CHAPTER 20

SORTING AND MERGING

CHAPTER OBJECTIVES

Upon completion of this chapter, you should be able to

- Explain how files may be sorted within a COBOL program.
- Explain how to process a file during a `SORT` procedure before it is actually sorted.
- Explain how to process a file during a `SORT` procedure after it is sorted but before it is created as output.
- Explain how to use the `MERGE` verb for merging files.

THE SORT STATEMENT

Records in files are usually sorted into specific sequences for updating, answering inquiries, or generating reports. A master payroll file, for example, might be updated in Social Security number sequence, whereas paychecks produced from the file may be needed in alphabetical order. *Sorting* is a common procedure used for arranging records into a specific sequence so that sequential processing can be performed.

COBOL has a `SORT` verb, which enables a file to be sorted as part of a COBOL program. Often, a COBOL program will `SORT` a file prior to processing it.

A simplified format for the `SORT` statement in COBOL is as follows:

Simplified Format

```
SORT file-name-1
    {ON {DESCENDING/ASCENDING} KEY data-name-1} . . .
     USING file-name-2
     GIVING file-name-3
```

ASCENDING OR DESCENDING KEY

The software developer must specify whether the key field is to be an `ASCENDING KEY` or a `DESCENDING KEY`, depending on which sequence is required. When `ASCENDING KEY` is specified in the sort statement, the records are sorted from lowest to highest value of the key field. When `DESCENDING KEY` is specified, sorting is from highest to lowest value.

Sorting a file into ascending department number sequence, for example, as in `ON ASCENDING KEY DEPARTMENT` where `DEPARTMENT` is defined with `PIC 9(3)`, would result in the following order: 001, 002, 003, and so on. The `SORT` can also be performed on nonconsecutive values of key fields. That is, records 009, 016, and 152 are sorted into their proper sequence even though they are not consecutive. The values of key fields do not have to be unique. Suppose several records had the same `DEPARTMENT`; *all* `DEPARTMENT 100` records would precede records with `DEPARTMENT 200`, and so on, if ascending sequence were specified.

A file can also be sorted into descending sequence, in which a key field of 500, for example, precedes 400, and so on.

Records may be sorted using numeric, alphabetic, or alphanumeric key fields. Ascending sequence used with an alphabetic field will cause sorting from A up to Z, and descending sequence will cause sorting from Z down to A.

COLLATING SEQUENCE

There are two major codes used for representing data in a computer. IBM mainframe and midrange computers use **EBCDIC** (Extended Binary Coded Decimal Interchange Code). All PCs use **ASCII** (American Standard Code for Information Interchange).

The sequencing of characters from lowest to highest, which is referred to as the **collating sequence**, is somewhat different between EBCDIC and ASCII:

	EBCDIC	**ASCII**
Lowest	b̸ (blank or space)	b̸ (blank or space)
|	Special characters	Special characters
|	Lowercase letters a–z	Integers 0-9
↓	Uppercase letters A–Z	Uppercase letters A–Z
Highest	Integers 0-9	Lowercase letters a–z

We have not included the collating sequence for the individual special characters here, because we rarely sort on special characters. See Appendix B for the collating sequence of all characters.

Basic numeric sorting and basic alphabetic sorting are performed the same way in EBCDIC and ASCII. These codes are, however, not the same when alphanumeric fields containing both letters and digits or special characters are sorted. Letters are considered "less than" numbers in EBCDIC but "greater than" numbers in ASCII. Moreover, lowercase letters are considered "less than" uppercase letters in EBCDIC but "greater than" uppercase letters in ASCII.

Thus, an ASCII computer could sort data into a different sequence than an EBCDIC computer if an alphanumeric field is being sorted or if a combination of upper- and lowercase letters is used. For example, "`Box 891`" appears before "`111 Main St.`" in an address field on EBCDIC computers but will appear *after* it on ASCII computers. Similarly, "`abc`" is less than "`ABC`" on EBCDIC computers, whereas the reverse is true of ASCII computers.

SEQUENCING RECORDS WITH MORE THAN ONE SORT KEY

The `SORT` verb may be used to sequence records *with more than one key field*. Suppose that we wish to sort the employee pay file so that it is in ascending alphabetic sequence by last name, within each department number, for each store number. That is:

Store number is the major sort field.
Department number is the intermediate sort field.
Last name is the minor sort field.

Thus for Store Number 100, we want the following sequence:

```
 STORE-NUMBER  DEPARTMENT-NUMBER  LAST-NAME
      100             111         ADAMS, J. R.
      100             111         BROCK, P. T.
      100             111         LEE, S.
      100             222         ARTHUR, Q. C.

      100             222         SHAH, J.
      100             333         RAMIREZ, A. P.
       .               .                 .
       .               .                 .
       .               .                 .
```

For Store Number 100, Department 111, records are in alphabetic order by last name. These are followed by Store Number 100, Department 222 entries in alphabetic order by last name, and so on.

We can use a *single* SORT procedure to perform this sequencing. The first KEY field indicated is the *major* field to be sorted; the next KEY fields represent *intermediate* sort fields, followed by *minor* sort fields.

The following is a SORT statement that sorts records into ascending alphabetic LAST-NAME sequence within DEPARTMENT-NUMBER within STORE-NUMBER:

```
SORT SORT-FILE
    ON ASCENDING KEY STORE-NUMBER       ← Major key
    ON ASCENDING KEY DEPARTMENT-NUMBER  ← Intermediate key
    ON ASCENDING KEY LAST-NAME          ← Minor key
        USING  EMPLOYEE-PAY-FILE
        GIVING SORTED-EMPLOYEE-PAY-FILE.
```

Because all key fields are independent, some key fields can be sorted in ASCENDING sequence and others in DESCENDING sequence. If all key fields are to be sorted in ascending sequence, as in the preceding, we can condense the coding by using the phrase ON ASCENDING KEY only once. For example:

```
SORT SORT-FILE
    ON ASCENDING KEY STORE-NUMBER       ← Major key
                     DEPARTMENT-NUMBER  ← Intermediate key
                     LAST-NAME          ← Minor key
        USING  EMPLOYEE-PAY-FILE
        GIVING SORTED-EMPLOYEE-PAY-FILE.
```

The words ON and KEY are optional words: ASCENDING STORE-NUMBER means the same as ON ASCENDING KEY STORE-NUMBER.

WITH DUPLICATES IN ORDER CLAUSE

With COBOL 74, if two or more records have the same value in the sort field (e.g., DEPARTMENT-NUMBER 111 in two or more records), you cannot predict which will appear first in the sorted file.

With COBOL 85, the SORT statement can include the WITH DUPLICATES IN ORDER clause that is used to records into the sort file *in the same order* that they appear in the original input file. The WITH DUPLICATES IN ORDER clause is added to the SORT statement to accomplish this:

```
SORT SORT-FILE
    ON ASCENDING KEY STORE-NUMBER
                     DEPARTMENT-NUMBER
    WITH DUPLICATES IN ORDER
        USING  EMPLOYEE-PAY-FILE
        GIVING SORTED-EMPLOYEE-PAY-FILE.
```

This means that if, for example, both the 106th record and the 428th record in the input file have `DEPARTMENT-NUMBER 111`, then record 106 would appear first in the sorted file. This is called the first in, first out (**FIFO**) principle.

CODING A SIMPLE SORT PROCEDURE WITH THE USING AND GIVING OPTIONS

There are three major files used in a sort:

1. Input file: File of unsorted input records.
2. Work or sort file: File used to store records temporarily during the sorting process.
3. Output file: File of sorted output records.

The input and output disk files are defined in the `ENVIRONMENT DIVISION` using standard `ASSIGN` clauses, which are system dependent. The `ASSIGN` clause is optional for the sort file because it is a temporary system work file:

```
SELECT EMPLOYEE-PAY-FILE ASSIGN TO EMPPAYPF.
SELECT SORT-FILE.
SELECT SORTED-EMPLOYEE-PAY-FILE ASSIGN TO EMPPAYPFS.
```

The `SORT-FILE` is actually assigned to a temporary work area that is used during processing but not saved. Only the unsorted disk file and the sorted output disk file are assigned standard file-names so that they can be permanently stored.

`FD`s are used in the `DATA DIVISION` to define and describe the input and output files in the usual way. The sort or work file in Figure 20.1 is described with an `SD` (sort file *d*escription) entry. The only difference between `SD` and `FD` entries is that an `SD` must *not* have a `LABEL RECORDS` clause. Note, too, that the field(s) specified as the `KEY` field(s) for sorting purposes must be defined *as part of the sort record format*. In Figure 20.1, the fields to be sorted are `SORT-STORE-NUMBER` and `SORT-DEPARTMENT` within the `SD` file called `SORT-FILE`.

```
*A   B    1    2    2    2    3    3    4    4    4    5    5    6    6    6    7
789012345678901234567890123456789012345678901234567890123456789012
 DATA DIVISION.

 FILE SECTION.

 FD   EMPLOYEE-PAY-FILE.
 01   EMPLOYEE-PAY-RECORD             PIC X(55).

 FD   SORTED-EMPLOYEE-PAY-FILE.
 01   SORTED-EMPLOYEE-PAY-RECORD      PIC X(55).

 SD   SORT-FILE.      ← Note that the sort file is defined
 01   SORT-RECORD.      with an SD.
      05                              PIC X(9).
      05  SORT-STORE-NUMBER           PIC 9(4).
      05                              PIC X(31).
      05  SORT-DEPARTMENT             PIC 9(3).
      05                              PIC X(8).
```

Figure 20.1 Defining a sort file with an ***SD*** *entry.*

The SORT procedure would then be coded as shown in Figure 20.2.

```
*A   B    1    2    2    2    3    3    4    4    4    5    5    6    6    6    7
789012345678901234567890123456789012345678901234567890123456789012
      SORT SORT-FILE
          ON ASCENDING KEY SORT-STORE-NUMBER ─┐ ← Defined within
                           SORT-DEPARTMENT   ─┘   the SD file
              WITH DUPLICATES IN ORDER
          USING            EMPLOYEE-PAY-FILE
          GIVING           SORTED-EMPLOYEE-PAY-FILE.
```

Figure 20.2 ***SORT*** *procedure that uses* ***USING*** *and* ***GIVING*** *clauses.*

The SORT statement in Figure 20.2 performs the following operations:

1. Opens EMPLOYEE-PAY-FILE and SORTED-EMPLOYEE-PAY-FILE.
2. Moves the records from the EMPLOYEE-PAY-FILE to the SORT-FILE.
3. Sorts SORT-FILE into ascending sequence by SORT-DEPARTMENT within SORT-STORE-NUMBER, which are fields defined as part of the SD SORT-FILE record.
4. Moves the sorted SORT-FILE to the output file called SORTED-EMPLOYEE-PAY-FILE.
5. Closes EMPLOYEE-PAY-FILE and SORTED-EMPLOYEE-PAY-FILE after all records have been processed.

The only field descriptions required in the sort record format are the ones used for sorting purposes. In this instance, only the SORT-STORE-NUMBER and SORT-DEPARTMENT must be defined as part of the SD, since they are the only key fields to be used for sorting.

With COBOL 85 the word GIVING can be followed by more than one file-name, which means that we can create multiple copies of the sorted file.

The EMPLOYEE-PAY-FILE and the SORTED-EMPLOYEE-PAY-FILE contain records with the same format, but the records are stored in the sorted master pay file in order of department number within store number.

