



mitp

Uwe
Rozanski

COBOL

Altsysteme
warten und
erweitern

Eine praktische Einführung

Inhaltsverzeichnis

	Einleitung	13
1	Was unterscheidet COBOL von modernen, objektorientierten Sprachen?	15
1.1	Die Geschichte von COBOL	15
1.2	Fest definierter Sprachumfang	16
1.3	Prozedurale Programmierung	17
1.4	Linearer Programmablauf	18
1.5	Datenfelder mit fester Länge	19
1.6	Module statt Instanzen	20
2	Programmstruktur und grundlegende Sprachelemente	23
2.1	COBOL-Programmstruktur	23
2.1.1	Die Bedeutung der Programmteile (DIVISIONs)	24
2.1.2	Die Hierarchie in einem COBOL-Programm	24
2.1.3	Das COBOL-Programm im Überblick	25
2.2	COBOL-Sprachelemente	26
2.2.1	Reservierte Wörter	26
2.2.2	Programmiererwörter	27
2.2.3	Literale	27
2.2.4	Figurative Konstanten	30
2.2.5	Trennsymbole	32
2.2.6	Operatoren	36
2.2.7	Sonderregister	37
2.3	COBOL-Zeichensatz	37
2.4	Interpretation der COBOL-Klausel- und -Anweisungsformate	38
2.5	Das Codierformat	39
2.5.1	Fixed-form reference format	39
2.5.2	Free-form reference format	42
3	Bedeutung der 4 DIVISIONs	45
3.1	IDENTIFICATION DIVISION	45
3.2	ENVIRONMENT DIVISION	47
3.2.1	CONFIGURATION SECTION	47
3.2.2	INPUT-OUTPUT SECTION	57
3.2.3	FILE-CONTROL	57
3.2.4	I-O-CONTROL	57

3.3	DATA DIVISION	57
3.3.1	FILE SECTION	58
3.3.2	WORKING-STORAGE SECTION	58
3.3.3	LOCAL-STORAGE SECTION	58
3.3.4	LINKAGE SECTION	59
3.4	PROCEDURE DIVISION	59
3.4.1	USING-Zusatz	60
3.4.2	RAISING-Zusatz	60
3.4.3	DECLARATIVES (Sondervereinbarungen)	60
3.4.4	END PROGRAM	60
3.4.5	Aufbau der PROCEDURE DIVISION	61
4	Definitionen von Datenfeldern	63
4.1	Stufennummer 77	63
4.2	PICTURE-Klausel	64
4.2.1	Alphabetische Datenfelder	65
4.2.2	Alphanumerische Datenfelder	66
4.2.3	Numerische Datenfelder	67
4.2.4	Boolesche Datenfelder	69
4.2.5	Alphanumerische druckaufbereitete Datenfelder	70
4.2.6	Numerische druckaufbereitete Datenfelder	71
4.3	VALUE-Klausel	80
4.4	USAGE-Klausel	80
4.4.1	DISPLAY	81
4.4.2	PACKED-DECIMAL (manchmal auch COMP-3)	82
4.4.3	COMP oder BINARY	82
4.4.4	BINARY-CHAR, BINARY-SHORT, BINARY-LONG und BINARY-DOUBLE	83
4.4.5	FLOAT-SHORT	85
4.4.6	FLOAT-LONG	85
4.4.7	FLOAT-EXTENDED	85
4.4.8	INDEX	86
4.4.9	NATIONAL	86
4.4.10	OBJECT REFERENCE	86
4.4.11	POINTER	86
4.4.12	PROGRAM POINTER	87
4.5	BLANK WHEN ZERO-Klausel	87
4.6	JUSTIFIED	88
4.7	SYNCHRONIZED-Klausel	89
4.8	SIGN-Klausel	89

5	Definitionen von Datenstrukturen und Datensätzen	91
5.1	Stufennummern 01 bis 49	91
5.2	REDEFINES-Klausel	94
5.3	Stufennummer 88.	96
5.4	Stufennummer 66.	98
5.5	Datengruppen mit BIT-Feldern	100
5.6	Konstante	100
6	Feldzuweisungen im Hauptspeicher	103
6.1	MOVE-Anweisung.	103
6.1.1	MOVE CORRESPONDING-Anweisung	107
6.2	INITIALIZE-Anweisung.	109
6.2.1	TO VALUE-Angabe	111
6.2.2	Ohne den Zusatz REPLACING	111
6.2.3	Mit dem Zusatz REPLACING	112
6.2.4	INITIALIZE für Tabellen	113
6.3	SET-Anweisung	113
6.4	Referenz-Modifikation	117
7	Die Anweisungen ACCEPT, DISPLAY und STOP	119
7.1	DISPLAY-Anweisung	119
7.1.1	Erweiterte Eigenschaften der DISPLAY-Anweisung	121
7.1.2	Löschen des Bildschirms	123
7.1.3	Vorpositionierung des Cursors	123
7.2	ACCEPT-Anweisung.	123
7.2.1	Gleichzeitige Eingabe in mehrere Felder	125
7.2.2	Eingabe in numerische Felder	126
7.2.3	ACCEPT-Anweisung und CURSOR-Klausel.	126
7.2.4	ACCEPT-Anweisung und CRT STATUS-Klausel.	129
7.2.5	Spezielle Dialogtechniken mit DISPLAY und ACCEPT.	132
7.3	ACCEPT-Anweisung Format 2.	132
7.4	STOP-Anweisung	134
8	Arithmetische Operationen	137
8.1	COMPUTE-Anweisung.	137
8.1.1	Der ROUNDED-Zusatz.	139
8.1.2	Der ON SIZE ERROR-Zusatz.	140
8.1.3	Der NOT ON SIZE ERROR-Zusatz.	141
8.1.4	Überlauf bei mehreren Ergebnisfeldern	141
8.2	ADD-Anweisung	142
8.3	SUBTRACT-Anweisung	144
8.4	Korrespondierendes Addieren und Subtrahieren.	146

8.5	MULTIPLY-Anweisung	146
8.6	DIVIDE-Anweisung	148
8.6.1	Rest der Division	150
9	Programmverzweigungen und interne Unterprogramme	153
9.1	GO TO-Anweisung	153
9.2	GO TO ... DEPENDING ON	156
9.3	Vorbemerkung zu internen Unterprogrammen	159
9.4	PERFORM-Anweisung	160
9.5	EXIT-Anweisung	176
9.6	EXIT PERFORM-Anweisung	177
9.7	EXIT SECTION-Anweisung	178
10	Die Anweisungen IF und EVALUATE	181
10.1	IF-Anweisung	181
10.1.1	Vorzeichenbedingung	191
10.1.2	Klassenbedingung	192
10.1.3	Bedingungsnamen-Bedingung	193
10.1.4	Abfragen eines Pointers	195
10.1.5	Zusammengesetzte Bedingungen	195
10.2	CONTINUE-Anweisung	199
10.3	EVALUATE-Anweisung	200
11	Quellcode wiederverwenden mit COPY	209
11.1	COPY-Anweisung	209
11.2	COPY-Bibliotheken	211
11.2.1	REPLACING-Zusatz	212
11.2.2	SUPPRESS-Angabe	213
11.3	REPLACE-Anweisung	213
12	Externe Unterprogramme	215
12.1	Sprachelemente für Unterprogramm-Technik	215
12.2	Die Programmverbindung	216
12.3	CALL-Anweisung	217
12.4	LINKAGE SECTION	219
12.5	USING-Zusatz der PROCEDURE DIVISION	221
12.5.1	BY REFERENCE	222
12.5.2	BY REFERENCE ADDRESS OF	223
12.5.3	BY CONTENT	225
12.5.4	BY CONTENT LENGTH OF	225
12.5.5	BY VALUE	226

12.6	EXIT PROGRAM-Anweisung	226
12.7	Rekursive COBOL-Programme	228
12.8	CANCEL-Anweisung	231
12.9	GOBACK-Anweisung	232
12.10	Weitere Angaben zur Programmkommunikation	232
12.11	EXTERNAL-Klausel	233
12.12	Schachtelung von Programmen	235
	12.12.1 Die Schachtelungsebene eines Unterprogramms	236
	12.12.2 Der Aufruf eines geschachtelten Unterprogramms	236
12.13	GLOBAL-Klausel	237
12.14	INITIAL-Klausel	238
13	Tabellenverarbeitung	239
13.1	OCCURS-Klausel	239
	13.1.1 Definition einer eindimensionalen Tabelle	240
	13.1.2 Adressierung von Elementen einer Tabelle	241
	13.1.3 Definition einer mehrdimensionalen Tabelle	243
	13.1.4 Adressierung von mehrdimensionalen Tabellen	244
13.2	Normalindizierung (Subskribierung)	246
13.3	Spezialindizierung	248
	13.3.1 INDEXED BY-Zusatz	248
	13.3.2 Vorteile der Spezialindizierungsmethode	248
	13.3.3 Die relative Adresse im Spezialindex	250
	13.3.4 Relative Spezialindizierung	251
	13.3.5 DEPENDING ON-Zusatz	251
	13.3.6 SET-Anweisung	251
	13.3.7 USAGE INDEX-Klausel	253
13.4	Vergleich zwischen Normal- und Spezialindizierung	254
13.5	Initialisieren von Tabellen	256
	13.5.1 VALUE-Klausel	256
	13.5.2 REDEFINES-Klausel	256
13.6	Sequenzielles Durchsuchen einer Tabelle mit der SEARCH-Anweisung	258
	13.6.1 VARYING-Zusatz	258
	13.6.2 AT END-Zusatz	259
	13.6.3 WHEN-Zusatz	259
	13.6.4 CONTINUE	259
	13.6.5 Durchsuchen einer mehrdimensionalen Tabelle	260
13.7	Binäres Durchsuchen einer Tabelle	261
	13.7.1 ASCENDING/DESCENDING KEY-Zusatz	262
	13.7.2 Sortieren einer Tabelle mit der SORT-Anweisung	262
	13.7.3 SEARCH ALL-Anweisung	263

14	Verarbeiten von Zeichenketten	267
14.1	INSPECT-Anweisung	267
14.1.1	Zählen mit TALLYING	269
14.1.2	BEFORE und AFTER	270
14.1.3	Ersetzen mit REPLACING	271
14.1.4	Konvertieren mit CONVERTING	273
14.2	STRING-Anweisung	275
14.3	UNSTRING-Anweisung	278
15	Sequenzielle Dateien	283
15.1	Eintragungen in der ENVIRONMENT DIVISION	284
15.1.1	SELECT-Klausel	284
15.1.2	ASSIGN-Klausel	285
15.1.3	ORGANIZATION-Klausel	287
15.1.4	ACCESS MODE-Klausel	287
15.1.5	FILE STATUS-Klausel	287
15.1.6	Sonstige Klauseln	290
15.2	Eintragungen in der DATA DIVISION	291
15.2.1	RECORD CONTAINS-Klausel	291
15.2.2	BLOCK CONTAINS-Klausel	293
15.2.3	LINAGE-Klausel	293
15.2.4	CODE-SET-Klausel	294
15.2.5	Datensatzbeschreibung	295
15.3	Anweisungen in der PROCEDURE DIVISION	295
15.3.1	OPEN-Anweisung	296
15.3.2	READ-Anweisung	298
15.3.3	WRITE-Anweisung	300
15.3.4	REWRITE-Anweisung	304
15.3.5	CLOSE-Anweisung	305
15.3.6	USE-Anweisung	306
16	Index-sequenzielle Dateiorganisation	311
16.1	Eintragungen in der ENVIRONMENT DIVISION	311
16.2	Eintragungen in der DATA DIVISION	318
16.3	Anweisungen in der PROCEDURE DIVISION	318
16.3.1	OPEN-Anweisung	318
16.3.2	READ-Anweisung	320
16.3.3	WRITE-Anweisung	323
16.3.4	REWRITE-Anweisung	324
16.3.5	DELETE-Anweisung	327
16.3.6	START-Anweisung	328

