3

How to write a program that prepares a report

In this chapter, you'll learn how to write a program that gets data from a disk file, prepares a report, and prints the report on a printer. This is a realistic use of COBOL in business. In fact, COBOL was designed for working with the data in files and for preparing printed reports from that data. Today, tens of thousands of COBOL report-preparation programs are in use throughout this country.

Mike Murach & Associates

2560 West Shaw Lane, Suite 101
Fresno, CA 93711-2765
(559) 440-9071 • (800) 221-5528

murachbooks@murach.com • www.murach.com

A simple report-preparation program

To start, this chapter is going to present a simple report-preparation program. This program reads the data from the customer records in a sequential disk file, processes the data into the form of a sales report, and prints the report. The file and report specifications follow.

The file specifications

The input to the program is a *sequential file* on disk that consists of customer master *records*. When data is stored in a sequential file, the program reads the records in sequence starting with the first record. As a result, the report that's prepared from the file is in the same sequence as the records in the file.

Each customer record in the sequential file consists of the *fields* shown in figure 3-1. If you look at the record layout at the top of this figure, you can see that the branch number field is stored in bytes 1 and 2; the salesrep number field is in bytes 3 and 4; and so on. You also can see that COBOL picture notation is used to indicate the format of each field. For instance, the customer number is an unsigned five-digit number; the customer name field consists of 20 characters; and the sales last year-to-date field is a signed seven-digit number with two decimal places.

Right after the record layout for this file, you can see how the record description will be coded in COBOL. Here, the 01 level is a group item that represents the entire record; the 05 levels represent the fields within that record. Notice in the field definitions that the prefix CM (short for Customer Master) is used as the start of each data name. This isn't required by COBOL, but it shows that the field is part of the customer master record. When a program consists of two or more record descriptions, this naming technique helps you keep the fields straight. As a result, you'll see this technique used in all of the record descriptions in this book.

This figure also presents the test data you can use for the first test run of this program. If you looked at the actual data in this file, you would see that there's nothing in each record to mark the start or end of each field. For example, the first record really looks like this:

```
121211111INFORMATION BUILDERS01234560111111
```

In other words, the data from one field is followed immediately by the data from the next field. In this figure, though, the fields are presented in a table so you can easily identify the contents of each field.

Notice that the data for the two sales fields doesn't include decimal points, even though the Pic clauses in the record descriptions for these fields indicate that they contain two decimal places. As you learned in chapter 1, that's because the V in the Pic clause just shows where the decimal point should be. When each record is read, it is the field definitions that determine how the data is interpreted.

The record layout

Bytes 1-2	Bytes 3-4	Bytes 5-9	Bytes 10-29	Bytes 30-36	Bytes 37-43
Branch number	Salesrep number	Customer number	Customer name	Sales this year-to-date	Sales last year-to-date
9(2)	9(2)	9(5)	X(20)	S9(5)V99	S9(5)V99

The record description in COBOL

```
01  CUSTOMER-MASTER-RECORD.
    05  CM-BRANCH-NUMBER        PIC 9(2).
    05  CM-SALESREP-NUMBER      PIC 9(2).
    05  CM-CUSTOMER-NUMBER      PIC 9(5).
    05  CM-CUSTOMER-NAME        PIC X(20).
    05  CM-SALES-THIS-YTD       PIC S9(5)V9(2).
    05  CM-SALES-LAST-YTD       PIC S9(5)V9(2).
```

The test data for the first test run

Branch number	Salesrep number	Customer number	Customer name	Sales this year-to-date	Sales last year-to-date
12	12	11111	INFORMATION BUILDERS	0123456	0111111
12	12	12345	CAREER TRAINING CTR	1234567	2222222
22	10	22222	HOMELITE TEXTRON CO	3454500	0000000
22	14	34567	NEAS MEMBER BENEFITS	0011111	0000000
22	14	55555	PILOT LIFE INS. CO.	1000000	0100000
34	10	00111	DAUPHIN DEPOSIT BANK	1409900	1993000
34	10	54321	AIRCRAFT OWNERS ASSC	0542612	4042000
34	17	33333	NORFOLK CORP	0639635	0446288
47	11	12121	GENERAL SERVICES CO.	1144400	1105956
47	11	24680	INFO MANAGEMENT CO.	1748145	1189247
47	21	99999	DOLLAR SAVINGS BANK	0505900	0462195
47	21	76543	NATL MUSIC CORP.	0238346	0443526

Description

- This program reads a *sequential file* of customer *records* that are stored on disk. Each of these records consists of 43 bytes of data that is divided into six *fields*.
- Because the COBOL record description gives the same information as the record layout, COBOL record descriptions are often used to document the layout of the fields in a disk record.
- If you study the test data, you can see that the decimal points aren't in the data. Remember that the V indicates the position of an assumed decimal point.
- Although the Pictures for the year-to-date fields indicate that they can carry signs, the data doesn't include any negative values. We avoided the use of negative values because signs are stored differently on different types of computers (see chapter 6).