COBOL program CPCH20A in Figure 20.3 provides the complete COBOL 85 example that utilizes the USING and GIVING clauses.

```
       *A   B   1   2   2   2   3   3   4   4   4   5   5   6   6   6   7
       789012345678901234567890123456789012345678901234567890123456789012
000100 PROCESS APOST.
000200
000300 IDENTIFICATION DIVISION.
000400
000500 PROGRAM-ID. CPCH20A.
000600
000700 AUTHOR.     Jill Programmer.
000800
000900*****************************************************************
001000*                                                               *
001100*  This program reads records from the Employee Pay file,       *
001200*  sorts the records by Department number within Store number,  *
001300*  and outputs a file with the sorted records.                  *
001400*                                                               *
001500*  The SORT statement uses USING and GIVING clauses.            *
001600*                                                               *
001700*****************************************************************
001800
001900 ENVIRONMENT DIVISION.
002000
002100 INPUT-OUTPUT SECTION.
002200
002300 FILE-CONTROL.
002400     SELECT EMPLOYEE-PAY-FILE
002500         ASSIGN TO DISK-EMPPAYPF.
002600
002700     SELECT SORTED-EMPLOYEE-PAY-FILE
002800         ASSIGN TO DISK-EMPPAYPFS.
002900
003000     SELECT SORT-FILE.
003100
003200 DATA DIVISION.
003300
003400 FILE SECTION.
003500
003600 FD  EMPLOYEE-PAY-FILE.
003700 01  EMPLOYEE-PAY-RECORD             PIC X(55).
003800
003900 FD  SORTED-EMPLOYEE-PAY-FILE.
004000 01  SORTED-EMPLOYEE-PAY-RECORD      PIC X(55).
004100
004200 SD  SORT-FILE.
004300 01  SORT-RECORD.
004400     05                              PIC X(9).
004500     05  SORT-STORE-NUMBER           PIC 9(4).
004600     05                              PIC X(31).
004700     05  SORT-DEPARTMENT             PIC 9(3).
004800     05                              PIC X(8).
004900
005000 WORKING-STORAGE SECTION.
005100
005200 PROCEDURE DIVISION.
005300
005400 000-MAIN-MODULE.
005500
005600     SORT SORT-FILE
005700         ON ASCENDING KEY SORT-STORE-NUMBER
005800                          SORT-DEPARTMENT
005900             WITH DUPLICATES IN ORDER
006000         USING            EMPLOYEE-PAY-FILE
006100         GIVING           SORTED-EMPLOYEE-PAY-FILE.
006200
006300     STOP RUN.
```

Figure 20.3 Solution for program CPCH20A.

PROCESSING DATA BEFORE AND/OR AFTER SORTING

The SORT statement can be used in conjunction with procedures that process records *before they are sorted, after they are sorted,* or both.

INPUT PROCEDURE

The **`INPUT PROCEDURE`** clause is used *in place of* the `USING` clause to perform some processing of incoming records *prior* to sorting.

SORT Format for INPUT PROCEDURE

```
SORT file-name-1
    {ON {ASCENDING/DESCENDING} KEY data-name-1 . . .} . . .

    {INPUT PROCEDURE IS procedure-name-1
           [{THRU/{THROUGH} procedure-name-2]}

    GIVING file-name-3
```

For example, an `INPUT PROCEDURE` could be used prior to sorting to: (1) validate data in the input records, (2) eliminate records with blank fields, (3) remove unwanted fields from the input records, or (4) count input records.

Let us consider a `SORT` routine that eliminates records *before sorting*. In this example, we wish to create a sorted output file containing employee records where the sales field is greater than $5,000.00. The test on the sales field is performed in an `INPUT PROCEDURE`. The first three `DIVISION`s for this COBOL program are shown in Figure 20.4.

```
*A   B   1   2   2   2   3   3   4   4   4   5   5   6   6   6   7
7890123456789012345678901234567890123456789012345678901234567890012
 IDENTIFICATION DIVISION.

 PROGRAM-ID. CPCH20B74.

 AUTHOR.     Jill Programmer.

 ENVIRONMENT DIVISION.

 INPUT-OUTPUT SECTION.

 FILE-CONTROL.
     SELECT EMPLOYEE-PAY-FILE
         ASSIGN TO DISK-EMPPAYPF.

     SELECT SORTED-EMPLOYEE-PAY-FILE
         ASSIGN TO DISK-EMPPAYPFS.

     SELECT SORT-FILE.

 DATA DIVISION.

 FILE SECTION.

 FD  EMPLOYEE-PAY-FILE.
 01  EMPLOYEE-PAY-RECORD.     Needed for INPUT PROCEDURE section
     05                                PIC X(52).
     05  EP-SALES                      PIC 9(5)    PACKED-DECIMAL.

 FD  SORTED-EMPLOYEE-PAY-FILE.
 01  SORTED-EMPLOYEE-PAY-RECORD     PIC X(55).

 SD  SORT-FILE.
 01  SORT-RECORD.                 Needed for ASCENDING KEY clause
     05                                PIC X(9).
     05  SORT-STORE-NUMBER             PIC 9(4).
     05                                PIC X(31).
     05  SORT-DEPARTMENT               PIC 9(3).
     05                                PIC X(8).

 WORKING-STORAGE SECTION.

 01  WS-CONTROL-FIELDS.
     05  ARE-THERE-MORE-RECORDS        PIC X(3)    VALUE 'YES'.
         88  NO-MORE-RECORDS                       VALUE 'NO '.
```

Figure 20.4 First three divisions of COBOL SORT program.

INPUT PROCEDUREs are coded in an easier and more structured way with COBOL 85. We discuss here techniques used for both COBOL 85 and COBOL 74.

As shown in Figure 20.5, with COBOL 85, procedure-names used with INPUT PROCEDURE can be regular paragraphs.

The 200-SELECT-INPUT-RECORDS-RTN paragraph must:

1. Open the input file. (With a USING option instead of the INPUT PROCEDURE, the input file is automatically opened by the SORT verb.)
2. Perform some processing of input records until there is no more data.
3. Close the input file.

```
*A   B    1    2    2    2    3    3    4    4    4    5    5    6    6    6    7
7890123456789012345678901234567890123456789012345678901234567890123456789012
 PROCEDURE DIVISION.

 000-MAIN-MODULE.

     SORT SORT-FILE
         ON ASCENDING KEY SORT-STORE-NUMBER
                          SORT-DEPARTMENT
             WITH DUPLICATES IN ORDER
         INPUT PROCEDURE  200-SELECT-INPUT-RECORDS-RTN
         GIVING           SORTED-EMPLOYEE-PAY-FILE.

     STOP RUN.

 200-SELECT-INPUT-RECORDS-RTN.

     OPEN INPUT EMPLOYEE-PAY-FILE.

     PERFORM UNTIL NO-MORE-RECORDS
         READ EMPLOYEE-PAY-FILE
             NOT AT END
                 PERFORM 210-PROCESS-RECORD-RTN
             AT END
                 SET NO-MORE-RECORDS TO TRUE
         END-READ
     END-PERFORM

     CLOSE EMPLOYEE-PAY-FILE.

 210-PROCESS-RECORD-RTN.
     IF EP-SALES > 5000.00
          MOVE EMPLOYEE-PAY-RECORD TO SORT-RECORD
          RELEASE SORT-RECORD
     END-IF
```

Figure 20.5 Procedure-names used with ***INPUT PROCEDURE*** *for COBOL 85 program.*

Paragraph 210-PROCESS-RECORD-RTN is executed from 200-SELECT-INPUT-RECORDS-RTN as long as there are more records to process. In 210-PROCESS-RECORD-RTN, for all records where EP-SALES is greater than $5,000.00, the input fields are moved to the sort record. We do not WRITE records to be sorted; instead, we RELEASE them for sorting purposes. We must release records to the sort file in an INPUT PROCEDURE. With a USING option, this is done for us automatically.

Note that the RELEASE verb is followed by a record-name, just like the WRITE statement. Note, too, that

```
RELEASE SORT-RECORD
    FROM EMPLOYEE-PAY-RECORD
```

can be substituted for

```
MOVE EMPLOYEE-PAY-RECORD TO SORT-RECORD
RELEASE SORT-RECORD
```

That is, the RELEASE verb functions just like a WRITE but is used to output sort records.

USING SECTION-NAMES WITH COBOL 74

With COBOL 74, the coding is more complex, because the procedure-name of an INPUT PROCEDURE must be a section-name and not a paragraph-name. A **section** is a series of PROCEDURE DIVISION paragraphs that is treated as a single entity or unit. Rules for forming section-names are the same as rules for forming paragraph-names. The word SECTION, however, follows a section-name (e.g., A000-MAIN SECTION). The end of a section is recognized when another section-name is encountered or when the end of the program is reached.

In Figure 20.6, the INPUT PROCEDURE identifies a separate section in which records are read and selected for processing by testing the sales field for a value greater than $5,000.00:

```
*A   B   1   2   2   2   3   3   4   4   4   5   5   6   6   6   7
7890123456789012345678901234567890123456789012345678901234567890123456789012
 PROCEDURE DIVISION.

 A000-MAIN SECTION.

 A100-MAIN-MODULE.

     SORT SORT-FILE
         ON ASCENDING KEY SORT-STORE-NUMBER
                          SORT-DEPARTMENT
             WITH DUPLICATES IN ORDER
         INPUT PROCEDURE  B000-SELECT-INPUT-RECORDS ← This must be a
         GIVING           SORTED-EMPLOYEE-PAY-FILE.    section with
                                                       COBOL 74.
     STOP RUN.

 B000-SELECT-INPUT-RECORDS SECTION.

 B200-SELECT-INPUT-RECORDS-RTN. ┐ In this section, the instructions
     .                          │ that eliminate records with sales
     .                          │ > $5,000.00 are executed.
     .                          ┘
```

Figure 20.6 Using section-names with COBOL 74.

A section-name must conform to the rules for forming paragraph-names. The COBOL reserved word SECTION follows the actual section-name. In Figure 20.6, A000-MAIN SECTION is executed first, *before the input file is sorted*. Then the input file is sorted in the main module, producing a sorted file called SORTED-EMPLOYEE-PAY-FILE. After the sorted records have been created as output, the STOP RUN is executed, terminating the program run.

<u>More about Sections and Naming Conventions</u>

A procedure may be a paragraph or section in the PROCEDURE DIVISION. In all programs thus far, we have used paragraphs as procedures. In this chapter we discuss sections *that can consist of one or more paragraphs*. The PROCEDURE DIVISION, then, can be divided into individual paragraphs or into sections, where each section contains one or more paragraphs.

With COBOL 74, the clause INPUT PROCEDURE IS procedure-name-1 *must refer to a section*. With COBOL 85, this INPUT PROCEDURE may reference either a section or a paragraph. With COBOL 85, using a paragraph-name results in less complex programs that are better structured.

Naming Procedures
When a program is subdivided into sections, we use a more detailed numbering convention for prefixes of paragraphs. This convention will highlight the fact that each paragraph is located within a particular section. A SECTION named A000 or with a prefix of A000- is followed by a paragraph with a prefix of A100-, A200-, and so on, with the letter A designating the first SECTION.

Example

```
PROCEDURE DIVISION.
A000-MAIN SECTION.
A100-PARA-1.
    .
    .
    .
A200-PARA-2.
    .
    .
    .
B000 SECTION.
B100-PARA-1.
    .
    .
    .
B200-PARA-2.
    .
    .
    .
```

The A000-MAIN SECTION has paragraphs with prefixes of A100, A200, and so on. Similarly, a section called A000-MAIN SECTION can be followed by a paragraph called A100-SELECT-RECORDS-RTN, then an A200-PROCESS-RECORDS-RTN paragraph, and so on.