16.3.7	CLOSE-Anweisung	329
16.3.8	Zulässige E/A-Anweisungen	330
16.3.9	USE-Anweisung	330
16.4	Alternative Schlüssel für Index-sequenzielle Dateien	331
16.4.1	ALTERNATE RECORD KEY-Klausel	331
16.4.2	ALTERNATE KEY in der READ-Anweisung	331
16.4.3	ALTERNATE KEY in der START-Anweisung	333
17	SORT-MERGE-Modul	335
17.1	Die SELECT-Klausel für Sortierdateien	335
17.2	Die SD-Stufenbezeichnung	336
17.3	SORT-Anweisung	337
17.3.1	ASCENDING/DESCENDING KEY	337
17.3.2	Automatische E/A-Operationen	338
17.3.3	INPUT PROCEDURE	338
17.3.4	OUTPUT PROCEDURE	339
17.3.5	COLLATING SEQUENCE	339
17.3.6	WITH DUPLICATES IN ORDER	339
17.3.7	Format 2	339
17.4	RELEASE-Anweisung	340
17.5	RETURN-Anweisung	340
17.6	MERGE-Anweisung	341
18	COBOL und Datenbanken (IMS, SQL)	343
18.1	Die hierarchische Datenbank IMS	343
18.1.1	Aufbau einer IMS-Datenbank	343
18.1.2	Beschreibung der Datenbank mittels DBD	344
18.1.3	Logische Datenbankstruktur mit PSB beschreiben	345
18.1.4	Definition eines PCB in COBOL	346
18.1.5	Programmeinsprung mittels ENTRY	346
18.1.6	Notwendige Ein-/Ausgabebereiche	347
18.1.7	Datenbankzugriff programmieren	348
18.1.8	Auswerten des Statuscodes	352
18.1.9	Laden einer IMS-Datenbank	353
18.2	Arbeiten mit relationalen Datenbanken	354
18.2.1	Aufbau einer relationalen Datenbank	354
18.2.2	Beschreibung der Datenbank mittels SQL	355
18.2.3	Der SQL Precompiler	355
18.2.4	Erstellen der COBOL-Struktur mittels DCLGEN	355
18.2.5	BIND und REBIND	357
18.2.6	Typischer Programmaufbau	358

18.2.7	Fehlerbehandlung über die SQLCA.	359
18.2.8	Programmieren von statischen SQL-Anweisungen	361
18.2.9	Programmieren von dynamischen SQL-Anweisungen	364
19	COBOL und CICS	375
19.1	CICS-Kommandoformat.	375
19.2	COBOL-Einschränkungen	377
19.3	Erstellung von CICS-Programmen	377
19.3.1	Strukturierte Programmierung	377
19.3.2	Conversational Processing	378
19.3.3	Pseudoconversational Processing	379
19.3.4	TRANSID.	379
19.3.5	COMMAREA.	379
19.3.6	EXECUTION INTERFACE BLOCK.	380
19.3.7	RETURN-Kommando.	380
19.4	Fehlerbehandlung unter CICS.	382
19.4.1	Statuscode EIBRESP abfragen	382
19.4.2	HANDLE CONDITION-Kommando.	383
19.4.3	EXECUTE INTERFACE BLOCK	385
19.5	Eigene Unterprogramme aufrufen	387
19.5.1	Programmkontrolle.	387
19.5.2	LINK und XCTRL	387
19.5.3	RETURN	388
19.5.4	Ausnahmebedingungen für LINK, XCTL und RETURN.	389
19.5.5	Daten über die COMMAREA übergeben	390
19.6	IBM-3270-Geräte – Bildschirmsteuerung	391
19.6.1	Basic Mapping Support BMS	391
19.6.2	Masken senden und empfangen	392
19.7	Dateiverarbeitung	396
19.7.1	Unterstützte Formate	396
19.7.2	Lesen mit READ	397
19.7.3	Schreiben mit WRITE.	398
19.7.4	REWRITE, DELETE und UNLOCK.	399
19.7.5	Dateien sequenziell lesen	400
19.7.6	Ausnahmebedingungen	402
	Stichwortverzeichnis.	405



Einleitung

Die Programmiersprache COBOL gibt es seit etwa 1960 und COBOL steht ausgeschrieben für *Common Business Oriented Language*, also für eine Programmiersprache zur Lösung von kaufmännischen Problemen.

Aufgrund des Alters und der Zielrichtung dieser Sprache ist es nicht verwunderlich, dass es sehr viele Implementierungen auf den unterschiedlichsten Plattformen gibt und diese bis heute noch in Funktion sind.

Auch wenn es heutzutage keine großen Neuentwicklungen mehr geben dürfte, die in COBOL stattfinden, spielt die Wartung der immer noch produktiv laufenden Systeme eine wichtige Rolle. Allerdings gibt es immer weniger geschultes oder gar erfahrenes Personal, das diese Wartung übernehmen könnte. Moderne, oft objektorientierte Programmiersprachen stehen heute im Fokus von Aus- und Weiterbildung und nicht so eine prozedurale Sprache wie COBOL.

Dieses Buch bemüht sich, alle Aspekte von COBOL so zu erklären, dass sie jeder versteht, der zwar programmieren kann, aber noch nicht mit COBOL gearbeitet hat. Die einzelnen Kapitel sind thematisch geordnet. Das erleichtert den Einstieg in diese Programmiersprache, aber vor allem auch die zielgerichtete Suche nach einem Thema, das Sie gerade benötigen.

COBOL ist eine sehr geschwätzige Programmiersprache, meint, der Quellcode ist oft recht umfangreich, dafür aber auch leicht zu lesen. Was in modernen Sprachen oft mit einem Methodenaufruf oder der Verwendung einer Funktion erledigt ist, wird in COBOL durch spezifische Befehle programmiert, die oft über mehrere Zeilen gehen. Programme mit 10.000 Zeilen und mehr sind keine Seltenheit und in COBOL durchaus üblich.

In der Praxis werden Sie aber selten reine COBOL Programme finden, also solche, die komplett mit dem Befehlsumfang dieser Sprache auskommen. Oft sind es Programme, die Datenbanken bearbeiten müssen und für diesen Zweck gibt es keine COBOL-Befehle. Neben relationalen Datenbanken gibt es auch hierarchische Datenbanksysteme wie IMS, die vor allem in älteren Implementierungen vorkommen können. Dem Thema COBOL und Datenbanken ist daher ein eigenes Kapitel gewidmet.

Auch auf Host-Systemen hat man schon sehr frühzeitig begonnen, Anwendungen zu entwickeln, mit denen Benutzer interagieren können, sogenannte *Onlineanwendungen* mit textbasierten Terminals. CICS spielt hier eine wichtige Rolle, vor allem, weil es auch für die Steuerung von Industriemaschinen verwendet wurde. Will man ein solches Pro-

gramm warten, muss man verstehen, wie es tickt. Auf die Besonderheiten der CICS-Programmierung unter COBOL geht daher ebenfalls ein eigenes Kapitel ein.

Die klare Zielrichtung dieses Buches ist, es erfahrenen Programmierern zu ermöglichen, auch ältere COBOL-Programme zu pflegen und zu erweitern, auch wenn sie mit Datenbank- oder Onlinesystemen arbeiten.

Was Sie in diesem Buch erwartet

Das vorliegende Buch behandelt alle COBOL-Definitionen und -Befehle, ohne sich auf einen bestimmten Dialekt oder einen spezifischen Hersteller zu beschränken.

Wer eine bestehende COBOL-Anwendung warten muss, hat es mit einem bestimmten COBOL-Compiler und Dialekt zu tun. Es kann nicht garantiert werden, dass der gesamte Umfang dieses Dialekts hier beschrieben ist, oder umgekehrt, dass alles, was hier erklärt wird, auch mit dem verwendeten Dialekt funktioniert. Sie werden aber mit Sicherheit genügend Informationen vorfinden, um die täglichen Herausforderungen zu bewältigen und die vorgefundenen Anweisungen verstehen zu können.

Absichtlich nicht behandelt werden in diesem Buch die objektorientierten Erweiterungen von COBOL. Tatsächlich ist es möglich, vollwertige, objektorientierte Programme in COBOL zu schreiben, inklusive eigener Klassendefinitionen mit Methoden und Attributen. Eigene COBOL-Klassen von bestehenden Klassen abzuleiten und Methoden zu überladen, ist vollständig implementiert. Hier alle Möglichkeiten aufzuzeigen, ist ein eigenes Buch wert und würde an dieser Stelle den Umfang sprengen.

Die objektorientierte Spracherweiterung kam zu spät. Zu dieser Zeit hatten ganz andere Programmiersprachen wie C++ oder Java bereits so viele Anhänger, dass sich kaum jemand für OO-COBOL interessiert hat. Ziel dieses Buches ist es, erfahrene Entwickler in die Lage zu versetzen, bestehende COBOL-Programme zu pflegen und diese dürften kaum objektorientiert sein. Viel wahrscheinlicher ist es, dass es sich um CICS-Programme und/oder um Anwendungen handelt, die mit Datenbanken arbeiten. Daher wurden die bereits erwähnten Kapitel in das Buch mit aufgenommen.

DOWNLOADS zum Buch

Unter <https://mitp.code-load.de> finden Sie zwei umfangreiche Bonuskapitel zu den Themen »Intrinsic-Funktionen« sowie »Konkurrierende Dateizugriffe« zum kostenlosen Download.

Was unterscheidet COBOL von modernen, objektorientierten Sprachen?

1.1 Die Geschichte von COBOL

An dieser Stelle soll nicht bis ins kleinste Detail beschrieben werden, an welchem Tag welches Feature in die Programmiersprache COBOL aufgenommen wurde, vielmehr soll ein grober Überblick über die Entstehung dieser Sprache und die verschiedenen Standards gegeben werden. In Tabelle 1.1 findet sich eine chronologische, wenn auch oberflächliche Aufstellung der Entstehungsgeschichte.

Jahr	Entwicklungsschritt
1960	Unter dem Namen COBOL-60 wurde eine erste Version von COBOL von einem Gremium mit den Namen CODASYL verabschiedet. Dieses Gremium bestand aus Vertretern von Regierung, Militär und Privatwirtschaft der USA und hat es sich zur Aufgabe gemacht, eine gemeinsame Programmiersprache zu entwickeln, die auf unterschiedlichen Computersystemen lauffähig sein sollte.
1961	Die Programmiersprache wurde komplett überarbeitet und war nicht abwärtskompatibel. Aus dieser Erfahrung heraus hat man festgelegt, künftig dafür zu sorgen, dass ältere Programme mit neuen Compiler-Versionen noch übersetzt werden können. In diesem Jahr wurden der COBOL-SORT und REPORT WRITER aufgenommen. Es handelt sich dabei um Module, um Dateien zu sortieren und Auswertungen zu erstellen.
1965	Definition von Tabellen und die Möglichkeit, Dateien zu bearbeiten, haben den Sprachumfang erweitert.
1968	In diesem Jahr wurde der ANSI-Standard X3.23-1968 für COBOL verabschiedet und seither ständig weiterentwickelt.
1974	Einzug der strukturierten Programmierung bestehend aus internen Unterprogrammen
1985	Unter der Bezeichnung COBOL-85 wurden beispielsweise die Begrenzer END-IF und END-PERFORM eingeführt. Auch die internen Funktionen wurden zu dieser Zeit definiert.
2002	Die Verarbeitung von Unicode und die objektorientierte Programmierung wurden aufgenommen.

Tabelle 1.1: Grobe Entstehungsgeschichte

Wichtig zu wissen ist, dass es sich bei COBOL um eine standardisierte Programmiersprache handelt und es unterschiedliche Hersteller von Compilern gibt, die natürlich auch immer eigene Erweiterungen eingebracht haben.

Interessant ist auch, dass es zuletzt eine standardisierte, objektorientierte Version von COBOL gegeben hat, die aber kaum zum Einsatz kam.

1.2 Fest definierter Sprachumfang

Moderne, objektorientierte Programmiersprachen kennen oft nur sehr wenige Befehle wie `if`, `while`, `for` usw. Ihre ganz große Stärke liegt darin, dass sie über mächtige Klassenbibliotheken mit einer Unzahl an Methoden verfügen, die durch eigene Entwicklungen permanent erweitert werden. Jedes noch so komplizierte Problem kann durch teilweise simple Methodenaufrufe gelöst werden.

