf in der SEARCH-Anweisung keine Klammer mit Subskript- oder Index-Angabe folgen.

Bedingung-1, Bedingung-2, etc. können beliebige bedingte Ausdrücke sein. Ein bedingter Ausdruck ist entweder eine einfache oder eine Mehrfach-Bedingung, wie sie auch bei der IF-Anweisung Verwendung finden (siehe dort).

Durch die SEARCH-Anweisung erfolgt ein serielles Absuchen der Tabelle, beginnend mit dem Inhalt des dem Identifier-1 zugeordneten Indexes. Vor Ausführung der SEARCH-Anweisung muß der Programmierer die gewünschte Anfangs-Tabellenpostennummer in den Index einsetzen. Die Absuchoperation endet unmittelbar zu Beginn ihrer Ausführung, wenn der Index einen Wert enthält, der einer Tabellenpostennummer entspricht, die größer ist als die höchstzulässige Tabellenpostennummer für Identifier-1. Es wird dann die unbedingte Anweisung-1 (AT END) ausgeführt. Ist jedoch keine AT END-Angabe vorhanden, geht die Steuerung zum nächsten ausführbaren Befehl nach dem Punkt über.

Die SEARCH-Anweisung wertet die Bedingungen in der Reihenfolge aus, in der sie geschrieben sind. Wird keine der Bedingungen in einem Tabellenposten erfüllt, wird der Index um 1 erhöht.

Der Vergleich wird mit dem neuen Indexinhalt wiederholt. Entspricht der Wert des Indexes einer den letzten Tabellenposten überschreitenden Adresse, endet die Absuche und es wird die unbedingte Anweisung-1 (AT END) ausgeführt. Ist keine AT END-Angabe vorhanden, geht die Steuerung zum nächsten ausführbaren Befehl nach dem Punkt über. Ist eine der Bedingungen erfüllt, endet die Suchoperation, und die dieser Bedingung zugeordnete unbedingte Anweisung wird ausgeführt. Der Inhalt des Indexfeldes behält die ermittelte Tabellenpostennummer bei. Steht NEXT SENTENCE für die unbedingte Anweisung, geht die Steuerung zum nächsten ausführbaren Befehl nach dem Punkt über.

Zum Absuchen einer zwei- oder dreidimensionalen Tabelle muß eine SEARCH-Anweisung mehrmals ausgeführt werden.

VARYING-Klausel

Sind mehr als ein Index-Name in der INDEXED BY-Klausel der Beschreibung des Tabellenpostens vergeben worden, kann die VARYING-Klausel der SEARCH-Anweisung verwendet werden. Diese Klausel bestimmt ausdrücklich den Index, der für die Suchoperation verwendet werden soll. Wird die VARYING-Klausel weggelassen, wird der erste in der INDEXED BY-Klausel für Identifier-1 definierte Index für die Suchoperation benützt. Alle anderen (Identifier-1 zugeordneten) Indizes bleiben unverändert.

Ist Identifier-2 in der VARYING-Klausel angegeben, muß es sich entweder um ein Index-Datenfeld (USAGE IS INDEX) oder um ein numerisches Elementarfeld handeln. Identifier-2 wird zugleich mit dem (Identifier-1 zugeordneten) Index erhöht.

Beispiel 1:

```
DATA DIVISION.
01  LAG-TAB.
    02  LAG-POSTEN OCCURS 10 TIMES INDEXED BY LAG-IND.
        03  LAGER-NR     PIC 9(5).
        03  PREIS-CODE   PIC 999.
        03  HANDLAGER    PIC 9999.
        03  RUECK-MENGE  PIC 9999.
        03  BEST-GRENZE  PIC 999.
WORKING-STORAGE SECTION.
01  LAG-NR PIC 9(5).

PROCEDURE DIVISION.
    SET LAG-IND TO 1.
    SEARCH LAG-POSTEN AT END GO TO NICHT-DA
        WHEN LAG-NR IS EQUAL TO LAGER-NR (LAG-IND)
            GO TO GEFUNDEN.

GEFUNDEN.
     ----------------
NICHT-DA.
     ----------------
```

In Beispiel 1 geht es um eine (LAG-TAB genannte) Tabelle mit 10 (LAG-POSTEN genannten) Posten. LAG-IND ist der Index, der verwendet wird, um auf Posten in der Tabelle zuzugreifen. Die SET-Anweisung setzt den Index LAG-IND auf 1.

Die SEARCH-Anweisung beginnt demzufolge die Suchoperation beim ersten Posten der Tabelle. Weist der erste Tabellenposten keine dem Datenfeld LAG-NR entsprechende Lagernummer auf, wird der Index LAG-IND auf die Tabellenpostennummer 2 erhöht und SEARCH untersucht die Lagernummer im zweiten Tabellenposten, usw.

Die SEARCH-Anweisung kann auf folgende zwei Arten enden:

1. Das Tabellenende wird erreicht: GO TO NICHT-DA wird ausgeführt.

2. Ein Posten erfüllt die Bedingung in der WHEN-Angabe: GO TO GEFUNDEN wird ausgeführt.

Beispiel 2:

```
DATA DIVISION.
01   LAG-TAB.
     02   LAG-POSTEN OCCURS 10 TIMES INDEXED BY LAG-IND.
          03   LAGER-NR     PIC 9(5).
          03   PREIS-CODE   PIC 999.
          03   HANDLAGER    PIC 9999.
          03   RUECK-MENGE  PIC 9999.
          03   LIMIT1       PIC 999.

PROCEDURE DIVISION.
     SET LAG-IND TO 5.
     SEARCH LAG-POSTEN AT END GO TO JA-WEG
          WHEN HANDLAGER (LAG-IND) < LIMIT1 (LAG-IND)
                    MOVE HANDLAGER (LAG-IND) TO IN-ARBEIT
          WHEN HANDLAGER (LAG-IND) = 0
                              GO TO NEIN-WEG.
     ADD 100 TO IN-ARBEIT.
          ------------
          ------------
JA-WEG.
          ------------
NEIN-WEG.
          ------------
```

Die SEARCH-Anweisung beginnt ihre Absuchoperation beim fünften Posten der Tabelle LAG-TAB, da der LAG-IND vorher auf 5 gesetzt wurde. Erfüllt der fünfte Tabellenposten keine der beiden in den WHEN-Klauseln enthaltenen Bedingungen, wird LAG-IND um 1 heraufgesetzt und SEARCH untersucht den sechsten Tabellenposten auf das Vorhandensein einer der beiden Bedingungen, usw.

Die SEARCH-Anweisung kann auf folgende drei Arten enden:

1. Das Tabellenende wird erreicht: GO TO JA-WEG wird ausgeführt.

2. Die Bedingung in der ersten WHEN-Angabe ist erfüllt: Die Anweisungen MOVE HANDLAGER (LAG-IND) TO IN-ARBEIT und ADD 100 TO IN-ARBEIT ... werden ausgeführt.

3. Die Bedingung in der zweiten WHEN-Angabe ist erfüllt: GO TO NEIN-WEG wird ausgeführt.

Beispiel 3:

```
DATA DIVISION.
01  LAG-TAB.
    02  LAG-POSTEN OCCURS 10 TIMES INDEXED BY LAG-IND.
        03  LAGER-NR     PIC 9(5).
        03  PREIS-CODE   PIC 999.
        03  HANDLAGER    PIC 9999.
        03  RUECK-MENGE  PIC 9999.
        03  LIMIT1       PIC 999.

PROCEDURE DIVISION.
    SET LAG-IND TO 4.
    SEARCH LAG-POSTEN AT END GO TO JA-WEG
        WHEN HANDLAGER (LAG-IND) IS LESS THAN
                LIMIT1  (LAG-IND)
          OR HANDLAGER (LAG-IND) IS EQUAL TO ZERO
          MOVE HANDLAGER (LAG-IND) TO IN-ARBEIT.
    MOVE "A" TO CODE-W.
      --------------
JA-WEG.
      --------------
```

Die SEARCH-Anweisung beginnt mit der Suche beim vierten Posten der Tabelle LAG-TAB, da der LAG-IND vorher auf 4 gesetzt wurde. Die SEARCH-Anweisung enthält eine WHEN-Angabe. Die Bedingung in der WHEN-Angabe ist eine kombinierte Bedingung (zwei durch OR verbundene Vergleichs-Bedingungen). Erfüllt der vierte Tabellenposten keine der beiden WHEN-Bedingungen, wird LAG-IND um 1 erhöht. Die Vergleichsoperation wird wiederholt und untersucht den fünften Tabellenposten, usw.

Die SEARCH-Anweisung kann auf folgende drei Arten enden:

1. Das Tabellenende wird erreicht: GO TO JA-WEG wird ausgeführt.

2. Die erste Bedingung in der WHEN-Angabe ist erfüllt: Die Anweisungen MOVE HANDLAGER (LAG-IND) TO IN-ARBEIT und MOVE "A" TO CODE-W werden ausgeführt.

3. Die zweite Bedingung in der WHEN-Angabe ist erfüllt: Die Anweisungen MOVE HANDLAGER (LAG-IND) TO IN-ARBEIT und MOVE "A" TO CODE-W werden ausgeführt.

Beispiel 4:

```
DATA DIVISION.
01   LAG-TAB.
     02   LAG-POSTEN OCCURS 30 TIMES INDEXED BY LAG-IND.
          03   LAGER-NR     PIC 9(5).
          03   PREIS-CODE   PIC 999.
          03   HANDLAGER    PIC 9999.
          03   RUECK-MENGE  PIC 9999.
          03   BEST-GRENZE  PIC 999.
01   PREIS-TAB.
     02   PREIS-POSTEN OCCURS 30 TIMES INDEXED BY PREIS-IND.
          03   PREIS-CODE-2     PIC 999.
          03   PREIS-DATEN      PIC X(26).
01   ANF-FELD     PIC 99.
01   ARB-1        PIC X(19).
PROCEDURE DIVISION.
     SET LAG-IND  PREIS-IND TO ANF-FELD.
     SEARCH LAG-POSTEN VARYING PREIS-IND
         AT END GO TO ENDE
       WHEN PREIS-CODE (LAG-IND) = PREIS-CODE-2 (PREIS-IND)
         AND HANDLAGER (LAG-IND) > 0
           NEXT SENTENCE.
     MOVE LAG-POSTEN (LAG-IND) TO ARB-1.
```

Die SET-Anweisung bringt erstens den Index LAG-IND in Grundstellung und ordnet ihn einem bestimmten Anfangsposten in der Tabelle LAG-TAB zu. Enthält ANF-FELD beispielsweise die Zahl 5, nimmt der Index LAG-IND Zugriff ab dem fünften Posten in der Tabelle LAG-TAB.

Die SET-Anweisung bringt zugleich den Index PREIS-IND in Grundstellung und ordnet ihn einem bestimmten Anfangsposten in der Tabelle PREIS-TAB zu. Enthält ANF-FELD beispielsweise die Zahl 5, nimmt der Index PREIS-IND Zugriff ab dem fünften Posten in der Tabelle PREIS-TAB.

Die SEARCH-Anweisung beginnt ihre Suchoperation ab dem durch LAG-IND indizierten Posten der Tabelle LAG-TAB. Ab hier wird jeder Posten auf das Vorhandensein der beiden in der WHEN-Angabe enthaltenen Vergleichs-Bedingungen hin untersucht. Die zweite Tabelle PREIS-TAB wird parallel mit abgefragt und beide Indizes sukzessive mitgeführt.

Die SEARCH-Anweisung kann auf folgende zwei Arten enden:
1. Das Tabellenende wird erreicht: GO TO ENDE.
2. Beide WHEN-Bedingungen sind erfüllt: Die Anweisung MOVE LAG-POSTEN (LAG-IND) TO ARB-1 wird ausgeführt.

DIE SEARCH ALL-ANWEISUNG

Die SEARCH ALL-Anweisung sucht eine Tabelle auf einen Tabellenposten hin ab, der eine gegebene Gleichheits-Bedingung erfüllt, und stellt die Nummer des ggf. gefundenen Postens im Indexfeld zur Verfügung. SEARCH ALL verwendet dabei den binären Suchvorgang (Halbierungsmethode) und zeigt dabei eine hohe Geschwindigkeitseffizienz. Voraussetzung ist hierbei, daß die Tabelle sortiert ist.

Format:

```
SEARCH ALL Identifier-1  [AT END unbedingte Anweisung-1]

      WHEN { Datenname-1  [IS EQUAL TO]   { Identifier-2          }  }
           {              [IS =       ]   { Literal-1             }  }
           {                               { arithmet. Ausdruck-1 }  }
           { Bedingungsname-1                                        }

 [    AND { Datenname-2  [IS EQUAL TO]   { Identifier-3          }  }  ]
 [        {              [IS =       ]   { Literal-2             }  }  ]
 [        {                               { arithmet. Ausdruck-2 }  }  ]
 [        { Bedingungsname-2                                        }  ]

     {unbedingte Anweisung-2}
     {NEXT SENTENCE         }

     [END-SEARCH]
```

Identifier-1 muß den Namen des Tabellenpostens ausweisen. Die Datendefinition von Identifier-1 muß eine OCCURS-Klausel mit einer KEY IS-Angabe und mit einer INDEXED BY-Angabe enthalten. Auf Identifier-1 darf in der SEARCH ALL-Anweisung keine Klammer mit Subskript- oder Index-Angabe folgen.

Die KEY IS-Angabe in der OCCURS-Klausel kennzeichnet die Datenfelder, die die Schlüsselwerte enthalten, nach denen die Tabellenposten in entweder aufsteigender oder absteigender Folge geordnet sind. Das erste in der KEY IS-Angabe der OCCURS-Klausel genannte Datenfeld enthält den Hauptschlüssel. Jedes weitere in der KEY IS-Angabe genannte Datenfeld enthält den nächstwichtigen Schlüssel. Diese Schlüsselfelder werden in der WHEN-Angabe der SEARCH ALL-Anweisung verwendet. Wird ein anderes als das Hauptschlüssel-Feld in einer WHEN-Angabe angegeben, müssen auch alle (gemäß der KEY IS-

Angabe der OCCURS-Klausel) vorrangigen Schlüsselfelder in der WHEN-Angabe der SEARCH ALL-Anweisung aufgeführt werden.

Bedingungsname-1, Bedingungsname-2, usw. (falls in der WHEN-Angabe verwendet) müssen alle in einer Bedingungsnamen-Datenbeschreibung mit je einem einzelnen Literal definiert sein (vergleichen Sie die Beschreibung zur Stufen-Nr. 88 in diesem Handbuch). Der Name des Datenfeldes (Bedingungs-Variable), dem der Bedingungsname zugewiesen ist, muß in der KEY IS-Angabe der OCCURS-Klausel des Identifiers-1 vorkommen.

Datenname-1, Datenname-2, usw. (falls in der WHEN-Angabe der SEARCH ALL-Anweisung verwendet) müssen in der KEY IS-Angabe der OCCURS-Klausel zum Identifier-1 vorkommen. Jeder Datenname-1, Datenname-2, usw. muß mit dem ersten Index von Identifier-1 indiziert werden, einhergehend mit weiteren Indizes oder Literalen, falls erforderlich. Jeder Datenname-1, Datenname-2, usw. kann qualifiziert werden.

Identifier-2, Identifier-3 oder Identifier, die in einem arithmetischen Ausdruck-1 bzw. arithmetischen Ausdruck-2 angegeben sind, dürfen keine Identifikationen von Schlüsselfeldern sein, die in der KEY IS-Angabe der OCCURS-Klausel in der Datenbeschreibung von Identifier-1 aufgeführt sind. Sie dürfen auch nicht von dem ersten (Identifier-1 zugeschriebenen) Index indexiert werden.

Vor Durchführung der SEARCH ALL-Anweisung ist es nicht erforderlich, den Index mittels der SET-Anweisung in Grundstellung zu bringen, da nach der Halbierungsmethode vorgegangen wird.

Bei mehreren in der Tabellendefinition angegebenen Indizes wird durch die Anweisung SEARCH ALL nur der erste manipuliert. Alle anderen bleiben unberührt.

SEARCH ALL prüft die Gleichheitsbedingungen in der WHEN-Angabe. Sind alle Gleichheitsbedingungen der WHEN-Angabe von einem Tabellen-Posten erfüllt, endet die Suchoperation und die unbedingte Anweisung-2 wird ausgeführt. Der Index enthält nunmehr die Nummer des Postens in der Tabelle, der alle Gleichheitsbedingungen erfüllt hat. Würde statt einer unbedingte Anweisung-2 der Passus NEXT SENTENCE programmiert, geht die Steuerung des Programmes auf den nächsten COBOL-Satz nach dem Punkt über.

Wird kein Posten gefunden, der die angegebenen Gleichheitsbedingungen erfüllt, geht die Steuerung auf die unbedingte Anweisung-1 der AT END-Klausel über. Wurde die AT END-Klausel nicht programmiert, geht die Steuerung auf den nächsten COBOL-Satz nach dem Punkt über. Insgesamt gilt, daß der Inhalt des Index bei erfolglosem Verlauf der Anweisung SEARCH ALL nicht vorhergesagt werden kann.

Der Eintrag "unbedingte Anweisung" im Format kann eine Reihe aufeinanderfolgender unbedingter Anweisungen sein, die durch einen Punkt oder eine weitere WHEN-Angabe abgeschlossen werden müssen. Wird eine Reihe aufeinanderfolgender unbedingter Anweisungen nicht mit einer Anweisung GO TO abgeschlossen, geht die Steuerung des Programmes auf den nächsten COBOL-Satz nach dem Punkt über.

Die Ergebnisse der SEARCH ALL-Anweisung sind nur unter folgenden Bedingungen voraussagbar:

1. Die Posten in der Tabelle müssen in der Art geordnet sein, wie in der KEY IS-Angabe der OCCURS-Klausel in der Datenbeschreibung von Identifier-1 angegeben.

2. Die Inhalte der Schlüsselfelder, auf die in der WHEN-Angabe der SEARCH ALL-Anweisung Bezug genommen wird, müssen zur Identifizierung eines einzelnen Tabellenpostens ausreichend aussagefähig, d. h. eindeutig, sein.

Beispiele für SEARCH ALL finden Sie ab der nächsten Seite:

Beispiel 1:

```
DATA DIVISION.
01  LAG-TA8.
    02  LAG-POSTEN OCCURS 10 TIMES ASCENDING KEY IS LAGER-NR
            INDEXED BY LAG-IND.
        03  LAGER-NR PIC 9(5).
        03  PREIS-CODE PIC 999.
        03  QUAN-ON-HAND PIC 9999.
        03  BESTELL-MENGE PIC 9999.
        03  MINDESTBESTAND PIC 999.
01  BEWEG-NR  PIC 9(5).

PROCEDURE DIVISION.
    SEARCH ALL LAG-POSTEN AT END GO TO NICHT-GEF
        WHEN LAGER-NR (LAG-IND) IS EQUAL TO BEWEG-NR
            GO TO GEFUNDEN.

GEFUNDEN.
       ----------------------
NICHT-GEF.
       ----------------------
```

Im Beispiel 1 besteht eine Tabelle namens LAG-TAB aus zehn Posten mit dem Namen LAG-POSTEN. Die 10 Posten sind nach den Inhalten des Feldes LAGER-NR in aufsteigender Folge geordnet. Der zum Zugriff auf die 10 Posten benutzte Index trägt den Namen LAG-IND.

Die Anweisung SEARCH ALL führt einen Suchlauf nach der Halbierungsmethode zum Auffinden des Postens aus, dessen Feld LAGER-NR den gleichen Inhalt wie das Vergleichsfeld BEWEG-NR aufweist. SEARCH ALL kann hierbei zwei verschiedene Ergebnisse haben:

1. Die Gleichheitsbedingung in der WHEN-Angbe ist erfüllt; die Anweisung GO TO GEFUNDEN wird ausgeführt.

2. Kein Posten der Tabelle erfüllt die Gleichheitsbedingung in der WHEN-Angabe; die Anweisung GO TO NICHT-GEF wird ausgeführt.

Beispiel 2:

```
DATA DIVISION.
01  WHTAB.
    02  WHPOSTEN OCCURS 24 TIMES ASCENDING KEY IS
          PRODUKT-NR  WH-NR  INDEXED BY WM-INDEX.
        03  PRODUKT-NR  PIC 999.
        03  WH-NR       PIC 99.
        03  LAGERORT    PIC 999.
        03  BESTAND     PIC 9(5).

PROCEDURE DIVISION.
    SEARCH ALL WHPOSTEN AT END GO TO TABELLEN-ENDE
        WHEN PRODUKT-NR (WM-INDEX) IS EQUAL TO BEW-PNR
          AND WH-NR (WM-INDEX) = BEW-WH-NR
            NEXT SENTENCE.
    MOVE LAGERORT (WH-INDEX) TO LAGERORT-NEU
     ---------------------
     ---------------------
TABELLEN-ENDE.
     ---------------------
```

In Beispiel 2 besteht eine Tabelle namens WHTAB aus 24 Posten mit dem Namen WHPOSTEN. Die 24 Posten sind nach den Inhalten der Hauptschlüssel-Felder PRODUKT-NR und der Nebenschlüssel-Felder WH-NR in aufsteigender Folge geordnet. Der zum Zugriff auf die 24 Posten benutzte Index trägt den Namen WM-Index.

Die Anweisung SEARCH ALL führt einen Suchlauf nach der Halbierungsmethode zum Auffinden des Postens aus, dessen Feld PRODUKT-NR den gleichen Inhalt wie das Vergleichsfeld BEW-PNR und dessen Feld WH-NR den gleichen Inhalt wie das Vergleichsfeld BEW-WH-NR aufweisen. SEARCH ALL kann hierbei zwei verschiedene Ergebnisse haben:

1. Die Gleichheitsbedingungen in der WHEN-Angabe sind beide erfüllt; der nächste COBOL-Satz MOVE LAGERORT (WH-INDEX) TO LAGERORT-NEU wird ausgeführt.

2. Kein Posten der Tabelle erfüllt die beiden Gleichheitsbedingungen in der WHEN-Angabe: die Anweisung GO TO TABELLEN-ENDE wird ausgeführt.

Beispiel 3:

```
DATA DIVISION.
01  WHTAB.
    02  WHPOSTEN  OCCURS 24 TIMES  ASCENDING KEY IS
          PRODUKT-NR  WH-NR  INDEXED BY WH-INDEX.
        03  PRODUKT-NR       PIC 999.
        03  WH-NR            PIC 99.
            88  WH-OST  VALUE IS 41.
            88  WH-WEST VALUE IS 67.
        03  LAGERORT         PIC 999.
        03  HANDLAGER        PIC 9(5).
PROCEDURE DIVISION.
    SEARCH ALL WHPOSTEN
        WHEN PRODUKT-NR (WH-INDEX) = BEW-PNR
          AND WH-OST (WH-INDEX)  GO TO OST-ROUTINE.
    SEARCH ALL WHPOSTEN  AT END GO TO KEIN-PRODUKT
        WHEN PRODUKT-NR (WH-INDEX) = BEW-PNR
          AND WH-WEST (WH-INDEX)  NEXT SENTENCE.
    ADD HANDLAGER (WH-INDEX) TO WESTSUMME.
      ------------------------------
OST-ROUTINE.
    ADD HANDLAGER (WH-INDEX) TO OSTSUMME.
      ------------------------------
KEIN-PRODUKT.
```

Im Beispiel 3 besteht eine Tabelle namens WHTAB aus 24 Posten mit dem Namen WHPOSTEN. Die 24 Posten sind nach den Inhalten der Hauptschlüssel-Felder PRODUKT-NR und der Nebenschlüssel-Felder WH-NR in aufsteigender Folge geordnet. Der zum Zugriff auf die 24 Posten benutzte Index trägt den Namen WH-Index.

Die erste Anweisung SEARCH ALL führt einen Suchlauf nach der Halbierungsmethode zum Auffinden des Postens aus, dessen Feld PRODUKT-NR den gleichen Inhalt wie das Vergleichsfeld BEW-PNR und dessen Feld WH-NR den Inhalt 41 aufweisen. Der Bedingungsname WH-OST wird dem Wert 41 im Datenfeld WH-NR zugeschrieben. Dieser SEARCH ALL kann zwei Ergebnisse haben:

1. Die Gleichheitsbedingungen in der WHEN-Angabe sind beide erfüllt: die Anweisung GO TO OST-ROUTINE wird ausgeführt.

2. Kein Posten der Tabelle erfüllt die beiden Gleichheitsbedingungen in der WHEN-Angabe: der nächste COBOL-Satz SEARCH ALL (zweiter SEARCH) wird ausgeführt.

Die zweite Anweisung SEARCH ALL führt einen Suchlauf nach

der Halbierungsmethode zum Auffinden des Postens aus, dessen Feld PRODUKT-NR den gleichen Inhalt wie das Vergleichsfeld BEW-PNR und dessen Feld WH-NR den Inhalt 67 aufweisen. Der Bedingungsname WH-WEST wird dem Wert 67 im Datenfeld WH-NR zugeschrieben. Auch dieser SEARCH ALL kann zwei Ergebnisse haben:

1. Die Gleichheitsbedingungen in der WHEN-Angabe sind beide erfüllt: der nächste COBOL-Satz ADD HANDLAGER (WH-INDEX) TO WESTSUMME wird ausgeführt.

2. Kein Posten der Tabelle erfüllt die beiden Gleichheitsbedingungen in der WHEN-Angabe: die Anweisung GO TO KEIN-PRODUKT wird ausgeführt.

DIE SEND-ANWEISUNG

Diese Anweisung dient zur Übergabe von Messages (Informationen), Message-Segmenten oder eines Teils einer Message oder eines Segments an eine oder mehrere Output-Queues des Message Control Systems (MCS).

Format 1:

```
SEND  CD-Name  FROM  Identifier-1
```

Format 2:

```
SEND  CD-Name  FROM  Identifier-1

      { WITH Identifier-2 }
      { WITH ESI          }
      { WITH EMI          }
      { WITH EGI          }
```

Zu beiden Formaten:

Unter CD-Name ist der Name einer Output Communication Description (CD) anzugeben.

Unter Identifier-2 (falls verwendet) ist der Name eines einstelligen numerischen Feldes ohne Vorzeichen anzugeben.

Ist die empfangende Kommunikationseinrichtung (Drucker, Bildschirm, usw.) auf eine Seite Zeilenlänge orientiert, gelten folgende Regeln:

- Jede Message oder jedes Message-Segment beginnt auf der linken äußeren Position der physischen Zeile.
- Kürzere Messages bzw. Message-Segmente in einer physischen Zeile werden rechts mit Spaces aufgefüllt.
- Überhänge längerer Messages oder Message-Segmente werden nicht abgeschnitten. Die physische Zeile wird gefüllt und ausgegeben. Daraufhin werden die Zeichen, aus denen der Überhang besteht, in die nachfolgende Zeile aufgenommen.

Ist die empfangende Kommunikationseinrichtung auf variabel lange Messages orientiert, beginnt jede Message oder jedes Message-Segment auf der nächsten freien Zeichenposition der Kommunikationseinrichtung.

Während der Ausführung einer SEND-Anweisung interpretiert MCS den Inhalt des Daten-Namens-2 (TEXT LENGTH) innerhalb des CD-Namen-Bereichs als Benutzer-Vorgabe in bezug auf die Anzahl linker äußerer Zeichen im Identifier-1, die an die Kommunikationseinrichtung zu senden sind. Ist der Inhalt von TEXT LENGTH gleich Null, werden keine Daten aus Identifier-1 übertragen. Ist der Inhalt von TEXT LENGTH größer als die Länge von Identifier-1, wird im Daten-Namen-3 (STATUS KEY) innerhalb des CD-Namen-Bereichs ein Status gesetzt und es werden keine Daten übertragen.

Eine einzelne Ausführung einer SEND-Anweisung des Formats 1 übergibt lediglich eine einzelne Portion einer Message oder eines Message-Segments an MCS. Eine einzelne Ausführung einer SEND-Anweisung des Formats 2 übermittelt an MCS nie mehr als eine einzelne Message oder Message-Segment, indiziert durch Identifier-2 oder die Angaben ESI, EMI oder EGI. MCS seinerseits reicht keine Teile einer Message an eine Kommunikationseinrichtung weiter, bevor nicht die Message in der Output-Queue komplett ist.

Das Schicksal einer nicht durch einen EMI oder EGI abgeschlossenen Message (= Message-Fragment) ist nicht vorhersagbar. Sie existiert für MCS logisch nicht und erreicht somit auch nicht den Empfänger (Kommunikationseinrichtung).

Im Falle einer portionsweisen Übertragung einer Message ist für jede Portion eine SEND-Anweisung erforderlich.

Zu Format 2:

Identifier-2 besagt, daß dem Identifier-1 ein "End-of-Segment-Indikator", ein "End-of-Message-Indikator" oder ein "End-of-Group-Indikator" mitgegeben werden soll (siehe hierzu nachstehende Übersicht). Jedes andere Zeichen als 1, 2 oder 3 wird als 0 interpretiert. Enthält Identifier-2 einen anderen Inhalt als 1, 2 oder 3 und es ist kein Identifier-1 angegeben, wird im Daten-Name-2 (STATUS KEY) des CD-Bereichs ein Fehlerstatus gesetzt und keine Daten übertragen.

Inhalt von Identifier-2	Für Identifier-1 angenommener Indikator	Bedeutung
0	keiner	kein Indikator
1	ESI	End of Segment
2	EMI	End of Message
3	EGI	End of Group

ESI dient der Mitteilung an MCS, daß das Message-Segment komplett ist. EMI, daß die Message komplett ist. EGI, daß die Gruppe von Messages komplett ist.

Die Hierarchie der Abschluß-Indikatoren ist: EGI, EMI, ESI. Einem EGI muß kein ESI oder EMI vorausgehen, einem EMI muß kein ESI vorausgehen.

DIE SET-ANWEISUNG (Variante Index-Name)

Diese Variante der SET-Anweisung ändert den Inhalt eines Indexes, um ihn einem bestimmten Posten in einer Tabelle zuzuordnen.

Format 1:

```
SET Index-Name-1  [Index-Name-2]  ...  TO  { Index-Name-3      }
                                           { Index-Datenfeld  }
                                           { Identifier-1     }
                                           { Zahl-1           }
```

Format 2:

```
SET Index-Name-1  [Index-Name-2]  ...  { UP BY   }  { Identifier-2 }
                                       { DOWN BY }  { Zahl-2       }
```

Ein Index-Name ist der Name für einen Index, der einer Tabelle zugeordnet ist. Die Angabe INDEXED BY in der Datenfeldbeschreibung des Tabellenpostens teilt den Index der Tabelle zu.

Ein Index-Datenfeld ist ein mit der Klausel USAGE IS INDEX beschriebenes Elementarfeld. Ein Index-Datenfeld ist ein Speicherplatz, in dem der Inhalt eines Indexes ohne Umwandlung vorübergehend gespeichert werden kann.

Identifier-1 und Identifier-2 müssen jeweils ein als eine Ganzzahl beschriebenes Elementarfeld bezeichnen. Zahl-1 und Zahl-2 sind numerische Literale, die eine ganze Zahl sein müssen und nicht Null sein dürfen. Zahl-1 kann ein positives Vorzeichen aufweisen. Zahl-2 kann entweder ein positives oder ein negatives Vorzeichen aufweisen.

Eine Tabellenpostennummer ist eine ganze Zahl, die anzeigt, wo ein Posten in einer Tabelle auftritt. Beispielsweise hat der dritte Posten in einer Tabelle die Tabellenpostennummer 3; der elfte Posten in einer Tabelle hat die Tabellenpostennummer 11.

Im Format 1 wird der Inhalt des Index-Namens-1 auf einen Wert gesetzt, der der Tabellenpostennummer entspricht, die entweder durch Index-Name-3, Index-Datenfeld, Identifier-1 oder Zahl-1 dargestellt wird. Die Verarbeitung wird, falls definiert, für Index-Name-2, etc., wiederholt. Bei jeder Wiederholung wird der Wert von Index-Name-3, Index-Datenfeld oder Identifier-1 so verwendet, wie er schon zu Beginn der Ausführung der Anweisung war. Alle vier Varianten von Format 1 der SET-Anweisung werden auf den nächsten Seiten erläutert.

Im Format 2 wird der Inhalt von Index-Name-1 durch Identifier-2 oder Zahl-2 erhöht oder vermindert. Die Verarbeitung wird, falls definiert, für Index-Name-2, etc., wiederholt. Bei jeder Wiederholung wird der Wert von Identifier-2 so verwendet, wie er schon zu Beginn der Ausführung der Anweisung war. Diese zwei Varianten von Format 2 der SET-Anweisung werden nachfolgend erläutert.