Figure 3-1 The file specifications for the program

The report specifications

Figure 3-2 presents the report specifications for this program. To plan a report, a *print chart* like the one at the top of this figure can be used. If you compare this chart with the printed output for the test run, you can see how the two are related.

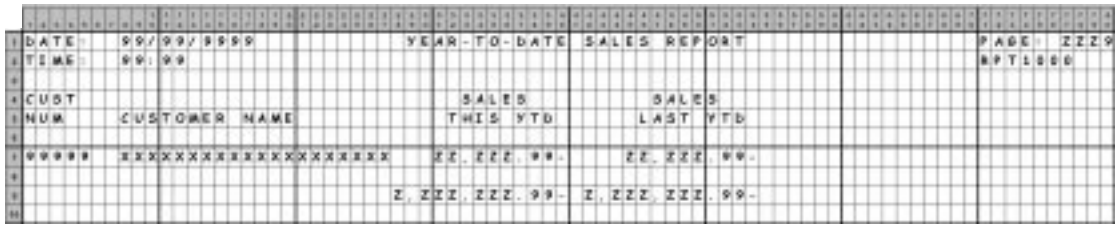
Where you see words like YEAR-TO-DATE SALES REPORT in the print chart, the words are supposed to be printed in those print positions. And where you see COBOL notation like X(20) or ZZ,ZZZ.99-, data is supposed to be printed. The other areas of the report are supposed to be left blank.

When you use a line printer on a mainframe, the printer usually has a line that is 132 characters long and the standard report form provides for a maximum of 66 lines on each page (6 lines per inch on an 11-inch form). In that case, each print line that's defined by your program must be 132 characters. You usually leave a few lines at the top and bottom of each page, though, so you print on approximately 60 lines of each page. In contrast, when you use a laser printer, the line length and number of lines per page can be adjusted to whatever is appropriate for the report.

In this example, the report is just 80 characters wide. If you develop the report for a mainframe, though, you need to fill out the other 52 characters in each line with spaces. In the program that follows, you'll see that 132 characters are printed in each line with a maximum of 55 customer lines on each page. If you add the six heading lines to the 55 lines, that means a total of 61 lines will be printed on each page.

When you develop a COBOL program that prepares a report, you need to realize that a printed report is treated as a file, and each line in the report is treated as one record in the file. In addition, the printed report is usually saved to a disk file before it is printed. This type of file can be called a *print file*.

The print chart



The printed output for the first test run

```
DATE:  01/26/2000          YEAR-TO-DATE SALES REPORT                PAGE:    1
TIME:  16:34                                                        RPT1000

CUST                                    SALES         SALES
NUM    CUSTOMER NAME                  THIS YTD      LAST YTD

11111  INFORMATION BUILDERS          1,234.56      1,111.11
12345  CAREER TRAINING CTR          12,345.67     22,222.22
22222  HOMELITE TEXTRON CO          34,545.00          0.00
34567  NEAS MEMBER BENEFITS            111.11          0.00
55555  PILOT LIFE INS. CO.          10,000.00      1,000.00
00111  DAUPHIN DEPOSIT BANK         14,099.00     19,930.00
54321  AIRCRAFT OWNERS ASSC          5,426.12     40,420.00
33333  NORFOLK CORP                  6,396.35      4,462.88
12121  GENERAL SERVICES CO.         11,444.00     11,059.56
24680  INFO MANAGEMENT CO.          17,481.45     11,892.47
99999  DOLLAR SAVINGS BANK           5,059.00      4,621.95
76543  NATL MUSIC CORP.              2,383.46      4,435.26

                                   120,525.72    121,155.45
```

Description

- A *print chart* is used to plan the format of a report. If you compare the output of the first test run with the print chart, you can see how they are related.
- A report usually has one or more heading lines that identify the report and provide information such as the date, time, and page number. Heading lines are also used to identify the data in the columns of the report.
- On a line printer for a mainframe, each line typically consists of 132 characters. Also, the standard report form provides for a maximum of 66 lines on each page.
- On a laser printer, the line length and number of lines on each page can be adapted to the requirements of the report.
- In COBOL, a printed report is thought of as a file, and each line in the report is one record in the file. In addition, the report is usually saved in a disk file before it is printed. This type of file can be called a *print file*.