Another naming convention is to use four digits, with no letters, as a numeric prefix. Sections could have prefixes 0000-, 1000-, 2000-, and so on. Paragraphs within section 0000- would have a prefix of 0100-, 0200-, and so on. These are just two of the conventions you could adopt for prefixes of procedure names.

As noted, in our example we wish to sort only those input records in which sales is greater than $5,000.00. Figure 20.7 represents the COBOL 74 coding required within the section specified by the INPUT PROCEDURE section-name.

Paragraph B200-SELECT-INPUT-RECORDS-RTN within B000-SELECT-INPUT-RECORDS SECTION functions like a main module. It opens the input file, performs an initial read, continually executes a paragraph that processes records until there is no more input, and closes the input file after all records have been processed.

B210-PROCESS-RECORD-RTN within B000-SELECT-INPUT-RECORDS SECTION is the paragraph that processes input records. Records with sales greater than $5,000.00 are released to the sort file in this paragraph.

Keep in mind that when using an INPUT PROCEDURE with COBOL 74 we divide our program into sections:

1. The first section in the PROCEDURE DIVISION contains the SORT instruction, any processing to be performed before or after the SORT verb is executed, and a STOP RUN.
2. The second section begins with the main module of the INPUT PROCEDURE. It opens the input file, reads the first record, and then performs a processing rou-

tine (in a separate paragraph within this second section) until there is no more data.

3. After the separate paragraph is executed until `ARE-THERE-MORE-RECORDS = 'NO  '`, control returns to the second section (`B200-SELECT-INPUT-RECORDS-RTN`). The input file is then closed. In order for this section to be terminated, control must pass to the *last statement* within the section. This means that a `GO TO` is required as the last sentence of the first paragraph. We code `GO TO B000-EXIT`. Since no operations are required in this last paragraph, `EXIT` is coded, which passes control back to the `SORT` statement, where the file is then sorted.

```
*A   B   1   2   2   2   3   3   4   4   4   5   5   6   6   6   7
7890123456789012345678901234567890123456789012345678901234567890123456789012
           .
           .
           SORT SORT-FILE
               ON ASCENDING KEY SORT-STORE-NUMBER
                                SORT-DEPARTMENT
                   WITH DUPLICATES IN ORDER
               INPUT PROCEDURE  B000-SELECT-INPUT-RECORDS
               GIVING           SORTED-EMPLOYEE-PAY-FILE.

           STOP RUN.

       B000-SELECT-INPUT-RECORDS SECTION.

       B200-SELECT-INPUT-RECORDS-RTN. ← On many systems, a paragraph-name
                                        must follow a section-name
           OPEN INPUT EMPLOYEE-PAY-FILE.

           READ EMPLOYEE-PAY-FILE
               AT END
                   MOVE 'NO ' TO ARE-THERE-MORE-RECORDS.

           PERFORM B210-PROCESS-RECORD-RTN
               UNTIL NO-MORE-RECORDS.

           CLOSE EMPLOYEE-PAY-FILE.

           GO TO B000-EXIT.

       B210-PROCESS-RECORD-RTN.

           IF EP-SALES > 5000.00
                MOVE EMPLOYEE-PAY-RECORD TO SORT-RECORD
                RELEASE SORT-RECORD. ← Writes the record to the sort file.

           READ EMPLOYEE-PAY-FILE
               AT END
                   MOVE 'NO ' TO ARE-THERE-MORE-RECORDS.

       B000-EXIT.
           EXIT. ← The last statement in the section must
                   be the last statement executed.
```

Figure 20.7 Using sections with COBOL 74 programs..

Regardless of whether you are using COBOL 85 or COBOL 74, an `INPUT PROCEDURE` opens the input file, processes input records, and releases them to the sort file. After all input records are processed, the input file is closed. The format for the `RELEASE` is

Format

```
RELEASE sort-record-name-1
    [FROM identifier-1]
```

The `RELEASE` is the verb used to write records to a sort file.

Examples

```
MOVE MASTER-RECORD TO SORT-RECORD.
RELEASE SORT-RECORD.
```

or

```
RELEASE SORT-RECORD FROM MASTER-RECORD. ← Functions like a
                                          WRITE ... FROM
```

INPUT PROCEDURE Summary (COBOL 85)

1. The `INPUT PROCEDURE` of the `SORT` should refer to a paragraph-name, but it could refer to a section-name.

   Example

```
000-MAIN-MODULE.

    SORT SORT-FILE
        ON ASCENDING KEY SORT-STORE-NUMBER
                         SORT-DEPARTMENT
            WITH DUPLICATES IN ORDER
        INPUT PROCEDURE  200-SELECT-INPUT-RECORDS-RTN
        GIVING           SORTED-EMPLOYEE-PAY-FILE.

    STOP RUN.

200-SELECT-INPUT-RECORDS-RTN.
   .
   .
```

2. In the paragraph specified in the `INPUT PROCEDURE`:
   a. `OPEN` the input file.
   b. `PERFORM` a paragraph that will read and process input records until there is no more data.
   c. After all records have been processed, `CLOSE` the input file.
   d. After the last sentence in the `INPUT PROCEDURE` paragraph is executed, control will then return to the `SORT`.

   Example

```
200-SELECT-INPUT-RECORDS-RTN.

    OPEN INPUT EMPLOYEE-PAY-FILE.

    PERFORM UNTIL NO-MORE-RECORDS
        READ EMPLOYEE-PAY-FILE
            NOT AT END
                PERFORM 210-PROCESS-RECORD-RTN
            AT END
                SET NO-MORE-RECORDS TO TRUE
        END-READ
    END-PERFORM

    CLOSE EMPLOYEE-PAY-FILE.
```

3. At the paragraph that processes input records prior to sorting:
   a. Perform any operations on input that are required.
   b. `MOVE` input data to the sort record.
   c. `RELEASE` each sort record, which makes it available for sorting.
   d. Continue to read input until there is no more data.

Example

```
210-PROCESS-RECORD-RTN.
    IF EP-SALES > 5000.00
         MOVE EMPLOYEE-PAY-RECORD TO SORT-RECORD
         RELEASE SORT-RECORD
    END-IF
```

Figure 20.8 *Summary of* ***INPUT PROCEDURE*** *for COBOL 85.*

INPUT PROCEDURE Summary: COBOL 74

1. The entire program should consist of sections. Each section is followed by a paragraph-name. The `INPUT PROCEDURE` of the `SORT` refers to a section-name followed by a paragraph-name.

   Example

```
A000-MAIN SECTION.  ┐ Start the program with a section-name
                    │ followed by a paragraph-name
A100-MAIN-MODULE.   ┘

    SORT SORT-FILE
        ON ASCENDING KEY SORT-STORE-NUMBER
                         SORT-DEPARTMENT
            WITH DUPLICATES IN ORDER
        INPUT PROCEDURE  B000-SELECT-INPUT-RECORDS
        GIVING           SORTED-EMPLOYEE-PAY-FILE.

    STOP RUN.

B000-SELECT-INPUT-RECORDS SECTION.  ┐ Section-names are followed
                                    │ by paragraph-names
B100-SELECT-INPUT-RECORDS-RTN.      ┘
```

2. In the main paragraph of the section specified in the `INPUT PROCEDURE`:
   a. `OPEN` the input file.
   b. `READ` an initial record from the input file.
   c. `PERFORM` a paragraph within the `INPUT PROCEDURE` section that will process input records, release them to the sort file, and continue to read records until there is no more data.
   d. After all records have been processed, `CLOSE` the input file.
   e. With COBOL 74, in order for the `INPUT PROCEDURE` to be terminated, the last statement in the last paragraph of the section must be executed. We must use a `GO TO` for branching to the paragraph that contains this last statement.

   Example

```
B000-SELECT-INPUT-RECORDS SECTION.

B200-SELECT-INPUT-RECORDS-RTN.

    OPEN INPUT EMPLOYEE-PAY-FILE.

    READ EMPLOYEE-PAY-FILE
        AT END
            MOVE 'NO ' TO ARE-THERE-MORE-RECORDS.

    PERFORM B210-PROCESS-RECORD-RTN
        UNTIL NO-MORE-RECORDS.

    CLOSE EMPLOYEE-PAY-FILE.
    GO TO B000-EXIT.
```

3. At the paragraph within the `INPUT PROCEDURE` section that processes input records prior to sorting:
   a. Perform any operations on input that are required.
   b. `MOVE` input data to the sort record.
   c. `RELEASE` each sort record, which writes the record to the sort file. This `RELEASE` makes the record available for sorting. `RELEASE ... FROM...` can be used in place of a `MOVE` and `RELEASE`.
   d. Continue to read and process input until there is no more data.

Example

```
B210-PROCESS-RECORD-RTN.

    IF EP-SALES > 5000.00                           ┐ Process
         MOVE EMPLOYEE-PAY-RECORD TO SORT-RECORD    │  input
         RELEASE SORT-RECORD.                       ┘   records

    READ EMPLOYEE-PAY-FILE
        AT END
            MOVE 'NO ' TO ARE-THERE-MORE-RECORDS.
```

4. As noted, the paragraph located physically at the end of the `INPUT PROCEDURE` section must be the last one executed with COBOL 74. Hence a `GO TO` in the section's main module is required to transfer control to this last paragraph. If no processing is required, code an `EXIT` statement as the only entry in this last paragraph of the section.

Example

```
    GO TO B000-EXIT.
    .
    .
B000-EXIT.
    EXIT. ← Must be the only entry in the paragraph
            for COBOL 74
```