Index in Index kopieren:

Die Anweisung SET Index-Name-1 TO Index-Name-3 bewirkt, daß der Inhalt des Index mit Index-Name-3 in den Index mit Index-Name-1 kopiert wird.

```
FILE SECTION.
02  MENGE    PIC 99  OCCURS 5 TIMES INDEXED BY Q1.

02  LIMIT1   PIC 99  OCCURS 5 TIMES INDEXED BY Q2.

PROCEDURE DIVISION.
    SET Q2 TO Q1.
    IF MENGE (Q1) IS LESS THAN LIMIT1 (Q2) GO TO NAECHSTE.
```

Index-Datenfeld in Index kopieren:

```
FILE SECTION.
02  SUMME    PIC 99V9 OCCURS 10 TIMES INDEXED BY INDEX-A.

WORKING-STORAGE SECTION.
01  INDEX-A-FELD USAGE IS INDEX.

PROCEDURE DIVISION.
    SET INDEX-A TO INDEX-A-FELD.
```

Identifier in Index kopieren:

```
WORKING-STORAGE SECTION.
01  START-TAG    PIC 99.

01  VERKAUF      PIC 9999V99 OCCURS 12 TIMES INDEXED BY TAG1.

PROCEDURE DIVISION.
    SET TAG1 TO START-TAG.
```

Zahl in Index kopieren:

```
FILE SECTION.
02  TOTAL-MENGE      PIC 9(5)V99 OCCURS 7 TIMES INDEXED BY TAG.

PROCEDURE DIVISION.
    SET TAG TO 5.
```

Index um Identifier erhöhen (UP) bzw. vermindern (DOWN):

```
FILE SECTION.
02  MENGE    PIC 9999 OCCURS 5 TIMES INDEXED BY DD.

WORKING-STORAGE SECTION.
01  WW       PIC 9.

PROCEDURE DIVISION.
    SET DD DOWN BY WW.
```

Index um Zahl erhöhen (UP) bzw. vermindern (DOWN):

```
FILE SECTION.
02  MENGE    PIC 999 OCCURS 5 TIMES INDEXED BY Q1.

PROCEDURE DIVISION.
    SET Q1 TO 1.
A1. ADD MENGE (Q1) TO TOTAL-MENGE.
    SET Q1 UP BY 1.
    IF Q1 IS LESS THAN 6 GO TO A1.
```

DIE SET-ANWEISUNG (Variante Index-Datenfeld)

Diese Variante der SET-Anweisung kopiert den Wert eines Indexes oder eines Index-Datenfeldes in ein Index-Datenfeld.

Format:

```
SET Index-Datenfeld-1  [Index-Datenfeld-2]  ...
                    TO { Index-Name        }
                       { Index-Datenfeld-3 }
```

Ein Index-Name ist der Name für einen Index, der einer Tabelle zugeordnet ist. Die Angabe INDEXED BY in der Datenfeldbeschreibung des Tabellenpostens teilt den Index der Tabelle zu.

Ein Index-Datenfeld ist ein mit der Klausel USAGE IS INDEX beschriebenes Elementarfeld. Ein Index-Datenfeld ist ein Speicherplatz, in dem der Inhalt eines Indexes ohne Umwandlung vorübergehend gespeichert werden kann.

Eine Tabellenpostennummer ist eine ganze Zahl, die anzeigt, wo ein Posten in einer Tabelle auftritt. Beispielsweise hat der dritte Posten in einer Tabelle die Tabellenpostennummer 3; der elfte Posten in einer Tabelle hat die Tabellenpostennummer 11.

Bei dieser Variante der SET-Anweisung wird der Inhalt des Index-Datenfeldes-1 auf einen Wert gesetzt, der gleich dem in einem Index oder einem Index-Datenfeld enthaltenen Wert ist. Die Verarbeitung wird, falls definiert, für Index-Datenfeld-2, etc., wiederholt. Bei jeder Wiederholung wird der Wert von Index-Name oder Index-Datenfeld-3 so kopiert, wie er schon zu Beginn der Ausführung der Anweisung war.

Index in Index-Datenfeld kopieren:

```
FILE SECTION.
02  SUMME    PIC 99V9 OCCURS 10 TIMES INDEXED BY INDEX-A.

WORKING-STORAGE SECTION.
01  INDEX-A FELD USAGE IS INDEX.

PROCEDURE DIVISION.
    SET INDEX-A-FELD TO INDEX-A.
```

Index-Datenfeld in Index-Datenfeld kopieren:

Im folgenden Beispiel wird der Inhalt von INDEX-A-FELD in INDEX-B-FELD kopiert.

```
WORKING-STORAGE SECTION.
01  INDEX-A-FELD USAGE IS INDEX.
01  INDEX-B-FELD USAGE IS INDEX.

PROCEDURE DIVISION.
    SET INDEX-B-FELD TO INDEX-A-FELD.
```

DIE SET-ANWEISUNG (Variante Identifier)

Diese Variante der SET-Anweisung kopiert den Inhalt eines Indexes in ein oder mehrere numerische Elementarfelder.

Format:

SET Identifier-1 [Identifer-2] ... TO Index-name

Ein Index-Name ist der Name für den Index, der einer Tabelle zugeordnet ist. Die Angabe INDEXED BY in der Datenfeldbeschreibung des Tabellenpostens teilt den Index der Tabelle zu.

Identifier-1 und Identifier-2 müssen jeweils ein als eine Ganzzahl beschriebenes Elementarfeld bezeichnen.

Eine Tabellenpostennummer ist eine ganze Zahl, die anzeigt, wo ein Posten in einer Tabelle auftritt. Beispielsweise hat der dritte Posten in einer Tabelle die Tabellenpostennummer 3; der elfte Posten in einer Tabelle hat die Tabellenpostennummer 11.

Index in einen Identifier kopieren:

```
FILE SECTION.
01  SUMME   PIC 99V9 OCCURS 10 TIMES INDEXED BY INDEX-A.

WORKING-STORAGE SECTION.
02  ZAHL-A  PIC 99.

PROCEDURE DIVISION.
    SET ZAHL-A TO INDEX-A.
```

DIE SET-ANWEISUNG (Variante Mnemonic-Name)

Diese Variante der SET-Anweisung dient zur Veränderung des Status eines Schalters (Switch).

Format:

```
SET  {{Mnemonic-Name}  ... TO {ON}}  ...
                              {OFF}
```

DIE SET-ANWEISUNG (Variante Condition-Name)

Diese Variante der SET-Anweisung überträgt den VALUE eines mit Stufen-Nr. 88 definierten Condition-Names in die Bedingungs-Variable. Hat dabei ein Condition-Name mehrere Values, wird der erste verwendet.

Format:

```
SET  {Condition-Name}  ...  TO TRUE
```

DIE SHIFT-ANWEISUNG

Die SHIFT-Anweisung verschiebt den Inhalt eines booleschen oder binären Datenfeldes nach links oder rechts um eine bestimmte Anzahl von Bit-Positionen.

Format:

```
SHIFT Identifier-1 {RIGHT} {Identifier-2} [POSITION ]
                   {LEFT } {Zahl-1      } [POSITIONS]
```

Identifier-1 muß entweder ein boolesches oder ein binäres Datenfeld sein. Identifier-1 bezeichnet das Datenfeld, dessen Bits durch die SHIFT-Anweisung verschoben werden sollen.

Der Inhalt von Identifier-2 bzw. Zahl-1 bezeichnet die Anzahl von Bit-Positionen, um die durch die SHIFT-Anweisung verschoben werden soll. Identifier-2 muß ein positives numerisches Elementarfeld sein. Zahl-1 stellt ein numerisches Literal dar, das eine Ganzzahl und ohne Vorzeichen sein muß.

Bei Vorhandensein des Schlüsselwortes RIGHT werden die Bits nach rechts verschoben. Links werden Null-Bits eingesetzt. Bits, die über die äußerste rechte Bit-Position des Datenfeldes hinausgeschoben werden, gehen verloren.

Bei Vorhandensein des Schlüsselwortes LEFT werden die Bits nach links verschoben. Rechts werden Null-Bits eingesetzt. Bits, die über die äußerste linke Bit-Position des Datenfeldes hinausgeschoben werden, gehen verloren.

Beispiel 1:

```
SHIFT A1-CODE RIGHT 2 POSITIONS.

A1-CODE vorher  11111111
A1-CODE nachher 00111111
```

Beispiel 2:

```
   SHIFT A2-CODE LEFT 1 POSITION.

   A2-CODE vorher  11111111
   A2-CODE nachher 11111110
```

Beispiel 3:

```
   SHIFT FELD-B RIGHT 9 POSITIONS.

   FELD-B vorher  0111111101110111
   FELD-B nachher 0000000000111111
```

Beispiel 4:

```
   SHIFT FELD-C RIGHT BIT-NUMMER POSITIONS.

   BIT-NUMMER vorher  06
   BIT-NUMMER nachher 06
   FELD-C vorher      1111111110111010
   FELD-C nachher     0000001111111110
```

DIE SORT-ANWEISUNG

Eine Beschreibung der COBOL-Anweisung SORT zum Sortieren von Dateien (mit Beispielen) finden Sie in einem eigenen Kapitel im hinteren Teil dieses Buches.

DIE START-ANWEISUNG (Relative Datei)

Die START-Anweisung schafft in einer relativen Datei die Möglichkeit des logischen Positionierens in Vorbereitung für nachfolgendes sequentielles oder dynamisches Lesen von Datensätzen.

Format:

```
START     Datei-Name
                  ┌IS EQUAL TO                       ┐
                  │IS =                              │
            ┌     │IS GREATER THAN                   │              ┐
     KEY    │    <│IS >                              │>  Identifier │
            │     │IS NOT LESS THAN                  │              ┘
            └     │IS NOT <                          │
                  │IS GREATER THAN OR EQUAL TO       │
                  └IS >=                             ┘
     [INVALID KEY unbedingte Anweisung 1]
     [NOT INVALID KEY unbedingte Anweisung 2]
     [END-START]
```

Der in der START-Anweisung gegebene Datei-Name muß einem der im FILE-CONTROL-Eintrag auftretenden Datei-Namen entsprechen. Der FILE-CONTROL-Eintrag für den Datei-Namen muß die Klausel ORGANIZATION IS RELATIVE und indirekt oder ausdrücklich die Klausel ACCESS MODE IS SEQUENTIAL bzw. die Klausel ACCESS MODE IS DYNAMIC aufweisen.

Vor Ausführung der START-Anweisung muß die dem Datei-Namen zugeordnete relative Datei entweder durch eine OPEN INPUT- oder durch eine OPEN I-O-Anweisung eröffnet worden sein.

Die START-Anweisung bewirkt, daß die Datei auf denjenigen Datensatz, wenn vorhanden, positioniert wird, dessen relative Satznummer gleich dem Inhalt des Satzschlüsselfeldes ist (bei Verwendung von KEY =) oder z. B. auf die erste belegte Satzposition (Record Slot) ab der, auf die das Satzschlüsselfeld zeigt (bei Verwendung von KEY NOT <). Das relative Satzschlüsselfeld muß durch die RELATIVE KEY-Angabe in der ACCESS MODE-Klausel des FILE-CONTROL-Eintrags für den

Datei-Namen bezeichnet werden. Der Programmierer muß vor Ausführung der START-Anweisung eine relative Satznummer in das Satzschlüsselfeld bringen.

Nachdem eine START-Anweisung für eine relative Datei mit sequentiellem Zugriff korrekt ausgeführt wurde, bewirkt die anschließende wiederholte Ausführung einer READ-Anweisung das sequentielle Lesen aller weiteren Sätze ab dem mit START gesetzten Eintrittspunkt.

Nachdem eine START-Anweisung für eine relative Datei mit dynamischem Zugriff korrekt ausgeführt wurde, bewirkt die anschließende wiederholte Ausführung einer READ NEXT-Anweisung das sequentielle Lesen aller weiteren Sätze ab dem mit START gesetzten Eintrittspunkt.

KEY-Angabe

Wird die KEY-Angabe verwendet, dann ist die Vergleichsart ausdrücklich definiert und umgeht so die bei nicht vorhandener KEY-Angabe als Default angenommene Gleichheitskontrolle. Die KEY-Angabe spezifiziert die durchzuführende Art des Vergleichs. Der Daten-Name (Identifier) in der KEY-Angabe muß der Daten-Name des in der RELATIVE KEY-Angabe des FILE-CONTROL-Eintrags für den Datei-Namen spezifizierten numerischen Satzschlüsselfeldes sein. Es kann qualifiziert werden.

Wird die GREATER THAN-Bedingung spezifiziert, wird die Datei auf den ersten aktiven (!) Datensatz positioniert, dessen relative Satznummer größer als der Inhalt des Satzschlüsselfeldes ist. Wird die NOT LESS THAN-Bedingung spezifiziert, wird die Datei auf den ersten aktiven Datensatz positioniert, dessen relative Satznummer nicht kleiner als der Inhalt des Satzschlüsselfeldes ist; d. h. die Datei wird auf einen Datensatz positioniert, dessen relative Satznummer entweder gleich oder größer als das relative Satzschlüsselfeld ist.

INVALID KEY-Klausel

Die INVALID KEY-Klausel muß für eine START-Anweisung spezifiziert werden, die auf eine relative Datei mit sequentiellem oder dynamischem Zugriff Bezug nimmt und für die keine

USE-Anweisung in den DECLARATIVES existiert.

Befinden sich in der gesuchten Satzposition keine Daten oder wurden sie daraus gelöscht, dann tritt eine INVALID KEY-Bedingung auf.

Wenn die INVALID KEY-Bedingung auftritt, werden die folgenden Aktionen in der festgelegten Reihenfolge durchgeführt:

1. Ist die FILE STATUS-Klausel im FILE-CONTROL-Eintrag für die relative Datei spezifiziert, wird der Wert 23 in das FILE STATUS-Datenfeld übertragen, um anzuzeigen, daß kein Datensatz in der spezifizierten Datensatzposition gefunden wurde.

2. Wurde die INVALID KEY-Klausel in der START-Anweisung verwendet, wird die Steuerung auf die unbedingte Anweisung übertragen. Keine der für diese Datei spezifizierten DECLARATIVE-Prozeduren wird ausgeführt.

3. Ist die INVALID KEY-Klausel nicht in der START-Anweisung verwendet, werden die DECLARATIVE-Prozeduren ausgeführt.

Beispiel 1:

```
FILE-CONTROL.
    SELECT DATEI-W  ASSIGN TO DISC
            ORGANIZATION IS RELATIVE
            ACCESS MODE IS SEQUENTIAL          (!)
            RELATIVE KEY IS SCHL-W.

DATA DIVISION.
FILE SECTION.
FD  DATEI-W BLOCK CONTAINS 25 RECORDS
            RECORD CONTAINS 20 CHARACTERS
            LABEL RECORD IS STANDARD.
01  SATZ-W  PIC X(20).

WORKING-STORAGE SECTION.
01  SCHL-W  PIC 9999.

PROCEDURE DIVISION.
ANFANG.
    OPEN INPUT DATEI-W.
       -------------
    MOVE 75 TO SCHL-W.
    START DATEI-W KEY IS EQUAL TO SCHL-W INVALID KEY GO TO
            KEIN-SATZ.
R2. READ DATEI-W AT END GO TO ENDE.
       -------------
    GO TO R2.
```

Im obigen Beispiel handelt es sich um eine relative Datei mit sequentiellem Zugriff. Die Datei hat den Namen DATEI-W. Damit Daten aus DATEI-W gelesen werden können, muß die Datei mit der OPEN INPUT-Anweisung eröffnet werden.

DIE MOVE-Anweisung überträgt 75 in das Satzschlüsselfeld SCHL-W, um die relative Satznummer des Datensatzes, auf den DATEI-W durch die START-Anweisung zu positionieren ist, zu spezifizieren. Die START-Anweisung veranlaßt, daß DATEI-W auf den Datensatz positioniert wird, dessen relative Satznummer gleich dem Inhalt von SCHL-W ist. Da SCHL-W 75 enthält, wird DATEI-W auf die 75. Satzposition (Record Slot) positioniert. Ist in der 75sten Satzposition von DATEI-W kein gültiger Datensatz enthalten, führt die START-Anweisung aufgrund INVALID KEY die Anweisung GO TO KEIN-SATZ aus.

Nach korrekter Ausführung der START-Anweisung macht die READ-Anweisung den 75sten Datensatz der DATEI-W zur Verarbeitung verfügbar. Nach korrekter Ausführung der READ-Anweisung befindet sich der 75ste Datensatz aus DATEI-W im Datenbereich SATZ-W.

Nachfolgende Ausführungen der READ-Anweisung nehmen Zugriff auf den jeweils sequentiell nächsten Datensatz aus DATEI-W, wobei nicht belegte Satzpositionen übersprungen werden.

Wenn die READ-Anweisung feststellt, daß in DATEI-W keine weiteren Sätze zur Verfügung stehen, wird AT END GO TO ENDE ausgeführt.

Beispiel 2:

```
FILE-CONTROL.
    SELECT DATEI-S ASSIGN TO DISC
           ORGANIZATION IS RELATIVE
           ACCESS MODE IS DYNAMIC                (!)
           RELATIVE KEY IS SCHL-S.

DATA DIVISION.
FILE SECTION.
FD  DATEI-S BLOCK CONTAINS 25 RECORDS
           RECORD CONTAINS 20 CHARACTERS
           LABEL RECORD IS STANDARD.
01  SATZ-S  PIC X(20).

WORKING-STORAGE SECTION.
01  SCHL-S  PIC 9999.

PROCEDURE DIVISION.
ANFANG.
    OPEN INPUT DATEI-S.
       -------------
    MOVE 22 TO SCHL-S.
    START DATEI-S INVALID KEY GO TO KEIN-SATZ.
S1. READ DATEI-S NEXT RECORD AT END GO TO ENDE.
       -------------
    GO TO S1.
```

Im obigen Beispiel handelt es sich um eine relative Datei mit dynamischem Zugriff; d. h., innerhalb desselben Programmes erfolgt sowohl direkter als auch sequentieller Zugriff auf die Datei. Die Datei hat den Namen DATEI-S. Damit Daten aus DATEI-S gelesen werden können, muß die Datei zuerst im INPUT-Modus eröffnet werden.

Der Programmierer bringt die Zahl 22 in das Satzschlüsselfeld SCHL-S, um die relative Satznummer des Datensatzes, auf den die START-Anweisung die DATEI-S positionieren soll, zu spezifizieren. Die START-Anweisung veranlaßt, daß DATEI-S auf den Datensatz positioniert wird, dessen relative Satznummer gleich dem Inhalt von SCHL-S ist. Da SCHL-S 22 enthält, wird DATEI-S auf den 22sten Datensatz positioniert. Ist in der 22sten Satz-Position in DATEI-S kein gültiger Datensatz enthalten, führt die START-Anweisung in ihrem

INVALID KEY-Zweig die Anweisung GO TO KEIN-SATZ aus.

Nach der korrekten Ausführung der START-Anweisung macht die READ NEXT-Anweisung den 22sten Datensatz aus DATEI-S zur Verarbeitung verfügbar. Nach der korrekten Ausführung der READ NEXT-Anweisung befindet sich der 22ste Datensatz aus DATEI-S im Datenbereich SATZ-S.

Nachfolgende Ausführungen der READ NEXT-Anweisung nehmen Zugriff auf den jeweils sequentiell nächsten Datensatz aus DATEI-S, wobei leere Satzpositionen übersprungen werden.

Wenn die READ NEXT-Anweisung feststellt, daß in DATEI-S keine weiteren Sätze mehr sind, wird AT END GO TO ENDE ausgeführt.

Beispiel 3:

```
FILE-CONTROL.
    SELECT DATEI-T ASSIGN TO DISC
           ORGANIZATION IS RELATIVE
           ACCESS MODE IS DYNAMIC                (!)
           RELATIVE KEY IS SCHL-T.

DATA DIVISION.
FILE SECTION.
FD  DATEI-T BLOCK CONTAINS 25 RECORDS
            RECORD CONTAINS 20 CHARACTERS
            LABEL RECORD IS STANDARD.
01  SATZ-T  PIC X(20).

WORKING-STORAGE SECTION.
01  SCHL-T  PIC 99.

PROCEDURE DIVISION.
ANFANG.
    OPEN I-O DATEI-T.
       -----------------
    MOVE 31 TO SCHL-T.
    START DATEI-T KEY IS NOT LESS THAN SCHL-T
          INVALID KEY GO TO KEIN-SATZ-2.
T1. READ DATEI-T NEXT RECORD AT END GO TO ENDE.
       -----------------
    REWRITE SATZ-T INVALID KEY GO TO KEIN-SATZ-3.
    GO TO T1.
```

Im obigen Beispiel handelt es sich um eine relative Datei mit dynamischem Zugriff; d. h., innerhalb desselben Programmes erfolgt sowohl direkter als auch sequentieller Zugriff auf die Datei. Die Datei hat den Namen DATEI-T. Da Datensätze sowohl aus DATEI-T zu lesen als auch in diese zurückzuschreiben sind, muß die Datei zuerst im I-O-Modus eröffnet werden.

Die START-Anweisung positioniert DATEI-T auf den ersten Datensatz, dessen relative Satznummer nicht kleiner als der Inhalt von SCHL-T ist. Da SCHL-T 31 enthält, wird DATEI-T auf den 31sten Datensatz positioniert. Ist in der 31sten Satz-Position der DATEI-T kein, in der 32sten Satz-Position jedoch ein Datensatz enthalten, wird DATEI-T auf den 32sten Datensatz positioniert.

Nachdem durch die START-Anweisung die DATEI-T erfolgreich positioniert wurde, macht die READ NEXT-Anweisung den markierten Datensatz der DATEI-T zur Verarbeitung verfügbar. Nach korrekter Ausführung der READ NEXT-Anweisung befindet sich dieser Datensatz aus DATEI-T im (SATZ-T genannten) Satz-Bereich.

Nachfolgende Ausführungen der READ NEXT-Anweisung für DATEI-T nehmen Zugriff auf den jeweils sequentiell nächsten Datensatz der Datei-T.

Wenn die READ NEXT-Anweisung feststellt, daß in DATEI-T keine weiteren Sätze mehr zur Verfügung stehen, wird AT END GO TO ENDE ausgeführt.

Die REWRITE-Anweisung schreibt den Inhalt von SATZ-T in die Datei zurück. Durch die vorangehende READ NEXT-Anweisung wurde in SCHL-T die relative Satznummer des gelesenen Datensatzes festgehalten. Somit wird die REWRITE-Anweisung den Inhalt von SATZ-T auf die in SCHL-T vorgemerkte Datensatzposition in DATEI-T zurückschreiben.

DIE START-ANWEISUNG (Indexierte Datei)

Die START-Anweisung schafft in einer indexierten Datei die Möglichkeit des logischen Positionierens in Vorbereitung für nachfolgendes sequentielles oder dynamisches Lesen von Datensätzen.

Format:

```
START     Datei-Name
                      ┌                                       ┐
                      │IS EQUAL TO                            │
       ┌              │IS =                                   │                ┐
       │              │IS GREATER THAN                        │                │
       │ KEY         {│IS >                                   │}  Identifier   │
       │              │IS NOT LESS THAN                       │                │
       └              │IS NOT <                               │                ┘
                      │IS GREATER THAN OR EQUAL TO            │
                      │IS >=                                  │
                      └                                       ┘
[INVALID KEY  unbedingte Anweisung 1]
[NOT INVALID KEY  unbedingte Anweisung 2]
[END-START]
```

Der in der START-Anweisung gegebene Datei-Name muß einem im FILE-CONTROL-Eintrag auftretenden Datei-Namen entsprechen. Der FILE-CONTROL-Eintrag für den Datei-Namen muß die Klausel ORGANIZATION IS INDEXED und indirekt oder ausdrücklich die Klausel ACCESS MODE IS SEQUENTIAL oder die Klausel ACCESS MODE IS DYNAMIC aufweisen.

Die indexierte Datei kann entweder Sätze fester oder variabler Länge enthalten. Besteht die indexierte Datei aus Sätzen variabler Länge, dürfen die RECORD CONTAINS Klausel und die Satzbeschreibungen nicht die von dem Variable-Längen-Indikator (VLI) eingenommenen Bytes beinhalten.

Vor Ausführung der START-Anweisung muß die dem Datei-Namen zugeordnete indexierte Datei durch eine OPEN INPUT- oder OPEN I-O-Anweisung eröffnet werden.

Die START-Anweisung bewirkt, daß die Datei auf denjenigen Datensatz positioniert wird, dessen Satzschlüssel gleich dem Inhalt des Satzschlüsselfeldes ist (Identifier RECORD KEY

oder ALTERNATE RECORD KEY). Der Programmierer muß vor Ausführung der START-Anweisung den gewünschten Wert in das Satzschlüsselfeld bringen.

Bei Beendigung der korrekten Ausführung einer START-Anweisung für eine indexierte Datei wird ein Bezugsschlüssel erstellt. Unter Bezugsschlüssel versteht man den Schlüssel, der laufend verwendet wird, um den Zugriff auf Datensätze in einer indexierten Datei zu haben. Ist die KEY-Angabe nicht in der START-Anweisung enthalten, dann ist der in der RECORD KEY-Klausel spezifizierte primäre Satzschlüssel der Bezugsschlüssel. Bezeichnet die KEY-Angabe den Satzschlüssel, dann ist letzterer der Bezugsschlüssel.

Nachdem eine START-Anweisung für eine indexierte Datei mit sequentiellem Zugriff korrekt ausgeführt wurde, bewirkt die wiederholte Ausführung einer READ-Anweisung mit sequentiellem Zugriff das sequentielle Lesen von Datensätzen ab der mit START gekennzeichneten Position.

Nachdem eine START-Anweisung für eine indexierte Datei mit dynamischem Zugriff korrekt ausgeführt wurde, bewirkt die wiederholte Ausführung einer READ NEXT-Anweisung ebenfalls das sequentielle Lesen von Datensätzen ab der mit START gekennzeichneten Position.

Das sequentielle Lesen von Sätzen nach einer START-Anweisung richtet sich nach der Index-Folge des jeweils in START verwendeten Primär- oder Alternativ-Schlüssels.

KEY-Klausel

Wird die KEY-Klausel verwendet, dann ist die Vergleichsart ausdrücklich definiert und umgeht so die bei nicht vorhandener KEY-Angabe indirekt zugrunde gelegte Gleichheitskontrolle. Die KEY-Angabe spezifiziert die Art des Vergleichs.

Der Identifier in der KEY-Klausel muß entweder der Daten-Name des primären Satzschlüsselfeldes oder eines alternativen Satzschlüsselfeldes sein. Der Daten-Name in der KEY-Angabe kann qualifiziert werden.

Wird die GREATER THAN-Bedingung durch die KEY-Klausel spezifiziert, wird die Datei auf den ersten in der Datei vorhandenen Datensatz positioniert, dessen Satzschlüssel größer als der Inhalt des in der KEY-Klausel spezifizierten Daten-Namens ist. Wird die NOT LESS THAN-Bedingung durch die

KEY-Klausel spezifiziert, wird die Datei auf den ersten in der Datei vorhandenen Datensatz positioniert, dessen Satzschlüssel nicht kleiner als der Inhalt des in der KEY-Klausel spezifizierten Daten-Namens ist; d. h. die Datei wird auf einen Datensatz positioniert, dessen Satzschlüssel entweder gleich dem oder größer als das durch den Daten-Namen in der KEY-Klausel spezifizierte Schlüsselfeld ist.

INVALID KEY-Klausel

Die INVALID KEY-Klausel muß für eine START-Anweisung spezifiziert werden, die auf eine indexierte Datei mit sequentiellem oder dynamischem Zugriff Bezug nimmt und für die keine USE-Anweisung in den DECLARATIVES enthalten ist.

Weist die indexierte Datei, auf die zugegriffen wird, keinen Datensatz auf, der der Bedingung der START-Anweisung entspricht, tritt eine INVALID KEY-Bedingung auf und die Ausführung der START-Anweisung gilt als erfolglos.

Wenn INVALID KEY zutrifft, werden die folgenden Aktionen in der angegebenen Reihenfolge durchgeführt:

1. Ist die FILE STATUS-Klausel im FILE CONTROL-Eintrag für die indexierte Datei spezifiziert, wird der Wert 23 in das FILE STATUS-Datenfeld übertragen, um anzuzeigen, daß kein der Bedingung in der START-Anweisung entsprechender Datensatz gefunden wurde.

2. Ist die INVALID KEY-Klausel in der START-Anweisung programmiert, wird die Steuerung auf die unbedingte Anweisung übertragen. Keine der für diese Datei spezifizierten DECLARATIVE-Prozeduren wird ausgeführt.

3. Ist die INVALID KEY-Klausel in der START-Anweisung nicht programmiert, werden für diese Datei DECLARATIVE-Prozeduren ausgeführt.

Auf den folgenden Seiten finden Sie Beispiele:

Beispiel 1:

```
FILE-CONTROL.
    SELECT DATEI-X  ASSIGN TO DISC
           ORGANIZATION IS INDEXED
           ACCESS MODE IS SEQUENTIAL
           RECORD KEY IS SCHL-X.
DATA DIVISION.
FILE SECTION.
FD  DATEI-X BLOCK CONTAINS 25 RECORDS
           RECORD CONTAINS 20 CHARACTERS
           LABEL RECORD IS STANDARD.
01  SATZ-X.
    02  SCHL-X  PIC X(5).
    02  FILLER  PIC X(15).
PROCEDURE DIVISION.
ANFANG.
    OPEN INPUT DATEI-X.
       -------------
    MOVE "A1468" TO SCHL-X.
    START DATEI-X KEY IS EQUAL TO SCHL-X
           INVALID KEY GO TO KEIN-SATZ.
X2. READ DATEI-X AT END GO TO ENDE.
       -------------
    GO TO X2.
```

Im Beispiel 1 handelt es sich um eine indexierte Datei mit sequentiellem Zugriff. Damit Daten aus DATEI-X gelesen werden können, muß die Datei mit OPEN INPUT eröffnet werden.

Die MOVE-Anweisung überträgt den Wert A1468 in das primäre Satzschüsselfeld SCHL-X, um den Satzschlüssel des Datensatzes zu spezifizieren, auf den die START-Anweisung die Datei positionieren soll. Die START-Anweisung veranlaßt, daß DATEI-X auf den Datensatz positioniert wird, dessen Satzschlüssel dem Inhalt von SCHL-X entspricht. Da SCHL-X A1468 enthält, wird DATEI-X auf den den Satzschlüssel A1468 aufweisenden Datensatz positioniert. Wird in der Datei kein Datensatz mit einem Satzschlüssel, der gleich A1468 ist, gefunden, führt die START-Anweisung (INVALID KEY) die Anweisung GO TO KEIN-SATZ aus.

Nach korrektem START macht der nachfolgende READ den einen Satzschlüssel A1468 aufweisenden Datensatz in SATZ-X zur Verarbeitung verfügbar.

Weitere READ-Anweisungen greifen auf den jeweils sequentiell nächsten Datensatz aus DATEI-X zu, orientiert am jeweiligen Index des primären oder alternativen Satzschlüssels.

Beispiel 2:

```
FILE-CONTROL.
    SELECT DATEI-U  ASSIGN TO DISC
           ORGANIZATION IS INDEXED
           ACCESS MODE IS DYNAMIC
           RECORD KEY IS SCHL-U.

DATA DIVISION.
FILE SECTION.
FD  DATEI-U BLOCK CONTAINS 25 RECORDS
           RECORD CONTAINS 20 CHARACTERS
           LABEL RECORD IS STANDARD.
01  SATZ-U.
    02  SCHL-U.
        03  SCHL-U1      PIC XXX.
        03  SCHL-U2      PIC X.
    02  FILLER           PIC X(16).

PROCEDURE DIVISION.
ANFANG.
    OPEN INPUT DATEI-U.
       -------------
    MOVE "AB4" TO SCHL-U1.
    START DATEI-U KEY IS EQUAL TO SCHL-U1 INVALID KEY GO
                              KEIN SATZ.
RF. READ DATEI-U NEXT RECORD AT END GO TO ENDE.
       -------------
    GO TO RF.
       -------------
    MOVE "FH95" TO SCHL-U.
    READ DATEI-U INVALID KEY GO TO KEIN-U-SATZ.
```

In diesem Beispiel handelt es sich um eine indexierte Datei mit dynamischem Zugriff, d. h., innerhalb desselben Programmes erfolgt sowohl direkter als auch sequentieller Zugriff auf die Datei. Die Datei hat den Namen DATEI-U. Damit Daten aus DATEI-U gelesen werden können, muß die Datei zuerst mit OPEN INPUT eröffnet werden.

Der Programmierer überträgt den Wert AB4 in das Datenfeld SCHL-U1, um den Datensatz, auf den die START-Anweisung die DATEI-U positionieren soll, zu spezifizieren. SCHL-U1 ist ein die ersten drei Zeichen des Satzschlüssels SCHL-U einnehmendes Datenfeld. Die START-Anweisung veranlaßt, daß die DATEI-U auf den ersten Datensatz positioniert wird, der

in den ersten drei Bytes seines primären Satzschlüssels den Wert AB4 aufweist. Weist kein Datensatz der DATEI-U in den ersten drei Bytes seines primären Satzschlüssels AB4 auf, führt die START-Anweisung (INVALID KEY) die unbedingte Anweisung GO TO KEIN-SATZ aus.

Nach korrekter Ausführung der START-Anweisung macht die READ NEXT-Anweisung den ersten, "AB4" in den ersten drei Bytes seines primären Satzschlüssels aufweisenden Datensatz in DATEI-U zur Verarbeitung verfügbar. Nach korrekter Ausführung der READ NEXT-Anweisung befindet sich der erste, "AB4" in den ersten drei Bytes seines primären Satzschlüssels aufweisende Datensatz in SATZ-U und kann bearbeitet werden.