Figure 3-2 The report specifications for the program

The COBOL code

If you understand what this program is supposed to do, you're ready to preview the code, which is presented in the four parts of figure 3-3. Although this program requires six new statements and one function (which are shaded), you shouldn't have any trouble understanding what they do. As a result, you should be able to understand how this program works before you learn the coding details for the new statements and function.

The Input-Output Section in the Environment Division

When your program works with disk and print files, it must contain a File-Control paragraph in the Environment Division. This paragraph contains one Select statement for each disk and print file. Each Select statement identifies the file by giving it a filename.

The File Section in the Data Division

For each Select statement you code in the Environment Division, you must code an FD (File Description) statement in the File Section of the Data Division. This statement is followed immediately by the record description for the file.

In the program in this figure, the FD statement for the customer master file is followed by a record description that includes the definitions for the six fields that make up the record. In contrast, the FD statement for the print file is followed by a record description named PRINT-AREA that is 132 characters long (the length of the print line). No fields are defined within this record, because the various lines that will be printed to this file are defined in the Working-Storage Section. You'll soon see how this works.

The end-of-file switch

In the Working-Storage Section of this program, you can see a SWITCHES group with one switch defined. It is named CUSTMAST-EOF-SWITCH, which is short for "customer master end-of-file switch." Notice that the Value clause for this switch initializes it to a value of N, which means the switch is off. Then, the switch is turned on when all of the records in the file have been read.

As you'll see as you progress through this book, all business programs require one or more switches. In programming jargon, a *switch* is a field that can have either a yes or no (or on or off) value. To simplify, we recommend that you define all switch fields as PIC X and that you treat a value of Y as Yes (or On) and a value of N as No (or Off).

The print-fields group

The four fields in the PRINT-FIELDS group are used to help control the printing of the report. Here, PAGE-COUNT keeps track of the current page number; LINES-ON-PAGE represents the number of customer lines that will be printed on each page (+55); LINE-COUNT is used to count the lines on each page; and SPACE-CONTROL is used to determine how many lines will be advanced before the next line is printed.

The report-preparation program **Page 1**

```
 IDENTIFICATION DIVISION.
*
 PROGRAM-ID. RPT1000.
*
 ENVIRONMENT DIVISION.
*
 INPUT-OUTPUT SECTION.
*
 FILE-CONTROL.
*
     SELECT CUSTMAST ASSIGN TO CUSTMAST.
     SELECT SALESRPT ASSIGN TO SALESRPT.
*
 DATA DIVISION.
*
 FILE SECTION.
*
 FD  CUSTMAST.
*
 01  CUSTOMER-MASTER-RECORD.
     05  CM-BRANCH-NUMBER        PIC 9(2).
     05  CM-SALESREP-NUMBER      PIC 9(2).
     05  CM-CUSTOMER-NUMBER      PIC 9(5).
     05  CM-CUSTOMER-NAME        PIC X(20).
     05  CM-SALES-THIS-YTD       PIC S9(5)V9(2).
     05  CM-SALES-LAST-YTD       PIC S9(5)V9(2).
*
 FD  SALESRPT.
*
 01  PRINT-AREA      PIC X(132).
*
 WORKING-STORAGE SECTION.
*
 01  SWITCHES.
     05  CUSTMAST-EOF-SWITCH    PIC X   VALUE "N".
*
 01  PRINT-FIELDS.
     05  PAGE-COUNT      PIC S9(3)   VALUE ZERO.
     05  LINES-ON-PAGE   PIC S9(3)   VALUE +55.
     05  LINE-COUNT      PIC S9(3)   VALUE +99.
     05  SPACE-CONTROL   PIC S9.
*
 01  TOTAL-FIELDS.
     05  GRAND-TOTAL-THIS-YTD   PIC S9(7)V99   VALUE ZERO.
     05  GRAND-TOTAL-LAST-YTD   PIC S9(7)V99   VALUE ZERO.
*
 01  CURRENT-DATE-AND-TIME.
     05  CD-YEAR         PIC 9999.
     05  CD-MONTH        PIC 99.
     05  CD-DAY          PIC 99.
     05  CD-HOURS        PIC 99.
     05  CD-MINUTES      PIC 99.
     05  FILLER          PIC X(9).
*
```

Figure 3-3 The COBOL code for the program (part 1 of 4)

Note here that PAGE-COUNT has a starting value of zero because 1 will be added to it before the first page is printed. Note too that LINE-COUNT has a starting value of +99. Since that value is greater than the starting value of LINES-ON-PAGE, the program will start a new page before the first line of the report is printed. Although LINE-COUNT could be given any starting value greater than +55, +99 is used so it will work even if the starting value of LINES-ON-PAGE is increased later on.

