

# Teil 5

## COBOL

### COmmon Business Oriented Language '60

- Kaufmännische, behördliche Anwendungen
    - E/A ist Hauptteil der Anwendung, große Zahl einfacher Berechnungen,
      - Dezimalarithmetik,
- DoD & IBM, 1959/60,

die treibenden Kräfte: Grace Hopper, Jean Sammet

Standardisierung 1968 durch CODASYL (COference on DATA SYstems Languages)

dann 1974, 1985, de facto Standard: IBM COBOL



# COBOL by the numbers

## (Quelle: ZDNet / Gartner Inc., 2001)

Crunching the COBOL numbers leads to one chilling conclusion: The projected proliferation of COBOL code over the next four years goes hand-in-hand with a drastic dwindling of the number of programmers who know anything about supporting the language. Here's the skills-hungry picture Gartner paints:

- 200 billion lines of COBOL code in existence in 2000
- 5 billion lines estimated annual growth of COBOL code over the next four years
- 90,000 Number of COBOL programmers in North America in 2000
- 13 percent estimated annual decrease in number of COBOL programmers due to retirement and death



# Sprache ohne Programmierer

- 85% des Welthandels gehen über Systeme, die in COBOL implementiert sind.
- 60% der weltweiten Codebasis ist COBOL, die Zahl vergrößert sich sogar!
- Die COBOL-Programmierer gehen zur Zeit in Rente oder sterben.  
<< Unfortunately, many four-year colleges and universities have cut back or stopped giving classes in COBOL. "You can't get the new kids—the dot-commers—to take a second look at COBOL. It's far easier to teach a COBOLer the dot-com stuff than vice versa. Knowing COBOL means you already know how the business runs." >>



# Entwurfsprinzipien

- Strikte Trennung der Beschreibung von Daten und Operationen.
- Selbst-Dokumentation: Programme wie Sätze, Flut von Schlüsselworten, oft optional
- Unterscheidung Umgebung - interne Namen
- Statische Speicherzuteilung
- Keine Unterprogramme (ursprünglich)
- Sehr umfangreiche Sprache



# ANSI-COBOL-85 Norm

- Bestehend aus elf Bausteinen
- Bausteine obligatorisch, optional oder zusätzlich
- Verschiedene Leistungsstufen einzelner Bausteine
- Keine zwei Übersetzer übersetzen die gleiche Sprache.
- IBM Übersetzer de facto Standard.
- 
- Neuer Standard: 2002, enthält unter anderem:
  - object orientation, portable arithmetic, exception handling, native binary data types, recursion, cultural adaptability, multilingual features, and tailoring for a given local language or culture.



# Obligatorische Bausteine

- Kern (*Nucleus*):  
allgemeine Sprachelemente, Datendefinitionen (Elemente, Gruppen, Sätze, Tabellen), Basisoperationen (Zuweisung, Arithmetik, etc.),  
Programmablaufsteuerung
- Sequentielle EA:  
sequentieller Zugriff auf (sequentielle) Dateien, Formatierung von Seiten in  
Druckdateien
- Programmkommunikation:  
Aufruf externer Programme, fremde Sprachen anbinden



# Optionale Bausteine

- Relative EA: sequentieller und wahlfreier Zugriff auf strukturierte Dateien
- Indizierte EA: Sätze einer Datei indiziert mit Schlüsseln, alternative Schlüssel, sequentieller und wahlfreier Zugriff über Schlüssel
- Sort/Merge: Sortieren und Mischen von Dateien, Operationen zur Datensatztransformation vor dem Sortieren/Mischen, Benutzergesteuerte Sortieroperationen
- Quelltext Manipulation: Bibliotheksmodell, Zusammenbinden verschiedener Programme



# Zusätzliche Bausteine

- Testhilfe:  
Sprachelemente zur Fehlersuche (z.B. Statusregister)
- Kommunikation:  
Datenfernübertragung
- Segmentierung:  
Auslagerung einzelner Programmteile zur Laufzeit
- *Report Writer*:  
Listenerstellung