Klassisches COBOL kennt solche Bibliotheken nicht. Hier werden alle Anforderungen durch fest vorgegebene Befehle gelöst. Will man beispielsweise in einer Zeichenkette den Buchstaben `Ä` durch `AE` ersetzen, geht das in Java recht einfach, wie in Listing 1.1 zu sehen.

```
String zeichenkette = "ÄÖÜ";  
String neu = zeichenkette.replace("Ä", "AE");
```

Listing 1.1: Ersetzen von Zeichen in Java

Um dieselbe Aufgabe in COBOL zu lösen, ist weit mehr Text erforderlich, wie in Listing 1.2 abgedruckt. Da sich in diesem Beispiel der Ausgangstext in seiner Länge maximal verdoppeln kann, muss das bei der Definition des neuen Feldes berücksichtigt werden. Ein Java-Entwickler macht sich darüber eher wenig Gedanken. In COBOL haben alle Datenfelder eine feste Länge. Die Angabe `PIC X(3)` bestimmt, dass das Datenfeld drei alphanumerische Zeichen beinhalten kann.

```
WORKING-STORAGE SECTION.  
01 ZEICHENKETTE          PIC X(3) VALUE "ÄÖÜ".  
01 NEU                   PIC X(6) VALUE SPACE.  
01 NEU-TABELLE REDEFINES NEU.  
    05 NEU-ELEMENT       PIC X OCCURS 6.  
01 I                     PIC 9.  
01 K                     PIC 9.  
PROCEDURE DIVISION.  
    MOVE 1 TO K.  
    MOVE SPACE TO NEU.  
    PERFORM VARYING I FROM 1 BY 1 UNTIL I > 3  
        IF ZEICHENKETTE(I:1) = "Ä"  
            MOVE "A" TO NEU-ELEMENT(K)  
            ADD 1 TO K
```

```
MOVE "E" TO NEU-ELEMENT(K)
ADD 1 TO K
ELSE
MOVE ZEICHENKETTE(I:1) TO NEU-ELEMENT(K)
ADD 1 TO K
END-IF
END-PERFORM.
```

Listing 1.2: Ersetzen von Zeichen in COBOL

Das Problem bei dem hier gezeigten Beispiel ist, dass aus einem Zeichen zwei Zeichen werden können und sich die restlichen Zeichen daran anschließen müssen. Ist dagegen gefordert, aus jedem Ä ein einfaches A zu machen, lässt sich die Aufgabe auch in COBOL viel kürzer lösen (siehe Listing 1.3).

```
WORKING-STORAGE SECTION.
01 ZEICHENKETTE          PIC X(3) VALUE "ÄÖÜ".
01 NEU                   PIC X(3) VALUE SPACE.
PROCEDURE DIVISION.
MOVE ZEICHENKETTE TO NEU.
INSPECT NEU REPLACING ALL "Ä" BY "A".
```

Listing 1.3: Ersetzen einzelner Zeichen in COBOL

Für die unterschiedlichen Aufgaben stehen in COBOL verschiedene Befehle zur Verfügung. Diese muss man geschickt kombinieren, um das gewünschte Ziel zu erreichen.

1.3 Prozedurale Programmierung

COBOL ist eine prozedurale Programmiersprache, was meint, dass man alle Anweisungen in einer einzigen Prozedur hintereinander schreibt und diese linear abgearbeitet werden.

In einer funktionsorientierten Programmiersprache wie beispielsweise C schreibt man eine Reihe von Funktionen und kombiniert diese geschickt. In objektorientierten Sprachen programmiert man dagegen Klassen, die mithilfe von Methoden die Attribute der Klasseninstanzen manipulieren.

In COBOL definiert man seine Daten in der DATA DIVISION und dort meist in der WORKING-STORAGE SECTION. Alle Datenfelder stehen allen Anweisungen innerhalb der Prozedur zur Verfügung, die in der PROCEDURE DIVISION programmiert werden. Es gibt keine Kapselung der Datenfelder wie beispielsweise die innerhalb einer Funktion oder wie die Attribute einer Instanz. Es ist auch nicht möglich, lokale Variablen innerhalb einer Schleife zu programmieren, deren Sichtbarkeit dann auf die Anweisungen innerhalb dieser Schleife begrenzt wäre.

In COBOL können externe Unterprogramme geschrieben werden, um die Komplexität einer Aufgabe in mehrere kleinere Programme aufzuteilen und um die Wiederverwendbarkeit von Logik zu ermöglichen. Typische COBOL-Programme sind aber dennoch meist mehrere Tausend Zeilen lang und bestehen nicht aus einer Unzahl an Unterprogrammaufrufen.

Die Möglichkeiten, heute in COBOL ebenfalls objektorientiert zu programmieren oder eigene Funktionen zu implementieren, findet man so gut wie gar nicht.

1.4 Linearer Programmablauf

Alle Anweisungen eines COBOL-Programms stehen in der `PROCEDURE DIVISION`. Das Programm beginnt mit der ersten dort stehenden Anweisung, die eine nach der anderen abgearbeitet werden.

Dieser lineare Programmablauf wird nur durch Befehle wie `GO TO` oder `PERFORM` unterbrochen. Während man mit `GO TO` schlicht zu einer anderen Stelle innerhalb der Prozedur springt, um von dort an wieder linear weiterzulaufen, ruft man mit `PERFORM` ein internes Unterprogramm auf, an dessen Ende man wieder an die rufende Stelle zurückspringt und es dann mit der Anweisung weitergeht, die auf die `PERFORM`-Anweisung folgt. In den Genen der Programmiersprache COBOL findet man eine solche Aufteilung in interne Unterprogramme jedoch nicht. Wenn sich der Programmierer nicht um den ordentlichen Ablauf innerhalb seines Programms kümmert, passieren durchaus überraschende Dinge.

Um interne Unterprogramme zu programmieren, benötigt man Sprungmarken, bei denen es sich um Sections oder Paragraphen handeln kann. Die Details werden in späteren Kapiteln erklärt. In Listing 1.4 ist ein COBOL-Programm zu sehen, das über drei interne Unterprogramme verfügt, die mithilfe der Anweisung `PERFORM` auch aufgerufen werden.

```
WORKING-STORAGE SECTION.  
01 FELD                PIC 99.  
PROCEDURE DIVISION.  
STEUER SECTION.  
    MOVE 0 TO FELD.  
    PERFORM UPRO-01.  
    PERFORM UPRO-02.  
UPRO-01 SECTION.  
    ADD 1 TO FELD.  
UPRO-02 SECTION.  
    ADD 2 TO FELD.  
ENDE SECTION.  
    DISPLAY FELD.
```

Listing 1.4: COBOL-Programm mit internen Unterprogrammen

Was passiert in Listing 1.4 genau? Das Programm beginnt mit der Anweisung `MOVE 0 TO FELD`, was dieses mit 0 initialisiert. Danach wird das interne Unterprogramm `UPRO-01` aufgerufen. Da es sich hier um eine `SECTION` handelt, endet das interne Unterprogramm mit dem Beginn der nächsten `SECTION`. Durch den Aufruf `PERFORM UPRO-01` wird also lediglich die Anweisung `ADD 1 TO FELD` ausgeführt, Der Inhalt der Variablen `FELD` ist jetzt 1. Die Steuerung geht an die nächste Anweisung nach dem `PERFORM` zurück. Dort steht `PERFORM UPRO-02`. Einzige Anweisung dieses internen Unterprogramms ist `ADD 2 TO FELD`. Diese wird ausgeführt und in der Variablen `FELD` steht jetzt der Wert 3. Die Steuerung geht wieder an die nächste Anweisung nach dem `PERFORM` zurück und die nächste Anweisung ist tatsächlich `ADD 1 TO FELD`. Diese wird ausgeführt und in `FELD` steht jetzt 4. Jetzt kommt auch noch `ADD 2 TO FELD` dran, womit wir schon bei dem Wert 6 sind. Am Ende wird dann noch `DISPLAY FELD` ausgeführt, was den Wert 6 auf dem Bildschirm ausgibt oder in die Standardausgabe schreibt. Danach folgt nichts mehr und die Steuerung geht an das Betriebssystem zurück.

Das meint COBOL mit einem linearen Programmablauf.

Um die erneuten Additionen, also den erneuten linearen Ablauf der internen Unterprogramme, zu verhindern, müsste nach dem zweiten `PERFORM` entweder ein `GO TO ENDE` stehen oder ein `PERFORM ENDE`, gefolgt von der Anweisung `STOP RUN`, die das Programm jetzt beendet und die Steuerung an das Betriebssystem zurückgibt.

1.5 Datenfelder mit fester Länge

Alle Datenfelder, mit denen man in einem COBOL-Programm arbeiten will, müssen in der `DATA DIVISION` definiert werden. Je nach Verwendung findet man diese Felder dort zum Beispiel in der `FILE SECTION` oder der `WORKING-STORAGE SECTION`.

Für alle Datenfelder gilt, dass ihr Datentyp und ihre Länge in Byte fest definiert sind. Datenfelder mit variabler Länge, die sich erst zur Laufzeit ergibt, gibt es in COBOL nicht.

Manchmal spricht man in COBOL von einem Feld oder einer Tabelle mit variabler Länge, meint damit aber nicht dasselbe wie ein Entwickler in einer objektorientierten Sprache. Will man beispielsweise in COBOL mit einem Datenbankfeld arbeiten, das vom Typ `VARCHAR` ist, muss man Folgendes definieren:

```
01  FELDFNAME .  
    05  FELDLAENGE          PIC S9(4) COMP .  
    05  FELDDINHALT        PIC X(200) .
```

Zu dem eigentlichen Datenfeld gehört zunächst ein Längenfild, gefolgt von einem weiteren Feld für den Felddinhalt. Dieses ist aber in dem Beispiel immer 200 Byte lang, egal, welcher Wert in `FELDLAENGE` steht. Der COBOL-Programmierer muss vielmehr selbst darauf achten, dass er maximal so viele Bytes verarbeitet, wie das Längenfild angibt.

Nicht selten füllt er FELDINHALT vor einem Zugriff mit lauter Leerzeichen, um so die Werte aus einem vorangegangenen Zugriff zu überschreiben.

Auch eine Tabelle mit einer variablen Anzahl an Elementen lässt sich in COBOL zwar definieren, dennoch belegt eine solche Tabelle im Hauptspeicher immer den maximal benötigten Platz.

```
01 ANZAHL          PIC 99.  
01 TABELLE.  
    05 ELEMENT     OCCURS 1 TO 20 DEPENDING ON ANZAHL.  
        10 DATENFELD PIC X(20).
```

Listing 1.5: Tabelle mit variabler Elementanzahl

In Listing 1.5 ist eine solche Tabelle definiert. Sie soll mindestens ein, maximal zwanzig Elemente besitzen, je nachdem, was zur Laufzeit in dem Feld ANZAHL steht. Und tatsächlich kommt es zu einem schweren Fehler, wenn das Programm auf ein ungültiges Element zugreift. Im Hauptspeicher befinden sich aber immer 20 Elemente, die jeweils 20 Byte lang sind.

Dieser Umstand muss bei dem Design einer COBOL-Anwendung bedacht werden, auch wenn heute der zur Verfügung stehende Hauptspeicher viel größer ist als früher.

Es gibt aber nicht nur das Problem, dass die Datenfelder in Summe zu groß sein könnten, manchmal sind sie schlicht auch zu klein. Will man beispielsweise eine XML-Datei lesen, weiß man gar nicht, wie groß die einzelnen Felder für die Aufnahme der Daten definiert werden müssen. Moderne COBOL-Compiler bieten tatsächlich die Möglichkeit, solche Dateien zu lesen, über entsprechende Statusfelder bekommt man dabei die Information, ob es dabei dazu gekommen ist, dass Feldinhalte abgeschnitten werden mussten.

1.6 Module statt Instanzen

Eine komplexe Anwendung besteht aus einer Menge einzelner COBOL-Programme, die sich untereinander aufrufen können. Typischerweise sind sie oft mehrere Hundert oder gar Tausende Zeilen lang. Das hängt einerseits damit zusammen, dass die Programmiersprache COBOL sehr geschwätzig ist, man also viel Quellcode für relativ wenig Funktion schreiben muss, andererseits gibt es aber auch keine wirkliche Motivation, stark zu modularisieren.

Ein COBOL-Programm erledigt typischerweise eine Aufgabe, und diese komplett. Dabei greift es gleichzeitig auf alle Datenfelder zu, die es dafür benötigt.

Objektorientierte Sprachen kapseln zusammengehörige Daten in Klassen in Form von Attributen. Diese Klassen bieten eine Reihe von Methoden, um ihre Attribute zu manipulieren. Benötigt man zur Laufzeit mehrere Daten desselben Typs, erzeugt man die passende Anzahl von Instanzen dieser Klassen.

Eine komplexe objektorientierte Anwendung besteht daher aus einer umfangreichen Menge von Klassen, die sich gegenseitig benutzen und ihre Daten vor anderen schützen.