Weitere Ausführungen der READ NEXT-Anweisung nehmen Zugriff auf den jeweils sequentiell nächsten Datensatz der Datei in aufsteigender Reihenfolge des Satzschlüssels SCHL-U.

Die zweite READ-Anweisung in dem Beispiel veranlaßt, daß der den Satzschlüssel "FH95" aufweisende Datensatz aus DATEI-U im Direktzugriff gelesen wird. Nach korrekter Ausführung der READ-Anweisung befindet sich der den Satzschlüssel "FH95" aufweisende Datensatz in SATZ-U. Wird in DATEI-U kein Datensatz mit einem Satzschlüssel gleich "FH95" gefunden, führt die READ-Anweisung (INVALID KEY) die Anweisung GO TO KEIN-U-SATZ aus.

Beispiel 3:

```
FILE-CONTROL.
    SELECT DATEI-V  ASSIGN TO DISC
           ORGANIZATION IS INDEXED
           ACCESS MODE IS DYNAMIC                    (!)
           RECORD KEY IS SCHL-V.

DATA DIVISION.
FILE SECTION.
FD  DATEI-V BLOCK CONTAINS 25 RECORDS
           RECORD CONTAINS 20 CHARACTERS
           LABEL RECORD IS STANDARD.
01  SATZ-V.
    02  FILLER       PIC XX
    02  SCHL-V       PIC XXXX.XX.
    02  FILLER       PIC X(14).

PROCEDURE DIVISION.
ANFANG.
    OPEN INPUT DATEI-V.
       --------------
    MOVE "1630" TO SCHL-V.
    READ DATEI-V INVALID KEY GO TO KEIN-SATZ-1.
       --------------
    MOVE "1936" TO SCHL-V.
    START DATEI-V KEY IS NOT LESS THAN SCHL-V
                  INVALID KEY GO TO KEIN-SATZ-2.
A.  READ DATEI-V NEXT RECORD AT END GO TO ENDE.
       --------------
    REWRITE SATZ-V INVALID KEY GO TO KEIN-SATZ-3.
    GO TO A.
```

Im obigen Beispiel handelt es sich um eine indexierte Datei mit dynamischem Zugriff, d. h., innerhalb desselben Programmes erfolgt sowohl direkter als auch sequentieller Zugriff auf die Datei. Die Datei hat den Namen DATEI-V. Da Datensätze sowohl aus DATEI-V zu lesen als auch in diese zurückzuschreiben sind, muß die Datei zuerst mit OPEN I-O eröffnet werden.

Der Programmierer überträgt den Wert 1630 in das primäre Satzschlüsselfeld SCHL-V, um den Datensatz, auf den durch die folgende Random-READ-Anweisung Direktzugriff erfolgen soll, zu spezifizieren. Die READ-Anweisung veranlaßt, daß der den Wert 1630 in seinem Satzschlüsselfeld aufweisende

Datensatz im Direktzugriff aus der Datei gelesen wird. Nachdem die READ-Anweisung korrekt ausgeführt wurde, befindet sich der betreffende Datensatz in SATZ-V. Wird in DATEI-V kein Datensatz gefunden, der 1630 als Satzschlüssel hat, führt die READ-Anweisung (INVALID KEY) GO TO KEIN-SATZ-1 aus.

Die START-Anweisung positioniert DATEI-V auf denjenigen Datensatz, dessen Satzschlüssel nicht kleiner als der Inhalt von SCHL-V ist. Da SCHL-V "1936" enthält, wird DATEI-V auf den ersten Datensatz positioniert, dessen Satzschlüssel entweder gleich dem oder größer als der Inhalt von SCHL-V ist. Angenommen, die aufsteigende Reihenfolge der Satzschlüssel in der DATEI-V ist 1934, 1937 und 1938, dann positioniert die START-Anweisung die DATEI-V auf den den Satzschlüssel 1937 aufweisenden Datensatz.

Nach korrekter Ausführung der START-Anweisung macht die READ NEXT-Anweisung den den Satzschlüssel 1937 aufweisenden Datensatz aus DATEI-V zur Verarbeitung verfügbar. Nach korrekter Ausführung der READ NEXT-Anweisung befindet sich der den Satzschlüssel 1937 aufweisende Datensatz in SATZ-V.

Weitere Ausführungen der READ NEXT-Anweisung nehmen Zugriff auf den jeweils sequentiell nächsten Datensatz in DATEI-V in aufsteigender Reihenfolge des Satzschlüssels SCHL-V.

Wenn die READ NEXT-Anweisung feststellt, daß in DATEI-V keine weiteren Sätze zur Verfügung stehen, wird AT END GO TO ENDE ausgeführt.

Die REWRITE-Anweisung schreibt den Inhalt von SATZ-V auf die DATEI-V zurück. Durch die vorangehende Ausführung der READ NEXT-Anweisung wurde in das Satzschlüsselfeld SCHL-V der Satzschlüsselwert desjenigen Datensatzes, auf den Zugriff erfolgte, geschrieben. Somit schreibt die REWRITE-Anweisung den Inhalt von SATZ-V auf die durch den Inhalt von SCHL-V vorgemerkte Datensatzposition in DATEI-V zurück.

DIE STOP-ANWEISUNG

Format:

```
STOP  { RUN     [Identifier]
      {         [Literal   ]
      {
      { Literal
```

Die STOP-Anweisung mit Literal bewirkt die Anzeige des Literals auf dem Bildschirm und einem Programm-Halt. Das Literal kann ein numerisches Literal, ein nicht-numerisches Literal oder eine figurative Konstante (ohne die figurative Konstante ALL Literal) sein. Ist ein numerisches Literal angegeben, muß dieses eine ganze Zahl ohne Vorzeichen sein.

Durch Drücken der NEW LINE-Taste wird die Ausführung des Objekt-Programmes wieder aufgenommen.

Durch die Anweisung STOP RUN kommt das Programm zum Abschluß und das Betriebssystem erlangt die Kontrolle über den Konsolbildschirm. Bei zusätzlicher Angabe eines Identifiers oder Literals in der STOP RUN-Anweisung wird der entsprechende Wert in das JCL-CODE-Register übertragen und kann nach der Beendigung des Programmes z. B. in einem Control-String abgefragt werden.

DIE STRING-ANWEISUNG

Die STRING-Anweisung bewirkt das Aneinanderreihen der Inhalte zweier oder mehrerer Datenfelder in einem einzigen Datenfeld.

Format:

```
STRING {Identifier-1} [Identifier-2] ... DELIMITED BY {Identifier-3}
       {Literal-1   } [Literal-2   ]                  {Literal-3   }
                                                      {SIZE        }

      [{Identifier-4} [Identifier-5] ... DELIMITED BY {Identifier-6}]
      [{Literal-4   } [Literal-5   ]                  {Literal-6   }]
                                                      {SIZE        }

  INTO Identifier-7  [WITH POINTER Identifier-8]

 [ON OVERFLOW unbedingte Anweisung 1]

 [NOT ON OVERFLOW unbedingte Anweisung 2]

 [END-STRING]
```

Durch die STRING-Anweisung werden zuerst die Inhalte des Identifiers-1 oder Literals-1 linksbündig in Identifier-7 übertragen. Daraufhin werden durch die STRING-Anweisung die Inhalte des Identifiers-2 oder des Literals-2 in Identifier-7 übertragen, und zwar so, daß das äußerste linke Zeichen von Identifier-2 oder Literal-2 rechts von dem äußersten rechten Zeichen zu liegen kommt, das vorher vom Identifier-1 oder Literal-1 übertragen wurde. Und so wird mit dem Inhalt des Identifiers-4 (oder des Literals-4) und des Identifiers-5 (oder des Literals-5) die Übertragung in Identifier-7 fortgesetzt.

In der STRING-Anweisung stellen Identifier-1, Literal-1, Identifier-2, Literal-2, Identifier-4, Literal-4 sowie Identifier-5 und Literal-5 die Sendefelder, Identifier-7 das Empfangsfeld dar.

Die Ausführung der STRING-Anweisung endet entweder, wenn alle Zeichenpositionen in Identifier-7 (dem Empfangsfeld)

Daten aus den Sendefeldern erhalten haben oder wenn keine Zeichen mehr von den Sendefeldern zu übertragen sind. Wenn die Gesamtzahl der in Identifier-7 übertragenen Zeichen geringer ist als die Anzahl der Zeichenpositionen im Identifier-7, bleiben die restlichen vorhandenen Zeichenpositionen in Identifier-7 unverändert.

Alle Literale in der STRING-Anweisung müssen nichtnumerische Literale sein. Jedes Literal in der STRING-Anweisung kann eine beliebige figurative Konstante sein. Ist ein Literal eine figurative Konstante, dann kommt es einem 1-Zeichen-Datenfeld gleich.

In der STRING-Anweisung müssen alle Identifier mit Ausnahme von Identifier-8 indirekt oder direkt mit der Klausel USAGE IS DISPLAY definiert sein. Wird irgendein Identifier, Identifier-8 ausgenommen, als ein numerisches Elementarfeld definiert, muß die Beschreibung für diesen Identifier eine Ganzzahl ohne das Symbol P in der PICTURE-Zeichenfolge definieren. Identifier-7 muß ein alphanumerisches Elementarfeld sein. Die Definition für Identifier-7 darf in der PICTURE-Zeichenfolge keine Editierzeichen enthalten und muß ohne JUSTIFIED-Klausel sein.

Alles, was in den folgenden Abschnitten über Identifier-1, Literal-1, Identifier-2, Literal-2 sowie Identifier-3 und Literal-3 gesagt wird, gilt gleichermaßen für Identifier-4, Literal-4, Identifier-5, Literal-5 sowie Identifier-6 und Literal-6.

DELIMITED BY

Die Klausel DELIMITED BY spezifiziert das oder die Zeichen (Trennsymbole), die die äußerste rechte Grenze der von einem Sendefeld zu übertragenden Zeichen bilden. Identifier-3 bzw. Literal-3 geben das oder die Zeichen an, derer Auftreten im Sendefeld die Übertragung dieses Datenfeldes in das Empfangsfeld (Identifier-7) beenden soll. Das oder die durch Identifier-3 bzw. Literal-3 als Trennsymbol oder Trennsymbole angegebene(n) Zeichen wird (werden) nicht zum Empfangsfeld mit übertragen. Die DELIMITED BY SIZE-Variante gibt an, daß alle Zeichen aus dem Sendefeld in das Empfangsfeld zu übertragen sind.

POINTER

Mit der POINTER-Klausel kann der Programmierer das Datenfeld spezifizieren, das die Zeichenposition des Empfangsfeldes bestimmt, ab der Daten einzufügen sind. Identifier-8 bezeichnet das Datenfeld, das als POINTER auf eine Zeichenposition im Empfangsfeld verwendet wird. Identifier-8 muß ein numerisch-ganzzahliges Elementarfeld sein; die Datendefinition für diesen Identifier darf in der PICTURE-Zeichenfolge kein Symbol P enthalten. Zeichenposition 1 ist die äußerste linke Zeichenposition im Empfangsfeld. Zeichenposition 2 ist die nächste, also zweite Zeichenposition des Empfangsfeldes, usw.

Der maximale Wert, der in Identifier-8 enthalten sein kann, entspricht der um eine Zeichenposition höheren Anzahl der in Identifier-7 vorhandenen Zeichenpositionen. Ist die POINTER-Klausel in der STRING-Anweisung angegeben, muß der Programmierer vor Ausführung der STRING-Anweisung einen Wert in Identifier-8 einsetzen. Dieser Anfangswert darf nicht kleiner als 1 sein.

Durch die STRING-Anweisung erfolgt eine Übertragung, und zwar Zeichen um Zeichen, in eine durch den Wert des Identifiers-8 angezeigte Zeichenposition des Identifiers-7. Nachdem ein Zeichen übertragen wurde, wird der Wert in Identifier-8 jedesmal vor Übertragung des nächsten Zeichens um 1 erhöht.

Ist die POINTER-Klausel nicht angegeben, hat der Programmierer keinen Zugriff zum POINTER-Datenfeld, da daraus ein vom Compiler generiertes Datenfeld wird. Zu Beginn einer STRING-Anweisung ohne POINTER-Klausel ist der Anfangswert dieses POINTER-Datenfeldes 1. Während der Ausführung der STRING-Anweisung wird der Inhalt dieses POINTER-Datenfeldes erhöht.

OVERFLOW

Eine OVERFLOW-Bedingung ist dann gegeben, wenn während der Ausführung einer STRING-Anweisung der Wert des POINTER-Datenfeldes (Identifier-8) kleiner als 1 ist oder die Anzahl der Zeichenpositionen in Identifier-7 überschreitet. In diesem Fall werden keine weiteren Daten in den Identifier-7 übertragen.

Bei bestehender OVERFLOW-Bedingung und spezifizierter OVERFLOW-Klausel wird die unbedingte Anweisung in der

OVERFLOW-Klausel ausgeführt. Bei bestehender OVERFLOW-Bedingung und weggelassener OVERFLOW-Klausel wird die auf die STRING-Anweisung folgende Anweisung ausgeführt und die OVERFLOW-Bedingung wird nicht festgestellt.

Beispiel 1:

```
MOVE 1 TO ZEICHEN-ZAHL.
STRING FELD-1  FELD-2 DELIMITED  BY SIZE INTO FELD-3
        WITH  POINTER ZEICHEN-ZAHL.
```

FELD-1	vorher	129	
FELD-2	vorher	35	
FELD-3	vorher	00287	
ZEICHEN-ZAHL	vorher	01	
FELD-1	nachher	129	unverändert
FELD-2	nachher	35	unverändert
FELD-3	nachher	12935	enthält FELD-1 und FELD-2
ZEICHEN-ZAHL	nachher	06	enthält die letzte betroffene Zeichenposition plus 1

Beispiel 2:

```
MOVE 1 TO ZEICHEN-ZAHL.
STRING FELD-4  FELD-5 DELIMITED BY SIZE INTO FELD-6
        WITH  POINTER ZEICHEN-ZAHL.
```

FELD-4	vorher	AB	
FELD-5	vorher	1524	
FELD-6	vorher	Z2Z3Z5Z92	
ZEICHEN-ZAHL	vorher	001	
FELD-4	nachher	AB	unverändert
FELD-5	nachher	1524	unverändert
FELD-6	nachher	AB1524Z92	enthält 6 Zeichen von FELD-4 und FELD-5
ZEICHEN-ZAHL	nachher	007	

Beispiel 3:

```
MOVE 1 TO ZEICHEN-ZAHL.
STRING FELD-7  FELD-8  FELD-9  DELIMITED  BY SIZE INTO
        FELD-10 WITH POINTER ZEICHEN-ZAHL.
```

FELD-7	vorher	ABC	
FELD-8	vorher	12345	
FELD-9	vorher	Z469	
FELD-10	vorher	XYPB3709J	
ZEICHEN-ZAHL	vorher	01	
FELD-7	nachher	ABC	unverändert
FELD-8	nachher	12345	unverändert
FELD-9	nachher	Z469	unverändert
FELD-10	nachher	ABC12345Z	enthält FELD-7, FELD-8 und ein Zeichen von FELD-9
ZEICHEN-ZAHL	nachher	10	

Beispiel 4::

```
MOVE 1 TO POSITION-FELD.
STRING FELD-11 DELIMITED BY "A" FELD-12 DELIMITED  BY "3"
        INTO FELD-13 WITH POINTER POSITION-FELD.
```

FELD-11	vorher	12ABA	
FELD-12	vorher	1493201	
FELD-13	vorher	003027876	
POSITION-FELD	vorher	1	
FELD-11	nachher	12ABA	unverändert
FELD-12	nachher	1493201	unverändert
FELD-13	nachher	121497876	enthält 12 von FELD-11 und 149 von FELD-12
POSITION-FELD	nachher	6	

Beispiel 5:

```
MOVE 1 TO ZAHL-S.
STRING FELD-14 DELIMITED  BY "AB", FELD-15 DELIMITED BY SIZE
        INTO FELD-16 WITH POINTER ZAHL-S.
```

FELD-14	vorher	12345ABCD	
FELD-15	vorher	MNTSU	
FELD-16	vorher	00000000000000	
ZAHL-S	vorher	01	
FELD-14	nachher	12345ABCD	
FELD-15	nachher	MNTSU	
FELD-16	nachher	12345MNTSU0000	enthält 12345 von FELD-14 und MNTSU von FELD-15
ZAHL-S	nachher	11	

Beispiel 6:

```
STRING FELD-17, "ABC" DELIMITED  BY SIZE INTO FELD-18.
```

FELD-17	vorher	44792	
Literal "ABC"	vorher	ABC	
FELD-18	vorher	XYZ00000	
FELD-17	nachher	44792	unverändert
Literal "ABC"	nachher	ABC	unverändert
FELD-18	nachher	44792ABC	enthält FELD-17 und das Literal "ABC"

Beispiel 7:

```
MOVE 2 TO ZEICHEN-POS.
STRING FELD-19, FELD-20 DELIMITED  BY "ϕ" INTO FELD-21
        WITH POINTER ZEICHEN-POS.
```

FELD-19	vorher	A257ϕϕ1	
FELD-20	vorher	ϕB24ϕϕϕϕ	
FELD-21	vorher	CDPAKJ12904	
ZEICHEN-POS	vorher	02	
FELD-19	nachher	A257ϕϕ1	unverändert
FELD-20	nachher	ϕB24ϕϕϕϕ	unverändert
FELD-21	nachher	CA257J12904	enthält A257 von FELD-19 in Zeichenposition 2 bis 5
ZEICHEN-POS	nachher	06	

DIE SUBTRACT-ANWEISUNG

Die SUBTRACT-Anweisung subtrahiert einen numerischen Operanden (oder die Summe zweier oder mehrerer numerischer Operanden) von einem numerischen Operanden und speichert das Ergebnis.

Format 1:

```
SUBTRACT { Identifier-1 } [ Identifier-2 ] ... FROM Identifier-m [ROUNDED]
         { Literal-1    } [ Literal-2    ]                [Identifier-n [ROUNDED]]...
    [ON SIZE ERROR  unbedingte Anweisung 1]
    [NOT ON SIZE ERROR  unbedingte Anweisung 2]
    [END-SUBTRACT]
```

Format 2:

```
SUBTRACT { Identifier-1 } [ Identifier-2 ] ... FROM { Identifier-m }
         { Literal-1    } [ Literal-2    ]         { Literal-m    }
GIVING Identifier-n [ROUNDED] [Identifier-o [ROUNDED]]...
    [ON SIZE ERROR  unbedingte Anweisung 1]
    [NOT ON SIZE ERROR unbedingte Anweisung 2]
    [END-SUBTRACT]
```

Bei Format 1 werden die dem Wort FROM vorausgehenden Operanden addiert. Die Gesamtsumme wird dann vom Identifier-m subtrahiert. Das Ergebnis der Subtraktion wird im Identifier-m gespeichert, der Empfangsfeld oder Ergebnisfeld genannt wird. Danach wird dieselbe Summe von Identifier-n subtrahiert. Das Ergebnis wird in Identifier-n gespeichert. Diese Operationen werden für alle Operanden wiederholt, die nach dem Wort FROM stehen.

Bei Format 2 werden die dem Wort FROM vorausgehenden Operanden addiert. Die Gesamtsumme wird dann vom Identifier-m oder Literal-m subtrahiert. Das Ergebnis dieser Subtraktion wird dann als der neue Wert in Identifier-n, Identifier-o, usw., gespeichert, die Empfangsfelder oder Ergebnisfelder genannt werden.

Bei Format 1 muß jeder Identifier ein numerisches Elementarfeld bezeichnen. Bei Format 2 muß jeder Identifier außer Identifier-n, Identifier-o, usw. ein numerisches Elementarfeld bezeichnen. Die Identifier-n, Identifier-o, usw. bei Format 2 können entweder ein numerisches oder ein numerisch-editiertes Elementarfeld sein. Jedes Literal in der SUBTRACT-Anweisung muß ein numerisches Literal sein.

SUBTRACT ist eine unbedingte Anweisung. Bei Vorhandensein von SIZE ERROR oder NOT SIZE ERROR wird SUBTRACT jedoch zu einer bedingten Anweisung.

ROUNDED

Weist das Resultat der Subtraktion mehr Dezimalstellen (Positionen rechts vom Dezimalpunkt) als das Ergebnisfeld auf, werden beide nach dem Dezimalpunkt ausgerichtet. Ohne die Angabe ROUNDED erfolgt ein Abschneiden. Bei Vorhandensein von ROUNDED erhöht sich der absolute Wert des Ergebnisfeldes immer dann um 1, wenn die erste abzuschneidende Ziffer größer als 4 ist.

SIZE ERROR

Nach Dezimalpunktausrichtung und Rundung wird, wenn SIZE ERROR spezifiziert ist, eine Prüfung auf eine SIZE ERROR-Bedingung hin vorgenommen. Die SIZE ERROR-Bedingung liegt vor, wenn der Wert des Ergebnisses den höchsten vom Ergebnisfeld aufnehmbaren Wert überschreitet. Lediglich die linke Seite des Empfangsdatenfeldes wird auf Überlauf hin geprüft. Bei Auftreten der SIZE ERROR-Bedingung wird der Wert des Ergebnisfeldes nicht im Ergebnisfeld gespeichert. Hat eine SUBTRACT-Anweisung mehr als ein Empfangsfeld, und in einem Feld entsteht eine SIZE ERROR-Bedingung, wird das Ergebnis in dieses Feld nicht übertragen. Ensteht in einem oder mehreren Empfangsfeldern eine SIZE ERROR-Bedingung, werden alle korrekten SUBTRACT-Vorgänge ausgeführt und danach die unbedingte Anweisung in SIZE ERROR ausgeführt.

Bei verwendeter SIZE ERROR-Klausel, jedoch ohne Vorliegen einer solchen Bedingung, wird das Ergebnis der Subtraktion in den Ergebnisfeldern gespeichert und die nächste Anweisung ausgeführt.

Ohne spezifizierte SIZE ERROR-Angabe ist bei Auftreten einer SIZE ERROR-Bedingung der Wert der Ergebnisfelder unvorhersagbar. Die auf die SUBTRACT-Anweisung folgende Anweisung wird ausgeführt und der Fehler nicht festgestellt.

Beispiel 1:

```
SUBTRACT WERT FROM SALDO.

WERT      vorher     036^20+
SALDO     vorher     100^04+
WERT      nachher    036^20+     unverändert
SALDO     nachher    063^84+     enthält Ergebnis der
                                 Subtraktion
```

Beispiel 2:

```
SUBTRACT 378.5  FROM TOTAL.

Literal 378.5  vorher    0378^5+
TOTAL          vorher     478+
Literal 378.5  nachher   0378^5+    unverändert
TOTAL          nachher    099+      enthält das
                                    abgeschnittene Ergebnis
                                    der Subtraktion
```

Beispiel 3:

```
SUBTRACT 378.5 FROM TOTAL ROUNDED.

Literal 378.5  vorher    0378^5+
TOTAL          vorher     478+
Literal 378.5  nachher   0378^5+    unverändert
TOTAL          nachher    100+      enthält das gerundete
                                    Ergebnis der Subtraktion
```

Beispiel 4:

```
SUBTRACT ABGANG FROM LAGER GIVING MENGE-NEU.
```

ABGANG	vorher	75	
LAGER	vorher	3940	
MENGE-NEU	vorher	0100	
ABGANG	nachher	75	unverändert
LAGER	nachher	3940	unverändert
MENGE-NEU	nachher	3865	enthält das Ergebnis der Subtraktion

Beispiel 5:

```
SUBTRACT -50 FROM TOTAL GIVING MENGE ROUNDED.
```

Literal-50	vorher	50-	
TOTAL	vorher	375^9-	
MENGE	vorher	492+	
Literal-50	nachher	50-	unverändert
TOTAL	nachher	375^9-	unverändert
MENGE	nachher	326-	enthält das gerundete Ergebnis

Beispiel 6:

```
SUBTRACT VERPACKUNG, PORTO FROM BRUTTO.
```

VERPACKUNG	vorher	0025^00	
PORTO	vorher	0005^26	
BRUTTO	vorher	0160^26	
VERPACKUNG	nachher	0025^00	unverändert
PORTO	nachher	0005^26	unverändert
BRUTTO	nachher	0130^00	enthält Ergebnis der Subtraktion

Beispiel 7:

```
SUBTRACT VERPACKUNG, PORTO FROM BRUTTO GIVING NETTO.

VERPACKUNG      vorher    0025^00
PORTO           vorher    0005^26
BRUTTO          vorher    0160^26
NETTO           vorher    0204^00
VERPACKUNG      nachher   0025^00      unverändert
PORTO           nachher   0005^26      unverändert
BRUTTO          nachher   0160^26      unverändert
NETTO           nachher   0130^00      enthält das Ergebnis
                                       der Subtraktion
```

Beispiel 8:

```
SUBTRACT VERPACKUNG, PORTO FROM BRUTTO, NETTO.

VERPACKUNG      vorher    0025^00
PORTO           vorher    0005^26
BRUTTO          vorher    0160^26
NETTO           vorher    0204^00
VERPACKUNG      nachher   0025^00      unverändert
PORTO           nachher   0005^26      unverändert
BRUTTO          nachher   0130^00      enthält das Ergebnis
                                       von 160.26 - 30.26
NETTO           nachher   0173^74      enthält das Ergebnis
                                       von 204.00 - 30.26
```

Beispiel 9:

```
SUBTRACT VERPACKUNG, PORTO FROM BRUTTO GIVING NETTO-1,
         NETTO-2.

VERPACKUNG      vorher    0025^00
PORTO           vorher    0005^26
BRUTTO          vorher    0160^26
NETTO-1         vorher    1000^50
NETTO-2         vorher      9999
VERPACKUNG      nachher   0025^00      unverändert
PORTO           nachher   0005^26      unverändert
BRUTTO          nachher   0160^26      unverändert
NETTO-1         nachher   0130^00      enthält das Ergebnis
                                       von 160.26 - 30.26
NETTO-2         nachher   0130         enthält das Ergebnis
                                       von 160.26 - 30.26
```

DIE SUBTRACT CORRESPONDING-ANWEISUNG

Bei der CORRESPONDING-Variante der SUBTRACT-Anweisung werden Felder gleichen Namens aus zwei Gruppen subtrahiert und die Differenz in den entsprechenden Feldern der Empfangsgruppe gespeichert.

Format:

```
SUBTRACT   {CORRESPONDING}
           {CORR         }

     Identifier-1 FROM Identifier-2  [ROUNDED]

  [ON SIZE ERROR unbedingte Anweisung 1]

  [NOT ON SIZE ERROR unbedingte Anweisung 2]

  [END-SUBTRACT]
```

Folgende Bedingungen müssen erfüllt sein, damit Datenfelder bei einer SUBTRACT CORRESPONDING-Anweisung subtrahiert werden.

- Das Wort FILLER in Identifier-1 und Identifier-2 gilt nicht als korrespondierender Name.
- Ein Datenfeld in Identifier-1 und Identifier-2 hat einen identischen Datennamen und identische Qualifikatoren mit Ausnahme des Identifier-1 und Identifier-2 selbst.
- Beide Datenfelder müssen Elementarfelder sein.
- Keine der beiden Datenbeschreibungen darf eine RENAMES-, REDEFINES-, OCCURS- oder USAGE IS INDEX-Klausel enthalten.

Der Inhalt der Elementarfelder in Identifier-1 wird vom Wert der korrespondierenden Datenfelder in Identifier-2 subtrahiert. Das Ergebnis wird in den korrespondierenden Datenfeldern in Identifier-2 gespeichert. Identifier-1 und Identifier-2 müssen Feldgruppen bezeichnen. Identifier-1 und Identifier-2 dürfen nicht auf Stufennummer 66, 77 oder 88 definiert sein. Identifier-1 und Identifier-2 dürfen eine REDEFINES- oder OCCURS-Klausel enthalten, sie können auch Unterstufen eines Datennamens sein, der eine OCCURS- oder REDEFINES-Klausel enthält.

ROUNDED

Weist das Resultat der Subtraktion mehr Dezimalstellen (Positionen rechts vom Dezimalpunkt) als das Ergebnisfeld auf, dann werden beide nach dem Dezimalpunkt ausgerichtet. Ohne die Angabe ROUNDED erfolgt ein Abschneiden. Bei Vorhandensein von ROUNDED erhöht sich der absolute Wert des Ergebnisfeldes immer dann um 1, wenn die signifikanteste Ziffer größer als 4 ist.

SIZE ERROR

Nach Dezimalpunktausrichtung und Rundung wird, wenn SIZE ERROR spezifiziert ist, eine Prüfung auf eine SIZE ERROR-Bedingung hin vorgenommen. Die SIZE ERROR-Bedingung liegt vor, wenn der Wert des Ergebnisses den höchsten vom Ergebnisfeld aufnehmbaren Wert überschreitet. Lediglich die linke Seite des Empfangsdatenfeldes wird auf Überlauf hin geprüft. Bei Auftreten der SIZE ERROR-Bedingung wird der Wert des Ergebnisses nicht im Ergebnisfeld gespeichert. Bei mehreren Empfangsfeldern wird in allen das Ergebnis gespeichert, die keine SIZE ERROR-Bedingung verursachen. Wird in einem der Empfangsfelder eine SIZE ERROR-Bedingung festgestellt, wird die SUBTRACT CORRESPONDING-Anweisung normal zu Ende geführt und danach die unbedingte Anweisung ausgeführt.

Bei verwendeter SIZE ERROR-Klausel, jedoch ohne Vorliegen einer solchen Bedingung, wird das Ergebnis der Subtraktion in den Ergebnisfeldern gespeichert und die nächste Anweisung ausgeführt.

Ohne spezifizierte SIZE ERROR-Angabe ist bei Auftreten einer SIZE ERROR-Bedingung der Wert des Ergebnisfeldes unvorhersagbar. Die auf die SUBTRACT CORRESPONDING-Anweisung folgende Anweisung wird ausgeführt und der Fehler nicht festgestellt.

Ein Beispiel finden Sie auf der nächsten Seite:

Beispiel:

```
SUBTRACT CORRESPONDING BESTELLUNG FROM BESTAND.
```

```
01 AUFTRAGSSATZ                01 LAGERSATZ
   02 KUNDENNR                    02 LAGERNR
   02 BESTELLUNG                  02 BESTAND
      03 MENGE1                   03 TOTALMENGE
      03 MENGE2                   03 MENGE1
      03 MENGE3                   03 MENGE2
      03 MENGE4                   03 MENGE3
      03 MENGE5                   03 MENGE4
```

Das Ergebnis der SUBTRACT CORRESPONDING-Anweisung entspricht dem Ergebnis folgender Einzelanweisungen:

```
SUBTRACT MENGE1 IN BESTELLUNG FROM MENGE1 IN BESTAND.
SUBTRACT MENGE2 IN BESTELLUNG FROM MENGE2 IN BESTAND.
SUBTRACT MENGE3 IN BESTELLUNG FROM MENGE3 IN BESTAND.
SUBTRACT MENGE4 IN BESTELLUNG FROM MENGE4 IN BESTAND.
```

Das Feld MENGE5 wird nicht subtrahiert, da kein korrespondierendes Datenfeld vorhanden ist.

DIE UNLOCK-ANWEISUNG

Mit dieser Anweisung kann konkurrierenden Programmen erlaubt werden, auf einen Block in einer Datei zuzugreifen, der im Augenblick durch das eigene Programm bearbeitet wird und somit einer automatischen Sperrung unterliegt.

Format:

UNLOCK Dateiname RECORD

Die Datei muß INPUT- oder I-O-eröffnet sein.

Ist in der Datei gar kein Block gesperrt, löst die UNLOCK-Anweisung auch keine Aktion aus.

UNLOCK beeinflußt ein ggf. definiertes FILE-STATUS-Feld.

UNLOCK kann an ein anderes Programm nur den letzten gesperrten Block freigeben. UNLOCK kann keine Blöcke freimachen, die von einem anderen Programm gesperrt wurden.

DIE UNSTRING-ANWEISUNG

Die UNSTRING-Anweisung bewirkt das Aufteilen von Daten eines einzelnen Datenfeldes in verschiedene andere Datenfelder.

Format:

```
UNSTRING  Identifier-1

[DELIMITED BY [ALL] {Identifier-2}
                    {Literal-1   }

[OR  [ALL] {Identifier-3}] ...]
           {Literal-2   }

INTO Identifier-4  [DELIMITER IN Identifier-5]
                   [COUNT IN Identifier-6]
     [Identifier-7 [DELIMITER IN Identifier-8]
                   [COUNT IN Identifier-9]]

[WITH POINTER Identifier-10]

[TALLYING IN Identifier-11]

[ON OVERFLOW unbedingte Anweisung 1]

[NOT ON OVERFLOW unbedingte Anweisung 2]

[END-UNSTRING]
```

Durch die UNSTRING-Anweisung wird das äußerste linke Zeichen des Identifiers-1 in die äußerste linke Zeichenposition des Identifiers-4 übertragen. Das nächste Zeichen des Identifiers-1 wird in die nächste Zeichenposition des Identifiers-4 übertragen. Die Zeichen um Zeichen erfolgende Übertragung der Daten in Identifier-4 geht weiter, bis alle Zeichenpositionen des Identifiers-4 Daten empfangen haben. Das nächste verfügbare Zeichen des Identifiers-1 wird dann in die äußerste linke Zeichenposition des Identifiers-7 übertragen. Die Zeichen um Zeichen erfolgende Übertragung der Daten von Identifier-1 in Identifier-7 geht weiter, bis alle Zeichenpositionen des Identifiers-7 Daten empfangen haben. In der UNSTRING-Anweisung ist Identifier-1 das Sende-

feld, Identifier-4, Identifier-7, usw. sind die Empfangsfelder.

