NEVADA COBOL™

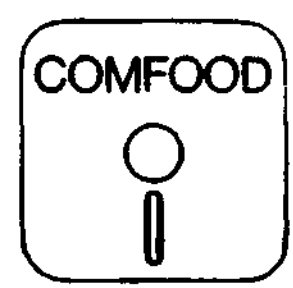
deutsch

ELLIS COMPUTING™
SOFTWARE TECHNOLOGY

ComFood

ComFood GmbH, Fleßkamp 24, 4400 Münster

– Wir füttern Ihren Computer –



NEVADA — COBOL
Version 3.3

Handbuch Deutsch

Übersetzt von Frank Bensberg

ComFood Software GmbH
4400 Münster, Flaßkamp 24
Tel: 0251/719768

Achtung : Um ein schnelles Nachschlagen von Informationen zu ermöglichen, beziehen sich die ersten vier Kapitel dieses Handbuchs auf die vier Abschnitte eines COBOL-Programms. Eine Einleitung und Informationen über die Bedienung des Compilers finden Sie in Kapitel V und VI. Sollten Sie mit der Programmiersprache COBOL noch nicht so vertraut sein, enthält das Kapitel IX eine kurze Einführung in das Konzept von COBOL.

Inhaltsverzeichnis

Kapitel I

IDENTIFICATION DIVISION - Feld

Funktion : Dieses Feld dient zur Dokumentation des Quellprogramms

Syntax :

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  Programmname.
[ AUTHOR.          Kommentar  . ]
[ INSTALLATION.    Kommentar  . ]
[ DATE-COMPILED.   Kommentar  . ]
[ SECURITY.        Kommentar  . ]
```

Hinweis : Dieses Feld dient ausschließlich zur Dokumentation und wird vom Compiler als Kommentar aufgefasst. Der Programmabschnitt wird nur auf gültige Schlüsselworte untersucht, die in Großbuchstaben geschrieben werden müssen.

Beispiel :

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID.           TEST1.
0003 AUTHOR.               ELLIS COMPUTING.
0004 INSTALLATION.         SAN FRANCISCO.
0005 DATE-WRITTEN.         DEZEMBER 1985.
0006 DATE-COMPILED.        DEZEMBER 1985.
0007 SECURITY.             NO COPYRIGHT.
0008*  Kommentare werden durch das Zeichen "*" gekennzeichnet,
       das direkt hinter der Zeilennummer in Spalte 5 steht.
       Diese Zeilen dürfen auch Kleinbuchstaben enthalten
```

COPY - Befehl

Funktion : Der COPY-Befehl fügt beim Compilieren Text in das Quellprogramm ein

Syntax : COPY u:Filename .

Regeln :

1. Ein COPY-Befehl darf nicht innerhalb eines anderen COPY-Befehls auftauchen.

2. Die Angabe des Laufwerks u: ist wahlweise. Falls keine Angabe erfolgt, wird das aktuelle Laufwerk angesprochen.

3. Der COPY-Befehl beginnt normalerweise in der Spalte 7 . Üblicherweise geht dem Befehl ein Leerzeichen (Space) voraus; das Ende des Befehls wird durch einen Dezimalpunkt gekennzeichenet.

4. Der Filetyp kann beim COPY-Befehl nicht angegeben werden; der Typ ist immer .CBL .

Beispiel :

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID.    TESTCOPY.
0003  COPY A:FILE1.
0008  COPY A:FILE2.
0015  COPY B:FILE3.
```

Durch eine besondere Zeilennummerierung könnte man beispielsweise folgende Zeilen unter dem Namen FILE1.CBL einfach in den Quelltext einfügen :

```
0004 AUTHOR.        ELLIS COMPUTING.
0005 INSTALLATION.  SAN FRANCISCO PROGRAMMING CENTER.
0006 DATE-WRITTEN.  25 Januar 1982.
0007 DATE-COMPILED.25 Januar 1982.
```

Kapitel II

ENVIRONMENT DIVISION - Konfigurationsfeld

Funktion : Erkennung des Systems, auf dem das Programm entwickelt und compiliert wurde.

Syntax :

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. Angabe des Systems [WITH DEBUGGING MODE].
OBJECT-COMPUTER. Angabe des Systems

[MEMORY SIZE ganze Zahl-1 < CHARACTERS >]
< WORDS >
< MODULS >

[MEMORY BEGINNING ganze Zahl-1 ENDING ganze Zahl-2]

[PROGRAM COLLATING SEQUENCE IS ASCII] .

SPECIAL-NAMES. [CURRENCY SIGN IS Stringvariable-1]

[DECIMAL-POINT IS COMMA] .

Regeln :

1.Das MEMORY SIZE-Feld gibt an, wieviel Speicherplatz das Programm verbraucht (höchstes Addressbyte). Das MEMORY BEGINNING -Feld gibt die Anfangs- und Endaddresse des Programms an. Die Endaddresse des Programms ist besonders für den Aufruf von Unterprogrammen mit dem CALL-Befehl wichtig. Falls diese Felder im Programm nicht angegeben werden, beansprucht das Programm den ganzen verfügbaren Speicherplatz.

2. Der COBOL-Compiler nimmt beim Compilationsvorgang immer den ganzen verfügbaren Speicherplatz in Anspruch.

3. Wenn der Befehl "WITH DEBUGGING MODE" gegeben wird, werden außerdem auch die Programmzeilen compiliert, in deren fünften Spalte der Buchstabe "D" steht.

4. MACHINE COLLATING SEQUENCE IS ASCII wird vom Compiler als Kommentar interpretiert, weil der Rechner die Zeichen im ASCII Format speichert.

5. Der Zeichenstring, der durch den Befehl CURRENCY SIGN IS angesprochen wird, ist für den PICTURE-Befehl notwendig, um das Währungszeichen zu bestimmen. Dieser String hat die Länge von genau einem Zeichen. Dieses Zeichen darf jedoch nicht eines der folgenden sein :

a. Zahlen von 0-9
b. Buchstaben A, B, C, D, L, P, R, S, V, X, Z oder Leertaste (Space)
c. spezifische Zeichen wie " *, +, -, ;, (,), ", /, =," und "."

Falls dieser Befehl nicht gegeben wird, wird beim PICTURE Befehl nur das Zeichen "$" anerkannt.

6. Der Ausdruck DECIMAL-POINT IS COMMA bedeutet, daß bei der Definition von Strings der Dezimalpunkt und das Komma ausgetauscht werden (wie in Deutschland üblich). Dies gilt auch für die Strings des PICTURE-Befehls.

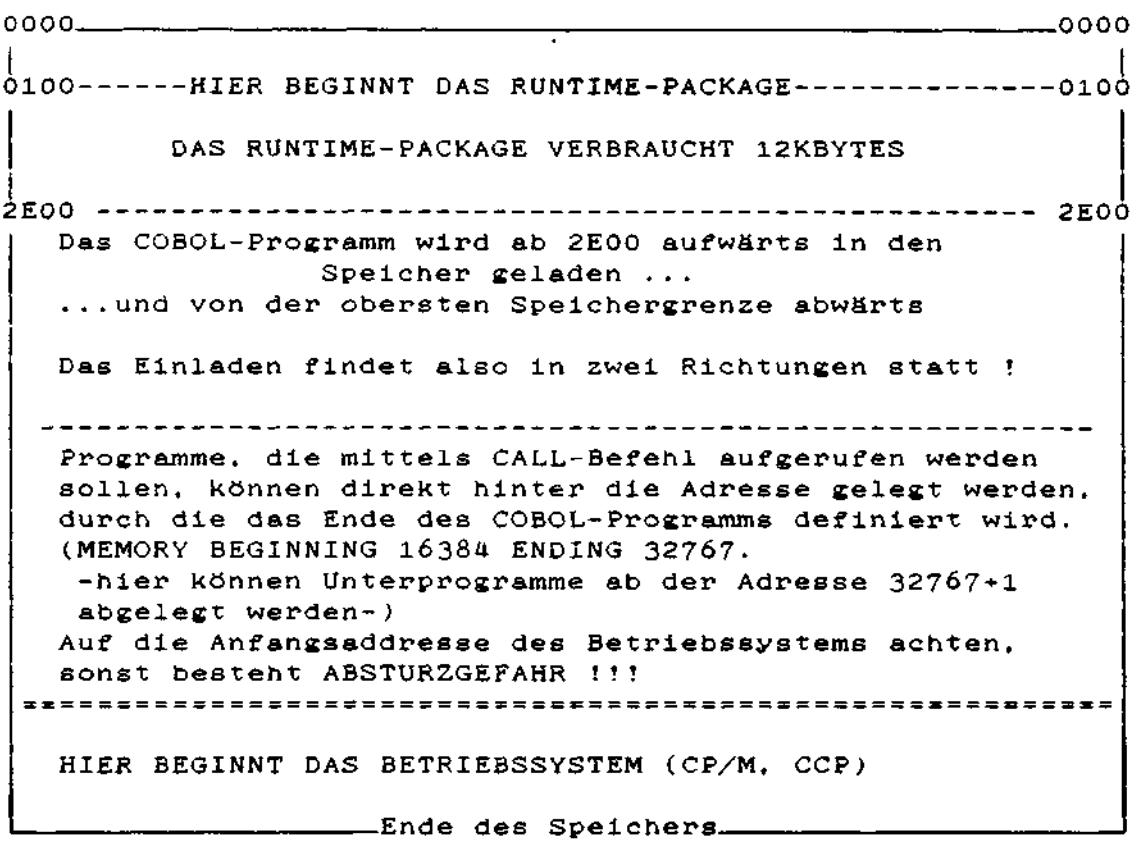
7. ganze Zahl-1 und ganze Zahl-2 sind Adressen. Durch die Anpassung dieser systemabhängigen Adressen können die Programme auch auf andere Rechner übertragen werden.

Beispiel :

```
0011 ENVIRONMENT DIVISION.
0012 CONFIGURATION SECTION.
0013 SOURCE-COMPUTER. 8080-CPU
0014      WITH DEBUGGING MODE.
0015 OBJECT-COMPUTER. 8080-CPU
0016      MEMORY SIZE 16383 CHARACTERS.
0017* Falls CALL-Aufrufe erfolgen sollen, muß Zeile 0016
0018* unbedingt so aussehen :
0016      MEMORY BEGINNING 16384 ENDING 32767.
```

SPEICHERAUFTEILUNG BEI NEVADA-COBOL

MEMORY-MAP



INPUT-OUTPUT - Befehl

Funktion : Festlegen der Files und der dazugehörigen externen Datenspeicher.

Syntax :

```
INPUT-OUTPUT SECTION.
FILE-CONTROL.
  SELECT Filename-1 ASSIGN TO

                          < PRINTER >
                          < DISK    >

    [. ORGANIZATION IS < SEQUENTIAL > ]
                       < RELATIVE   >
    [. ACCESS MODE IS  < SEQUENTIAL > ]
                       < RANDOM     >
    [  RELATIVE KEY IS Datenname-1   ]

    [  RECORD DELIMITER IS STANDARD  ]

    [.FILE STATUS IS  Datenname-2  ].

I-O-CONTROL.
    SAME [ RECORD ] AREA FOR Filename-1, Filename-2, ...
```

Regeln :

1. Die Filenamen müssen eindeutig definiert werden. Ein Filen kann nur für ein File verwendet werden.

2. Das RECORD DELIMITER-Feld darf nicht in Zusammenhang mit einem Drucker initialisiert werden.

3. Falls eine Festlegung im RECORD DELIMITER-Feld erfolgt, so haben alle Daten-Records eine variable Länge und werden du das Carriage-Return Zeichen (CR, Wagenrücklauf) oder durch Line-Feed getrennt.

4. Beim Schreiben wird der Daten-Record, der mit dem RECORD DELIMITER definiert wurde, von rechts nach links auf das erste Zeichen untersucht, das kein Leerzeichen ist.
Der Begrenzer (Delimiter) wird dann nun eine Position hinter dieses Zeichen gesetzt. Danach erst erfolgt das Abspeichern de Daten-Records.

5. Das Einlesen eines Records in den Speicher geschieht so lange, bis entweder der Speicher voll ist, oder im Record ein Begrenzer gefunden wird. Der Begrenzer wird dabei nicht eingelesen.

6. Die Variable DATENNAME-2 (FILE STATUS) muß im WORKING-STORA Feld als 2 Zeichen langes Datenfeld definiert werden.

Position 1 (Statusbyte 1)	Position 2 (Statusbyte 2)
0=Bearbeitung erfolgreich beendet	0=keine Informationen
1=Ende des Records erreicht	verfügbar
2=ungültiger Schlüssel (key)	X=siehe Fehlercodes
3=permanenter Fehler	
9=siehe Statusbyte 2	

7. ORGANIZATION IS RELATIVE erlaubt nur die Bearbeitung von Disk-Files mit festgelegter Länge. Falls dieses Feld nicht angegeben wird, so wird ORGANIZATION IS SEQUENTIAL angenommen.

8. RELATIVE KEY ordnet jedem Record eine ganze Zahl zu, die seine logische Position im Datenfile und Datenspeicher angibt. Diese Zahl ist muß größer als Null sein: so wird z.B. dem zehnten Record eines Files die Zahl 10 zugewiesen. Diese Zuordnung kann allerdings nur bei wahlfreiem (RANDOM) Dateizugriff erfolgen.

9. Es gilt : Blockgröße ist gleich RELATIVE KEY multipliziert mit der Recordlänge geteilt durch die physikalische Blöckgröße.

10. Der Wert von RELATIVE KEY ist immer eine positive ganze Zahl mit maximal 7 Dezimalstellen.

11. Das Feld SAME RECORD AREA dient ausschließlich zu Dokumentationszwecken.

12. Ein relatives File wird immer mit einer feststehenden Länge generiert....

13. Wenn kein Begrenzer (DELIMITER) definiert wird, werden die Records mit einer konstanten Länge geschrieben. Jeder Record ist dann so lang wie der längste Record des aktuellen Files.

14. Bei der Fehlermeldung INVALID KEY bleibt der Speicher für die Records leer (ist mit dem Wert 1A hexadezimal gefüllt).

15. Wenn Records mit festgelegter Länge gelesen werden und dabei der letzte Record kurz ist, so wird der Rest des Recordspeichers mit Leerzeichen gefüllt.

16. Falls das Einlesen der Records mit Begrenzer (DELIMITER) erfolgt, so kann es bei kürzeren Records passieren, daß die Zeichen, die sich hinter dem letzten gültigen Record-Zeichen im Speicher befinden, auch als Daten erkannt werden. Da dieses aber zu fehlerhaften Daten führt, ist es sinnvoll, den Datenspeicher vor dem Einlesen mit Leerzeichen (Spaces) zu füllen.

17. Beim Einlesen mit Begrenzern werden Tabulator-Zeichen (Wert 09 hex.) ebenfalls übertragen. Ein Expandieren dieses Zeichen in Leerzeichen wäre nicht sinnvoll, da der Code 09H eine gültige Komponente bei der Dezimaldarstellung sein kann. Daher ist eine Verwendung von dezimalen Datentypen (COMP-3) beim Arbeiten mit Begrenzern möglich. Unter dem Betriebssystem CP/M existiert das Programm PIP, das TAB-Zeichen expandieren kann. Genauere Hinweise finden Sie in einem CP/M Systemhandbuch.

18. Wird das Diskettenlaufwerk als Ausgabe/Eingabe gewählt, so müssen die Daten unbedingt auf das Diskettenlaufwerk ausgegeben werden; wird jedoch der Drucker angewählt, so besteht die Möglichkeit, zwischen Drucker und Diskettenlaufwerk zu wählen. Diese Entscheidung wird beim Compilieren und beim Starten eines Programms getroffen, je nachdem, welche Daten im FD-Feld (FD=File Description) angegeben werden.

Beispiel :

```
0021 INPUT-OUTPUT SECTION.
0022 FILE-CONTROL.
0023     SELECT DATENFILE                        (Name des Files)
0024           ASSIGN TO DISK                    (File befindet sich
                                                  auf Diskette)
0025           ORGANIZATION IS SEQUENTIAL        (Struktur des Files
                                                  ist sequentiell)
0026           ACCESS MODE IS SEQUENTIAL         (Bearbeitung ist
                                                  sequentiell)
0027           RECORD DELIMITER IS STANDARD      (Es wird der
                                                  Standard-Begrenzer
                                                  gewählt)
0028           STATUS IS STA-1.                  (Statusvariable
                                                  definieren)
0029     SELECT LISTING ASSIGN TO PRINTER.       (Druckerfile
                                                  definieren)
0030     SELECT KUNDENDATEI                      (Name des Files)
0031           ASSIGN TO DISK                    (File befindet sich
                                                  auf Diskette)
0032           ACCESS MODE IS RANDOM             (wahlfreier Zugriff
0033           RELATIVE KEY IS KEY3              (Relativen Schlüsse
                                                  definieren)
0034           STATUS IS STA-2.                  (Statusvariable
                                                  definieren)
```

Bitte beachten Sie auch die Beispielprogramme im Anhang !
Die Kommentare in Klammern ( ) sind nicht bestandteil des Programms!

COPY - Befehl

Funktion : Der COPY-Befehl fügt beim Compilieren Text in das Quellprogramme ein.

Syntax : COPY u:Filename .

Regeln :

1. Ein COPY-Befehl darf nicht bei der Ausführung eines anderen COPY-Befehls vorkommen.

2. Die Angabe des Laufwerks kann wahlweise erfolgen. Entfällt die Angabe, so wird das aktuelle Laufwerk angesprochen.

3. Der COPY-Befehl in Spalte 7.

4. Der Filetyp wird nicht angegeben. Er muß immer vom Typ .CBL sein.

Beispiel :

```
0011 ENVIRONMENT DIVISION.
0012  COPY A:FILE4.
0013* der folgende Befehl sucht nach dem File FILE5.CBL auf
0014* dem aktuellen Laufwerk
0015  COPY FILE5.
```

Kapitel III

DATA DIVISION-Feld

Funktion : Bestimmung der Eigenschaften eines Files.

Syntax :

DATA DIVISION.

FILE SECTION.
FD Filename

```
                                          <RECORDS>
     [, BLOCKS CONTAINS ganze Zahl-1      <CHARACTERS>]

          <RECORD IS OMITTED>
     LABEL <RECORDS ARE STANDARD>

                           <Datenname-1>
     VALUE OF FILE-ID IS <Zeichenstring-1>

           <RECORD IS>
     [ DATA <RECORDS ARE> Recordname-1 [Recordname-2] ].
```

Regeln :

1. Die Angabe BLOCK CONTAINS dient nur zur Dokumentation des Programms

2. LABEL RECORDS ARE STANDARD muß für alle Files auf Diskette benutzt werden, kann aber auch auf Files angewendet werden, deren Ausgabe auf Drucker erfolgen soll. Ebenso verhält es sich mit der Angabe der File-ID (VALUE OF FILE-ID IS): für Diskfiles muß dieses Feld definiert werden, für Druckerfiles ist es nicht notwendig.

3. Der Zeichenstring-1 ist bis zu 14 Zeichen lang und enthält den Filenamen sowie die Laufwerksangaben. Wird kein Laufwerk angesprochen, so wird mit dem aktuellen Laufwerk gearbeitet.

4. Um ein File direkt auf Drucker auszugeben, muß folgender Befehl gegeben werden : VALUE OF FILE-ID IS "A:PRINTER". Jeder andere Filename veranlasst die Ausgabe auf Diskette.

5. Die Anweisung LABEL RECORD IS OMITTED kann man nur auf Files anwenden, die auf Drucker ausgegeben werden (siehe SELECT-Feld). Die Ausgabe eines Files geschieht direkt auf Drucker und kann beim Programmablauf nicht beeinflußt werden. Falls dieses Feld angegeben wird, so darf gleichzeitig nicht das Feld VALUE OF FILE-ID festgelegt werden.

Wenn in einem COBOL-Programm der Befehl LABEL RECORDS ARE STANDARD zusammen mit einer (im INPUT-OUTPUT Feld definierten) Druckerdatei benutzt wird, dann muß der Befehl VALUE OF FILE-ID benutzt werden. Dies hat den Vorteil, daß der Anwender beim Programmablauf oder Compilieren die Ausgabe auf Diskette umlenken kann.

Beim Compilieren kann die Ausgabe auf Drucker erfolgen, indem ein Zeichenstring eingegeben wird, der das Schlüsselwort "PRINTER" enthält. Jede andere Eingabe wird als normaler Filename behandelt und die vorhandenen Informationen werden unter diesem Namen auf Diskette geschrieben. Wenn das File noch nicht im Directory der Diskette vorhanden ist, wird es erzeugt.

Beim Programmablauf kann die Ausgabe auf Drucker erfolgen, indem anstatt des Zeichenstrings ein Daten-Name verwendet wird, der wiederum das Schlüsselwort "PRINTER" enthält, wenn die Informationen ausgedruckt werden sollen. Im Falle, daß das Schlüsselwort nicht im Daten-Name enthalten ist, werden die Daten unter dem vorhandenen Daten-Namen auf Diskette geschrieben.

Beispiel :

```
0041 DATA DIVISION.
0042 FILE SECTION.
0043 FD  ARBEITSFILE
0043* File heißt Arbeitsfile
0044          LABEL RECORDS ARE STANDARD
0044          VALUE OF FILE-ID IS "A:MASTER.DAT"
0044* das Arbeitsfile wird unter dem Namen MASTER.DAT bearbeitet
0045          DATA RECORDS ARE  EINKOMMEN, STEUERN.
0045* die Datenrecords Einkommen und Steuern werden definiert
0066 FD  LISTING  LABEL RECORDS ARE STANDARD
0067* die nächsten Zeilen leiten die Daten direkt an den Drucker;
    * siehe den CP/M STAT-Befehl, um den Drucker anzusteuern
    * (mit LST:).
0067          VALUE OF FILE-ID IS "PRINTER"
0067* Schlüsselwort PRINTER veranlasst Ausgabe auf Drucker
0068          DATA RECORD IS PRINT-LINE.
0068* PRINT-LINE ist der Record
0100 FD  THE-SOURCE  LABEL RECORDS ARE STANDARD
0101          VALUE OF FILE-ID IS THE-FILE
0102          DATA RECORD IS DISK-IN.
0103 FD  LIST-SPOOL
0104          LABEL RECORDS ARE STANDARD
0105* die nächste Zeile speichert Daten auf Diskette, um diese
    * dann später ausdrucken zu lassen; siehe auch das CP/M-
    * Kommando TYPE kombiniert mit CONTROL-P
0105          VALUE OF FILE-ID IS "B:LIST.TXT"
0105* das File wird unter LIST.TXT auf Laufwerk B: bearbeitet
0106          DATA RECORD IS PRT-LINE.
0106* PRT-LINE ist der Record
0107 FD   LIST2
0108* folgende Zeile sendet Daten direkt an den Drucker
0108          LABEL RECORD IS OMITTED
0108* VALUE OF FILE-ID darf hier nicht verwendet werden !
0109          DATA RECORD IS PRT-LINE2.
```

RECORD DESCRIPTION-Feld

Funktion : Festlegung der Eigenschaften von Datenfeldern

Syntax :

```
                <Daten-Name-1>
level-number <FILLER       > [ REDEFINES  Daten-Name-2 ]

                    [. OCCURS ganze Zahl-1 TIMES ]

                       <PIC     >
                    [. <PICTURE> IS <Zeichenstring-1>]

                       <SYNC          >   <LEFT >
                    [  <SYNCHRONIZED>  (<RIGHT>)]

                       <JUST      >
                    [  <JUSTIFIED>  RIGHT ]


                    [   BLANK WHEN ZERO    ]

                                          < COMP                >
                                          < COMP-3              >
                                          < DISPLAY             >
                                          < COMPUTATIONAL-3>
            {[. USAGE IS ]      < COMPUTATIONAL    >]...  .
```

WORKING-STORAGE SECTION.

(hier stehen normalerweise die üblichen Daten)
(und :)

```
                                  < [ALL]  Variable >
                                  < QUOTE > < HIGHT VALUE >
                                  < ZERO  > < LOW   VALUE >
                {. VALUE IS       < SPACE > ] ...
```

LINKAGE SECTION.

(hier stehen die normalen Anweisungen s.o., jedoch ohne VALUE-Befehl)

Regeln :

1. Die Variable LEVEL-NUMBER ist eine ganze Zahl aus dem Bereich von 1 bis 49 oder 77.

2. Der VALUE-Befehl darf nicht zusammen mit den Befehlen OCCURS und REDEFINES verwendet werden.

3. Der OCCURS-Befehl kann nicht für ein Feld benutzt werden, dessen Ordnungszahl (LEVEL-NUMBER) 01 oder 77 ist.

4. Das WORKING-STORAGE Feld muß unbedingt initialisiert werden, da sein anfänglicher Inhalt nicht festgelegt ist.

5. Außerdem können die Plural-Formen von den folgenden Befehlen benutzt werden : SPACE, ZERO, HIGH-VALUE, LOW-VALUE und QUOTE .

6. Der PICTURE-Befehl darf nur für elementare Felder verwendet werden.

7. Der Zeichenstring-1 darf maximal 30 Zeichen enthalten. Dieser String beschreibt die Eigenschaften der Daten, wie z.B. Datentyp, Größe und Wertzuweisung. Mit dem PICTURE-Befehl können insgesamt fünf Datentypen beschrieben werden :
- alphabetische Zeichenstrings enthalten die Symbole "A" und "B". Der Inhalt solcher Strings beschränkt sich auf die 26 Großbuchstaben des Alphabets sowie auf das Leerzeichen (Space).
- numerische Strings enthalten die Symbole "9","S" und "V"; die Anzahl der Stellen muß im Bereich 1-18 liegen. Die Darstellung der Zahlen erfolgt dezimal mit den Ziffern von 0-9, sowie den Vorzeichen "+" und "-".
- alphanumerische Strings enthalten die Symbole "A","X" und "9"; dieser Typ kann jedes ASCII-Zeichen enthalten, das auf Bildschirm oder Drucker ausgegeben werden kann.
- Alphanumerische Zeichenstrings mit Editierzeichen enthalten die Symbole "A", "X", "9", "B", "0", "/".
- Numerische Zeichenstrings mit Editierzeichen enthalten die Symbole "B", "V", "Z", "0", "9", ",", ".", "*", "+", "-", "$", "CR", "DB".

Die einzelnen Zeichen haben folgende Funktion :
- das Zeichen "A" steht für einen Buchstaben des Alphabets oder das Leerzeichen (Space).
- an die Position des Zeichens "B" rückt ein Leerzeichen.
- das Zeichen "S" repräsentiert ein Vorzeichen und steht deshalb im String des PICTURE-Befehls ganz links.
- das Zeichen "V" zeigt die Position des Dezimalpunkts an und darf im String nur einmal vorkommen.
- das Zeichen "X" steht für ein beliebiges ASCII-Zeichen, das auf Bildschirm oder Drucker ausgegeben werden kann.
- das Zeichen "Z" steht für ein numerisches Zeichen; wenn diese Position eine Null enthält, dann wird ein Leerzeichen eingesetzt; dieses Zeichen wird bei der Berechnung der Größe berücksichtigt.
- an die Position des Zeichens "0" wird jeweils die herkömmliche Null eingefügt; dieses Zeichen wird bei der Berechnung der Größe ebenfalls berücksichtigt.
- das Zeichen "9" steht für ein numerisches Zeichen (0-9); (wird bei Größenberechnung berücksichtigt).
- an die Stelle des Komma-Zeichens (",") tritt ein Komma; dies gilt auch für die folgenden Zeichen : ".", "+", "-", "*", "$" . (werden bei Größenberechnung berücksichtigt).
- an die Stelle der Symbole "CR" und "DB" rücken die Zeichen "+" und "-" (diese werden bei Größenberechnung mitgezählt).

8. Der "USAGE IS"-Befehl bestimmt das Format, in dem die numerischen Daten intern und extern gespeichert werden sollen. Üblicherweise wird DISPLAY angegeben; in diesem Fall wird das ASCII-Format angewählt. Das Vorzeichen wird durch Bit 7 dargestellt, wobei ein negatives Vorzeichen durch Setzen von Bit 7 erreicht wird, ein positives durch Löschen von Bit 7.
Wird eine negative Zahl als Zeichen ausgegeben, so erfolgt die Darstellung in Kleinbuchstaben (LOWER CASE).
COMPUTATIONAL-3 (COMP3) veranlasst den Compiler, die Daten im 2-Byte Dezimalformat abzulegen; das Vorzeichen wird durch die letzten vier Bits rechts angegeben :

0000 bedeutet positives Vorzeichen
0001 bedeutet negatives Vorzeichen

COMPUTATIONAL (COMP) bewirkt, daß die Daten im INTEL-8080 Binärformat abgespeichert werden. Bei dieser Methode ist die größte darstellbare Zahl 32767; jeder abgespeicherte Wert verbraucht 2 Bytes Speicherplatz, unabhängig von seiner Größe.

9. Binäre Datentypen sollten nicht in Files verwendet werden, die mit Begrenzer (Delimiter) arbeiten, da die Gefahr besteht, den Begrenzer zu duplizieren.

10. Wenn numerische Werte größer als 32767 binär dargestellt werden sollen, so erhält man kein (richtiges) Ergebnis.
Wenn der String des binären PICTURE-Befehls 99V999 lautet, so wird der Wert 67.000 größer als 32767 bewertet.

11. JUSTIFIED (rechter Randausgleich) darf nur für elementare Datenfelder verwendet werden, die weder numerisch noch editiert sein dürfen.

12. Der REDEFINES-Befehl darf nicht auf Felder mit der Ordnungszahl 01 (LEVEL 01) angewendet werden.

13. Für ganze Datengruppen kann auch das COMP und COMP-3 Format gewählt werden.