The total-fields group

The TOTAL-FIELDS group on page 1 of the program defines two fields that are used for the grand totals of the report. They are given a starting value of zero. Then, as each record in the file is read, the last year-to-date and this year-to-date sales fields are added to the total fields. At the end of the program, these fields are moved to the grand total line and printed.

The current-date-and-time group

The last group on page 1 receives the current date and time fields when the program gets them from the computer system. These fields are then moved to the heading of the report.

The print-line groups

The six groups on page 2 of this program are the record descriptions for the six types of lines that are required by the sales report. The first four groups define the four heading lines. The next group defines the customer line. And the last group defines the grand total line. If you compare the definitions for the fields in these lines with the print chart, you'll see that they correspond directly to that layout.

Note that a prefix is used for each data name in these print lines to indicate which line the field belongs to. For instance, HL1 is the prefix for fields in the first heading line, and CL is the prefix for fields in the customer line. Later on, this makes it easier to code the statements in the Procedure Division that refer to those fields.

What's new here is the use of the word FILLER. When that reserved word is coded instead of a data name, it means that the field isn't going to be referred to by statements in the Procedure Division. Nevertheless, the field is required for a complete definition of the print line.

When you code a FILLER field, you need to give it a starting value of an alphanumeric literal or SPACE. Otherwise, leftover data from a previous program may be printed in the field. This is a common programming error, but one that's easy to fix.

Before you turn the page, you may want to take note of the definitions for the third and fourth heading lines. For each line, three 20-character FILLER fields are used to define the literals for the first 60 characters. This is a common programming technique for defining the literals that are needed in the heading of a report. Of course, you can code the literals for these lines in whatever style you prefer, but this style has some merit.

The report-preparation program **Page 2**

```
01  HEADING-LINE-1.
     05  FILLER           PIC X(7)    VALUE "DATE:  ".
     05  HL1-MONTH        PIC 9(2).
     05  FILLER           PIC X(1)    VALUE "/".
     05  HL1-DAY          PIC 9(2).
     05  FILLER           PIC X(1)    VALUE "/".
     05  HL1-YEAR         PIC 9(4).
     05  FILLER           PIC X(11)   VALUE SPACE.
     05  FILLER           PIC X(20)   VALUE "YEAR-TO-DATE SALES R".
     05  FILLER           PIC X(20)   VALUE "EPORT               ".
     05  FILLER           PIC X(8)    VALUE "  PAGE: ".
     05  HL1-PAGE-NUMBER  PIC ZZZ9.
     05  FILLER           PIC X(52)   VALUE SPACE.
*
 01  HEADING-LINE-2.
     05  FILLER           PIC X(7)    VALUE "TIME:  ".
     05  HL2-HOURS        PIC 9(2).
     05  FILLER           PIC X(1)    VALUE ":".
     05  HL2-MINUTES      PIC 9(2).
     05  FILLER           PIC X(58)   VALUE SPACE.
     05  FILLER           PIC X(10)   VALUE "RPT1000".
     05  FILLER           PIC X(52)   VALUE SPACE.
*
 01  HEADING-LINE-3.
     05  FILLER           PIC X(20)   VALUE "CUST                ".
     05  FILLER           PIC X(20)   VALUE "            SALES   ".
     05  FILLER           PIC X(20)   VALUE "      SALES         ".
     05  FILLER           PIC X(72)   VALUE SPACE.
*
 01  HEADING-LINE-4.
     05  FILLER           PIC X(20)   VALUE "NUM    CUSTOMER NAME".
     05  FILLER           PIC X(20)   VALUE "           THIS YTD ".
     05  FILLER           PIC X(20)   VALUE "     LAST YTD       ".
     05  FILLER           PIC X(72)   VALUE SPACE.
*
 01  CUSTOMER-LINE.
     05  CL-CUSTOMER-NUMBER   PIC 9(5).
     05  FILLER               PIC X(2)     VALUE SPACE.
     05  CL-CUSTOMER-NAME     PIC X(20).
     05  FILLER               PIC X(3)     VALUE SPACE.
     05  CL-SALES-THIS-YTD    PIC ZZ,ZZ9.99-.
     05  FILLER               PIC X(4)     VALUE SPACE.
     05  CL-SALES-LAST-YTD    PIC ZZ,ZZ9.99-.
     05  FILLER               PIC X(78)    VALUE SPACE.
*
 01  GRAND-TOTAL-LINE.
     05  FILLER               PIC X(27)   VALUE SPACE.
     05  GTL-SALES-THIS-YTD   PIC Z,ZZZ,ZZ9.99-.
     05  FILLER               PIC X(1)    VALUE SPACE.
     05  GTL-SALES-LAST-YTD   PIC Z,ZZZ,ZZ9.99-.
     05  FILLER               PIC X(78)   VALUE SPACE.
*
```