Die UNSTRING-Anweisung endet entweder, wenn in alle Empfangsfelder Daten übertragen wurden oder wenn alle Zeichen aus Identifier-1 (Sendefeld) übertragen wurden. Die Übertragung erfolgt wie eine alphanumerische Übertragung mittels MOVE.

Alle Literale in der UNSTRING-Anweisung müssen nichtnumerische Literale sein. Jedes Literal in der UNSTRING-Anweisung kann eine beliebige figurative Konstante sein. Ist ein Literal eine figurative Konstante, dann stellt diese ein 1-Zeichen-Datenfeld dar.

In der UNSTRING-Anweisung müssen Identifier-1, Identifier-2, Identifier-3, Identifier-4, Identifier-5, Identifier-7 und Identifier-8 indirekt oder direkt mit der Klausel USAGE IS DISPLAY definiert sein. Identifier-4 und Identifier-7 können als entweder alphabetisch, alphanumerisch oder numerisch definiert sein. Ist Identifier-4 oder Identifier-7 alphabetisch, dann darf die PICTURE-Zeichenfolge dieses Datenfeldes nicht das Symbol B enthalten. Ist Identifier-4 oder Identifier-7 numerisch, dann darf die PICTURE-Zeichenfolge dieses Datenfeldes nicht das Symbol P enthalten.

In der UNSTRING-Anweisung müssen Identifier-6, Identifier-9, Identifier-10 und Identifier-11 als numerisch-ganzzahlige Elementarfelder definiert sein. Die Datenfeldbeschreibung für diese Identifier darf in der PICTURE-Zeichenfolge nicht das Symbol P enthalten.

Jede mit Identifier-1, Identifier-10 oder Identifier-11 verbundene Tabellen-Subskribierung oder -Indizierung wird nur einmal ausgewertet. Diese Auswertung erfolgt unmittelbar bevor irgendwelche Daten übertragen werden. Jede mit Identifier-2, Identifier-3, Identifier-4, Identifier-5 oder Identifier-6 verbundene Subskribierung oder Indizierung wird unmittelbar vor Übertragung der Daten in das jeweilige Datenfeld berücksichtigt.

Alles, was in den folgenden Abschnitten über Identifier-2 oder Literal-1 gesagt wird, gilt gleichermaßen für Identifier-3 oder Literal-2. Alles, was über Identifier-4, Identifier-5 und Identifier-6 gesagt wird, trifft auch auf Identifier-7, Identifier-8 und Identifier-9 zu.

DELIMITED BY

Die Klausel DELIMITED BY spezifiziert das oder die Zeichen (Trennsymbole), die die äußerste rechte Grenze der aus Identifier-1 in eines der Empfangsfelder übertragenen Zeichen bilden. Identifier-2 bzw. Literal-1 geben das oder die Zeichen an, deren Auftreten im Sendefeld die Übertragung in ein Empfangsfeld beenden soll. Das oder die Trennsymbol(e) wird (werden) vom Sendefeld nicht ins Empfangsfeld mit übertragen.

Das durch Identifier-2 bzw. Literal-1 spezifizierte Trennsymbol kann ein oder alle Zeichen des ASCII-Zeichenvorrats umfassen. Besteht das durch Identifier-2 bzw. Literal-1 definierte Trennsymbol aus zwei oder mehr Zeichen, müssen alle diese Zeichen in der (gemäß DELIMITED BY) gegebenen Reihenfolge vorliegen.

Folgen zwei Trennsymbole innerhalb des Identifiers-1 (dem Sendefeld) unmittelbar aufeinander, beendet das erste Trennsymbol die Übertragung der Daten zum entsprechenden Empfangsfeld. Das zweite Trennsymbol verursacht, daß das nächste Empfangsfeld entweder mit Leerstellen oder Nullen aufgefüllt wird, je nach Definition dieses Empfangsfeldes (je nach Datenkategorie).

Wenn die Klausel DELIMITED BY eine ALL-Klausel enthält, wird das mehrmalige aufeinanderfolgende Auftreten des betreffenden Trennsymbols als ein einziges Trennsymbol betrachtet; somit wird davon nur ein Empfangsfeld beeinflußt. Enthält die Klausel DELIMITED BY wiederum eine OR-Klausel, so gibt dies die Möglichkeit, zwei oder mehr Trennsymbole für das Sendefeld zu definieren. Jedes Trennsymbol wird in der spezifizierten Reihenfolge für das Sendefeld verwendet. Das Sendefeld wird zuerst nach einem Identifier-2 oder Literal-1 entsprechenden Trennsymbol abgesucht. Wird im Sendefeld kein solches Trennsymbol gefunden, wird das Sendefeld nach einem Identifier-3 oder Literal-2 entsprechenden Trennsymbol abgesucht.

Nachdem die Anzahl der aus dem Sendefeld in das entsprechende Empfangsfeld zu übertragenden Zeichen bestimmt ist, werden die Zeichen übertragen. Ist das Ende des Empfangsfeldes erreicht, noch ehe alle durch ein Trennsymbol isolierten Zeichen übertragen wurden, erfolgt ein Abschneiden der restlichen Zeichen.

Wird das Ende des Sendefeldes erreicht, noch ehe ein Trennsymbol gefunden wurde, dann werden die auf das Trennsymbol hin abgesuchten Zeichen nach den Regeln der MOVE-Anweisung wie ein alphanumerisches Elementarfeld in das entsprechende Empfangsfeld übertragen.

Wird in einer UNSTRING-Anweisung die Klausel DELIMITED BY weggelassen, entspricht die Anzahl der in das entsprechende Empfangsfeld übertragenen Zeichen der Größe dieses Empfangsfeldes. Bei Empfangsfeldern, die mit einer separaten Vorzeichen-Stelle beschrieben sind, wird nur der Datenteil ohne Vorzeichen-Stelle aus dem Sendefeld übertragen. Ist die im Sendefeld befindliche Anzahl von Zeichen geringer als die Anzahl von Zeichenpositionen im entsprechenden Empfangsfeld, werden die Zeichen aus dem Sendefeld nach den Regeln der MOVE-Anweisung wie ein alphanumerisches Elementarfeld in das entsprechende Empfangsfeld übertragen.

DELIMITER IN

Enthält eine UNSTRING-Anweisung die Klausel DELIMITED BY, so kann der Programmierer die Klausel DELIMITER IN spezifizieren. Die Klausel DELIMITER IN bewirkt, daß das Trennsymbol, das die Übertragung von Daten zum entsprechenden Empfangsfeld beendet, im Identifier-5 gespeichert wird. Das oder die das Trennsymbol bildende(n) Zeichen wird (werden) wie ein alphanumerisches Elementarfeld in das durch den Identifier-5 spezifizierte Datenfeld übertragen.

Ist das Ende des Sendefeldes erreicht, noch ehe ein Trennsymbol gefunden wurde, werden Leerzeichen in Identifier-5 gebracht.

COUNT IN

Enthält eine UNSTRING-Anweisung die Klausel DELIMITED BY, so kann der Programmierer die Klausel COUNT IN spezifizieren. Die Klausel COUNT IN bewirkt, daß eine Zählung der übertragenen Zeichen ausgeführt wird.

POINTER

Durch die POINTER-Klausel kann der Programmierer das Datenfeld spezifizieren, das eine Anfangsposition in dem durch die UNSTRING-Anweisung zu behandelnden Sendefeld angibt. Identifier-10 spezifiziert dieses als POINTER verwendete Datenfeld. Identifier-10 muß ein numerisch-ganzzahliges Elementarfeld sein; seine PICTURE-Zeichenfolge darf das Symbol P nicht enthalten. Zeichenposition 1 ist die äußerste linke Zeichenposition des Sendefeldes. Zeichenposition 2 ist die zweite Zeichenposition von der linken Seite des Sendefeldes aus gerechnet. Ist die POINTER-Klausel in der UNSTRING-Anweisung angegeben, muß der Programmierer vor Ausführung der UNSTRING-Anweisung einen Wert in Identifier-10 übertragen. Dieser Anfangswert des Identifiers-10 darf nicht kleiner sein als 1.

Die UNSTRING-Anweisung bewirkt, daß der POINTER in Identifier-10 für jedes vom Sendefeld übertragene Zeichen und für jedes Trennsymbol um 1 erhöht wird. Somit enthält Identifier-10, wenn die UNSTRING-Anweisung beendet ist, einen Wert, der der nächsten Zeichenposition nach dem in dem Sendefeld verarbeiteten letzten Zeichen oder Trennsymbol entspricht.

Wird die POINTER-Klausel in einer UNSTRING-Anweisung weggelassen, beginnt die Auflösung des Sendefeldes (Identifier-1) mit dessen äußerster linken Zeichenposition (Zeichenposition 1).

TALLYING

Durch die TALLYING-Klausel kann der Programmierer ein Datenfeld spezifizieren, das für jedes Empfangsfeld, in das durch UNSTRING Daten übertragen werden, um 1 erhöht wird. Identifier-11 muß ein numerisch-ganzzahliges Elementarfeld sein; seine PICTURE-Zeichenfolge darf das Symbol P nicht enthalten. Wird die TALLYING-Klausel in der UNSTRING-Anweisung verwendet, muß der Programmierer vor Ausführung der UNSTRING-Anweisung einen Wert in den Identifier-11 bringen (z. B. Null). Somit enthält Identifier-11, wenn die UNSTRING-Anweisung beendet ist, einen Wert, der seinem Anfangswert plus der Anzahl der beschickten Datenfelder entspricht.

OVERFLOW

Die UNSTRING-Anweisung führt zu einem OVERFLOW, wenn eine der folgenden Bedingungen auftritt:

1. Der Wert des POINTER-Datenfeldes (Identifier-10) ist kleiner als 1 oder überschreitet die Anzahl der Zeichenpositionen im Sendefeld (Identifier-1).

2. Daten wurden in alle Empfangsfelder übertragen, aber das Ende des Sendefeldes (Identifier-1) ist noch nicht erreicht.

Bei gegebener OVERFLOW-Bedingung und spezifizierter ON-OVERFLOW-Klausel wird die unbedingte Anweisung 1 ausgeführt.

Tritt eine OVERFLOW-Bedingung auf, die OVERFLOW-Klausel wurde jedoch in der Anweisung nicht verwendet, wird die auf die UNSTRING-Anweisung folgende Anweisung ausgeführt und die OVERFLOW-Bedingung wird nicht festgestellt.

Beispiele folgen ab der nächsten Seite:

Beispiel 1:

```
MOVE 1 TO ZEICHEN-ZAHL.
MOVE ZERO TO FELD-ZAHL.
UNSTRING FELD-A INTO FELD-B, FELD-C
     WITH POINTER ZEICHEN-ZAHL TALLYING IN FELD-ZAHL.
```

FELD-A vor Ausführung	A B C D E F G	
FELD-B vor Ausführung	0 0 9 8	
FELD-C vor Ausführung	J P R	
ZEICHEN-ZAHL vor Ausführung	0 1	
FELD-ZAHL vor Ausführung	0 0	
FELD-A nach Ausführung	A B C D E F G	unverändert
FELD-B nach Ausführung	A B C D	enthält ABCD von FELD-A
FELD-C nach Ausführung	E F G	enthält EFG von FELD-A
ZEICHEN-ZAHL nach Ausführung	0 8	enthält die nächste in FELD-A zu verarbeitende Zeichenposition
FELD-ZAHL nach Ausführung	0 2	enthält die Anzahl der Daten empfangenden Datenfelder

Beispiel 2:

```
MOVE 1 TO ZEICHEN-ZAHL.
MOVE ZERO TO FELD-ZAHL.
UNSTRING FELD-D INTO FELD-E, FELD-F
   WITH POINTER ZEICHEN-ZAHL   TALLYING IN FELD-ZAHL.
```

FELD-D vor Ausführung	1 2 3 4 5	
FELD-E vor Ausführung	A B C D	
FELD-F vor Ausführung	X Y Z	
ZEICHEN-ZAHL vor Ausführung	0 1	
FELD-ZAHL vor Ausführung	0 0	
FELD-D nach Ausführung	1 2 3 4 5	unverändert
FELD-E nach Ausführung	1 2 3 4	enthält 1234 von FELD-D
FELD-F nach Ausführung	5 ¢ ¢	enthält 5 von FELD-D
vorausgesetzt, FELD-F ist ein alphanumerisches Feld		
ZEICHEN-ZAHL nach Ausführung	0 6	enthält die nächste in FELD-D zu verarbeitende Zeichenposition
FELD-ZAHL nach Ausführung	0 2	enthält die Anzahl der Daten empfangenden Datenfelder

Beispiel 3:

```
MOVE 1 TO ZEICHEN-ZAHL.
MOVE ZERO TO FELD-ZAHL.
UNSTRING FELD-D INTO FELD-E, FELD-F
   WITH POINTER ZEICHEN-ZAHL   TALLYING IN FELD-ZAHL.
```

FELD-D vor Ausführung	1 2 3 4 5	
FELD-E vor Ausführung	A B C D	
FELD-F vor Ausführung	6 0 0	
ZEICHEN-ZAHL vor Ausführung	0 1	
FELD-ZAHL vor Ausführung	0 0	
FELD-D nach Ausführung	1 2 3 4 5	unverändert
FELD-E nach Ausführung	1 2 3 4	enthält 1234 von FELD-D
FELD-F nach Ausführung; vorausgesetzt, FELD-F ist ein numerisches Datenfeld	0 0 5	enthält 5 von FELD-D
ZEICHEN-ZAHL nach Ausführung	0 6	enthält die nächste in FELD-D zu verarbeitende Zeichenposition
FELD-ZAHL nach Ausführung	0 2	enthält die Anzahl der Daten empfangenden Datenfelder

Beispiel 4:

```
MOVE 1 TO ZEICHEN-ZAHL.
MOVE ZERO TO FELD-ZAHL.
UNSTRING FELD-G INTO FELD-H, FELD-I
  WITH POINTER ZEICHEN-ZAHL   TALLYING IN FELD-ZAHL
        ON OVERFLOW GO TO UEBERLAUF.
```

FELD-G vor Ausführung	A B C E F G H M N	
FELD-H vor Ausführung	H 4 7	
FELD-I vor Ausführung	1 2 3 4	
ZEICHEN-ZAHL vor Ausführung	0 0 1	
FELD-ZAHL vor Ausführung	0	
FELD-G nach Ausführung	A B C E F G H M N	unverändert
FELD-H nach Ausführung	A B C	enthält ABC von FELD-G
FELD-I nach Ausführung	E F G H	enthält EFGH von FELD-G
ZEICHEN-ZAHL nach Ausführung	0 0 8	enthält die nächste in FELD-G zu verarbeitende Zeichenposition
FELD-ZAHL nach Ausführung	2	enthält die Anzahl der Daten empfangenden Datenfelder

Nach der UNSTRING-Anweisung wird die Prozedur UEBERLAUF ausgeführt, weil infolge der Zeichen MN, die im Sendefeld (FELD-G) verblieben, nachdem Daten in alle Empfangsfelder übertragen wurden, eine OVERFLOW-Bedingung auftritt.

Beispiel 5:

```
MOVE 1 TO ZEICHEN-ZAHL.
MOVE ZERO TO FELD-ZAHL.
UNSTRING FELD-J DELIMITED  BY "Z"
        INTO FELD-K, FELD-L, FELD-M
        WITH POINTER ZEICHEN-ZAHL
        TALLYING IN FELD-ZAHL.
```

FELD-J vor Ausführung	1 2 Z 3 4 5 6 Z 7 8 9 A Z	
FELD-K vor Ausführung	8 7 6 .	
FELD-L vor Ausführung	0 0 0 0	
FELD-M vor Ausführung	M M M M M M M	
ZEICHEN-ZAHL vor Ausführung	0 1	
FELD-ZAHL vor Ausführung	0	
FELD-J nach Ausführung	12Z3456Z789AZ	unverändert
FELD-K nach Ausführung; vorausgesetzt, FELD-K ist ein numerisches Datenfeld	0 1 2	enthält 12 von FELD-J
FELD-L nach Ausführung	3 4 5 6	enthält 3456 von FELD-J
FELD-M nach Ausführung; vorausgesetzt, FELD-M ist ein alphanumerisches Datenfeld	7 8 9 A ¢ ¢ ¢	enthält 789A von FELD-J
ZEICHEN-ZAHL nach Ausführung	1 4	enthält die nächste in FELD-J zu verarbeitende Zeichenposition
FELD-ZAHL nach Ausführung	3	enthält die Anzahl der Daten empfangenden Datenfelder

Beispiel 6:

```
MOVE ZERO TO FELD-ZAHL.
UNSTRING FELD-N DELIMITED  BY "A"
    INTO FELD-O COUNT IN ZAHL-O,
         FELD-P COUNT IN ZAHL-P,
         FELD-Q COUNT IN ZAHL-Q,
    TALLYING IN FELD-ZAHL.
```

FELD-N vor Ausführung	4 7 A A 8 9 A	
FELD-O vor Ausführung	Z Z Z	
ZAHL-O vor Ausführung	7	
FELD-P vor Ausführung	A 2 3 4	
ZAHL-P vor Ausführung	9	
FELD-Q vor Ausführung	9 8 7 6	
ZAHL-Q vor Ausführung	5	
FELD-ZAHL vor Ausführung	0	
FELD-N nach Ausführung	4 7 A A 8 9 A	unverändert
FELD-O nach Ausführung; vorausgesetzt, alphanum.	4 7 ∅	enthält 47 von FELD-N
ZAHL-O nach Ausführung	2	enthält die Anzahl der von FELD-N ins FELD-O übertragenen Zeichen
FELD-P nach Ausführung; vorausgesetzt, FELD-P ist alphanumerisch	∅ ∅ ∅ ∅	enthält Leerzeichen, da in FELD-N Trennsymbol A folgte
ZAHL-P nach Ausführung	0	enthält die Anzahl der von FELD-N ins FELD-P übertragenen Zeichen
FELD-Q nach Ausführung; vorausgesetzt, numerisch	0 0 8 9	enthält 89 von FELD-N
ZAHL-Q nach Ausführung	2	enthält die Anzahl der von FELD-N ins FELD-Q übertragenen Zeichen
FELD-ZAHL nach Ausführung	3	enthält die Anzahl der Empfangsfelder

Beispiel 7:

```
MOVE ZERO TO FELD-ZAHL.
UNSTRING FELD-N  DELIMITED  BY ALL "A"
    INTO FELD-O COUNT IN ZAHL-O,
         FELD-P COUNT IN ZAHL-P,
         FELD-Q COUNT IN ZAHL-Q,
    TALLYING IN FELD-ZAHL.
```

FELD-N vor Ausführung	4 7 A A 8 9 A	
FELD-O vor Ausführung	Z Z Z	
ZAHL-O vor Ausführung	7	
FELD-P vor Ausführung	A 2 3 4	
ZAHL-P vor Ausführung	9	
FELD-Q vor Ausführung	9 8 7 6	
ZAHL-Q vor Ausführung	5	
FELD-ZAHL vor Ausführung	0	
FELD-N nach Ausführung	4 7 A A 8 9 A	unverändert
FELD-O nach Ausführung; vorausgesetzt, alphanum.	4 7 ϕ	enthält 47 von FELD-N
ZAHL-O nach Ausführung	2	enthält die Anzahl der von FELD-N ins FELD-O übertragenen Zeichen
FELD-P nach Ausführung; vorausgesetzt, alphanum.	8 9 ϕ ϕ	enthält 89 von FELD-N
ZAHL-P nach Ausführung	2	enthält die Anzahl der von FELD-N ins FELD-P übertragenen Zeichen
FELD-Q nach Ausführung	9 8 7 6	unverändert
ZAHL-Q nach Ausführung	5	unverändert
FELD-ZAHL nach Ausführung	2	enthält die Anzahl der Daten empfangenden Datenfelder

Beispiel 8:

```
MOVE ZERO TO FELD-ZAHL.
UNSTRING FELD-R  DELIMITED  BY "X" OR "Z"
    INTO FELD-S  DELIMITER  IN SPEICHER-S COUNT IN ZAHL-S,
         FELD-T  DELIMITER  IN SPEICHER-T COUNT IN ZAHL-T,
         FELD-U  DELIMITER  IN SPEICHER-U COUNT IN ZAHL-U,
    TALLYING IN FELD-ZAHL.
```

FELD-R vor Ausführung	A B X 1 Z J M K P X	
FELD-S vor Ausführung	4 6 A	
SPEICHER-S vor Ausführung	B	
ZAHL-S vor Ausführung	9	
FELD-T vor Ausführung	3 4 5	
SPEICHER-T vor Ausführung	K	
ZAHL-T vor Ausführung	4	
FELD-U vor Ausführung	A A A	
SPEICHER-U vor Ausführung	J	
ZAHL-U vor Ausführung	P	
FELD-ZAHL vor Ausführung	0 0	
FELD-R nach Ausführung	A B X 1 Z J M K P X	unverändert
FELD-S nach Ausführung; vorausgesetzt, alphanum.	A B ¢	enthält AB von FELD-R
SPEICHER-S nach Ausführung	X	enthält das Trennsymbol X, das die Zeichenübertragung in FELD-S beendete
ZAHL-S nach Ausführung	2	enthält die Anzahl der zwischen Trennsymbolen isolierten Zeichen
FELD-T nach Ausführung; vorausgesetzt, numerisch	0 0 1	enthält 1 von FELD-R

SPEICHER-T nach Ausführung	Z	enthält das Trennsymbol Z, das die Zeichenübertragung ins FELD-T beendete
ZAHL-T nach Ausführung	1	enthält die Anzahl der zwischen Trennsymbolen isolierten Zeichen
FELD-U nach Ausführung; vorausgesetzt, FELD-U ist ein alphanumerisches Datenfeld	J M K	enthält JMK von FELD-R
SPEICHER-U nach Ausführung	X	enthält das Trennsymbol X, das die Zeichenprüfung zur Übertragung zum FELD-U beendete
ZAHL-U nach Ausführung	4	enthält die Anzahl von Zeichen, die durch Trennsymbole isoliert sind
FELD-ZAHL nach Ausführung	3	enthält die Anzahl der Daten empfangenden Datenfelder

Beispiel 9:

```
UNSTRING FELD-U INTO FELD-V, FELD-W, FELD-X.
```

FELD-U vor Ausführung	0 1 2 3 A 7 B 9	
FELD-V vor Ausführung	A B C	
FELD-W vor Ausführung	0 9 8 7	
FELD-X vor Ausführung	Z	
FELD-U nach Ausführung	0 1 2 3 A 7 B 9	unverändert
FELD-V nach Ausführung	0 1 2	enthält 012 von FELD-U
FELD-W nach Ausführung	3 A 7 B	enthält 3A7B von FELD-U
FELD-X nach Ausführung	9	enthält 9 von FELD-U

Beispiel 10:

```
MOVE 5 TO POSITION-FELD.
UNSTRING FELD-Y INTO FELD-Z, FELD-A
    WITH POINTER POSITION-FELD.
```

FELD-Y vor Ausführung	A B C D E F G H I J K	
FELD-Z vor Ausführung	1 2 3 4	
FELD-A vor Ausführung	9 8 7	
POSITION-FELD vor Ausführung	0 5	
FELD-Y nach Ausführung	A B C D E F G H I J K	unverändert
FELD-Z nach Ausführung	E F G H	enthält EFGH von FELD-Y
FELD-A nach Ausführung	I J K	enthält IJK von FELD-Y
POSITION-FELD nach Ausführung	1 2	enthält die nächste in FELD-Y verarbeitbare Zeichenposition

DIE USE FOR DEBUGGING Anweisung

Diese Anweisung dient zur Identifikation der Benutzerfelder, die durch die entsprechende Debugging-(Testhilfe-)Section geführt werden.

Das Debug-Modul ist nicht in ITX COBOL 85 aufgenommen worden. Verwenden Sie zum Testen Ihrer Programme den ITX Symbolic Debugger, der in einem eigenen Manual beschrieben ist.

```
USE FOR DEBUGGING { CD-Name                                  }
                  { [ALL REFERENCES OF] Identifier           } ...
                  { ON Dateiname                             }
                  { Prozedurname                             }
                  { ALL PROCEDURES                           }
```

DIE WRITE-ANWEISUNG (Sequentielle Datei)

Durch die WRITE-Anweisung wird ein Datensatz auf dem Drucker oder in eine auf einem magnetischen Datenträger gespeicherte sequentielle Datei ausgegeben.

Format:

```
WRITE        Satz-Name     [FROM {Literal-1   }]
                           [     {Identifier-1}]

[{BEFORE}                  {{Identifier-2}      [LINE ]}]
[{AFTER }  ADVANCING       {{Zahl       }      [LINES]}]
[                          {{mnemonischer-Name}       }]
                           {{PAGE             }       }

[AT          {END-OF-PAGE}  unbedingte Anweisung 1]
             {EOP        }

[NOT AT      {END-OF-PAGE}  unbedingte Anweisung 2]
             {EOP        }

[END-WRITE]
```

Der Satz-Name ist der Name eines Datensatzes in der FILE SECTION der DATA DIVISION. In der WRITE-Anweisung kann ein Satz-Name durch einen Datei-Namen qualifiziert werden.

Der in der WRITE-Anweisung angegebene Satz-Name muß einem der Satz-Namen entsprechen, die in den auf die Dateibeschreibung folgenden Satzbeschreibungen vorkommen. Der FILE-CONTROL-Eintrag muß indirekt oder ausdrücklich die Klauseln ORGANIZATION IS SEQUENTIAL und ACCESS MODE IS SEQUENTIAL aufweisen.

Eine Druckdatei muß Datensätze fester Länge enthalten. Eine gespeicherte sequentielle Datei kann Sätze fester oder variabler Länge aufweisen. Besteht eine sequentielle Datei aus Sätzen variabler Länge, dürfen die RECORD CONTAINS-Klausel und die Satzbeschreibungen nicht die von dem Variable-Längen-Indikator (VLI) eingenommenen Bytes beinhalten. Bei COBOL bestimmt der Satz-Name in der WRITE-Anweisung die Länge der Satz-Ausgabe für eine sequentielle Datei mit Sätzen variabler Länge.

Vor Ausführung der WRITE-Anweisung muß die Datei durch eine OPEN OUTPUT- oder OPEN EXTEND-Anweisung eröffnet sein.

Die erste WRITE-Anweisung schreibt den ersten Datensatz auf die sequentielle Datei. Nachfolgende Ausführungen der WRITE-Anweisung für dieselbe Datei schreiben den sequentiell nächsten Datensatz auf die Datei.

Der durch die korrekte Ausführung der WRITE-Anweisung geschriebene Datensatz steht danach nicht mehr in dem dem Satz-Namen zugeordneten Satz-Bereich; mit einer Ausnahme: Ist die Datei in einer SAME RECORD AREA-Klausel genannt, ist der Datensatz als ein Datensatz anderer in dieser SAME RECORD AREA-Klausel genannter Dateien für das Programm verfügbar. Der Datensatz steht auch nach erfolgloser Ausführung der WRITE-Anweisung infolge eines Dateiüberlaufs im Satz-Bereich weiter zur Verfügung.

FROM-Angabe

Ist FROM angegeben, findet eine Datenübertragung aus dem durch Identifier bezeichneten Bereich in den durch Satz-Name bezeichneten Bereich statt. Diese Übertragung erfolgt nach den für die MOVE-Anweisung bestimmten Regeln. Satz-Name und Identifier dürfen sich nicht auf denselben Speicherbereich beziehen.

Der Inhalt des Identifiers ist nach Ausführung der WRITE-Anweisung immer noch vorhanden.

ADVANCING-Klausel

Die ADVANCING-Klausel steuert die vertikale Positionierung der durch die WRITE-Anweisung vorgesehenen Zeilenausgabe in einer Druck-Datei. Wird das Wort BEFORE in der ADVANCING-Klausel angegeben, wird vor dem Vorschub gedruckt; bei der Angabe des Wortes AFTER in der ADVANCING-Klausel wird die Zeile nach dem Vorschub gedruckt.

Die Klausel AFTER ADVANCING Zahl LINES verursacht, daß der Vorschub des Papiers um die dem Wert der Zahl entsprechende Anzahl von Zeilen erfolgt. Zahl ist ein numerisches Literal, das eine Ganzzahl ohne Vorzeichen sein muß. Diese Zahl darf einen Nullwert aufweisen (Drucken auf der Stelle). Enthält die Dateibeschreibung für die Druck-Datei die LINAGE-Klausel, wird das Register LINAGE-COUNTER während der WRITE-

Anweisung um die Zahl erhöht. Angenommen, das Papier ist auf Zeile 36 positioniert, dann bewirkt die WRITE-Anweisung im nachfolgenden Beispiel 1 den Vorschub des Papiers auf Zeile 39 und den Druck des Inhalts von DRUCKSATZ auf Zeile 39.

Beispiel 1:

```
WRITE DRUCKSATZ AFTER ADVANCING 3 LINES.
```

Angenommen, das Papier ist auf Zeile 16 positioniert, dann bewirkt die WRITE-Anweisung im nachfolgenden Beispiel 2, daß der Inhalt von DRUCKSATZ ohne Papiervorschub auf Zeile 16 gedruckt wird.

Beispiel 2:

```
WRITE DRUCKSATZ AFTER ADVANCING  0  LINES.
```

Die Klausel BEFORE ADVANCING Zahl LINES bewirkt, daß ein Datensatz gedruckt wird und daraufhin Papiervorschub um die entsprechende Zeilenanzahl erfolgt. Angenommen, das Papier ist auf Zeile 27 positioniert, dann bewirkt die WRITE-Anweisung im nachfolgenden Beispiel 3, daß der Inhalt von DRUCKSATZ auf Zeile 27 gedruckt wird und daraufhin Papiervorschub zur Zeile 29 erfolgt.

Beispiel 3:

```
WRITE DRUCKSATZ BEFORE ADVANCING 2 LINES.
```

Die Klausel AFTER ADVANCING Identifier-2 LINES bewirkt Papiervorschub um die entsprechende Anzahl von Zeilen. Daraufhin wird gedruckt. Identifier-2 muß ein ganzzahliges Elementarfeld sein. Der im Identifier-2 enthaltene Wert darf Null sein.

Enthält die Dateibeschreibung für die Druck-Datei die LINAGE-Klausel, wird der LINAGE-COUNTER während der Ausführung der WRITE-Anweisung um den Wert des Identifiers-2 erhöht. Angenommen, das Papier ist auf Zeile 24 positioniert, dann bewirkt die WRITE-Anweisung im nachfolgenden Beispiel 4 den Vorschub des Papiers auf Zeile 29 und anschließend den Druck auf Zeile 29.

Beispiel 4:

```
DATA DIVISION.
01  VORSCHUB     PIC 99.

PROCEDURE DIVISION.
    MOVE 5 TO VORSCHUB.
    WRITE DRUCKSATZ AFTER ADVANCING VORSCHUB.
```

Die Klausel AFTER ADVANCING mnemonischer-Name schiebt das Papier auf die im SPECIAL-NAMES-Paragraphen der ENVIRONMENT DIVISION für den spezifizierten mnemonischen Namen bestimmte Zeile vor. Der Implementor-Name LINE-n ist durch die Implementor-Namen-Klausel im SPECIAL-NAMES-Paragraphen einem mnemonischen Namen gleichgesetzt. Der mit n bezeichnete Wert reicht von 1 bis 256 und spezifiziert die Zeile, auf die durch die Klausel AFTER ADVANCING mnemonischer Name der Papiervorschub erfolgen soll. Nach erfolgtem Papiervorschub wird gedruckt. Wird die Klausel AFTER ADVANCING mnemonischer-Name verwendet, darf die Dateibeschreibung keine LINAGE-Klausel enthalten. Angenommen, das Papier ist auf Zeile 10 positioniert, dann bewirkt die WRITE-Anweisung im nachfolgenden Beispiel 5 den Vorschub des Papiers auf Zeile 52 und den Druck des Inhalts von SUMMENSATZ auf Zeile 52.

Beispiel 5:

```
SPECIAL-NAMES.
    LINE-52 IS SUMMEN-ZEILE.

PROCEDURE DIVISION.
    WRITE SUMMENSATZ AFTER ADVANCING SUMMEN-ZEILE.
```

Die Klausel AFTER ADVANCING PAGE bewirkt den Papiervorschub auf die nächste Seite und nachfolgend den Druck. Enthält die Dateibeschreibung eine LINAGE-Klausel, dann erfolgt der Papiervorschub auf die erste logische Zeile der nächsten Seite und der LINAGE-COUNTER wird auf 1 zurückgestellt. Enthält die Dateibeschreibung keine LINAGE-Klausel, dann erfolgt der Papiervorschub auf eine bestimmte Zeile der nächsten Seite. Diese bestimmte Zeile ergibt sich aus der Einstellung des Zeileneinstellrades am Drucker.