Figure 20.9 *Summary of* ***`INPUT PROCEDURE`*** *for COBOL 74.*

Figure 20.10 illustrates the complete COBOL 85 program with an `INPUT PROCEDURE` that selects only those employees that have sales greater than $5,000.00. Figure 20.11 represents the same program using COBOL 74.

```
       *A   B   1   2   2   2   3   3   4   4   4   5   5   6   6   6   7
       789012345678901234567890123456789012345678901234567890123456789012
000100 PROCESS APOST.
000200
000300 IDENTIFICATION DIVISION.
000400
000500 PROGRAM-ID. CPCH20B85.
000600
000700 AUTHOR.     Jill Programmer.
000800
000900*****************************************************************
001000*  COBOL 85 Version.                                             *
001100*                                                                *
001200*  This program reads records from the Employee Pay file,        *
001300*  selects those employees with sales greater than $5,000.00,   *
001400*  sorts the records by Department number within Store number,  *
001500*  and outputs a file with the sorted records.                  *
001600*                                                                *
001700*  The SORT statement uses INPUT PROCEDURE and GIVING clauses.  *
001800*                                                                *
001900*****************************************************************
```

```
002000
002100 ENVIRONMENT DIVISION.
002200
002300 INPUT-OUTPUT SECTION.
002400
002500 FILE-CONTROL.
002600     SELECT EMPLOYEE-PAY-FILE
002700         ASSIGN TO DISK-EMPPAYPF.
002800
002900     SELECT SORTED-EMPLOYEE-PAY-FILE
003000         ASSIGN TO DISK-EMPPAYPFS.
003100
003200     SELECT SORT-FILE.
003300
003400 DATA DIVISION.
003500
003600 FILE SECTION.
003700
003800 FD  EMPLOYEE-PAY-FILE.
003900 01  EMPLOYEE-PAY-RECORD.
004000     05                               PIC X(52).
004100     05  EP-SALES                     PIC 9(5)    PACKED-DECIMAL.
004200
004300 FD  SORTED-EMPLOYEE-PAY-FILE.
004400 01  SORTED-EMPLOYEE-PAY-RECORD      PIC X(55).
004500
004600 SD  SORT-FILE.
004700 01  SORT-RECORD.
004800     05                               PIC X(9).
004900     05  SORT-STORE-NUMBER            PIC 9(4).
005000     05                               PIC X(31).
005100     05  SORT-DEPARTMENT              PIC 9(3).
005200     05                               PIC X(8).
005300
005400 WORKING-STORAGE SECTION.
005500
005600 01  WS-CONTROL-FIELDS.
005700     05  ARE-THERE-MORE-RECORDS  PIC X(3)    VALUE 'YES'.
005800         88  NO-MORE-RECORDS                 VALUE 'NO '.
005900
006000 PROCEDURE DIVISION.
006100
006200 000-MAIN-MODULE.
006300
006400     SORT SORT-FILE
006500         ON ASCENDING KEY SORT-STORE-NUMBER
006600                          SORT-DEPARTMENT
006700             WITH DUPLICATES IN ORDER
006800         INPUT PROCEDURE  200-SELECT-INPUT-RECORDS-RTN
006900         GIVING           SORTED-EMPLOYEE-PAY-FILE.
007000
007100     STOP RUN.
007200
007300 200-SELECT-INPUT-RECORDS-RTN.
007400
007500     OPEN INPUT EMPLOYEE-PAY-FILE.
007600
007700     PERFORM UNTIL NO-MORE-RECORDS
007800         READ EMPLOYEE-PAY-FILE
007900             NOT AT END
008000                 PERFORM 210-PROCESS-RECORD-RTN
008100             AT END
008200                 SET NO-MORE-RECORDS TO TRUE
008300         END-READ
008400     END-PERFORM
008500
008600     CLOSE EMPLOYEE-PAY-FILE.
008700
008800 210-PROCESS-RECORD-RTN.
008900     IF EP-SALES > 5000.00
009000          MOVE EMPLOYEE-PAY-RECORD TO SORT-RECORD
009100          RELEASE SORT-RECORD
009200     END-IF
```

Figure 20.10   Solution for program CPCH20B85.

```
      *A   B   1   2   2   2   3   3   4   4   4   5   5   6   6   6   7
      789012345678901234567890123456789012345678901234567890123456789012
000100 PROCESS APOST.
000200
000300 IDENTIFICATION DIVISION.
000400
000500 PROGRAM-ID. CPCH20B74.
000600
000700 AUTHOR.     Jill Programmer.
000800
000900*****************************************************************
001000*  COBOL 74 Version using a section-name.                       *
001100*                                                               *
001200*  This program reads records from the Employee Pay file,       *
001300*  selects those employees with sales greater than $5,000.00,   *
001400*  sorts the records by Department number within Store number,  *
001500*  and outputs a file with the sorted records.                  *
001600*                                                               *
001700*  The SORT statement uses INPUT PROCEDURE and GIVING clauses. *
001800*                                                               *
001900*****************************************************************
002000
002100 ENVIRONMENT DIVISION.
002200
002300 INPUT-OUTPUT SECTION.
002400
002500 FILE-CONTROL.
002600     SELECT EMPLOYEE-PAY-FILE
002700         ASSIGN TO DISK-EMPPAYPF.
002800
002900     SELECT SORTED-EMPLOYEE-PAY-FILE
003000         ASSIGN TO DISK-EMPPAYPFS.
003100
003200     SELECT SORT-FILE.
003300
003400 DATA DIVISION.
003500
003600 FILE SECTION.
003700
003800 FD  EMPLOYEE-PAY-FILE.
003900 01  EMPLOYEE-PAY-RECORD.
004000     05                              PIC X(52).
004100     05  EP-SALES                    PIC 9(5)    PACKED-DECIMAL.
004200
004300 FD  SORTED-EMPLOYEE-PAY-FILE.
004400 01  SORTED-EMPLOYEE-PAY-RECORD    PIC X(55).
004500
004600 SD  SORT-FILE.
004700 01  SORT-RECORD.
004800     05                              PIC X(9).
004900     05  SORT-STORE-NUMBER           PIC 9(4).
005000     05                              PIC X(31).
005100     05  SORT-DEPARTMENT             PIC 9(3).
005200     05                              PIC X(8).
005300
005400 WORKING-STORAGE SECTION.
005500
005600 01  WS-CONTROL-FIELDS.
005700     05  ARE-THERE-MORE-RECORDS      PIC X(3)    VALUE 'YES'.
005800         88  NO-MORE-RECORDS                     VALUE 'NO '.
005900
006000 PROCEDURE DIVISION.
006100
006200 A000-MAIN SECTION.
006300
006400 A100-MAIN-MODULE.
006500
006600     SORT SORT-FILE
006700         ON ASCENDING KEY SORT-STORE-NUMBER
006800                          SORT-DEPARTMENT
006900             WITH DUPLICATES IN ORDER
007000         INPUT PROCEDURE  B000-SELECT-INPUT-RECORDS
007100         GIVING           SORTED-EMPLOYEE-PAY-FILE.
007200
007300     STOP RUN.
007400
007500 B000-SELECT-INPUT-RECORDS SECTION.
007600
```

```
007700 B200-SELECT-INPUT-RECORDS-RTN.
007800
007900     OPEN INPUT EMPLOYEE-PAY-FILE.
008000
008100     READ EMPLOYEE-PAY-FILE
008200         AT END
008300             MOVE 'NO ' TO ARE-THERE-MORE-RECORDS.
008400
008500     PERFORM B210-PROCESS-RECORD-RTN
008600         UNTIL NO-MORE-RECORDS.
008700
008800     CLOSE EMPLOYEE-PAY-FILE.
008900
009000     GO TO B000-EXIT.
009100
009200 B210-PROCESS-RECORD-RTN.
009300
009400     IF EP-SALES > 5000.00
009500          MOVE EMPLOYEE-PAY-RECORD TO SORT-RECORD
009600          RELEASE SORT-RECORD.
009700
009800     READ EMPLOYEE-PAY-FILE
009900         AT END
010000             MOVE 'NO ' TO ARE-THERE-MORE-RECORDS.
010100
010200 B000-EXIT.
010300     EXIT.
```

Figure 20.11 Solution for program CPCH20B74.

Note that some enhanced versions of COBOL 74 are like COBOL 85 in that they permit you to use paragraph-names in place of section-names in an INPUT PROCEDURE. This not only makes programming easier but also eliminates the need for GO TOs.

Note, too, that we never OPEN or CLOSE the sort file-name specified in the SD. It is always opened and closed automatically, as are files specified with USING or GIVING. Only the input file processed in an INPUT PROCEDURE needs to be opened and closed by the program. In the next section, we will see that output files processed in an OUTPUT PROCEDURE must also be opened and closed.

OUTPUT PROCEDURE

After records have been sorted, they are placed in the sort file in the sequence required. If the GIVING option is used, then the sorted records are automatically written to the output file after they are sorted.

We may, however, wish to process the sorted records *prior* to, or perhaps even instead of, placing them in the output file. We would then use an OUTPUT PROCEDURE instead of the GIVING option. This OUTPUT PROCEDURE is very similar to the INPUT PROCEDURE. The full format for the SORT, including both INPUT and OUTPUT PROCEDURE options, is as follows:

SORT Format for OUTPUT PROCEDURE

```
SORT file-name-1
    {ON {ASCENDING/DESCENDING} KEY data-name-1 . . .} . . .

    {INPUT PROCEDURE IS procedure-name-1
           [{THRU/THROUGH} procedure-name-2]}

    {OUTPUT PROCEDURE IS procedure-name-3
           [{THRU/THROUGH} procedure-name-4]}
```

As indicated, an INPUT PROCEDURE, if used, is processed prior to sorting. When the SORT verb is encountered, control goes to the INPUT PROCEDURE. When the INPUT PROCEDURE is complete, the file is then sorted. An **OUTPUT PROCEDURE** processes all sorted records *in the sort file* and handles the transfer of these records to the output file.

In an INPUT PROCEDURE we RELEASE records to a sort file rather than writing them. In an OUTPUT PROCEDURE we **RETURN** records from the sort file rather than reading them. The format for the RETURN is as follows:

Format	

Format	RETURN sort-file-name-1
	  AT END imperative statement-1
	  [NOT AT END imperative statement-2*]
	[END-RETURN]*

*Valid with COBOL 85 only.

Figure 20.12 and 20.13 provide a summary of OUTPUT PROCEDURE coding rules for both COBOL 85 and COBOL 74.

OUTPUT PROCEDURE **Summary: COBOL 85**

1. The OUTPUT PROCEDURE of the SORT should refer to a paragraph-name, but it could refer to a section-name.

Example

```
000-MAIN-MODULE.
    .
    .
     SORT SORT-FILE
         ON ASCENDING KEY SORT-DEPARTMENT
             WITH DUPLICATES IN ORDER
         INPUT PROCEDURE  200-SELECT-INPUT-RECORDS-RTN
         OUTPUT PROCEDURE 300-PRINT-REPORT-RTN.

     CLOSE SALES-REPORT-FILE.
     STOP RUN.

300-PRINT-REPORT-RTN.
   .
   .
```

2. In the paragraph specified in the OUTPUT PROCEDURE:
   a. If the same end-of-file field is used as in the INPUT PROCEDURE (ARE-THERE-MORE-RECORDS), it must be reset to 'YES'.
   b. PERFORM a paragraph that will process records from the sort file and continue to RETURN (which is like a read) until there is no more records.
   c. After all records have been processed, set NO-MORE-RECORDS TO TRUE.
   e. When the OUTPUT PROCEDURE paragraph has been fully executed, control will then return to the SORT.

Example

```
300-PRINT-REPORT-RTN.

    MOVE 'YES' TO ARE-THERE-MORE-RECORDS.

    PERFORM UNTIL NO-MORE-RECORDS
        RETURN SORT-FILE
            AT END
                SET NO-MORE-RECORDS TO TRUE
            NOT AT END
                PERFORM 310-WRITE-DETAIL-LINE-RTN
        END-RETURN
    END-PERFORM.

310-WRITE-DETAIL-LINE-RTN.
```

3. At the paragraph that processes the sort records after they have been sorted:
   a. Perform any operations on the work or sort records.
   b. `MOVE` the work or sort record to the output area.
   c. `WRITE` each sort record to the output file. (A `WRITE... FROM` can be used in place of a `MOVE` and `WRITE`.)
   d. Continue to `RETURN` sort file records until there is no more data.

Example

```
310-WRITE-DETAIL-LINE-RTN.
   . ┐
   . │ Process records from the sort file.
   . ┘
```

Figure 20.12 Summary of ***`OUTPUT PROCEDURE`*** *for COBOL 85.*

`OUTPUT PROCEDURE` **Summary: COBOL 74**

1. The entire program should consist of sections. The `OUTPUT PROCEDURE` should refer to a section-name followed by a paragraph-name.