# Programmformate

Festes Format (veraltet):

- Spalte 1-6: Marke, numerisch, führende Nullen ignoriert
- 7: Kommentar- (\* , /) oder Fortsetzungszeichen (-)
- 8-11: Feld A - Stufennummern, Hauptabschnitte
- 12-72: Feld B - Namen in Datenbeschreibungen, Anweisungen
- 73-80: Kommentar (ignoriert, Zeilenkennung/Numerierung)

Alternative: Freies Format

- Groß-/Kleinbuchstaben äquivalent, Minuszeichen Bestandteil von Bezeichnern
- Überschriften, Vereinbarungen, Anweisungen abgeschlossen mit Punkt.



# Programmaufbau

Hierarchische Gliederung:  
Division - Section - Paragraph

vier Abteilungen:

- IDENTIFICATION DIVISION
  - Verfasser , Datum, usw.
- ENVIRONMENT DIVISION
  - Konfiguration, EA-Steuerung: Zuordnung externe-interne Datei
- DATA DIVISION
  - Dateivereinbarungen, EA-Puffer, Datenvereinbarungen
- PROCEDURE DIVISION
  - Anweisungen. Abschnitts-/Absatznamen als Sprungziele.



# IDENTIFICATION DIVISION

- Name des Programms
- Allgemeine Programminformationen
- Optionale Zusatzangaben ohne semantische Bedeutung



# ENVIRONMENT DIVISION

- Konfigurationsabschnitt:
  - Übersetzung (Quell-Ziel-System),
  - Internationalisierung (Dezimaltrenner, Sortierangaben, etc.)
- Ein-Ausgabe-Informationen:
  - Zuordnung externe (physikalische) Dateien - interne (logische) Dateien
  - Vereinbarung der EA-Puffer



# ENVIRONMENT DIVISION

```
ENVIRONMENT DIVISION.  
CONFIGURATION SECTION.  
SOURCE-COMPUTER.  
    Atari ST  
OBJECT-COMPUTER.  
    Cray  
SPECIAL-NAMES.  
    DECIMAL-POINT IS COMMA.  
INPUT-OUTPUT SECTION.  
SELECT datafile ASSIGN TO Data.dat
```



# Datei (Data-Set)

- *zuordnung* ::= SELECT *intern* ASSIGN TO  
    *system* [*organisation*] [*zugriff*] [*schlüssel*] [*status*]
- *organisation* ::= ORGANIZATION IS  
    (SEQUENTIAL / RELATIVE / INDEXED)
- *zugriff* ::= ACCESS IS  
    (SEQUENTIAL / RELATIVE / INDEXED)
- *schlüssel* ::= RELATIVE KEY IS *ganzzahlFeld*  
    RECORD KEY IS *feld*  
    [ALTERNATIVE RECORD KEY IS *feld*  
    [ WITH DUPLICATES ] ]\*
- *status* ::= FILE STATUS IS *zweiZeichenFeld*



# DATA DIVISION

- FILE SECTION
  - Persistente Daten und Dateistrukturen
- WORKING-STORAGE SECTION
  - Temporäre Daten
- LINKAGE SECTION
  - Formale Parameter



# Typkonzept

- Skalare Werte:  
*datenelement ::= sn datenname PICTURE IS*  
*art [ USAGE IS *format* ] [ VALUE IS *literal* ]*
  - Statische Reihungen (Tabellen):  
*tabelle ::= datenelement OCCURS ganze zahl*  
*[ ASCENDING KEY IS [ datenname ]<sup>+</sup>*  
*[ INDEXED BY datenname ]*  
*zugriff ::= tabellenname ( ganze zahl )*
  - Records (Datensätze):  
*satz ::= [ datenelement ] \**  
*zugriff ::= datenname [ . datenname ] \**
- Reihungen maximal dreistufig
  - Satznummern, um Hierarchiestufen in Records zu definieren