Das nachfolgende Beispiel 6 zeigt die Klausel AFTER ADVANCING PAGE bei einer Druck-Datei, in deren Dateibeschreibung keine LINAGE-Klausel vorgesehen ist. Angenommen, das Papier ist auf Zeile 55 positioniert und der physische Seitenanfang auf Zeile 3 vorgesehen, dann bewirkt die WRITE-Anweisung in Beispiel 6 den Papiervorschub bis zu Zeile 3 der nächsten Seite und daraufhin den Druck von DRUCKSATZ auf dieser Zeile.

Beispiel 6:

```
WRITE DRUCKSATZ AFTER ADVANCING PAGE.
```

Das nachfolgende Beispiel 7 zeigt die Klausel AFTER ADVANCING PAGE in Anwendung auf eine Druck-Datei, in deren Dateibeschreibung eine LINAGE-Klausel vorgesehen ist. Die LINAGE-Klausel beschreibt eine logische Seite mit folgenden Dimensionen:

- Der Kopf-Bereich umfaßt 3 Zeilen, in die nicht gedruckt wird; somit bilden die physischen Zeilen 1, 2 und 3 den Kopf-Bereich der Seite

- Die logische Seite (Seitenrumpf) umfaßt 59 Zeilen, auf die gedruckt werden kann. Die erste Zeile des Seitenrumpfes hat die logische Zeilennummer 1; die letzte Zeile des Seitenrumpfes hat die logische Zeilennummer 59. Die erste Zeile des Seitenrumpfes entspricht der physischen Zeile 4; die letzte Zeile des Seitenrumpfes entspricht der physischen Zeile 62.

- Der Fußzeilen-Bereich umfaßt vier Zeilen, in die nicht gedruckt wird; folglich bilden die physischen Zeilen 63, 64, 65 und 66 den Fußzeilen-Bereich der Seite.

Angenommen, das Papier ist auf der physischen Zeile 62 (logische Zeilennummer 59) positioniert, dann bewirkt die WRITE-Anweisung im nachfolgenden Beispiel 7 einen Papiervorschub auf die physische Zeile 4 (logische Zeilennummer 1) der nächsten Seite. Danach wird der Inhalt des aus KOPF-1 bezogenen DRUCKSATZ auf der physischen Zeile 4 (logische Zeilennummer 1) gedruckt. Durch die Ausführung dieser WRITE-Anweisung mit der Klausel AFTER ADVANCING PAGE wird der LINAGE-COUNTER auf den Wert 1 zurückgestellt.

Beispiel 7:

```
DATA DIVISION.
FD  DRUCKDATEI   BLOCK CONTAINS 1 RECORDS
                 RECORD CONTAINS 132 CHARACTERS
                 LABEL RECORD IS OMITTED
                 LINAGE IS 59 LINES, LINES AT TOP 3, LINES
                                     AT BOTTOM 4.

01  DRUCKSATZ   PIC X(132).

PROCEDURE DIVISION.
    WRITE DRUCKSATZ FROM KOPF-1 AFTER ADVANCING PAGE.
```

Die Klausel BEFORE ADVANCING PAGE verursacht, daß ein Datensatz zuerst gedruckt wird und daraufhin der Papiervorschub auf die nächste Seite erfolgt.

Bei Weglassen der ADVANCING-Klausel in einer WRITE-Anweisung für eine Druck-Datei erfolgt ein automatischer Papiervorschub, als ob der Programmierer die Klausel AFTER ADVANCING 1 LINE spezifiziert hätte. Enthält die Dateibeschreibung für die Drucker-Datei die LINAGE-Klausel, dann wird der LINAGE-COUNTER während der Ausführung der WRITE-Anweisung um den Wert 1 erhöht. Angenommen, das Papier ist auf Zeile 40 positioniert, dann bewirkt die WRITE-Anweisung im nachfolgenden Beispiel 8 den Vorschub des Papiers auf Zeile 41 und den Druck auf Zeile 41.

Beispiel 8:

```
WRITE DRUCKSATZ.
```

END-OF-PAGE-Klausel (EOP-Klausel)

Die EOP-Klausel kann in der WRITE-Anweisung einer Druck-Datei nur spezifiziert werden, wenn in der Dateibeschreibung eine LINAGE-Klausel vorhanden ist. Die Steuerung geht auf die unbedingte Anweisung innerhalb der EOP-Klausel über, wenn entweder eine END-OF-PAGE-Bedingung oder eine PAGE-OVERFLOW-Bedingung erfüllt ist.

Eine END-OF-PAGE-Bedingung ist immer dann erfüllt, wenn die Ausführung einer WRITE-Anweisung mit einer EOP-Klausel einen Druck oder einen Papiervorschub im Fußzeilen-Bereich bewirkt. Die FOOTING-Angabe in der LINAGE-Klausel bestimmt die logische Zeilennummer im Seitenrumpf, bei der der Fußzeilen-Bereich beginnt. Wird die FOOTING-Angabe nicht spezifiziert, dann wird als Beginn des Fußzeilen-Bereiches die letzte Zeile des Seitenrumpfes angenommen. Die END-OF-PAGE-Bedingung ist erfüllt, wenn nach Ausführung einer WRITE-Anweisung vom LINAGE-COUNTER die in der FOOTING-Angabe als Beginn des Fußzeilen-Bereiches bestimmte Zeile erreicht oder überschritten wird. Ist die FOOTING-Angabe nicht vorgesehen, dann ist die END-OF-PAGE-Bedingung erfüllt, wenn nach Ausführung einer WRITE-Anweisung vom LINAGE-COUNTER die letzte Zeile des Seitenrumpfes erreicht wird. Bei erfüllter END-OF-PAGE-Bedingung werden während der Ausführung der (die BEFORE ADVANCING-Klausel enthaltenden) WRITE-Anweisung die folgenden Aktionen durchgeführt:

1. Der Datensatz wird auf der Zeile gedruckt, auf der das Papier gerade positioniert ist.

2. Das Papier wird danach gemäß der in der WRITE-Anweisung enthaltenden Klausel BEFORE ADVANCING vorgeschoben. Der LINAGE-COUNTER wird gemäß dem Operanden in der Klausel BEFORE ADVANCING erhöht, der sich ergebende Wert kommt dem Beginn des Fußzeilen-Bereiches gleich oder überschreitet diesen.

3. Die Steuerung wird daraufhin auf die unbedingte Anweiweisung END-OF-PAGE übertragen.

Bei erfüllter END-OF-PAGE-Bedingung werden während der Ausführung der (die AFTER ADVANCING-Klausel enthaltenden) WRITE-Anweisung die folgenden Aktionen durchgeführt:

1. Das Papier wird zunächst gemäß der in der WRITE-Anweisung enthaltenen Klausel AFTER ADVANCING vorgeschoben. Der LINAGE-COUNTER wird gemäß dem Operanden in der Klausel AFTER ADVANCING erhöht; der sich ergebende Wert kommt dem Beginn des Fußzeilen-Bereiches gleich oder überschreitet diesen.

2. Der Datensatz wird noch auf der Zeile gedruckt, auf der das Papier durch die erste Aktion positioniert wurde.

3. Die Steuerung wird daraufhin auf die unbedingte Anweisung END-OF-PAGE übertragen.

Eine automatische PAGE-OVERFLOW-Bedingung ist immer dann erfüllt, wenn der Ausführung einer WRITE-Anweisung (mit oder ohne die Angabe END-OF-PAGE) im Seitenrumpf nicht voll entsprochen werden kann. Diese Situation ist dann gegeben, wenn bei Ausführung einer WRITE-Anweisung vom LINAGE-COUNTER die Zeilennummer des Seitenrumpfes überschritten wird. Bei erfüllter automatischer PAGE-OVERFLOW-Bedingung werden während der Ausführung der (die BEFORE ADVANCING-Klausel enthaltenden) WRITE-Anweisung die folgenden Aktionen durchgeführt:

1. Der Datensatz wird auf der Zeile gedruckt, auf der das Papier gerade positioniert ist.

2. Es erfolgt daraufhin automatisch ein Papiervorschub auf die erste Zeile des Seitenrumpfes der nächsten Seite. Der LINAGE-COUNTER wird auf den Wert 1 zurückgestellt.

3. Die Steuerung wird danach auf die unbedingte Anweisung END-OF-PAGE oder, wenn die WRITE-Anweisung keine EOP-Klausel enthält, auf die nächste auszuführende Anweisung übertragen.

Bei erfüllter automatischer PAGE-OVERFLOW-Bedingung werden während der Ausführung der (die AFTER ADVANCING-Klausel enthaltenden) WRITE-Anweisung die folgenden Aktionen durchgeführt:

1. Es erfolgt zunächst automatisch ein Papiervorschub auf die erste Zeile des Seitenrumpfes der nächsten Seite. Der LINAGE-COUNTER wird auf den Wert 1 zurückgestellt.

2. Der Datensatz wird daraufhin auf der ersten Zeile des Seitenrumpfes der nächsten Seite gedruckt.

3. Die Steuerung wird danach auf die unbedingte Anweisung END-OF-PAGE oder, wenn die WRITE-Anweisung keine EOP-Klausel enthält, auf die nächste auszuführende Anweisung übertragen.

Wenn die WRITE-Anweisung für den LINAGE-COUNTER bedeutet, daß er sowohl den dem Beginn des Fußzeilen-Bereiches als auch den dem Ende des Seitenrumpfes entsprechenden Wert überschreitet, dann treten die END-OF-PAGE-Bedingung und die PAGE-OVERFLOW-Bedingung gleichzeitig auf. In diesem Fall werden die der PAGE-OVERFLOW-Bedingung zugeordneten Aktionen durchgeführt, wie vorab beschrieben.

Beispiel 1:

```
FILE-CONTROL.
    SELECT DATEI-Y  ASSIGN TO PRINTER
           ORGANIZATION IS SEQUENTIAL
           ACCESS MODE IS SEQUENTIAL.

DATA DIVISION.
FILE SECTION.
FD  DATEI-Y BLOCK CONTAINS 1 RECORDS
            RECORD CONTAINS 132 CHARACTERS
            LABEL RECORD IS OMITTED.
01  SATZ-Y  PIC X(132).

PROCEDURE DIVISION.
ANFANG.
    OPEN OUTPUT DATEI-Y.
       -------------
    WRITE SATZ-Y AFTER ADVANCING 1 LINE.
       -------------
    CLOSE DATEI-Y.
```

Im Beispiel 1 handelt es sich um eine Druck-Datei (eine sequentielle Datei mit sequentiellem Zugriff). Die Datei hat den Namen DATEI-Y. Da die Dateibeschreibung für DATEI-Y keine LINAGE-Klausel enthält, muß der Programmierer zur Steuerung der Anzahl der auf jede Seite gedruckten Zeilen eine Zeilenzählung vorsehen. Damit gedruckt werden kann, muß die Datei zuerst mit der OPEN OUTPUT-Anweisung eröffnet werden.

Die WRITE-Anweisung gibt sequentiell einen Datensatz aus SATZ-Y auf die Druck-Datei DATEI-Y aus: Es erfolgt zunächst ein Papiervorschub um eine Zeile, und daraufhin wird der Datensatz auf der Zeile gedruckt, auf der das Papier nun positioniert ist.

Beispiel 2:

```
FILE-CONTROL.
    SELECT DATEI-Z ASSIGN TO PRINTER
           ORGANIZATION IS SEQUENTIAL
           ACCESS MODE IS SEQUENTIAL.
DATA DIVISION.
FILE SECTION.
FD  DATEI-Z BLOCK CONTAINS 1 RECORDS
            RECORD CONTAINS 132 CHARACTERS
            LABEL RECORD IS OMITTED
            LINAGE IS 59 LINES, LINES AT TOP 3,  BOTTOM 4.
01  SATZ-Z  PIC X(132).
PROCEDURE DIVISION.
ANFANG.
    OPEN OUTPUT DATEI-Z.
       ------------
    WRITE SATZ-Z AFTER ADVANCING 1 LINE
          AT END OF PAGE PERFORM TOP-PAGE.
       ------------
TOP-PAGE.
    WRITE SATZ-Z FROM KOPF-1 AFTER ADVANCING PAGE.
```

Die Druck-Datei hat den Namen DATEI-Z. Durch die LINAGE-Klausel in der Dateibeschreibung wird das Sonderregister LINAGE-COUNTER durch den Compiler generiert. Die Anzahl von Zeilen, die auf einer Seite gedruckt werden, wird durch die Zeilenzählung, die im LINAGE-COUNTER erfolgt, automatisch gesteuert. Die LINAGE-Klausel definiert eine aus 66 Zeilen bestehende logische Seite. Dabei bilden die physischen Zeilen 1, 2 und 3 den Kopf-Bereich, die physischen Zeilen 4 bis einschließlich 62 sind die den Seitenrumpf bildenden 59 Zeilen, und die physischen Zeilen 63, 64, 65 und 66 bilden den Fußzeilen-Bereich der Seite.

Die erste WRITE-Anweisung gibt sequentiell einen Datensatz aus SATZ-Z auf die Druck-Datei DATEI-Z frei. Es erfolgt ein Papiervorschub um eine Zeile, und daraufhin wird der Datensatz auf der Zeile gedruckt, auf der das Papier positioniert ist. Bei jeder Ausführung der WRITE-Anweisung wird der LINAGE-COUNTER um 1 erhöht. Wenn durch die WRITE-Anweisung ein Vorschub des Papiers auf die logische Zeile 59 (physische Zeile 62) erfolgt und eine Einzelzeile auf diese Zeile gedruckt wird, ist die END-OF-PAGE-Bedingung erfüllt, denn der Wert des LINAGE-COUNTER kommt der letzten Zeile des Seitenrumpfes gleich. Ist die END-OF-PAGE-Bedingung erfüllt, wird die Steuerung auf die PERFORM-Anweisung innerhalb der END-OF-PAGE-Klausel übertragen.

Durch die Anweisung WRITE AFTER ADVANCING PAGE erfolgt der Vorschub des Papiers auf die erste Zeile des Seitenrumpfes der nächsten Seite und der LINAGE-COUNTER wird auf 1 zurückgestellt. Nachdem das Papier auf die logische Zeile 1 (physische Zeile 4) der neuen Seite vorgeschoben wurde, werden die Daten aus KOPF-1 auf der logischen Zeile 1 (physische Zeile 4) gedruckt.

Ein drittes Beispiel finden Sie auf der nächsten Seite:

Beispiel 3:

```
FILE-CONTROL.
    SELECT DATEI-B  ASSIGN TO MAGNETIC-TAPE
           ORGANIZATION IS SEQUENTIAL
           ACCESS MODE IS SEQUENTIAL.
DATA DIVISION.
FILE SECTION.
FD  DATEI-B BLOCK CONTAINS 60 RECORDS
           RECORD CONTAINS 40 CHARACTERS
           LABEL RECORD IS STANDARD.
01  SATZ-B  PIC X(40).

PROCEDURE DIVISION.
ANFANG.
    OPEN OUTPUT DATEI-B.
       ------------
A.     ------------
    WRITE SATZ-B.
       ------------
       ------------
```

Wenn eine Magnetband-Datei erstellt wird, werden die Datensätze der Datei vom Anfang bis zum Ende sequentiell geschrieben. Das Beispiel zeigt die Erstellung einer Magnetband-Datei. DATEI-B ist eine sequentielle Datei mit sequentiellem Zugriff.

Damit Daten auf DATEI-B geschrieben werden können, muß die Datei zuerst im OUTPUT-Modus eröffnet werden. Die WRITE-Anweisung nimmt sequentiellen Zugriff auf DATEI-B, um den in SATZ-B befindlichen Datensatz auf die DATEI zu schreiben. Dies kann durch Bildung einer Programmschleife beliebig oft wiederholt werden.

Beispiel 4:

```
FILE-CONTROL.
    SELECT DATEI-C ASSIGN TO DISK
           ORGANIZATION IS SEQUENTIAL
           ACCESS MODE IS SEQUENTIAL.
DATA DIVISION.
FILE SECTION.
FD  DATEI-C BLOCK CONTAINS 50 RECORDS
           RECORD CONTAINS 40 CHARACTERS
           LABEL RECORD IS STANDARD.
01  SATZ-C  PIC X(40).

PROCEDURE DIVISION.
ANFANG.
    OPEN EXTEND DATEI-C.
       ------------
A.     ------------
    WRITE SATZ-C.
       ------------
       ------------
```

Die COBOL-Programmiersprache ermöglicht es, daß im Anschluß an bestehende Datensätze einer sequentiellen Datei neue Datensätze sequentiell angefügt werden. Dies gilt für Dateien auf Magnetplatte oder Magnetband. Aus dem Beispiel 4 ist das Hinzufügen von Datensätzen an das Ende einer Magnetplatten-Datei ersichtlich. DATEI-C ist eine sequentielle Datei mit sequentiellem Zugriff.

Die Datei muß zuerst im EXTEND-Modus eröffnet werden. Die OPEN EXTEND-Anweisung positioniert die Datei unmittelbar nach ihrem letzten Datensatz.

Die WRITE-Anweisung fügt Datensätze sequentiell der Datei hinzu, als ob diese mit der OPEN OUTPUT-Anweisung eröffnet worden wäre.

Beispiel 5:

```
FILE-CONTROL.
    SELECT DATEI-D  ASSIGN TO DISC
           ORGANIZATION IS SEQUENTIAL
           ACCESS MODE IS SEQUENTIAL.

DATA DIVISION.
FILE SECTION.
FD  DATEI-D BLOCK CONTAINS 12 RECORDS
           RECORD CONTAINS 40 CHARACTERS
           LABEL RECORD IS STANDARD.
01  SATZ-D  PIC X(40).

WORKING-STORAGE SECTION.
01  ARB-SATZ    PIC X (40).

PROCEDURE DIVISION.
ANFANG.
    OPEN OUTPUT DATEI-D.
       -----------------
A.     -----------------
    WRITE SATZ-D FROM ARB-SATZ.
       -----------------
```

Wennn eine Magnetplatten-Datei mit sequentieller Organisation erstellt wird, werden die Datensätze vom Anfang bis zum Ende sequentiell geschrieben. Das Beispiel 5 zeigt die Erstellung einer Magnetplatten-Datei mit sequentieller Organisation.

Damit Daten auf DATEI-D geschrieben werden können, muß die Datei zuerst im OUTPUT-Modus eröffnet werden. Die WRITE-Anweisung überträgt zuerst die in ARB-SATZ enthaltenen Daten in SATZ-D. Dann schreibt die WRITE-Anweisung den in SATZ-D befindlichen Datensatz auf die DATEI-D.

DIE WRITE-ANEISUNG (Relative Datei - Sequentieller Zugriff)

Bei sequentiellem Zugriff schreibt die WRITE-Anweisung einen Datensatz in eine auf einer Magnetplatte gespeicherte relative Datei.

Format:

```
WRITE       Satz-Name      [FROM Identifier]
            [INVALID KEY       unbedingte Anweisung 1]
            [NOT INVALID KEY   unbedingte Anweisung 2]
            [END-WRITE]
```

Satz-Name ist der Name eines Datensatzes in der FILE SECTION der DATA DIVISION. In der WRITE-Anweisung kann der Satz-Name durch den Datei-Namen qualifiziert werden.

Der FILE-CONTROL-Eintrag muß die Klausel ORGANIZATION IS RELATIVE und indirekt oder direkt die Klausel ACCESS MODE IS SEQUENTIAL aufweisen.

Vor Ausführung einer WRITE-Anweisung muß die relative Datei durch eine OPEN OUTPUT-Anweisung eröffnet werden.

Die erste Ausführung der WRITE-Anweisung schreibt einen Datensatz in die die relative Satznummer 1 aufweisende Satzposition der relativen Datei. Nachfolgende Ausführungen der WRITE-Anweisung schreiben die nächsten Datensätze in die die relative Satznummer 2, 3, 4, ... aufweisenden Satzpositionen der relativen Datei.

Der durch die korrekte Ausführung der WRITE-Anweisung geschriebene Datensatz steht danach nicht mehr im Satz-Bereich; mit einer Ausnahme: Ist die Datei in einer SAME RECORD AREA-Klausel genannt, dann ist der Datensatz als ein Datensatz anderer in dieser SAME RECORD AREA-Klausel genannter Dateien für das Programm verfügbar.

Der FILE-CONTROL-Eintrag der relativen Datei kann in der Klausel ACCESS MODE IS SEQUENTIAL wahlweise die RELATIVE KEY-Angabe enthalten. Ist auf diese Weise ein relatives Satzschlüsselfeld definiert, dann wird die relative Satznummer des durch die WRITE-Anweisung gerade geschriebenen Datensatzes in das relative Satzschlüsselfeld übertragen.

FROM-Angabe

Ist FROM angegeben, dann findet eine Datenübertragung aus dem durch Identifier bezeichneten Bereich in den durch Satz-Name bezeichneten Satz-Bereich statt. Diese Übertragung erfolgt nach den für die MOVE-Anweisung bestimmten Regeln. Satz-Name und Identifier dürfen sich nicht auf denselben Speicherbereich beziehen.

Der Inhalt des Identifiers ist nach Ausführung der WRITE-Anweisung immer noch vorhanden.

INVALID KEY-Angabe

Die INVALID KEY-Angabe muß für eine WRITE-Anweisung spezifiziert werden, die auf eine relative Datei mit sequentiellem Zugriff Bezug nimmt, für die keine USE-Anweisung in den DECLARATIVES vorgesehen wurde.

Bei einem Versuch, über die Grenzen der relativen Datei hinaus zu schreiben, tritt die INVALID KEY-Bedingung auf und die Ausführung der WRITE-Anweisung gilt als erfolglos. Somit findet die Schreiboperation nicht statt und die Daten im Satz-Bereich bleiben erhalten.

Tritt die INVALID KEY-Bedingung auf, dann werden die folgenden Aktionen durchgeführt:

1. Ist die FILE STATUS-Klausel im FILE-CONTROL-Eintrag für die relative Datei spezifiziert, wird der Wert 24 in das FILE STATUS-Datenfeld übertragen, um einen Datenüberlauf anzuzeigen.

2. Wurde die INVALID KEY-Angabe in der WRITE-Anweisung verwendet, wird die Steuerung auf die unbedingte Anweisung übertragen. Keine der für diese Datei spezifizierten DECLARATIVE-Prozeduren wird ausgeführt.

3. Wurde die INVALID KEY-Angabe nicht verwendet, werden für diese Datei DECLARATIVE-Prozeduren ausgeführt.

Beispiel:

```
FILE-CONTROL.
    SELECT DATEI-F  ASSIGN TO DISC
           ORGANIZATION IS RELATIVE
           ACCESS MODE IS SEQUENTIAL
           RELATIVE KEY IS SCHL-F.
DATA DIVISION.
FILE SECTION.
ID  DATEI-F BLOCK CONTAINS 51 RECORDS
           RECORD CONTAINS 10 CHARACTERS
           LABEL RECORD IS STANDARD.

01  SATZ-F  PIC X(10).

WORKING-STORAGE SECTION.
01  SCHL-F  PIC 9999.

PROCEDURE DIVISION.
ANFANG.
    OPEN OUTPUT DATEI-F.
       -----------
A.     -----------
    WRITE SATZ-F INVALID KEY GO TO FEHLER.
       -----------
```

Wenn eine Magnetplatten-Datei mit relativer Organisation durch sequentiellen Zugriff erstellt wird, werden die Datensätze der Datei vom Anfang bis zum Ende sequentiell geschrieben. Das Beispiel zeigt die Erstellung einer Magnetplatten-Datei mit relativer Organisation durch sequentiellen Zugriff.

Damit Daten sequentiell auf die Datei geschrieben werden können, muß die Datei zuerst im OUTPUT-Modus eröffnet werden. Die erste Ausführung der WRITE-Anweisung schreibt einen Datensatz aus SATZ-F in die relative Satzposition 1 der Datei. Nachfolgende Ausführungen der WRITE-Anweisung schreiben Datensätze in die relativen Satzpositionen 2, 3, 4 usw.

Die relative Satznummer des durch die WRITE-Anweisung jeweils geschriebenen Datensatzes wird in das (DATEI-F zugeordnete) Satzschlüsselfeld (SCHL-F) übertragen.

DIE WRITE-ANWEISUNG
(Relative Datei - Direktzugriff/Dynamischer Zugriff)

Bei direktem oder dynamischem Zugriff schreibt die WRITE-Anweisung einen bestimmten Datensatz in eine relative Datei.

Format:

```
WRITE        Satz-Name        [FROM Identifier]
             [INVALID KEY              unbedingte Anweisung 1]
             [NOT INVALID KEY          unbedingte Anweisung 2]
             [END-WRITE]
```

Satz-Name ist der Name eines Datensatzes in der FILE SECTION der DATA DIVISION. In der WRITE-Anweisung kann der Satz-Name durch den Datei-Namen qualifiziert werden.

Der FILE-CONTROL-Eintrag muß die Klauseln ORGANIZATION IS RELATIVE und ACCESS MODE IS RANDOM oder DYNAMIC aufweisen. Vor Ausführung einer WRITE-Anweisung muß die relative Datei im OUTPUT- oder I-O-Modus eröffnet werden.

Bei direktem Zugriff auf die relative Datei schreibt die WRITE-Anweisung einen Datensatz in diejenige Satzposition der Datei, welche die im Satzschlüsselfeld enthaltene relative Satznummer aufweist. Das Satzschlüsselfeld muß durch die RELATIVE KEY-Angabe in der ACCESS MODE-Klausel definiert sein. Der Programmierer muß die gewünschte relative Satznummer vor Ausführung der WRITE-Anweisung in das Satzschlüsselfeld übertragen. Die Daten werden in der Datei in die durch das Satzschlüsselfeld bestimmte relative Satz-Position geschrieben.

Der bei korrekter Ausführung der WRITE-Anweisung geschriebene Datensatz steht anschließend nicht mehr im Satzbereich zur Verfügung, mit einer Ausnahme: Ist die Datei in einer SAME RECORD AREA-Klausel genannt, dann ist der Datensatz als ein Datensatz anderer in dieser SAME RECORD AREA-Klausel genannter Dateien für das Programm verfügbar.

FROM-Angabe

Ist FROM angegeben, findet eine Datenübertragung aus dem durch Identifier bezeichneten Bereich in den durch Satz-Name bezeichneten Satz-Bereich statt. Diese Übertragung erfolgt nach den für die MOVE-Anweisung gültigen Regeln. Satz-Name und Identifier dürfen sich nicht auf denselben Speicherbereich beziehen. Der Inhalt des Identifiers ist nach Ausführung der WRITE-Anweisung immer noch vorhanden.

INVALID KEY-Angabe

Die INVALID KEY-Angabe muß in einer WRITE-Anweisung spezifiziert werden, die auf eine relative Datei mit direkter oder dynamischer Zugriffsart Bezug nimmt, für die keine USE-Anweisung in den DECLARATIVES vorgesehen wurde.

Wenn eine der folgenden Bedingungen besteht, ist auch die INVALID KEY-Bedingung gegeben:

1. Es wird versucht, über die Grenzen der Datei hinaus zu schreiben.

2. Das Satzschlüsselfeld gibt eine relative Satz-Position an, die bereits einen gültigen Datensatz enthält.

Bei Auftreten der INVALID KEY-Bedingung gilt die Ausführung der WRITE-Anweisung als erfolglos und die Daten im Satz-Bereich bleiben erhalten.

Tritt die INVALID KEY-Bedingung auf, werden folgende Aktionen durchgeführt:

1. Ist die FILE STATUS-Klausel im FILE-CONTROL-Eintrag spezifiziert, wird der Wert 22 in das FILE STATUS-Datenfeld übertragen, um eine bereits belegte Satzposition anzuzeigen; der Wert 24 wird in das FILE STATUS-Datenfeld übertragen, wenn versucht wird, über die Grenzen der Datei hinauszuschreiben.

2. Ist die INVALID KEY-Angabe in der WRITE-Anweisung vorhanden, wird die Steuerung auf die unbedingte Anweisung übertragen. Keine der für diese Datei spezifizierten DECLARATIVE-Prozeduren wird ausgeführt.

3. Ist die INVALID KEY-Angabe nicht vorhanden, dann werden die für diese Datei spezifizierten DECLARATIVE-Prozeduren ausgeführt.

Beispiel 1:

```
FILE-CONTROL.
    SELECT DATEI-G  ASSIGN TO DISC
            ORGANIZATION IS RELATIVE
            ACCESS MODE IS RANDOM, RELATIVE KEY IS SCHL-G.

DATA DIVISION.
FILE SECTION.
FD  DATEI-G BLOCK CONTAINS 20 RECORDS
            RECORD CONTAINS 25 CHARACTERS
            LABEL RECORD IS STANDARD.
01  SATZ-G  PIC X(25).
WORKING-STORAGE SECTION.
01  SCHL-G  PIC 9(5).

PROCEDURE DIVISION.
ANFANG.
    OPEN OUTPUT DATEI-G.
       -------------
    MOVE 44 TO SCHL-G.
    WRITE SATZ-G INVALID KEY GO TO FEHLER.
       -------------
```

Wenn eine Magnetplatten-Datei mit relativer Organisation im Direktzugriff erstellt wird, werden die Datensätze per Direktzugriff in bestimmte Satz-Positionen der Datei geschrieben. Beispiel 1 zeigt die Erstellung einer Magnetplatten-Datei relativer Organisation im Direktzugriff.

Die Datei muß im OUTPUT-Modus eröffnet werden.

Der Programmierer überträgt die Ganzzahl 44 in das Satzschlüsselfeld SCHL-G, um die rel. Satzposition, in welche ein Datensatz geschrieben werden soll, zu spezifizieren. Die WRITE-Anweisung schreibt die im Satzbereich SATZ-G bereitgestellten Daten in die Satzposition 44 der Datei. Würde ein Versuch gemacht, einen Satz über die Grenzen der relativen Datei hinaus zu schreiben, würde die WRITE-Anweisung in INVALID KEY die Anweisung GO TO FEHLER durchführen.

Beispiel 2:

```
 FILE-CONTROL.
     SELECT DATEI-H  ASSIGN TO DISC
              ORGANIZATION IS RELATIVE
              ACCESS MODE IS RANDOM, RELATIVE KEY IS H1.
*        ODER ACCESS MODE IS DYNAMIC
 DATA DIVISION.
 FILE SECTION.
 FD  DATEI-H BLOCK CONTAINS 25 RECORDS
              RECORD CONTAINS 20 CHARACTERS
              LABEL RECORD IS STANDARD.
 01  SATZ-H  PIC X(20).
 WORKING-STORAGE SECTION.
 01  H1  PIC 999.

 PROCEDURE DIVISION.
 ANFANG.
     OPEN I-O DATEI-H.
        -----------
     MOVE 4 TO H1.
     WRITE SATZ-H INVALID KEY GO TO EE.
        -----------
```

Da Datensätze in die bereits bestehende DATEI-H geschrieben werden sollen, muß die Datei im I-O-Modus eröffnet werden.

DIE WRITE-ANWEISUNG
(Indexierte Datei - Sequentieller Zugriff)

Bei sequentiellem Zugriff schreibt die WRITE-Anweisung einen Datensatz in eine indexierte Datei.

Format:

```
WRITE        Satz-Name        [FROM Identifier]
             [INVALID KEY     unbedingte Anweisung 1]
             [NOT INVALID KEY unbedingte Anweisung 2]
             [END-WRITE]
```

Satz-Name ist der Name eines Datensatzes in der FILE SECTION der DATA DIVISION. In der WRITE-Anweisung kann der Satz-Name durch einen Datei-Namen qualifiziert werden.

Der FILE-CONTROL-Eintrag muß die Klausel ORGANIZATION IS INDEXED und indirekt oder direkt die Klausel ACCESS MODE IS SEQUENTIAL aufweisen.

Die indexierte Datei kann Datensätze fester oder variabler Längen enthalten. Besteht die Datei aus Sätzen variabler Länge, dürfen die RECORD CONTRAINS-Klausel und die Satzbeschreibungen nicht die vom variable-Längen-Indikator (VLI) eingenommenen Bytes beinhalten.

Vor Ausführung einer WRITE-Anweisung muß die indexierte Datei durch eine OPEN OUTPUT-Anweisung eröffnet werden.

Die Datensätze müssen in aufsteigender Reihenfolge der Satzschlüssel geschrieben werden. Vor Ausführung der WRITE-Anweisung muß vom Programmierer der richtige Wert in das Satzschlüsselfeld gebracht werden. Das Satzschlüsselfeld muß in der RECORD KEY-Klausel im FILE-CONTROL-Eintrag angegeben sein.

Die erste Ausführung der WRITE-Anweisung schreibt den ersten Datensatz in die indexierte Datei. Nachfolgende Ausführungen der WRITE-Anweisung schreiben den sequentiell jeweils nächsten Datensatz in die Datei gemäß der aufsteigenden Folge der Satzschlüssel.

Satz-Name bezeichnet den Datensatz, dessen Inhalt in die Datei geschrieben wird. Der bei korrekter Ausführung der WRITE-Anweisung geschriebene Datensatz steht nicht mehr in dem dem Satz-Namen zugeordneten Satz-Bereich zur Verfügung mit einer Ausnahme: Ist die Datei in einer SAME RECORD AREA-Klausel genannt, ist der Datensatz als ein Datensatz anderer in dieser SAME RECORD AREA-Klausel genannter Dateien für das Programm weiterhin verfügbar.

FROM-Angabe

Ist FROM angegeben, findet vorab eine Datenübertragung aus dem Identifier in den Satz-Bereich statt. Diese Übertragung erfolgt nach den für die MOVE-Anweisung bestimmten Regeln. Satz-Name und Identifier dürfen sich nicht auf denselben Speicherbereich beziehen.