Example

```
A000-MAIN SECTION.  ┐ Start the program with a section-name
                    │ followed by a paragraph-name
A000-MAIN-MODULE.   ┘
    .
    .
    SORT SORT-FILE
        ON ASCENDING KEY SORT-STORE-NUMBER
                         SORT-DEPARTMENT
            WITH DUPLICATES IN ORDER
        INPUT PROCEDURE  B000-SELECT-INPUT-RECORDS
        OUTPUT PROCEDURE C000-PRINT-REPORT.

    CLOSE SALES-REPORT-FILE.
    STOP RUN.

C000-PRINT-REPORT SECTION. ┐ Section-names are followed by
                           │ paragraph-names
C300-PRINT-REPORT-RTN.     ┘
    .
    .
```

2. In the main module of the section specified in the `OUTPUT PROCEDURE`:
   a. If the same end-of-file field (`ARE-THERE-MORE-RECORDS`) is used, it must be reset to `'YES'`.
   b. `RETURN` an initial record from the sort-file. The `RETURN` functions like a `READ`.
   c. `PERFORM` a paragraph within the section that will process records from the sort file and continue to process them until there is no more data.
   d. Code a `GO TO`, branching to the paragraph located physically at the end of the section.

Example

```
C000-PRINT-REPORT SECTION.

C300-PRINT-REPORT-RTN.

    MOVE 'YES' TO ARE-THERE-MORE-RECORDS.

     RETURN SORT-FILE
```

```
                AT END
                    MOVE 'NO ' TO ARE-THERE-MORE-RECORDS.

           PERFORM C310-WRITE-DETAIL-LINE-RTN
               UNTIL ARE-THERE-MORE-RECORDS = 'NO '.

           GO TO C000-EXIT.

       C310-WRITE-DETAIL-LINE-RTN.
```

3. At the paragraph that processes sort records after sorting:
   a. Perform any operations on work or sort records.
   b. Move the sort record to the output area.
   c. `WRITE` each sorted record to the output file.
   d. Continue to `RETURN` sort file records until there is no more data.

   Example

```
       C310-WRITE-DETAIL-LINE-RTN.
           . ┐
           . │ Process records in the sort file.
           . ┘

           RETURN SORT-FILE
               AT END
                   MOVE 'NO ' TO ARE-THERE-MORE-RECORDS.
```

4. The paragraph located physically at the end of the `OUTPUT PROCEDURE` section must be the last one executed. Hence a `GO TO` in the section's main module is required to transfer control to this last paragraph. If no processing is required, code an `EXIT` statement as the only entry in this last paragraph.

   Example

```
           GO TO C000-EXIT.
           .
           .
       C000-EXIT.
           EXIT.  ← Must be the only entry in the paragraph
                    for COBOL 74
```

*Figure 20.13 Summary of **`OUTPUT PROCEDURE`** for COBOL 74.*

Figures 20.14 provides an example COBOL 74 program that uses both `INPUT PROCEDURE` and `OUTPUT PROCEDURE` clauses. Sections are permissible with either COBOL 85 or COBOL 74 but required only for COBOL 74 programs.

In the `INPUT PROCEDURE`, the program reads records from the employee pay file and selects those records with sales greater than $5,000.00. The records are then sorted using the `SORT` statement. In the `OUTPUT PROCEDURE`, records are returned from the sort file and a report is printed.

```
       *A   B   1   2   2   2   3   3   4   4   4   5   5   6   6   6   7
       7890123456789012345678901234567890123456789012345678901234567890 12
000100 PROCESS APOST.
000200
000300 IDENTIFICATION DIVISION.
000400
000500 PROGRAM-ID. CPCH20C74.
000600
000700 AUTHOR.     JILL PROGRAMMER.
000800
000900****************************************************************
001000*  COBOL 74 Version using section-names.                       *
001100*                                                              *
```

```
*  This program reads records from the Employee Pay file,      *
*  selects those employees with sales greater than $5,000.00,  *
*  sorts the records by Department number within Store number, *
*  and prints a report from the sorted records.                *
*                                                              *
*  The SORT statement uses INPUT and OUTPUT PROCEDURE clauses. *
*                                                              *
****************************************************************

 ENVIRONMENT DIVISION.

 INPUT-OUTPUT SECTION.

 FILE-CONTROL.
     SELECT EMPLOYEE-PAY-FILE
         ASSIGN TO DISK-EMPPAYPF.

     SELECT SORT-FILE.

     SELECT SALES-REPORT-FILE
         ASSIGN TO PRINTER-QPRINT.

 DATA DIVISION.

 FILE SECTION.

 FD  EMPLOYEE-PAY-FILE.
 01  EMPLOYEE-PAY-RECORD.
     05                              PIC X(52).
     05  EP-SALES                    PIC 9(5)    PACKED-DECIMAL.

 SD  SORT-FILE.
 01  SORT-RECORD.
     05  SORT-EMPLOYEE-NUMBER        PIC 9(9).
     05  SORT-STORE-NUMBER           PIC 9(4).
     05                              PIC X(31).
     05  SORT-DEPARTMENT             PIC 9(3).
     05                              PIC X(5).
     05  SORT-SALES                  PIC 9(5)    PACKED-DECIMAL.

 FD  SALES-REPORT-FILE.
 01  PRINT-RECORD-OUT                PIC X(80).

 WORKING-STORAGE SECTION.

 01  WS-CURRENT-DATE.
     05  WS-CURRENT-YEAR             PIC 9(4).
     05  WS-CURRENT-MONTH            PIC 9(2).
     05  WS-CURRENT-DAY              PIC 9(2).

 01  WS-CONTROL-FIELDS.
     05  ARE-THERE-MORE-RECORDS      PIC X(3)    VALUE 'YES'.
         88  NO-MORE-RECORDS                     VALUE 'NO '.
     05  WS-PAGE-COUNTER             PIC 9(3)    PACKED-DECIMAL
                                                 VALUE ZEROS.
     05  WS-LINE-COUNTER             PIC 9(3)    PACKED-DECIMAL
                                                 VALUE 60.
     05  WS-LINE-LIMIT               PIC 9(3)    PACKED-DECIMAL
                                                 VALUE 60.

 01  HEADING-1.
     05                              PIC X(5)    VALUE SPACES.
     05                              PIC X(7)    VALUE 'CPCH20C'
     05                              PIC X(10)   VALUE SPACES.
     05                              PIC X(16)
                                     VALUE 'BEST DEAL STORES'.

 01  HEADING-2.
     05                              PIC X(5)    VALUE SPACES.
     05  HL-CURRENT-MONTH            PIC 9(2).
     05                              PIC X       VALUE '/'.
     05  HL-CURRENT-DAY              PIC 9(2).
     05                              PIC X       VALUE '/'.
     05  HL-CURRENT-YEAR             PIC 9(4).
     05                              PIC X(9)    VALUE SPACES.
     05                              PIC X(12)
                                     VALUE 'SALES REPORT'.
     05                              PIC X(9)    VALUE SPACES.
```

```
    05                              PIC X(4)    VALUE 'PAGE'.
    05                              PIC X(1)    VALUE SPACE.
    05  HL-PAGE                     PIC Z9      VALUE ZEROS.

01  HEADING-3.
    05                              PIC X(5)    VALUE SPACES.
    05                              PIC X(4)    VALUE 'DEPT'.
    05                              PIC X(9)    VALUE SPACES.
    05                              PIC X(8)    VALUE 'EMPLOYEE

01  HEADING-4.
    05                              PIC X(4)    VALUE SPACES.
    05                              PIC X(6)    VALUE 'NUMBER'.
    05                              PIC X(9)    VALUE SPACES.
    05                              PIC X(6)    VALUE 'NUMBER'.
    05                              PIC X(12)   VALUE SPACES.
    05                              PIC X(5)    VALUE 'SALES'.

01  DETAIL-LINE.
    05                              PIC X(6).
    05  DL-DEPARTMENT               PIC Z(3).
    05                              PIC X(7).
    05  DL-EMPLOYEE-NUMBER          PIC XXXBXXBXXXX.
    05                              PIC X(7).
    05  DL-SALES                    PIC $$$,$$9.99.

PROCEDURE DIVISION.

A000-MAIN SECTION.

A000-MAIN-MODULE.

    PERFORM A100-INITIALIZATION-RTN.

    SORT SORT-FILE
        ON ASCENDING KEY SORT-STORE-NUMBER
                         SORT-DEPARTMENT
            WITH DUPLICATES IN ORDER
        INPUT PROCEDURE  B000-SELECT-INPUT-RECORDS
        OUTPUT PROCEDURE C000-PRINT-REPORT.

    CLOSE SALES-REPORT-FILE.
    STOP RUN.

A100-INITIALIZATION-RTN.

    OPEN OUTPUT SALES-REPORT-FILE.

    MOVE FUNCTION CURRENT-DATE TO WS-CURRENT-DATE.
    MOVE WS-CURRENT-MONTH TO HL-CURRENT-MONTH.
    MOVE WS-CURRENT-DAY TO HL-CURRENT-DAY.
    MOVE WS-CURRENT-YEAR TO HL-CURRENT-YEAR.

B000-SELECT-INPUT-RECORDS SECTION.

B200-SELECT-RECORDS-RTN.

    OPEN INPUT EMPLOYEE-PAY-FILE.

    READ EMPLOYEE-PAY-FILE
        AT END
            MOVE 'NO ' TO ARE-THERE-MORE-RECORDS.

    PERFORM B210-PROCESS-RECORD-RTN
        UNTIL NO-MORE-RECORDS.

    CLOSE EMPLOYEE-PAY-FILE.

    GO TO B000-EXIT.

B210-PROCESS-RECORD-RTN.

    IF EP-SALES > 5000.00
        MOVE EMPLOYEE-PAY-RECORD TO SORT-RECORD
        RELEASE SORT-RECORD.

    READ EMPLOYEE-PAY-FILE
        AT END
```

```
            MOVE 'NO ' TO ARE-THERE-MORE-RECORDS.

B000-EXIT.
    EXIT.

C000-PRINT-REPORT SECTION.

C300-PRINT-REPORT-RTN.

    MOVE 'YES' TO ARE-THERE-MORE-RECORDS.

     RETURN SORT-FILE
         AT END
             MOVE 'NO ' TO ARE-THERE-MORE-RECORDS.

    PERFORM C310-WRITE-DETAIL-LINE-RTN
        UNTIL ARE-THERE-MORE-RECORDS = 'NO '.

    GO TO C000-EXIT.

C310-WRITE-DETAIL-LINE-RTN.
    MOVE SORT-DEPARTMENT TO DL-DEPARTMENT.
    MOVE SORT-EMPLOYEE-NUMBER TO DL-EMPLOYEE-NUMBER.
    MOVE SORT-SALES TO DL-SALES.
    IF WS-LINE-COUNTER IS >= WS-LINE-LIMIT
        PERFORM C315-HEADING-RTN.
    WRITE PRINT-RECORD-OUT FROM DETAIL-LINE
        AFTER ADVANCING 1 LINE.
    ADD 1 TO WS-LINE-COUNTER.

     RETURN SORT-FILE
         AT END
             MOVE 'NO ' TO ARE-THERE-MORE-RECORDS.

C315-HEADING-RTN.
    ADD 1 TO WS-PAGE-COUNTER.
    MOVE WS-PAGE-COUNTER TO HL-PAGE.
    WRITE PRINT-RECORD-OUT FROM HEADING-1
       AFTER ADVANCING PAGE.
    WRITE PRINT-RECORD-OUT FROM HEADING-2
       AFTER ADVANCING 2 LINES.
    WRITE PRINT-RECORD-OUT FROM HEADING-3
       AFTER ADVANCING 2 LINES.
    WRITE PRINT-RECORD-OUT FROM HEADING-4
       AFTER ADVANCING 1 LINE.
    MOVE SPACES TO PRINT-RECORD-OUT.
    WRITE PRINT-RECORD-OUT
       AFTER ADVANCING 1 LINE.
    MOVE 7 TO WS-LINE-COUNTER.

C000-EXIT.
    EXIT.
```

Figure 20.14 Program CPCH20C74.