# Datenart

- PICTURE IS (Kurzform PIC)
- X - Alphanumerisch
- A - Alphabetisch
- 9 - Numerisch
- Anzahl Stellen definiert durch
  - Klammerung, z.B. X(3)
  - Wiederholung, z.B. XXX



# Numerische Typen

- PICTURE IS [S](9\*|9(*zahl*))  
[V(9\*|9(*zahl*))] USAGE *art*
- *art* ::= BINARY | COMPUTATIONAL-1 |  
COMPUTATIONAL-2 | DISPLAY | PACKED-DECIMAL
- **Beispiel:** 01 bezeichner PIC S99V99 1.



# Druckformate

- **Beispiel**  
01 bezeichner PIC Z99.99CB SIGN LEADING
- **Dezimalpunkt oder -Komma:** . | ,
- **Keine Nullen vorn:** Z
- **Soll und Haben:** CB | DB
- **Formate gehören zum Datenelement nicht zu EA Operationen**



# DATA DIVISION

- FILE SECTION.  
FD *interne-datei-1*.  
01 *daten-satz-name* PIC ...  
02 ...  
FD *interne-datei-2*. ...
- WORKING-STORAGE SECTION.  
01 *daten-satz-name-1* PIC ...  
02 ...  
01 *daten-satz-name-2* PIC ...
- LINKAGE SECTION.  
01 *daten-satz-name* PIC ...

Ebene 02 bedeutet Unterverbund, Ebene 03 Unter-Unterverbund, usw.



# PROCEDURE DIVISION

- Anweisungsteil
- Anweisungen bilden Sätze
- Sätze bilden Paragraphen
- Paragraphen bilden Abschnitte
- Prozedur - Oberbegriff für Abschnitte und Paragraphen
- Paragraph- und Abschnittsbezeichner: Marken und Untermarken



# Bildschirm Ein- und Ausgabe

- *ausgabe* ::= DISPLAY ( *literal* | *datename* )

- *eingabe* ::= ACCEPT *datename*

- Beispiel:

```
WORKING-STORAGE SECTION.
```

```
    01 zeile PIC X(80) VALUE String.
```

```
PROCEDURE DIVISION.
```

```
    DISPLAY zeile,
```

```
    ACCEPT zeile,
```

```
    DISPLAY zeile.
```



# Eingabemasken

```
DATA DIVISION.  
WORKING-STORAGE SECTION.  
01 kunde.  
02 name PIC X(30).  
    02 vorname PIC X(30).  
    02 adresse PIC X(100).  
SCREEN SECTION.  
01 eingabemaske.  
    02 blank screen.  
    02 line 1 column 5 value "Name:".  
    02 line 1 column 15 PIC X(30) to name.  
    02 line 2 column 5 value "Vorname:".  
    02 line 2 column 15 PIC X(30) to vorname.  
    02 line 3 column 5 value "Adresse:".  
    02 line 3 column 15 PIC X(100) to adresse.  
PROCEDURE DIVISION:  
display eingabemaske.  
accept eingabemaske.
```



# Perform-Anweisung, Unterprogramme, Schleifen

- *perform* ::= PERFORM *wiederholung* [ *anweisung* | *Marken* ]\* END-PERFORM

- *wiederholung* ::= ( *ganze zahl* | *datename* ) TIMES

- Beispiel:

```
PROCEDURE DIVISION.  
    PERFORM 100 TIMES  
        ACCEPT zeile,  
        DISPLAY zeile  
END-PERFORM.
```

- Beispiel:

```
PROCEDURE DIVISION.  
    PERFORM 100 TIMES PARAGRAPH-1 THRU PARAGRAPH-7  
END-PERFORM.
```

```
...  
PARAGRAPH-1. ...
```

```
...  
PARAGRAPH-7. ...
```