```
Beispiel :
0047 01 STUNDE.
0048     02 PAY-TYPE PICTURE IS X.            (Variable für 1
                                               ASCII-Zeichen)
0049     02 VOR-NAME PICTURE IS X(20).        (VOR-NAME kann 20
                                               ASCII-Zeichen
                                               enthalten)
0050     02 NACH-NAME PICTURE IS X(20).
0051     02 BETRAG PIC 9(9) USAGE IS COMP-3. (BETRAG kann eine
                                               9-stellige Zahl
                                               enthalten)
0052     02 ITM1 PICTURE IS X.
0053     02 ITM11 REDEFINES ITM1 PIC 9.
0054     02 EINKOMMEN PIC S9(16)V99.          (EINKOMMEN kann eine
                                               positive oder
                                               negative Dezimal-
                                               zahl enthalten)
0055     02 STEUERN OCCURS 10 TIMES PICTURE IS S9(10)V99.
0056 01 MONAT.
0057     05 FILLER PIC X.
0058     05 GRP-ITM.
0059     10 GRP-ITM2.
0060     15 GRP-AMT PIC 9(6)V99.
0061     15 GRP-AMT-1 PIC 9(6)V99.
0056 01 LEERZEILE PICTURE IS X(132).       (Leerzeile auf
                                               Drucker ist 132
                                               Zeichen lang)

0081 WORKING-STORAGE SECTION.
0082 01 INVENTORY.
0083     02 PART-NUM PICTURE 9(5) USAGE IS COMP-3.
0084     02 QTY-IN-STOCK PIC 9(6) COMP-3.
0085     02 W-INDEX PICTURE 99 VALUE IS 01 COMP.
0086     02 W-ITM2 PIC X(5) VALUE "TEST1".
0087 01 A-TABELLE.
0088     02 T1 PIC X(5) VALUE "ERSTER".
0089     02 T2 PIC X(5) VALUE "ZWEITER".
0090     02 T3 PIC X(5) VALUE "DRITTER".
0091 01 B-TABELLE REDEFINES A-TABELLE.
0092     02 TABELLE OCCURS 3 TIMES PICTURE X(5).
0093 01 EDITIERT.
0094     02 E-1  PICTURE $,$$$,$$$,$$$,$$$.99CR.
0095     02 E-2  PIC  99V999+.
0096     02 E-3  PIC ZZ,ZZZ,ZZZ.99-.
0097     02 E-4  PIC $,$$$,$$$.99DB.
0098* durch Verwendung des ACCEPT-Befehls kann der folgende File-
0099* name beim Programmablauf verändert werden: ACCEPT FILE-NAME
0100 01 FILE-NAME PICTURE X(14) VALUE "A:FILENAME.WRK".
0101* Jetzt Variable für den relativen Schlüssel definieren und
0101* den Wert 1 zuweisen, um den ersten Record zu lesen :
0101 01 KEY3 PIC 9(7) COMP-3 VALUE 1.
0102 01 STA-1.
0103     02 KEY1 PIC X.
0104     02 KEY2 PIC X.
0105* maximale Record-Größe beträgt 4095 Bytes
0106 01 BIG-ITEM PIC X(4095).
```

COPY-Befehl

Funktion : Der COPY-Befehl fügt beim Compilieren Text in das Quellprogramm ein.

Syntax :

    COPY u:Filename .

Regeln :

1. Ein COPY-Befehl darf nicht innerhalb eines anderen auftauchen.

2. Die Angabe des Diskettenlaufwerks u: kann wahlweise erfolgen; wird kein Laufwerk angegeben, so wird das aktuelle Laufwerk angesprochen.

3. Der COPY-Befehl steht normalerweise in der 7. Spalte nach einem Leerzeichen und wird durch einen Punkt (.) abgeschlossen.

4. Der Filetyp wird nicht angegeben, muß aber unbedingt vom Typ CBL sein.

Beispiel :

```
0041 DATA DIVISION.
0042  COPY A:FILE6.
0043  COPY A:FILE7.
0044  COPY A:FILE8.
```

Kapitel IV

Der PROCEDURE DIVISION-Teil

Funktion : In diesem Programmteil stehen die Prozeduren, die zur Problemlösung notwendig sind und die Daten bearbeiten.

Syntax :

```
PROCEDURE DIVISION
          [USING Datenname-1 [, Datenname-2] ...] .

[Sektions-Name SECTION [Segmentnummer] ].
Abschnitt-Name.

     (Hier stehen die Befehle, um ein Problem zu lösen)

Abschnitt-Name.

     (Hier stehen die Befehle, um ein Problem zu lösen)
```

Regeln :

1. Der erste Eintrag in der PROCEDURE DIVISION muß der Name einer Sektion, eines Abschnitts oder die USING-Anweisung sein.

2. Kein Abschnitt oder Sektions-Name darf mehrmals vorkommen.

3. Hinter jedem Abschnitt-Namen steht ein Punkt.

4. Um ein Problem zu lösen, werden Befehle verwendet, die aus reservierten oder vorher definierten Worten, Abschnitts-Namen, symbolischen Konstanten, numerischen/nicht-numerischen Daten und/oder syntaktischen Zeichen bestehen.

Beispiel :

```
... Hier stehen die anderen Programmteile ...

0100 PROCEDURE DIVISION.
0101 BEGIN.
0102      DISPLAY " DIES IST MEIN ERSTES COBOL-PROGRAMM !".
0103      STOP RUN.
0104 END PROGRAM TEST1.
```

Auf den folgenden Seiten werden alle Schlüsselwörter in alphabetischer Reihenfolge erklärt, um so ein leichtes Nachschlagen zu ermöglichen.

ACCEPT-Anweisung

Funktion : Einlesen von Daten über Tastatur.

Syntax :

ACCEPT Datenname

Regeln :

1. Mit der ACCEPT-Anweisung können Daten nur von der Tastatur eingelesen werden.

2. Die Daten werden von links nach rechts eingelesen, bis das empfangende Datenfeld voll ist oder die Eingabetaste (Return) gedrückt wird. Die Eingabetaste dient nur dazu, die Dateneingabe zu beenden; ihr Code wird nicht übertragen.

3. Man kann durch Betätigen der DELETE-Taste jeweils das zuletzt eingegebene Zeichen löschen.

4. Mit der DELETE-Taste kann man das Eingabefeld nicht verlassen.

5. Benutzt man unter CP/M die Funktion 1 & 2, so meldet sich der Computer mit dem Zeichen "<", wenn das Ende des Felds überschritten wurde und daher das letzte Zeichen nicht in den Speicher geschrieben wurde. Dies geschieht, weil CP/M jedes eingebene Zeichen sofort auf Bildschirm ausgibt, obwohl es noch nicht in den Speicher gelesen wurde. Benutzt das RUN-TIME-Modul die Funktion 6 oder direkten BIOS-Zugriff, so werden Zeichen nicht auf dem Bildschirm ausgegeben, wenn das Datenfeld überschritten wurde.

6. Bitte schlagen Sie unter dem Befehl DISPLAY UNIT und dem Konfigurationsprogramm CONFIG nach, um die CRT-Treiber zu initialisieren.

7. Das Return-Zeichen wird nicht ausgegeben, außer wenn man die CP/M-Funktionen 1 & 2 benutzt.

Beispiel :

```
...

0101 PROCEDURE DIVISION.
0102 BEGIN.
0103      ACCEPT KUNDEN-NAME.
0104      ACCEPT DATUM.
0105      DISPLAY " BITTE GEBEN SIE DEN DATEINAMEN AN : ".
0106      DISPLAY "   D:FILENAME.TYP".
0107* Löschen des Bildschirms auf einem SOL-20
0108      DISPLAY ""0B"".
0109* Füllen des Bildschirms 80*24 = 1920 Zeichen erforderlich
0109* BILDSCHIRM-MASKE ist also 1920 Zeichen groß
0110      DISPLAY BILDSCHIRM-MASKE.
0111* Setzen des Cursors mit hexadezimalen String
0112      DISPLAY ""1B,01,3F"".
0113      ACCEPT EINGABE-DATEN.
...
```

ADD-Anweisung

Funktion : Addiert zwei Zahlen und speichert das Ergebnis ab.

Syntax :

```
          < Zahl-1       >         < Zahl-2       >
      ADD < Datenname-1 > [TO] < Datenname-2 >

      [ GIVING Datenname-3 ] [ ROUNDED ]

      [, ON SIZE ERROR  unbedingte Anweisung ]
```

Regeln :

1. Jede ADD-Anweisung muß zwei Summanden enthalten.

2. Symbolische Konstanten können nicht verwendet werden.

3. Man darf nur numerische Variablen oder Zahlen verwenden. Datenname-3 kann eine numerische Variable sein, die durch Editierzeichen wie Komma, Dezimalpunkt oder Dollarzeichen formatiert ausgegeben wird.

4. Die Summe aus beiden Operanden darf nicht länger als 18 Stellen werden.

5. Ein Datennamen steht für eine elementare Variable.

6. Die Operanden können ein Vorzeichen und ein Dezimalpunkt enthalten.

7. Bei Auswertung der Operanden und die Berechnung werden die Dezimalpunkte berücksichtigt.

8. Wählt man die ROUNDED-Option, so prüft der Computer, ob rechts Stellen abgeschnitten wurden. Ist dies der Fall, so wird 1 addiert, wenn die abgetrennte Ziffer 5 oder größer ist.

9. Tritt beim Addieren ein Überlauf auf, so wird die unbedingte Anweisung nach ON SIZE ERROR ... ausgeführt.

Beispiel :

```
0150      ADD PORTO-KOSTEN TO EINZEL-PREIS GIVING GESAMT-PREIS.
0151              ON SIZE ERROR GO TO  FEHLER-ROUTINE.
```

ALTER-Anweisung

Funktion : Verändern einer bestimmten Befehlsfolge.

Syntax :

ALTER Abschnitt-Namen-1 TO PROCEED TO Abschnitt-Namen-2.

Regeln :

1. Abschnitt-Name-1 muß der Name eines Abschnitts sein, der nur aus dem Satz :
GO TO Abschnitt-Name. besteht.

2. Der ALTER-Befehl modifiziert den Satz so, daß bei einem Aufruf von Abschnitt-Namen-1 der Abschnitt-Namen-2 angesprungen wird.

Beispiel :

```
0200 ABSCHNITT-6. GO TO BEGIN.
0201 ABSCHNITT-7.
0202        ALTER ABSCHNITT-6 TO PROCEED TO END-OF-JOB.
0203        GO TO ABSCHNITT-6.
0204 END-OF-JOB.
```

CALL-Anweisung

Funktion : Der CALL-Befehl Übergibt die Kontrolle an ein anderes Objekt-Programm innerhalb des Speichers.

Syntax :

```
            <Textstring-1>
      CALL <Datenname-1 >
                  [USING Datenname-1 [Datenname-2] ...]
```

Regeln :

1. Datenname-1 muß eine alphanumerische Variable sein, so daß der Wert ein Programmname sein kann.

2. Der USING-Befehl wird nur verwendet, wenn auch in der PROCEDURE DIVISION des aufgerufenen Programms ein USING-Befehl vorhanden ist. Die Anzahl der Operanden der beiden Befehle muß gleich sein.

3. Jeder Operand der USING-Anweisung wird in der FILE-SECTION oder in der WORKING-STORAGE SECTION definiert. Die Operanden haben immer die Ordnungszahl 01 oder 77.

4. Datenname-1 oder Textstring-1 enthalten den Namen des Programms, das ausgeführt werden soll. Das Programm, in dem der CALL-Befehl steht, ist das aufrufende Programm.

5. Bei Ausführung des CALL-Befehls wird die Kontrolle an das aufgerufene Programm übergeben.

6. Das Programm, das mittels CALL-Befehl aufgerufen wird, bleibt unverändert, d.h. es behält die Daten, die beim letzten Aufruf bearbeitet wurden.

7. Aufgerufene Programme dürfen auch CALL-Anweisungen verwenden. Allerdings darf das Programm, von dem der Aufruf erfolgte, nicht direkt oder indirekt aufgerufen werden.

8. Soll das aufgerufene Programm auf die Daten des Hauptprogramms zugreifen, so müssen die Datennamen in der USING-Anweisung deklariert werden.

NEVADA-COBOL Einzelheiten :

1. Die aufgerufenen Programme müssen vom Typ .OBJ sein.

2. Wird ein Programm zum ersten Mal aufgerufen, wird es in den Speicher geladen und in einer Tabelle des RUN-Time-Moduls verzeichnet. Zukünftige Aufrufe gehen direkt an das Programm.

3. Bis zu fünf Programme können sich gleichtzeitig im Speicher befinden. Will man ein weiteres Programm laden, so muß vorher ein Programm mittels CANCEL-Befehl gelöscht werden.

4. Durch den CALL-Befehl kann man auch ein anderes Hauptprogramm aufrufen, auch wenn dieses das erste Programm überlagert. Da die WORKING-STORAGE-Sektion immer an der gleichen Adresse steht, behalten die Variablen, die vom zweiten Programm nicht mit dem VALUE-Befehl initialisiert werden, ihren Wert aus dem ersten Programm bei. Deken Sie bitte daran, das erstere Programm mit CANCEL aus der Tabelle des RUNTIME-Pakets abzumelden. Wenn die Tabelle voll ist, führt jeder weitere CALL-Befehl zum Programmabbruch.

5. Aufgerufene Programme müssen nicht unbedingt COBOL-Programme sein. Es muß sich lediglich um OBJekt-Programme handeln (Typ OBJ) die richtig assembliert/compiliert wurden. Das OBJekt-Programm enthält außerdem die Speicheraddresse, wohin es geladen wird, und die Einsprungaddresse :

Bytes	Funktion
2	Anzahl der Bytes in dem Programmsegment
2	Speicheraddresse, wohin das Segment geladen wird
....	hier folgt nun das eigentliche Programm/Daten

Das RUN-Time-Moduls lädt jedes Segment zur voreingestellten Adresse, wenn keine Einsprungadresse gefunden wird.

6. Zum Umwandeln von CP/M HEX-Files in OBJekt-Files kann man das mitgeliefgerte Programm CONVHEX.COM verwenden.

7. Das RUN-Time-Modul übergibt die Kontrolle an das aufgerufene Programm unter Verwendung der 8080-CALL Anweisung. Die Rückkehr zum Hauptprogramm erfolgt über die 8080-RET Anweisung. Für aufgerufene Programme sollte ein eigener Stack definiert werden.

8. Die Parameterübergabe geschieht über die Register :
H & L - Parameter 1
D & E - Parameter 2
B & C - Parameter 3 : falls mehr Parameter benötigt werden, enthält B & C die Addresse einer Tabelle mit Parameteraddressen. Die Parameter sind 16-Bit Werte und zeigen auf die Endaddresse des jeweiligen Datennamen.

9. Testet man Programme, die mit dem CALL-Befehl angesprochen werden, so geschieht das mit dem RUN-Time-Modul. Allerdings kann diese Methode nur angewendet werden, wenn keine Parameter übergeben werden.

Beispiel :

```
0001          CALL "NEXTPROG"  USING REC-1, REC-2.
0005          CALL NEXT-PROG USING REC-1, REC-2.
    * Bitte beachten Sie die Beispielprogramme im Anhang des
    * Handbuchs !
```

CANCEL-Anweisung

Funktion : Löscht ein Programm aus dem Speicher und stellt den entsprechenden Platz für weitere Anwendungen zur Verfügung.

Syntax :

```
                < Zeichenstring-1 >
      CANCEL    < Datenname-1     >
```

Regeln :

1. Die CANCEL-Anweisung löscht ein Programm aus der Tabelle des RUN-Time-Moduls. Der Speicherplatz, der von diesem Programm belegt wurde, wird freigegeben und steht damit wieder zur Verfügung.

2. Die CANCEL-Anweisung darf nicht auf ein Programm angewendet werden, das aufgerufen, aber noch nicht durch den EXIT PROGRAM-Befehl beendet wurde.

3. Wird mit der CANCEL-Anweisung ein Programm aufgerufen, das nicht in der Tabelle des RUN-TIME-Moduls enthalten ist, so erfolgt keine Fehlermeldung. Es wird einfach die nächste Anweisung ausgeführt.

Beispiel :

```
0100      CANCEL "UNTERPROGRAMM".
0200      CANCEL  ROUTINE.
```

CLOSE-Anweisung

Funktion : Beendet die Bearbeitung der Eingabe- und Ausgabefiles.

Syntax :

CLOSE Filename-1 [, Filename-2.] ...

Regeln :

1. Bevor ein File geschlossen wird, muß es mittels OPEN-Befehl geöffnet werden.

2. Wenn nötig, schreibt der CLOSE-Befehl den letzten Block aus dem Puffer noch einmal in das aktuelle File.

Beispiel :

```
0300 PROGRAMM-ENDE.
0301           CLOSE  KUNDEN-DATEI.
0302           CLOSE  KONTO-STAND.
0303           CLOSE  AUSGABEN, EINNAHMEN.
```

COPY-Anweisung

Funktion : Beim Compilieren wird Quelltext in das Programm eingefügt.

Syntax :

COPY u:FILE-NAME.

Regeln :

1. Eine COPY-Anweisung darf nicht innerhalb einer anderen stehen.

2. Fällt die Laufwerksangabe u: weg, so wird das aktuelle Laufwerk angesprochen.

3. Der COPY-Befehl steht normalerweise in Spalte 7 und wird durch einen Punkt abgeschlossen.

4. Der Filetyp wird nicht angegeben, muß jedoch immer .CBL sein.

Beispiel :

```
0100 PROCEDURE DIVISION.
0101 ABSCHNITT-A.
0102  COPY A:TEIL1.
0200 ABSCHNITT-B.
0201  COPY B:TEIL2.
```

DISPLAY-Anweisung

Funktion : Ausgabe von Daten auf Bildschirm.

Syntax :

Format 1

```
                < String      >   < String      >
      DISPLAY   < Datenname-1 >   [< Datenname-2 >] ...
                                  [WITH NO ADVANCING]
```

Format 2

```
                    < String      >
      DISPLAY UNIT  < Datenname-1 >.
```

Regeln :

1. Der DISPLAY-Befehl spricht nur den Bildschirm an.

2. Handelt es sich um einen numerischen String, darf dieser kein Vorzeichen haben, da anstelle des Vorzeichens ein Buchstabe ausgegeben würde.

3. Bevor die Ausgabe erfolgt, wird ein Carriage Return & Line Feed ausgeführt. Wird WITH NO ADVANCING angeführt, so entfällt ein Zeilenvorschub.

4. Sind die Daten länger als eine Zeile, so wird die Ausgabe in der nächsten Zeile weitergeführt. Die jeweilige Zeilenlänge kann man mit dem CONFIG-Programm festlegen.

5. Der String kann auch jede symbolische Konstante sein, außer ALL.

Wenn eine symbolische Konstante als Operand auftritt, so wird diese nur einmal ausgegeben.

7. Die Ausgabe der Daten erfolgt in der angegebenen Reihenfolge.

8. Beim Programmablauf ändert der DISPLAY UNIT-Befehl den I/O-Treiber wie folgt :

"0X" das Programm benutzt direkt die BIOS-Routinen
"2X" das Programm benutzt die CP/M-Funktionen 1 & 2
"6X" das Programm benutzt die CP/M 2.2 Funktion 6

X erlaubt die Eingabe von beliebigen Zeichen; steht anstelle des X ein anderes Zeichen, so können nur Zeichen aus dem ASCII-Satz eingegeben werden. Diese Anderungen sind nur vorübergehend.

9. Um die Bildschirmausgabe dauerhaft zu verändern, lesen Sie bitte die Informationen zum Programm CONFIG durch.

10. Wollen Sie Zeichen aus/eingeben, die nicht zum ASCII-Satz gehören (z.B. Steuerzeichen), werden die Befehle UNIT 0 oder UNIT 6 verwendet. Diese Methode ist erforderlich, da die CP/M-Funktion 1 die Steuerzeichen nicht richtig einliest.

Beispiel :

```
0350 FEHLER-ROUTINE.
0351     DISPLAY FEHLER-MELDUNG (FEHLER-CODE).
0352     DISPLAY VOR-NAME, NACH-NAME, "NAME".
0353D    DISPLAY "DEBUG-MODUS   FEHLER-ROUTINE".
0354     DISPLAY " KEIN ZEILENVORSCHUB " WITH NO ADVANCING.
0360* Die nächste Zeile löscht den Bildschirm auf einem SOL-20
0361* oder auf einem VDM-1
0370     DISPLAY ""0B"".
0371* Der folgende Befehl löscht den Bildschirm auf einem
0372* HAZELTINE-1520
0390     DISPLAY ""7E,1C"".
0400* Wenn die Steuersequenzen bekannt sind, kann auch der Cursor
0401* positioniert werden und Reversdarstellung erfolgen
0402* Der nächste Befehl initialisiert den I/O-Treiber für die
0403* BIOS-Routinen: jedes eingelesene Zeichen wird verarbeitet.
0410     DISPLAY UNIT "0X".
```

DIVIDE-Anweisung

Funktion : Der DIVIDE-Befehl dividiert zwei Zahlen und legt das Ergebnis (Quotient) in einer Variablen ab.

Syntax :

```
          < Datenname-2 > BY    < Datenname-1 >
DIVIDE < Datenname-1 > INTO < Datenname-2 >
  [GIVING Datenname-3]

  [ROUNDED]    [, ON SIZE ERROR  unbedingte Anweisung ]
```

Regeln :

1. Jede DIVIDE-Anweisung muß einen Dividenden und einen Divisor enthalten.

2. Jeder Datenname muß eine elementare, numerische Variable sein. nur Datenname-3 darf eingefügte Zeichen zur formatierten Ausgabe beinhalten.

3. Kein Operand darf mehr als 18 Stellen umfassen.

4. Ein Datenname darf nur auf eine elementare Variable zugreifen.

5. Jeder Operand kann ein Vorzeichen und einen Dezimalpunkt enthalten.

6. Die Bewertung der Operanden geschieht mittels Dezimalpunkt.

7. ROUNDED prüft, ob Dezimalstellen abgeschnitten werden. Ist dies der Fall, so wird 1 addiert, wenn die abgetrennte Ziffer größer oder gleich 5 sein.

8. ON SIZE ERROR testet, ob ein Überlauf auftritt. Bei einem Überlauf wird die unbedingte Anweisung ausgeführt.

Beispiel :

```
0400 BERECHNUNG-1.
0401      DIVIDE STUNDEN INTO BEZAHLUNG GIVING STUNDEN-LOHN.
0402         ROUNDED  ON SIZE ERROR GO TO  UNTERPROG-2.
0403      DIVIDE STUNDEN INTO KILOMETER.
```

END PROGRAM-Anweisung

Funktion : Kennzeichnet das physikalische Ende eines Programms.

Syntax :

END PROGRAM Programmname.

Regeln :

1. Diese Zeile muß in jedem Programm die letzte physikalische Anweisung sein.

2. Dies ist ein Compiler-wirksamer Befehl. Anhand dieses Befehls erkennt der Compiler das Programmende.

Beispiel :

```
0900* Hier steht Ihr COBOL-Programm
9999 END PROGRAM TEST.
```

EXIT-Anweisung

Funktion : Beendet die Ausführung mehrerer Prozeduren.

Syntax :

Format 1

EXIT.

Format 2

EXIT PROGRAM.

Regeln :

1. Die EXIT-Anweisung steht isoliert in einem eigenen Abschnitt.

2. Wird ein Programm mit dem CALL-Befehl aufgerufen und die EXIT PROGRAM Anweisung ausgeführt, so wird die Kontrolle an das aufrufende Programm übergeben. Sollte sich der EXIT PROGRAM-Befehl nicht in einem aufgerufenen Programm befinden, wird er wie ein normaler EXIT-Befehl behandelt.

Beispiel :

```
0500 ABSCHNITT-ENDE.
0501         EXIT.
0602 ENDE-UNTERPROGRAMM.
0603         EXIT PROGRAM.
```

GO TO-Anweisung

Funktion : Verzweigen des Programms zu anderen Prozeduren.

Syntax :

Format 1

```
      GO TO Prozedurenname-1.
```

Format 2

```
      GO TO Prozedurenname-1.  [ Prozedurenname-2 ] ....
            DEPENDING ON Datennamen.
```

Regeln :

1. Der GO TO-Befehl stellt in einer Folge von Anweisungen den letzten Befehl dar.

2. Der Datenname bezieht sich auf eine numerische Variable, die als ganze Zahl definiert wurde.

3. Wird ein Abschnitt mittels ALTER-Befehl geändert, so kann in diesem Abschnitt nur der GO TO-Befehl im Format 1 stehen.

4. Der GO TO-Befehl überträgt die Programmausführung an die angeführte Prozedur. Das Programm wird dort fortgesetzt.

5. Verwendet man den GO TO-Befehl im Format 2, wird die Programmausführung bei Prozedur-1, Prozedur-2...fortgesetzt, abhängig vom Wert der Variablen 1,2,...,n. Ist der Wert der Variablen negativ oder nicht im Bereich von 1,2,...,n, so wird die GO TO-Anweisung ignoriert. Diese Verwendung entspricht in etwa dem ON...GOTO-Befehl in BASIC.

Beispiel :

```
0100           IF  WERT  IS NOT EQUAL TO 1
0101               MOVE X-WERT TO Y-WERT
0102               GO TO A-UNTERPROGRAMM.
0103           GO TO HAUPT-PROGRAMM.
0104 ZUWEISUNG-ABSCHNITT.
0105      GO TO  FUNKTION, FUNKTION-2, FUNKTION-3 DEPENDING ON X.
0200 ALTER-ABSCHNITT.
0201      GO TO  BEGINN.
```

IF-Anweisung

Funktion : Abhängig von einer Bedingung können verschiedene Anweisungen ausgeführt werden. Die resultierende Anweisung ist davon abhängig, ob die Bedingung wahr oder falsch ist.

Syntax :

Format 1

```
                          < Anweisung-1    > < ELSE Anweisung-2    >
      IF < Bedingung > < NEXT SENTECE > < ELSE NEXT SENTENCE >
    Bedingung :
                               < = < >          >
                               < EQUAL TO       >
                               < LESS THAN      >  < String       >
      Datenname-1 IS   [NOT] < GREATER THAN >  < Datenname-2 >
    Bedingung :
                               < NUMERIC        >
      Datenname-3 IS   [NOT] < ALPHANUMERIC >
```

Format 2

```
                     < OR  >
      IF  Bedingung < AND > Bedingung
```

Regeln :

1. Anweisung-1 und Anweisung-2 müssen unbedingte Befehle sein.

2. Bei nicht-numerischen Vergleichen werden die ASCII-Zeichen von links nach rechts verglichen.

3. Bei ganzen Zahlen wird ein Dezimalpunkt nicht angegeben.

4. Datenname-3 muß ein ASCII-Datentyp sein.

5. Ist die Bedingung wahr, so wird die Anweisung-1 ausgeführt. Enthält Anweisung-1 einen Sprungbefehl, so wird dieser ausgeführt. Umfasst Anweisung-1 keinen Sprungbefehl, so wird die ELSE-Anweisung ignoriert und die nächste Zeile bearbeitet.

6. Folgt auf ELSE NEXT SENTENCE sofort der Punkt, so kann dieser Ausdruck weggelasen werden.

7. Ist die Bedingung wahr und steht anstelle der Anweisung-1 der Befehl NEXT SENTENCE, so wird die ELSE-Anweisung ignoriert. Dann wird die nächste Zeile bearbeitet.

8. Ist die Bedingung falsch, so wird Anweisung-1/NEXT SENTENCE ignoriert. Enthält Anweisung-2 einen Sprungbefehl, so wird dieser ausgeführt. Sollte in Anweisung-2 kein Sprungbefehl vorhanden sein, so wird die nächste Zeile bearbeitet. Ist ELSE Anweisung-2 nicht gegeben, so wird Anweisung-1 ignoriert und die nächste Programmzeile bearbeitet.

9. Ist die Bedingung falsch und sollte ELSE NEXT SENTENCE gegeben sein, so wird Anweisung-1 ignoriert. Daraufhin wird die nächste Zeile bearbeitet.

10. Zwei Bedingungen können mit den logischen Operatoren AND und OR verknüpft werden.

Beispiel :

```
0100      IF NACH-NAME IS NOT ALPHABETIC
0101         MOVE ERROR-CODE TO FEHLER-MELDUNG
0102         ADD 1.TO FEHLER-ZÄHLER
0103         GO TO  FEHLER-AUSGABE
0104       ELSE
0105         PERFORM A-ABSCHNITT THRU B-ABSCHNITT.
0106      IF ÜBERWEISUNG < RECHNUNGSBETRAG OR
0107         ÜBERWEISUNG > RECHNUNGSBETRAG
0108          GO TO  FEHLBETRAG.
0109      IF A = B
0110         OR = C
0111         OR = D
0112         OR X NOT >
0113           MOVE S TO W
0114         ELSE
0115           MOVE S TO AW.
```

INSPECT-Anweisung

Funktion : Die INSPECT-Anweisung ermöglicht das Suchen und/oder Ersetzen von einzelnen Zeichen in Textstrings.

Syntax :

Format 1

```
INSPECT  Datenname-1  TALLYING
                      < ALL       < String-1     >>
                      < LEADING < Datenname-3 >>

    Datenname-3  FOR  << CHARACTERS              >
                       < AFTER >
                                      < String-2     >
              [ < BEFORE > INITIAL < Datenname-4 >] >...>...
```

Format 2

```
INSPECT  Datenname-1  REPLACING

                    < String-4     >
  < CHARACTERS BY < Datenname-6 >
                     < AFTER >           < String-5     >
                    [< BEFORE > INITIAL < Datenname-7 >
 < ALL       >
 < FIRST     >      < String-3     >    < String-4     >
 < LEADING >  << Datenname-5 > BY < Datenname-6 >
                   < AFTER >           < String-5     >
                  [< BEFORE > INITIAL < Datenname-7 >
```

Format 3

```
INSPECT  Datenname-1  TALLYING
                       < ALL       < String-1     >>
                       < LEADING < Datenname-3 >>
  < Datenname-2  FOR  << CHARACTERS              >
           < AFTER >              < String-2     >
          [< BEFORE > INITIAL < Datenname-4 >] >...>...