Consider Figure 20.15, which offers an alternative that simplifies the coding for COBOL 85 users.

```
*A   B   1   2   2   2   3   3   4   4   4   5   5   6   6   6   7
789012345678901234567890123456789012345678901234567890123456789012
 PROCESS APOST.

 IDENTIFICATION DIVISION.

 PROGRAM-ID. CPCH20C85.

 AUTHOR.     JILL PROGRAMMER.


****************************************************************
*  COBOL 85 Version.                                           *
*                                                              *
*  This program reads records from the Employee Pay file,      *
*  selects those employees with sales greater than $5,000.00,  *
*  sorts the records by Department number within Store number, *
*  and prints a report from the sorted records.                *
*                                                              *
*  The SORT statement uses INPUT and OUTPUT PROCEDURE clauses. *
****************************************************************

 ENVIRONMENT DIVISION.

 INPUT-OUTPUT SECTION.

 FILE-CONTROL.
     SELECT EMPLOYEE-PAY-FILE
         ASSIGN TO DISK-EMPPAYPF.

     SELECT SORT-FILE.

     SELECT SALES-REPORT-FILE
         ASSIGN TO PRINTER-QPRINT.

 DATA DIVISION.

 FILE SECTION.

 FD  EMPLOYEE-PAY-FILE.
 01  EMPLOYEE-PAY-RECORD.
     05                              PIC X(52).
     05  EP-SALES                    PIC 9(5)    PACKED-DECIMAL.

 SD  SORT-FILE.
 01  SORT-RECORD.
     05  SORT-EMPLOYEE-NUMBER        PIC 9(9).
     05                              PIC X(35).
     05  SORT-DEPARTMENT             PIC 9(3).
     05                              PIC X(5).
     05  SORT-SALES                  PIC 9(5)    PACKED-DECIMAL.

 FD  SALES-REPORT-FILE.
 01  PRINT-RECORD-OUT                PIC X(80).

 WORKING-STORAGE SECTION.

 01  WS-CURRENT-DATE.
     05  WS-CURRENT-YEAR         PIC 9(4).
     05  WS-CURRENT-MONTH        PIC 9(2).
     05  WS-CURRENT-DAY          PIC 9(2).

 01  WS-CONTROL-FIELDS.
     05  ARE-THERE-MORE-RECORDS  PIC X(3)    VALUE 'YES'.
         88  NO-MORE-RECORDS                 VALUE 'NO '.
     05  WS-PAGE-COUNTER         PIC 9(3)    PACKED-DECIMAL
                                             VALUE ZEROS.
     05  WS-LINE-COUNTER         PIC 9(3)    PACKED-DECIMAL
                                             VALUE 60.
     05  WS-LINE-LIMIT           PIC 9(3)    PACKED-DECIMAL
                                             VALUE 60.

 01  HEADING-1.
     05                          PIC X(5)    VALUE SPACES.
     05                          PIC X(7)    VALUE 'CPCH20C'.
     05                          PIC X(10)   VALUE SPACES.
     05                          PIC X(16)
                                 VALUE 'BEST DEAL STORES'.

 01  HEADING-2.
     05                          PIC X(5)    VALUE SPACES.
     05  HL-CURRENT-MONTH        PIC 9(2).
     05                          PIC X       VALUE '/'.
     05  HL-CURRENT-DAY          PIC 9(2).
     05                          PIC X       VALUE '/'.
     05  HL-CURRENT-YEAR         PIC 9(4).
     05                          PIC X(9)    VALUE SPACES.
     05                          PIC X(12)
                                 VALUE 'SALES REPORT'.
     05                          PIC X(9)    VALUE SPACES.
     05                          PIC X(4)    VALUE 'PAGE'.
     05                          PIC X(1)    VALUE SPACE.
     05  HL-PAGE                 PIC Z9      VALUE ZEROS.

```

```
01  HEADING-3.
    05                          PIC X(5)    VALUE SPACES.
    05                          PIC X(4)    VALUE 'DEPT'.
    05                          PIC X(9)    VALUE SPACES.
    05                          PIC X(8)    VALUE 'EMPLOYEE'.

01  HEADING-4.
    05                          PIC X(4)    VALUE SPACES.
    05                          PIC X(6)    VALUE 'NUMBER'.
    05                          PIC X(9)    VALUE SPACES.
    05                          PIC X(6)    VALUE 'NUMBER'.
    05                          PIC X(12)   VALUE SPACES.
    05                          PIC X(5)    VALUE 'SALES'.

01  DETAIL-LINE.
    05                          PIC X(6).
    05  DL-DEPARTMENT           PIC Z(3).
    05                          PIC X(7).
    05  DL-EMPLOYEE-NUMBER      PIC XXXBXXBXXXX.
    05                          PIC X(7).
    05  DL-SALES                PIC $$$,$$9.99.

PROCEDURE DIVISION.

000-MAIN-MODULE.

    PERFORM 100-INITIALIZATION-RTN.

    SORT SORT-FILE
        ON ASCENDING KEY SORT-DEPARTMENT
            WITH DUPLICATES IN ORDER
        INPUT PROCEDURE  200-SELECT-INPUT-RECORDS-RTN
        OUTPUT PROCEDURE 300-PRINT-REPORT-RTN.

    CLOSE SALES-REPORT-FILE.
    STOP RUN.

100-INITIALIZATION-RTN.

    OPEN OUTPUT SALES-REPORT-FILE.

    MOVE FUNCTION CURRENT-DATE TO WS-CURRENT-DATE.
    MOVE WS-CURRENT-MONTH TO HL-CURRENT-MONTH.
    MOVE WS-CURRENT-DAY TO HL-CURRENT-DAY.
    MOVE WS-CURRENT-YEAR TO HL-CURRENT-YEAR.

200-SELECT-INPUT-RECORDS-RTN.

    OPEN INPUT EMPLOYEE-PAY-FILE.

    PERFORM UNTIL NO-MORE-RECORDS
        READ EMPLOYEE-PAY-FILE
            NOT AT END
                PERFORM 210-PROCESS-RECORD-RTN
            AT END
                SET NO-MORE-RECORDS TO TRUE
        END-READ
    END-PERFORM

    CLOSE EMPLOYEE-PAY-FILE.

210-PROCESS-RECORD-RTN.
    IF EP-SALES > 5000.00
        MOVE EMPLOYEE-PAY-RECORD TO SORT-RECORD
        RELEASE SORT-RECORD
    END-IF.

300-PRINT-REPORT-RTN.

    MOVE 'YES' TO ARE-THERE-MORE-RECORDS.

    PERFORM UNTIL NO-MORE-RECORDS
        RETURN SORT-FILE
            AT END
                SET NO-MORE-RECORDS TO TRUE
            NOT AT END
                PERFORM 310-WRITE-DETAIL-LINE-RTN
        END-RETURN
    END-PERFORM.

310-WRITE-DETAIL-LINE-RTN.
    MOVE SORT-DEPARTMENT TO DL-DEPARTMENT.
    MOVE SORT-EMPLOYEE-NUMBER TO DL-EMPLOYEE-NUMBER.
    MOVE SORT-SALES TO DL-SALES.
    IF WS-LINE-COUNTER IS >= WS-LINE-LIMIT
        PERFORM 315-HEADING-RTN
    END-IF
    WRITE PRINT-RECORD-OUT FROM DETAIL-LINE
        AFTER ADVANCING 1 LINE.
    ADD 1 TO WS-LINE-COUNTER.

315-HEADING-RTN.
    ADD 1 TO WS-PAGE-COUNTER.
    MOVE WS-PAGE-COUNTER TO HL-PAGE.
    WRITE PRINT-RECORD-OUT FROM HEADING-1
       AFTER ADVANCING PAGE.
    WRITE PRINT-RECORD-OUT FROM HEADING-2
       AFTER ADVANCING 2 LINES.
    WRITE PRINT-RECORD-OUT FROM HEADING-3
       AFTER ADVANCING 2 LINES.
    WRITE PRINT-RECORD-OUT FROM HEADING-4
       AFTER ADVANCING 1 LINE.
    MOVE SPACES TO PRINT-RECORD-OUT.
    WRITE PRINT-RECORD-OUT
       AFTER ADVANCING 1 LINE.
    MOVE 7 TO WS-LINE-COUNTER.
```

Figure 20.15 Program CPCH20C85.

With COBOL 85, you may use an END-RETURN scope terminator with the RETURN statement. Also, the RETURN statement may include a NOT AT END clause.

WHEN TO USE INPUT AND/OR OUTPUT PROCEDURES

Sometimes it is more efficient to process data *before* it is sorted in an INPUT PROCEDURE, whereas other times it is more efficient to process data *after* it is sorted in an OUTPUT PROCEDURE. For instance, suppose we wish to sort a large file into DEPARTMENT-NUMBER sequence. Suppose, further, we wish to eliminate from our file all records with a blank PRICE or blank QUANTITY field. We could eliminate the designated records *prior to* sorting in an INPUT PROCEDURE, or we could eliminate the records *after* sorting in an OUTPUT PROCEDURE.

If we expect only a few records to be eliminated during a run, then it really would not matter much whether we sort first and then eliminate those records we do not wish to put on the output file. If, however, there are many records that need to be eliminated, it is more efficient to remove them *before* sorting. In this way, we do not waste computer time sorting numerous records that will then be removed from the sorted file. Thus, in the case where a large number of records is removed, an INPUT PROCEDURE should be used.

Keep in mind that you must use either an INPUT or an OUTPUT PROCEDURE if the unsorted and sorted files have different-sized fields or have fields in different order. This is because the input record must be moved to a record with a different format either prior to or after sorting.

Figure 20.16 provides a summary of the SORT feature and its options.

SORT **Options: A Brief Summary**

	Format	**Result**
1.	USING GIVING	File is sorted, no special handling.

2.	INPUT PROCEDURE GIVING	Used for processing the unsorted input records before they are sorted. Write records to the sort file with a RELEASE verb. After an INPUT PROCEDURE is completed, the records are automatically sorted.
3.	USING OUTPUT PROCEDURE	Used for processing the sorted records before writing them on the output file. Access or read records from the sort file with a RETURN verb.
4.	INPUT PROCEDURE OUTPUT PROCEDURE	Used for processing the data both before and after it is sorted.

*Figure 20.16 Summary of **SORT** options.*

THE MERGE STATEMENT

COBOL has a MERGE statement that will combine two or more files into a single file. Its format is similar to that of the SORT:

Format

```
MERGE file-name-1 {ON {ASCENDING/DESCENDING} KEY data-name-1. . .} . . .

USING file-name-2 {file-name-3} . . .

{OUTPUT PROCEDURE IS procedure-name-1 [{THROUGH/THRU} procedure-name-2]}
{GIVING {file-name-4}
```

File-name-1 is a work file designated with an SD. The key field specified as data-name-1, and any subsequent key fields, are defined within the SD. The first key field indicated in the ASCENDING or DESCENDING KEY clause of the MERGE is the major one, followed by intermediate and minor key fields. Rules for ASCENDING/DESCENDING KEY, USING, GIVING, and OUTPUT PROCEDURE are the same as for the SORT.