Benötigt man in COBOL mehrere Daten desselben Typs, definiert man sich eine Tabelle, die groß genug ist. Die eigenen Daten werden an externe Unterprogramme übergeben, die diese manipulieren können. Von einer Kapselung der Daten kann hier nicht die Rede sein.

Um ein COBOL-Programm zu verstehen und um es ändern zu können, muss man seinen Blickwinkel ändern. In COBOL stehen nicht die Daten im Vordergrund, sondern die Befehle der programmierten Prozedur. Ein COBOL-Programmierer fragt sich, welche Daten er in Gänze zur Bewältigung seiner Aufgabe benötigt, und definiert diese dann komplett in seiner `DATA DIVISION`. Er wird nur dann ein externes Unterprogramm aufrufen, wenn er die dort programmierte Prozedur auch in anderen COBOL-Programmen verwenden will. Für das eigene Programm ist das externe lediglich eine Art verlängerte Werkbank.

Um zu verstehen, was das bedeutet, stellen Sie sich ein Hauptprogramm vor, das in Folge seiner Verarbeitung dasselbe externe Unterprogramm zweimal aufruft. Beim ersten Aufruf befindet es sich noch in seinem initialen Zustand. Die Datenfelder sind leer oder mit Initialwerten gefüllt. Beim zweiten Aufruf stehen in den Feldern jedoch noch genau die Werte, die die Felder am Ende des ersten Aufrufs hatten, außer man gibt das externe Modul nach dem Aufruf explizit wieder frei. Über die `LINKAGE SECTION` teilen sich beide Module einen Teil der Daten, die sie beide manipulieren können. Es gibt auch keine Instanzen von externen Unterprogrammen.

Programmstruktur und grundlegende Sprachelemente

Die meisten Programmiersprachen erlauben die Definition von Variablen, Konstanten und Dateien, aber auch die Codierung von ausführbaren Anweisungen an beliebigen Stellen im Quellprogramm. In COBOL sieht es dagegen anders aus. In diesem Kapitel werden die Struktur und die Elemente eines COBOL-Programms beschrieben, sodass Sie sich in einem bestehenden COBOL-Programm leichter zurechtfinden.

2.1 COBOL-Programmstruktur

In der Programmiersprache COBOL hat man für Übersicht im Quellprogramm gesorgt, indem man das Quellprogramm in vier Programmteile, DIVISIONs genannt, untergliedert hat. Jedem Programmteil wurde ein fester Name als Überschrift und ein Verwendungszweck gegeben:

```
IDENTIFICATION DIVISION.  
ENVIRONMENT DIVISION.  
DATA DIVISION.  
PROCEDURE DIVISION.
```

Listing 2.1: Die vier Teile eines COBOL-Programms

Diese DIVISIONs müssen in der hier angegebenen Reihenfolge im Programm erscheinen. Manche sind optional und können weggelassen werden.

DIVISIONs unterteilen sich weiter in SECTIONs, deren Namen durch die COBOL-Syntax vorgegeben sind, außer in der PROCEDURE DIVISION.

SECTIONs (auch Kapitel genannt) können sich weiter in Paragraphen unterteilen. Auch hier gilt, dass deren Namen vorgeschrieben sind, außer in der PROCEDURE DIVISION.

Die eigentlichen COBOL-Definitionen und Anweisungen schreibt man schließlich in Sätzen, wobei diese Bezeichnung wörtlich zu nehmen ist. Sätze enden mit einem Punkt und der spielt in der Syntax von COBOL eine wichtige Rolle. Spätestens in dem Kapitel 9 über Verzweigungen und interne Unterprogramme wird das deutlich.

Sätze bestehen aus Klauseln und Wörtern, die teils durch die Syntax vorgegeben und teils frei durch den Programmierer wählbar sind.

2.1.1 Die Bedeutung der Programmteile (DIVISIONs)

Der Erkennungsteil IDENTIFICATION DIVISION enthält eine Reihe von Informationen zur Benennung und Dokumentation des Quellprogramms. Dieser Teil hat wenig Einfluss auf das Programm. Die hier gemachten Angaben werden – für spätere Bezugnahme – *Kommentareintragungen* genannt. Der Maschinenteil ENVIRONMENT DIVISION beschreibt die für das Programm notwendige Umgebung. Zusätzlich werden Beziehungen zwischen den logischen Dateien, die im Quellprogramm definiert sind, und den tatsächlichen Ein/Ausgabeeinheiten, auf denen sich diese Dateien befinden, hergestellt. Die an dieser Stelle gemachten Angaben werden *Klauseln* genannt.

Der Datenteil DATA DIVISION dient dazu, die Daten zu beschreiben, die im Programm verarbeitet werden sollen. Das umfasst Dateisatzbeschreibungen, Konstanten und Variablen. Diese Angaben werden *Definitionen* und *Klauseln* genannt.

Der Prozedurteil PROCEDURE DIVISION enthält eine Reihe von ausführbaren Anweisungen, die zusammen mit den definierten Daten das Objektprogramm bilden. Die in diesem Teil gemachten Angaben werden *Anweisungen* genannt.

2.1.2 Die Hierarchie in einem COBOL-Programm

```

IDENTIFICATION DIVISION
  |--> Paragraphen
        |--> Kommentare

ENVIRONMENT DIVISION
  |--> SECTIONs
        |--> Paragraphen
              |--> Sätze
                    |--> Klauseln
                          |--> Wörter

DATA DIVISION
  |--> SECTIONs
        |--> Definitionen
              |--> Sätze
                    |--> Klauseln
                          |--> Wörter

PROCEDURE DIVISION
  |--> SECTIONs
        |--> Paragraphen
              |--> Sätze
                    |--> Anweisungen
                          |--> Wörter

```

Listing 2.2: COBOL-Hierarchie

Alle Namen der SECTIONS und der Paragraphen in den ersten drei DIVISIONs sind von COBOL fest vorgegeben. In der PROCEDURE DIVISION können Sie beliebige Namen verwenden.

2.1.3 Das COBOL-Programm im Überblick

- ```
(1) IDENTIFICATION DIVISION. Kommentareintragungen ...

(2) ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.
 Name des Umwandlungssystems
 OBJECT-COMPUTER.
 Name des ausführenden Systems
 SPECIAL-NAMES.
 Speziell vom Programmierer
 festzulegende Namen und Regeln
 REPOSITORY.
 Namen von objektorientierten Klassen,
 Funktionen, Interfaces, die in diesem
 Programm benutzt werden sollen. Diese sind
 nicht Gegenstand dieses Buches
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
 Klauseln zur Definition von Dateien
 I-O-CONTROL.
 Klauseln zu speziellen Ein/Ausgabetechniken

(3) DATA DIVISION.
 FILE SECTION.
 Definition von Datensätzen
 WORKING-STORAGE SECTION.
 Definition von Konstanten und Variablen
 LOCAL-STORAGE SECTION.
 Definition von Variablen, die bei jedem
 Programmaufruf dynamisch angelegt werden
 LINKAGE SECTION.
 Felder für den Datenaustausch in einem
 Unterprogramm
 REPORT SECTION.
 Für Definitionen für den REPORT-WRITER
 SCREEN SECTION.
 Definition von Eingabemasken

(4) PROCEDURE DIVISION.
 Anweisungen für die Verarbeitung
```

Listing 2.3: Aufbau eines COBOL-Programms

Eine detaillierte Beschreibung der einzelnen DIVISIONs und SECTIONs finden Sie in Kapitel 3.

## 2.2 COBOL-Sprachelemente

Sie haben im vorangehenden Abschnitt gesehen, dass ein COBOL-Programm letztendlich aus Wörtern besteht.

### 2.2.1 Reservierte Wörter

Unter einem *reservierten Wort* versteht man ein Wort, das für die Darstellung einer Klausel oder einer Anweisung reserviert worden ist. Diese Wörter umfassen:

#### Schlüsselwörter

Dies sind Wörter, die vorhanden sein müssen, um eine korrekte Anweisung zu programmieren. Es gibt drei Arten von Schlüsselwörtern:

- Verben wie MOVE, PERFORM, COMPUTE
- Notwendige Wörter, die in den Klauseln und Anweisungen vorkommen, z.B. TO, FROM
- Wörter, die eine besondere funktionelle Bedeutung haben, z.B. NEGATIVE, NUMERIC

#### Kontextsensitive Schlüsselwörter

Verschiedene Schlüsselwörter sind nur dann reserviert, wenn sie innerhalb einer Anweisung verwendet werden, für die sie als Schlüsselwort vorgesehen sind. Wird dasselbe Wort in einem anderen Zusammenhang genutzt, wird es als Programmiererwort betrachtet (Programmiererwörter). Beispiele für solche Wörter sind ARITHMETIC, BACKGROUND-COLOR, BYTE-LENGTH.

#### Wahlwörter

Die Wahlwörter können wahlweise, wo sie erlaubt sind, verwendet werden. Sie haben keinen Einfluss auf die Wirkung einer Klausel oder einer Anweisung und dienen ausschließlich der besseren Lesbarkeit des Programms, z.B. IS, ARE.

#### Verknüpfers

Ein Verknüpfers kann sein

- ein *Kennzeichnerbindewort*: IN, OF verknüpft einen Datennamen oder Paragraphennamen mit seinem Kennzeichner, z.B. NAME IN KUNDENSATZ
- oder ein *boolescher Operator*: AND, OR, AND NOT, OR NOT wird verwendet zur Herstellung von zusammengesetzten Bedingungen.

## 2.2.2 Programmiererwörter

Ein Programmiererwort ist ein COBOL-Wort, das vom Programmierer selbst gewählt werden kann. Es wird als symbolische Adresse zur Benennung von Datenbereichen, Dateien oder Programm-Verzweigungszielen verwendet.

### Aufbau

1. Ein Wort besteht aus 1 bis 31 Zeichen des folgenden Vorrates: A bis Z, 0 bis 9, – (Bindestrich) und \_ (Unterstrich).
2. Ein Wort darf nicht mit einem Bindestrich beginnen oder enden.
3. Es darf kein Leerzeichen enthalten.
4. Alle Programmiererwörter, ausgenommen Segmentnummern, Stufennummern und Paragraphennamen in der PROCEDURE DIVISION, müssen eindeutig sein. Paragraphennamen innerhalb einer einzigen SECTION dürfen sich nicht wiederholen.  
Es ist zwar möglich, mehrere Variablen mit demselben Namen zu definieren; um sie dann aber auch verwenden zu können, müssen sie sich eindeutig qualifizieren lassen, also zum Beispiel innerhalb verschiedener Datengruppen angelegt worden sein.
5. Alle Programmiererwörter, ausgenommen Paragraphennamen, SECTION-Namen, Stufennummern und Segmentnummern, müssen mindestens ein alphabetisches Zeichen enthalten. Stufennummern dienen der hierarchischen Deklaration von Variablen und Segmentnummern der Segmentierung der PROCEDURE DIVISION. Auf Letzteres wird im Rahmen dieses Buches nicht weiter eingegangen.

### Beispiele

```
Datennamen ----> BETRAG MWST KUNDEN-SATZ
Kapitel u. Paragraphennamen ----> VERARBEITUNG LESEN
```

**Listing 2.4:** Beispiele für Programmiererwörter

## 2.2.3 Literale

Ein Literal ist eine Konstante, die in der Form einer Zeichenfolge angegeben wird. Diese kann mittels einer MOVE-Anweisung übertragen oder für die Vorbesetzung eines Datenfelds mittels der VALUE-Klausel verwendet werden.

### Nicht numerische Literale

Ein nicht numerisches Literal ist eine Zeichenfolge von 1 bis 160 Zeichen, die in einfachen oder doppelten Anführungszeichen eingeschlossen ist.

Das Literal kann alle Zeichen aus dem aktuellen Zeichenvorrat enthalten. Sollen die Anführungszeichen selbst Bestandteil der Zeichenkette sein, müssen sie verdoppelt werden.

**Beispiele**

```
"Nachricht ""in Anführungszeichen""
'16%'
"Umsatzliste"
```

**Listing 2.5:** Beispiele für nicht numerische Variable

**Hexadezimale Literale**

Damit Sie jeden Wert aus der aktuellen Codetabelle von 0 bis 255 in einem Literal im Programm verwenden können, bietet COBOL die Möglichkeit, ein hexadezimaler Literal von X"00" bis X"FF" anzugeben. Das Literal wird als nicht numerisches betrachtet und muss für jedes Byte zwei hexadezimale Ziffern beinhalten.