REPLACING
                    < String-4     >
  < CHARACTERS BY < Datenname-6 >
                  < AFTER  >            < String-5     >
                 [< BEFORE > INITIAL < Datenname-7 >]

  < ALL       >
  < FIRST     >   < String-3     >      < String-4     >
  < LEADING >  << Datenname-5 > BY < Datenname-6 >
                     < AFTER  >                < String-5     >
                    [< BEFORE > INITIAL < Datenname-7 >]
```

Regeln :

1. Datenname-1 bezieht sich auf eine Datengruppe oder eine elementare Variable. Der Datentyp muß dabei als DISPLAY (ASCII) definiert sein (siehe USAGE IS-Befehl in der WORKING -STORAGE Sektion).

2. Datenname-3...Datenname-n sind elementare alphabetische, alphanumerische oder numerische Variablen, die ebenfalls vom Typ DISPLAY sein müssen.

3. Jeder String ist nicht-numerisch und kann auch jede symbolische Konstante sein, außer ALL.

4. String-1,-2,-3,-4,-5 und die Variablen Datenname-3,-4,-5,-6,-7 sind jeweils ein Zeichen lang

Für Format 1 und 3 gilt :

5. Datenname-2 ist eine elementare numerische Variable.

6. Ist String-1 oder String-2 eine symbolische Konstante, so ist sie ein Zeichen lang.

Für Format 2 und 3 gilt :

7. Die Daten von String-4 und Datenname-6 müssen genauso lang sein wie die Daten von String-3 und Datenname-5. Wählt man nun für String-4 eine symbolische Konstante, so sind String-3 oder Datenname-5 genauso lang wie die Konstante.

8. Wird das Schlüsselwort CHARACTERS verwendet, sind String-4 und String-5 oder Datenname-6 und Datenname-7 jeweils ein Zeichen lang.

9. Ist String-3 eine symbolische Konstante, so müssen String-4 oder Datenname-6 ein Zeichen lang sein.

Allgemeingültige Regeln

1. Will man eine Variable auf einzelne Zeichen prüfen, so gibt man ihren Namen als Datennamen-1 an. Die Überprüfung findet von links nach rechts statt.

2. Die Inhalte der Variablen Datenname-1,-3,-4,-5,-6, oder -7 werden wie folgt behandelt :

a. Ist eine Variable (Datennamen-1,-3...-7) vom Typ alphanumerisch, so wird sie wie ein Zeichenstring behandelt.

b. Ist eine Variable (Datennamen-1,-3,...-7) editiert alphanumerisch, editiert numerisch oder nur numerisch (ohne Vorzeichen), so findet die Überprüfung statt, als wenn die Variable als alphanumerisch definiert wurde.

c. Ist eine Variable (Datennamen-1,-3,...-7) numerisch und besitzt ein Vorzeichen, wird die Variable so überprüft, als wenn sie kein Vorzeichen hat (Regel b. wird angewendet).

3. In den folgenden Regeln 4-11 beziehen sich die Angaben für die Strings (String-1..-5) ebenfalls auf den Inhalt der Variablen (Datenname-3..-7).

4. Bei der Überprüfung der Variablen (Datenname-1) wird jedes Vorkommen von String-1 gespeichert (Formate 1 & 3) und/oder jeder String-3, der enthalten ist, wird durch String-4 ersetzt (Formate 2 & 3).

5. Die Vergleichsfunktion, die feststellt, wie häufig String-1 vorkommt, und das Ersetzen von String-3 verläuft wie folgt :

a. Die Operanden der TALLYING und REPLACING-Anweisung werden von links nach rechts verarbeitet, in der Reihenfolge, in der sie eingegeben wurden. String-1 und String-3 werden mit der gleichen Anzahl von fortlaufenden Zeichen verglichen, die mit der linken Position der Variablen Datenname-1 beginnen. Die beiden Strings (String-1,-3) und der Inhalt von Datenname-1 stimmen nur dann überein, wenn sie Zeichen für Zeichen den gleichen Inhalt haben.

b. Wenn keine Übereinstimmung von den Strings (String-1,-3) mit der Variablen festgestellt wird, so wird der Vergleich mit den nachfolgenden Strings (String-1,-3) wiederholt, bis eine Übereinstimmung gefunden wird oder bis kein nachfolgender String mehr vorhanden ist. Ist kein nachfolgender String (-1,-3) vorhanden, so wird der Vergleich mit dem nächsten Segment der Variablen fortgesetzt (wird keine Übereinstimmung festgestellt, wird das Vergleichssegment aus Datennamen-1 jeweils um ein Zeichen nach rechts verschoben).

c. Sind String-1,-3 mit dem Vergleichssegment identisch, so wird dies verzeichnet und/oder ersetzt, wie in den allgemeinen Regeln 8-10 beschrieben. Danach wird der Vergleich wieder mit String-1,-3 fortgesetzt. Die Position des neuen Vergleichssegment beginnt hinter der Position des gefundenen Segments.

d. Der Vergleich dauert so lange an, bis das letzte Zeichen der Variablen (Datenname-1) im Vergleichssegment enthalten ist.

e. Wird die CHARACTERS-Anweisung verwendet, wird in dem Prozess (5a-5d) ein Operand verwendet, der beim Vergleich jeweils das linke Zeichen des Vergleichssegments aus der Variablen (Datenname-1) anpasst.

6. Die Vergleichsoperation aus Regel 5 wird von den BEFORE und AFTER-Anweisungen wie folgt beeinflußt :

a. Wird BEFORE und AFTER nicht angegeben, so findet der Vergleich der Strings (String-1,-3) und des Operanden aus CHARACTER-Anweisung wie in Regel 5 beschrieben statt.

b. Wird die BEFORE-Anweisung verwendet, werden die Strings und der Operand der CHARACTER-Anweisung nur so lange angewendet, bis String-2, String-5 in der Variablen (Datenname-1) gefunden werden. Die Stelle, an der diese Strings zum ersten Mal vorkommen, wird bestimmt, bevor der Vergleichsvorgang (siehe auch Regel 5), beginnt. Sind die Strings-1,-3 oder der Operand der CHARACTER-Anweisung nicht anwendbar, so werden diese so behandelt als wenn sie nicht mit dem Vergleichssegment der Variablen übereinstimmen. Sind in der Variablen die Strings (String-2,-5) nicht enthalten, so verläuft der Vergleich als wenn keine BEFORE-Befehl gegeben wurde.

c. Wird die AFTER-Anweisung verwendet, so findet ein Vergleich nur dann statt, nachdem die Strings-2,-5 in der Variablen gefunden wurden. Der Vergleich beginnt dann mit String-1,-3 und dem Operanden der CHARACTER-Anweisung sofort hinter der Position von String-2,-5. Die Position der Strings-2,-5 wird festgestellt, bevor der Vergleichsvorgang (Regel 5) begonnen wird. Sind die Strings-1,-3 oder der Operand der CHARACTERS-Anweisung nicht anwendbar, so werden diese so behandelt, als wenn sie nicht mit dem Vergleichssegment der Variablen übereinstimmen. Sind die Strings-2,-5 in der Variablen nicht enthalten, so sind die Strings-1,-3 und der Operand der CHARACTERS-Anweisung ebenfalls nicht anwendbar.

FORMAT 1

7.Der Inhalt von Datenname-2 wird nicht durch die Ausführung des INSPECT-Befehls initialisiert.

8. Die Regeln für das Verzeichnen (to tally) lauten wie folgt :

a. Wird die ALL-Anweisung verwendet, so wird für jedes Vorkommen von String-1 in der Variablen (Datenname-1) der Inhalt von Datenname-2 um eins (1) erhöht.

b. Verwendet man die LEADING-Anweisung, so wird der Inhalt von Datennamen-2 um eins (1) erhöht für jedes fortlaufende Vorkommen von String-1 innerhalb der Variablen (Datenname-1), wobei das erste Vorkommen sofort im ersten Vergleichsvorgang passiert.

c. Ist die CHARACTERS-Anweisung angegeben, so wird der Inhalt von Datenname-2 um eins (1) erhöht, wenn eine Übereinstimmung festgestellt wurde (siehe Regel 5e).

FORMAT 2

9. Die Worte ALL, LEADING und FIRST sind Adjektive.

10. Die Regeln für das Ersetzen lauten wie folgt :

a. Wird die CHARACTERS-Anweisung gegeben, erfolgt ein Ersetzen aller passenden Zeichen durch String-4.

b. Wenn das Adjektiv ALL verwendet wird, erfolgt ein Ersetzen des String-3 durch String-4 in der Variablen.

c. Ist die Anweisung LEADING gegeben, wird jedes fortlaufende Vorkommen von String-3 in der Variablen durch String-4 ersetzt.

d. Wird der Befehl FIRST gegeben, so wird das erste Vorkommen von String-3 in der Variablen durch String-4 ersetzt.

FORMAT 3

11. Wird ein INSPECT-Befehl im Format 3 eingegeben, wird er wie zwei einzelne INSPECT-Anweisungen ausgeführt. Dieses Format kombiniert die TALLYING-Anweisung des ersten Formats und die REPLACING-Anweisung des zweiten Formats miteinander. Die allgemeinen Regeln für das Anpassen und Verzeichnen sind für das erste Format anwendbar, die allgemeinen Regeln für das Anpassen und Ersetzen sind auf das zweite Format anwendbar.

Beispiele :

```
INSPECT  Wort  TALLYING  Zähler  FOR LEADING "L" BEFORE
     INITIAL  "A",  Zähler-1 FOR LEADING "A" BEFORE INITIAL "L".

Wobei  Wort = LARGE, Zähler = 1, Zähler-1 = 0.
       Wort = ANALYST, Zähler = 0, Zähler-1 = 1.

INSPECT  Wort  TALLYING  Zähler  FOR ALL "L", REPLACING LEADING
      "A" BY "E" AFTER INITIAL "L"..

Wobei  Wort = CALLAR, Zähler = 2, Wort = CALLAR.
       Wort = SALAMI, Zähler = 1, Wort = SALEMI.
       Wort = LATTER, Zähler = 1, Wort = LETTER.

INSPECT  Wort  REPLACING ALL "A" BY "G" BEFORE INITIAL "X".

Wobei  Wort = ARXAX, Wort = GRXAX.
       Wort = HANDAX, Wort = HGNDGX.

INSPECT  Wort  TALLYING  Zähler  FOR CHARACTERS AFTER INITIAL
      "J" REPLACING ALL "A" BY "B".

Wobei  Wort = ADJECTIVE, Zähler = 6, Wort = BDJECTIVE.
       Wort = JACK, Zähler = 3, Wort = JBCK.
       Wort = JUJMAB, Zähler = 5, Wort = JUJMBB.


INSPECT  Wort  REPLACING ALL "X" BY "Y"  "B" BY "Z"  "W" BY "Q"
               AFTER INITIAL "R".

Wobei  Wort = RXXBQWY, Wort = RYYZQQY.
       Wort = YZACDWBR, Wort = RAQRYEZ.

INSPECT  Wort  REPLACING CHARACTERS BY "B" BEFORE INITIAL "A".

Wort vorher : 12 XZABCD
Wort nachher: BBBBBABCD
```

MOVE-Anweisung

Funktion : Transferiert Daten von einem Speicherbereich in einen anderen.

Syntax :

```
          <String-1    >
MOVE      <Datenname-1>  TO  Datenname-2    [Datenname-3] ...
```

Regeln :

1. Datennname-1 und String-1 sind Sende-Felder. Dort befinden sich die Daten, die transferiert werden. Datenname-2,-3,.... sind Empfangs-Felder.

2. Die Daten aus dem Sende-Feld werden zuerst in das Datenfeld transferiert, das durch Datenname-2 angegeben wird. Danach geschieht das gleiche mit Datenname-3, usw...
Um auf ein einzelnes Element einer Datengruppe zugreifen zu können, gibt man bei Datennamen-2,-3... jeweils die Ordnungszahl mit an.

3. Handelt es sich bei Datennamen-1 und Datennamen-2 jeweils um elementare Datenfelder, so ist es ein elementarer MOVE-Befehl. Ein elementares Datenfeld kann von folgendem Typ sein : numerisch, alphabetisch, alphanumerisch, formatiert numerisch, oder formatiert alphanumerisch. Den Typ eines Datenfeldes definiert man im PICTURE-Feld.
Numerische Strings gehören zum Typ "numerisch", nicht-numerische Strings sind vom Typ "alphanumerisch". Die symbolische Konstante ZERO ist numerisch, die symbolische Konstante SPACE ist alphabetisch. Alle anderen symbolischen Konstanten sind vom Typ "alphanumerisch".
Die folgenden Regeln gelten für den elementaren MOVE-Befehl :

a. Die Konstante SPACE, numerisch formatierte, alphanumerisch formatierte oder alphabetische Daten können nicht in numerische oder formatierte numerische Felder "gesendet" werden.

b. Numerische Strings, die Konstante ZERO, numerisohe oder formatierte numerische Daten können nicht in ein alphabetisches Feld transferiert werden.

c. Eine numerische Dezimalzahl oder ein Datenfeld mit einer Dezimalzahl kann nicht in ein alphanumerisches oder formatiertes alphanumerisches Feld gesendet werden.

d. Alle sonstigen Kombinationen sind gültig. Die Durchführung erfolgt laut Regel 4.

4. Bei elementaren MOVE-Anweisungen erfolgt die notwendige Bearbeitung der Daten, z.B. formatieren des Empfangsfeldes :

a. Wenn ein formatiertes alphanumerisches oder alphanumerisches Feld die Daten empfängt, werden die Daten übertragen und gleichzeitig eventuelle Leerzeichen eingefügt. Sollte das Sendefeld länger als das Empfangsfeld sein, so gehen auf der rechten Seite Zeichen verloren, wenn das Empfangsfeld voll ist. Ist das Sendefeld numerisch und hat ein Vorzeichen, so wird das Vorzeichen beim MOVE-Befehl nicht übertragen. In diesem Fall wird das Empfangsfeld um ein Zeichen kürzer.

b. Ist das Empfangsfeld numerisch oder numerisch formatiert, so findet die Übertragung der Werte anhand der Dezimalpunkte statt. Wo es notwendig ist, werden Nullen eingesetzt.

1. Definiert man mit dem PICTURE-Befehl ein numerisches Empfangsfeld mit Vorzeichen, so wird bei der Übertragung der Daten dort das Vorzeichen des Sendefeldes eingesetzt. Hat das Sendefeld kein Vorzeichen, so wird im Empfangsfeld ein positives Vorzeichen gesetzt.
2. Ist für das Empfangsfeld kein Vorzeichen definiert, so wird bei Ausführung des MOVE-Befehls nur der Absolutwert gesendet.
3. Ist das Sendefeld alphanumerisch, findet die Übertragung statt, als ob das Sendefeld eine numerische ganze Zahl (ohne Vorzeichen) ist.

5. Jedes MOVE, das kein elementares MOVE ist, wird genauso wie ein elementares alphanumerisches MOVE bearbeitet, mit der Ausnahme, daß die interne Darstellung der Daten sich nicht ändert. Bei dieser Art des MOVE-Befehls wird das Empfangsfeld mit den Daten aus dem Sendefeld gefüllt. Dabei werden die einzelnen Datenfelder und die Datengruppen in Sende- und Empfangsfeld nicht berücksichtigt.

6. Ist String-1 SPACE, QUOTE oder ZERO, dann wird das gesamte Empfangsfeld (Datenname-2) mit dem Wert der symbolischen Konstanten gefüllt.

7. Führt man ein nicht-numerisches MOVE aus, so findet die Übertragung von links nach rechts statt.

Beispiel :

```
0400 ZUWEISUNGS-ROUTINE.
0410      MOVE  SPACES  TO  LEERZEILE.
0420      MOVE  NAME  TO  RECHNUNGS-NAME.
0430      MOVE  ELEMENT (INDEX)  TO  RECHNUNGS-BETRAG.
0440      MOVE  ZEROS  TO  KUNDEN-RABATT, HÄNDLER-RABATT.
0450      MOVE  SPACES  TO  VOR-NAME, NAME.
```

MULTIPLY-Anweisung

Funktion : Multiplikation zweier Werte und abspeichern des Resultats.

Syntax :

```
                < String-1     >       < String-2     >
      MULTIPLY  < Datenname-1 >  BY    < Datenname-2 >

      [ GIVING  Datenname-3  ]         [ ROUNDED ]

      [, ON SIZE ERROR  unbedingter Befehl ]
```

Regeln :

1. Jeder Datenname muß für ein elementares numerisches Datenfeld stehen. Datenname-3 kann auch ein formatiertes numerisches Feld sein.

2. Die Strings müssen numerisch sein.

3. Das Resultat darf nicht länger als 18 Stellen sein.

4. Ein Datenname kann sich nur auf ein elementares Datenfeld beziehen.

5. Jeder Operand kann ein Vorzeichen und einen Dezimalpunkt enthalten.

6. Die Werte werden anhand der Dezimalpunkte erkannt und berechnet.

7. Wählt man die ROUNDED-Option, wird ein Test durchgeführt, der überprüft, ob auf der rechten Seite Stellen abgeschnitten werden. Ist dies der Fall, so wird 1 addiert, wenn die abgetrennte Ziffer größer oder gleich 5 ist.

8. Verwendet man die Option ON SIZE ERROR, so wird bei einem Übertrag (Overflow) der folgende Befehl ausgeführt.

Beispiel :

```
0..0 KALKULATIONS-ROUTINE.
0..0     MULTIPLY  WOCHEN-LOHN  BY  WOCHEN  GIVING  GEHALT.
0..0     MULTIPLY  EINZEL-PREIS  BY  BESTELL-ZAHL
0..0        GIVING  RECHNUNGS-BETRAG  ON SIZE ERROR GO TO
0..0             FEHLER-ROUTINE.
0..0     MULTIPLY  ÜBERSTUNDEN  BY  ÜBERSTUNDEN-PAUSCHALE
0..0        GIVING  GEHALT-2  ROUNDED.
```

OPEN-Anweisung

Funktion : OPEN bereitet Files für Ausgabe oder Eingabe vor.

Syntax :

```
            < I-O    >
            < INPUT  >
      OPEN  < OUTPUT >  Filename-1 [, Filename-2 ] ...
```

Regeln :

1. Bevor man ein File einlesen, schreiben oder schließen kann, muß es mit OPEN geöffnet werden.

2. Der OPEN-Befehl verursacht keinerlei Datentransfer vom oder zum File.

3. Existiert ein Ausabe-File bei der sequentiellen Bearbeitung (SEQUENTIAL ACCESS) noch nicht, so wird es erzeugt.

4. Bei wahlfreiem Zugriff (RANDOM ACCESS) muß das File schon vorhanden sein.

5. Die I-O-Option (INPUT-OUTPUT) ist nur für Files auf Diskette gültig.

Beispiel :

```
0700 BEGIN.
0710      OPEN OUTPUT MAHNUNGS-DATEI.
0720      OPEN INPUT KONTO-STAND.
0730      OPEN I-O  BESTELLUNGEN, AUSLIEFERUNGEN, BESTAND.
```

PERFORM-Anweisung

Funktion : Aufruf eines anderen Programmteils. Nach Ausführung springt das Programm wieder hinter den PERFORM-Befehl. Prozedurenname-1 und Prozedurenname-2 stellen dabei Anfangs- und Endadresse dar.

Syntax :

Format 1

```
                                        <THROUGH>
      PERFORM  Prozedurenname-1   [ <THRU>   Prozedurenname-2 ]
```

Format 2

```
      PERFORM  Prozedurenname-1   [ <THRU> Prozedurenname-2 ]
                                    < ganze Zahl-1 >
                                    < Datenname-1    >  TIMES
```

Format 3

```
      PERFORM  Prozedurenname-1   [ <THRU> Prozedurenname-2 ]
                                                < OR >
                        UNTIL  Bedingung-1 < AND > Bedingung-2
```

Regeln :

1. Jeder Datenname steht für ein elementares numerisches Datenfeld. Dieses Datenfeld muß im DATA DIVISION-Teil als ganze Zahl definiert sein.

2. Die Worte THRU und THROUGH sind gleichbedeutend.

3. Trifft das Programm auf eine PERFORM-Anweisung, so wird die die Prozedur mit dem Namen Prozedurenname-1 abgearbeitet. Danach springt das Programm hinter den PERFORM-Befehl und setzt dort die Bearbeitung fort. Die geltenden Regeln sind :

a. Ist Prozedurenname-1 ein Abschnitt-Name und ist Prozedurenname-2 nicht angegeben, so wird das Programm hinter dem PERFORM-Befehl fortgesetzt, sobald die letzte Anweisung aus Prozedurenname-1 ausgeführt ist.

b. Ist Prozedurenname-1 ein Sektions-Name und ist Prozedurenname-2 nicht angegeben, so wird das Programm hinter dem PERFORM-Befehl fortgesetzt, sobald der letzte Befehl aus dem letzten Abschnitt von Prozedurenname-1 bearbeitet ist.

c. Wird Prozedurenname-2 angegeben und handelt es sich dabei um einen Abschnitt-Name, dann springt das Programm nach der letzten Anweisung des Abschnitt zurück (hinter den PERFORM -Befehl).

d. Gibt man Prozedurenname-2 an und handelt es sich dabei um einen Sektions-Namen, so springt das Programm nach Ausführung des letzten Abschnitt der Sektion zurück.

4. Alle Anweisungen zwischen Prozedurenname-1 und Prozedurenname -2 werden der Reihe nach ausgeführt. Nach Bearbeitung der Prozedur mit dem Namen Prozedurennamen-2 springt das Programm zur PERFORM-Anweisung zurück. Zwischen Anfangs- und Endprozedur können auch Sprungbefehle wie GO TO und PERFORM auftauchen. Ist dies der Fall, sollten alle Programmverzweigungen wieder zur Endprozedur führen (Prozedurenname-2), die nur aus der EXIT-Anweisung besteht.

5. Spricht man diese Prozeduren durch anderen Sprungbefehle außer PERFORM an, so wird jeweils der letzte Befehl ausgeführt und dann die nächste Anweisung bearbeitet.

6. Für alle verschiedenen Formate des PERFORM-Befehls gilt Regel 5 und :

a. Format 1 ist die herkömmliche PERFORM-Anweisung. Sind alle Prozeduren ausgeführt, springt das Programm zum nächsten Befehl hinter der PERFORM-Anweisung zurück.

b. Format 2 ist die PERFORM....TIMES-Anweisung. Damit ist es möglich, Prozeduren beliebig oft auszuführen. Die Anzahl der Durchläufe wird durch die ganze Zahl-1 oder Datennamen-1 festgelegt. Wird bei Ausführung eines PERFORM-Befehls die Variable mit dem Namen Datennamen-1 negativ oder gleich null, so springt das Programm hinter die PERFORM-Anweisung zurück. Ist die mehrmalige Ausführung der Prozedur(en) beendet, arbeitet das Programm ebenfalls den nächsten Befehl nach PERFORM ab.

Wird bei Ausführung eines PERFORM-Befehls der Inhalt der Variablen verändert, so beeinflusst dies jedoch nicht die Anzahl der Durchläufe. Die Anzahl ist nur vom anfänglichen Wert der Variablen abhängig.

c. Format 3 ist die PERFORM....UNTIL-Anweisung. Die Prozeduren werden so lange ausgeführt, bis die Bedingung(en) erfüllt ist. Ist dies gegeben, so springt das Programm hinter die PERFORM-Anweisung zurück. Trifft das Programm auf eine PERFORM....UNTIL Anweisung und ist die Bedingung schon erfüllt, so werden die Prozeduren nicht ausgeführt. Die Bearbeitung des nächsten Befehls findet statt.

7. Ruft eine PERFORM-Anweisung eine Prozedur auf, die wiederum eine PERFORM-Anweisung enthält, so müssen sich die Prozeduren der zweiten PERFORM-Anweisung entweder innerhalb oder außerhalb der Prozedurenfolge befinden, die durch den ersten PERFORM-Befehl aufgerufen wird.
Liegt das Ende eines PERFORM-Befehls innerhalb eines Bereichs, den auch ein zweiter PERFORM-Befehl anspricht, so darf es nicht passieren, daß der erste PERFORM-Befehl auf das Ende des zweiten PERFORM-Befehls trifft. Außerdem dürfen zwei oder mehr PERFORM-Befehle nicht den gleichen Endpunkt haben.

8. Ein PERFORM-Befehl, der sich in einer Sektion befindet, die sich in einem abhängigen Segment befindet, kann nur folgende Programmteile ansprechen :

a. Sektionen und/oder Abschnitten, die ganz in einem oder mehreren abhängigen Segmenten enthalten sind.

b. Sektionen und/oder Abschnitten, die ganz in einem einzigen unabhängigen Segment enthalten sind.

9. Ein PERFORM-Befehl in einem unabhängigen Segment kann folgende Programmteile ansprechen :

a. Sektionen und/oder Abschnitten, die ganz in einem oder mehreren abhängigen Segmenten enthalten sind.

b. Sektionen und/oder Abschnitten, die sich ganz in dem selben unabhängigen Segment wie die PERFORM-Anweisung befinden.

Beispiel :