With the USING clause, we indicate the files to be merged. At least two file-names must be included for a merge, but more than two are permitted. Unlike the SORT, however, an INPUT PROCEDURE may *not* be specified with a MERGE statement. That is, using the MERGE statement, you may process records only *after* they have been merged, *not* before. The OUTPUT PROCEDURE has the same format as with the SORT, and the same distinctions between COBOL 85 and COBOL 74 apply.

The **MERGE** statement automatically handles the opening, closing, and input/output (READ/WRITE functions) associated with the files. See Figure 20.17 for an illustration of a program with the MERGE instruction.

```
*A   B   1   2   2   2   3   3   4   4   4   5   5   6   6   6   7
789012345678901234567890123456789012345678901234567890123456789012
 PROCESS APOST.

 IDENTIFICATION DIVISION.

 PROGRAM-ID. CPCH20D.

 AUTHOR.     Jill Programmer.

*****************************************************************
*                                                               *
*  This program reads records from two Employee Pay files,      *
*  EMPPAYPF33 from store 1133 and EMPPAYPF57 from store 2257,   *
*  merges the records by Department number                      *
*  and outputs a merged file with the merged sorted records.    *
*                                                               *
*  The MERGE statement uses USING and GIVING clauses.           *
*                                                               *
*****************************************************************
```

```
001900
002000 ENVIRONMENT DIVISION.
002100
002200 INPUT-OUTPUT SECTION.
002300
002400 FILE-CONTROL.
002500     SELECT EMPLOYEE-PAY-FILE-STORE-1133
002600         ASSIGN TO DISK-EMPPAYPF33.
002700
002800     SELECT EMPLOYEE-PAY-FILE-STORE-2257
002900         ASSIGN TO DISK-EMPPAYPF57.
003000
003100     SELECT MERGED-EMPLOYEE-PAY-FILE
003200         ASSIGN TO DISK-EMPPAYPFM.
003300
003400     SELECT MERGE-FILE.
003500
003600 DATA DIVISION.
003700
003800 FILE SECTION.
003900
004000 FD  EMPLOYEE-PAY-FILE-STORE-1133.
004100 01  EMPLOYEE-PAY-RECORD-1133       PIC X(55).
004200
004300 FD  EMPLOYEE-PAY-FILE-STORE-2257.
004400 01  EMPLOYEE-PAY-RECORD-2257       PIC X(55).
004500
004600 FD  MERGED-EMPLOYEE-PAY-FILE.
004700 01  MERGED-EMPLOYEE-PAY-RECORD     PIC X(55).
004800
004900 SD  MERGE-FILE.
005000 01  MERGE-RECORD.
005100     05                             PIC X(44).
005200     05  MERGE-DEPARTMENT           PIC 9(3).
005300     05                             PIC X(8).
005400
005500 WORKING-STORAGE SECTION.
005600
005700 PROCEDURE DIVISION.
005800
005900 000-MAIN-MODULE.
006000
006100     MERGE MERGE-FILE
006200         ON ASCENDING KEY MERGE-DEPARTMENT
006300         USING            EMPLOYEE-PAY-FILE-STORE-1133
006400                          EMPLOYEE-PAY-FILE-STORE-2257
006500         GIVING           MERGED-EMPLOYEE-PAY-FILE.
006600
006700     STOP RUN.
```

*Figure 20.17   COBOL program that uses **MERGE** statement.*

The files to be merged must each be in sequence by the key field. If ASCENDING KEY is specified, then the merged output file will have records in increasing order by key field, and if DESCENDING KEY is specified, the merged output file will have key fields from high to low.

An OUTPUT PROCEDURE for a MERGE may be used, for example, to

1. Flag duplicate records as errors.
   If an UPSTATE-PAYROLL-FILE and a DOWNSTATE-PAYROLL-FILE are being merged to produce a MASTER PAYROLL-FILE in Social Security number sequence, we may use an OUTPUT PROCEDURE to ensure that no two records on the merged file have the same Social Security number.

2. Ensure duplicate records.
   If an UPSTATE-INVENTORY-FILE and a DOWNSTATE-INVENTORY-FILE store the same PART-NUMBERs, we may MERGE them into a MASTER-INVENTORY-FILE and in an OUTPUT PROCEDURE check to see that there are always two records for each PART-NUMBER—an UPSTATE and a DOWNSTATE record.

The same rules apply to `OUTPUT PROCEDURE`s for the `MERGE` as for the `SORT`. Section-names are required for COBOL 74, but paragraph-names can be used for COBOL 85.

Suppose we want to merge an employee pay file from Store 1133 with an employee pay file from Store 2257 into one merged employee pay file sorted on department number. In an output procedure, we want to print a sales report. Figure 20.18 illustrates the full COBOL 85 program that uses paragraph-names as procedure-names.

```
      *A   B   1   2   2   2   3   3   4   4   4   5   5   6   6   6   7
      789012345678901234567890123456789012345678901234567890123456789012
000100 PROCESS APOST.
000200
000300 IDENTIFICATION DIVISION.
000400
000500 PROGRAM-ID. CPCH20E.
000600
000700 AUTHOR.     JILL PROGRAMMER.
000800
000900*****************************************************************
001000*                                                               *
001100*  This program reads records from two Employee Pay files,      *
001200*  EMPPAYPF33 from store 1133 and EMPPAYPF57 from store 2257,   *
001300*  merges the records from both files into one file,            *
001400*  sorts the records into Department number sequence,           *
001500*  and outputs a report sorted by Department number.            *
001600*                                                               *
001700*  The MERGE statement uses USING and OUTPUT PROCEDURE clauses.*
001800*                                                               *
001900*****************************************************************
002000
002100 ENVIRONMENT DIVISION.
002200
002300 INPUT-OUTPUT SECTION.
002400
002500 FILE-CONTROL.
002600     SELECT EMPLOYEE-PAY-FILE-STORE-1133
002700         ASSIGN TO DISK-EMPPAYPF33.
002800
002900     SELECT EMPLOYEE-PAY-FILE-STORE-2257
003000         ASSIGN TO DISK-EMPPAYPF57.
003100
003200     SELECT MERGED-EMPLOYEE-PAY-FILE
003300         ASSIGN TO DISK-EMPPAYPFM.
003400
003500     SELECT MERGE-FILE.
003600
003700     SELECT SALES-REPORT-FILE
003800         ASSIGN TO PRINTER-QPRINT.
003900
004000 DATA DIVISION.
004100
004200 FILE SECTION.
004300
004400 FD  EMPLOYEE-PAY-FILE-STORE-1133.
004500 01  EMPLOYEE-PAY-RECORD-1133        PIC X(55).
004600
004700 FD  EMPLOYEE-PAY-FILE-STORE-2257.
004800 01  EMPLOYEE-PAY-RECORD-2257        PIC X(55).
004900
005000 FD  MERGED-EMPLOYEE-PAY-FILE.
005100 01  MERGED-EMPLOYEE-PAY-RECORD      PIC X(55).
005200
005300 SD  MERGE-FILE.
005400 01  MERGE-RECORD.
005500     05  M-EMPLOYEE-NUMBER           PIC 9(9).
005600     05  M-STORE-NUMBER              PIC 9(4).
005700     05  M-FIRST-NAME                PIC X(15).
005800     05  M-MIDDLE-INITIAL            PIC X(1).
005900     05  M-LAST-NAME                 PIC X(15).
006000     05  M-DEPARTMENT                PIC 9(3).
006100     05  M-HOURLY-RATE               PIC 9(3)V99 PACKED-DECIMAL.
```

```
    05  M-HOURS-WORKED              PIC 9(2)V9  PACKED-DECIMAL.
    05  M-SALES                     PIC 9(5)    PACKED-DECIMAL.

FD  SALES-REPORT-FILE.
01  PRINT-RECORD-OUT                PIC X(80).

WORKING-STORAGE SECTION.

01  WS-CURRENT-DATE.
    05  WS-CURRENT-YEAR         PIC 9(4).
    05  WS-CURRENT-MONTH        PIC 9(2).
    05  WS-CURRENT-DAY          PIC 9(2).

01  WS-CONTROL-FIELDS.
    05  ARE-THERE-MORE-RECORDS  PIC X(3)    VALUE 'YES'.
        88  NO-MORE-RECORDS                 VALUE 'NO '.
    05  WS-PAGE-COUNTER         PIC 9(3)    PACKED-DECIMAL
                                            VALUE ZEROS.
    05  WS-LINE-COUNTER         PIC 9(3)    PACKED-DECIMAL
                                            VALUE 60.
    05  WS-LINE-LIMIT           PIC 9(3)    PACKED-DECIMAL
                                            VALUE 60.

01  HEADING-1.
    05                          PIC X(5)    VALUE SPACES.
    05                          PIC X(7)    VALUE 'CPCH20E'.
    05                          PIC X(10)   VALUE SPACES.
    05                          PIC X(16)
                                VALUE 'BEST DEAL STORES'.

01  HEADING-2.
    05                          PIC X(5)    VALUE SPACES.
    05  HL-CURRENT-MONTH        PIC 9(2).
    05                          PIC X       VALUE '/'.
    05  HL-CURRENT-DAY          PIC 9(2).
    05                          PIC X       VALUE '/'.
    05  HL-CURRENT-YEAR         PIC 9(4).
    05                          PIC X(9)    VALUE SPACES.
    05                          PIC X(12)
                                VALUE 'SALES REPORT'.
    05                          PIC X(9)    VALUE SPACES.
    05                          PIC X(4)    VALUE 'PAGE'.
    05                          PIC X(1)    VALUE SPACE.
    05  HL-PAGE                 PIC Z9      VALUE ZEROS.

01  HEADING-3.
    05                          PIC X(5)    VALUE SPACES.
    05                          PIC X(4)    VALUE 'DEPT'.
    05                          PIC X(9)    VALUE SPACES.
    05                          PIC X(8)    VALUE 'EMPLOYEE'.

01  HEADING-4.
    05                          PIC X(4)    VALUE SPACES.
    05                          PIC X(6)    VALUE 'NUMBER'.
    05                          PIC X(9)    VALUE SPACES.
    05                          PIC X(6)    VALUE 'NUMBER'.
    05                          PIC X(12)   VALUE SPACES.
    05                          PIC X(5)    VALUE 'SALES'.

01  DETAIL-LINE.
    05                          PIC X(6).
    05  DL-DEPARTMENT           PIC Z(3).
    05                          PIC X(7).
    05  DL-EMPLOYEE-NUMBER      PIC XXXBXXBXXXX.
    05                          PIC X(7).
    05  DL-SALES                PIC $$$,$$9.99.

PROCEDURE DIVISION.

000-MAIN-MODULE.
    PERFORM 100-INITIALIZATION-RTN.

    MERGE MERGE-FILE
        ON ASCENDING KEY M-DEPARTMENT
        USING            EMPLOYEE-PAY-FILE-STORE-1133
                         EMPLOYEE-PAY-FILE-STORE-2257
        OUTPUT PROCEDURE 300-PRINT-REPORT-RTN.