**Beispiele**

```
X"00000F" (3 Byte)
X"2020202020" (5 Byte)
X'313233343536' (6 Byte)
```

**Listing 2.6:** Beispiele für hexadezimale Literale

**Verketten von nicht numerischen Literalen**

Mit dem Ampersandzeichen & lassen sich sowohl nicht numerische als auch hexadezimale Literale in verschiedenen Befehlen verketteten.

**Beispiele**

```
01 EINGABE-NAME PIC X(40) VALUE "PETER" & " SCHULZ".
```

Auch in der MOVE- oder in jeder anderen Anweisung lässt sich die Verkettung anwenden:

```
MOVE "PETER" & " SCHULZ" TO EINGABE-NAME
DISPLAY "PETER" & " SCHULZ" AT 1001
```

**Listing 2.7:** Beispiele für das Verketteten nicht numerischer Literale

**Numerische Literale (Festkommazahlen)**

Ein numerisches Literal ist eine Folge aus den Zeichen:

1. Ziffern von 0 bis 9
2. Vorzeichen + oder -
3. Dezimalpunkt .



### Aufbau

1. Das Literal darf maximal 1 bis 31 Ziffern enthalten.
2. Das Literal darf nur ein Vorzeichen enthalten und muss dann auch mit diesem beginnen. Wird das Vorzeichen weggelassen, wird + angenommen.
3. Das Literal darf nur einen Dezimalpunkt enthalten, der nie als letztes Zeichen angegeben werden darf.
4. Es muss mindestens eine Ziffer verwendet werden.

### Beispiele

```
-123.45
+9876
.99
00000
```

Listing 2.8: Beispiele für numerische Literale

### Numerische Literale (Fließkommazahlen)

Ein solches Literal besteht aus zwei Festpunktzahlen, die durch den Buchstaben E getrennt sind.

### Aufbau

1. Das erste Literal darf maximal 1 bis 31 Ziffern enthalten und kann mit einem Vorzeichen und einem Dezimalpunkt ausgestattet sein.
2. Das zweite Literal ist der Exponent. Es kann vorzeichenbehaftet sein, darf aber maximal drei Ziffern umfassen. Die Angabe eines Dezimalpunkts ist nicht erlaubt.
3. Es muss jeweils mindestens eine Ziffer verwendet werden.

### Beispiele

```
-123.45E5
+9876E123
.99E-3
0E0
```

Listing 2.9: Beispiele für Fließkommazahlen

### Boolesche Literale

Ein boolesches Literal besteht aus den Zeichen 0 und 1 oder einer hexadezimalen Ziffer 0 bis F und kann eine Gesamtlänge von 160 Zeichen annehmen. Es beginnt mit einem B bzw. BX und ist in einfachen oder doppelten Anführungszeichen eingeschlossen.

Boolesche Literale können zusammen mit den booleschen Operatoren B-XOR, B-AND, B-OR und B-NOT verwendet werden.

**Beispiele**

```
B"1100"
B'11110000'
BX"8"
```

**Listing 2.10:** Beispiele für boolesche Literale

**Nationale Literale**

COBOL kennt neben den alphanumerischen Literalen (USAGE DISPLAY) auch nationale Literale (USAGE NATIONAL). Letztere kennzeichnen sich besonders dadurch, dass sie für die interne Darstellung jedes Zeichens mehr Speicherplatz benötigen als alphanumerische Zeichen. Bei Verwendung von UTF-16 ist jedes Zeichen zwei Byte groß.

Nationale Literale lassen sich auch in hexadezimaler Form darstellen.

**Beispiele**

```
N"Text mit nationalem Zeichensatz"
N'123'
NX"02A102A2"
```

**Listing 2.11:** Beispiele für nationale Literale

**2.2.4 Figurative Konstanten**

Eine figurative Konstante ist ein COBOL-Wort, für das vom Compiler ein bestimmter Wert erzeugt wird.

**ZERO bzw. ZEROS bzw. ZEROES**

Der Inhalt des Datenfelds, das diese figurative Konstante enthält, hängt von seinem Attribut ab.

**Beispiel**

| Feldbeschreibung | Binär    | entpackt | Gepackt  |
|------------------|----------|----------|----------|
| Inhalt (hex)     | "000000" | "303030" | "00000F" |

**SPACE bzw. SPACES**

Eine oder mehrere Wiederholungen des Zeichens »Leerzeichen«.

**Beispiel**

Löschen des Bereichs KUNDEN-SATZ:

```
MOVE SPACE TO KUNDEN-SATZ
```

Inhalt des Bereichs in hexadezimaler Schreibweise (lauter Leerzeichen):

```
X"20202020202020202020202020202020"
```

Die MOVE-Anweisung überträgt Daten zu einem Feld und wird im 6 detailliert ausgeführt.

### **HIGH-VALUE bzw. HIGH-VALUES**

Damit ist das Zeichen mit der höchsten Ordnungsnummer im aktuell verwendeten Zeichensatz gemeint. Im ASCII-Zeichensatz entspricht das X"FF".

#### **Beispiel**

```
MOVE HIGH-VALUE TO KENNZEICHEN
```

Inhalt des Datenfelds:

```
X"FFFFFF"
```

### **LOW-VALUE bzw. LOW-VALUES**

Eine oder mehrere Wiederholungen des Zeichens X"00". Damit ist das Zeichen mit der niedrigsten Ordnungsnummer im aktuell verwendeten Zeichensatz gemeint.

#### **Beispiel**

```
MOVE LOW-VALUE TO KENNZEICHEN.
```

Inhalt des Datenfelds:

```
X"000000"
```

### **QUOTE bzw. QUOTES**

Eine oder mehrere Wiederholungen des Zeichens »Anführungszeichen«.

#### **Beispiel**

```
MOVE QUOTE TO DATENFELD
```

### **ALL Literal**

Die figurative Konstante ALL Literal wurde für den Programmierer freigelassen. Er kann damit bestimmen, welches Zeichen hier eingesetzt werden soll.

**Beispiel 1**

Aufbauen einer Linie

```
MOVE ALL "-" TO LINIE
```

Inhalt des Datenfelds:

```
"-----"
```

**Beispiel 2**

Aufbauen einer Tabulatorzelle

```
MOVE ALL "I----" TO TABZEILE
```

Inhalt des Datenfelds:

```
"I----I----I----I----I----I"
```

Sie können anhand dieses Beispiels sehen, dass dies so lange wiederholt wird, bis die Feldlänge nicht mehr ausreicht.

## 2.2.5 Trennsymbole

### Interpunktion

#### Leerzeichen

Das Leerzeichen muss immer nach jedem COBOL-Element angegeben werden. Wo ein Leerzeichen vorhanden ist, können auch mehrere angegeben werden.

#### Beispiel

```
COMPUTE SUMME = ZAHL1 + ZAHL2
```

#### Komma und Semikolon

Diese Zeichen haben keine Bedeutung für die Interpretation des Quellprogramms. Sie verbessern lediglich die Lesbarkeit der Klauseln und Anweisungen.

#### Beispiel

```
ADD 1 TO ZAHL1, ZAHL2, ZAHL3.
01 STEUER PIC S9(0); COMP; VALUE ZERO.
```

**Listing 2.12:** Interpunktionsbeispiel

### Punkt

Der Punkt stellt das Endkriterium einer Aussage dar, z.B. das Ende

- einer Teil- oder Kapitelüberschrift
- einer Dateibeschreibung
- einer Felddbeschreibung
- einer Anweisung

### Beispiel

```
WORKING-STORAGE SECTION.
01 SCHALTER PIC 9 VALUE ZERO.

PROCEDURE DIVISION.
VERARBEITUNG SECTION.
 IF SCHALTER = 1
 DISPLAY "ENDE"
 STOP RUN.
```

**Listing 2.13:** Korrekte Verwendung des COBOL-Punkts

Insbesondere zeigt sich die Bedeutung des Punkts in der Beendigung von bedingten Anweisungen, wie in Listing 2.14.

```
 IF ZZ = 50
 MOVE ZERO TO ZZ
 ADD 1 TO SZ
 WRITE A-SATZ AFTER PAGE.

 WRITE A-SATZ FROM POSTENZEILE
```

**Listing 2.14:** Fehlerhafte Verwendung des COBOL-Punkts

Der Punkt beendet in diesem Beispiel die IF-Anweisung; eine nachfolgende Anweisung wird in jedem Fall ausgeführt.

### Anweisungsbegrenzer

In ANSI85 wurden einige Anweisungen um einen Anweisungsbegrenzer erweitert. Dieser Begrenzer hat die Aufgabe, eine Anweisung zu beenden und syntaxmäßig von der nachfolgenden Anweisung zu trennen. Der Begrenzer ersetzt damit die Funktion des Punkts, wie in Listing 2.15.

```
 IF ZZ = 50
 MOVE ZERO TO ZZ
 ADD 1 TO SZ
 WRITE A-SATZ AFTER PAGE
```

```
END-IF
WRITE A-SATZ FROM POSTENZEILE
```

**Listing 2.15:** Verwendung von Anweisungsbegrenzern

Ohne Anweisungsbegrenzer war es oft problematisch, ein nicht so sauber geschriebenes COBOL-Programm richtig zu lesen. Wenn der Entwickler nicht auf korrekte Einrückungen in seinem Quellcode geachtet hat, konnte ein COBOL-Punkt schnell mal überlesen werden.

```
IF ZZ = 50
 MOVE ZERO TO ZZ
 ADD 1 TO SZ.
 WRITE A-SATZ AFTER PAGE. *> ACHTUNG !!
WRITE A-SATZ FROM POSTENZEILE.
```

**Listing 2.16:** Gefährliche Verwendung des COBOL-Punkts

Verwendet man für jede entsprechende Anweisung ihren Anweisungsbegrenzer, wird man vom Compiler auf fehlerhaft eingesetzte COBOL-Punkte aufmerksam gemacht. Beispiel: Am Ende eines Befehls einer IF-Anweisung wird versehentlich ein Punkt gesetzt. Außerdem wird das IF durch ein END-IF beendet. Der COBOL-Compiler wird nun dieses END-IF als fehlerhaft markieren, weil das IF ja bereits durch den Punkt beendet wurde.

Wird also ein Anweisungsbegrenzer als fehlerhaft betrachtet, muss man den darüber liegenden Block auf versehentlich gesetzte COBOL-Punkte oder natürlich auch auf fehlerhaft benutzte Anweisungsbegrenzer hin untersuchen.

```
IF ZZ = 50
 MOVE ZERO TO ZZ
 ADD 1 TO SZ.
 WRITE A-SATZ AFTER PAGE
END-IF. *> Syntaxfehler !!
WRITE A-SATZ FROM POSTENZEILE.
```

**Listing 2.17:** Syntaktisch fehlerhafte Verwendung des COBOL-Punkts

Tabelle 2.1 enthält alle Anweisungsbegrenzer.

| Begrenzer   | Anweisung |
|-------------|-----------|
| END-ACCEPT  | ACCEPT    |
| END-ADD     | ADD       |
| END-CALL    | CALL      |
| END-COMPUTE | COMPUTE   |

**Tabelle 2.1:** Die Anweisungsbegrenzer von COBOL

| Begrenzer    | Anweisung |
|--------------|-----------|
| END-DELETE   | DELETE    |
| END-DISPLAY  | DISPLAY   |
| END-DIVIDE   | DIVIDE    |
| END-EVALUATE | EVALUATE  |
| END-IF       | IF        |
| END-MULTIPLY | MULTIPLY  |
| END-PERFORM  | PERFORM   |
| END-READ     | READ      |
| END-RECEIVE  | RECEIVE   |
| END-RETURN   | RETURN    |
| END-REWRITE  | REWRITE   |
| END-SEARCH   | SEARCH    |
| END-START    | START     |
| END-STRING   | STRING    |
| END-SUBTRACT | SUBTRACT  |
| END-UNSTRING | UNSTRING  |
| END-WRITE    | WRITE     |

**Tabelle 2.1:** Die Anweisungsbegrenzer von COBOL (Forts.)

## Anführungszeichen

Diese dürfen nur paarweise zur Begrenzung von nicht numerischen Literalen auftreten, außer wenn das Literal fortgesetzt wird. Es können wahlweise die doppelten oder das einfache Anführungszeichen paarig verwendet werden.