- also: *perform*-Anweisung erlaubt Zählschleife und Unterprogrammaufruf (parameterlos, keine Rekursion)

- Problem: wie unterscheidet man Absätze, die normal durchlaufen werden



# While Schleife

- *perform ::=* PERFORM *abbruch* [ *anweisung* ]\*  
END-PERFORM
- *abbruch ::=* WITH TEST ( BEFORE | AFTER )  
UNTIL *bedingung*
- Beispiel:  
PERFORM  
WITH TEST AFTER zeile=„...“ END  
ACCEPT zeile,  
DISPLAY zeile  
END-PERFORM.



# PERFORM - Anweisung

- *perform ::=* PERFORM *prozedurname*  
[THRU *prozedurname*][*wiederholung* | *abbruch*]

- Beispiel:

```
BEGIN.  
    PERFORM EINLESEN THRU AUSGEBEN END-PERFORM.  
    EINLESEN.  
        ACCEPT zeile  
    BERECHNEN.  
* Mach irgendetwas.  
    AUSGEBEN.  
        DISPLAY zeile,  
        STOP RUN.
```



# Zuweisung

- *zuweisung* ::= MOVE ( *literal* | *datename* ) TO *datename*
- *zuweisung* ::= MOVE CORRESPONDING *datensatz-name-1* TO *datensatz-name-2*
  - CORRESPONDING: gleichbezeichnete Felder der Sätze
- Beispiel:

```
WORKING-STORAGE SECTION.
```

```
01 englisches-datum.
```

```
    02 monat PIC 99.
```

```
    02 FILLER PIC X VALUE '/'.
```

```
    02 tag PIC 99.
```

```
    02 FILLER PIC X VALUE '/'.
```

```
    02 jahr PIC 99.
```

```
01 deutsches-datum.
```

```
    02 tag PIC 99.
```

```
    02 FILLER PIC X VALUE '.'.
```

```
    02 monat PIC 99.
```

```
    02 FILLER PIC X VALUE '.'.
```

```
    02 jahr PIC 99.
```



# Beispiel

```
PROCEDURE DIVISION.  
  ACCEPT englisches-datum.monat ,  
  ACCEPT englisches-datum.tag ,  
  ACCEPT englisches-datum.jahr .  
  MOVE CORRESPONDING  
    englisches-datum TO deutsches-datum.
```



# Arithmetische Operationen

- *compute* ::= COMPUTE [ *datename* ]\* =  
*arithmetischer ausdrück mit Operatoren*  
+ - \* / \*\*
- Aber:
  - Kaufleuten kann man Zuweisungen und Ausdrücke der Form  
 $a := a*b+c$   
nicht zumuten (Ansicht von 1960).
  - daher Lösung wie auf den nächsten Seiten



# Addition und Subtraktion

- *addition ::= ADD [literal | datenname ]<sup>+</sup>  
TO [datenname ]<sup>+</sup> [ GIVING datenname ]\**
- *addition ::= ADD CORRESPONDING  
datensatz-name-1 TO datensatz-name-2*
- *subtraktion ::= SUBTRACT [literal | datenname ]<sup>+</sup> FROM [datenname ]<sup>+</sup> [  
GIVING datenname ]\**
- *subtraktion ::= SUBTRACT CORRESPONDING  
datensatz-name-1 FROM datensatz-name-2*

ADD GIN TO VERMOUTH GIVING MARTINI



# Multiplikation und Division

- *multiplikation* ::= MULTIPLY [*literal* | *datenname*]<sup>+</sup> BY [*datenname*]<sup>+</sup> [GIVING *datenname* ]\*
- *multiplikation* ::= MULTIPLY CORRESPONDING *datensatz-name-1* BY *datensatz-name-2*
- *division* ::= DIVIDE (*literal* | *datenname*) (BY | INTO) *datenname* [GIVING *datenname*]\* [ REMAINDER *datenname*]