```
0800        PERFORM  BERECHNUNG  THRU  ABSCHNITT-ENDE.
0801        PERFORM  HAUPT-PROGRAMM.
0802        PERFORM  PRÜFSUMME 5 TIMES.
0803        PERFORM  TEST-ROUTINE UNTIL  PRÜFSUMME = WERT.
0804        PERFORM  ABSCHNITT-1  THRU  ABSCHNITT-2
0805           UNTIL  A = B  OR  X = Y  AND  C = F.
```

READ-Anweisung

Funktion : Einlesen des nächsten logischen Records aus einem geöffneten File.

Syntax :

```
                                  < AT END       >
      READ  Filename  RECORD  < INVALID KEY >  unbedingter Befehl
```

Regeln :

1. Jedes File muß per OPEN-Befehl geöffnet werden, bevor es ausgelesen wird.

2. Die Anweisung AT END muß für sequentielle Files verwendet werden. Sie wird ausgeführt, wenn das Ende eines sequentiellen Files erreicht ist.

3. INVALID KEY wird für Dateien mit wahlfreiem Zugriff (RANDOM ACCESS) verwendet. Wird diese Anweisung ausgeführt, so ist der Inhalt des Puffers undefiniert.

4. Die Nummer des angeforderten Records aus einem RANDOM-File wird durch den relativen Schlüssel bestimmt (RELATIVE KEY). Dies geschieht, bevor der READ-Befehl ausgeführt wird.

5. Bearbeitet man verschieden lange Records, so sollte der Datenpuffer vorher mit Leerzeichen gefüllt werden.
Wird ein Record eingelesen, der kürzer als der vorherige ist, so sind die Daten verfälscht, da der alte Record nicht ganz überschrieben wurde.

6. Werden Files von verschiedener Länge gelesen, so werden die Tabulatorzeichen (09H) nicht expandiert. Dies geschieht, um die gepackte Dezimaldarstellung (COMP-3) nicht zu verfälschen. Sollen Tabulatorzeichen auseinandergezogen werden, so kann man sich des PIP-Befehls unter CP/M 2.2 mit der "T"-Option bedienen.

Beispiel :

```
0800 LESE-ROUTINE.
0801      MOVE  SPACES  TO  DATEN-RECORD.
0802      READ  DATEN-FILE
0803          AT END GO TO  SPEICHER-ROUTINE.
0900 WAHLFREI-LESEN.
     * wenn Record 100 eines RANDOM-Files gelesen werden soll :
0901      MOVE  100  TO  RECORD-SCHLÜSSEL.
0902      READ  RANDOM-KARTEI
0903           INVALID KEY  DISPLAY "Ungültiger Record-Schlüssel".
```

REWRITE-Anweisung

Funktion : Ersetzen eines Records aus einem File.

Syntax :

REWRITE Record-Name [INVALID KEY unbedingte Anweisung]

Regeln :

1. Das File muß vorher im I-O-Modus geöffnet werden.

2. Der Record-Name muß der Name eines logischen Records sein, der in der FILE-SECTION des DATA DIVISION-Teils definiert ist.

3. Soll ein Record geändert werden, so muß vor dem REWRITE-Befehl ein erfolgreiches Einlesen unter sequentiellem Zugriff erfolgen.

4. Die Anweisung INVALID KEY muß für RANDOM-Dateien verwendet werden.

5. Wählt man den wahlfreien Zugriff (RANDOM ACCESS), so wird der Record des Files durch den relativen Schlüssel bestimmt, dessen Name vorher definiert wird.

Beispiel :

```
0800 SEQUENTIELLES-REWRITE.
0801      READ  EINGABE-FILE  RECORD AT END GO TO  ENDE.
0802      MOVE  NEUE-DATEN  TO  EINGABE-RECORD.
0803      REWRITE  EINGABE-RECORD.
0900 RANDOM-REWRITE.
0901      MOVE 100 TO  RELATIVER-SCHLÜSSEL.
0902      REWRITE  NEUER-RECORD  INVALID KEY GO TO FEHLER-ROUTINE.
```

STOP-Anweisung

Funktion : Unterbricht den Ablauf des Objekt-Programms zeitweise oder endgültig.

Syntax :

```
          < String >
STOP      < RUN    >
```

Regeln :

1. Alle Files sollten vor Ausführung einer STOP RUN-Anweisung geschlossen werden.

2. Der STOP RUN-Befehl ist der letzte Befehl des Programms. Nach Ausführung erfolgt Ausstieg in das Betriebssystem.

3. Der String wird auf Bildschirm ausgegeben. Danach muß der Anwender ein Zeichen eingeben und die Eingabe mit Return abschließen :

```
Zeichen :  C  <CR>  =  Programm wird weiter bearbeitet
           E  <CR>  =  Ausstieg zum Betriebssystem
```

Beispiel :

```
5000 PROGRAMM-ENDE. STOP RUN.
2500 FEHLER-ROUTINE. STOP "Fehler !  geben Sie C für weiter ein".
```

SUBTRACT-Anweisung

Funktion : Subtrahieren zweier Zahlen und Abspeichern des Resultats in einer Variablen.

Syntax :

```
                < String-1     >          < String-2     >
SUBTRACT        < Datenname-1 >  FROM     < Datenname-2 >
  [ GIVING      Datenname-3 ]    [ ROUNDED ]

    [, ON SIZE ERROR  unbedingte Anweisung ]
```

Regeln :

1. Jeder Datenname bezieht sich auf eine elementare numerische Variable. Datenname-3 darf zusätzliche auch Editierzeichen enthalten.

2. Das Ergebnis darf nicht länger als 18 Stellen sein.

3. Jeder Datenname kann sich nur auf eine elementare Variable beziehen.

4. Jeder Operand darf ein Vorzeichen und einen Dezimalpunkt haben.

5. Die Auswertung der Operanden und die Berechnung erfolgt anhand der Dezimalpunkte.

6. Die Option ROUNDED prüft, ob auf der rechten Seite des Resultats Stellen abgeschnitten wurden. Ist dies der Fall und ist die abgeschnittene Stelle größer als 5, so wird 1 addiert

7. Die ON SIZE ERROR-Option überprüft, ob ein Überlauf auftrifft. Ist ein Überlauf festgestellt, so wird die unbedingte Anweisung ausgeführt.

Beispiel :

```
0800     SUBTRACT  STEUERN  FROM  GESAMT-GEHALT GIVING  GUBTHABEN
0801          ROUNDED  ON SIZE ERROR GO TO  FEHLER-ROUTINE.
```

WRITE-Anweisung

Funktion : Schreiben eines Records in ein File. Erlaubt außerdem vertikales Positionieren, wenn der Drucker angesprochen ist.

Syntax :

```
  WRITE  Record-Name [ FROM Datenname-1 ]

                              <              PAGE  >
                              <              LINE  >
        [ BEFORE ADVANCING    < ganze Zahl   LINES >]
                              < Datenname-2  LINES >

        {< BEFORE > ADVANCING  < Datenname-2 >
         < AFTER  >              < ganze Zahl  >
                               < PAGE         >

  WRITE  Record-Name  [ INVALID KEY  unbedingte Anweisung ]
```

Regeln :

1. Der Record-Name muß ein logischer Record sein, der in der FILE SECTION des DATA DIVISION-Teils definiert ist.

2. Das reservierte Wort PAGE verursacht ein FORM-FEED (0CH).

3. Mit ganze Zahl LINES gibt man an, wie oft ein Carriage Return Line-Feed ausgeführt werden soll.

4. Der Satz INVALID KEY muß für RANDOM-Files benutzt werden.

5. Bearbeitet man ein RANDOM-File, so muß die erwünschte Record-Nummer durch den relativen Schlüssel angegeben werden.

Beispiel :

```
0900 DRUCKER-ROUTINE.
0901     WRITE  DRUCK-ZEILE  BEFORE ADVANCING  2 LINES.
0902     MOVE  SPACES  TO  DRUCK-ZEILE.
0903     WRITE  DRUCK-ZEILE  BEFORE ADVANCING  PAGE.
1000 RANDOM-SCHREIBEN.
1001     MOVE  1000 TO  RELATIVER-SCHLÜSSEL.
1002     WRITE  RANDOM-RECORD
1003          INVALID KEY  DISPLAY " Ungültiger Schlüssel !".
1100 SEQUENTIELL-SCHREIBEN.
1101     WRITE  SEQ-RECORD.
```

Kapitel V

Einleitung :

COBOL (Common Business Oriented Language) ist eine leicht zu erlernende Programmiersprache, die seit den 60er Jahren für Anwendungen in kaufmännischen und wirtschaftlichen Bereichen benutzt wird. Die Programmiersprache COBOL ist sehr anwenderfreundlich, da die Programme aus einfachen englischen Anweisungen bestehen und eine einfache Syntax verwendet wird. So brauch sich der Programmierer nicht mit systemabhängigen Einzelheiten aufhalten, sondern kann sich auf die eigentliche Problemlösung konzentrieren.

NEVADA-COBOL wurde speziell für kleinere Betriebe entwickelt, die Computer einsetzen, um Probleme im kaufmännischen Bereich zu lösen.

Außerdem bietet COBOL Vorteile, die bei anderen Programmiersprachen nicht anzutreffen sind. So sind COBOL-Programme leicht zu verstehen, da der COBOL-Wortschatz ausschließlich englische Wörter und Verben enthält. Dadurch ist es leicht möglich, Programme für andere Systeme umzuschreiben. Die Sprache COBOL ist standardisiert, das bedeutet, daß Programme auf andere Rechner übertragbar sind, ohne Veränderungen durchzuführen.

COBOL-Zeichensatz

Der ANSI-1974 Standard-Zeichensatz besteht aus den folgenden Zeichen :

0,1,...9	Ziffern
A,B,...Z	Buchstaben
+	Plus-Zeichen
-	Minus-Zeichen
*	Asterik-Zeichen
/	Schrägstrich
=	Gleichheits-Zeichen
$	Dollar-Zeichen
,	Komma
;	Semikolon
.	(Dezimal-) Punkt
"	Anführungs-Zeichen
( )	runde Klammern
< >	größer/kleiner als

Benutzer-definierte Ausdrücke

Ein Benutzer-definierter Ausdruck besteht aus höchstens 30 Zeichen aus : A-Z, 0-9 und - . Der Ausdruck darf nicht mit einem Minus-Zeichen beginnen oder abschließen, muß jedoch mindestens ein alphabetisches Zeichen beinhalten.

Syntax :

Punkt, Komma und Semikolon müssen unmittelbar hinter dem letzten Zeichen des Ausdrucks stehen und von einem Leerzeichen gefolgt werden. Komma und Semikolon sind austauschbar. Auf die linke runde Klammer darf nie ein Leerzeichen folgen; die rechte runde Klammer darf nie auf ein Leerzeichen folgen.

Beispiel :

```
0001            MOVE  WERT  (10)  TO  VARIABLEN-NAME
```

Variablen

Numerische Variablen

1. Eine numerische Variable wird durch bis zu 18 Zeichen dargestellt.

2. Numerische Variablen werden nicht von Anführungszeichen begrenzt.

3. Um negative Werte darzustellen, kann eine numerische Variable als linkes Zeichen ein Minus-Zeichen beinhalten. Andernfalls ist ist der Wert positiv.

4. Innerhalb der Variablen kann ein Dezimalpunkt "." vorkommen; dieser darf jedoch nicht vor dem Minus-Zeichen stehen oder am Ende der Variablen. Enthält eine Variable keinen Dezimalpunkt, so wird sie als ganze Zahl erkannt.

Nicht-numerische Variablen

1. Eine nicht-numerische Variable (Textstring) wird von Anführungszeichen eingegrenzt und kann bis zu 120 Zeichen Zeichen enthalten.

2. Um innerhalb eines Strings ein Anführungszeichen darzustellen, müssen zwei Anführungszeichen direkt hintereinander stehen.

3. Um hexadezimale Werte als Zeichen auszugeben, setzt man doppelte Anführungszeichen.

Beispiel :

```
0000* die nächsten beiden Zeilen geben ABC auf dem Bildschirm aus
0001           DISPLAY "ABC".
0002* hier folgt der hexadezimale String für ABC
0003  GRAPHICS. DISPLAY ""41,42,43"".
0004* die nächste Zeile gibt zwei Anführungszeichen aus
0005           DISPLAY """"
0006           DISPLAY " DIESER SATZ PASST NICHT IN DIESE ZEILE
0007-     "WEIL ER ZU LANG IST".
0008* numerische Variablen :
0009 MATH.     ADD 1 TO RESULT.
0010           ADD 3.75 TO NEWRESULT.
```

Symbolische Konstanten

1. Die Ausdrücke ZERO, ZEROS und ZEROES sind reserviert und stehen für eine (beliebige) Anzahl von Nullen (0).

2. Ebenso verhält es sich mit den Wörtern SPACE und SPACES, die für ein oder mehrere Leerzeichen (Blanks) stehen können.

3. Die Wörter QUOTE und QUOTES stellen Anführungszeichen dar.

4. Die Ausdrücke HIGH-VALUE und HIGH-VALUES stehen für den höchsten Wert, den man mit einem Byte hexadezimal darstellen kann, nämlich $FF ($=hexadezimales Zahlensystem).

5. LOW-VALUE und LOW-VALUES symbolisieren den niedrigsten Wert, der hexadezimal dargestellt werden kann, nämlich $00 .

6. Der Befehl ALL "String" füllt ein Feld mit einem Zeichen.

Beispiel :

```
0001           MOVE ALL "X" TO CUSTOMER-NAME.
0002           IF CUSTOMER-NAME IS EQUAL TO ALL "X"
0003             GO TO PRT-ALIGNMENT.
0004           MOVE HIGH-VALUES TO OUT-RECORD.
```

Datentabellen und Dimensionieren von Variablen

In einer Datentabelle wird jedem Inhalt eine ganze Zahl zugewiesen, anhand der die Daten erkannt und gelesen werden können. Zwar sind die Datentabellen auf 4095 Bytes beschränkt, aber es ist trotzdem möglich, Tabellen zu verarbeiten, die 30 - 40 KBytes Speicherplatz erfordern. Wenn dies der Fall ist, werden mehrere einzelne Tabellen angelegt und miteinander verbunden. Wird aber versucht, einen Tabelleninhalt auszulesen, obwohl dieser gar nicht definiert ist, so erhält man kein gültiges Ergebnis.

Beispiel :

```
0001 WORKING-STORAGE.
0002 01 TABELLE.
0003    02 FILLER PIC X(9) VALUE "JANUAR   ".
0004    02 FILLER PIC X(9) VALUE "FEBRUAR  ".
0005    02 FILLER PIC X(9) VALUE "MÄRZ     ".
0006    02 FILLER PIC X(9) VALUE "APRIL    ".
0007    02 FILLER PIC X(9) VALUE "MAI      ".
0008    02 FILLER PIC X(9) VALUE "JUNI     ".
0009    02 FILLER PIC X(9) VALUE "JULI     ".
0010    02 FILLER PIC X(9) VALUE "AUGUST   ".
0011    02 FILLER PIC X(9) VALUE "SEPTEMBER".
0012    02 FILLER PIC X(9) VALUE "OKTOBER  "
0013    02 FILLER PIC X(9) VALUE "NOVEMBER "
0014    02 FILLER PIC X(9) VALUE "DEZEMBER "
0015 01 M-TBL REDEFINES TABELLE.
0016    02 MONAT OCCURS 12 TIMES PIC IS X(9).
0017 PROCEDURE DIVISION.
0018 DATUM-ABSCHNITT.
0019    MOVE MONAT (MONAT-NUMMER) TO PRT-MONATS-NAME.
0020*
0021*    weitere Beispiele :
0022*
1234          MOVE ITEM TO TABLE (7).
1235          MOVE TABLE (7) TO PRINT-ITEM-SEVEN.
1236          MOVE 007 TO INDEX-1.
1237          MOVE TABLE (INDEX-1) TO PRINT-ITEM-SEVEN.
1238          MOVE ZEROS TO TABLE (3000).
1239          MOVE SPACES TO PRINT-LINE.
******************** wichtig ! **************
1240* Wenn BIN-1 und X1 binäre Datentypen sind, so ist die
    * Berechnung ca. 20 mal schneller als bei dezimalen
    * Datentypen !
1242          ADD BIN-1 TO X1.
1243          IF ITEM (X1) IS EQUAL TO SPEED GO TO FAST.
1244          MOVE ALL "A" TO PRINT-LINE.
```

Symbole und Konventionen

Die Darstellung der Befehle und Zeichen in diesem Handbuch sieht wie folgt aus :

1. Kleingedruckte Namen sind beliebig definierbare Variablen oder Zahlen.
2. Unterstrichene und großgeschriebene Zeichen sind Schlüsselwörter, die zwingend benutzt werden müssen.
3. Großgeschriebene Zeichen, die nicht unterstrichen sind, stellen reservierte Ausdrücke dar.
4. Die Klammern < > zeigen an, daß zwischen den eingeklammerten Ausdrücken gewählt werden muß.
5. Eckige Klammern [ ] beinhalten Informationen, die optional angegeben werden können.
6. (...) zeigt an, daß der vorhergehende Ausdruck weitergeführt werden kann.
7. Das CARRIAGE RETURN (Return-Taste) wird durch <CR> dargestellt

Kapitel VI

Benötigte Hardware :

1. 8080/Z80/8085 Mikroprozessor
2. mindestens 32KBytes RAM für den Compiler
3. ein Diskettenlaufwerk
4. Monitor/Tastatur

Benötigte Software :

CP/M Version 2.2 von Digital Research
einen herkömmlichen Texteditor

Programmfiles auf der COBOL-Diskette :

CC.COM - dies ist der eigentliche COBOL-Compiler

W4.COM - ist ein Arbeitsfile des Compilers und muß beim Compilieren verfügbar sein

W5.CBL - dieses File anthält die Fehlermeldungen, die vom Compiler ausgeben werden. Daher muß es beim Compilieren immer verfügbar sein. Es handelt sich um ein herkömmliches Textfile, daß vom Benutzer verändert werden kann.

RUN.COM- das ist das RUN-TIME-Moduls, das zum Starten von compilierten COBOL-Programmen notwendig ist; beim Compilieren muß es nicht unbedingt verfügbar sein.

ERRORS.COM - dieses Programm zeigt die Fehlermeldungen vom letzten Compilationsvorgang noch einmal; (wird zum Compilieren nicht verwendet)

RENUMBER.CBL - dieses Programm ist auf der Diskette im COBOL-Quelltext vorhanden und muß vor Gebrauch unbedingt compiliert werden; es nummeriert die Zeilen eines anderen COBOL-Programms neu durch.

CONFIG.CBL - ebenfalls ein Programm im Quelltext; es konfiguriert das CRT-Terminal neu, z.B. Zeilenlänge, Delete-Code, usw...

CONVHEX.COM - dieses Programm kann in Verbindung mit einem CP/M Assembler benutzt werden, um HEX-Files in das OBJect Format zu bringen. CONVHEX hilft dem Anwender, Subroutinen in Assembler herzustellen.

Syntax : CONVHEX Filename [.HEX]

Das Programm erzeugt das OBJect File, das unter dem selben Namen des HEX-Files auf Diskette wieder gespeichert wird. Wenn Sie intensiver mit Assembler arbeiten, können Sie auch den Assembler von NEVADA-FORTRAN verwenden, der zu NEVADA-COBOL kompatibel ist.

Die Files W1.WRK und W3.WRK befinden sich nicht auf der Programmdiskette; sie werden vom Compiler erzeugt :

W1.WRK ist ein WORK-File des Compilers

W3.WRK ist das WORK-File für Fehlermeldungen

Arbeiten mit dem COBOL-Compiler :

ACHTUNG : Wenn die Original-Diskette noch nicht schreibgeschützt sein sollte, so tun Sie es jetzt !!!

1. NEVADA-COBOL wird auf einer Datendiskette geliefert, die kein Betriebssystem enthält. Booten Sie das System von Ihrer CP/M-Systemdiskette.
2. Versuchen Sie nicht, die NEVADA-COBOL Original-Diskette mit einem BACKUP/COPY-Programm zu kopieren ! Benutzen Sie bitte das CP/M Utility PIP.
3. Formatieren Sie eine Diskette und kopieren das CP/M-System auf die Systemspuren; kopieren Sie außerdem die Programme PIP und STAT auf diese Diskette.
4. Legen Sie diese Diskette in das Laufwerk A ein und die Originaldiskette in Laufwerk B. Um das CP/M-System neu zu initialisieren, drücken Sie bitte die Tasten CONTROL und C gleichzeitig (Ctrl+C). Um nun eine lauffähige COBOL-Arbeitsdiskette herzustellen, muß man alle Files der Original-Diskette kopieren; dies geschieht durch den CP/M-Befehl PIP :

PIP A:=B:*.*[VO]

Wenn nach Eingabe dieser Zeile die Fehlermeldung "BDOS WRITE ERROR" erscheinen sollte, ist auf der Zieldiskette in Laufwerk A: nicht genügend Platz für den COBOL-Compiler vorhanden. In diesem Fall sollte man einige Files von Diskette A: löschen und den Kopiervorgang noch einmal starten.

Entfernen Sie nun die Original-Diskette aus Laufwerk B: und bewahren Sie sie bitte an einem sicheren Ort auf.

Die COBOL-Arbeitsdiskette in Laufwerk A: darf nicht mit einem Schreibschutz versehen werden, da verschiedene Files beim Compilieren abgespeichert werden.

Booten Sie noch einmal das CP/M-System.
Als nächstes muß man das CONFIGurations-Programm compilieren; dies geschieht durch : CC CONFIG
Nun beginnt der Compiler zu arbeiten und meldet sich wieder mit "SUCCESFUL COMPILE", wenn alles glatt geht.
Falls das Compilieren jedoch nicht erfolgreich abgeschlossen wird, kann dies verschiedene Gründe haben :
- es ist möglich, daß auf der Diskette nicht genügend Platz vorhanden ist: löschen Sie in diesem Fall einige unwichtige Files oder verteilen Sie die Files auf verschiedene Laufwerke.
- die Kopie der Original-Diskette ist fehlerhaft; kopieren Sie das originale NEVADA-COBOL noch einmal.
Wenn Sie trotz allem keinen Erfolg haben, wenden Sie sich bitte an Ihren Händler.

Nach dem erfolgreichen Compilieren starten Sie das Programm CONFIG, indem Sie eingeben : RUN CONFIG
Bei dieser Eingabe ist darauf zu achten, daß sich die Programme RUN.COM und CONFIG.OBJ auf der angemeldeten Diskette befinden.
Die Aufgabe des CONFIG-Programms besteht nämlich darin, das RUN-Time-Programm (RUN.COM) an den verwendeten Computer anzupassen.
Das Programm meldet sich wie folgt :

```
ENTER SCREEN INFORMATION
ENTER 2-DIGIT HEXADECIMAL CODE FOR DELETE-KEY
 Der Code für Ctrl-H unter CP/M ist 08; für Apple 15H
 (H bedeutet hexadezimal-System)
ENTER 2-DIGIT HEXADECIMAL CODE SENT TO BACKSPACE CURSOR
 Standard-Code ist 08
IS THE BACKSPACE PRECEEDED WITH AN ESCAPE CHARACTER (Y/N) ?
 Standardantwort ist N
ENTER # OF CHARACTERS ACROSS SCREEN
 Standard sind 80 oder 64 Zeichen pro Zeile
ENTER # OF LINES PER SCREEN PAGE
 Standard sind 24 oder 16 Zeilen
DOES YOUR BIOS ISSUE A CR/LF
AT THE END OF EACH LINE ?
 Standardantwort Y
DOES YOUR PRINTER REQUIRE A LINE FEED (Y/N) ?
 Standardantwort Y
DO YOU WANT TO USE CP/M FUNCTION 1 & 2
CONSOLE I-O (Y/N) ?
 je nach Wunsch Y oder N eingeben
DOES YOUR CP/M BACKSPACE AND BLANK THE
DELETED CHARACTERS (Y/N) ?
 Standardantwort ist N
DO YOU WANT TO ACCEPT ANY HEX CHARACTER
OR ONLY DISPLAY ASCII (H/A) ?
 Standardantwort ist A
EOJ CONFIG RETURNING TO CPM
```

Wenn Sie nun mit der Bildschirmausgabe zufrieden sind, kann das CONFIG-Programm gelöscht werden, nachdem das RUN-Time-Programm angepaßt wurde.
ACHTUNG : CONFIG nur von der ARBEITSDISKETTE löschen !!!
Eingabe : ERA CONFIG.*

Da manche Diskettenformate nur eine geringe Aufzeichnungskapazität haben (z.B. 5.25 Zoll mit Single Density), ist es sinnvoll, den Compiler auf mehrere Disketten zu verteilen, damit keine Fehler durch mangelnden Speicherplatz auftreten :
 1 Compilerdiskette mit den Programmen : CC.COM, W4.COM, W5.COM
 1 RUN-Time Diskette, die nur das RUN.COM-Programm enthält
Auf diese Weise läuft der COBOL-Compiler auch auf Laufwerken mit geringer Kapazität.

Erstellen eines COBOL-Programms

Um ein COBOL-Programm zu schreiben, benötigt man einen Text-Editor wie z.B. NEVADA-EDIT. Um Zeit zu sparen, ist es sinnvoll, fertige Programme wie RENUMBER.CBL mit dem Editor umzuschreiben und unter einem anderen Namen wie MYPROG.CBL abzuspeichern. Das erspart eine Menge Tipparbeit und man gerät nicht in die Gefahr, die Schlüsselworte falsch einzugeben.

Außerdem muß jede Zeile des COBOL-Programms mit einem Carriage-Return abgeschlossen werden, was von Texteditoren wie NEVADA EDIT, ED (wird mit CP/M ausgeliefert) und WORDSTAR automatisch gemacht wird. Weiterhin sollten TAB's vermieden werden; bei der Verwendung von WORDSTAR muß auf jeden Fall im NON DOCUMENT Modus geschrieben werden. Für alle Texteditoren gilt, daß der Filetyp eines COBOL-Quellprogramms immer .CBL lauten muß.

Das COBOL - Programmformat

		ANSI-1974 Spalte	NEVADA-COBOL Spalte
Zeilennummern	:	1-6	1-4
Indikator-Feld	:	7	5
A-Feld	:	8-11	6-9
B-Feld	:	12-72	10-70

1. Die Zeilennummern in NEVADA-COBOL sind immer 4 Zeichen lang. Um Programme von NEVADA-COBOL für den ANSI-Standard umzuschreiben, müssen 2 Zeichen hinzugefügt werden.

2. Das Indikator-Feld darf folgende Zeichen beinhalten :
   - * = wenn dieses Zeichen im Indikator-Feld steht, ist die Zeile nur zu Dokumentation gedacht und wird vom Compiler ignoriert.
   - / = es handelt sich um eine Kommentarzeile
   - D = bei dieser Zeile handelt es sich um eine Debug-Zeile
   - - = die vorherige Zeile wurde getrennt und hier weitergeführt. Wenn Strings nicht in eine Zeile passen und getrennt werden, so muß in der zehnten Spalte dieser Zeile auch ein Anführungszeichen stehen.

   Steht im Indikator-Feld ein Leerzeichen, so handelt es sich um eine übliche COBOL-Programmzeile. Sollten andere Zeichen im Indikator-Feld stehen, so werden die jeweiligen Zeilen als Kommentar betrachtet und nicht compiliert.

3. Bei Eingabe eines Programms müssen Zeilennummern verwendet werden, da bei Fehlermeldungen jeweils die Angabe der Zeilennummer und der Fehlerart erfolgt.

4. Jede Zeile muß durch ein Return (Carriage Return) beendet werden.

Beispiel :

```
Spalten :
12345678901234567890123456789O
0001 IDENTIFICATION DIVISION.
0002* dies ist eine Kommentar-Zeile und dient ausschließlich
0003* zu Dokumentationszwecken.
```

Compilieren eines Programms

Um ein Programm zu compilieren gibt man einfach folgendes ein :

CC Filename

Dann meldet sich der Compiler mit einer Copyright-Information; falls im Programm keine Fehler entdeckt wurden, wird "SUCCESSFUL COMPILE" ausgegeben; andernfalls erfolgt der Error-Report mit Fehlerbeschreibungen und Zeilennummern. Durch Drücken von Ctrl+C kann der Compiler angehalten werden.
Hier noch ein Beispiel (A> ist das CP/M-Eingabezeichen/Prompt)

A>CC RENUMBER <CR>

Das Programm RENUMBER ist auf der gerade angemeldeten Diskette A: enthalten; deshalb wird auch das Objekt-File unter dem Namen RENUMBER.OBJ auf der Diskette abgespeichert. Außerdem wird das Work-File W1 erzeugt.

A>CC QUELLPRG.BBB <CR>

Bei dieser Eingabe wird das Programm QUELLPRG von Laufwerk B: gelesen und nach Bearbeitung wird das Objekt-File QUELLPRG.OBJ und W1 auf die Diskette in Laufwerk B: geschrieben. Hier enthält der Filetyp die notwendigen Informationen für den Compiler, wo sich das Quellprogramm befindet, und wo das Objekt-File und das Workfile W1 abzuspeichern werden sollen.

Position 1 des Filetyps bestimmt das Laufwerk mit dem Quellprogramm.
Position 2 des Filetyps bestimmt, auf welchem Laufwerk das Objekt-File abgespeichert werden soll.
Position 3 des Filetyps bestimmt, auf welchem Laufwerk das Work-File W1 abgespeichert werden soll.

Beispiel :

A>CC CONFIG.ABB <CR>

In diesem Fall befindet sich das Quellprogramm CONFIG.CBL auf der Diskette in Laufwerk A:; das Objekt-File CONFIG.OBJ und das Work-File wird auf Diskette in Laufwerk B: abgelegt.

Der Filetyp des Quellprogramms muß immer .CBL sein !
Bei dem Compileraufruf darf allerdings nie der Filetyp mit eingegeben werden, da der Compiler die Laufwerke C: und L: ansprechen würde, was wiederum eine Fehlermeldung verursacht (BDOS SELECT ERROR).

Ausführen eines Programms

Wenn ein Programm fehlerfrei compiliert wurde, kann es durch das RUN-Time-Modul gestartet werden. Das RUN-Programm wird in den Speicher von 0100H bis 2EFFH geladen und enthält den Programm-Lader und die RUN-Time Routinen. Der Programmablauf kann durch Ctrl + C gestoppt werden.