Einem öffnenden Anführungszeichen muss ein Leerzeichen oder eine runde Klammer unmittelbar vorausgehen. Einem schließenden Anführungszeichen muss eines der folgenden Trennzeichen unmittelbar folgen:

Leerzeichen, Komma, Semikolon, Punkt oder schließende runde Klammer.

### Beispiel

```
MOVE "FALSCHES KENNZEICHEN" TO FEHLER-MELDUNG.
```

## Linke und rechte Rundklammern

Diese dürfen nur paarweise als Begrenzer von Normal- und Spezialindizes, arithmetischen Ausdrücken oder Bedingungen verwendet werden.

Runden Klammern können Leerzeichen vorausgehen und/oder folgen; sie müssen es aber nicht.

Den Rundklammern kommt beim Aufruf von Funktionen eine besondere Bedeutung zu, da sie hier dazu verwendet werden können, die Parameterliste der Funktion zu umschließen.

## 2.2.6 Operatoren

### Arithmetische Operatoren

Folgende arithmetische Operatoren sind in COBOL bekannt und müssen immer durch mindestens ein Leerzeichen getrennt angegeben werden:

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

Außerdem kann das Additions- und Subtraktionszeichen auch als Vorzeichen für eine Konstante oder Variable verwendet werden.

### Boolesche Operatoren

Diese Operatoren dienen zur bitweisen Verknüpfung zweier boolescher Datenfelder beziehungsweise Konstanten.

- B-AND Boolesche UND-Verknüpfung
- B-OR Boolesche ODER-Verknüpfung
- B-XOR Boolesche EXKLUSIV-ODER-Verknüpfung
- B-NOT Boolesche Negation (nur ein Operand erlaubt)

### Vergleichsoperatoren

Auch hier muss vor und nach jedem Operator ein Leerzeichen vorhanden sein. Einen eigenen Operator für »ungleich« gibt es nicht. Die Abfrage auf »gleich« muss mit dem Schlüsselwort NOT negiert werden.

- > größer
- < kleiner
- = gleich
- >= größer gleich
- <= kleiner gleich
- NOT = nicht gleich



# Stichwortverzeichnis

3270-Gerät 392

\*> 42

>> 42

>>D 43

## A

A-Bereich 40

ACCEPT 123

AT 124

CRT STATUS 129

CURSOR 126

Format 1 124

Format 2 124, 132

ON EXCEPTION 125

ACCESS MODE 287, 312

ADD 142

Format 1 142

Format 2 143

Format 3 146

ADD CORR 146

Addieren 138

mehrere Datenfelder  
gleichzeitig 146

Addition 142

ADDRESS OF 195, 223

Adresse

relative 250

Adressfeld 116

übertragen 114

Adressierung

mehrdimensionale

Tabelle 244

Tabellenelemente 241

ADVANCING 302

AFTER 169, 270

AID 395

ALARM 393

ALIGNED 100

ALL 31, 281

ALPHABET 53

ALPHABETIC 192

ALPHABETIC-LOWER 192

ALPHABETIC-UPPER 192

Alphabetisches Datenfeld 65

Alphanumerisches Datenfeld  
66

Alphanumerisches druckauf-  
bereitetes Datenfeld 70

ALSO 202

ALTERNATE RECORD KEY  
331

FILE STATUS 332

AND 196

Anfangswert 80, 168

Anführungszeichen 35

ANSI85 141

ANSI-Standard 15

Anweisung 24, 40

Anweisungsbegrenzer 33

Arithmetische Operation

ADD 142

ADD CORR 146

COMPUTE 137

DIVIDE 148

MULTIPLY 146

SUBTRACT 144

SUBTRACT CORR 146

Arithmetischer Operator 36

ASCENDING KEY 262, 337

ASCII 53, 294

Assemblermakro 344

ASSIGN 285

AT END 299, 321

Attention Identifier 395

Ausgabemodus 297

## B

B-AND 29

Basic Mapping Support 391,  
392

Batchverarbeitung 375

B-Bereich 40

BDAM 396

Bedingung 181

Wahrheitswerte 197

zusammengesetzte 195

Bedingungsname 97, 114

Bedingungsnamen-Bedin-  
gung 193

Bedingungsvariable 97

BEFORE 270

Begrenzer 281

Bezugsschlüssel 331

BI 82

Bibliothek 16

Bildschirm

löschen 123

Bildschirmgröße 122

Binäres Suchen 261

BINARY 82

BINARY-CHAR 83

BINARY-DOUBLE 84

BINARY-LONG 84

BINARY-SHORT 84

BIND 357

BIT 100

BLANK WHEN ZERO 87

BLOCK CONTAINS 293

BMS 392

BMS Map 392

B-NOT 29

BOOLEAN 192

Boolescher Ausdruck 137

Boolescher Operator 26, 36

Boolesches Datenfeld 69

Boolesches Literal 29

B-OR 29

BOTTOM 293

B-XOR 29

BY CONTENT 225

BY REFERENCE 222

BY VALUE 226

Byte

reservieren 64

BYTE-LENGTH 101

## C

CALL 217, 348

BY REFERENCE 59

BY VALUE 59

OMITTED 221

ON EXCEPTION 219

ON OVERFLOW 219

USING 219

CALL-CONVENTION 232

CANCEL 218, 231

CASE 157

CICS 375

Basic Mapping Support  
391

COMMAREA 379, 390

Conversational Proces-  
sing 378

Dateiformate 396

Dateiverarbeitung 396

Daten sequenziell lesen  
400

DELETE 399

EIBRESP 382

ENDBR 400

EXECUTE INTERFACE

BLOCK 385

Fehlerbehandlung 382

- HANDLE CONDITION 383  
 Kommandoformat 375  
 LINK 387  
 Pseudoconversational Processing 379  
 READ 397  
 READNEXT 401  
 READPREV 401  
 RECEIVE MAP 394, 396  
 RESETBR 400  
 RETURN 380, 388  
 REWRITE 399  
 SEND CONTROL 394  
 SEND MAP 392  
 Service-Routine 382  
 STARTBR 400  
 strukturierte Programmierung 377  
 transaktionsorientiert 379  
 TRANSID 379  
 UNLOCK 399  
 Unterprogramme 387  
 WRITE 398  
 XCTRL 387  
 CLASS 55  
 CLOSE 305, 329  
 COBOL-60 15  
 COBOL-85 15  
 COBOL-Programm rekursive 228  
 CODASYL 15  
 CODE-SET 294  
 Codierformat 39  
   Compiler-Direktiven 42  
   Kommentare 42  
   Spalte 12–72 40  
   Spalte 1–6 40  
   Spalte 7 40  
   Spalte 73–80 41  
   Spalte 8–11 40  
   Spalten 8–72 40  
 COLLATING SEQUENCE 339  
 COMMAREA 379, 388, 390  
   Daten übergeben 390  
 COMMON 46, 236  
 COMP 82  
 Compiler 14  
 Compiler-Direktive 42  
 COMPUTE 137  
   NOT ON SIZE ERROR 141  
   ON SIZE ERROR 140  
   ROUNDED 139  
 CONFIGURATION SECTION 47  
 CONSOLE IS CRT 120  
 CONTINUE 183, 199, 259  
 Conversational Processing 378  
 CONVERTING 273  
 COPY 209  
   Bibliotheken 211  
   REPLACING 212  
   SUPPRESS 213  
 CR 76  
 CREATE TABLE 355  
 CRT STATUS 51, 129  
 CURRENCY SIGN 51  
 CURSOR 51, 126, 393  
 Cursor 363  
   Position 123, 124, 125, 126  
 CYCLE 154  
**D**  
 DAM 396  
 DATA DIVISION 24, 57, 291  
 DATALENGTH 388  
 DATAONLY 393  
 DATE 133  
 Datei  
   Eröffnungsmodus 296  
   Fehlerbehandlung 306  
   Fehlercode 287  
   Index-sequenziell 311  
   lesen 298  
   öffnen 296  
   Satzaufbau 291  
   schließen 305  
   schreiben 300, 304  
   sequenzielle 283  
   Status 300  
   Zugriffsmodus 312  
   Zustand 287  
 Dateiende 299, 300, 315, 341  
 Dateiname 284  
   prüfen 279  
   Variable 286  
 Dateiorganisation 311  
 Dateistatus  
   File-Status 0 287, 315  
   File-Status 1 287, 315  
   File-Status 2 315  
   File-Status 3 288, 316  
   File-Status 4 288, 316  
   File-Status 5 288, 317  
   File-Status 6 289, 317  
   File-Status 9 289, 317  
 Dateiverarbeitung  
   ACCESS MODE 287, 312  
   ALTERNATE RECORD KEY 331  
   ASSIGN 285  
   AT END 299  
   BLOCK CONTAINS 293  
   CLOSE 305, 329  
   CODE-SET 294  
   Dateiende 287, 315  
   DECLARATIVES 307  
   DELETE 327  
   dynamische Dateizuweisung 286  
   FILE STATUS 287, 315  
   File-Sharing-Konflikt 289, 317  
   LINAGE 293  
   LOCK MODE 290  
   logischer Fehler 288, 316  
   NOT AT END 300  
   OPEN 296, 318  
   ORGANIZATION 287, 312  
   PADDING CHARACTER 290  
   permanenter Fehler 288, 316  
   READ 298, 320  
   RECORD CONTAINS 291  
   RECORD DELIMITER 290  
   RECORD KEY 313  
   RESERVE 290  
   REWRITE 304, 324  
   Satzsperrern 288, 317  
   Schlüsselfehler 315  
   SELECT 284  
   START 328  
   System-Fehler 289  
   USE 306, 330  
   variable Satzlänge 291  
   WITH LOCK 300  
   WRITE 300, 323  
 Dateizuweisung  
   dynamische 286  
 Daten  
   statische 238  
 Datenbank 343  
   relationale 354  
 Datendefinition 91  
   Alphabetische Datenfelder 65  
   Alphanumerische Datenfelder 66  
   Alphanumerische druckaufbereitete Datenfelder 70  
   BINARY 82  
   BINARY-CHAR 83  
   BINARY-DOUBLE 84  
   BINARY-LONG 84  
   BINARY-SHORT 84  
   BIT-Felder 100  
   BLANK WHEN ZERO 87  
   Boolesche Datenfelder 69  
   COMP 82  
   CR 76  
   DB 76  
   Einfügungssymbole (B 0 / ) 70  
   FILLER 92  
   FLOAT-EXTENDED 85  
   FLOAT-LONG 85  
   FLOAT-SHORT 85  
   Gleitendes Vorzeichen 78  
   INDEX 86  
   INDEXED BY 248  
   JUSTIFIED 88  
   Konstante 100  
   NATIONAL 86  
   Numerische Datenfelder 67