# Datei Ausgabe

- *öffnen* ::= OPEN *interner dateiname* [ *rechte* ]
- *schließen* ::= CLOSE *interner dateiname*
- *schreiben* ::= WRITE *datensatzname*  
[ FROM *datenname* ]  
[ INVALID KEY *anweisungen* ]  
[ NOT INVALID KEY *anweisungen* ]  
[ END-WRITE ]
- *überschreiben* ::= REWRITE ...  
[ END-REWRITE ]
- *löschen* ::= DELETE *dateiname* RECORD ...  
[ END-DELETE ]



# Datei Eingabe

- *lesen* ::= READ *dateiname* RECORD  
    [ INTO *datenname* ] [ KEY IS *schlüssel* ]  
    [ INVALID KEY *anweisungen* ]  
    [ NOT INVALID KEY *anweisungen* ]  
    [ AT END *anweisungen* ]  
    [ NOT AT END *anweisungen* ]  
    [ END-READ ]
- *positionieren* ::= START *dateiname* RECORD  
    [ KEY IS (EQUAL TO | GREATER THAN | ...) *schlüssel* ... ]  
    [ END-START ]



# Bedingungen

- *if* ::= IF *bedingung* THEN *anweisungen*  
[ ELSE *anweisungen* ] [ END-IF ]
- *bedingung* ::= ( *literal* | *datenname* )  
*vergleichsausdruck* ( *literal* | *datenname* )
- *bedingung* ::= *datenname* IS *klasse*
- *bedingung* ::= *bedingungsname*
- *case* ::= EVALUATE *ausdruck* [ ALSO *ausdruck* ]\*  
[ WHEN *ausdruck* [ ALSO *ausdruck* ]\* *anweisungen* ]\*  
[ WHEN OTHERS *anweisungen* ]  
[ END-EVALUATE ]



# Beispiel

```
01 FAMILIENSTAND PIC 9
```

```
88 LEDIG VALUE 1.
```

```
88 VERHEIRATET VALUE 2
```

```
...
```

```
IF LEDIG THEN ...
```

```
IF FAMILIENSTAND = 1 THEN ...
```

```
SET LEDIG TO TRUE.
```



# Weitere Steuerstrukturen

- *sprung* ::= GO TO  
    *prozedurname* [ [ *prozedurname* ]\*  
    DEPENDING ON *datenname* ]
- *ende* ::= STOP RUN
- Unterprogramme -
  - nicht im ursprünglichen Standard.
  - Später ohne Parameter.
  - Immer noch ohne Rekursion im 85-Standard, im 2002 Standard mit Rekursion.
- Formale Parameter in der LINKAGE-SECTION vereinbart,
- Kein Speicherplatz vorgesehen.
- Statische Bindung und Speicherzuteilung



# Unterprogramme

- *aufruf* ::= CALL (*unterprogramm* | *datenname*)  
[ USING [[*übergabe*] *aktuelle parameter*]\* ]
- *übergabe* ::= BY REFERENCE | BY CONTENT |  
BY CONTENT LENGTH
- *definition* ::= PROCEDURE-DIVISION  
[ *formaler parameter* ]\*



# Tabellen verarbeiten

```
Identification division. program-id. Tabellen-demo.
Environment division. ...
data division.
working-storage section.
01 kontakt-tabelle.
    02 kontakt occurs 100 ascending key is name vorname indexed by kontakt-idx.
        03 name pic X(30).
        03 vorname      pic X(30).
        03 kontakt-art pic 9.
            88 interner-kontakt value 1.
            88 externer-kontakt value 2.
        03 adresse occurs 3.
            04 strasse pic X(30).
            04 plz      pic XXXXX.
            04 ort      pic X(30).
01 temporaere-felder.
    02 such-ende pic 9.
        88 suche-lauft value 0.
        88 suche-beendet value 1.
    03 such-name pic X(30).
```