Figure 3-3 The COBOL code for the program (part 2 of 4)

The Procedure Division

Now that you're familiar with the records and fields defined in the Data Division, you should be able to follow the code in the Procedure Division. Yes, it contains a few new statements, but it's not hard to tell what these statements do. As in all programs, the real trick is following the logic of the program.

As you can see, procedure 000 starts with an Open statement that opens the two files used by a program. Then, after it prepares the report, this procedure issues a Close statement that closes the files. You'll learn how and why you code these statements in just a moment.

Between these statements, though, this procedure issues three Perform statements that drive the logic of the program. The first one performs procedure 100 to get the current date and time and move those fields into the report heading. The second Perform statement repeatedly performs procedure 200 to prepare the lines of the report until all of the records in the customer file have been read. And the third Perform statement performs procedure 300 to print the grand totals at the end of the report.

In procedure 200, the first Perform statement performs procedure 210 to read the next customer record. Then, if the customer end-of-file switch hasn't been turned on, this procedure performs procedure 220 to print a customer line.

To make this logic work, procedure 210 contains a Read statement that reads the next record in the file. However, if there are no more records in the file (AT END), this statement moves Y to the end-of-file switch. That's the switch that's tested by the Perform Until statement in procedure 000 and the If statement in procedure 200.

The report-preparation program **Page 3**

```
 PROCEDURE DIVISION.
*
 000-PREPARE-SALES-REPORT.
*
     OPEN INPUT  CUSTMAST
          OUTPUT SALESRPT.
     PERFORM 100-FORMAT-REPORT-HEADING.
     PERFORM 200-PREPARE-SALES-LINES
         UNTIL CUSTMAST-EOF-SWITCH = "Y".
     PERFORM 300-PRINT-GRAND-TOTALS.
     CLOSE CUSTMAST
           SALESRPT.
     STOP RUN.
*
 100-FORMAT-REPORT-HEADING.
*
     MOVE FUNCTION CURRENT-DATE TO CURRENT-DATE-AND-TIME.
     MOVE CD-MONTH   TO HL1-MONTH.
     MOVE CD-DAY     TO HL1-DAY.
     MOVE CD-YEAR    TO HL1-YEAR.
     MOVE CD-HOURS   TO HL2-HOURS.
     MOVE CD-MINUTES TO HL2-MINUTES.
*
 200-PREPARE-SALES-LINES.
*
     PERFORM 210-READ-CUSTOMER-RECORD.
     IF CUSTMAST-EOF-SWITCH = "N"
         PERFORM 220-PRINT-CUSTOMER-LINE.
*
 210-READ-CUSTOMER-RECORD.
*
     READ CUSTMAST
         AT END
             MOVE "Y" TO CUSTMAST-EOF-SWITCH.
*
```

Figure 3-3 The COBOL code for the program (part 3 of 4)

The last three procedures of the program print the customer line, the heading lines, and the grand-total lines of the report. They make use of Write statements that print a line after advancing to the next page of the report or to a later line in the report.

Procedure 220 starts with an If statement that tests whether the value of LINE-COUNT is greater than or equal to LINES-ON-PAGE. Since this condition is true when the program starts (+99 is greater than +55), procedure 220 performs procedure 230 to skip to the top of the next page and print the four heading lines.