- Numerische druckaufbe-  
 reitete Datenfelder 71  
 OBJECT REFERENCE 86  
 OCCURS 239  
 PACKED-DECIMAL 82  
 PICTURE 64  
 POINTER 86  
 PROGRAM POINTER 87  
 REDEFINES 94  
 SIGN 89  
 Stufennummer 66 98  
 Stufennummer 77 63  
 Stufennummer 88 96  
 Stufennummern 01–49  
 91  
 SYNCHRONIZED 89  
 USAGE 80  
 VALUE 80  
 Vorzeichen 75  
 Währungssymbol 79  
 Datenein-/ausgabe  
 ACCEPT 123  
 DISPLAY 119  
 Datenelement 92  
 Datenfeld 19, 63  
 Alphabetisches 65  
 Alphanumerisches 66  
 Alphanumerisches druck-  
 aufbereitetes 70  
 an Unterprogramm über-  
 geben 219  
 binär 82, 83  
 Boolesches 69  
 Feld 239  
 Numerisches 67  
 Numerisches druckaufbe-  
 reitetes 71  
 Datenformat 80  
 Datengruppe 91, 92, 242  
 Dateninitialisierung  
 INITIALIZE 109  
 Datenkategorie  
 Datenfeld 65  
 Datenname 40, 92  
 Datensatz 91  
 Länge 291  
 lesen 298  
 mischen 335  
 sortieren 335  
 zurückschreiben 304  
 Datensatzbeschreibung 295  
 Datenstruktur 91  
 Datentransfer 294  
 Datenzuweisung  
 MOVE 103  
 MOVE CORR 107  
 SET 113, 251  
 Datum und Uhrzeit  
 ACCEPT Format 2 132  
 DAY 133  
 DAY-OF-WEEK 134  
 DB 76  
 DBD 344  
 DBRM 355  
 DCLGEN 355, 361  
 Debugging 48  
 DECIMAL-POINT IS COMMA  
 50, 71  
 DECLARATIVES 60, 307  
 DECLARE TABLE 358  
 Definition 24, 57  
 DELETE 327, 399  
 DEPENDING ON 157, 251  
 DESCENDING KEY 262, 337  
 Dezimalpunkt 50  
 Dezimalzeichen 71  
 DFHCOMMAREA 380  
 DFHRESP() 383  
 Dialekt 14  
 DISABLED 402  
 DISPLAY 119  
 AT 120  
 LOW-VALUES 123  
 ON EXCEPTION 121  
 UPON 121  
 USAGE 81  
 Distributed Program Link 388  
 DIVIDE 148  
 Format 1 149  
 Format 2 150  
 Format 3 150  
 Dividieren 138  
 DIVISION 23, 61  
 Division 148  
 Rest 150  
 DL/1 346  
 Druckdatei 293, 302  
 Druckersteuerung 303  
 DSIDERR 402  
 DUPKEY 402  
 DUPREC 402  
 Dynamische Dateizuweisung  
 286  
**E**  
 E/A-Anweisung 306, 330  
 EBCDIC 53, 294  
 ED 81  
 EIB 379  
 EIBDS 386  
 EIBRCODE 386  
 EIBREP 382  
 EIBRESP 382  
 EIBRSRCE 386  
 Ein/Ausgabe 286  
 Eindimensionale Tabelle 240  
 Adressierung 241  
 Einfügungssymbol (B 0 /) 70  
 Eingabe 123  
 Datenprüfung 128  
 numerische Felder 126  
 Eingabemodus 296  
 Eingangspunkt 218  
 Elementnummer 239, 245  
 ELSE 181, 182  
 Empfangsfeld 275  
 END PROGRAM 60  
 ENDBR 400  
 ENDFILE 402  
 END-IF 182  
 END-READ 323  
 Endwert 168  
 ENTRY 346  
 Entscheidungstabelle 205  
 ENVIRONMENT DIVISION  
 24, 47, 284  
 EOC 396  
 EODS 396  
 EQUAL 398  
 ERASE 393  
 ERASEAUP 393  
 Eröffnungsmodus 306  
 ERROR 307  
 Erweiterungsmodus 297  
 ESCAPE-Sequenz 303  
 EVALUATE 200  
 ALSO 202  
 ANY 204  
 FALSE 203  
 THRU 204  
 TRUE 203  
 EXCEPTION 307  
 Exception 60  
 EXEC CICS 375  
 EXEC SQL 355  
 EXECUTE INTERFACE  
 BLOCK 385  
 EXECUTE SQL 355  
 EXECUTION INTERFACE  
 BLOCK 380  
 EXIT 176  
 EXIT PERFORM 155, 177  
 EXIT PROGRAM 226  
 EXIT SECTION 178  
 EXKLUSIV-ODER-Verknüp-  
 fung 36  
 EXTERNAL 233  
 Externe Unterprogramme  
 CALL 217  
 CANCEL 231  
 EXIT PROGRAM 226  
 EXTERNAL 233  
 GLOBAL 237  
 GOBACK 232  
 INITIAL 238  
 LINKAGE SECTION 219  
 LOCAL-STORAGE SEC-  
 TION 229  
 Rekursives COBOL-Pro-  
 gramm 228  
**F**  
 FALSE 98, 115  
 FCT 397  
 FD 58, 291  
 Fehler  
 logischer 316  
 permanenter 316  
 Fehlerbehandlung 125  
 Fehlercode 287, 315

- 21 324
- 22 324
- 23 323, 329
- Feld**
  - Anzahl 279
  - aufteilen 278
  - automatisches 229
  - definieren 239
  - Inhalt ändern durch
    - Unterprogramm 222, 225, 226
  - Initialisierung 220
  - Länge übergeben an
    - Unterprogramm 225
  - numerisches 126
  - statische 228
  - zusammenfügen 275
- Feldbeschreibung 92
- Feldkategorie 104
- Festkommazahl 28
- FETCH 363
- FIELD 345
- Figurative Konstante 30
  - ALL 31
  - HIGH-VALUE 31
  - LOW-VALUE 31
  - QUOTE 31
  - SPACE 30
  - ZERO 30
- FILE 397
- File Control Table 397
- FILE SECTION 25, 58
  - FD 58
  - REDEFINES 96
  - SELECT 58
- FILE STATUS 287, 315
- FILE-CONTROL 25, 57
- FILENOTFOUND 402
- File-Sharing-Konflikt 289, 317
- File-Status
  - 0 315
  - 1 315
  - 2 315
  - 3 316
  - 4 316
  - 5 317
  - 6 317
  - 9 317
- File-Status 0 287
- File-Status 1 287
- File-Status 3 288
- File-Status 4 288
- File-Status 5 288
- File-Status 6 289
- File-Status 9 289
- FILLER 92, 121
- Fixed-form reference format
  - siehe Codierformat 39
- Fixed-List-Select 367
- Fließkommazahl 29
- FLOAT-EXTENDED 85
- FLOAT-LONG 85
- FLOAT-SHORT 85
- FOOTING 293
- Fortsetzungsbereich 40
- Fortsetzungszeile 41
- Free-form reference format
  - siehe Codierformat 42
- FREEKB 394
- FROM 393
- FRSET 394
- Funktion 51
  - selbst geschriebene 232
- Funktionstaste 129
- G**
  - Geschachtelter Programmauf-  
ruf 216
  - Geschichte 15
  - GIVING 143, 145
  - Gleitendes Vorzeichen 78
  - GLOBAL 237
  - GO TO 18, 153
  - GO TO DEPENDING ON 156
    - Fehlerroutine 158
  - GOBACK 232
  - GTEQ 398
- H**
  - HANDLE CONDITION 383
  - Hexadezimaler Literal 28
  - HIGH-VALUE 31
- I**
  - I-0-CONTROL 25
  - ID 82
  - IDENTIFICATION DIVISION
    - 24, 45
    - COMMON 46
    - INITIAL 46
    - PROGRAM-ID 46
    - RECURSIVE 46
  - IF 181
    - ELSE 181
    - END-IF 182
    - POINTER 195
    - Schachtelung 184
    - THEN 181
  - IGNORE CONDITION 385
  - ILLOGIC 402
  - IMMEDIATE 389
  - IMS 343
    - Datenbankzugriff 348
    - DBD 344
    - PCB 345
    - PSB 345
    - Segmente 343
    - SSA 350
    - Statuscode 352
  - INCLUDE SQLCA 359
  - INDEX 86
  - Index 239
  - Indexbestand 311
  - Indexdatenname 252, 253
  - INDEXED 312
- INDEXED BY 248
- Indexname 252
- Index-sequenzielle Datei 311
  - DATA DIVISION 318
  - ENVIRONMENT DIVI-  
SION 311
  - Fehlercodes 315
  - PROCEDURE DIVISION  
318
- Indikator-Variable 362
- Indizierung 246
- INITIAL 46, 238
- Initialisierung 238
- INITIALIZE 109, 220
  - REPLACING 112
  - TO VALUE 111
- IN-LINE PERFORM 164
- INPUT PROCEDURE 338
- INPUTMSG 388
- INPUTMSGLEN 388
- INPUT-OUTPUT SECTION
  - 25, 57
- INSPECT 267
  - AFTER 270
  - BEFORE 270
  - CONVERTING 273
  - REPLACING 271
  - TALLYING 269
- Interpunktion 32
- INVALID KEY 322, 324, 327
- INVMP5Z 396
- INVPARTN 396
- INVREQ 389, 396, 402
- I-O-CONTROL 57
- IOERR 402
- IS INITIAL PROGRAM 218
- ISCINVREQ 402
- J**
  - JUSTIFIED 88, 105
- K**
  - Kapitel 23
  - Kapitelüberschrift 40
  - Kennzeichnerbindewort 26
  - KEYLEN 345
  - KFBA 346
  - Klasse 55
  - Klassenbedingung 192
  - Klausel 23, 24, 40
  - Komma 32
  - Kommentar 40, 42
  - Kommentareintragung 24
  - Konstante 100
  - Kontextsensitives Schlüssel-  
wort 26
  - KSDS 397
- L**
  - Laufvariable 167, 169, 171
  - Laufvariablenfeld 167

- Leerzeichen 32  
 LENGERR 390, 402  
 LENGTH 101, 388  
 LENGTH OF 225  
 Lesen  
   wahlfreies 322  
 LINAGE 293, 302  
 LINK 387, 390  
 LINKAGE SECTION 25, 59,  
   219  
   INITIALIZE 220  
 Literal 27  
   Boolesche Literale 29  
   fortsetzen 42  
   Hexadezimale Literale 28  
   Nationale Literale 30  
   Nicht numerische Literale  
   27  
   Numerische Literale 28  
   Verketteten 28  
 LOAD 387  
 LOADING 403  
 LOCAL-STORAGE SECTION  
   25, 46, 58, 229  
 LOCK 306  
 LOCK MODE 290  
 LOCKED 403  
 Logischer Fehler 316  
 LOW-VALUE 31
- M**
- MAP 393  
 MAPFAIL 396  
 MAPONLY 393  
 Mehrdimensionale Tabelle 243  
   Adressierung 244  
   durchsuchen 260  
 MERGE 341  
 Merkmale 113, 114  
 Mischen 335, 341  
   MERGE 341  
 Modularisierung 215  
 MOVE 103  
 MOVE CORR 107  
 Multiplikation 146  
 Multiplizieren 138  
 MULTIPLY 146  
   Format 1 147  
   Format 2 148  
 MVS 392
- N**
- NATIONAL 86  
 Nationales Literal 30  
 NATIVE 53  
 NEGATIVE 191  
 NEXT 321  
 NEXT SENTENCE 183  
 Nicht numerisches Literal 27  
 NO REWIND 297  
 NOHANDLE 385  
 Normalindex 246  
 Normalindizierung 246  
 NOSPACE 403  
 NOT AT END 300, 321  
 NOT ON SIZE ERROR 141  
 NOTAUTH 389, 403  
 NOTFND 403  
 NOTOPEN 403  
 Null  
   führende unterdrücken  
   72  
 Nullenunterdrückung 73  
 NUMERIC 192  
 Numerisches Datenfeld 67  
 Numerisches druckaufbereite-  
   tes Datenfeld 71  
 Numerisches Feld 126  
 Numerisches Literal 28
- O**
- OBJECT REFERENCE 86  
 OBJECT-COMPUTER 25, 48  
 Objektmodul 235  
 OCCURS 239  
 ODER-Verknüpfung 36  
 OMITTED 221  
 ON SIZE ERROR 140  
 OO-COBOL 14  
 OPEN 296, 318  
   NO REWIND 297  
   SHARING 319  
   Update-Modus 319  
 OPEN I-O 319  
 Operator 69, 138  
   Arithmetische Operatoren  
   36  
   Boolesche Operatoren 36  
   Vergleichsoperatoren 36  
 OPTIONAL 284  
 OR 196  
 Ordnungsbegriff 335  
 Organisationsform 283, 287  
   sequenzielle 283  
 ORGANIZATION 287, 312  
 OUT-OF-LINE PERFORM 161  
 OUTPUT PROCEDURE 339  
 Overflow 278
- P**
- PACKED-DECIMAL 82  
 PADDING CHARACTER 290  
 Paragraph 23, 61  
 Paragraphenname 40  
 Parameter  
   an Unterprogramm über-  
   geben 219, 221  
   Unterprogramm 232  
 PARENT 345  
 PARTNFAIL 396  
 PCB 346  
 PCB-Maske 349  
 PERFORM 18, 160  
   AFTER 169  
   EXIT PERFORM 155, 177  
   EXIT SECTION 178  
   geschachtelt 174  
   IN-LINE PERFORM 164  
   Interner Ablauf 171  
   OUT-OF-LINE PER-  
   FORM 161  
   THRU 162  
   TIMES 164  
   UNTIL 164  
   VARYING 167  
   WITH TEST BEFORE/  
   AFTER 165  
 Permanenter Fehler 316  
 PGMIDERR 389  
 PICTURE 64  
   - 71  
   0 71  
   1 69  
   9 66, 67, 71  
   , 71  
   . 71  
   \* 71  
   / 71  
   + 71  
   \$ 71  
   A 65, 66  
   B 65, 71  
   CR 71  
   DB 71  
   N 66  
   P 68, 71  
   S 68  
   V 67, 71  
   X 66  
   Z 71  
 PICTURE SYMBOL 51  
 POINTER 86, 117, 277, 282  
 POP HANDLE 385  
 POSITIVE 191  
 Potenzieren 138  
 Precompiler 355  
 Prepared-Statement 358, 364,  
   365  
 PREVIOUS 321  
 PROCEDURE DIVISION 24,  
   59, 295  
   USING 221  
 PROCOPT 345  
 PROGRAM COLLATING  
   SEQUENCE 48  
 PROGRAM POINTER 87  
 PROGRAM-ID 46  
   COMMON 236  
   INITIAL PROGRAM 218  
   RECURSIVE 229  
 Programm  
   Schachtelung 235  
 Programmaufruf  
   geschachtelter 216  
 Programmidentifikationsbe-  
   reich 41  
 Programmieren  
   strukturiertes 154