```
A>RUN u:OBJEKT <CR>
```

In diesem Fall ist das RUN-Programm auf Laufwerk A: vorhanden; das OBJEKT-Programm kann sich auf einem anderen Laufwerk u: befinden. Wird u: nicht angegeben, spricht das RUN-Programm automatisch das aktuelle Laufwerk (A:) an.

```
B>RUN A:OBJEKT <CR>
```

Bei diesem Beispiel enthält Laufwerk B: das RUN-Programm, wobei sich das OBJEKT-Programm auf der Diskette in Laufwerk A: aufhält.

```
A>RUN RENUMBER <CR>
```

Dieser Befehl startet das RENUMBER-Programm von der Diskette in Laufwerk A:. Nachdem das Programm geladen ist, erscheint folgende Meldung : ENTER FILE NAME A:FILENAME.TYP
Hier können Sie den Namen eines Ihrer eigenen Programme eingeben, oder auch den Namen von CONFIG.CBL. Die Funktion des RENUMBER-Programms besteht darin, die Zeilennummern eines COBOL-Quellprogramms neu zu nummerieren und dadurch das Programm übersichtlich zu gestalten. Wenn die Ausführung beendet ist, verabschiedet sich das Programm mit RENUMBERING COMPLETE. Sollten Programmzeilen jedoch Tabulator-Zeichen enthalten oder Leerzeilen vorhanden sein, so werden diese Zeilen gelöscht, da der verwendete REWRITE-Befehl die Eingabedatei nicht expandieren kann.
Achtung, unter einigen CP/M-Systemen kann es passieren, daß Files mit Leerzeilen oder Tab's beim Bearbeiten mit RENUMBER gelöscht werden !

Beispiel :

```
Spalten
12345678901234567890123456789O
0001*
     *
9999/ Dies ist ein Kommentar am Ende des Programms
```

Auflisten eines Programms

Um eine Quellprogramm aufzulisten, benutzen Sie bitte den CP/M-Befehl TYPE. Wenn gleichzeitig eine Ausgabe auf Drucker erfolgen soll, muß Ctrl + P gedrückt werden.

```
A>TYPE RENUMBER.CBL [CTRL+P] <CR>
A>TYPE CONFIG.CBL <CR>
A> TYPE W5.CBL <CR>
```

Kapitel VII

Fehlermeldungen und Codes

Fehlermeldungen des Compilers

Beim Compilieren werden alle Fehlermeldungen unter dem File W3.WRK auf Diskette gespeichert. Immer, wenn der Compiler ein Programm-Feld bearbeitet hat, wird nach schwerwiegenden Fehlern gesucht. Falls das Programm fehlerhaft ist, beendet der Compiler die Bearbeitung. Nach Beendigung des Compilierens steht dem Anwender ein Fehlerprotokoll zur Verfügung, welches mit dem Programm ERRORS aufgelistet wird : A>ERRORS <CR>
Mit Ctrl+P erfolgt eine Ausgabe auf Drucker, mit Ctrl+S kann die Ausgabe angehalten werden; nochmaliges Drücken von Ctrl+S führt die Ausgabe weiter.

Alle Fehlermeldungen sind im File W5.CBL enthalten und können vom Benutzer beliebig verändert werden. Sie können sowohl in Groß- oder Kleinschrift erscheinen und auch länger als eine Zeile sein.

Der Compiler unterscheidet bei den Fehlermeldungen zwischen unbedingten (fatal) und möglichen Fehlern. Im Fall von unbedingten Fehlern wird kein Objekt-Code erzeugt und der Compiler beendet die Bearbeitung.
In der folgenden Tabelle bedeutet F=unbedingter (fatal) Fehler, W=Warnung, möglicher Fehler.

SEQ NO.	ERR COL	LEVEL	TEXT
9999	70 001	F	SYNTAX ERROR

Die vollständige Liste der Fehlermeldungen finden Sie im Anhang Nevada Cobol Fehlermeldungen.

RUN-TIME Fehlermeldungen

Das RUN-Time Programm gibt den Filenamen und den Fehlercode an.
Codes :

90 KEINE INFORMATIONEN VERFÜGBAR
1 FEHLER BEIM ERWEITERN DES FILES
92 DISKETTE IST VOLL
93 FILE WURDE NICHT GEÖFFNET
94 KEIN DIRECTORY-EINTRAG MEHR VERFÜGBAR - DISKETTE VOLL
95 FILE NICHT GEFUNDEN
96 FILE WURDE SCHON GEÖFFNET
97 KEINE DATEN IM WAHLFREIEM ZUGRIFF VERFÜGBAR
98 REWRITE-BEFEHL OHNE VORHERIGEN READ-BEFEHL IM I-O MODUS
99 ES WURDE VERSUCHT EIN OUTPUT-FILE ZU LESEN ODER IN EIN INPUT-FILE ZU SCHREIBEN
100 FEHLERMELDUNG NICHT IN TABELLE
101 FEHLER BEIM DIMENSIONIEREN
102 RECORD-GRENZE ÜBERSCHRITTEN

Achtung : Es muß immer das RUN-Time Modul verwendet werden, das mit dem Compiler geliefert wird. Ältere Versionen laufen nicht ! Falls Programme mit einem älteren Compiler erstellt wurden, sind diese ebenfalls zu diesem RUN-Time Programm nicht kompatibel. In diesem Fall muß der Quelltext recompiliert werden.

Kapitel VIII

Liste des ANSI-74 Standards und des NEVADA-COBOL Sprachumfangs

ACCEPT	X
ACCESS	X
ADD	X
ADVANCING	X
AFTER	X
ALL	X
ALPHABETIC	X
ALSO	
ALTER	X
ALTERNATE	
AND	X
ARE	X
AREA	X
AREAS	
ASCENDING	
ASSIGN	X
AT	X
AUTHOR	X
BEFORE	X
BLANK	
BLOCK	X
BOTTOM	
BY	X
CALL	X
CANCEL	X
CD	
CF	
CH	
CHARACTER	
CHARACTERS	X
CLOCK-UNITS	
CLOSE	X
COBOL	
CODE	
CODE-SET	
COLLATING	X
COLUMN	
COMMA	X
COMMUNICATION	
COMP	X
COMPUTATIONAL	X
COMPUTE	
CONFIGURATION	X
CONTAINS	X
CONTROL	
CONTROLS	
COPY	X
CORR	
CORRESPONDING	
COUNT	
CURRENCY	X

DATA	X
DATE	
DATE-COMPILED	X
DATE-WRITTEN	X
DAY	
DE	
DEBUG-CONTENTS	
DEBUG-ITEM	
DEBUG-LINE	
DEBUG-NAME	
DEBUG-SUB-1	
DEBUG-SUB-2	
DEBUG-SUB-3	
DEBUGGING	X
DECIMAL-POINT	X
DECLARATIVES	
DELETE	
DELIMITED	
DELIMITER	X
DEPENDING	X
DESCENDING	
DESTINATION	
DETAIL	
DISABLE	
DISPLAY	X
DIVIDE	X
DIVISION	X
DOWN	
DUPLICATES	
DYNAMIC	
EGI	
ELSE	X
EMI	
ENABLE	
END	X
END-OF-PAGE	
ENTER	
ENVIRONMENT	X
EOP	
EQUAL	X
ERROR	X
ESI	
EVERY	
EXCEPTION	
EXIT	X
EXTEND	

FD	X	MEMORY	X
FILE	X	MERGE	
FILE-CONTROL	X	MESSAGE	
FILLER	X	MODE	X
FINAL		MODULES	X
FIRST	X	MOVE	X
FOOTING		MULTIPLE	
FOR	X	MULTIPLY	X
FROM	X		
		NATIVE	
GENERATE		NEGATIVE	
GIVING	X	NEXT	X
GO	X	NO.	X
GREATER	X	NOT	X
GROUP		NUMBER	
		NUMERIC	X
HEADING			
HIGH-VALUE	X	OBJECT-COMPUTER	X
HIGH-VALUES	X	OCCURS	X
		OF	X
I-O	X	OFF	X
I-O-CONTROL	X	OMITTED	X
IDENTIFICATION	X	ON	X
IF	X	OPEN	X
IN		OPTIONAL	
INDEX		OR	X
INDEXED		ORGANIZATION	X
INDICATE		OUTPUT	X
INITIAL	X	OVERFLOW	
INITIATE			
INPUT	X	PAGE	X
INPUT-OUTPUT	X	PAGE-COUNTER	
INSPECT	X	PERFORM	X
INSTALLATION	X	PF	
INTO	X	PH	
INVALID	X	PIC	X
IS	X	PICTURE	X
		PLUS	
JUST	X	POINTER	
JUSTIFIED	X	POSITION	
		POSITIVE	
KEY	X	PRINTING	
		PROCEDURE	X
LABEL	X	PROCEDURES	
LAST		PROCEED	X
LEADING	X	PROGRAM	X
LEFT	X	PROGRAM-ID	X
LENGTH		QUEUE	
LESS	X	QUOTE	X
LIMIT		QUOTES	X
LIMITS			
LINAGE		RANDOM	X
LINAGE-COUNTER		RD	
LINE	X	READ	X
LINE-COUNTER		RECEIVE	
LINES	X	RECORD	X
LINKAGE	X	RECORDS	X
LOCK		REDEFINES	X
LOW-VALUE	X	REEL	
LOW-VALUES	X	REFERENCES	

RELATIVE	X	TABLE	
RELEASE		TALLYING	X
REMAINDER		TAPE	
REMOVAL		TERMINAL	
RENAMES		TERMINATE	
REPLACING	X	TEXT	
REPORT		THAN	X
REPORTING		THROUGH	X
REPORTS		THRU	X
RERUN		TIME	
RESERVE		TIMES	X
RESET		TO	X
RETURN		TOP	
REVERSED		TRAILING	
REWIND		TYPE	
REWRITE	X		
RF		UNIT	
RH		UNSTRING	
RIGHT	X	UNTIL	X
ROUNDED	X	UP	
RUN	X	UPON	
		USAGE	X
SAME	X	USE	
SD		USING	X
SEARCH			
SECTION	X	VALUE	X
SECURITY	X	VALUES	
SEGMENT		VARYING	
SEGMENT-LIMIT			
SELECT	X	WHEN	
SEND		WITH	X
SENTENCE	X	WORDS	X
SEPARATE		WORKING-STORAGE	X
SEQUENCE	X	WRITE	X
SEQUENTIAL	X		
SET		ZERO	X
SIGN	X	ZEROES	X
SIZE	X	ZEROS	X
SORT			
SORT-MERGE		+	
SOURCE		-	
SOURCE-COMPUTER	X	*	
SPACE	X	/	
SPACES	X	**	
SPECIAL-NAMES	X	>	X
STANDARD	X	<	X
STANDARD-1		=	X
START			
STATUS	X		
STOP	X		
STRING			
SUB-QUEUE-1			
SUB-QUEUE-2			
SUB-QUEUE-3			
SUBTRACT	X		
SUM			
SUPPRESS			
SYMBOLIC			
SYNC	X		
SYNCHRONIZED	X		

Kapitel IX

Der Aufbau von COBOL

Einleitung

Das NEVADA-COBOL System besteht aus drei Komponenten, nämlich dem Programm, das in einer speziellen Sprache (COBOL) geschrieben ist, dem Compiler, der dieses Programm in den Objekt-Code übersetzt (Maschinensprache), und schließlich dem RUN-Time-Modul, welches das compilierte Programm startet und außerdem einige wichtige Routinen enthält.

Das COBOL-Programm
Wenn die Lösung für ein bestehendes Problem ausgearbeitet ist, kann das COBOL-Programm erstellt werden. Die Sprache COBOL wurde speziell dazu entwickelt, die anfallenden Daten aus dem kaufmännischen und industriellen Bereich leicht zu verarbeiten.

Jedes COBOL-Programm besteht aus vier Abschnitten :

1. IDENTIFICATION DIVISION - dieser Teil dient zur Beschreibung und Dokumentation des Programms
2. ENVIRONMENT DIVISION - hier stehen Informationen über die verwendete Hardware
3. DATA DIVISION - in diesem Abschnitt werden die benötigten Datenspeicher definiert, z.B. Ausgabe/Eingabedateien
4. PROCEDURE DIVISION - hier erfolgt die eigentliche Problemlösung durch Berechnung, etc...

Der IDENTIFICATION DIVISION - Teil dient ausschließlich zur Dokumentation des Programms und enthält Informationen über die Funktion des Programms, Programmierer, Copyright, Datum und die verwendete Hardwarekonfiguration.
Genauere Angaben können Sie der Beschreibung entnehmen.

Der zweite Teil, der ENVIRONMENT DIVISION - Abschnitt, erfüllt eine ähnliche Funktion wie das IDENTIFICATION DIVISON-Feld. Dieser Teil ist in einzelne Sektionen aufgeteilt, deren Reihenfolge beliebig ist. Diese Sektionen erscheinen wie einzelne Einträge im ENVIRONMENT DIVISION - Feld und haben verschiedene Aufgaben. Jede Sektion wird durch den speziellen Sektions-Namen eingeleitet; auf diesen Namen folgt nun ein Leerzeichen, dann das Wort "SECTION" und ein Punkt (.).

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. - Der Computer, der zum Compilieren des Programms verwendet wurde (z.B. 8080 CPU).
OBJECT-COMPUTER. - Der Computer, auf dem das Programm lauffähig ist (8080 CPU).
INPUT-OUTPUT SECTION.
FILE-CONTROL. - In diesem Abschnitt wird der Computer über die verwendeten Datenfiles informiert :
   1. Filename, der vom Programmierer gewählt wird und bis zu 30 Zeichen lang sein darf.
   2. Definition der externen Speicher, auf die das File ausgegeben oder von denen das File eingelesen wird; entweder Drucker oder Diskettenlaufwerk.

Für jedes File lautet der Befehl in der FILE-CONTROL Sektion :

SELECT Filename ASSIGN TO Drucker/Diskettenlaufwerk

Der dritte Teil eines COBOL-Programms ist das DATA DIVISION-Feld. Für COBOL-Programme existieren vier Kategorien von Daten :

1. Daten, die aus Eingabefiles gelesen werden und dann wieder in Ausgabefiles geschrieben werden. Für diese Daten müssen Speicherbereiche reserviert werden, um Daten einlesen und abspeichern zu können.
2. Daten, die intern berechnet werden und in einem Zwischenspeicher abgelegt werden.
3. Konstanten und Variable
4. Daten von anderen COBOL-Programmen

Das DATA DIVISION - Feld besteht aus drei Sektionen.

1. FILE SECTION
2. WORKING-STORAGE SECTION
3. LINKAGE SECTION.

Im FILE-SECTION-Feld werden die Filenamen und die dazugehörigen Ein/Ausgabegeräte definiert. Für jedes File wird der Record-Name, der Record-Aufbau, Größe und Format eines Feldes beschrieben. Das WORKING-STORAGE-Feld bestimmt die Konstanten (wie z.B. ZERO) und die Speicherbereiche. Format, Größe und Inhalt von Zeigern, Konstanten und Zwischenspeichern werden dort deklariert. Der LINKAGE SECTION-Teil ermöglicht die Bearbeitung von Daten aus anderen Programmen.

Der vierte und letzte Teil eines COBOL-Programms besteht aus dem PROCEDURE DIVISION-Abschnitt. Dieser Programmteil enthält die Anweisungen und Befehle, die die Daten bearbeiten und ist auch in Sektionen und Prozeduren gegliedert.

Austesten von COBOL-Programmen

Meistens treten beim Compiliern Fehlermeldungen auf, da kaum ein Programm auf Anhieb fehlerfrei geschrieben wird. Bei diesen Fehlermeldungen unterscheidet der Compiler zwischen fatalen Fehlern und Warnungen. Findet der Compiler einen fatalen Fehler, beendet dieser das Compilieren und kein Objekt-Code wird generiert. Gibt der Compiler lediglich eine Warnung aus, so wird die Bearbeitung fortgesetzt und Objekt-Code erzeugt.

Um einen lauffähigen Objekt-Code zu erhalten, muß das COBOL-Programm (Quelltext) so lange "entwanzt" (debugged;bug=Wanze) werden, bis beim Compilationsvorgang keine Fehlermeldungen ausgegeben werden und ein fehlerfreier Objekt-Code erzeugt wird.

Logische Fehler werden vom Compiler nicht entdeckt; deshalb sollte man Daten eingeben und ausprobieren, ob das Programm das gestellte Problem eindeutig löst.

Befehle

Die Befehle der Sprache COBOL sind standardisiert. Eine typische Zeile in einem COBOL-Programm würde lauten :

```
ADD ZAHL-1 TO ZAHL-2 GIVING RESULTAT-DER-BERECHNUNG
```

Wie dieses Beispiel zeigt, ist COBOL verhältnismäßig leicht zu erlernen, da größtenteils englische Ausdrücke und Wörter verstanden werden und keine komplizierten Syntaxregeln existieren. Das Programm ist außerdem leicht zu lesen und gut dokumentiert (z.B. IDENTIFICATION DIVISION-Feld), was für den Anwender sehr wichtig ist, da Programme öfters geändert werden müssen. Viele Anweisungen, die nur zur Dokumentation dienen, brauchen vom Programmierer nicht berücksichtigt werden; diese sollten aber trotzdem verwendet werden, damit auch anderen Anwendern genügend Informationen zur Verfügung stehen, um das vorhandene Programm benutzen zu können.

In COBOL gibt es zwei Kategorien von Ausdrücken, nämlich die reservierten und nicht-reservierten, wobei der Programmierer die reservierten Ausdrücke (Schlüsselwörter) nicht als File- oder Variablennamen verwenden darf.

```
   MOVE  Datenteil-1  TO  Datenteil-2.
```

In dieser Programmzeile sind die Schlüsselwörter in Großschrift dargestellt; die Datennamen wie z.B. Datenteil-1 können beliebig gewählt werden, sofern kein reservierter Ausdruck verwendet wird.

Im Anhang des Handbuchs befindet sich eine Tabelle mit den reservierten Schlüsselwörtern, der Sie entnehmen können, ob ein gewünschter Variablenname auch benutzt werden darf.

Datennamen können in COBOL bis zu dreißig Zeichen lang sein und aus den Zeichen A-Z, 0-9 und "-"; jedoch muß berücksichtigt werden, daß das Minuszeichen "-" nicht zu Beginn oder am Ende eines Datennamens stehen darf. Ferner müssen Datennamen mindestens ein alphabetisches Zeichen enthalten.

A1985 wäre ein gültiger Datenname, wogegen jedoch A27.12.85 nicht erlaubt wäre, da ein Dezimalpunkt enthalten ist. Auf einen Datennamen darf ein Leerzeichen (Space), Punkt, runde Klammer, Komma oder Semikolon direkt folgen. Außerdem kann innerhalb eines Datennamens kein Leerzeichen stehen; MEHRWERT-STEUER ist ein gültiger Name, wogegen jedoch MEHRWERT STEUER nicht verwendbar ist.

Die Regeln für die Konstruktion von Datennamen gelten ebenso für Filenamen, Recordnamen und Variablennamen.

Wenn Sie keinen Datennamen mehr angeben wollen, kann man den Namen FILLER verwenden. Dies ist auch auf die unbenutzten Felder eines logischen Records anwendbar. Sollen diese Felder trotzdem benutzt werden, kann dieses nicht mit dem Datennamen FILLER geschehen; auch eine Zuweisung mit MOVE FILLER TO DATEN-NAME ist nicht durchführbar, da auf FILLER nicht direkt zugegriffen werden kann. In diesem Fall muß eine Daten-Gruppe definiert werden.

Syntax

Um das Ende eines Felds, einer Sektion oder einer Programmzeile zu kennzeichnen, wird der Dezimalpunkt "." verwendet. Der erste Programmteil, IDENTIFICATION DIVISION. , muß von einem Punkt gefolgt werden. Der Inhalt der einzelnen Einträge im IDENTIFICATION DIVISION-Abschnitt ist beliebig; am Ende jedes Abschnitts muß jedoch immer ein Punkt stehen.

Die häufigste Fehlerursache beruht auf Syntaxfehlern des Programmierers. Sollte der Compiler die Fehlermeldung "SYNTAX ERROR" ausgeben, kann dies sowohl die Compilierung beenden, also ein fataler Fehler sein, als auch eine Warnung sein. Syntaxfehler können mehrere Ursachen haben, z.B. müssen die Filenamen, die in der DATA DIVISION deklariert werden, mit denen in der ENVIRONMENT DIVISION und PROCEDURE DIVISION übereinstimmen, ansonsten wird für diesen Fehler ein SYNTAX ERROR ausgegeben und der Compiler beendet seine Arbeit.

Format

Bitte lesen Sie im Handbuch über die Symbole und Konventionen nach. Das COBOL-Programmformat und Befehlsformat ist standardisiert: optionale (wahlweise) Befehle stehen immer in eckigen Klammern. Entschließt man sich dazu, einen optionalen Befehl zu verwenden, so müssen alle Elemente aus der Klammer formal eingegeben werden :

```
[: BLOCK CONTAINS ganze Zahl CHARACTERS]
```

Für den Fall, daß man diesen optionalen Befehl verwenden will, müssen unbedingt die Schlüsselwörter BLOCK und CHARACTERS benutzt werden; CONTAINS kann verwendet werden. Außerdem muß die Integerzahl ebenfalls angegeben werden, da diese Opernden des Befehls sind.

Das DATA DIVISON-Feld

Das DATA DIVISION-Feld definiert die Daten, die in dem PROCEDURE DIVISION-Feld bearbeitet werden sollen.

DATA DIVISION.
FILE SECTION.
Für jedes File, das im FILE-CONTROL-Feld der INPUT-OUTPUT-Sektion deklariert wurde, gibt es in der FILE-SECTION zwei Felder :

1. File-Definition (FD=FILE DESCRIPTION)
   a. Filename (der gleiche Filename wie im FILE-CONTROL-Feld)
   b. LABEL RECORDS ARE STANDARD oder LABEL RECORDS ARE OMITTED.
      VALUE OF FILE-ID IS "Wert" oder Datenname.
   c. Namen des Records
      DATA RECORD IS (Record-Name).
2. Record-Definition (01)
   Die Beschreibung der Record-Eigenschaften findet genau nach der File-Beschreibung statt. Dieses Feld veranlaßt den Compiler, für den Record Speicherbereich zu reservieren, um die Daten einlesen, bearbeiten und abspeichern zu können. Außerdem wird das Format des Records angegeben (PICTURE-Feld). Jedes Feld beginnt mit der Ordnungszahl; darauf folgt ein Leerzeichen und dann der Datenname mit einzelnen Definitionen

Das Format eines Records wird von links nach rechts angegeben.

Ordnungszahl
Die hierarchische Ordnung der Daten innerhalb eines logischen Records geschieht mit Hilfe der Ordnungszahl. Es können bis zu 49 verschiedene Datensätze spezifiziert werden (von 01 bis 49 durchnummeriert, wobei 01 ranghöchster Satz ist)

Wird einem Record ein Name zugewiesen, so hat dieser immer die höchste Ordnungszahl, nämlich 01. Größere Datenfelder besitzen die nächstkleinere Ordnungszahl 02. Werden untergeordnete Datenfelder definiert (Subfields), so werden diesen kleinere Ordnungszahlen zugewiesen, wie 03,04,05, etc... Um anzuzeigen, daß ein elementares (nicht unterteiltes) Datenfeld zu einer Datengruppe gehört, weist man ihm eine höhere Ordnungszahl als der Datengruppe zu :