# Tabellen verarbeiten, Fortsetzung

```
procedure division.  
...  
move "Müller" to such-name.  
perform suche-alle-passenden.  
perform suche-alle-passenden-schneller.  
...  
suche-alle-passenden section.  
  set suche-laeuft to true.  
  set kontakt-idx to 1.  
  perform until suche-beendet.  
  search kontakt.  
    at end set suche-beendet to true.  
    when such-name = name  
      perform gefunden.  
    end search.  
exit.  
...
```

```
suche-alle-passenden-schneller  
section.  
  set suche-laeuft to true.  
  set kontakt-idx to 1.  
  perform until suche-beendet.  
  search all kontakt.  
    at end set suche-beendet to true.  
    when such-name = name  
      perform gefunden.  
    end search.  
exit.  
  
gefunden section.  
  display "Gefunden: " name " ,"  
  vorname.  
exit.
```



# Beispiel

(IBM 1965)

IDENTIFICATION DIVISION.

PROGRAM-ID. 'SORT360'.

AUTHOR. IBM.

DATE-WRITTEN. 1965.

DATE-COMPILED. UNKNOWN.

SECURITY. CONFIDENTIAL.

REMARKS. THIS PROGRAM WAS WRITTEN TO DEMONSTRATE THE USE OF THE SORT  
FEATURE. THIS PROGRAM PERFORMS THE FOLLOWING TASKS -

1. SELECTS, FROM A FILE OF 1000-CHARACTER RECORDS, THOSE  
RECORDS HAVING FIELD-A NOT EQUAL TO FIELD-B
2. EXTRACTS INFORMATION FROM THE SELECTED RECORDS.
3. SORTS THE SELECTED RECORDS INTO SEQUENCE, USING FIELD-AA,  
FIELD-BB, AND FIELD-CC AS SORT KEYS.
4. WRITES THOSE SORTED RECORDS HAVING FIELD-FF EQUAL TO  
FIELD-EE ON FILE-3 AND WRITES SELECTED DATA OF THE OTHER  
RECORDS ON FILE-2.



```
ENVIRONMENT DIVISION.  
CONFIGURATION SECTION.  
SOURCE-COMPUTER. IBM-360 F50.  
OBJECT-COMPUTER. IBM-360 F50.  
INPUT-OUTPUT SECTION.  
FILE-CONTROL. SELECT INPUT-FILE-1 ASSIGN TO 'F401' UTILITY.  
                SELECT SORT-FILE-1 ASSIGN 'SF1' UTILITY.  
                SELECT FILE-2 ASSIGN 'F402' UTILITY.  
                SELECT FILE-3 ASSIGN 'F403' UTILITY.
```



DATA DIVISION.

FILE SECTION.

FD INPUT-FILE-1 BLOCK CONTAINS 5 RECORDS  
RECORDING MODE IS F  
LABEL RECORDS ARE STANDARD  
DATA RECORD IS INPUT-RECORD.

01 INPUT-RECORD

02 FIELD-A PICTURE X (20).  
02 FIELD-C PICTURE 9 (10).  
02 FIELD-D PICTURE X (15).  
02 FILLER PICTURE X (900).  
02 FIELD-B PICTURE X (20).  
02 FIELD-E PICTURE 9 (5).  
02 FIELD-G PICTURE X (25).  
02 FIELD-F PICTURE 9 (5).



```
01 FILE-2-RECORD
    02 FIELD-EEE PICTURE $$$9.
    02 FILLER-A PICTURE 9 (2).
    02 FIELD-FFF PICTURE 9 (5).
    02 FILLER-B PICTURE X (2).
    02 FIELD-AAA PICTURE X (20).
    02 FIELD-BBB PICTURE X (20).
FD FILE-3 BLOCK CONTAINS 15 RECORDS
RECORDING MODE IS F
LABEL RECORDS ARE STANDARD
DATA RECORD IS FILE-3-RECORD.

01 FILE-3-RECORD PICTURE X(75).
```



```
SD  SORT-FILE-1 DATA RECORD IS SORT-RECORD.  
01  SORT-RECORD  
    02 FIELD-AA    PICTURE X (20).  
    02 FIELD-CC    PICTURE 9 (10).  
    02 FIELD-BB    PICTURE X (20).  
    02 FIELD-DD    PICTURE X (15).  
    02 FIELD-EE    PICTURE 9 (5).  
    02 FIELD-FF    PICTURE 9 (5).  
FD  FILE-2 BLOCK CONTAINS 10 RECORDS  
    RECORDING MODE IS F  
    LABEL RECORDS ARE STANDARD  
    DATA RECORD IS FILE-2-RECORD.
```