- Programmiererwort 27  
 Programmkontrolle 387  
 Programmname 41  
 Programmschleife 160  
 Programmstruktur 23  
 Programmtextbereich 40  
 Programmverbindung 216  
 PSB 345  
 Pseudoconversational Processing 379  
 Pseudotext 213  
 Puffer 290, 298, 301  
 Punkt 23, 33  
 PUSH HANDLE 385
- Q**
- QUOTE 31
- R**
- RAISING 60  
 RBA 398  
 RDATT 396  
 READ 298, 320, 397  
   AT END 321  
   END-READ 323  
   INTO 298  
   INVALID KEY 322  
   NEXT 321  
   NOT AT END 321  
   PREVIOUS 321  
 READNEXT 401  
 READPREV 401  
 REBIND 357  
 RECEIVE MAP 382, 394, 396  
 Rechenfeld 138  
 Rechenoperation  
   Datendefinition 82  
   Prioritäten 139  
 Rechenoperationen  
   Datendefinition 82  
 Rechtsbündig 88  
 RECORD CONTAINS 291  
 RECORD DELIMITER 290  
 RECORD KEY 313  
 RECURSIVE 46, 229  
 REDEFINES 80, 94, 99, 233, 256  
 Referenz-Modifikation 117  
 Rekursives COBOL-Programm 228  
 Relative Adresse 250  
 RELEASE 292, 339, 340  
 REMAINDER  
   DIVIDE 150  
 RENAMES 98  
 REPLACE 213  
 REPLACING 112, 271  
 REPORT SECTION 25  
 REPOSITORY 25, 55  
 REQID 401  
 RESERVE 290  
 Reserviertes Wort 26
- RESETBR 400  
 RETURN 292, 339, 340, 379, 380, 387, 388  
 RETURNING 232  
 REWRITE 304, 324, 399  
   FILE 304  
   FROM 304  
 RIDFLD 398, 401  
 ROLLEDBACK 390  
 RRN 398  
 Rundklammer 35
- S**
- Satz 23, 182  
 Satzaufbau 291  
 Satzlänge  
   variable 291, 292  
 Satzschlüssel 313  
 Satzsperrung 317  
 Schachtelung  
   Programme 235  
 Schalter 52, 96, 98, 114  
 Schleife 160  
   fußgesteuerte 166  
   geschachtelte 169  
 Schlüssel 311  
 Schlüsselfehler 315, 322, 324  
 Schlüsselfeld 338  
 Schlüsselwort 26  
   Kontextsensitive Schlüsselwörter 26  
 Schriftart 303  
 Schrittweite 168  
 SCREEN SECTION 25  
 SD 336  
 SEARCH 249, 258  
   AT END 259  
   VARYING 258  
   WHEN 259  
 SEARCH ALL 263  
 SECTION 23, 25, 61  
 SEGM 345  
 Segment (IMS) 343  
 Segmentnummer 27  
 Segmentsuchargument 350  
 SELECT 58, 284  
   ACCESS MODE 287, 312  
   ASSIGN 285  
   FILE STATUS 287  
   OPTIONAL 284  
   ORGANIZATION 287, 312  
   RECORD KEY 313  
   Sortierdateien 335  
 Selektions-Objekt 201  
 Selektions-Subjekt 201  
 Semikolon 32  
 SEND CONTROL 394  
 SEND MAP 382, 392  
 Sendefeld 275  
 SENFLD 345  
 SENSEG 345  
 SEPARATE CHARACTER 90
- Sequenz 235  
 Sequenzielle Datei 283  
 SET 98, 113, 251  
 SHARING 319  
 SIGN 89  
 Sonderregister 37  
 SORT 262, 337  
   INPUT PROCEDURE 338  
   OUTPUT PROCEDURE 339  
 Sortierdatei 335  
 Sortierdateiname 337  
 Sortieren 335  
   MERGE 341  
   RELEASE 340  
   RETURN 340  
   SORT 337  
 Sortierfolge 53  
 Sortierschlüssel 262, 338  
 SORT-MERGE-Modul 335  
 SOURCE-COMPUTER 25, 47  
 SPACE 30  
 SPECIAL-NAMES 25, 49  
 Speicher  
   freigeben im Unterprogramm 231  
 Speicherplatzreservierung 82  
 Spezialindex  
   relative Adresse 250  
 Spezialindizierung 248  
 Split-Schlüssel 314  
 Sprachelement 26  
 Sprungmarke 18  
 SQL 354, 355  
   Cursor 363  
   DCLGEN 355  
   dynamisches 364  
   EXEC SQL 355  
   Platzhalter 365  
   Precompiler 355  
   SQLCA 359  
   SQLDA 368  
   statisches 361  
 SQLCA 358, 359  
   Felder 360  
 SQLCODE 360  
   Abfrage 360  
 SQLD 369  
 SQLDA 368  
 SQLDABC 369  
 SQLDAID 369  
 SQLDATA 369  
 SQLERRD(3) 360  
 SQLERRM 360  
 SQLERRMC 360  
 SQLIND 369  
 SQLLEN 369  
 SQLN 369  
 SQLNAME 369  
 SQLTYPE 369  
 SQLVAR 369  
 SSA 350  
   qualifizierter 350  
   unqualifizierter 351, 352

- START 328  
   WITH LENGTH 329  
 STARTBR 400  
 Statische Daten 238  
 Statuscode 352  
 Steueranweisung  
   CONTINUE 199  
   EVALUATE 200  
   EXIT 176  
   EXIT PERFORM 177  
   EXIT SECTION 178  
   GO TO 153  
   GO TO DEPENDING ON  
     156  
   IF 181  
   PERFORM 160  
   STOP 134  
 STOP 134  
 STRING 275  
   DELIMITED BY 275  
   Overflow 278  
   POINTER 277  
   SIZE 275  
 Strukturiertes Programmieren  
   154  
 Stufenbezeichnung 40  
 Stufennummer 27, 40, 63  
   01 100  
   01–49 91  
   66 98  
   77 63  
   88 96  
 Subskribierung 246  
 Subskript 246  
 SUBSTRACT  
   Format 1 144  
   Format 2 145  
   Format 3 146  
 Substraktion 144  
 SUBTRACT 144  
 SUBTRACT CORR 146  
 Subtrahieren 138  
   mehrere Datenfelder  
   gleichzeitig 146  
 Suchen  
   binäres 261  
 SWITCH 52, 114  
 SYMBOLIC CHARACTERS 55  
 SYNCHRONIZED 89  
 SYNCONRETURN 388  
 Syntax 23  
   Alternative 39  
   Pflichteintrag 38  
   Wahlfreie Alternative 39  
   Wahlfreier Eintrag 38  
   Wiederholungen 39  
   Wörter in Großbuchsta-  
   ben 38  
   Wörter in Kleinbuchsta-  
   ben 38  
 SYSID 388  
 SYSIDERR 390, 403  
 Systemfehler 289
- T**  
 Tabelle 239  
   Anfangswert 256  
   durchsuchen 258  
   eindimensionale 240  
   mehrdimensionale 243  
   mehrdimensionale durch-  
   suchen 260  
   sequenziell durchsuchen  
   258  
   sortieren 262, 339  
 Tabellendefinition 239  
 Tabellenelement  
   Adressierung 241  
 Tabellenverarbeitung  
   DEPENDING ON 251  
   INDEXED BY 248  
   INITIALIZE 113  
   mehrdimensionale Tabel-  
   len 243  
   Normalindizierung 246  
   OCCURS 239  
   REDEFINES 256  
   SEARCH 258  
   SEARCH ALL 263  
   SORT 262  
   Spezialindizierung 248  
   Subskribierung 246  
   USAGE INDEX 253  
   VALUE 256  
 TALLYING 269, 279  
 Tastatureingabe 123  
 Teilüberschrift 40  
 TERMERR 390  
 Testhilfezeile 40  
 THEN 181, 183  
 THROUGH 98  
 TIME 134  
 TIMES 164  
 TO VALUE 111  
 TOP 293  
 Transaktionsorientiert 379  
 TRANSID 379, 387, 388  
 TRUE 115
- U**  
 Überlauf 140, 141, 278  
 UND-Verknüpfung 36  
 UNEXPIN 396  
 Unicode 86  
 UNLOCK 399  
 UNSTRING 278  
   ALL 281  
   POINTER 282  
   TALLYING 279  
 Unterprogramm 159, 215  
   Daten zurückgeben 232  
   Datenfelder übergeben  
   219  
   Endpunkt 176  
   Feldinhalt verändern 225,  
   226  
   Feldlänge übergeben 225
- Parameter 232  
 Parameter übergeben 219,  
   221  
 Pointer 223  
 Rückgabewert 232  
 Speicher freigeben 231  
 Status 218  
 Steuerung an Hauptpro-  
 gramm zurückgeben  
   226, 232  
   Variable 218  
 UNTIL 164  
 Update-Modus 297  
 USAGE 80  
   BINARY 82  
   BINARY-CHAR 83  
   BINARY-DOUBLE 84  
   BINARY-LONG 84  
   BINARY-SHORT 84  
   COMP 82  
   FLOAT-EXTENDED 85  
   FLOAT-LONG 85  
   FLOAT-SHORT 85  
   INDEX 86  
   NATIONAL 86  
   OBJECT REFERENCE 86  
   PACKED-DECIMAL 82  
   POINTER 86  
   PROGRAM POINTER 87  
 USAGE BIT 100  
 USAGE COMP 82  
 USAGE DISPLAY 30, 81  
 USAGE INDEX 253  
 USAGE NATIONAL 30  
 USAGE PACKED-DECIMAL  
   82  
 USE 306, 330  
 USING 60, 219, 221  
   ADDRESS OF 223  
   BY CONTENT 225  
   BY REFERENCE 222  
   BY VALUE 226  
   LENGTH OF 225  
   RETURNING 232  
 UTF-16 30
- V**  
 VALUE 80, 256  
 Variable  
   globale 237  
 Variable Satzlänge 291  
 VARYING 167  
 Varying-List-Select 370  
 Verb 26  
 Vergleich 54  
 Vergleichsbedingung 187  
 Vergleichsoperator 36, 351  
 Verknüpfen 26  
 Verknüpfung 197  
 Vorzeichen 75, 83, 89  
   Gleitendes 78  
 Vorzeichenbedingung 191  
 VSAM 396  
 VSE 392

**W**

Wahlfreies Lesen 322  
Wahlwort 26  
Wahrheitswert 197  
Währungssymbol 79  
WHENEVER 360  
WITH DEBUGGING MODE  
40, 48  
WITH DUPLICATES IN  
ORDER 339  
WITH LENGTH 329  
WITH LOCK 300  
WITH TEST BEFORE/AFTER  
165  
WORKING-STORAGE SEC-  
TION 25, 46, 58  
Wort 23  
Reserviertes 26

WRITE 300, 323, 398  
ADVANCING 302  
FILE 301  
FROM 301  
INVALID KEY 324  
LINAGE 302

**X**

XCTL 387, 390  
XCTRL 387

**Y**

YYYYDDD 133  
YYYYMMDD 133

**Z**

Zeichen  
ersetzen 268  
zählen 267  
Zeichenkettenverarbeitung  
INSPECT 267  
STRING 275  
UNSTRING 278  
Zeichensatz 37  
Zeilennummerierung 40  
Zeilenvorschub 302  
ZERO 30, 191  
Zugriffsmodus 287, 312  
Zusammengesetzte Bedin-  
gung 195