```
      02 KUNDEN-CODE
      02 KUNDEN-NAME
         03 NAME
         03 VOR-NAME
      02 KUNDEN-ADRESSE
```

Um das Ende einer Datengruppe zu kennzeichnen, benutzt man eine Ordnungszahl, die gleich oder kleiner ist als die Ordnungszahl der Datengruppe ist.
Im File-Description-Feld folgt auf die Buchstaben "FD" unmittelbar der Filename. Bei der Record-Definition folgt der Record-Name auf die Ordnungszahl 01

Innerhalb des Records wird jedem Datenfeld ein Name zugewiesen, der bis zu dreißig Zeichen lang sein kann. Unter diesem Namen kann man von dem PROCEDURE DIVISON-Teil aus auf die Daten zugreifen. Werden Datenfelder nicht benötigt, können sie mit dem Namen FILLER belegt werden.

Format

Für jedes Datenfeld wird die Größe, der Typ (numerisch, alphabetisch oder alphanumerisch) und die Position des Dezimalpunkts bestimmt; dieses Format nennt man auch die Ein- und Ausgabe-Maske des Datenfelds. Die Definition der Masken geschieht mit Hilfe des PICTURE-Befehls und besonderen Symbolen :

9 - steht für ein numerisches Zeichen (0-9)
A - steht für ein alphabetisches Zeichen (A-Z)
X - steht für ein alphanumerisches Zeichen; wird auch verwendet, um FILLER-Positionen zu beschreiben.
V - imaginärer Dezimalpunkt, verbraucht im Datenfeld keinerlei Platz

Zeichen für Ausgabe auf Drucker, speziell für Geldbeträge :
. - Dezimalpunkt, wird in das Datenfeld eingesetzt und wird bei der Bestimmung der Feldgröße mit berechnet
, - Komma, wird in das Datenfeld eingesetzt
Z - wird dieses Symbol verwendet, so werden alle vorangehenden Nullen und Kommas bei Druckerausgabe ignoriert

Außerdem ist es möglich, das PICTURE-Feld abzukürzen, indem man das gewünschte Symbol und dann in Klammern die Anzahl der Wiederholungen eingibt; ein siebenstelliges numerisches Feld kann so aussehen :

9999999 oder 9(7)

Mit Dezimalpunkt :

999999V99 oder 9(6)V99

Ein alphanumerisches Feld aus zwanzig Zeichen :

XXXXXXXXXXXXXXXXXXXX oder X(20)

Beim PICTURE-Befehl ist darauf zu achten, daß nur dreißig Zeichen lange Felder beschrieben werden können.

Mit Hilfe des PICTURE-Befehls kann man auch das Dollarzeichen "$" Kommas und Dezimalpunkte in numerische Werte einfügen. Dies ist besonders nützlich, wenn Geldbeträge auf Drucker ausgegeben werden. Bei den folgenden Beispielen erfolgt zuerst die Eingabe der Werte (z.B. über Tastatur), danach werden die Daten den Ausgabe-Masken zugewiesen (mit Hilfe des MOVE-Befehls) und können danach auf Drucker ausgegeben werden :

Eingabe-Masken	Ausgabe-Masken (für Drucker)
PICTURE IS 9(4)	PICTURE IS $9,999
PICTURE 9V99	PICTURE IS $9.99
PICTURE 9(8)V99	PICTURE IS $99,999,999.99

Bei numerischen Daten muß man berücksichtigen, daß immer genügend Stellen (9=numerischer Wert) vorhanden sind, um die Daten zu erfassen; ist dies nicht der Fall, so werden die Daten gerundet oder verkürzt.

Insbesondere Ein/Ausgabe-Masken von numerischen Daten, die ausgedruckt werden, beinhalten Dezimalpunkte oder Kommas. Sonstige Daten, die auf Diskette abgespeichert werden, enthalten hauptsächlich algebraische Zeichen wie "+","-" und ".". Diese werden durch die Symbole S und V vertreten.

Der Dezimalpunkt in Zahlen wird normalerweise durch das Symbol V markiert. Wenn Daten in arithmetischen Operationen verwendet werden, so muß immer der (imaginäre) Dezimalpunkt immer durch V in der Ein/Ausgabe-Maske markiert sein, auf keinen Fall jedoch durch "." . Eine Ausgabe des Plus-Zeichens erfolgt nur, wenn es auch in der Maske angegeben wurde. Sollten die Symbole "-", "CR" oder "DB" in einer Maske enthalten sein, die Daten jedoch positiv sein, so werden nur Leerzeichen ausgegeben. (CR und DB zeigen positive/negative Werte an : CREDIT/DEBIT)

Solange kein Vorzeichen angegeben wird, werden die Daten als positiv interpretiert. Um negative Werte zu erhalten, muß das Datenfeld, in das die Werte eingelesen werden sollen, ein Vorzeichen enthalten. Das Datenfeld, von dem die Werte ausgegeben werden sollen (Drucker), muß entweder ein +,-,CR oder DB Symbol beinhalten.

X-FELD PICTURE IS 9(3)V99. -----> Y-FELD PICTURE IS +9(3).99.

Bringt man unter Verwendung des MOVE-Befehls die Daten aus dem X-Feld ins Y-Feld, so sind diese immer positiv. Der Grund dafür ist, daß in der Maske des X-Felds kein Vorzeichensymbol markiert wurde.

X-FELD PICTURE IS S9(3)V99. ----> Y-FELD PICTURE IS +9(3).99.

Nun können die Daten im Y-Feld entweder positiv oder negativ sein, da der Maske des X-Felds ein Vorzeichen-Symbol (S) hinzugefügt wurde.

Um die Ausgabe von Nullen zu verhindern, existieren zwei Sorten von Symbolen : die fließenden Zeichen +,-,$, und die (nicht-fließenden) Zeichen Z,*.

Wird der Wert (-)003V19 einem Datenfeld zugewiesen, dessen PICTURE-Maske als $$$$.99DB definiert ist, so enthält das Datenfeld einen Wert von $3.19DB

Das Dollar-Zeichen "$" ist gleichbedeutend mit dem Symbol "9". Das erste Dollar-Zeichen in der Maske wird jedoch immer auch als solches ausgegeben. Das bedeutet, daß das Feld mit der Maske $$$$.99 nur fünft numerische Zeichen enthalten kann.

45612V34	Dieser Wert ist im X-Feld enthalten und wird in das Y-Feld gebracht (mit dem MOVE-Befehl)
$$$$$.99	Maske des Y-Felds
$5612.34	Resultat

In diesem Beispiel wurde das höchstwertige Zeichen, nämlich die vier, abgeschnitten, um für das Dollar-Zeichen Platz zu schaffen. Um das zu verhindern, muß man die Maske des Y-Felds so definieren : PICTURE IS $$$$$$.99

Der OCCURS-Befehl

Wenn ein Datensatz gleich mehrmals hintereinander vorkommen soll, verwendet man den OCCURS-Befehl. Sobald man den Datensatz abfragt, muß auch angegeben werden, welcher Datensatz gemeint ist. Dies geschieht mit Hilfe der Ordnungszahl :

```
IF  DATEN-SATZ (9)  = "N"  THEN ...
```

Das Beispiel setzt voraus, daß die Daten in DATEN-SATZ durch den OCCURS-Befehl mindestens neun mal angelegt wurden. Mit der Ordnungszahl 9 kann man das neunte Element des Datensatzes lesen.

Bei dem Zugriff muß beachtet werden :

1. Der linken Klammer muß ein Leerzeichen vorangehen; der rechten Klammer muß ein Leerzeichen folgen
2. Auf eine Linke Klammer darf kein Leerzeichen folgen; vor einer rechten Klammer darf kein Leerzeichen stehen

Die Ordnungszahl muß immer eine ganze Zahl sein.

Ein Record umfasst 4095 Bytes. Ein 32-Byte langer Datensatz darf also nicht 128 mal hintereinander vorkommen. Indem man untergeordnete Records definiert, kann man dieses Hindernis umgehen. Auf diese Records kann man ebenfalls über Ordnungszahlen zugreifen.

Der REDEFINES-Befehl

Der REDEFINES-Befehl ermöglicht es, über verschiedene Namen auf Daten zuzugreifen. In der FILE SECTIOn darf die REDEFINES-Anweisung nicht bei der Ordnungszahl 01 vorkommen. Um einen ganzen Record neu zu definieren, muß lediglich sein Name im DATA RECORDS Feld vorhanden sein.

Ein File besteht aus einzelnen Records. Ein Drucker-File besteht aus einzelnen Zeilen, die jede einen Record von der Länge 132 Bytes darstellt.

Sollen Files auf Diskette geschrieben werden, so ist die Größe jedes Records auf 4095 Bytes beschränkt.

Die FILE-Sektion des DATA DIVISION-Felds besteht aus mehreren FD-Einträgen (File Description). Werden vom Programm drei Files bearbeitet, sind in der DATA DIVISION auch drei FD-Einträge vorhanden.

Files dürfen verschiedene Records enthalten. Dies geschieht mit dem DATA RECORDS-Befehl in dem FD-Feld :

```
FD DISK-FILE . . .
        DATA RECORDS ARE  NAME, VORNAME, ADRESSE.
```

WORKING-STORAGE-Sektion

Der zweite Teil des DATA-DIVISION-Teils ist der WORKING-STORAGE-Abschnitt. Hier werden Zähler und Konstanten definiert, die zum Erstellen des Programms nützlich sind. Das Format ähnelt der Definition der Datenfelder in dem FILE-SECTION-Abschnitt.
Jedes elementare Datum muß mit dem PICTURE-Befehl bestimmt weden. Neben der Ordnungszahl, dem Datenname und dem Format kann der Programmierer noch Anfangswerte bestimmen. Dies geschieht unter Verwendung des VALUE IS-Befehls. Die Werte werden als numerische oder nicht-numerische Variablen angegeben. Werte für numerische Felder (durch die Symbole 9 und V deklariert) werden als numerische Zahlen angegeben :

```
  100
  -65
2.50
```

Daten für nicht-numerische Felder (durch die Symbole A oder X deklariert) werden mit nicht-numerischen Zeichen angegeben und in Anführungszeichen geschrieben :

```
"   MEHRWERTSTEUER"
"REINGEWINN 1985"
```

Eine nicht-numerische Zeichenfolge kann bis zu 120 Zeichen enthalten.

Sind Daten für eine formatierte Ausgabe auf Drucker bestimmt, so können sie nicht für arithmetische Operationen verwendet werden. Um die Daten jederzeit bearbeiten zu können, sind sie in einem Feld enthalten, das keine formatierte Ausgabe erlaubt. Dieses Feld legt man im WORKING-STORAGE-Abschnitt fest.
(formatierte Ausgabe : z.B. eingefügter Dezimalpunkt oder Komma)

In das WORKING-STORAGE-Feld trägt man ausschließlich die Beschreibung der logischen Records ein, keinesfalls jedoch die Beschreibung von Files.

Ist es erforderlich, Datenfelder zu definieren, die nich in einer Datengruppe enthalten sind, kann dies mit Hilfe der Ordnungszahl 77 geschehen. Diese Ordnungszahl (LEVEL-Number) zeigt ein Feld an, das in keinerlei Verbindung zu den übrigen steht.
Diese einzelne Record-Beschreibung muß mit der Zahl 77 beginnen, mindestens einen Datennamen und einen PICTURE-Befehl enthalten.

PROCEDURE DISVION-Feld

In dem PROCEDURE-DIVISION-Feld, dem letzten Abschnitt eines COBOL Programms, werden die Bearbeitungsvorgänge mit den Befehlswörtern unter Berücksichtigung der Syntax formuliert. Verben und Worte bilden zusammen Sätze, die die grundlegende Einheit dieses Feldes sind. Ein oder mehrere Sätze bilden einen Abschnitt. Diese Programmabschnitte werden kombiniert und formen Sektionen. Wie es von den anderen COBOL-Programmteilen bekannt ist, müssen Abschnitte und Sektionen benannt werden.

So ist es zum Beispiel möglich, den Vorspann und Titel eines Programms als Abschnitt zu schreiben und diesen dann "VORSPANN" zu nennen. Soll nun der Vorspann ausgegeben werden, so gibt man an dieser Stelle einfach den Befehl : PERFORM VORSPANN

In der Programmiersprache COBOL werden die Ausdrücke, die Prozeduren beschreiben, Verben genannt.
Beispiel :

```
      OPEN  INPUT DATEN-FILE.
      READ  DATEN-FILE AT END CLOSE DATEN-FILE. STOP RUN.
      ADD   WERT  TO  GESAMTSUMME  GIVING
              GIVING  ERGEBNIS-DER-ADDITION.
```

In diesem Beispiel sind OPEN, READ und ADD Verben, die hier die Bearbeitung von Daten erledigen.

Die zur Verfügung stehenden Befehle sind also von den vorhanden Verben (ADD, GO TO, IF, MOVE, READ,etc...) abhängig, die der Compiler versteht. Jede Anweisung innerhalb einer Prozedur muß mit einem Verb beginnen; dabei hat jedes Verb eine eigene Syntax. Ferner sollte man nur eine Anweisung in eine Zeile schreiben.

Oft enthalten Ausdrücke ein Verb, auf das ein oder zwei Operanden folgen. Das Verb beschreibt die Bearbeitung der Daten, die Operanden geben die Daten an, die bearbeitet werden.

```
      SUBTRACT  SKONTO  FROM  GESAMT-BETRAG.
```

SUBTRACT ist das Verb und beschreibt die Bearbeitung, SKONTO und GESAMT-BETRAG sind die dazugehörigen Operanden.

Die minimale Form einer Anweisung besteht aus dem Verb und dessen Operanden.

```
   MOVE  KUNDEN-NAME  TO  DRUCKER-DATEI; GO TO  ENDABRECHNUNG.
```

Anweisungen können auch aus zwei oder mehr einzelnen Anweisungen bestehen. Die obige Zeile stellt drei Anweisunge dar (zwei Einzelanweisungen und Gesamtanweisung). Eine Trennung der beiden Einzelanweisungen durch Semikolon ist optional, jedoch erhöht es die Lesbarkeit eines Programms. Anstelle des Semikolons kann auch ein Komma benutzt werden.

Sätze

Ein Satz besteht aus einer Folge von einer oder mehreren Anweisungen; die letzte Anweisung wird durch einen Punkt abgeschlossen. Sätze werden untereinander geschrieben und bilden Prozeduren.

```
PROCEDURE DIVISION.
BEGINING.
      OPEN INPUT KUNDEN-TRANSAKTIONEN;
      OPEN OUTPUT KUNDEN-ÜBERWEISUNGEN.
      MOVE SPACES TO HEADING.
      READ KUNDEN-TRANSAKTIONEN; AT END STOP RUN.


0001 TRANSAKTION-AUSGABE.
0002       MOVE  BETRAG  TO  GESAMT-BETRAG.
0003       WRITE HEADING.
0004 DISK-FILE LESEN.
0005       READ  KUNDEN-TRANSAKTIONEN;
0006         AT END GO TO TRANSFER-DISK.
0007       GO TO  BETRAG-VERGLEICHEN.
```

Der Name eines Abschnitts steht immer im A-Feld (Spalte 6-9), die Sätze, die den Abschnitt bilden, stehen im B-Feld (Spalte 10-70). Im obigen Beispiel stehen die zwei Abschnitt-Namen im A-Feld, die Satz-Namen stehen im B-Feld.

Die Verwendung von Abschnitt-Namen bietet den Vorteil, daß eine Prozedur eine andere einfach durch ihren Namen aufrufen kann. Der Programmablauf kann verändert werden, indem man mit dem GO TO - Befehl von einer Prozedur zu einer anderen springt.

Syntax : GO TO Abschnitt-Name.

Anweisungen müssen mindestens aus einem Verb und den dazugehörigen Operanden bestehen, können aber auch aus mehreren Verben+ Operanden zusammengesetzt werden. Eine oder mehrere Anweisungen bilden einen Satz; Sätze wiederum formen einen Programmabschnitt. Darauf folgen die Sektionen, die aus mehreren Abschnitten bestehen.

Die verschiedenen Verb-Kategorien

Alle Verben, die man in der PROCEDURE-DIVISION-Sektion benutzen kann, zerfallen in drei funktionelle Kategorien : unbedingte Befehle, bedingte Befehle und Compiler-direktive Befehle.

ALTER gehört zu den unbedingten, IF zu den bedingten und COPY zu den Compiler-direktiven Verben. Der Großteil der Verben sind unbedingt (imperativ), d.h. sie geben eine direkte Befehlsanweisung. Weitere unbedingte Verben sind z.B. READ und ADD.

Bedingte Verben prüfen eine Bedingung, wie z.B. A=B, und führen einen bestimmten Befehl aus, wenn die Bedingung wahr/unwahr ist.

Compiler-direktive Verben beeinflussen den Compilationsvorgang. Zu dieser Kategorie gehören die Verben COPY und EXIT.

Die Anweisung ADD A TO B ist unbedingt. ADD A TO B; ON SIZE ERROR GO TO X-PROZEDUR und READ FILE-1; AT END GO TO END-ROUTINE sind beide bedingte Anweisungen. Außerdem ist jede Anweisung, die mit dem Befehl IF anfängt, bedingt.

IF A = B GO TO ADDITIONS-ROUTINE... .

Der Teil GO TO ... ist eine unbedingte Anweisung; durch den IF-Befehl wird der Satz aber zu einer bedingten Anweisung.

Noch einmal : Eine Anweisung kann aus mehreren Einzelanweisungen bestehen (s.o.). Wird eine oder mehrere Anweisungen durch einen Dezimalpunkt abgeschlossen, so nennt man dies einen Satz.

Die EIN- und AUSgabe Verben

Der Datenaustausch zwischen Hauptspeicher und Peripheriegeräten kann eine schwierige Sache sein. Die Sprache COBOL vereinfacht den Datentransfer, indem sie die Auswahl der Puffer, Zugriffskanäle und die Speicherverwaltung dem Compiler überläßt. Der Programmierer muß lediglich die EIN/AUSgabe-Anweisungen geben.

Bitte schlagen Sie an dieser Stelle unter dem OPEN-Befehl nach. Dieser Befehl ermöglicht die Bearbeitung von Ein- oder Ausgabe-Files. Denn bevor mit Hilfe des READ-Befehls Daten aus einem File gelesen werden können, müssen intern Standardroutinen ausgeführt werden; diese Aufgabe fällt dem OPEN-Befehl zu.

Das Gegenstück zu OPEN ist der CLOSE-Befehl. Dieses Verb beendet die Bearbeitung eines Files; erst nach dem CLOSE-Befehl sollte die Programmausführung durch den STOP RUN-Befehl beendet werden, andernfalls kann es zu Datenverlust kommen !

Um einen Datenrecord aus einem Eingabefile in den Hauptspeicher zu transferieren, benutzt man den READ-Befehl. Auch wenn das File mehrere Records enthält, ist immer nur einer auf einmal zugänglich. Liest man den Record aus einem File, so wird dieser in dem selben Speicherbereich abgelegt, wo auch der letzte Record gespeichert wurde. Das bedeutet, daß beim Lesen eines anderen Records der alte, im Speicher befindliche Record, gelöscht wird. Da in Programmen oft zu einer READ-Anweisung zurückgesprungen, muß dafür gesorgt werden, daß das Ende eines Files erkannt wird. Andernfalls erscheint eine Fehlermeldung und Datenverlust könnte die Folge sein. Um zu prüfen, ob das Ende eines Files erreicht ist, verwendet man den AT END-Befehl. Der AT END-Befehl muß in einem Programm unbedingt verwendet werden, auch wenn das Ende eines Records nie erreicht wird !

```
READ EINGABE-FILE; AT END IF BESTELLWERT IS GREATER THAN 500
THEN GO TO  PORTO-UND-VERPACKUNG.
```

Bei dieser Programmzeile würde der Compiler eine Fehlermeldung ausstoßen, da auf den AT END-Befehl auf keinen Fall eine bedingte Anweisung (wie IF...THEN) folgen darf.

Der WRITE-Befehl dient dazu, Daten extern zu speichern.
Er schreibt jeweils einen Record auf das angesprochene Peripheriegerät. Möchte man den Record BANK-KONTO speichern, so gibt man ein :

```
WRITE  BANK-KONTO.
```

Bitte vergleichen Sie die READ und WRITE-Befehle. Beim READ-Befehl handelt es sich um den Filenamen, beim WRITE-Befehl dagegen um den Record-Namen !

Das MOVE-Verb

Häufig müssen Daten von einem Speicherbereich in einen anderen transferiert werden, z.B. Daten aus einer Eingabe-Datei von Diskette in eine Ausgabe-Datei für den Drucker. Diese Aufgabe übernimmt der MOVE-Befehl.

Das Format des MOVE-Befehls können Sie dem Handbuch entnehmen. Der MOVE-Befehl bezieht sich auf ein "sendendes" Feld und ein "empfangendes" Feld. Wenn man beispielsweise Daten aus dem Feld AUFTRAG-GEBER in das Feld LIEFERADRESSE bringen möchte, so lautet der Befehl :

```
MOVE AUFTRAG-GEBER TO LIEFERADRESSE.
```

Will man die Daten aus AUFTRAG-GEBER in zwei verschiedene Felder bringen :

```
MOVE AUFTRAG-GEBER TO LIEFERADRESSE,
                      RECHNUNGS-ADRESSE.
```

Werden Daten aus einem bestimmten Bereich transferiert, so wird dieser Bereich nicht verändert oder gelöscht. Nach dem MOVE-Befehl sind die Daten mindestens zweimal im Hauptspeicher vorhanden (im sendenden und empfangenden Feld). Wie sich an den Beispielen erkennen läßt, sind die Operanden des MOVE-Befehls Variablen, die entweder einen numerischen oder nicht-numerischen Inhalt besitzen. So ist die Zahl 1985 eine numerische Variable, "1985" jedoch wäre wegen der Anführungszeichen eine nicht-numerische Variable. Die Verwendung von Variablen bringt den Vorteil mit sich, daß die veränderten Daten nicht unbedingt bekannt sein müssen. Der jeweilige Variablennamen bezieht sich auf eine Speicheraddresse, wo die Daten abgelegt sind. Wird nun dieser Variablennamen verwendet, so regelt der Computer den Zugriff auf den Speicher.

Variablennamen (Datennamen) müssen mindestens ein alphabetisches Zeichen beinhalten, damit der Compiler zwischen normalen Zahlen und Variablennamen unterscheidet. Die Zahl 40960 wird nicht als Variablennamen anerkannt, da sie kein alphabetisches Zeichen hat (alphabetische Zeichen : A-Z).

Um zwischen nicht-numerische Variablen (Textstrings) und Datennamen zu unterscheiden, werden die Textstrings durch Anführungszeichen eingegrenzt.

Ein Datennamen wie EINZEL PREIS ist wegen des Leerzeichens nicht gültig. "EINZEL PREIS" dagegen ist ein gültiger Textstring. Textstrings kann man durch jedes beliebige Zeichen darstellen, numerische Daten jedoch nur durch die Zahlen 0-9 und die Vorzeichen "+" und "-". Wird bei numerischen Daten ein Plus- oder Minuszeichen benötigt, so steht es ganz links. Enthalten numerische Daten keinen Dezimalpunkt, so werden sie als Integerzahlen behandelt.

Ein Textstring kann aus bis zu 120 Zeichen bestehen, numerische Daten dagegen nur aus 18 Ziffern.

Soll ein Datenfeld mit einem Konstanten Wert gefüllt werden, so benutzt man symbolische Konstanten. Bestimmte Datennamen belegt man mit wichtigen Konstanten. Im Programm kann man nun über den Datennamen auf diese Konstanten zugreifen. Bei der Verwendung von symbolischen Konstanten wird das empfangende Datenfeld mit dem konstanten Wert gefüllt, unabhängig von der Länge.
Möchte man das Datenfeld KUNDEN-ADRESSE löschen, so lautet die symbolische Konstante SPACES :

```
MOVE  SPACES  TO  KUNDEN-ADRESSE.
```

Ist es erforderlich, das Feld KUNDEN-ADRESSE mit dem Buchstaben X zu füllen, so lautet die Programmzeile :

```
MOVE ALL "X" TO  KUNDEN-ADRESSE.
```

Falls das sendende und das empfangende Feld beide numerisch sind, so ist der MOVE-Befehl ebenfalls numerisch. Sind das sendende und das empfangende Feld nicht-numerisch, so handelt es sich um einen nicht-numerischen MOVE-Befehl. Sollte eines der beiden Felder nicht-numerisch sein, ist der MOVE-Befehl auf jeden Fall auch nicht-numerisch.

Jedoch kann die Situation auftreten, daß Sende- und Empfangsfeld nicht gleich groß sind. In diesem Fall hängt das Resultat von dem jeweiligen Format der Daten ab. Im folgenden Beispiel wird ein nicht-numerisches MOVE durchgeführt :

[FRITZ MEIER] [FRITZ MEIER]

Sollte das empfangende Feld länger als die gesendeten Daten sein, so werden die unbenutzten Positionen mit Leerzeichen ausgefüllt.

Ein numerischer MOVE-Befehl ist vergleichbar mit einem nicht-numerischen MOVE, mit der Ausnahme, daß der Dezimalpunkt gesetzt werden muß. Jede Zahl hat einen Dezimalpunkt; wird dieser jedoch nicht angezeigt, so handelt es sich um eine ganze Zahl (Dezimalpunkt steht an letzter Stelle).

Wird das Symbol "V" verwendet, so bedeutet dies einen scheinbaren Dezimalpunkt, d.h. er ist im Datenfeld nicht vorhanden, dient jedoch zum richtigen erfassen der empfangenen Daten.

Hier nun einige Beispiele, wie Dezimalbrüche richtig erfasst werden. Dabei müssen die Zahlen in beiden Feldern so dargestellt werden :

sendendes Feld		empfangendes Feld
echter Dezimalpunkt	---->	echter Dezimalpunkt
scheinbarer Dezimalpunkt (V-Symbol)	---->	scheinbarer oder echter Dezimalpunkt

Beispiele :

[123.45]	[0123.450]
[123v45]	[0123.450]
[123v45]	[0123v4500]

Ein numerisches MOVE besteht aus drei Schritten :
1. Erkennen der Dezimalpunkte
2. Übertragen der Ziffern
3. unbenutzten Positionen mit Nullen füllen

Sollten die gesendeten Daten länger als das empfangende Feld sein, so werden Teile abgetrennt !
Beispiel :

[EDV/DATENTECHNIK MÜLLER] wird zu : [EDV/DATENTECHN]

Bei nicht-numerischen MOVE-Befehlen wird jeweils der rechte Teil der gesendeten Daten abgeschnitten.

[1234.56] wird zu : [34.5]

Bei numerischen MOVE-Befehlen werden je nach Position des Dezimalpunkts auf beiden Seiten Daten abgeschnitten.

Zu beachten ist, daß mit dem MOVE-Befehl sowohl ganze Datenfelder als auch elementare Einträge transferiert werden können. So kann man sich einige Programmierarbeit sparen, denn hat man elementare Einträge eines Records bearbeitet, so kann man sie alle mit einem einzigen MOVE wieder abspeichern :

```
MOVE  KUNDEN-DATEI  TO ... .
```

Die größte Datengruppe ist der jeweilige Recordname. Beim Bearbeiten eines Records ist das Format des eingelesenen und geschriebenen Records gleich. Hat man einen Record überarbeitet, so schreibt man ihn in das Feld des auszugebenen Records :

MOVE (Name des eingelesenen Records) TO (Name des auszugebenen Records).

Hier sind alle MOVEs nicht-numerischer Art.

Die arithmetischen Verben

Die meisten arithmetischen Funktionen bei geschäftlichen oder wirtschaftlichen Anwendungen beschränken sich auf Addition, Subtraktion, Multiplikation und Division. Für diese vier Rechenarten gibt es in COBOL die Befehle : ADD, SUBTRACT, MULTIPLY und DIVIDE.

Möchte man den Wert 5 zum Feld PORTO-KOSTEN addieren, würde dies lauten :

```
ADD  5  TO  PORTO-KOSTEN.
```

Schlagen Sie bitte im Handbuch unter dem Befehl ADD nach. Dort ist vermerkt, daß die Ausdrücke TO und GIVING kombiniert werden können. Geschieht dies, so wird die Summe im Datennamen gespeichert, der auf GIVING folgt.

```
        EINGABE                    AUSGABE
HUNDE-STEUER [3029v73]        GESAMT-BETRAG [9999v99]
AUTO-STEUER  [7637v69]
```

Die Felder auf der linken Seite (EINGABE) stellen normale Operanden dar, wie sie beim ADDitions-Befehl benutzt werden. Auf der rechten Seite steht das Empfangs-Feld. Auf diese Felder kann man mit dem Datennamen zugreifen. Das Format der Felder definiert man in dem DATA DIVISION-Abschnitt. Addiert man nun HUNDE-STEUER und AUTO-STEUER, so erhält man 10667v42. Wollte man dieses Resultat im Feld GESAMT-BETRAG speichern, so erhält man die Fehlermeldung SIZE ERROR, weil das Empfangs-Feld zu klein ist. Bitte achten Sie beim Erstellen eines Programms darauf, daß die Felder ausreichend groß definiert sind !

```
ANZAHL [0005023v7]      RESULTAT [9999v99]
QUOTE        [v042]
```

Versucht man nun, ANZAHL und QUOTE zu multiplizieren und das Ergebnis in RESULTAT zu speichern, erhält man wiederum die Fehlermeldung SIZE ERROR, da die Stellen rechts nicht abgespeichert werden.

Sollen Zahlen gespeichert werden und ist das empfangende Feld zu klein, so werden auf der linken und/oder rechten Seite Daten abgeschnitten. Danach gelten folgende Regeln :

1. Enthält das empfangende Feld weniger Integer-Stellen, so werden auf der linken Seite Daten abgetrennt.
2. Enthält das empfangende Feld weniger Dezimal-Stellen, so werden auf der rechten Seite Daten abgetrennt.

Erhält man einen SIZE ERROR, so werden verfälschte Zahlen abgespeichert. Die SIZE ERROR-Option zeigt an, ob Integer-Stellen abgeschnitten werden. Der Programmierer muß außerdem angeben, wie Summen abgespeichert werden und das Abtrennen von Stellen verhindern. So sollte nach einer Addition eine Routine aufgerufen werden, die das Resultat auf Richtigkeit überprüft.

Um das Verfälschen von Daten zu verhindern, die beim Abtrennen von Stellen auftritt, gibt es die ROUNDED-Option. Die ROUNDED-Option besorgt automatisch das Abspeichern des richtigen Resultats.

Formatierte Darstellung

Um Zahlen formatiert auf Drucker auszugeben, kann man einige Zeichen einfügen, insbesondere Dollarzeichen, Komma oder Dezimalpunkt (sofern noch nicht enthalten).

```
EINGABE                          AUSGABE (Drucker)
KONTO-STAND [      v]            GUTHABEN [$     .  ]
AUSGABEN     [     v ]
```

Durch Einsetzen der gewünschten Zeichen (Punkt,Komma od. Dollarzeichen) in das empfangende Feld können die Daten formatiert werden. Bei der Ausgabe von GUTHABEN auf Drucker wird so ein Dollarzeichen und der Dezimalpunkt dargestellt.

Numerische Variable

```
ADD "29" TO LINE-COUNT.
```

Diese Programmzeile ist ungültig, da "29" ein nicht-numerischer Datentyp, nämlich ein Textstring, ist.