```

```
           PERFORM 400-TERMINATION-RTN.
           STOP RUN.

       100-INITIALIZATION-RTN.

           OPEN OUTPUT SALES-REPORT-FILE.

           MOVE FUNCTION CURRENT-DATE TO WS-CURRENT-DATE.
           MOVE WS-CURRENT-MONTH TO HL-CURRENT-MONTH.
           MOVE WS-CURRENT-DAY TO HL-CURRENT-DAY.
           MOVE WS-CURRENT-YEAR TO HL-CURRENT-YEAR.

       300-PRINT-REPORT-RTN.

           MOVE 'YES' TO ARE-THERE-MORE-RECORDS.

           PERFORM UNTIL NO-MORE-RECORDS
               RETURN MERGE-FILE
                   AT END
                       SET NO-MORE-RECORDS TO TRUE
                   NOT AT END
                       PERFORM 310-WRITE-DETAIL-LINE-RTN
               END-RETURN
           END-PERFORM.

       310-WRITE-DETAIL-LINE-RTN.
           MOVE M-DEPARTMENT TO DL-DEPARTMENT.
           MOVE M-EMPLOYEE-NUMBER TO DL-EMPLOYEE-NUMBER.
           MOVE M-SALES TO DL-SALES.
           IF WS-LINE-COUNTER IS >= WS-LINE-LIMIT
               PERFORM 315-HEADING-RTN
           END-IF
           WRITE PRINT-RECORD-OUT FROM DETAIL-LINE
               AFTER ADVANCING 1 LINE.
           ADD 1 TO WS-LINE-COUNTER.

       315-HEADING-RTN.
           ADD 1 TO WS-PAGE-COUNTER.
           MOVE WS-PAGE-COUNTER TO HL-PAGE.
           WRITE PRINT-RECORD-OUT FROM HEADING-1
              AFTER ADVANCING PAGE.
           WRITE PRINT-RECORD-OUT FROM HEADING-2
              AFTER ADVANCING 2 LINES.
           WRITE PRINT-RECORD-OUT FROM HEADING-3
              AFTER ADVANCING 2 LINES.
           WRITE PRINT-RECORD-OUT FROM HEADING-4
              AFTER ADVANCING 1 LINE.
           MOVE SPACES TO PRINT-RECORD-OUT.
           WRITE PRINT-RECORD-OUT
              AFTER ADVANCING 1 LINE.
           MOVE 7 TO WS-LINE-COUNTER.

       400-TERMINATION-RTN.
           CLOSE SALES-REPORT-FILE.
```

Figure 20.18 Solution for program CPCH04A.

Note that the elementary items within the two input files need not have been specified since they are not used. Instead, we coded

```
01  EMPLOYEE-PAY-FILE-STORE-1133   PIC X(55)

01  EMLOYEE-PAY-FILE-STORE-2257    PIC X(55).
```

END-OF-CHAPTER AIDS

CHAPTER SUMMARY

A. The SORT is used for sorting records in either ascending or descending order.

1. A program can simply sort a file on key fields:

```
 SORT file-name-1
 {ON  {ASCENDING/DESCENDING} KEY  data-name-1 ...} ...
  USING  file-name-2
  GIVING  file-name-3
```

   a. File-name-1 is a work or sort file that is described with an SD (sort file description) in the FILE SECTION.
   b. The KEY field(s) to be sorted are data-names defined within the SD or sort file.
   c. Files can be sorted into ascending or descending sequence.
   d. Files can be sorted using more than one key field. The first field specified is the main sort field followed by intermediate and/or minor ones. SORT ... ON ASCENDING KEY DEPARTMENT ON DESCENDING KEY NAME ... will sort a file into ascending department number order (01 to 99) and, within that, into descending NAME order (Z to A). For Department 01, ZACHARY precedes YOUNG, who precedes VICTOR, etc.

2. A program can include an entirely separate routine that processes an unsorted file prior to performing the SORT and/or an entirely separate routine that processes the sorted file after the SORT is executed:

[can open, read, and process the unsorted file; then close it before sorting]
```
SORT ...
    ...
    USING ...
    GIVING ...
```
[can open, read, and process the sorted file]
```
STOP RUN.
```

3. An INPUT PROCEDURE that is part of the SORT statement permits processing of the unsorted file just before the sort is performed, yet under the control of the SORT itself:

```
SORT file-name-1
    ...
    INPUT PROCEDURE procedure-name-1
    GIVING file-name-2
```

   a. COBOL 85 uses a paragraph-name when specifying an INPUT PROCEDURE. COBOL 74 uses a section-name. Since the physical end of a section must be reached to terminate the section, an INPUT PROCEDURE for COBOL 74 must end with an EXIT statement.
   b. With COBOL 85 paragraph-names can be substituted for section-names, so there is no need for a GO TO to branch to the end of a section.

c. The clause `RELEASE sort-record FROM unsorted-record` is necessary in an `INPUT PROCEDURE` to make each processed input record available for sorting.

4. An `OUTPUT PROCEDURE` that is part of the `SORT` statement permits processing of the sorted work (or sort) file records after they are sorted:

```
SORT file-name-1 ...
 {USING file-name-2/INPUT PROCEDURE procedure-name-1}
  OUTPUT PROCEDURE procedure-name-2
```

a. As with the `INPUT PROCEDURE`, the procedure-name specified must be a section-name for COBOL 74 but can be either a section- or paragraph-name with COBOL 85 (or enhanced versions of COBOL 74). Using paragraph-names simplifies the coding and eliminates the need for `GO TO`s.
b. An `OUTPUT PROCEDURE`
   (1) Includes a `RETURN` sort-file-name `AT END ...,` which is like a `READ`, for all inputting of sort file records.
   (2) Processes all records from the sort file.
c. Both an `INPUT` and an `OUTPUT PROCEDURE` can be used in a program.

B. The `MERGE` statement can be used to merge two or more files. It is very similar to the `SORT`. It can have a `USING` and `GIVING` option or an `OUTPUT PROCEDURE` in place of the `GIVING` option. It *cannot*, however, have an `INPUT PROCEDURE`.

KEY TERMS

ASCII
Collating sequence
EBCDIC
FIFO (first in, first out)
`INPUT PROCEDURE`
`MERGE`
`OUTPUT PROCEDURE`
`RELEASE`
`RETURN`
Section
`SORT`

CHAPTER SELF-TEST

TRUE-FALSE QUESTIONS

___ 1. If the `OUTPUT PROCEDURE` is used with the `SORT` verb, then the `INPUT PROCEDURE` is required.
___ 2. `RELEASE` must be used in an `INPUT PROCEDURE`.
___ 3. `RETURN` must be used in an `OUTPUT PROCEDURE`.
___ 4. The `RELEASE` statement is used in place of the `WRITE` statement in an `INPUT PROCEDURE`.
___ 5. A maximum of three `SORT` fields are permitted in a single `SORT` statement.
___ 6. The only method for sorting a disk file is with the use of the `SORT` statement in COBOL.
___ 7. Data may be sorted in either ascending or descending sequence.
___ 8. With COBOL 85, the procedure-name specified in the `INPUT PROCEDURE` clause can be a paragraph-name.
___ 9. If a file is described by an SD, it is not defined in a `SELECT` clause.
___ 10. In the EBCDIC collating sequence, a blank has the lowest value.

FILL–IN–THE BLANKS

1. It is possible to process records before they are sorted by using the ______________ option in place of the ______________ option.
2. In place of a `WRITE` statement in an `INPUT PROCEDURE`, the ______________ verb is used to write records onto the sort or work file.
3. In place of a `READ` statement in an `OUTPUT PROCEDURE`, the ______________ verb is used to read records from the sort or work file.
4. If section-names are used in the `PROCEDURE DIVISION`, they should be followed by ______________.
5. The work or sort file is defined as an ______________ in the `DATA DIVISION`.

CHAPTER REVIEW QUESTIONS

1. Suppose we want `EMPLOYEE-FILE` records in alphabetic order by `NAME` within `DISTRICT` within `TERRITORY`, all in ascending sequence. The output file is called `SORTED-EMPLOYEE-FILE`. Complete the following `SORT` statement:

   ```
   SORT  WORK-FILE ...
   ```

2. How many files are required in a `SORT` routine? Describe these files.
3. Suppose we have an `FD` called `NET-FILE-IN`, an `SD` called `NET-FILE`, and an `FD` called `NET-FILE-OUT`. We want `NET-FILE-OUT` sorted into ascending `DEPT-NUMBER` sequence. Code the `PROCEDURE DIVISION` entry.

PROGRAMMING ASSIGNMENTS

Use the record description in Figure 20.19 for Programming Assignments 1 through 3.

Field Description	**Type**	**Size**	**COBOL Field-name**
Employee Number	S	5,0	PM-EMPLOYEE-NUMBER
Employee Name	A	20	PM-EMPLOYEE-NAME
Territory Number	S	3,0	PM-TERRITORY-NUMBER
Office Number	S	2,0	PM-OFFICE-NUMBER
Annual Salary	P	7,0	PM-ANNUAL-SALARY
Social Security Number	S	9,0	PM-SOC-SEC-NUMBER

Figure 20.19 *Record description for* ***PAYROLL-MASTER*** *and* ***SORTED-PAYROLL-MASTER*** *payroll files.*

1. Sort the input file into descending sequence by territory number and office number, but eliminate, before sorting, all records that have a blank territory number, office number, or Social Security number. Print all records that have been eliminated.

2. Develop an interactive program that sorts the input file into ascending sequence by territory number, and, after sorting, adds $1,000 to the salaries of employees who earn less than $35,000. Display on the screen the names and salaries of all employees who get increases.

3. Sort the input file into ascending territory number sequence. Then write a control break program to print a report with the format shown in Figure 20.20.

```
                 1         2         3         4         5         6
        123456789012345678901234567890123456789012345678901234567890
H | 1 |      99/99/2099    Employee Summary Report       Page Z9
  | 2 |
H | 3 |      Territory
H | 4 |       Number       Total Number of Employees
  | 5 |
D | 6 |        999         Z,ZZ9
D | 7 |        999         Z,ZZ9
  | 8 |
```

Figure 20.20 Printer spacing chart for employee summary report.

4. A large corporation with two plants has discovered that some of its employees are on the payrolls of both of its plants. Each plant has a payroll file in Social Security Number sequence. Write a program to merge the two files and to print the names of the "double-dippers"; that is, the employees who are on both files.

5. The SmartBell Telephone Company maintains transaction records of long-distance calls made by its customers. A transaction record, shown in Figure 20.21, is created for each long-distance call made. The transaction file is in no specific order, since a record is automatically created when a call is made.

Field Description	**Type**	**Size**	**COBOL Field-name**
Caller telephone number	S	10,0	LD-CALLER-PHONE-NUMBER
Called telephone number	A	10,0	LD-CALLED-PHONE-NUMBER
Number of minutes of call	S	3,0	LD-TIME-IN-MINUTES
Charge	P	5,2	LD-CHARGE

*Figure 20.21 Record description for **`LONG-DISTANCE`** file.*

A separate master file, shown in Figure 20.22, is maintained for each customer.

Field Description	**Type**	**Size**	**COBOL Field-name**
Customer Telephone Number (K)	S	10,0	CM-CUSTOMER-PHONE-NUMBER
Customer Last name	A	20	CM-CUSTOMER-LAST-NAME
Customer first name	S	3,0	CM-CUSTOMER-FIRST-NAME
Street Address	P	5,2	CM-STREET-ADDRESS
City	A	20	CM-CITY
State	A	2	CM-STATE
Zip	S	9,0	CM-ZIP
Monthly charge	P	5,2	CM-MONTHLY-CHARGE

*Figure 20.22 Record description for **`CUSTOMER-TELEPHONE-MASTER`** file.*

The customer master file is in sequence by telephone number. Create monthly telephone bills for each customer in customer name sequence. Design the format of the bills yourself.