Der Inhalt des Identifiers ist nach Ausführung der WRITE-Anweisung immer noch verfügbar.

INVALID KEY-Angabe

Die INVALID KEY-Angabe muß für eine WRITE-Anweisung spezifiziert werden, die auf eine indexierte Datei mit sequentiellem Zugriff Bezug nimmt, für die keine USE-Anweisung in den DECLARATIVES vorgesehen wurde.

Wenn eine der folgenden Bedingungen besteht, ist auch die INVALID KEY-Bedingung gegeben:

1. Es wird versucht, über die Grenzen der Datei hinaus zu schreiben (Dateiüberlauf).

2. Der Wert des Satzschlüssels im zu schreibenden Datensatz ist nicht größer als der Wert des Satzschlüssels im zuvor in die Datei geschriebenen Satz.

Beim Auftreten der INVALID KEY-Bedingung gilt die Ausführung der WRITE-Anweisung als erfolglos und die Daten im Satz-Bereich bleiben erhalten.

Wenn die INVALID KEY-Bedingung auftritt, werden folgende Aktionen durchgeführt:

1. Ist die FILE STATUS-Klausel im FILE-CONTROL-Eintrag für die indexierte Datei spezifiziert, wird der Wert 21 in das FILE STATUS-Feld übertragen, um einen Datenüberlauf anzuzeigen.

2. Wurde die INVALID KEY-Angabe in der WRITE-Anweisung verwendet, wird die Steuerung auf die unbedingte Anweisung übertragen. Keine der für diese Datei spezifizierten DECLARATIVE-Prozeduren wird ausgeführt.

3. Wurde die INVALID KEY-Angabe in der WRITE-Anweisung nicht verwendet, werden für diese Datei DECLARATIVE-Prozeduren ausgeführt.

Auf der nächsten Seite finden Sie ein Beispiel:

Beispiel 1:

```
FILE-CONTROL.
    SELECT DATEI-J  ASSIGN TO DISC
           ORGANIZATION IS INDEXED
           ACCESS MODE IS SEQUENTIAL
           RECORD KEY IS SCHL-J.
DATA DIVISION.
FILE SECTION.
FD  DATEI-J BLOCK CONTAINS 25 RECORDS
           RECORD CONTAINS 20 CHARACTERS
           LABEL RECORD IS STANDARD.
01  SATZ-J.
    02  SCHL-J  PIC XXXX.
    02  FILLER  PIC X(16).

PROCEDURE DIVISION.
ANFANG.
    OPEN OUTPUT DATEI-J.
       ------------
    MOVE NR TO SCHL-J.
A.  WRITE SATZ-J INVALID KEY GO TO FEHLER.
       -----------
```

Wenn eine Magnetplatten-Datei mit indexierter Organisation durch sequentiellen Zugriff erstellt wird, werden die Datensätze der Datei vom Anfang bis zum Ende sequentiell geschrieben in der aufsteigenden Reihenfolge der Satzschlüsselwerte. Das Beispiel zeigt die sequentielle Erstellung einer Magnetplatten-Datei indexierter Organisation. SCHL-J gibt die Position des Satzschlüsselfeldes im Satzbereich SATZ-J im Speicher an.

Damit Daten sequentiell in die DATEI-J geschrieben werden können, muß die Datei im OUTPUT-Modus eröffnet werden. Die MOVE-Anweisung überträgt einen Wert (z. B. eine Konto-Nr.) in das Satzschlüsselfeld SCHL-J. Die WRITE-Anweisung schreibt den Datensatz aus SATZ-J in die indexierte Datei. Ist der Satzschlüssel im zu schreibenden Datensatz nicht größer als der Satzschlüssel in dem Datensatz, der zuletzt in die Datei geschrieben wurde, führt die WRITE-Anweisung (INVALID KEY) die Anweisung GO TO FEHLER durch.

DIE WRITE-ANWEISUNG
(Indexierte Datei - Direktzugriff/Dynamischer Zugriff)

Bei direktem oder dynamischem Zugriff schreibt die WRITE-Anweisung einen bestimmten Datensatz in eine indexierte Datei.

Format:

```
WRITE      Satz-Name          [FROM Identifier]
           [INVALID KEY       unbedingte Anweisung 1]
           [NOT INVALID KEY   unbedingte Anweisung 2]
           [END-WRITE]
```

Satz-Name ist der Name eines Datensatzes in der FILE SECTION der DATA DIVISION. In der WRITE-Anweisung kann der Satz-Name durch einen Datei-Namen qualifiziert werden.

Der FILE-CONTROL-Eintrag muß die Klauseln ORGANIZATION IS INDEXED und ACCESS MODE IS RANDOM oder DYNAMIC aufweisen.

Die indexierte Datei kann Datensätze fester oder variabler Länge enthalten. Besteht die indexierte Datei aus Sätzen variabler Länge, dürfen die RECORD CONTAINS-Klausel und die Satzbeschreibungen nicht die vom variable-Längen-Indikator (VLI) beanspruchten Bytes beinhalten.

Vor Ausführung der WRITE-Anweisung muß die Datei im OUTPUT-Modus oder im I-O-Modus eröffnet werden.

Die RECORD KEY-Klausel muß im FILE-CONTROL-Eintrag zur Ausführung zweier Funktionen vorhanden sein. Ihre erste Funktion ist, den Satzschlüssel in jedem Datensatz der indexierten Datei zu spezifizieren. Ihre zweite Funktion ist, das Satzschlüsselfeld im Speicher zu bestimmen. Vor Ausführung der WRITE-Anweisung muß der Programmierer den entsprechenden Satzschlüssel (z. B. Kto.-Nr.) in das Satzschlüsselfeld übertragen. Die Daten werden daraufhin in die indexierte Datei geschrieben gemäß dem Wert des Satzschlüsselfeldes.

Der bei korrekter Ausführung der WRITE-Anweisung geschriebene Datensatz ist mit einer Ausnahme nicht mehr im Satz-Bereich des Speichers verfügbar: Ist die zugeordnete Datei in einer SAME RECORD AREA-Klausel genannt, ist der Datensatz als ein Datensatz anderer in dieser SAME RECORD AREA-Klausel genannter Dateien weiterhin für das Programm verfügbar.

FROM-Angabe

Ist FROM angegeben, findet vorab eine Datenübertragung aus dem Identifier in den Satz-Bereich statt. Diese Übertragung erfolgt nach den für die MOVE-Anweisung bestimmten Regeln. Satz-Name und Identifier düren sich nicht auf denselben Speicherbereich beziehen.

INVALID KEY-Angabe

Die INVALID KEY-Angabe muß für eine WRITE-Anweisung spezifiziert werden, die auf eine indexierte Datei mit direktem oder dynamischem Zugriff Bezug nimmt, für die keine USE-Anweisung in den DECLARATIVES vorgesehen wurde.

Wenn eine der folgenden Bedingungen besteht, ist auch die INVALID KEY-Bedingung gegeben:

1. Es wird versucht, über die Grenzen der Datei hinaus zu schreiben (Datei-Überlauf).

2. Der Wert des Satzschlüssels in dem zu schreibenden Datensatz ist gleich dem Wert des Satzschlüssels eines bereits in der Datei vorhandenen Datensatzes.

Beim Auftreten der INVALID KEY-Bedingung gilt die WRITE-Anweisung als erfolglos. Die Daten im Satz-Bereich bleiben erhalten.
Wenn INVALID KEY auftritt, werden folgende Aktionen durchgeführt:

1. Ist die FILE STATUS-Klausel im FILE-CONTROL-Eintrag für die Datei spezifiziert, wird der Wert 22 in das FILE STATUS-Feld übertragen, um einen doppelten Schlüssel anzuzeigen; der Wert 24 wird in das FILE STATUS-Feld übertragen, um einen Dateiüberlauf anzuzeigen.

2. Wurde die INVALID KEY-Angabe in der WRITE-Anweisung verwendet, wird die Steuerung auf die unbedingte Anweisung übertragen. Keine der für diese Datei spezifizierten DECLARATIVE-Prozeduren wird ausgeführt.

3. Wurde die INVALID KEY-Angabe in der WRITE-Anweisung nicht verwendet, werden für diese Datei DECLARATIVE-Prozeduren ausgeführt.

Beispiel 1:

```
FILE-CONTROL.
    SELECT DATEI-K  ASSIGN TO DISC
           ORGANIZATION IS INDEXED
           ACCESS MODE IS RANDOM
           RECORD KEY IS SCHL-K.
DATA DIVISION.
FILE SECTION.
FD  DATEI-K BLOCK CONTAINS 25 RECORDS
           RECORD CONTAINS 20 CHARACTERS
           LABEL RECORD IS STANDARD.
01  SATZ-K.
    02  SCHL-K  PIC XXX.
    02  FILLER  PIC X(17).

PROCEDURE DIVISION.
ANFANG.
    OPEN OUTPUT DATEI-K.
       ----------
    MOVE SATZ-NR TO SCHL-K.
    WRITE SATZ-K INVALID KEY GO TO SCHL-FEHLER.
       ----------
```

Wenn eine Indexdatei durch Direktzugriff erstellt wird, werden die Datensätze gemäß dem Wert des Satzschlüsselfeldes direkt in die Datei geschrieben.

Damit Daten im Direktzugriff in die DATEI-K geschrieben werden können, muß die Datei im OUTPUT-Modus eröffnet werden. SCHL-K gibt die Position des Satzschlüsselfeldes im Satzbereich SATZ-K im Speicher an.

Der Programmierer überträgt mittels MOVE einen Wert in das Satzschlüsselfeld SCHL-K. Die WRITE-Anweisung schreibt dann den Datensatz aus SATZ-K in die indexierte Datei. Enthält die Datei bereits einen Satz mit einem entsprechenden Satz-Schlüssel, wird die unbedingte Anweisung GO TO SCHL-FEHLER ausgeführt (INVALID KEY).

Beispiel 2:

```
FILE-CONTROL.
    SELECT DATEI-L  ASSIGN TO DISC
           ORGANIZATION IS INDEXED
           ACCESS MODE IS RANDOM     (oder DYNAMIC)
           RECORD KEY IS SCHL-L.
DATA DIVISION.
FILE SECTION.
FD  DATEI-L BLOCK CONTAINS 20 RECORDS
           RECORD CONTAINS 25 CHARACTERS
           LABEL RECORD IS STANDARD.
01  SATZ-L.
    02  SCHL-L  PIC X(7).
    02  FILLER  PIC X(18).

PROCEDURE DIVISION.
ANFANG.
    OPEN I-O DATEI-L.
       ------------
    MOVE SATZ-NR TO SCHL-L.
    WRITE SATZ-L INVALID KEY GO TO SCHL-FEHLER.
       -----------
```

SCHL-L spezifiziert die Position des in SATZ-L im Speicher befindlichen Satzschlüsselfeldes. Da Datensätze in die bereits bestehende Datei zu schreiben sind, muß sie im I-O-Modus eröffnet werden.

Der Programmierer überträgt mittels MOVE einen Wert in das Satzschlüsselfeld SCHL-L. Die WRITE-Anweisung schreibt den Inhalt des (SATZ-L genannten) Datensatzbereiches in die DATEI-L. Wird in der DATEI-L ein Datensatz gefunden, der bereits einen SCHL-L entsprechenden Satzschlüssel hat, führt die WRITE-Anweisung (INVALID KEY) die Anweisung GO TO SCHL-FEHLER aus.

KAPITEL 10: TABELLEN

EINLEITUNG

Zunächst sei auf die bereits an vorhergehender Stelle in diesem Handbuch beschriebenen OCCURS- und INDEXED BY-Klauseln der Datenfeldbeschreibung und auf die Befehle SET, SEARCH und PERFORM VARYING verwiesen. Zusammenfassend kann hier gesagt werden, daß unter COBOL zwei Methoden zum Zugriff auf Tabellen möglich sind: Die Indexmethode und die Subskriptmethode. Die Subskriptmethode unterscheidet sich von der Indexmethode dadurch, daß die Rolle des Indexfeldes durch ein ganz normales numerisches Datenfeld übernommen wird, also in der OCCURS-Klausel der Tabelle der Zusatz INDEXED BY weggelassen wird, und: Bei der Subskriptmethode können die Befehle SEARCH und SET nicht angewendet werden.

Bei ITX COBOL 85 sind mehr als 3 Tabellendimensionen möglich.

BEARBEITUNG EINER EINDIMENSIONALEN EINGABETABELLE

Von einer Eingabetabelle wird gesprochen, wenn sie zu Beginn der Verarbeitung Daten enthält. Nachstehend finden Sie die graphische Darstellung einer eindimensionalen Eingabetabelle:

AROT
BGELB
CGRUEN
DBLAU
ZORANGE

Im nachfolgenden Definitionsbeispiel definiert der Programmierer zunächst den Inhalt der Tabelle und legt anschließend per Redefinition die Tabelle sozusagen wie ein Raster über den Tabelleninhalt:

```
01 TABINH.
05 FILLER PIC X(7) VALUE "AROT   ".
05 FILLER PIC X(7) VALUE "BGELB  ".
05 FILLER PIC X(7) VALUE "CGRUEN ".
05 FILLER PIC X(7) VALUE "DBLAU  ".
05 FILLER PIC X(7) VALUE "ZORANGE".

01 TABELLE  REDEFINES  TABINH.
05 TPOSTEN  OCCURS  5  INDEXED BY  TI.
10 FARBCODE PIC X.
10 FARBE    PIC X(6).
```

Angenommen, es bestünde nun folgende Aufgabenstellung: Bei Eingabe eines Farbcodes in ein Feld EIN-FELD soll die Tabelle nach diesem gewünschten Farbcode abgesucht werden und - falls gefunden - der entsprechende Farbtext in ein Feld AUS-FELD übertragen werden. Zunächst ist hierzu noch die Definition des Feldes EIN-FELD erforderlich:

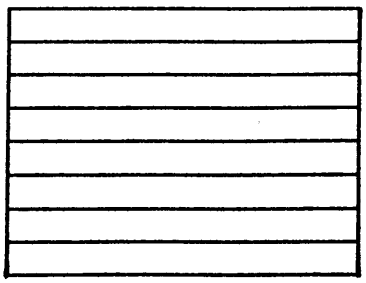
```
77 EIN-FELD  PIC X.
```

Mit der nachfolgenden Befehlsreihe kann nun auf einen gewünschten Farbtext der Tabelle zugegriffen werden.

```
PROCEDURE DIVISION.
ANF.
     ACCEPT EIN-FELD.
     SET T1 TO 1.
     SEARCH TPOSTEN
            AT END GO NICHTGEFUNDEN
            WHEN FARBCODE(TI) = EIN-FELD
            MOVE FARBE(TI) TO AUSFELD.
     WRITE AUSGABE-SATZ.
```

BEARBEITUNG EINER EINDIMENSIONALEN AUSGABETABELLE

Von einer Ausgabetabelle wird gesprochen, wenn sie zu Beginn der Verarbeitung leer ist. Nachstehend finden Sie die graphische Darstellung einer eindimensionalen Ausgabetabelle:



Die Tabelle kann folgendermaßen definiert werden:

```
01 TINH  PIC X(72) VALUE ALL "0".
01 TABELLE REDEFINES TINH.
05 TPOST PIC 9(7)V99
                  OCCURS 8
                  INDEXED BY TI.
```

Angenommen, es bestünde folgende Aufgabenstellung: Bei Eingabe einer Filialnummer zwischen 1 und 8 in ein Feld FIL-NR und eines Umsatzes in ein Feld UMSATZ soll der Umsatz in den Tabellenposten addiert werden, auf den die Filialnummer zeigt. Dazu wären noch folgende zwei Felddefinitionen erforderlich:

```
77 FIL-NR PIC 9.
77 UMSATZ PIC 9(6)V99.
```

Mit der nachfolgenden Befehlsreihe kann nun ein jeder eingegebene Umsatz dem entsprechenden Filialposten in der Tabelle zugeordnet werden:

```
EING.
     ACCEPT FIL-NR.
     IF FIL-NR = 0 GO TO ENDE.
     IF FIL-NR > 8 GO TO EING.
     ACCEPT UMSATZ.
     SET TI TO FIL-NR.
     ADD UMSATZ TO TPOST(TI).
     GO TO EING.
```

MEHRDIMENSIONALE TABELLEN

Eine zwei-dimensionale Tabelle enthält in jedem Tabellenposten eine weitere Tabelle.

Eine drei-dimensionale Tabelle enthält in jedem Tabellenposten eine Tabelle, deren Posten wiederum je eine Tabelle enthalten.

Mehr als 3 Tabellen-Dimensionen sind in ITX COBOL 85 möglich.

Beispiel einer zwei-dimensionalen Tabelle:

Abteilung	Kasse 1	Kasse 2	Kasse 3	Kasse 4
A	000000	000000	000000	000000
B	000000	000000	000000	000000
H	000000	000000	000000	000000
M	000000	000000	000000	000000
S	000000	000000	000000	000000

1. Dimension: Abteilungen
2. Dimension: Kassen innerhalb jeder Abteilung

Im folgenden Beispiel wird die Tabelle mit den Abteilungscodes und den genullten Kassen-Salden gefüllt.

```
1   TAB.
 2  FILLER  PIC X(25)  VALUE  "A00000000000000000000000".
 2  FILLER  PIC X(25)  VALUE  "B00000000000000000000000".
 2  FILLER  PIC X(25)  VALUE  "H00000000000000000000000".
 2  FILLER  PIC X(25)  VALUE  "M00000000000000000000000".
 2  FILLER  PIC X(25)  VALUE  "S00000000000000000000000".

1   UMSATZ-TAB REDEFINES TAB.
 2  ABTEILUNG OCCURS 5.
  3 ABT-CODE  PIC X.
  3 KASSE     PIC 9999V99  OCCURS 4.
```

In der Definition einer 2-dimensionalen Tabelle erscheint die OCCURS-Klausel zweimal:

1. In der Definition der "groben" Tabelle (1. Dimension).
2. In einer Definitionszeile, die der Definition mit der ersten OCCURS-Klausel untergeordnet ist (2. Dimension).

Der Zugriff auf ein Datenfeld der 2. Dimension erfordert zwei Subskript- bzw. Index-Angaben in der Klammer nach dem Daten-Namen. Die erste Dimension muß vor der zweiten angegeben werden.

Beispiel:

Zugriff auf Kasse 3 (2. Dimension) innerhalb der 5. Abteilung (1. Dimension) in der vorangehend definierten Tabelle (Subskriptmethode, da keine Indizes!):

```
MOVE KASSE  (5   3)  TO ARB-KASSE.
```

oder:

```
MOVE  5 TO  NUMFELD1.
MOVE  3 TO  NUMFELD2.
MOVE  KASSE (NUMFELD1, NUMFELD2) TO ARB-KASSE.
```

Beispiel einer drei-dimensionalen Tabelle:

	Abteilung	Kasse 1	Kasse 2	Kasse 3	Kasse 4
Filiale 10	A	000000	000000	000000	000000
	B	000000	000000	000000	000000
	H	000000	000000	000000	000000
Filiale 20	A	000000	000000	000000	000000
	B	000000	000000	000000	000000
	H	000000	000000	000000	000000

1. Dimension: Filialen
2. Dimension: Abteilungen innerhalb der Filialen
3. Dimension: Kassen innerhalb der Abteilungen innerhalb der Filialen

Im folgenden Beispiel werden Filial- und Abteilungscodes als Konstanten vorgegeben, die Kassen-Salden mit Nullen gefüllt.

```
1    TAB2.
 2   FILLER  PIC X(27)  VALUE  "10A000000000000000000000000".
 2   FILLER  PIC X(25)  VALUE    "B00000000000000000000000".
 2   FILLER  PIC X(25)  VALUE    "H00000000000000000000000".
 2   FILLER  PIC X(27)  VALUE  "20A000000000000000000000000".
 2   FILLER  PIC X(25)  VALUE    "B00000000000000000000000".
 2   FILLER  PIC X(25)  VALUE    "H00000000000000000000000".

1    UMSATZ-TAB REDEFINES TAB2.
 2   FILIALE    OCCURS 2.
  3  FILIALE-CODE PIC 99.
  3  ABTEILUNG  OCCURS 3.
   4 ABT-CODE     PIC X.
   4 KASSE        PIC 9999V99 OCCURS 4.
```

In der Definition einer 3-dimensionalen Tabelle erscheint die OCCURS-Klausel dreimal:

1. In der Definition der "groben" Tabelle (1. Dimension)

2. In der Definition, die der ersten Dimension untergeordnet ist (2. Dimension)

3. In einer Definition, die der zweiten Dimension untergeordnet ist (3. Dimension)

Beim Zugriff auf eine 3-dimensionale Tabelle müssen in der Klammer nach dem Datennamen 1 bis 3 Subskripts bzw. Indizes angegeben werden, je nachdem, auf welche Dimension zugegriffen wird.

Die Reihenfolge der Subskripts bzw. Indizes ist:

1. erste (gröbste) Dimension
2. zweite (mittlere) Dimension
3. dritte (feinste) Dimension

Zugriff auf die zweite Dimension einer dreidimensionalen Tabelle:

```
1     L-SUB      PIC 9.
1     F-SUB      PIC 9.          Subskripts
1     V-SUB      PIC 99.

 1    TABELLE.
  2   LAND              OCCURS 4.
   3  FILIALE           OCCURS 9.
    4 VERTRETER PIC 999  OCCURS 20.

 PROCEDURE DIVISION.

    MOVE 1 TO L-SUB
    MOVE 4 TO F-SUB.

    MOVE FILIALE (L-SUB  F-SUB)  TO FILIAL-ZEILE.
```

Zugriff auf die dritte Dimension derselben Tabelle:

```
MOVE 2 TO L-SUB
MOVE 4 TO F-SUB
MOVE 7 TO V-SUB.

ADD VERTRETER-UMSATZ
    TO VERTRETER (L-SUB  F-SUB  V-SUB).
```

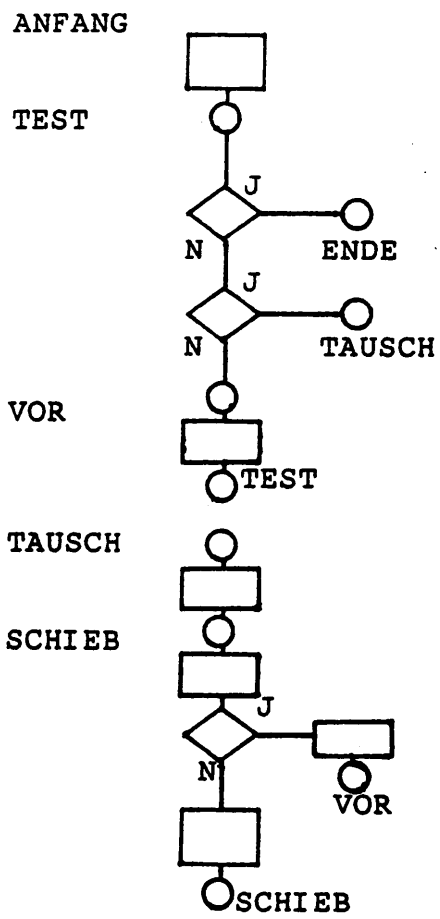
Subskript und Index gemischt

In einem Zugriffsbefehl auf eine mehrdimensionale Tabelle können in derselben Klammer Index- und Subskript-Angaben stehen:

```
1     TABELLE.
 2    POSTEN OCCURS 10 INDEXED BY X1.
  3   FELD   OCCURS 20 INDEXED BY X2 PIC 9(6).
      .
      .
1     NUM    PIC 99.
                      Index Subskript

      ADD WERT TO FELD  (X1 NUM).
```

TABELLE AUFSTEIGEND SORTIEREN

(1000 POSTEN TP; 2 INDIZES I UND J; SCHLÜSSEL TA; HILFSFELD NP)




```
ANFANG.
    SET I TO 1000, SET TABEND TO I,
    SET I TO 1, SET TABSTART TO I.
TEST.


    IF I = TABEND GO TO ENDE.


    IF TA(I) > TA(I + 1) GO TO TAUSCH.


VOR.
    SET I UP BY 1, GO TO TEST.


TAUSCH.
    MOVE TP(I + 1) TO NP,
    SET J TO I.
SCHIEB.
    MOVE TP(J) TO TP(J + 1)

    IF J = TABSTART OR NA NOT < TA(J - 1)
      MOVE NP TO TP(J)  GO TO VOR.

    SET J DOWN BY 1

    GO TO SCHIEB.
```

DEFINITIONEN:

```
77  TABSTART    USAGE INDEX.
77  TABEND      USAGE INDEX.

01  NP.
    02  NA  PIC 9(4).
    02  NB  PIC S9(6) COMP.

01  TABELLE.
    02  TP  OCCURS  1000 TIMES
             INDEXED BY I J.
        03  TA  PIC 9(4).
        03  TB  PIC S9(6) COMP.
```

KAPITEL 11: DRUCK-DATEIEN

Aus der Gesamtheit der Beschreibungen in diesem Handbuch

a) der LINE-n Klausel bei SPECIAL-NAMES,
b) der LINAGE-Klausel in der Druckdateibeschreibung unter FILE SECTION sowie
c) der WRITE-Anweisung

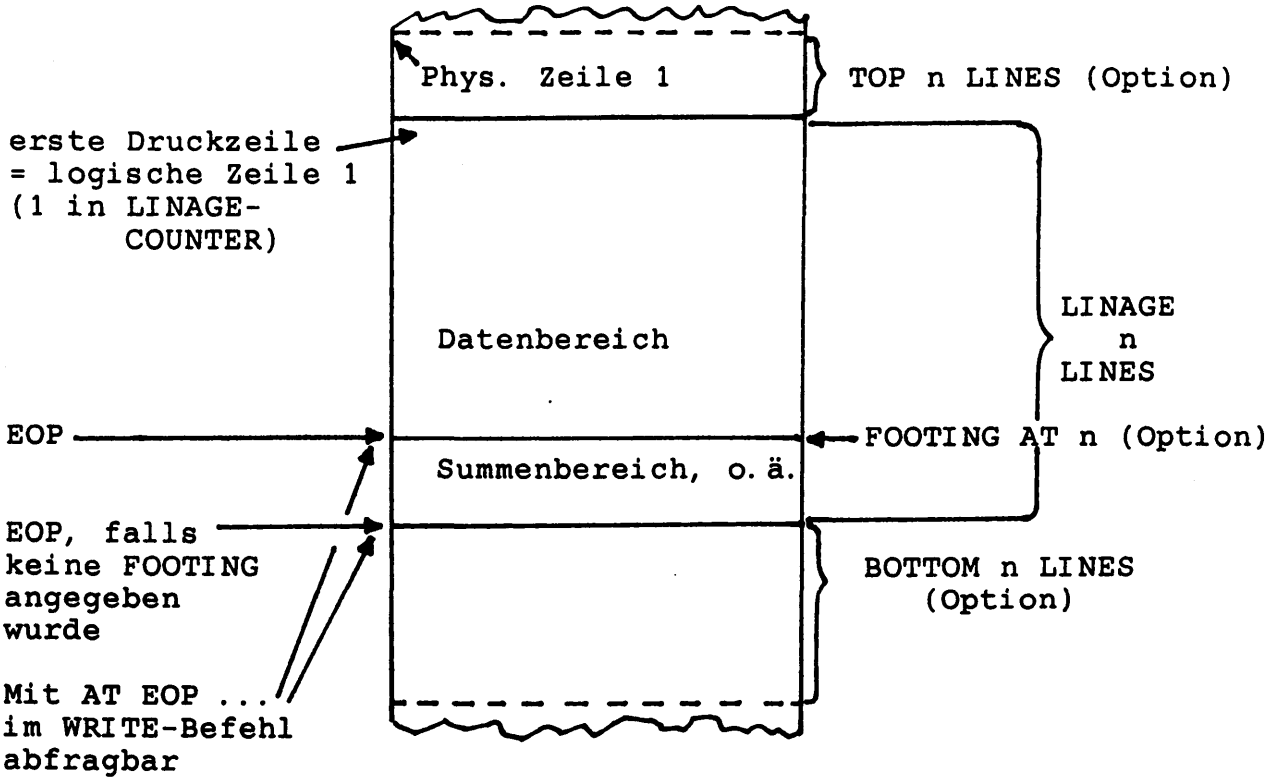
ergeben sich drei verschiedene Möglichkeiten der Programmierung einer Druckdatei:

1. Der Verzicht auf die Klauseln LINE-n sowie LINAGE.
   Hierbei ist der Programmieraufwand höher.

2. Die Verwendung der LINE-n Klausel.
   Diese schließt die gleichzeitige Verwendung der LINAGE-Klausel aus. Das bedeutet, daß ein ggf. erforderlicher Zeilenzähler selbst zu definieren und zu behandeln ist.

3. Die Verwendung der LINAGE-Klausel.
   Diese ermöglicht eine umfangreiche Reduzierung des Programmieraufwandes. Sie schließt die gleichzeitige Verwendung der LINE-n Klausel aus. Sie stellt automatisch das stets von der Software aktualisierte und vom Programm jederzeit abfragbare Zeilenzähl-Register LINAGE COUNTER zur Verfügung.

Auf der nächsten Seite folgt ein Beispiel einer Formularaufteilung mittels der Linage-Klausel:


Phys. Zeile 1
TOP n LINES (Option)
erste Druckzeile
= logische Zeile 1
(1 in LINAGE-
COUNTER)
Datenbereich
LINAGE
n
LINES
EOP
FOOTING AT n (Option)
Summenbereich, o.ä.
EOP, falls
keine FOOTING
angegeben
wurde
BOTTOM n LINES
(Option)
Mit AT EOP ...
im WRITE-Befehl
abfragbar

KAPITEL 12:

UNABHÄNGIG COMPILIERTE MODULE (INTER-PROGRAM-COMMUNICATION)

ALLGEMEINES

Um ein größeres Software-Projekt überschaubarer zu machen, können einzelne Programmteile als separate Programm-Module geschrieben und umgewandelt werden.

Zum Zeitpunkt der Ausführung des gesamten Programms müssen sämtliche benötigten Module auf derselben Platte gespeichert sein. Die Module können mit $LINK zusammengebunden werden.

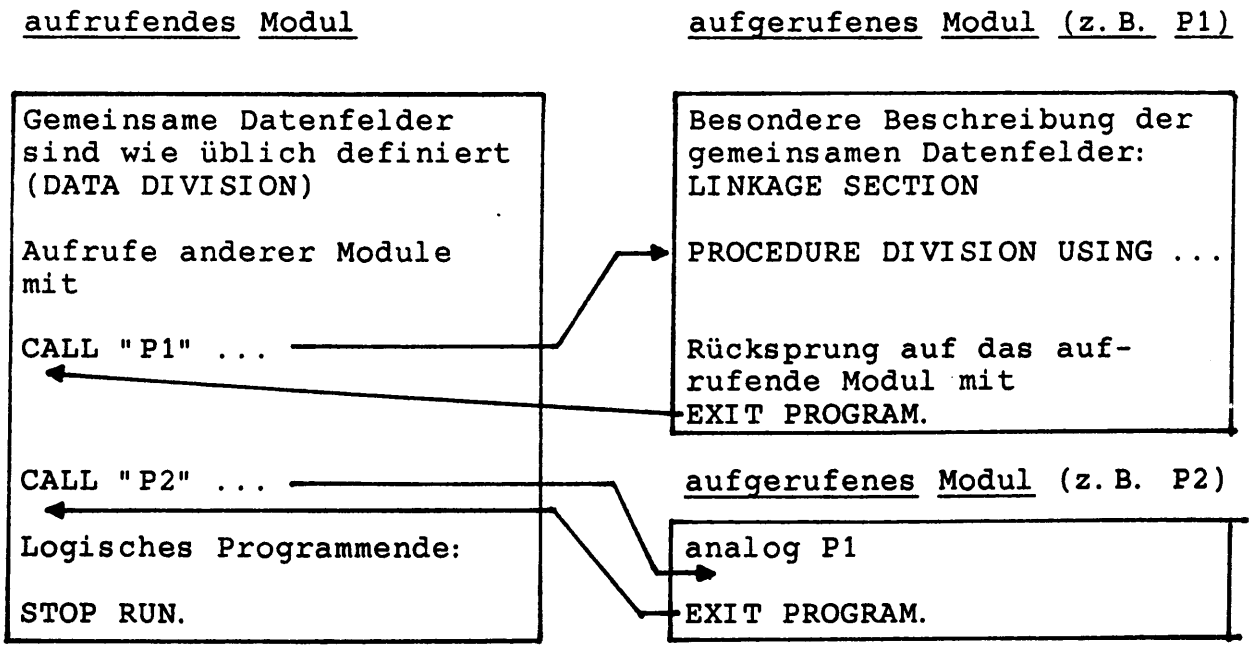
Dieses Kapitel umfaßt die Definitionen und Befehle zur Herstellung von Verbindungen zu anderen Programm-Modulen. Dabei handelt es sich um zwei Gruppen von Anweisungen:

- Übergabe der Steuerung von Modul zu Modul
- Beschreibung von Datenfeldern, auf welche in mehreren Modulen gemeinsam zugegriffen wird.