```
ADD 41.86 TO X-FELD.
```

Diese Anweisung ist gültig, da 41.86 numerisch ist. Bevor die Addition stattfindet, wird die Position des Dezimalpunkts im X-FELD festgestellt.
Bitte schlagen Sie nun den SUBTRACT-Befehl nach. Die Syntax des SUBTRACT-Befehls ist ähnlich der des ADD-Befehls. Beim ADD-Befehl ist das Wort "TO" optional, wogegen bei SUBTRACT das Wort "FROM" immer erforderlich ist.

```
EINKOMMEN - STEUERN = KONTO-STAND
```

Um diese Rechenoperation auszuführen, muß man eingeben :

```
SUBTRACT  STEUERN FROM EINKOMMEN  GIVING  KONTO-STAND.
```

Bitte schlagen Sie nun den MULTIPLY-Befehl nach. Das Wort "BY" ist immer erforderlich. Zu beachten ist, daß der Befehl nur zwei Operanden bearbeitet, nämlich den Multiplikator (Datenname-1) und den Multiplikanden (Datennamen-2).

Die beiden folgenden Zeilen liefern das gleiche Ergebnis :

```
MULTIPLY  A BY B  GIVING  ERGEBNIS.
MULTIPLY  B BY A  GIVING  ERGEBNIS.
```

Der letzte Operand (ERGEBNIS) muß immer ein Datenname sein, da es das Empfangs-Feld ist. Um das Programm übersichtlicher zu gestalten, sollte man anstelle von MULTIPLY FELD BY 2 GIVING EK besser MULTIPLY 2 BY FELD GIVING EK schreiben.

Bitte schlagen Sie nun den DIVIDE-Befehl nach. Die Syntax ist dieselbe wie bei der MULTIPLY-Anweisung. Die Anzahl der berechneten Dezimalstellen hängt von dem empfangenden Feld ab. Sind im Feld drei Dezimalstellen vorgesehen, so wird das Ergebnis nur auf drei Dezimalstellen berechnet.

Sobald die Division alle verfügbaren Dezimalstellen im Feld gefüllt hat, endet die Berechnung. Dies bedeutet, daß alle weiteren Dezimalstellen unberechnet bleiben (abgetrennt werden). Auf der linken Seite werden die Zahlen ebenfalls verfälscht, sollte das Feld kleiner als die empfangenen Daten sein.

Versucht man durch Null zu dividieren, so erhält man einen SIZE ERROR, da das Ergebnis nicht definiert ist. Die Fehlermeldung kann man durch die ON SIZE ERROR-Option abfangen.

Die ROUNDED-Option verhindert ein rechtsseitiges Abschneiden der Zahlen. Da dies bei einer Division nicht passieren kann, sind die Regeln für die ROUNDED-Option unterschiedlich. Wird diese Option gewählt, so werden auf der rechten Seite zwei Stellen hinzugefügt, die dann normalerweise abgerundet werden.

Sprungbefehle

Startet man ein COBOL-Programm, so beginnt die Ausführung mit dem ersten Befehl der PROCEDURE DIVISION. Alle weiteren Anweisungen im Programm werden der Reihe nach ausgeführt. Natürlich ist es auch möglich, diese Reihenfolge zu verlassen und die Kontrolle an andere Programmteile zu übergeben. Dazu existieren in COBOL zwei Sprungbefehle, nämlich der GO TO- und der PERFORM-Befehl. Der Unterschied zwischen diesen beiden Befehlen ist, daß die GO TO-Anweisung die Kontrolle an ein anderes Programmteil übergibt, die PERFORM-Anweisung jedoch nur ein Unterprogramm (Subroutine) aufruft und nach dessen Ausführung wieder an die Stelle zurückspringt, von wo der der Aufruf erfolgte.

Die COBOL-Anweisungen GO TO und PERFORM sind vergleichbar mit den BASIC-Befehlen GOTO und GOSUB.

Bitte schlagen Sie unter dem GO TO-Befehl nach. In einem Programm werden prozedurale Anweisungen in der Reihenfolge ausgeführt, in der sie eingegeben wurden. Der GO TO-Befehl erlaubt es, diese Reihenfolge zu verlassen und die Programmausführung an eine andere Prozedur zu übergeben. Die Befehlssyntax lautet :

```
GO TO  Prozedurenname.
```

Steht in einem Programm die Anweisung GO TO ANFANG., so wird in dem gesamten PROCEDURE DIVISION-Teil nach einem Abschnitt oder einer Sektion mit dem Namen ANFANG gesucht. Dann wird die Kontrolle an diesen Programmteil übergeben.

Der zweite Sprungbefehl ist die PERFORM-Anweisung. PERFORM übergibt nur zeitweise die Kontrolle an ein Unterprogramm.

Am Ende eines Unterprogramms muß immer die EXIT-Anweisung stehen. Auf keinen Fall darf ein Unterprogramm mehr als eine EXIT-Anweisung enthalten.

```
UNTERPROGRAMM  SECTION.
 ABSCHNITT-A.
        MOVE  EINGABE-DATEI  TO  AUSGABE-DATEI.
        WRITE  AUSGABE-DATEI.
        IF  FLAG = 1 THEN GO TO  ABSCHNITT-B.
        WRITE  AUSGABE-DATEI.
 ABSCHNITT-B.  EXIT.
```

Der ABSCHNITT-B des Unterprogramms enthält nur die Anweisung EXIT. Sind in einer Prozedur mehrere Programmpfade möglich, so lautet die letzte Anweisung meistens EXIT.
Da EXIT in einem Programmteil stehen muß, ist es erforderlich, daß auch die Prozeduren, die ausgeführt werden sollen, auch in mindestens einem Teil zusammengefasst werden. Die PERFORM-Anweisung bezieht sich auf den Abschnitt-A und den Abschnitt-B. Daher können beide Abschnitte zusammen eine Sektion bilden. Unter dem Sektions-Namen kann der PERFORM-Befehl nun beide Abschnitte aufrufen. Es ist darauf zu achten, daß die Sektion immer aus zwei Teilen besteht: im Ersten stehen die Prozeduren, im Zweiten die EXIT-Anweisung. Will man keine Sektionen verwenden, so lautet die Anweisung :

```
PERFORM ABSCHNITT-A THRU ABSCHNITT-B.
```

Will man alle Abschnitte von STUNDEN-LOHN bis WOCHEN-LOHN viermal ausführen :

```
PERFORM STUNDEN-LOHN THRU WOCHENLOHN  4 TIMES.
```

Die IF-Anweisung

In einem Programm ist es wichtig, abhängig von Resultaten verschiedene Befehle ausführen zu lassen. So kann man beispielsweise Tastatureingaben des Anwenders auf Richtigkeit überprüfen und gegebenenfalls wiederholen.

Lesen Sie bitte im Handbuch die IF-Anweisung nach.
Es gibt zwei Formen bedingter Anweisungen. Zur ersten Form zählt die IF-Anweisung, zur zweiten Form gehören Anweisungen wie AT END und ON SIZE ERROR. Beachten Sie das Format der IF-Anweisung.
Nach der Bedingung folgt entweder eine Anweisung oder NEXT SENTENCE.

```
IF BESTELL-WERT IS NOT LESS THAN 500 PERFORM PORTO-ROUTINE
      ELSE PERFORM  AUSGABE-ROUTINE.
```

Falls der Bestellwert kleiner als 500 ist, wird die Ausgabe-Routine aufgerufen. Ansonsten wird die Porto-Routine ausgeführt. Soll die nächste Programmzeile ausgeführt werden, so verwendet man die Schlüsselwörter NEXT SENTENCE. Diese Anweisung kann direkt auf die Bedingung folgen. Sollte NEXT SENTENCE nach ELSE stehen, so kann der ganze Ausdruck ELSE NEXT SENTENCE weggelassen werden :

```
      IF KONTO-STAND IS POSITIVE PERFORM ZINS-BERECHNUNG.
      WRITE KONTO-STAND.
```

Sollte der Kontostand nicht positiv sein, so wird automatisch in die nächste Zeile gesprungen.

```
0001F SYNTAX FEHLER
0002F KEIN COBOL WORT
0003F SYNTAX FEHLER ODER FEHLENDER PUNKT IN DER
        VORHERIGEN ZEILE
0004F FILE IN DER INPUT-OUTPUT SEKTION NICHT DEFINIERT
0005F OCCURS IST AUF EIN LEVEL BESCHRÄNKT
0006F AUF DIMENSIONIERTE FELDER KANN DIE REDEFINE-ANWEISUNG NICHT
      ANGEWENDET WERDEN
0007F PICTURE DATENFELDER MÜSSEN ELEMENTAR SEIN
0008F PICTURE ANWEISUNG MIT EDITIERZEICHEN ENTHÄLT
        UNERLAUBTE KOMBINATIONEN
0009F MAXIMALE RECORDLÄNGE VON 4095 BYTES WURDE ÜBERSCHRITTEN
0010F ELEMENTARES FELD HAT KEINE PICTURE ANWEISUNG
0011F UNERLAUBTE REDEFINE-ANWEISUNG
0012F FEHLER BEIM DIMENSIONIEREN
0013F UNERLAUBTE KOMBINATION VON ZEICHEN IN
        DER PICTURE-ANWEISUNG
0014F DOPPELTE VERWENDUNG DES NAMEN IST NICHT ERLAUBT
0015F ENVIRONMENT DIVISION FEHLT
0016F FD MUSS LABEL RECORD ANWEISUNG ENTHALTEN
0017F VALUE OF FILE-ID FEHLT
0018F DIMENSIONIERTE VARIABLE ENTHÄLT UNERLAUBTES ZEICHEN
0019F FALSCHE ANWENDUNG VON USAGE-ANWEISUNG
0020F OCCURS BEFEHL IST BEI LEVEL 01 NICHT ERLAUBT
0021F VALUE IST ZUSAMMEN MIT OCCURS NICHT ERLAUBT
0022F VALUE IST FÜR REDEFINIERTE DATENFELDER NICHT ERLAUBT
0023F UNERLAUBTES ZEICHEN IN WORT
0024F RELATIVER SCHLÜSSEL FEHLT
0025F MUSS IN WORKING STORAGE-SEKTION STEHEN
0026F SCHLÜSSEL NICHT ELEMENTAR
0027F RELATIVER SCHLÜSSEL HAT FALSCHE GRÖSSE  PICTURE 9(7) MAXIMAL
0028F NAME DES ABSCHNITTS IST NICHT DEFINIERT
0029F NAME DES ABSCHNITTS IST NICHT ÄNDERBAR
0030F ZU VIELE FILES ANGEWÄHLT  MAXIMALE ZAHL IST 12
0031F ZU WENIG SPEICHER VERFÜGBAR   ANZAHL DER VARIABLEN REDUZIEREN
0032  LETZTE ZEILE DIE GELESEN WURDE
      F=> FATAL'.  OBJEKT FILE IST NICHT LAUFFÄHIG !!!
                BITTE VERBESSERN
      UND NOCHMALS COMPILIEREN.
0033F FEHLENDE DIVISION ANWEISUNG
0034F ZU WENIG SPEICHER VERFÜGBAR   (ZU VIELE ABSCHNITT-NAMEN)
0035F HÄUFIG BENUTZTE ABSCHNITTE AN PROGRAMMANFANG SETZEN
0036F TEMP ÜBERLAUF
0037F LEVELS 01-49 UND 77 NUR ERLAUBT
0038F NICHT DEFINIERT
0039F " FEHLT IN SPALTE 10 FÜR WEITERGEFÜHRTEN ZEICHENSTRING
0040F UNERLAUBTES HEXADEZIMALES ZEICHEN (0-9 UND A-F)
0041F UNERLAUBTE FILE-ID  "U:FILE.TYP"
0042F ASCII (DISPLAY) DATENTYP WIRD BENÖTIGT
0043F RANDOM FILES MÜSSEN  INVALID KEY-ANWEISUNG BENUTZEN
0044F OPEN I-O MUSS GLEICH SELECT I-O SEIN
0045F VALUE/PICTURE : VORZEICHEN FEHLER
0046F WIRD EIN TEXT DURCH COPY EINGEFÜGT, SO DARF ER KEINE
        WEITERE COPY-ANWEISUNG ENTHALTEN
0047F COPY FILE NAME DARF HÖCHSTENS 10 ZEICHEN LANG SEIN
0048F KANN REDEFINIERTEN DATENNAME NICHT FINDEN
0049F STRING IST LÄNGER ALS 120 ZEICHEN
0050W STRING WIRD AN DER RECHTEN SEITE ABGESCHNITTEN
0051W WORT BESTEHT AUS MEHR ALS 30 ZEICHEN
0052F STRING LÄNGER ALS IN PICTURE-ANWEISUNG DEFINIERT
```

```
0053W REDEFINIERTER BEREICH ANGEPASST
0054W EDITIERTER PICTURE-BEFEHL MODIFIZIERT
0055W ZWEI RECORDS IN EINEM FILE HABEN
        VERSCHIEDENE LÄNGEN
0056W SPALTE 5  WIRD ALS KOMMENTAR INTERPRETIERT
0057W ZEILEN FALSCH DURCHNUMMERIERT
0058W RANDOM FILE KANN NICHT MIT BEGRENZER BEARBEITET WERDEN
0059W PUNKT FEHLT HINTER DEM VORHERIGEN WORT
0060F DEZIMAL-PUNKTE FALSCH GESETZT
0061W DRUCKER KANN NICHT MIT BEGRENZER BEARBEITET WERDEN
0062F WERT ÜBERSCHREITET 5 ZIFFERN FÜR COMP-TYP
0063F UNERLAUBTER WERT FÜR COMP-TYP
0064F UNERLAUBTES WÄHRUNGSZEICHEN
0065F COPY MUSS LAUTEN  U:FILE-NAME.  U=LAUFWERKANGABE
0066W ALL-STRING IST AUF EIN BYTE BEGRENZT
0067F ZERO FEHLT BEI  BLANK WHEN ZERO
0068F BLANK WHEN ZERO  NICHT FÜR DATENGRUPPEN ERLAUBT
0069F BLANK WHEN ZERO  MUSS  ASCII/DISPLAY  SEIN
0070F BLANK WHEN ZERO  NUR FÜR NUMERISCHEN AUSDRUCK
0071F JUSTIFIED-ANWEISUNG FORDERT ELEMENTARES DATENFELD
0072F JUSTIFIED-ANWEISUNG KANN NICHT AUF NUMERISCHES
        ODER EDITIERTES DATENFELD ANGEWENDET WERDEN
0073W SPEICHERADRESSE ÜBERSCHREITET AKTUELLE CP/M-ANFANGSADRESSE
0074F MEHR ALS 255 DATENNAMEN IN DER LINKAGE SEKTION
0075F PD USING WITH NO LINKAGE SECTION
0076F IF/UNTIL BEDINGUNGEN SIND NICHT ERLAUBT
0077F DAS WORT  SENTENCE  FEHLT ODER FALSCH EINGEGEBEN
0078F NUR BEI DRUCKER-FILES KANN 'OMITTED' ANGEGEBEN WERDEN
0079F DOPPELTE LABEL ANWEISUNG
0080FEHLERFREIE COMPILATION. HERZLICHEN GLÜCKWUNSCH !!!
    *   NIEDRIGSTE ADRESSE DES OBJEKT-MODULS
    *   VERFÜGBARER SPEICHER (KANN VON DER FOLGENDEN ADRESSE
         SUBTRAHIERT WERDEN FÜR  MEMORY SIZE-BEFEHL)
    *   HÖCHSTE ADRESSE DES OBJEKT-MODULS
0081F NICHT GENÜGEND SPEICHER FÜR OBJEKT-PROGRAMM
        BITTE VERBESSERN UND NOCHMALS COMPILIEREN
0082F ADVANCING  NUR FÜR DRUCKER-FILES
0083F REDEFINES BEI LEVEL 01 IN DER FILE SECTION IST UNERLAUBT
0084F COMP UND COMP-3 KÖNNEN KEINE EDITIER-ZEICHEN ENTHALTEN
0085<=PROGRAMM-NAME
0086KEYED   C SEQ  FEHLER LVL FEHLER-MELDUNGEN
0087ZEILE #     #  SPALTE  #
0088BITTE GEBEN SIE RETURN FÜR DIE NÄCHSTE SEITE EIN
0089 :FEHLER-MELDUNG NACH ZEILE :
0090 90 KEINE WEITEREN INFORMATIONEN
0091 91 FEHLER BEIM VERGRÖSSERN DES FILES
0092 92 DISKETTE IST VOLL
0093 93 FILE NICHT WURDE NICHT GEÖFFNET
0094 94 KEIN PLATZ IM DIRECTORY VERFÜGBAR - DISKETTE IST VOLL
0095 95 FILE NICHT GEFUNDEN
0096 96 FILE WURDE SCHON GEÖFFNET
0097 97 DATEN BEI WAHLFREIEM ZUGRIFF NICHT VORHANDEN
0098 98 REWRITE OHNE VORHERIGES READ IM I-O MODUS
0099 99 ES WURDE VERSUCHT, EIN AUSGABE-FILE ZU LESEN ODER
        IN EIN EINGABE-FILE ZU SCHREIBEN
0100FEHLERMELDUNG NICHT IN TABELLE  W4.COM ODER W5.CBL FEHLERHAFT
     BITTE ERSTELLEN SIE EINE NEUE KOPIE DER ORIGINAL-DISKETTE
0101 FEHLERHAFT DIMENSIONIERT - VARIABLEN ÜBERSCHREITEN 64 KBYTES
0102 RECORD ÜBERSCHRITTEN.
0103F IDENTIFICATION? WORT FEHLT ODER FALSCH EINGEGEBEN
0104F DIVISION? WORT FEHLT ODER FALSCH EINGEGEBEN
```

```
0105F PROGRAM-ID? WORT FEHLT ODER FALSCH EINGEGEBEN
0106F CONFIGURATION, INPUT-OUTPUT ODER DATA DIVISION?
        WORT FEHLT ODER FALSCH EINGEGEBEN
0107F SECTION? WORT FEHLT ODER FALSCH EINGEGEBEN
0108F MODE? WORT FEHLT ODER FALSCH EINGEGEBEN
0109F ASCII? WORT FEHLT ODER FALSCH EINGEGEBEN
0110F BEGINNING? WORT FEHLT ODER FALSCH EINGEGEBEN
0111F ENDING? WORT FEHLT ODER FALSCH EINGEGEBEN
0112F IS? WORT FEHLT ODER FALSCH EINGEGEBEN
0113F COMMA? WORT FEHLT ODER FALSCH EINGEGEBEN
0114F ASSIGN? WORT FEHLT ODER FALSCH EINGEGEBEN
0115F DISK ODER PRINTER? WORT FEHLT ODER FALSCH EINGEGEBEN
0116F DELIMITER? WORT FEHLT ODER FALSCH EINGEGEBEN
0117F STANDARD? WORT FEHLT ODER FALSCH EINGEGEBEN
0118F PROCEDURE? WORT FEHLT ODER FALSCH EINGEGEBEN ODER SYNTAX FEHLER
0119F RECORDS? WORT FEHLT ODER FALSCH EINGEGEBEN
0120F OF? WORT FEHLT ODER FALSCH EINGEGEBEN
0121F FILE-ID? WORT FEHLT ODER FALSCH EINGEGEBEN
0122F RECORD? WORT FEHLT ODER FALSCH EINGEGEBEN
0123F OMITTED? WORT FEHLT ODER FALSCH EINGEGEBEN
0124F UNGÜLTIGE LEVEL-NUMMER. NUR  1-49 UND 77 ERLAUBT.
          LEVEL 66 ODER 88 NICHT GÜLTIG.
0125F UNGÜLTIGER FILE-NAME ODER STRING
0126F MUSS GÜLTIGES WORT SEIN.
0127F FILE-NAME FEHLT ODER FALSCH EINGEGEBEN.
0128F PUNKT FEHLT.
0129F NUMERISCHE GANZE ZAHL FEHLT.
0130F WORT FALSCH GESCHRIEBEN ODER ABSCHNITT NICHT IM A-FELD.
0131F UNGÜLTIGER ABSCHNITT ODER SEKTIONS-NAME.
0132F PROGRAM? WORT FEHLT ODER FALSCH EINGEGEBEN
0133F UNGÜLTIGE BEDINGUNG.
0134F TO? WORT FEHLT ODER FALSCH EINGEGEBEN.
0135F PROCEED? WORT FEHLT ODER FALSCH EINGEGEBEN.
0136F UNGÜLTIGER PROCEDUREN-NAME.
0137F UNGÜLTIGER VARIABLEN-NAME.
0138F INPUT, OUTPUT ODER I-O FEHLT ODER FALSCH EINGEGEBEN.
0139F TIMES? WORT FEHLT ODER FALSCH EINGEGEBEN.
0140F END? WORT FEHLT ODER FALSCH EINGEGEBEN.
0141F UNGÜLTIGER DATEN-NAME ODER STRING.
0142F UNGÜLTIGER DATEN-NAME.
0143F UNGÜLTIGER NUMERISCHER DATEN-NAME ODER STRING.
0144F UNGÜLTIGER NUMERISCHER DATEN-NAME.
0145F ON? WORT FEHLT ODER FALSCH EINGEGEBEN.
0146F SIZE? WORT FEHLT ODER FALSCH EINGEGEBEN.
0147F ERROR? WORT FEHLT ODER FALSCH EINGEGEBEN.
0148F UNGÜLTIGER RECORD-NAME.
          REMEMBER: WRITE A RECORD  AND READ A FILE.
0149F UNGÜLTIGER DATEN-NAME ODER LEVEL.
0150F UNGÜLTIGER DATEN-NAME FÜR LINKAGE-FELD.
0151F FOR? WORT FEHLT ODER FALSCH EINGEGEBEN.
0152W ÜBERPRÜFUNG DES SPEICHERS AN DIESER ADRESSE NICHT ERFOLGREICH
0153                <=KOPIERTES QUELLPROGRAMM ENTHÄLT FOLGENDEN FEHLER
0154F EXIT MUSS ALLEINE IN EINEM ABSCHNITT STEHEN.
0155F AUF DIE VORHERIGE FD-ANWEISUNG MUSS EIN 01 RECORD
        DIREKT FOLGEN.
0156F FILE NAME FALSCH EINGEGEBEN ODER PUNKT FEHLT.
0157F FILE KANN NUR AUF EINE ART BEARBEITET WERDEN
          SEQUENTIELL ODER WAHLFREI (RANDOM FILE).
0158F RANDOM FILES BENÖTIGEN RELATIVEN SCHLÜSSEL IN
          DER SELECT-ANWEISUNG.
```

0159F SEQUENTIELLE BEARBEITUNG HAT KEINEN RELATIVEN
SCHLÜSSEL IN DER SELECT-ANWEISUNG.
0160F SOURCE-COMPUTER, OBJECT-COMPUTER ODER SPECIAL-NAMES
WORT FEHLT ODER FALSCH EINGEGEBEN
0161F DATA DIVISION? WORT FEHLT ODER FALSCH EINGEGEBEN
0162W LINE ÜBERSCHREITET SPALTE 70/72, WELCHE NUR FÜR KOMMENTARE
GEDACHT SIND
DATEN WERDEN ALS KOMMENTAR BEHANDELT. DIES KÖNNTE NOCH
EINMAL VORKOMMEN. BITTE ÜBERPRÜFEN. DA DIESE MELDUNG NICHT
WIEDERHOLT WIRD.
0163F WRITE-BEFEHL FEHLT ODER FALSCH EINGEGEBEN.
0164F NACH ADVANCING "INTEGER", "DATA-NAME" ODER "PAGE"
FEHLEND ODER FALSCH EINGEGEBEN.
0165F DAS WORT `FROM` FEHLT ODER IST FALSCH EINGEGEBEN.
0166F DAS WORT `BY` FEHLT ODER IST FALSCH EINGEGEBEN.
0167F DIE WÖRTER `INTO` ODER `BY` FEHLEN ODER SIND FALSCH EINGEGEBEN.
0168F MULTIPLIZIEREN MIT STRINGS IST OHNE GIVING UNERLAUBT.
0169

Stichwortverzeichnis

Abschnitt :

Ein Programmteil kann zu einem Abschnitt zusammengefasst werden. Dies hat den Vorteil, daß er durch Eingabe seines Namens jederzeit aufrufbar ist. Der jeweilige Abschnitt-Name steht am Anfang des Abschnitts.

A-Feld :

Umfasst die Spalten 6-9 im COBOL-Quelltext. Dort stehen die Sektions- und Abschnittnamen des Programms.

Alphabetisches Zeichen :

Der alphabetische Zeichensatz enthält die Zeichen A..Z und das Leerzeichen (Space).

Alphanumerisches Zeichen :

Ein alphanumerisches Zeichen kann ein beliebiges Zeichen aus dem verfügbaren Zeichensatz des Computers sein.

Arithmetische Operatoren :

Ein arithmetischer Operator ist ein einzelnes Zeichen oder eine Kombination aus zwei Zeichen aus dem folgenden Satz :

Zeichen	Funktion
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
**	Exponentialfunktion

Aufrufendes Programm :

Durch den CALL-Befehl kann das Hauptprogramm verlassen werden und ein anderes Programm ausgeführt werden.

Aufgerufenes Programm :

Das Programm, das durch den CALL-Befehl aufgerufen wird.

B-Feld :

Umfasst die Spalten 10-70 im COBOL-Quelltext. Dort stehen die Befehle der einzelnen Abschnitte und Sektionen.

Begrenzer :

Der Begrenzer kennzeichnet das Ende eines Zeichenstrings, ohne selbst zum String zu gehören. Er besteht aus einem oder mehreren Zeichen.

Editierzeichen :

Editierzeichen werden durch ein Zeichen oder durch eine Kombination von zwei Zeichen dargestellt :

Zeichen	Bedeutung
B	Leerzeichen
0	Null
+	Plus
-	Minus
CR	Plus (Credit)
DB	Minus (Debit)
Z	Nullausgabe unterdrücken
$	Währungszeichen
,	Komma (Dezimalpunkt)
.	Punkt (Dezimalpunkt)
/	Querstrich

Elementares Datenfeld :

Einzelnes Datenfeld aus einer Datengruppe.

Filename :

Benutzer-definiertes Word für das FILE SECTION-Feld. Definiert den Namen der Files.

Indikator-Feld :

Das Indikator-Feld bei NEVADA-Cobol ist die fünfte Spalte jeder Zeile. In diesem Indikator-Feld wird angegeben, ob es sich um eine reguläre Programmzeile, eine Kommentarzeile oder um eine Zeile handelt, die nicht compiliert werden soll. Außerdem besteht die Möglichkeit, Zeichenstrings in der folgenden Zeile weiterzuführen. Dies passiert dadurch, daß im Indikator-Feld das Zeichen "-" gesetzt wird. Der Zeichenstring wird dann in Spalte 11 weitergeführt, wobei in Spalte 10 ein Anführungszeichen stehen muß.

Zeichen im Indikator-Feld (Spalte 5)	Bedeutung
Leerzeichen (Space)	Es handelt sich um eine reguläre Programmzeile
"/", "*"	Dies sind Kommentarzeilen
"D"	Es handelt sich um eine Programmzeile, die nur compiliert wird, wenn in dem ENVIRONMENT DIVISION-Feld die Anweisung WITH DEBUGGING MODE gemacht wird
"-"	In dieser Zeile wird ein Zeichenstring weitergeführt, der für die vorherige Zeile zu lang ist. In Spalte 10 muß ein Anführungszeichen stehen.

Kommentarzeile :

Wird in Spalte 5 einer Zeile ein "*", "/" oder ein anderes Zeichen eingesetzt, so wird diese Zeile als Kommentar aufgefasst.

COBOL Zeichensatz :

Der komplette COBOL-Zeichensatz besteht aus folgenden 51 Zeichen :

0,1,2...9	Ziffern
A,B...Z	Buchstaben (Groß/Kleinbuchstaben)
	Leerzeichen (SPACE)
+	Pluszeichen
-	Minuszeichen
*	Asterik (Multiplikation)
/	Querstrich (Division)
=	Gleichheitszeichen
$	Dollarzeichen
,	Komma (Dezimalpunkt)
;	Semikolon
.	Punkt (Dezimalpunkt)
"	Anführungszeichen
( )	Runde Klammern
<>	Spitze Klammern (Größer/Kleiner)

Compiler-wirksamer Befehl :

Mit Hilfe Compiler-wirksamer Befehle kann die Compilation beinflusst werden. So kann durch den COPY-Befehl beispielsweise ein Text in ein Quellprogramm eingefügt werden, der sich auf Diskette befindet.

CONFIG-Programm (CONFIG.GBL) :

Das CONFIG-Programm ist dazu gedacht, die Ausgabefunktionen des RUN-TIME-Moduls an das CP/M-System anzupassen. Da das Programm im COBOL-Quelltext vorhanden ist, muß es vorher compiliert werden (A>CC RENUMBER).

Datenname :

Benutzer-definierter Name eines Datenfeldes.

DEBUGGING-MODUS :

Gibt man hinter der Anweisung SOURCE COMPUTER den Befehl WITH DEBUGGING MODE, so werden außerdem alle Zeilen compiliert, in deren Indikator-Feld (Spalte 5) das Zeichen D steht. Wird dieser Befehl nicht gegeben, so werden diese Zeilen als Kommentar aufgefasst.

Dimensionieren :

Will man mehrere Daten unter einem Namen ablegen, so muß man Variablen dimensionieren. Zu jedem einzelnen Datenfeld einer dimensionierten Variablen gehört eine Zahl, die die Position des Datenfeldes in der Variablen angibt. Durch Angabe des Variablennamen und der dazugehörigen Zahl kann man auf jedes Datenfeld zugreifen.

Nicht-numerischer String :

Ein Zeichenstring, der von Anführungszeichen eingegrenzt wird. Um Anführungszeichen darzustellen, werden zwei Anführungszeichen hintereinander eingegeben.

Numerisches Zeichen :

Zeichen aus dem Satz 1,2,3,4,5,6,7,8,9,0.

Numerischer String :

Ein String, der aus mehreren numerischen Zeichen besteht. Um Dezimalzahlen darzustellen, kann ein Dezimalpunkt eingefügt werden. Die Vorzeichen +/- stehen ganz links.

OBJECT-COMPUTER :

In dem OBJECT-COMPUTER-Feld der ENVIRONMENT DIVISION werden Angaben über die Systemkonfiguration gemacht, auf der das Programm lauffähig ist.

Objekt-Programm :

Das Objekt-Programm besteht aus Anweisungen in Maschinensprache und wird durch die Compilation des Quellprogramms erzeugt. Um ein Objekt-Programm zu starten, verwendet man das RUN-TIME-Modul. Im RUN-Time-Modul (RUN.COM) sind Routinen enthalten, die unter anderem die Ausgabe von Zeichen auf Bildschirm steuern.
Will man das Programm RENUMBER.OBJ starten, erfolgt die Eingabe :

A>RUN RENUMBER <CR>

(A> ist die Bereitschaftsmeldung von CP/M, A gibt das angemeldete Laufwerk an)

Optionales Wort :

Wort ist für die Funktion des Befehls nicht notwendig und kann somit weggelassen werden. Um jedoch die Lesbarkeit eines Programms zu erhöhen, ist es dennoch sinnvoll, auch optionale Worte anzugeben.

PICTURE-Anweisung :
Die PICTURE-Anweisung dient dazu, die Art und Größe von Variablen festzulegen.

Prozedur :

Eine Prozedur ist eine Folge von Abschnitten oder Sektionen innerhalb der PROCEDURE DIVISION.

Quellprogramm :

Programmtext, der vom Benutzer in COBOL geschrieben wird und später durch Compilation den lauffähigen Objekt-Code ergibt.

Relativer Schlüssel :

Der relative Schlüssel wird ausschließlich für Files mit wahlfreiem Zugriff (RANDOM-Files) benötigt. Er gibt an, welcher logische Record gelesen/geschrieben werden soll. Die Variable, die den relativen Schlüssel enthält, wird in der INPUT-OUTPUT-SECTION definiert.

RENUMBER-Programm (RENUMBER.CBL) :

Das RENUMBER-Programm dient dazu, die Zeilennummern eines Quellprogramms neu zu nummerieren. Dies hat den Vorteil, daß das Programm leichter zu lesen ist. Allerdings muß das Quellprogramm, das neu durchnummeriert werden soll, ein Dateiende-Zeichen enthalten (Control + Z). Sollte dies nicht der Fall sein, so funktioniert das nummerieren nicht.

Sektions-Name :

Benutzer-definierter Name für eine Sektion in dem PROCEDURE DIVISION-Teil.

Sequentieller Zugriff :

Beim sequentiellen Zugriff werden Records der Reihe nach bearbeitet. Das bedeutet, daß beim Lesen oder Schreiben immer der nächste logische Record des Files bearbeitet wird.

SOURCE-COMPUTER :

In dem SOURCE-COMPUTER-Feld der ENVIRONMENT DIVISION werden Angaben über die Systemkonfiguration gemacht, auf der das Programm compiliert wurde.

Unterprogramm :

Siehe aufgerufenes Programm.

Wahlfreier Zugriff :

Beim wahlfreiem Zugriff ist es möglich, beliebige Records aus einem RANDOM-File zu lesen und zu schreiben. Dabei muß der relative Schlüssel, der die Position des Records im File angibt, in einer numerischen Variablen abgelegt werden. Die numerische Variable im FILE-CONTROL-Feld der INPUT-OUTPUT SECTION definiert.

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID.
0003     T6WF.
0004* THIS PROGRAM CREATES A FILE OF FIXED LENGTH RECORDS
0004* IF THE RECORD SIZES ARE CHANGED TO YOUR NEEDS, CAN BE
0004* USED TO CREATE THE SPACE NEEDED (ALLOCATE) FOR A
0004* RANDOM FILE.
0005 ENVIRONMENT DIVISION.
0006 CONFIGURATION SECTION.
0007 SOURCE-COMPUTER.
0008     8080-CPU.
0009 OBJECT-COMPUTER.
0010     8080-CPU.
0011 INPUT-OUTPUT SECTION.
0012 FILE-CONTROL.
0013     SELECT FILE1 ASSIGN TO DISK
0014       ORGANIZATION IS SEQUENTIAL
0015       ACCESS MODE IS SEQUENTIAL.
0016 DATA DIVISION.
0017 FILE SECTION.
0018 FD FILE1
0019     LABEL RECORDS ARE STANDARD
0020     VALUE OF FILE-ID IS OUT-FILE-NAME
0021     BLOCK CONTAINS 1 RECORD
0022     DATA RECORDS ARE O-RECORD.
0023 01 O-RECORD.
0024     02 SEQ PIC 9999.
0025     02 REC1 PIC IS X(156).
0026     02 SEQ2 PIC 9999.
0027 WORKING-STORAGE SECTION.
0028 01 OUT-FILE-NAME PIC X(14)
0029         VALUE "A:TESTF.WRK".
0030 01 X1 PIC 9999
0031         VALUE 0001.
0032 PROCEDURE DIVISION.
0033 BEGIN.
0034     DISPLAY "ENTER OUTPUT FILE NAME ".
0035     DISPLAY OUT-FILE-NAME WITH NO ADVANCING.
0036* to accept and use the file-name just displayed you can
0036* hit the <CR> key.  see # 2 under accept.
0036     ACCEPT OUT-FILE-NAME.
0037     OPEN OUTPUT FILE1.
0038     MOVE SPACES TO O-RECORD.
0039 BEGIN2.
0040     MOVE X1 TO SEQ.
0041     MOVE X1 TO SEQ2.
0042     ADD 1 TO X1.
0043     DISPLAY O-RECORD.
0044     WRITE O-RECORD.
0045     IF X1 IS = TO 201
0046       GO TO EOJ.
0047     GO TO BEGIN2.
0048 EOJ.
0049     CLOSE FILE1.
0050     STOP RUN.
0051 END PROGRAM T6WF.
```