**PROCEDURE DIVISION.**

OPEN INPUT-FILE-1, OUTPUT FILE-2, FILE-3.  
SORT SORT-FILE-1 ASCENDING FIELD-AA, DESCENDING FIELD-BB,  
ASCENDING FIELD-CC INPUT PROCEDURE RECORD-SELECTION  
OUTPUT PROCEDURE PROCESS-SORTED-RECORDS.  
CLOSE INPUT-FILE-1, FILE-2, FILE-3.  
STOP RUN.

**RECORD-SELECTION SECTION.**

PARAGRAPH-1. READ INPUT-FILE-1 AT END GO TO PARAGRAPH-2.  
IF FIELD-A = FIELD-B GO TO PARAGRAPH-1 ELSE  
MOVE FIELD-A TO FIELD-AA  
MOVE FIELD-F TO FIELD-FF  
MOVE FIELD-C TO FIELD-CC  
MOVE FIELD-B TO FIELD-BB  
MOVE FIELD-D TO FIELD-DD  
MOVE FIELD E TO FIELD-EE  
RELEASE SORT-RECORD. GO TO PARAGRAPH-1.  
PARAGRAPH-2. EXIT.



PROCEDURE DIVISION.

OPEN INPUT-FILE-1, OUTPUT FILE-2, FILE-3.

SORT SORT-FILE-1 ASCENDING FIELD-AA, DESCENDING FIELD-BB,  
ASCENDING FIELD-CC INPUT PROCEDURE RECORD-SELECTION  
OUTPUT PROCEDURE PROCESS-SORTED-RECORDS.

CLOSE INPUT-FILE-1, FILE-2, FILE-3.

STOP RUN.

RECORD-SELECTION SECTION.

PARAGRAPH-1. READ INPUT-FILE-1 AT END GO TO PARAGRAPH-2.

IF FIELD-A = FIELD-B GO TO PARAGRAPH-1 ELSE

MOVE FIELD-A TO FIELD-AA

MOVE FIELD-F TO FIELD-FF

MOVE FIELD-C TO FIELD-CC

MOVE FIELD-B TO FIELD-BB

MOVE FIELD-D TO FIELD-DD

MOVE FIELD E TO FIELD-EE

RELEASE SORT-RECORD. GO TO PARAGRAPH-1.

PARAGRAPH-2. EXIT.

```
PROCESS-SORTED-RECORDS SECTION.  
PARAGRAPH-3. RETURN SORT-FILE-1 AT END GO TO PARAGRAPH-4.  
    IF FIELD-FF = FIELD-EE  
        WRITE FILE-3-RECORD FROM SORT-RECORD GO TO PARAGRAPH-3  
    ELSE  
        MOVE FIELD-EE TO FIELD-EEE  
        MOVE FIELD-FF TO FIELD-FFF  
        MOVE FIELD-AA TO FIELD-AAA  
        MOVE FIELD-BB TO FIELD-BBB  
        MOVE SPACES TO FILLER-A, FILLER-B  
        WRITE FILE-2-RECORD.  
        GO TO PARAGRAPH-3.  
PARAGRAPH-4. EXIT.
```