Ein Programm-Modul, welches andere Module unterprogrammartig aufruft, nennt man "calling program" oder aufrufendes Modul. Die aufgerufenen Module werden auch als "called programs" angesprochen.

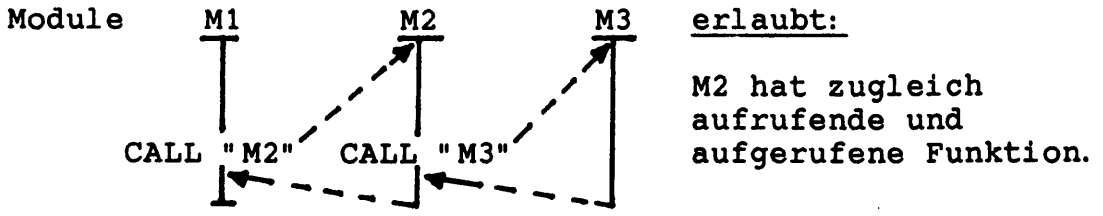
Die Module, die zusammen ein arbeitsfähiges Programm darstellen, werden auch als Run-Unit bezeichnet.

SCHEMA



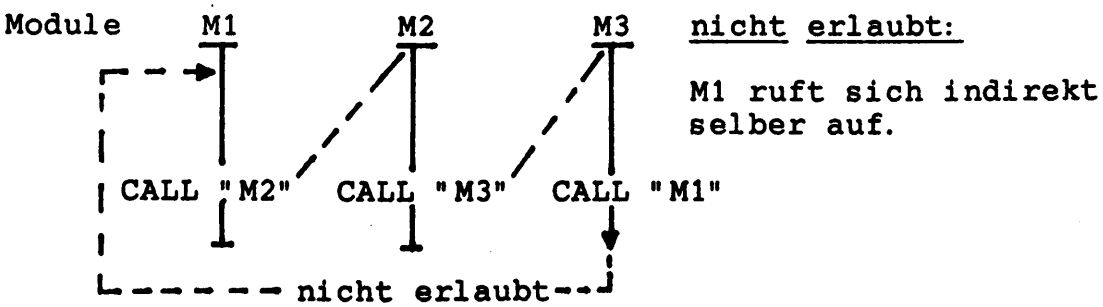
Ein Programm-Modul kann sowohl aufrufende wie aufgerufene Funktion haben. Dabei ist jedoch darauf zu achten, daß kein Modul direkt oder indirekt sich selber aufruft.

Beispiel:



erlaubt:

M2 hat zugleich aufrufende und aufgerufene Funktion.



nicht erlaubt:

M1 ruft sich indirekt selber auf.

Bemerkungen:

Tabellen: Ein Tabellen-Index (definiert mit INDEXED BY ...) kann nur im Modul angesprochen werden, in dem er definiert ist.

Dateien: Auf eine Datei kann nur im Modul zugegriffen werden, in dem sie definiert ist. Anweisungen wie OPEN, CLOSE, READ usw. müssen deshalb im gleichen Modul stehen wie der SELECT-Satz, der FD-Paragraph und die Satz-Definitionen.

Jede Datei muß mindestens vor dem ersten Zugriff der Run-Unit eröffnet und nach dem letzten Zugriff abgeschlossen werden. Eine Datei kann zum gleichen Zeitpunkt nur einmal pro Run-Unit mit demselben Dateinamen eröffnet sein.

Anweisungen:

In aufrufenden Modulen:

- Die CALL-Anweisung zum Aufrufen anderer Module.
- Die CANCEL-Anweisung zum Freigeben von Modulen.

In aufgerufenen Modulen:

- Die LINKAGE SECTION in der DATA DIVISION zur Beschreibung von Datenfeldern, die aus dem rufenden Modul übergeben wurden.
- Die erweiterte Kopfzeile der PROCEDURE DIVISION.
- Die Anweisung EXIT PROGRAM am logischen Ende des Moduls.

AUFRUFENDE MODULE

DIE CALL-ANWEISUNG

In Modulen mit nur aufrufender Funktion werden die IDENTIFICATION, ENVIRONMENT und DATA DIVISION wie üblich geschrieben. In der PROCEDURE DIVISION wird zum Aufrufen anderer Module die CALL-Anweisung verwendet.

Funktion:

Die CALL-Anweisung ruft ein Programm-Modul auf. Ihre logische Funktion ist mit der PERFORM-Anweisung vergleichbar: Wenn das logische Ende des aufgerufenen Moduls erreicht wird (EXIT PROGRAM-Anweisung), werden die der CALL-Anweisung folgenden Anweisungen ausgeführt.

Format 1:

```
CALL { Daten-Name } [ USING { [ BY REFERENCE ] Datenname-1 ... } ... ]
     { Literal     }        { BY CONTENT Datenname-1 ...      }
          [ ON OVERFLOW  unbedingte Anweisung ]
          [ END-CALL ]
```

FORMAT 2:

```
CALL { Daten-Name }
     { Literal    }
[ USING { [ BY REFERENCE ] Daten-Name-1 ... } ... ]
        { BY CONTENT Daten-Name-1 ...       }
  [ ON EXCEPTION unbedingte Anweisung 1 ]
  [ NOT ON EXCEPTION unbedingte Anweisung 2 ]
  [ END-CALL ]
```

SYNTAX:

- Das Literal bzw. der Daten-Name müssen alphanumerisch sein

- Die USING-Klausel wird benötigt, wenn das aufgerufene Modul auf die gleichen Datenfelder zugreifen soll wie das aufrufende Modul.

- Daten-Name-1 usw. müssen Daten-Namen mit Stufennummern 1 oder 77 aus der FILE-, WORKING-STORAGE oder LINKAGE SECTION sein. Derselbe Daten-Name kann mehrmals in einer CALL-Anweisung erscheinen. Daten-Namen aus der FILE SECTION dürfen qualifiziert werden.

BESCHREIBUNG:

Das Literal bzw. der Daten-Name bezeichnet den physischen Namen des aufzurufenden Programms auf der Platte, z. B. "PROG10".

Beim ersten CALL-Aufruf in einer Run-Unit oder nach einer CANCEL-Anweisung wird dieses Modul im Anfangszustand übernommen.

Jeder weitere CALL-Aufruf übernimmt das Modul im Zustand, wie es beim letzten Austritt (EXIT PROGRAM) bestanden hat. Dies betrifft alle Datenbereiche, den Status und die Positionierung von Dateien und alle veränderbaren Switch-Zustände.

Um ein Modul wieder in den Anfangszustand zu versetzen, wird die CANCEL-Anweisung benötigt.

Die USING-Klausel

Die Daten-Namen in der USING-Klausel bezeichnen die Datenbereiche, die im aufrufenden Modul definiert sind und auch im aufgerufenen Modul angesprochen werden können. Reihenfolge und Anzahl der Daten-Namen muß mit der Folge der Daten-Namen in der USING-Klausel der PROCEDURE DIVISION im aufgerufenen Modul übereinstimmen. Übernommene Datenfelder, die hier den Vorspann BY REFERENCE aufweisen, dürfen durch das gerufene Modul inhaltlich verändert werden. Felder, die hier den Vorspann BY CONTENT haben, stellen für das gerufene Modul Konstanten dar, die es nicht inhaltlich verändern darf.

Die OVERFLOW-Klausel

Stellt sich bei der Ausführung einer Anweisung CALL heraus, daß das in einem Datenfeld angegebene Programm nicht vorhanden ist, wird die unbedingte Anweisung in der ON OVERFLOW-Klausel ausgeführt. Bei weggelassener ON OVERFLOW-Klausel erfolgt ein Abbruch der gesamten Run-Unit.

Stellt sich bei der Ausführung der Anweisung CALL heraus, daß das als Literal angegebene Programm nicht vorhanden ist, erfolgt ein Abbruch der gesamten Run-Unit, unabhängig davon, ob die Klausel ON OVERFLOW programmiert worden ist oder nicht.

Die ON EXCEPTION-Klausel

Für diese Klausel gilt dasselbe wie das vorab für die OVERFLOW-Klausel Gesagte.

DIE CANCEL-ANWEISUNG

Funktion:

Die CANCEL-Anweisung gibt den Speicherbereich eines oder mehrerer früher aufgerufener Module wieder frei. Eine spätere CALL-Anweisung lädt die betreffenden Module in ihrem Ursprungszustand.

Format:

```
CANCEL { Daten-Name-1 }  [ { Daten-Name-2 } ]  ...
       { Literal-1    }    { Literal-2    }
```

SYNTAX:

- Die Literale-1, -2 usw. und Daten-Namen-1, -2 usw. müssen alphanumerisch sein.

BESCHREIBUNG:

Die Programm-Module mit den physischen Namen gemäß Literal-1, -2 usw., Daten-Name-1, -2 usw. werden freigegeben.

Keines der genannten Module darf zum Zeitpunkt der CANCEL-Anweisung mit CALL aufgerufen und noch nicht bis zum EXIT PROGRAM durchlaufen sein.

Falls das genannte Modul seit dem Programmstart noch nie mit CALL aufgerufen wurde, wird die CANCEL-Anweisung ignoriert; ebenso, falls das Modul bereits mit CANCEL freigegeben wurde.

AUFGERUFENE MODULE

In einem aufgerufenen Modul gelten für die IDENTIFICATION und ENVIRONMENT DIVISION keine besonderen Regeln. Zusätzliche Bestimmungen betreffen nur die DATA und PROCEDURE DIVISION:

- LINKAGE SECTION in der DATA DIVISION
- PROCEDURE DIVISION mit USAGE-Klausel
- EXIT PROGRAM-Anweisung

LINKAGE SECTION

Diese Section dient der Beschreibung aller Datenfelder, die vom rufenden Modul übernommen werden. Wenn keine solchen Felder bestehen, kann die LINKAGE SECTION weggelassen werden.

Die LINKAGE SECTION muß nach der FILE- und WORKING-STORAGE SECTION im Programm stehen.

Für die LINKAGE SECTION gelten dieselben Regeln wie für die WORKING-STORAGE SECTION mit folgender Ausnahme:

- Die VALUE-Klausel ist nicht erlaubt.

Für die Datenfelder der LINKAGE SECTION wird kein Speicherplatz zugeordnet. Da es sich um Bereiche handelt, auf die auch im aufrufenden Modul zugegriffen wird, sind sie alle in jenem Modul bereits enthalten. Die Angaben in der LINKAGE SECTION haben somit lediglich die Funktion von "Schablonen", welche beim Zusammenbauen der Module den Definitionen im aufrufenden Modul zugeordnet werden.

Die Daten-Namen der entsprechenden Felder im aufrufenden und im aufgerufenen Modul können gleich oder verschieden gewählt werden. Die Zuordnung erfolgt durch die USING-Klausel in der CALL-Anweisung (aufrufendes Modul) und in der PROCEDURE DIVISION (aufgerufenes Modul).

Bemerkung:

Arbeitsbereiche, die nur im aufgerufenen Modul angesprochen werden, sind in dessen WORKING-STORAGE SECTION oder GLOBAL SECTION zu definieren.

PROCEDURE DIVISION USING

Wenn in einem aufgerufenen Modul Daten angesprochen werden sollen, die im rufenden Modul definiert sind, wird die Kopfzeile der PROCEDURE DIVISION des aufgerufenen Moduls durch eine USING-Klausel erweitert.

Format:

```
PROCEDURE DIVISION [USING Daten-Name-1 [Daten-Name-2] ...]
```

Jeder Daten-Name in der USING-Klausel muß in der LINKAGE SECTION dieses Moduls mit einer Stufennummer 01 oder 77 definiert sein.

Ein bestimmter Daten-Name darf nur einmal in der PROCEDURE DIVISION USING-Klausel erscheinen.

Die USING-Klausel in der CALL-Anweisung des aufrufenden Moduls muß die gleiche Anzahl und Reihenfolge von Daten-Namen aufweisen und die entsprechenden Felder müssen von gleicher Länge sein.

Beispiel:

aufrufendes Modul:

```
CALL "MODULX" USING ARB  REC1  ZAHL.
```

aufgerufenes Modul MODULX:

```
PROCEDURE DIVISION USING A  R1  Z.
```

Die folgenden Daten-Namen bezeichnen physisch dieselben Bereiche mit derselben Länge:

1. ARB und A
2. REC1 und R1
3. ZAHL und Z

Die gemäß ihrer Reihenfolge einander entsprechenden Daten-Namen können verschieden oder gleich lauten. Die Zuordnung geschieht allein aufgrund der Reihenfolge in den USING-Klauseln.

Die Reihenfolge der Definitionen in der DATA DIVISION ist nicht von Bedeutung.

DIE EXIT PROGRAM-ANWEISUNG

Diese Anweisung bezeichnet das logische Ende eines aufgerufenen COBOL-Moduls. Sie bewirkt einen Rücksprung in das aufrufende Modul zur Anweisung nach der CALL-Anweisung.

Format:

```
EXIT PROGRAM.
```

Diese Anweisung muß unmittelbar auf einen Paragraphen-Namen folgen und die einzige Anweisung in diesem Paragraphen sein.

Falls die Anweisung durchlaufen wird, ohne daß das Modul mit CALL aufgerufen wurde, wirkt sie wie STOP RUN.

BEISPIELE EINES AUFRUFENDEN UND EINES AUFGERUFENEN MODULS:

Das aufrufende Modul (EINAUS)

```
IDENTIFICATION DIVISION.
 PROGRAM-ID.    EINAUS.
*            **** AUFRUFENDES MODUL MIT CALL ********.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. NCR-ITX.
OBJECT-COMPUTER. NCR-ITX.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
     SELECT EIN ASSIGN DISC.
     SELECT AUS ASSIGN DISC.

DATA DIVISION.
FILE SECTION.
FD EIN BLOCK 507 CHARACTERS
        RECORD 39
        LABEL RECORD STANDARD.
1  EINREC   PIC X(39).
FD AUS BLOCK 500 CHARACTERS
        RECORD 20
        LABEL RECORD STANDARD.
1  AUSREC   PIC X(20).
WORKING-STORAGE SECTION.
1  ZAHLARB.
 2 FILLER          PIC X(5)        VALUE HIGH-VALUE.
 2 ZTOTAL          PIC S9(5)V99    TRAILING SEPARATE.
 2 ANZAHL          PIC 999         VALUE 0.
1  TOTAL           PIC S9(5)V99    COMP   VALUE 0.
PROCEDURE DIVISION.
START.
     OPEN INPUT EIN OUTPUT AUS.
LESEN.
     READ EIN   AT END GO ENDE.
AUFRUF.
     CALL "RECHNEN" USING EINREC AUSREC TOTAL.
     MOVE TOTAL TO ZTOTAL.
     ADD 1 TO ANZAHL.
     WRITE AUSREC.
     GO TO LESEN.
ENDE.
     WRITE AUSREC FROM ZAHLARB.
     CLOSE EIN AUS.
     STOP RUN.
```

Das aufgerufene Modul (RECHNEN)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. RECHNEN.
*          *** AUFGERUFENES MODUL MIT LINKAGE SECTION *****.

ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.   NCR-ITX.
 OBJECT-COMPUTER.   NCR-ITX.

DATA DIVISION.
WORKING-STORAGE SECTION.
1  MELDUNG.
 2 MS              PIC X(5)BB.
 2 MBETRAG         PIC ----9.99.

LINKAGE SECTION.
1  TOTAL           PIC S9(5)V99  COMP.
1  EINARB.
 2 ESCHL           PIC X(5).
 2 F1              PIC 9999V99.
 2 F2              PIC 9999V99.
 2 F3              PIC 9999V99.
 2 AUSGABEN        PIC 9999V99.
 2 FILLER          PIC X(10).
1  AUSARB.
 2 ASCHL           PIC X(5).
 2 SUMME1          PIC 9(5)V99.
 2 SUMME2          PIC S9(5)V99 TRAILING SEPARATE.

PROCEDURE DIVISION USING EINARB, AUSARB, TOTAL.
P00.
     ADD F1 F2 F3 GIVING SUMME1.
     SUBTRACT AUSGABEN FROM SUMME1 GIVING SUMME2.
     IF SUMME2 POSITIVE GO P10.
     MOVE ESCHL TO MS.
     MOVE SUMME2 TO MBETRAG.
     DISPLAY MELDUNG.
P10.
     MOVE ESCHL TO ASCHL.
     ADD SUMME2 TO TOTAL.
ENDE.
     EXIT PROGRAM.
```

KAPITEL 13: SORTIEREN VON DATEIEN

ALLGEMEINES

Hierbei geht es um die COBOL-Leistung zum Sortieren von einer oder mehreren Dateien nach maximal 20 Schlüsseln.

FILE-CONTROL-EINTRAG FÜR DIE ARBEITSDATEI (WORK-FILE)

Zur Rolle der Arbeitsdatei: Der in 3 Phasen ablaufende SORT liest in der Phase 1 die Eingabedatei(en) und überträgt deren Sätze mit oder ohne Eingriff in die Arbeitsdatei, sortiert in der Phase 2 die in der Arbeitsdatei befindlichen Sätze und überträgt in der Phase 3 die sortierten Sätze mit oder ohne Eingriff auf die Ausgabedatei(en). Jede im Programm angewendete Anweisung SORT muß ihre eigene Arbeitsdatei aufweisen.

Format:

SELECT Datei-Name ASSIGN TO DISC.

KLAUSEL SAME SORT AREA IM I-O-CONTROL-PARAGRAPHEN

Da in einem Programm zwar mehrere Anweisungen SORT mit jeweils separaten Arbeitsdateien vorkommen, aber nur nacheinander aktiviert werden können, kann durch folgende Angabe dafür gesorgt werden, daß alle Arbeitsdateien denselben Speicherplatz benutzen:

```
SAME SORT AREA FOR Datei-Name-1 Datei-Name-2 ...
```

DIE EINTRAGUNG FÜR EINE ARBEITSDATEI IN DER FILE SECTION

```
SD Datei-Name
                ┌                                          ┐
                │  CONTAINS [Zahl-1 TO]                    │
                │           Zahl-2 CHARACTERS              │
RECORD          { IS VARYING IN SIZE                       }
                │    [[FROM Zahl-1]                        │
                │     [TO Zahl-2]  CHARACTERS]             │
                │     [DEPENDING ON Daten-Name-1]]         │
                └                                          ┘.

01    Satzbeschreibung ...
```

Diese Eintragung kommt der in diesem Handbuch bereits beschriebenen FD-Klausel gleich.

DIE COBOL-ANWEISUNG SORT

Format:

```
 SORT Datei-Name-1

{ [ ON { ASCENDING  } KEY Daten-Name-1 ... ] ... }
       { DESCENDING }

[WITH DUPLICATES IN ORDER]

[COLLATING SEQUENCE IS Alphabet-Name]

{ INPUT PROCEDURE IS Section-Name-1                    }
{       [{ THROUGH } Section-Name-2]                   }
{        { THRU    }                                   }
{ USING    Datei-Name-2  [Datei-Name-3] ...            }

{ OUTPUT PROCEDURE IS Section-Name-3                   }
{       [{ THROUGH } Section-Name-4]                   }
{        { THRU    }                                   }
{ GIVING Datei-Name-4                                  }
```

Datei-Name-1
  Hier ist der Name der Arbeitsdatei anzugeben.

ASCENDING/DESCENDING KEY
  Hier können bis zu 20 in der Satzbeschreibung der SD-Klausel definierte Schlüsselfelder angegeben werden sowie, ob aufsteigend oder absteigend sortiert werden soll.

COLLATING SEQUENCE
  Beim Wunsch, von der Sortierfolge gemäß dem NCR-Standard-

Code abzuweichen, ist hier ein entsprechender Alphabet-Name anzugeben. Siehe auch das für den Alphabet-Namen einschlägige Kapitel in diesem Handbuch.

USING
Hier können, falls keine SORT-Intervention (Eingriff) erwünscht ist, die Eingabedatei oder die Eingabedateien bestimmt werden.

INPUT PROCEDURE
Sollen zwischen der SORT-Phase 1 (Einlesen) und der SORT-Phase 2 (Sortieren) Eingriffe erfolgen (wie z. B. das Prüfen, Ändern und Weglassen von Sätzen), müssen in einer separaten Section der PROCEDURE DIVISION diese Eingriffe programmiert werden (siehe SORT-Beispiel 2). Der Name der betreffenden Section ist hier anzugeben.

GIVING
Wird vor Ausgabe der Sätze aus der Arbeitsdatei auf die Ausgabedatei keine Intervention (Eingriff) erwünscht und soll auch nur eine Ausgabedatei erstellt werden, ist hier der Name der Ausgabedatei anzugeben.

OUTPUT PROCEDURE
Sollen zwischen der SORT-Phase-2 (Sortieren) und der SORT-Phase-3 (Ausgeben) Eingriffe erfolgen (wie z. B. Prüfen, Ändern oder Weglassen von Sätzen bzw. Ausgabe auf nicht nur 1 Ausgabedatei), müssen in einer separaten Section der PROCEDURE DIVISION diese Eingriffe programmiert werden (siehe SORT-Beispiel 2). Der Name der betreffenden Section ist hier anzugeben.

DIE ANWEISUNG RELEASE

Format:

RELEASE Satzname der Arb. Datei [FROM {Literal / Identifier}]

Die Anweisung RELEASE kann nur in einer Section programmiert werden, deren Name als INPUT PROCEDURE in der Anweisung SORT angegeben wurde. Mit dieser Anweisung wird ein vorab eingelesener Satz (Identifier) oder ein Literal in die Arbeitsdatei übernommen (siehe Programmbeispiel 2).

DIE ANWEISUNG RETURN

Format:

```
RETURN Datei-Name RECORD
       [INTO Identifier]
  [AT END unbedingte Anweisung 1]
  [NOT AT END unbedingte Anweisung 2]
  [END-RETURN]
```

Die Anweisung RETURN kann nur in einer Section programmiert werden, deren Name als OUTPUT PROCEDURE in der Anweisung SORT angegeben wurde. Mit dieser Anweisung wird ein Satz aus der bereits fertig sortierten Arbeitsdatei z. B. in den Satz-Ausgabebereich der Ausgabedatei übernommen (Identifier), von wo er, falls gewünscht, mit einer WRITE-Anweisung auf die Ausgabedatei ausgegeben werden kann (siehe Programmbeispiel 2).

Im oben dargestellten Format der Anweisung RETURN ist als Datei-Name der Name der Arbeitsdatei, als Identifier z. B. der Ausgabebereich der Ausgabedatei, anzugeben.

Bei Datenende in der Arbeitsdatei wird die AT END-Verzweigung der Anweisung RETURN wirksam.

Beispiel 1: EIN BASIS-SORT-PROGRAMM

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SORTBEW.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.   NCR-ITX.
OBJECT-COMPUTER.   NCR-ITX.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
     SELECT BEW-ROH ASSIGN TO DISC.
     SELECT SORT-ARB ASSIGN TO DISC.
     SELECT BEW-OK ASSIGN TO DISC.

DATA DIVISION.
FILE SECTION.
FD   BEW-ROH BLOCK CONTAINS 39 RECORDS
             RECORD CONTAINS 13 RECORDS
             LABEL RECORD IS STANDARD.
01   BEW-ROH-SATZ     PIC X(13).

SD   SORT-ARB RECORD CONTAINS 13 CHARACTERS.
01   SORT-SATZ.
     02   KD-NR       PIC X(6).
     02   FILLER      PIC X(6).
     02   BEW-ART     PIC X.

FD   BEW-OK   BLOCK CONTAINS 39 RECORDS
              RECORD CONTAINS 13 CHARACTERS
              LABEL RECORD IS STANDARD.
01   BEW-OK-SATZ      PIC X(13).

PROCEDURE DIVISION.
ANFANG.
     SORT  SORT-ARB  ON ASCENDING KEY KD-NR  BEW-ART
         USING  BEW-ROH   GIVING  BEW-OK.
     STOP RUN.
```

Das Programm sortiert die Sätze einer Bewegungsdatei BEW-ROH in aufsteigender Folge nach der Kundennummer. Haben mehrere Sätze dieselbe Kundennummer, werden diese Sätze in aufsteigender Folge nach der Bewegungsart sortiert. Die Arbeitsdatei trägt den Namen SORT-ARB, die Ausgabedatei den Namen BEW-OK.

Beispiel 2: EIN SORT-PROGRAMM MIT EINER EINGABE-PROZEDUR "PRUEF-STATUS SECTION" UND EINER AUSGABE-PROZEDUR "PRUEF-SALDO SECTION".

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SORTEND.


ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. NCR-ITX.
OBJECT-COMPUTER. NCR-ITX.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT ROHDATEI ASSIGN TO DISC.
    SELECT SDATEI   ASSIGN TO DISC.
    SELECT ENDDATEI ASSIGN TO DISC.


DATA DIVISION.
FILE SECTION.

FD  ROHDATEI BLOCK CONTAINS 13 RECORDS
        RECORD CONTAINS 38 CHARACTERS
        LABEL RECORD IS STANDARD.
01  ROHSATZ.
    02  FILLER          PIC X(26).
    02  RECH-STATUS     PIC X.
    02  FILLER          PIC X(11).

FD  ENDDATEI BLOCK CONTAINS 13 RECORDS
        RECORD CONTAINS 38 CHARACTERS
        LABEL RECORD IS STANDARD.
01  EDSATZ.
    02  FILLER          PIC X(6).
    02  SALDO           PIC 9(5)V99.
    02  FILLER          PIC X(25).

SD  SDATEI  RECORD CONTAINS 38 CHARACTERS.
01  SRECORD.
    02  ENR             PIC X(6).
    02  FILLER          PIC X(32).
```

```
PROCEDURE DIVISION.
SORTIER SECTION.
BEGINN.
    SORT  SDATEI  ON ASCENDING KEY  ENR
        INPUT PROCEDURE IS  PRUEF-STATUS
        OUTPUT PROCEDURE IS  PRUEF-SALDO.
    STOP RUN.


PRUEF-STATUS SECTION.
P1. OPEN INPUT  ROHDATEI.
P2. READ ROHDATEI AT END GO TO ROHDAT-CLOS.
    IF RECH-STATUS IS = ZERO GO TO P2.
    RELEASE SRECORD FROM ROHSATZ.
    GO TO P2.
ROHDAT-CLOS.
    CLOSE ROHDATEI.


PRUEF-SALDO SECTION.
P3. OPEN OUTPUT ENDDATEI.
P4. RETURN SDATEI INTO EDSATZ  AT END GO TO ENDDAT-CLOS.
    IF SALDO IS = ZERO GO TO NULL-SALDO-AUSGABE.
W1. WRITE EDSATZ.
    GO TO P4.
NULL-SALDO-AUSGABE.
       ---------
    GO TO W1.
ENDDAT-CLOS.
    CLOSE ENDDATEI.
```

Für beide SORT-Beispiele gilt: Die Arbeits-Datei ist eine temporäre Datei, die nach der Sortierung wieder verfällt.

Aufruf des SORT-Beispiels 1:

```
AS BEW-ROH (n,DI),OW
AS SW1 SORT-ARB1 (n,DI),NEW,nnn,SC
AS SW2 SORT-ARB2 (n,DI),NEW,nnn,SC
AS BEW-OK (n,DI),NE,nnn
EX Programm-Name (n,DI)
```

Aufruf des SORT-Beispiels 2:

```
AS ROHDATEI (n,DI),OW
AS SW1 SORT-ARB1 (n,DI),NEW,nnn,SC
AS SW2 SORT-ARB2 (n,DI),NEW,nnn,SC
AS ENDDATEI (n,DI),NEW,nnn
EX Programm-Name (n,DI)
```

KAPITEL 14: MISCHEN VON DATEIEN

Hierbei geht es um die COBOL-Leistung Mischen von zwei oder mehreren Dateien nach maximal 20 Schlüsseln. Dabei wird davon ausgegangen, daß die zu mischenden Eingabedateien bereits sortiert sind. Das Mischen von Dateien wird ähnlich gehandhabt wie das Sortieren (siehe Kapitel 13).

Die Absätze des Kapitels 13

FILE-CONTROL-EINTRAG FÜR DIE ARBEITSDATEI,
KLAUSEL SAME SORT AREA IM I-O-CONTROL-PARAGRAPHEN,
DIE EINTRAGUNG FÜR EINE ARBEITSDATEI IN DER FILE SECTION

gelten auch hier unverändert.

Format:

```
MERGE Datei-Name-1

 {ON {ASCENDING } KEY   Daten-Name-1   ...} ...
     {DESCENDING}

 [COLLATING SEQUENCE IS Alphabet-Name]

  USING Datei-Name-2 Datei-Name-3   ...

 {OUTPUT PROCEDURE IS Section-Name-1
         [{THROUGH} Section-Name-2]
         {THRU    }
 {GIVING   Datei-Name-4             }
```

Die Anweisung RETURN ist bei programmierter OUTPUT-PROCEDURE wie bei der Sortierung anzuwenden (siehe Kapitel 13, Anweisung RETURN).

Beispiel eines Mischprogramms:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. MISCH-AB.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. NCR-ITX.
OBJECT-COMPUTER. NCR-ITX.

 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
    SELECT DATEI-A ASSIGN TO DISC.
    SELECT DATEI-B ASSIGN TO DISC.
    SELECT MISCHDAT ASSIGN TO DISC.
    SELECT DATEI-C ASSIGN TO DISC.

DATA DIVISION.
FILE SECTION.
FD  DATEI-A BLOCK CONTAINS 17 RECORDS
            RECORD CONTAINS 30 CHARACTERS
            LABEL RECORD IS STANDARD.
01  SATZ-A  PIC X(30).

FD  DATEI-B BLOCK CONTAINS 17 RECORDS
            RECORD CONTAINS 30 CHARACTERS
            LABEL RECORD IS STANDARD.
01  SATZ-B  PIC X(30).

SD  MISCHDAT RECORD CONTAINS 30 CHARACTERS.
01  MISCHSATZ.
    02  FILLER          PIC XX.
    02  KONTO-NUMMER    PIC X(8).
    02  FILLER          PIC X(20).

FD  DATEI-C BLOCK CONTAINS 17 RECORDS
            RECORD CONTAINS 30 CHARACTERS
            LABEL RECORD IS STANDARD.
01  SATZ-C  PIC X(30).


PROCEDURE DIVISION.
BEGIN.
    MERGE MISCHDAT ON ASCENDING KEY  KONTO-NUMMER
      USING DATEI-A DATEI-B   GIVING DATEI-C.
    STOP RUN.
```

Aufruf des Mischprogrammes:

```
AS DATEI-A (n,DI),OW
AS DATEI-B (n,DI),OW
AS SW1 MISCHDATX (n,DI),NE,nnn,SC
AS SW2 MISCHDATY (n,DI),NE,nnn,SC
AS DATEI-C (n,DI),NE,nnn
EX Programm-Name (n,DI)
```

ANHANG

Die Datei-Status-Werte (File Status Values):

FILE STATUS VALUE COBOL 85	FILE STATUS VALUE COBOL 74	STATUS OF INPUT-OUTPUT OPERATION	OPEN	CLOSE	READ	WRITE	REWRITE	DELETE	START
00	00	Successful completion	SRI	SRI	SRI	SRI	SRI	RI	RI
02	02	Duplicate key warning			I	I	I		
04	-	Wrong record length			SRI				
05	-	File not present	SRI						
07	-	File non reel/unit		S					
10	10	At end condition			SRI				
14	-	Wrong key size			R				
21	21	Sequence error					I		
22	22	Duplicate				RI	I		
23	23	No record found			RI		RI	RI	RI
24	24	Boundary violation				RI	I		
30	30	Permanent error due to: • hardware read error	SRI	SRI	SRI				
		• hardware write error		SRI		SRI	SRI	RI	
		• physical block length error			SRI				
34	34	Permanent error due to boundary violation				S			
35	90	File not available	SRI						
37	95	Invalid open mode	SRI						
38	90	File is locked	SRI						
39	95	Conflicting file attributes	SRI						
39	98	Conflicting file attributes	SRI						
39	9B	Conflicting file attributes	SRI						
41	91	File is open	SRI						
42	92	File not open		SRI					
43		Unsuccessful print READ					SRI	RI	
44	93	Boundary violation				SRI	SRI		
46	-	No next record found			SRI				
47		Invalid for input mode			SRI				RI
48		Invalid for output mode				SRI			
49		Invalid for I-O mode					SRI	RI	
-	90	File not available	SRI						
-	91	File not closed	SRI						
-	92	File not open		SRI	SRI	SRI	SRI	RI	RI
-	93	Illegal operation for open mode			SRI	SRI	SRI	RI	RI
-	94	Boundary violation			SRI				
-	95	Wrong attribute	SRI						
-	96	Device mismatch	SRI						
97	97	Invalid tape label	S						
-	98	Record length conflict	SRI		SRI	SRI	SRI		
99	99	Resource not available			SRI	SRI	SRI	RI	RI
9A	9A	Insufficient memory	SRI						
-	9B	File organization mismatch	S						
9C	9C	Power failure problem	S	S	S	S	S		
9D	9D	Clear trailer			S	S			
0E	0E	Tape mark			S				
0F	0F	Boundary warning			S	SRI	I		

Hinweis: Die unter der Spalte COBOL 85 angegebenen Werte sind Default. Die unter der Spalte COBOL 74 angegebenen Werte werden für solche Programme generiert, die mit den Optionen C 74 oder STATUS 74 compiliert werden (siehe unter COMPILATION auf den nächsten Seiten).