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID.
0003     T6RD.
0004* THIS PROGRAM READS A VARIABLE LENGTH (DELIMITED) FILE.
     *  the kind of file created by most text editors.  Each
     *  record in the file is terminated with a carriage
     *  return and line feed.
0005 ENVIRONMENT DIVISION.
0006 CONFIGURATION SECTION.
0007 SOURCE-COMPUTER.
0008     8080-CPU.
0009 OBJECT-COMPUTER.
0010     8080-CPU.
0011 INPUT-OUTPUT SECTION.
0012 FILE-CONTROL.
0013     SELECT FILE1 ASSIGN TO DISK
0014       ORGANIZATION IS SEQUENTIAL
0015       ACCESS MODE IS SEQUENTIAL
     * the next statement tells the compiler the records will
     * end with a carriage return and line feed.
0016       RECORD DELIMITER IS STANDARD.
0017 DATA DIVISION.
0018 FILE SECTION.
0019 FD FILE1
0020     LABEL RECORDS ARE STANDARD
0021     VALUE OF FILE-ID IS IN-FILE
0022     DATA RECORDS ARE I-RECORD.
0023 01 I-RECORD.
0024     02 SEQ PIC 9999.
0025     02 REC1 PIC IS X(160).
0026 WORKING-STORAGE SECTION.
0027 01 IN-FILE PIC X(14)
0028         VALUE "A:TESTB.WRK".
0029 PROCEDURE DIVISION.
0030 BEGIN.
0031     DISPLAY "ENTER INPUT FILE NAME ".
0032     DISPLAY IN-FILE WITH NO ADVANCING.
0033     ACCEPT IN-FILE.
0034     OPEN INPUT FILE1.
0035 BEGIN2.
0036* the next statement is necessary because the delimited
0036* read only transfers data into the record area and if
0036* short the data from prior reads will be in the record
0036* area on the right end.
0036     MOVE SPACE TO I-RECORD.
0037     READ FILE1
0038       AT END
0039       GO TO EOJ.
0040     DISPLAY I-RECORD.
0041     GO TO BEGIN2.
0042 EOJ.
0043     CLOSE FILE1.
0044     STOP RUN.
0045 END PROGRAM T6RD.
```

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID. TST-PRT.
    * this sample program reads in a variable length file
    * and outputs it to the printer.
0003 ENVIRONMENT DIVISION.
0004 CONFIGURATION SECTION.
0005 SOURCE-COMPUTER. 8080-CPU.
0006 OBJECT-COMPUTER. 8080-CPU.
0008 INPUT-OUTPUT SECTION.
0009 FILE-CONTROL.
0010     SELECT FILE1 ASSIGN TO DISK
0011     RECORD DELIMITER IS STANDARD.
    * the next line is for printers and/or printer-files.
0012     SELECT FILE2  ASSIGN TO PRINTER.
0013 DATA DIVISION.
0014 FILE SECTION.
0015 FD  FILE1
0016     LABEL RECORDS ARE STANDARD
0017     VALUE OF FILE-ID IS IN-FILE1-NAME
0018     DATA RECORD IS TESTB.
0019 01  TESTB PIC X(80).
0020 FD  FILE2
0021     LABEL RECORDS ARE STANDARD
0022     VALUE OF FILE-ID IS OUT-FILE2-NAME
0023     DATA RECORD IS PRINT-LINE.
0024 01  PRINT-LINE PICTURE IS X(132).
0025 WORKING-STORAGE SECTION.
    * the input file-name can be a cobol source file to be
    * listed on the printer. this file-name can be changed
    * at run time see lines 0030-0032.
0026 01 IN-FILE1-NAME PIC X(14) VALUE "A:T01.CBL".
    * in line 0027  "printer" is the key word to send output
    * to the physical printer.
    * any other file-name sends output to the named disk
    * file.  this option of either printing or sending
    * output to the printer can be made at run time. see
    * lines 0033-0035.
0027 01 OUT-FILE2-NAME PIC X(14) VALUE "PRINTER".
0028 PROCEDURE DIVISION.
0029 BEGIN.
0030     DISPLAY "ENTER INPUT FILE ".
0031     DISPLAY IN-FILE1-NAME WITH NO ADVANCING.
0032     ACCEPT IN-FILE1-NAME.
0033     DISPLAY "ENTER PRINTER FILE ".
0034     DISPLAY OUT-FILE2-NAME WITH NO ADVANCING.
    * no need to re-enter the word "printer" just hit <cr>
0035     ACCEPT OUT-FILE2-NAME.
0036     OPEN INPUT FILE1.
0037     OPEN OUTPUT FILE2.
0038     MOVE SPACES TO PRINT-LINE.
0039 PARA-3.
0040     MOVE SPACES TO TESTB.
0041     READ FILE1 AT END GO TO EOJ.
0042     MOVE TESTB TO PRINT-LINE.
0043     WRITE PRINT-LINE BEFORE ADVANCING 1 LINE.
0044     GO TO PARA-3.
0045 EOJ.
0046     MOVE SPACES TO PRINT-LINE.
0047     WRITE PRINT-LINE BEFORE ADVANCING PAGE.
0048     CLOSE FILE1.
0049     CLOSE FILE2.
0050     STOP RUN.
0051 END PROGRAM TST-PRT.
```

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID.
0003     T8WR.
0004* THIS PROGRAM WRITES RANDOM FIXED LENGTH RECORDS TO A
0004* FILE THAT HAS BEEN CREATED USING A SEQUENTIAL FIXED
0004* LENGTH WRITE PROGRAM TO ALLOCATE THE REQUIRED FILE
0004* SPACE.
0005 ENVIRONMENT DIVISION.
0006 CONFIGURATION SECTION.
0007 SOURCE-COMPUTER.
0008     8080-CPU.
0009 OBJECT-COMPUTER.
0010     8080-CPU.
0011 INPUT-OUTPUT SECTION.
0012 FILE-CONTROL.
0013     SELECT FILE1 ASSIGN TO DISK
0014       ORGANIZATION IS
0015       RELATIVE
0016       ACCESS MODE IS RANDOM
0017       RELATIVE KEY IS KEY-1.
0018 DATA DIVISION.
0019 FILE SECTION.
0020 FD FILE1
0021     LABEL RECORDS ARE STANDARD
0022     VALUE OF FILE-ID IS OUT-FILE
0023     DATA RECORDS ARE O-RECORD.
0024 01 O-RECORD.
0025     02 SEQ PIC 9999.
0026     02 REC1 PIC IS X(160).
0027 WORKING-STORAGE SECTION.
0028 01 OUT-FILE PIC X(14)
0029         VALUE "A:TESTF.WRK".
0030 01 KEY-1 PIC 9(7) COMP-3.
0031 01 XX-KEY PIC 9(4) VALUE 1.
0032 PROCEDURE DIVISION.
0033 BEGIN.
0034     DISPLAY "ENTER OUTPUT FILE NAME ".
0035     DISPLAY OUT-FILE WITH NO ADVANCING.
0036     ACCEPT OUT-FILE.
0037     OPEN OUTPUT FILE1.
0038 BEGIN2.
0039     MOVE SPACE TO O-RECORD.
0040     MOVE 0001 TO XX-KEY.
0041     DISPLAY "ENTER RECORD NUMBER 0001 ".
0042     ACCEPT XX-KEY.
0043     IF XX-KEY IS NOT NUMERIC
0044       GO TO BEGIN2.
0045     IF XX-KEY = 9999
0046       GO TO EOJ.
0047     MOVE XX-KEY TO KEY-1.
0048     MOVE XX-KEY TO SEQ.
0049     DISPLAY "ENTER DATA FOR RECORD ".
0050     ACCEPT REC1.
0051     WRITE O-RECORD
0052       INVALID KEY
0053       DISPLAY "INVALID KEY" GO TO BEGIN2.
0054     DISPLAY O-RECORD.
0055     GO TO BEGIN2.
0056 EOJ.
0057     CLOSE FILE1.
0058     DISPLAY "EOJ".
0059     STOP RUN.
0060 END PROGRAM T8WR.
```

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID.
0003     T8RR.
0004*  THIS PROGRAM READS RANDOM FIXED LENGTH RECORDS.
0005 ENVIRONMENT DIVISION.
0006 CONFIGURATION SECTION.
0007 SOURCE-COMPUTER.
0008     8080-CPU.
0009 OBJECT-COMPUTER.
0010     8080-CPU.
0011 INPUT-OUTPUT SECTION.
0012 FILE-CONTROL.
0013     SELECT FILE1 ASSIGN TO DISK
0014       ORGANIZATION IS
0015       RELATIVE
0016       ACCESS MODE IS RANDOM
0017       RELATIVE KEY IS KEY-1.
0018 DATA DIVISION.
0019 FILE SECTION.
0020 FD FILE1
0021     LABEL RECORDS ARE STANDARD
0022     VALUE OF FILE-ID IS IN-FILE
0023     DATA RECORDS ARE I-RECORD.
0024 01 I-RECORD.
0025     02 PART-NUMBER PIC 9999.
0026     02 ITEM-DESCRIPTION PIC IS X(160).
0027 WORKING-STORAGE SECTION.
0028 01 IN-FILE PIC X(14)
0029         VALUE "A:TESTF.WRK".
0030 01 KEY-1 PIC 9(7) COMP-3.
0031 01 XX-KEY PIC 9(4).
0032 PROCEDURE DIVISION.
0033 BEGIN.
0034     DISPLAY "ENTER INPUT FILE NAME".
0035     DISPLAY IN-FILE WITH NO ADVANCING.
0036     ACCEPT IN-FILE.
0037     OPEN INPUT FILE1.
0038     DISPLAY "OPEN".
0039 BEGIN2.
0040     MOVE SPACE TO I-RECORD.
0041     MOVE 0001 TO XX-KEY.
0042     DISPLAY "ENTER RECORD NUMBER 0001 "
0043     ACCEPT XX-KEY.
0044     IF XX-KEY NOT NUMERIC
0045       GO TO BEGIN2.
0046     IF XX-KEY = 9999
0047       GO TO EOJ.
0048     MOVE XX-KEY TO KEY-1.
0049     READ FILE1
0050       INVALID KEY
0051       DISPLAY "INVALID KEY" GO TO BEGIN2.
0051* don't display on invalid key as data is unspecified.
0052     DISPLAY I-RECORD.
0053     GO TO BEGIN2.
0054 EOJ.
0055     CLOSE FILE1.
0056     DISPLAY "EOJ".
0057     STOP RUN.
0058 END PROGRAM T8RR.
```

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID.
0003     T8IOR.
0004* THIS PROGRAM READS THEN REWRITES FIXED LENGTH RECORDS
0005* IN RANDOM MODE.
0006 ENVIRONMENT DIVISION.
0007 CONFIGURATION SECTION.
0008 SOURCE-COMPUTER.
0009     8080-CPU.
0010 OBJECT-COMPUTER.
0011     8080-CPU.
0012 INPUT-OUTPUT SECTION.
0013 FILE-CONTROL.
0014     SELECT FILE1 ASSIGN TO DISK
0015       ORGANIZATION IS
0016       RELATIVE
0017       ACCESS MODE IS RANDOM
0018       RELATIVE KEY IS KEY-1.
0019 DATA DIVISION.
0020 FILE SECTION.
0021 FD FILE1
0022     LABEL RECORDS ARE STANDARD
0023     VALUE OF FILE-ID IS I-O-FILE
0024     BLOCK CONTAINS 1 RECORD
0025     DATA RECORDS ARE A-RECORD.
0026 01 A-RECORD.
0027     02 SEQ PIC 9999.
0028     02 REC1 PIC IS X(160).
0029 WORKING-STORAGE SECTION.
0030 01 I-O-FILE PIC X(14)
0031         VALUE "A:TESTF.WRK".
0032 01 KEY-1 PIC 9(7) COMP-3.
0033 01 XX-KEY PIC 9(4)
0034         VALUE 1.
0035 PROCEDURE DIVISION.
0036 BEGIN.
0037     DISPLAY "ENTER I-O FILE NAME "
0038     DISPLAY I-O-FILE WITH NO ADVANCING.
0039     ACCEPT I-O-FILE.
0040     OPEN I-O FILE1.
0041 BEGIN2.
0042     MOVE SPACE TO A-RECORD.
0043     MOVE 1 TO XX-KEY.
0044     DISPLAY "ENTER RECORD NUMBER 0001 ".
0045     ACCEPT XX-KEY.
0046     IF XX-KEY IS NOT NUMERIC
0047       GO TO BEGIN2.
0048     IF XX-KEY = 9999
0049       GO TO EOJ.
0050     MOVE XX-KEY TO KEY-1.
0051     READ FILE1
0052       INVALID KEY
0053       DISPLAY "READ INVALID KEY" GO TO BEGIN2.
0054     DISPLAY A-RECORD.
0055     DISPLAY "ENTER NEW DATA ".
0056     ACCEPT REC1.
0057     REWRITE A-RECORD
0058       INVALID KEY
0059       DISPLAY "REWRITE INVALID KEY".
0060     DISPLAY A-RECORD.
0061     GO TO BEGIN2.
0062 EOJ.
0063     CLOSE FILE1.
0064     DISPLAY "EOJ".
0065     STOP RUN.
0066 END PROGRAM T8IOR.
```

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID.
0003     T20.
0004*  THIS PROGRAM CALLS PROGRAM T20A WHICH IN TURN CALLS
0005*  PROGRAM T20B.
0006 ENVIRONMENT DIVISION.
0007 CONFIGURATION SECTION.
0008 SOURCE-COMPUTER.
0009     8080-CPU.
0010 OBJECT-COMPUTER.
    * the following memory statement is necessary for memory
    * mapping as it marks the upper boundary address(16383).
    * the data from this program loads from the bottom-up
    * and from the top-down.  free space, if any, is
    * somewhere between the top address and the starting
    * address.
0011     8080-CPU MEMORY SIZE 16383 CHARACTERS.
0012 DATA DIVISION.
0013 WORKING-STORAGE SECTION.
0014 01 M1.
0015    02 M1-2.
0016     03 M1-3 PIC XXX.
0017     02 M1-4 PIC 99.
0018     02 M1-5 PIC 99V99 COMP VALUE 11.11.
0019     02 M1-6 PIC 999999V99 COMP-3 VALUE 012345.78.
0020     02 M1-7 PIC $99,999.99.
0021 01 M2 PIC S9V9999 VALUE 0.6143.
0022 01 M3 PIC X(10) VALUE "A:T20A".
0023 01 M4 PIC X (120).
0024 01 M5 PIC X(20) JUSTIFIED.
0025 PROCEDURE DIVISION.
0026 BEGIN.
0027     DISPLAY "START T20".
0028     MOVE ALL "A" TO M4.
0029     CALL "T20A" USING M1, M2, M3, M4, M5.
0030     DISPLAY "EOJ-T20".
0031     STOP RUN.
0032 END PROGRAM T20.
```

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID.
0003     T20A.
0004*  THIS PROGRAM IS CALLED BY T20 AND IN TURN CALLS
0005*   PROGRAM T20B.
0006 ENVIRONMENT DIVISION.
0007 CONFIGURATION SECTION.
0008 SOURCE-COMPUTER.
0009     8080-CPU.
0010 OBJECT-COMPUTER.
    * the following memory statement is necessary. it must
    * be at least 1 byte higher than the previous programs
    * ending address (16383+1=16384) in this example.
0011     8080-CPU MEMORY BEGINNING 16384 ENDING 20000.
0012 DATA DIVISION.
0013 WORKING-STORAGE SECTION.
0014 01 L3 PIC X(10) VALUE "A:T20A".
0015 LINKAGE SECTION.
0016 01 M1.
0017    02 M1-2.
0018    03 M1-3 PIC XXX.
0019    02 M1-4 PIC 99.
```

```
0020     02 M1-5 PIC 99V99 COMP.
0021     02 M1-6 PIC 999999V99 COMP-3.
0022     02 M1-7 PIC $99,999.99.
0023 01 M2 PIC S9V9999.
0024 77 M3 PIC X(10).
0025 77 M4 PIC X(120).
0026 77 M5 PIC X(20) JUSTIFIED.
0027 PROCEDURE DIVISION
0028* no period after the word division when using using
0029         USING M1, M2, M3, M4, M5.
0030 BEGIN.
0031     DISPLAY "THIS IS T20A".
0032     DISPLAY M3.
0033     DISPLAY M4.
0034     CALL "T20B" USING L3.
0035     CANCEL "T20B".
0036 EOJ1.
0036     EXIT PROGRAM.
0037 EOJ.
0038     STOP RUN.
0039 END PROGRAM T20A
```

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID.
0003     T20B.
0004*  THIS PROGRAM IS CALLED BY T20A AND EXITS BACK TO IT.
0005*  NOTE HOW THE MEMORY IS ALLOCATED.
0006 ENVIRONMENT DIVISION.
0007 CONFIGURATION SECTION.
0008 SOURCE-COMPUTER.
0009     8080-CPU.
0010 OBJECT-COMPUTER.
     * the following memory statement is necessary to
     * control the memory mapping of this third program
     * module. it starts at address 20001 just one byte
     * higher than the previous programs ending address.
0011     8080-CPU MEMORY BEGINNING 20001 ENDING 24000.
0012 DATA DIVISION.
0013 FILE SECTION.
0014 WORKING-STORAGE SECTION.
0015 01 L1 PIC X(10) VALUE SPACE.
0016 LINKAGE SECTION.
0017 01 L3 PIC X(10).
0018 PROCEDURE DIVISION
0019     USING L3.
0020 BEGIN.
0021     DISPLAY "THIS IS T20-B".
0022     DISPLAY L3.
0023 EOJ1.
0024     EXIT PROGRAM.
0025 EOJ.
0026     STOP RUN.
0027 END PROGRAM T20B
```

```
0001 IDENTIFICATION DIVISION.
0002 PROGRAM-ID.
0003     T6WD.
0004*  THIS PROGRAM CREATES A FILE OF VARIABLE LENGTH
0005*  (DELIMITED) RECORDS.  Most text editors create
    *  this type of file.  Each record ends with a carriage
    *  return and line feed.
0005 ENVIRONMENT DIVISION.
0006 CONFIGURATION SECTION.
0007 SOURCE-COMPUTER.
0008     8080-CPU.
0009 OBJECT-COMPUTER.
0010     8080-CPU.
0011 INPUT-OUTPUT SECTION.
0012 FILE-CONTROL.
0013     SELECT FILE1 ASSIGN TO DISK
0014       ORGANIZATION IS SEQUENTIAL
0015       ACCESS MODE IS SEQUENTIAL
    * the next statement tells the compiler each record is
    * to be delimited (separated) by or ended with a
    * carriage return and line feed.
0016       RECORD DELIMITER IS STANDARD.
0017 DATA DIVISION.
0018 FILE SECTION.
0019 FD FILE1
0020     LABEL RECORDS ARE STANDARD
0021     VALUE OF FILE-ID IS OUT-FILE
0022     DATA RECORDS ARE O-RECORD.
0023 01 O-RECORD.
0024     02 SEQ PIC 9999.
0025     02 REC1 PIC IS X(156).
0026     02 SEQ2 PIC 9999.
0027 WORKING-STORAGE SECTION.
0028 01 OUT-FILE PIC X(14)
0029         VALUE IS "A:TESTB.WRK".
0030 01 X1 PIC 9999
0031         VALUE 0001.
0032 01 PAD.
0033     02 FILLER PIC X(30)
0034         VALUE SPACE.
0035     02 FILLER PIC X(30)
0036         VALUE SPACE.
0037     02 FILLER PIC X(30)
0038         VALUE SPACE.
0039     02 FILLER PIC X(30)
0040         VALUE SPACE.
0041     02 FILLER PIC X(30)
0042         VALUE SPACE.
0043     02 FILLER PIC X(05)
0044         VALUE "AAAAA".
0045 PROCEDURE DIVISION.
0046 BEGIN.
0047     DISPLAY "ENTER OUTPUT FILE NAME ".
0048     DISPLAY OUT-FILE WITH NO ADVANCING.
0049     ACCEPT OUT-FILE.
0050     MOVE SPACES TO O-RECORD.
0051     OPEN OUTPUT FILE1.
0052     DISPLAY "OPEN".
0053     MOVE PAD TO REC1.
0054 BEGIN2.
0055     MOVE X1 TO SEQ.
0056     MOVE X1 TO SEQ2.
0057     ADD 1 TO X1.
0058     DISPLAY O-RECORD.
0059     WRITE O-RECORD.
0060     IF X1 = 011
0061       GO TO EOJ.
0062     GO TO BEGIN2.
0063 EOJ.
0064     CLOSE FILE1.
0065     STOP RUN.
0066 END PROGRAM T6WD.
```