DIE COMPILATION

Der Aufruf einer Compilation erfolgt nach folgendem Format:

```
ASSIGN SI TO Ursprungsprogr. (n,DI),  Parameters

ASSIGN Library-Name

ASSIGN BO TO Objektprogr. (n,DI),  Parameters

ASSIGN LO TO {(n,LP)                                    }
             {Dateiname [/Generation] (n,DI), [Parameters]}

ASSIGN COBOLWORKA (n,DI),  Parameters

EXECUTE $COBOL85,  Compiler-Parameters
```

SI
: Die Angabe der Source-Input-Datei (Ursprungsprogramm) ist stets erforderlich.

Library-Name
: Name einer COBOL-Library-Datei, auf die durch eine COPY-Anweisung im Programm zugegriffen werden soll. Diese Zeile ist nur erforderlich, wenn die Library-Datei auf einer anderen Plattenstation liegt als das Ursprungsprogramm (SI).

BO
: Diese Angabe ist dann erforderlich, wenn ein Objektprogramm ausgegeben werden soll.

LO
: Diese Angabe bezüglich der Compilations-Liste ist dann erforderlich, wenn Druckausgabe oder Ausgabe auf eine Spooldatei erwünscht ist.

COBOLWORKA
: Diese Hilfsdatei ist als Scratch-Datei zuzuordnen, wenn das Ursprungsprogramm 2000 oder mehr Zeilen umfaßt. Weitere Workfiles (z. B. COBOLWORKB-Z oder COBOLWORK1-9) können zugeordnet werden.

Compiler-Parameters (Optionen)

Über diese Parameters (eine Auflistung finden Sie ab der nächsten Seite) kann Einfluß auf die Art der Compilation genommen werden. Die Parameters sind durch Kommas oder Spaces voneinander zu trennen. Reicht eine Zeile nicht aus, ist das Zeilenfortsetzungszeichen (!) anzuwenden.

Beispiel:

```
AS SI TO ABCPROG (3,DI),OW

AS LO TO DRUKTEST (1,LP)

AS BO TO TESTOBJ (3,DI),NE,350

AS COBOLWORKA (2,DI),NE,1560,SCR

EX $COBOL85 LIST
```

COMPILER-PARAMETERS (OPTIONEN)

Beispiel:

EX $COBOL85 NEST CUE NOINCLUDE

Nachfolgend werden die Compiler-Optionen für ITX-COBOL85 in alphabetischer Folge vorgestellt:

BOUNDS — Veranlaßt eine Bereichsgrößenprüfung für alle indizierten bzw. subskribierten Datenbereiche variabler Länge (Tabellen).

C74 — Wenn mit dem Compiler COBOL85 ein Programm entsprechend den Regeln von COBOL74 erstellt werden soll, ist dieser Parameter erforderlich.

CSEG=nnK — Die normale Code Segment Size ist 8 KB. Um eine andere Größe zu vereinbaren, wird der genannte Parameter verwendet. Hierbei kann nn einen maximalen Wert von 32 KB bekommen.

CUE — zeichnet die einzelnen Phasen eines Compilervorgangs am Bildschirm auf. Jeweils nach einer Phase wird eine Meldung ausgegeben, so daß der Compilationsstand verfolgt werden kann. Folgende Phasen werden durchlaufen:

1. Die LEX-Phase übersetzt die Ursprungs-Zeilen in "tokens", einen Code, der von den anschließenden Phasen verwendet werden kann.

2. Die ED-PARSE-Phase untersucht die "tokens" in bezug auf die Environment Division und exakter Syntax und Semantik.

3. Die DD-PARSE-Phase führt die gleiche Arbeit für den Bereich Data Division durch.

4. Die PD-PARSE-Phase untersucht die Anweisungen der Procedure Division, übersetzt

sie in einen "intermediate text", der vom Code-Generator verstanden wird.

5. Die OPTIMIZER Phase führt verschiedene lokale und globale Optimierungen aufgrund der in der OPT-Option angegebenen Optimierungsstufe durch.

6. Während der ALLOCATOR-Phase wird in den Daten-Segmenten der Object-Datei (BO) Platz für die Felder der Data Division reserviert.

7. In der CODE-GEN-Phase wird der "intermediate text" in den 9VM-Object-Code umgewandelt. Die Object-Datei (BO) wird formatiert und gegebenenfalls wird eine Ausgabeliste (LO) erstellt, sofern LIST oder LISTOBJ als Option spezifiziert wurde.

Es werden zusätzlich verschiedene statistische Daten ausgegeben wie:

Die Anzahl der Plattenzugriffe während der Applikations-Phase des Compilers (DISK), die Zeit in Sekunden, die zur Durchführung der Phase erforderlich waren (SECONDS), die Anzahl der CPU-Sekunden, die die Phase benötigte (CPU).

DEFEXT — Diese Option ist anzuwenden, um im Daten-Segment Platz für externe Datenfelder zu schaffen.

DSEG=nnK — Die Data Segment Size ist im Normalfall 4 KB (= Default) groß. Sie kann mit diesem Parameter bis zu 32 KB vergrößert werden (nn ist der KB-Wert).

DUMP — veranlaßt die Ausgabe eines Data-Dumps auf der Compiler-Liste.

DYNAMIC — Diese Option sorgt für dynamisches Procedure-Calling. Die Objekt-Programme brauchen nicht mit der LINK-Prozedur gebunden werden.

ENTRY=X	Diese Option gibt an, welches Programm (X) als erstes ausgeführt wird (nur bei "External" Programmen).
ESEG=nnK	Die External Segment Size ist normal 8 KB. Mit diesem Parameter kann sie von 1 bis 32 KB festgelegt werden.
HEAP=nnK	Es kann eine System-HEAP-Größe von 1 bis 32 K angegeben werden, die dazu dient, die defaultmäßige HEAP-Größe von 16 KB zu ändern. Der Parameter ist angebracht, wenn während der Compilation OUT OF MEMORY gemeldet wird.
ICMP	Mit diesem Parameter wird die Behandlung von nichtnumerischen Werten in numerischen Datenfeldern wie im Interpretive-COBOL-74 (RPOPS) festgelegt.
LIST	Erstellt eine Ursprungs-Liste mit Zeilennummern und virtuellen Adressen.
LISTOBJ	Es wird eine Ursprungs-Liste mit erweitertem Objekt-Code und einem Daten-Dump generiert. Stellt der Compiler während der Compilation einen Fehler fest, wird kein Objekt-Code generiert und damit auch keine Objekt-Code-Liste ausgedruckt.
LSEG=nKB	Wenn die Literal-Segment-Size nicht die Default-Größe von 4 KB haben soll, kann mit diesem Parameter eine Änderung vorgenommen werden. Es sind für n die Eingaben von 1 bis 4 möglich.
MAP	Generiert am Ende der Compiler-Liste eine Datenliste, die Auskunft über die Adresse im Speicher, Länge und Typ eines jeden Datenfeldes gibt.
MIGRATION	DISPLAY gibt bei der Anweisung DISPLAY A, B, C diese auf drei verschiedenen Zeilen, in Anlehnung an IMOS, aus. Wenn dieser Parameter fehlt, erfolgt die Ausgabe auf einer Zeile.
NCALL	ermöglicht die Verwendung von NCRL-Calls und ihren zugehörigen Parameter-Referenzen.

NEST — Um die im Programm vorhandenen Verknüpfungen (Nesting Level of Programs) mit der Source-Liste ausgedruckt zu erhalten, ist dieser Parameter zu verwenden.

NOCOB85 — Um die für COBOL85 reservierten Worte zu unterdrücken, wird dieser Parameter verwendet.

NOINCLUDE — Um zu verhindern, daß Zeilen von Copy-Dateien in der Source-Liste gedruckt werden, ist dieser Parameter zu verwenden.

NONCR — Um für NCR reservierte Worte zu unterdrücken, ist dieser Parameter erforderlich.

NOOBJ — Um die Generierung eines Objekt-Programms zu unterdrücken, kann dieser Parameter verwendet werden.

NOOPT — Wenn die OPTIMIZER Phase des Compilers ausgeschlossen werden soll, ist dieser Parameter zu wählen.

NORM — Um die RM/COBOL-reservierten Worte zu unterdrücken, ist dieser Parameter zu verwenden.

NORMDATA — Normalerweise sind RM/COBOL-Datentypen verwendbar. Wenn diese Datentypen nicht verwendet werden sollen, ist der Parameter NORMDATA einzusetzen.

NOUSAGE — Um die Fähigkeit zu unterdrücken, den USAGE-Typ zu verändern, ist der Parameter NOUSAGE zu verwenden.

NOWARN — Um Diagnose-Nachrichten zu unterdrücken, muß dieser Parameter benutzt werden.

OPT level — Um die Optimizer Phase zu beeinflussen, wird der hier angegebene Parameter verwendet. Es können die Stufen 1, 2 oder 3 angegeben werden (Default ist die Stufe 3). Die Stufe 1 optimiert die Operator-Eingabe-Muster und die Bool'schen Ausdrücke. Die Stufe 2 umfaßt die Stufe 1, optimiert darüber hinaus Konstanten und unterdrückt allgemeine Unterausdrücke in individuellen Anweisungen. Die Stufe 3 umfaßt

die Stufen 1 und 2, unterdrückt allgemeine Unterausdrücke zwischen Anweisungen, Sprungverknüpfungen und unerreichbare Zuweisungen.

PBCALL Diese Option gilt als Default und erzeugt eine (9VM)COBOL 74- und Interpretive-COBOL (RPOPS)-kompatible CALL-Sequenz für mittels CALL aufzurufende Programme und Parameter-Referenzen.

RMEXT Um die nachfolgenden Möglichkeiten aus dem RM/COBOL (Ryan-MacFarland COBOL) nutzen zu können, ist der angegebene Parameter erforderlich:

- Die USAGE-Klausel USAGE IS COMPUTATIONAL-1 (oder COMP-1), die ein 16 Bit langes, binäres Feld zur Verfügung stellt (-32768 bis +32767);
  die USAGE-Klausel USAGE IS COMPUTATIONAL (oder COMP), die 1 Byte pro Ziffer verfügbar macht;
  die USAGE-Klausel USAGE IS INDEX, die ein 2 Bytes langes Feld zur Verfügung stellt.
- zum Zweck der Dokumentation kann die VALUE OF LABEL Klausel verwendet werden.

STATIC veranlaßt die Compilation, dafür zu sorgen, daß alle Verarbeitungs-Calls statisch (d. h. nicht dynamisch) aufgerufen werden. Die so compilierten Programme müssen mit $LINK gebunden werden.

STATUS74 Ein Programm, das mit diesem Parameter compiliert worden ist, führt die I/Os nach den Regeln von COBOL74 aus.

SYMTBL Zum compilierten Programm werden die für den ITX Symbolic Debugger erforderlichen Tabellen geladen.

TRUNC Bei Verwendung dieses Parameters werden binäre Resultate entsprechend den in der PICTURE-Klausel angegebenen 9en dezimal abgeschnitten.

USAGE	Um die Veränderung des USAGE-Typs zu ermöglichen, ist dieser Parameter anzugeben.
XREF	generiert eine Cross-Reference-Liste als Teil der DATA-MAP auf der Compiler-Liste.

DIE COMPILER-LISTE

Die Compiler-Liste besteht aus 1 bis 7 Segmenten, abhängig von den Compiler-Optionen:

- o die Ursprungs-Programm-Liste
- o die erweiterte Objekt-Liste (eingestreut)
- o die Diagnose-Liste
- o die Daten-Bereichs-Liste (Data Area)
- o die Daten-Zusammenstellung (Data Map)
- o die Cross-Reference-Liste
- o die Compilations-Zusammenfassung (Compilation Summary)

Die Ursprungs-Programm-Liste

Sie beinhaltet die COBOL-Code-Zeilen. Bei Verwendung von COPY-Anweisungen werden auch die einkopierten Zeilen in die Liste mit aufgenommen. Jede Seite dieser Liste hat eine Kopfzeile, die alle wichtigen Informationen enthält. Nachfolgend werden einige Begriffe aus der Ursprungsprogramm-Liste erklärt.

Die Versionsnummer
Es werden zwei Versionsnummern angegeben. Die erste ist die Compiler-Versionsnummer, die den verwendeten Compiler identifiziert. Die zweite Versionsnummer ist die des Betriebssystems.

Die Versionsnummern haben das Format n.vv.rr+n.vv.rr.

n bezeichnet die Release-Nummer des Compilers bzw. des Betriebssystems, vv gibt die Update-Stufe der betreffenden Release an und rr bezeichnet die Korrektur-Stufe der betreffenden Release.

Die Ursprungs-Zeilen-Adresse
Jede Zeile in der Procedure Division eines Programms ist mit einer Adresse versehen. Sie sagt aus, an welcher Stelle der diesbezügliche Objekt-Code im Code-Segment des Objekt-Programms beginnt. Wenn der Code-Generator nicht ausgeführt worden ist, fehlen diese Adreßangaben. Die ausgedruckten Adressen können folgenden Zwecken dienen:

- o dem Suchen eines Fehlers
- o um Unterbrechungspunkte für den Debugger zu bestimmen
- o um festzustellen, wie umfangreich die Code-Generierung ist (als Hilfe zur Optimierung)

Diagnose
Der Diagnose-Bereich bildet ein eigenes Segment der Liste.

Folgenummern
Die ersten 6 Spalten der Liste beinhalten die laufende Zeilennummer.

Kommentarzeilen
Kommentarzeilen beginnen mit einem Stern in der Spalte 7. Außerdem werden alle Eintragungen in den Spalten 73 und höher als Kommentare aufgefaßt.

COPY-Zeilen
Die Zeilennummern der aus einer Datei oder einer Library einkopierten Zeilen stehen in spitzen Klammern.

Die erweiterte Objekt-Liste

Bei Verwendung des LISTOBJ-Parameters wird für jede Source-Zeile unmittelbar im Anschluß auf diese Zeile der Objekt-Code ausgedruckt.

Die Diagnose-Liste

Der Compiler ist in der Lage, mehr als tausend verschiedene Diagnose-Meldungen auszugeben. Sie geben genaue Auskunft über festgestellte formale Fehler und machen Vorschläge zur Vermeidung dieser Fehler.

Die Diagnose-Meldungen werden in vier Gruppen aufgeteilt:

Informative Nachrichten (INFORMATIONAL)
teilen dem Programmierer mit, daß es sinnvoll wäre, eine näher bezeichnete Änderung vorzunehmen. Einen Einfluß auf den Compilationsvorgang hat die Ursache für diese Nachricht jedoch nicht.

Warnung (WARNING)
Wenn der Compiler einen ungewöhnlichen Ursprungstext entdeckt, der nicht verwendbar ist, obwohl er den COBOL-Regeln nicht widerspricht, wird eine Warnung ausgegeben. Bei kleineren Verstößen gegen die Syntax gibt der Compiler ebenso eine Warnung aus. In diesem Fall versucht der Compiler selbst, den Fehler zu korri-

gieren oder er gibt eine Nachricht aus, die darüber informiert, welche Aktion zur Fehlerbeseitigung vorgenommen worden ist.

Formale Fehler (ERRORS)
Diese Nachrichten zeigen einen Verstoß gegen die Syntax bzw. die COBOL-Regeln an. Formale Fehler müssen beseitigt werden. Nur wenn ein Programm formal fehlerfrei ist, wird auch ein Objekt-Code generiert.

Die Meldung FATAL
Diese Meldungen sind selten. Sie zeigen an, daß der Compiler aufgrund eines Fehlers seinen Lauf abbrechen mußte.

Daten-Zusammenstellung

Diese Liste, die ausgegeben wird, wenn der MAP-Parameter verwendet wird, gibt die Adressen der Datenfelder im Objektprogramm einschließlich weiterer Informationen an:

- o die Adresse des Datenfeldes
- o die Ursprungs-Zeile, in welcher das Feld definiert worden ist
- o die Feldlänge in Bits oder Bytes
- o den Feldnamen
- o den Feld-Typ
- o Precision- und Scale-Faktoren

Cross-Reference-Liste

Sie wird ausgegeben, wenn der Parameter XREF verwendet wurde und sie befindet sich in der Daten-Zusammenstellung. Sie sagt aus, wo jeder einzelne Datenname, Oberbegriff und jedes einzelne COBOL-Wort innerhalb des Programms verwendet wird.

ÜBERLEGUNGEN ZUM COMPILERLAUF

Compiler-Option SYMTBL

Wenn beim Aufruf des mit dem Parameter SYMTBL compilierten Programms der Parameter SYMDEB mitgegeben wird, wird der Symbolic Debugger mit aufgerufen. Der Symbolic Debugger erlaubt die Programmprüfung auf der Stufe der Source-Zeilen.

Ein Abrufbeispiel hierfür ist:

```
EX obj-progr (nn,di) SYMDEB
```

Aufrufbare Hilfsmittel

Verschiedene System-Prozeduren und Programme, die mit anderen ITX-Sprachen erstellt und compiliert worden sind, können von einem COBOL85-Programm mit CALL sowohl statisch wie auch dynamisch aufgerufen werden.

Aufrufbare System-Prozeduren

Die CALL-Anweisung eines Programms kann die Kontrolle an eine Betriebssystem-Prozedur oder an ein System-Interface übergeben. Dies macht die Definition eines Satzes in der Working-Storage erforderlich, der die Parameter für das System-Interface beinhaltet. Ein System-Interface wird entweder durch ein £- oder $-Zeichen gekennzeichnet. Nachfolgend die von einem COBOL-Programm aufrufbaren Interfaces:

```
$FCOPY
$FLOAD
#DUMP-DIRECTORY
#AUDIT-LOG
#GET-PID
#GET-PROCESS-INFO
#GET-SCL-COMMAND-LINE
#GET-SCL-DISK-INFO
#GET-SCL-SYSTEM-STATUS
#GET-SYSTEM-DEPENDENT-INFO
#GET-UNBUNDLING-FLAG
#OUTPUT-AND-REPLY
#QUERYFILE
```

#SCL-DISPLAY-ERROR
#SCL-INTERPRETER
#TIME-OUT
#TRANSLATE
#UNIT-TO-VOLUME-SERIAL-NUMBER
#VOLUME-SERIAL-TO-UNIT-NUMBER
#PIO-CONFIGURE-LINE
#PIO-LOAD-CODE-TABLE
#PIO-GET-BOOT-CONFIGURATION
#PIO-GET-CURRENT-CONFIGURATION

Aufrufbare Object-Module

Ein Source-Programm, das mit dem COBOL85-Compiler umgewandelt wurde, darf CALL-Anweisungen auf Object-Module haben, die mit einem ITX-Compiler in ein 9VM-COBOL-Programm (Native COBOL), in ein C-, ein PASCAL- oder in ein Interpretativ-COBOL-Programm (RPOPS-COBOL) umgewandelt worden sind.

- Aufrufe auf C-Object-Module, compiliert mit $CC. Sie müssen das Schlüsselwort NCRL spezifizieren. Die COBOL-Module müssen mit NCALL und STATIC compiliert worden sein.

- Aufrufe auf PASCAL-Object-Module, compiliert mit $PASCAL. Die COBOL-Module müssen mit NCALL und STATIC compiliert worden sein.

- Aufrufe auf ITX-Interpretativ-COBOL-Object-Programme, compiliert mit $COBOL. Die COBOL-Module müssen mit DYNAMIC und PBCALL compiliert worden sein.

- Aufrufe auf ITX-Native-COBOL-Object-Programme, compiliert mit $COBOL9. Die nicht gebundenen COBOL-Module müssen mit DYNAMIC und PBCALL compiliert worden sein. Programm-Module, die gebunden sind, müssen mit STATIC und PBCALL compiliert worden sein.

Binden von Programmen

COBOL-Programme, die gebunden werden sollen, müssen zuvor mit der Compiler-Option STATIC compiliert worden sein. Nur dann können im gebundenen Programm CALLs ausgeführt werden. Der physische Dateiname des compilierten und zu bindenden Programms muß identisch sein mit dem Programmnamen in der IDENTIFICATION DIVISION. In der FLIST, der Liste der zu bindenden Programme, muß das Hauptprogramm an erster Stelle stehen. Zur Durchführung des Bindeprozesses sind die nachfolgenden SCL-Anweisungen erforderlich:

```
AS FLIST filename [/generation]
AS LO(LP)
AS BO filename [/generation]
EX $LINK
```

Diagnose-Nachrichten

Fehler, die während der COBOL85-Compilation auftreten, werden mittels Diagnose-Nachrichten gemeldet.

Der Compiler ist in der Lage, mehr als tausend verschiedene Nachrichten zu Fehlern mitzuteilen, die er bei der Verarbeitung entdeckt hat. Nachrichten werden auf den Bildschirm gesendet und in der Compiler-Liste vermerkt. Jede Nachricht bietet eine ausreichende Erklärung zu dem aufgetretenen Fehler.

Die Diagnosemeldungen werden in vier Gruppen aufgeteilt:

- INFORMATIONAL, reine Informationen, die keine Korrektur notwendig machen
- WARNING, sie machen nicht unbedingt eine Korrektur erforderlich; es könnte aber ein potentielles Problem vorliegen
- ERROR, sie verlangen eine Fehlerkorrektur
- FATAL, der Fehler ist so schwerwiegend, daß der Compiler abbricht.

Reserved Word List

The following is a list of COBOL reserved words. Reserved words new for COBOL 85 are marked with an asterisk (*). Reserved words and implementor names specific to NCR are marked with a plus sign (+). Reserved words special to Ryan-McFarland COBOL are marked with a dagger symbol (†). Separate lists follow the general list.

ACCEPT
ACCESS
ACK
ACTUAL-DATE +
ADD
ADVANCING
AFTER
ALL
ALPHABET + *
ALPHABETIC
ALPHABETIC-LOWER
ALPHABETIC-UPPER
ALPHANUMERIC *
ALPHANUMERIC-EDITED *
ALSO
ALTER
ALTERNATE
AND
ANY *
ARE
AREA
AREAS
ASCENDING
ASCII-CONTROL +
ASSIGN
AT
AUTHOR

BEEP + †
BEFORE
BINARY
BIT
BLANK
BLINK
BLOCK
BOTTOM
BY

CALL
CANCEL
CD
CF
CH
CHARACTER
CHARACTERS
CLASS
CLOCK-UNITS
CLOSE
COBOL
CODE
CODE-SET
COLLATING
COLUMN
COMMA
COMMON
COMMUNICATION
COMP
COMP-1
COMP-2
COMP-3
COMP-4
COMP-5
COMP-6
COMP-7
COMP-8
COMP-9
COMP-10
COMP-11
COMP-12
COMPUTATIONAL
COMPUTATIONAL-1
COMPUTATIONAL-2
COMPUTATIONAL-3
COMPUTATIONAL-4
COMPUTATIONAL-5
COMPUTATIONAL-6

COMPUTATIONAL-7
COMPUTATIONAL-8
COMPUTATIONAL-9
COMPUTATIONAL-10
COMPUTATIONAL-11
COMPUTATIONAL-12
COMPUTE
CONFIGURATION
CONTAINS
CONTENT *
CONTINUE *
CONTROL
CONTROLS
CONVERT
CONVERTING *
COPY
CORR
CORRESPONDING
COUNT
CURRENCY
CURSOR

DATA
DATE
DATE-COMPILED
DATE-WRITTEN
DAY
DAY-OF-WEEK * +
DE
DEBUG-CONTENTS
DEBUGGING
DEBUG-ITEM
DEBUG-LINE

DEBUG-NAME
DEBUG-SUB-1
DEBUG-SUB-2
DEBUG-SUB-3
DECIMAL-POINT
DECLARATIVES
DELETE
DELIMITED
DELIMITER
DEPENDING
DESCENDING
DESTINATION
DETAIL
DISABLE
DISPLAY
DIVIDE
DIVISION
DOWN
DUPLICATES
DYNAMIC

ECHO + †
EGI
ELSE
EMI
ENABLE
END
END-ADD *
END-CALL *
END-COMPUTE *
END-DELETE *
END-DIVIDE *
END-EVALUATE *
END-IF *
END-MULTIPLY *
END-OF-PAGE *
END-PERFORM *
END-READ *
END-RECEIVE *
END-RETURN *
END-REWRITE *
END-SEARCH *
END-START *
END-STRING *
END-SUBTRACT *
END-UNSTRING *
END-WRITE *
ENTER
ENVIRONMENT
EOP
EQUAL
EQUALS +
ERASE
ERROR
ESI
EVALUATE *
EVERY
EXACT +
EXCEEDS +
EXCEPTION
EXIT
EXOR +
EXTEND
EXTERNAL

FALSE *
FD
FILE
FILE-CONTROL
FILLER
FINAL
FIRST
FOOTING
FOR
FROM

GENERATE
GIVING
GLOBAL + *
GO
GREATER
GROUP
HEADING
HEXADECIMAL +
HIGH + †
HIGH-VALUE
HIGH-VALUES

I-O
I-O-CONTROL
IDENTIFICATION
IF
IN
INCOMPLETE +
INDEX
INDEXED
INDICATE
INITIAL
INITIALIZE *
INITIATE
INPUT
INPUT-OUTPUT
INSPECT
INSTALLATION
INTO
INVALID
INVERT +
IS

JCL-CODE +
JUST
JUSTIFIED

KEY

LABEL
LAST
LEADING
LEFT
LENGTH
LESS
LIMIT
LIMITS

LINAGE
LINAGE-COUNTER
LINE
LINE-COUNTER
LINES
LINKAGE
LOCK
LOW + †
LOW-VALUE
LOW-VALUES

MEMORY
MERGE
MESSAGE
MODE
MODULES
MONITOR-FLAG
MOVE
MULTIPLE
MULTIPLY

NATIVE
NCR-CONSTANT-STORAGE †
NCR-CRITICAL-DATA +
NCR-LOCK +
NCR-RECOVERY +
NCR-REPLY +
NCR-SAVE-DATA +
NCR-SHARED-STORAGE +
NCR-TIMER +
NCR-UNLOCK +
NCR-VRX-OUTCOME +
NEGATIVE
NEXT
NO
NOT
NUMBER
NUMERIC
NUMERIC-EDITED *

OBJECT-COMPUTER
OCCURS
OF
OFF
OMITTED
ON
OPEN
OPTIONAL
OR
ORDER *
ORGANISATION +
ORGANIZATION
OTHER *
OUTPUT
OVERFLOW
OVERLAP +

PACKED-DECIMAL
PADDING *
PAGE
PAGE-COUNTER
PERFORM
PF
PH
PIC
PICTURE
PLUS
POINTER
POSITION
POSITIONS +
POSITIVE
PRINTING
PROCEDURE
PROCEDURES
PROCEED
PROGRAM
PROGRAM-ID
PROMPT
PURGE * +

QUEUE
QUOTE
QUOTES

RANDOM
RD
READ
READ-ONLY +
RECEIVE
RECORD
RECORDS
REDEFINES
REEL
REFERENCE *
REFERENCES
RELATIVE
RELEASE
REMAINDER
REMOVAL
RENAMES
REPLACE *
REPLACING
REPORT
REPORTING
REPORTS
RERUN
RESERVE
RESET
RETRIEVAL +
RETURN
REVERSE + †
REVERSED
REWIND
REWRITE
RF
RH
RIGHT
ROUNDED
RUN

SAME
SD
SEARCH
SECTION
SECURITY
SEGMENT
SEGMENT-LIMIT
SELECT

SEND
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET
SHIFT
SIGN
SIZE
SORT
SORT-MERGE
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STANDARD-1
STANDARD-2 *
START
STATUS
STOP
STRING
SUB-QUEUE-1
SUB-QUEUE-2
SUB-QUEUE-3
SUBTRACT
SUM
SUPPRESS
SYMBOLIC
SYNC
SYNCHRONIZED

TABLE
TALLYING
TAPE
TERMINAL
TERMINATE
TEST * +
TEXT
THAN
THEN
THROUGH
THRU
TIME
TIMES
TO
TOP
TRAILING
TRANSPORT +
TRUE *
TYPE

UNEQUAL +
UNIT
UNLOCK + †
UNSTRING
UNTIL
UP
UPDATE +
UPON
USAGE
USE
USING

VALUE
VALUES
VARYING

WHEN
WITH
WORDS
WORKING-STORAGE
WRITE

ZERO
ZEROES
ZEROS
+
—
*
/
**
>
<
=
> =
< =

* COBOL 85 new reserved words
+ NCR reserved words
† RM reserved words

Reserved Words New in COBOL 85

ALPHABET
ALPHABETIC-LOWER
ALPHABETIC-UPPER
ALPHANUMERIC
ALPHANUMERIC-EDITED
ANY

BINARY

CLASS
COMMON
CONTENT
CONTINUE
CONVERTING
DAY-OF-WEEK

END-ADD
END-CALL
END-DELETE
END-DIVIDE
END-EVALUATE
END-IF
END-MULTIPLY
END-PERFORM
END-READ
END-RECEIVE
END-RETURN
END-REWRITE
END-SEARCH
END-START
END-STRING
END-SUBTRACT
END-WRITE
EVALUATE
EXTERNAL

FALSE

GLOBAL

INITIALIZE

NUMERIC-EDITED

ORDER
OTHER

PACKED-DECIMAL
PADDING
PURGE

REFERENCE
REPLACE

STANDARD-2

TEST
TRUE

NCR COBOL reserved words

ACTUAL-DATE
ALPHABET
ASCII-CONTROL

BEEP
BINARY
BIT
BLINK

CONVERT

DAY-OF-WEEK

ECHO
EQUALS
ERASE
EXACT
EXCEEDS
EXOR

GLOBAL

HEXADECIMAL
HIGH

INCOMPLETE
INVERT

JCL-CODE

LOW

NCR-CONSTANT-STORAGE
NCR-CRITICAL-DATA
NCR-LOCK
NCR-RECOVERY
NCR-REPLAY
NCR-SAVE-DATA
NCR-SHARED-STORAGE
NCR-TIMER
NCR-UNLOCK
NCR-VRX-OUTCOME

ORGANISATION
OVERLAP

POSITIONS
PROMPT
PURGE

READ-ONLY
RETRIEVAL
REVERSE

SHIFT

TEST
TRANSPORT

UNEQUAL
UNLOCK
UPDATE

Ryan-McFarland COBOL Reserved Words

BEEP
BINARY
BLINK

CONVERT

ECHO
ERASE

HIGH

LOW

PROMPT

REVERSE

UNLOCK

ASCII Character Set

D	O	H	C	D	O	H	C	D	O	H	C	D	O	H	C
0	000	00	NUL	32	040	20	space	64	100	40	@	96	140	60	`
1	001	01	SOH	33	041	21	!	65	101	41	A	97	141	61	a
2	002	02	STX	34	042	22	"	66	102	42	B	98	142	62	b
3	003	03	ETX	35	043	23	#	67	103	43	C	99	143	63	c
4	004	04	EOT	36	044	24	$	68	104	44	D	100	144	64	d
5	005	05	ENQ	37	045	25	%	69	105	45	E	101	145	65	e
6	006	06	ACK	38	046	26	&	70	106	46	F	102	146	66	f
7	007	07	BEL	39	047	27	'	71	107	47	G	103	147	67	g
8	010	08	BS	40	050	28	(	72	110	48	H	104	150	68	h
9	011	09	HT	41	051	29	)	73	111	49	I	105	151	69	i
10	012	0A	LF	42	052	2A	*	74	112	4A	J	106	152	6A	j
11	013	0B	VT	43	053	2B	+	75	113	4B	K	107	153	6B	k
12	014	0C	FF	44	054	2C	´	76	114	4C	L	108	154	6C	l
13	015	0D	CR	45	055	2D	-	77	115	4D	M	109	155	6D	m
14	016	0E	SO	46	056	2E	.	78	116	4E	N	110	156	6E	n
15	017	0F	SI	47	057	2F	/	79	117	4F	O	111	157	6F	o
16	020	10	DLE	48	060	30	0	80	120	50	P	112	160	70	p
17	021	11	DC1	49	061	31	1	81	121	51	Q	113	161	71	q
18	022	12	DC2	50	062	32	2	82	122	52	R	114	162	72	r
19	023	13	DC3	51	063	33	3	83	123	53	S	115	163	73	s
20	024	14	DC4	52	064	34	4	84	124	54	T	116	164	74	t
21	025	15	NAK	53	065	35	5	85	125	55	U	117	165	75	u
22	026	16	SYN	54	066	36	6	86	126	56	V	118	166	76	v
23	027	17	ETB	55	067	37	7	87	127	57	W	119	167	77	w
24	030	18	CAN	56	070	38	8	88	130	58	X	120	170	78	x
25	031	19	EM	57	071	39	9	89	131	59	Y	121	171	79	y
26	032	1A	SUB	58	072	3A	:	90	132	5A	Z	122	172	7A	z
27	033	1B	ESC	59	073	3B	;	91	133	5B	[	123	173	7B	{
28	034	1C	FS	60	074	3C	<	92	134	5C	\	124	174	7C	|
29	035	1D	GS	61	075	3D	=	93	135	5D	]	125	175	7D	}
30	036	1E	RS	62	076	3E	>	94	136	5E	^	126	176	7E	~
31	037	1F	US	63	077	3F	?	95	137	5F	_	127	177	7F	DEL

Stichwortverzeichnis

NCR GmbH
Postfach 10 00 90
8900 Augsburg 1
Telefon 08 21/40 51