In procedure 230, you can see that 1 is added to PAGE-COUNT (which had a starting value of zero) and that value is moved to the first heading line before any lines are printed. Then, each of the heading lines is moved to the record area named PRINT-AREA and a Write statement writes the line from that area to the print file. The first of these Write statements advances to the next page before printing its line; the second one advances one line before printing; the third one advances two lines before printing; and the fourth one advances one line before printing. That is consistent with the vertical spacing shown on the print chart.

After all four lines have been printed, procedure 230 moves zero to LINE-COUNT, which will be used in procedure 220 to count the customer lines printed on each page. Then, procedure 230 moves 2 to SPACE-CONTROL, which is used by procedure 220 to determine how many lines to advance before printing the next customer line.

Back in procedure 220, you can see that four fields in the customer master record are moved to the customer line. Then, the customer line is moved to PRINT-AREA, and the Write statement writes the line from that area…after advancing the number of lines that are specified by the SPACE-CONTROL field. After that, one is added to LINE-COUNT, and the sales fields in the customer record are added to the grand total fields. Then, one is moved to SPACE-CONTROL, which means that the next customer line will be printed after advancing only one line.

If you follow this code, procedure 300 is just more of the same. After the two grand total fields are moved to the grand total line, that line is moved to PRINT-AREA. Then, the Write statement writes that line after advancing 2 lines.

At this point, you should understand how this program works. But if you still have some questions about it, you'll get a chance to step through the program when you do exercise 3-1 at the end of this chapter. Before that, though, you'll learn the coding specifics for the new statements and the function that this program uses.

The report-preparation program **Page 4**

```
 220-PRINT-CUSTOMER-LINE.
*
     IF LINE-COUNT >= LINES-ON-PAGE
         PERFORM 230-PRINT-HEADING-LINES.
     MOVE CM-CUSTOMER-NUMBER   TO CL-CUSTOMER-NUMBER.
     MOVE CM-CUSTOMER-NAME     TO CL-CUSTOMER-NAME.
     MOVE CM-SALES-THIS-YTD    TO CL-SALES-THIS-YTD.
     MOVE CM-SALES-LAST-YTD    TO CL-SALES-LAST-YTD.
     MOVE CUSTOMER-LINE TO PRINT-AREA.
     WRITE PRINT-AREA AFTER ADVANCING SPACE-CONTROL LINES.
     ADD 1 TO LINE-COUNT.
     ADD CM-SALES-THIS-YTD TO GRAND-TOTAL-THIS-YTD.
     ADD CM-SALES-LAST-YTD TO GRAND-TOTAL-LAST-YTD.
     MOVE 1 TO SPACE-CONTROL.
*
 230-PRINT-HEADING-LINES.
*
     ADD 1 TO PAGE-COUNT.
     MOVE PAGE-COUNT     TO HL1-PAGE-NUMBER.
     MOVE HEADING-LINE-1 TO PRINT-AREA.
     WRITE PRINT-AREA AFTER ADVANCING PAGE.
     MOVE HEADING-LINE-2 TO PRINT-AREA.
     WRITE PRINT-AREA AFTER ADVANCING 1 LINES.
     MOVE HEADING-LINE-3 TO PRINT-AREA.
     WRITE PRINT-AREA AFTER ADVANCING 2 LINES.
     MOVE HEADING-LINE-4 TO PRINT-AREA.
     WRITE PRINT-AREA AFTER ADVANCING 1 LINES.
     MOVE ZERO TO LINE-COUNT.
     MOVE 2 TO SPACE-CONTROL.
*
 300-PRINT-GRAND-TOTALS.
*
     MOVE GRAND-TOTAL-THIS-YTD TO GTL-SALES-THIS-YTD.
     MOVE GRAND-TOTAL-LAST-YTD TO GTL-SALES-LAST-YTD.
     MOVE GRAND-TOTAL-LINE TO PRINT-AREA.
     WRITE PRINT-AREA AFTER ADVANCING 2 LINES.
```

Figure 3-3 The COBOL code for the program (part 4 of 4)

How to code Select statements and FD statements

For each file that a program reads or writes, you need to code one Select statement in the Environment Division and one FD statement in the Data Division. Because you code a Select statement for Personal COBOL differently than you do for mainframe COBOL, the syntax you use for each platform is presented separately.

How to code Select statements for Personal COBOL

Figure 3-4 shows how to code Select statements for sequential files when using Micro Focus Personal COBOL. This statement gives a *filename* to each file that can be used to refer to the file within the program. It also includes a *system name* that identifies the specific file on disk.

In the first example, you can see the Select statement for the customer master file. Here, CUSTMAST is the filename that will be used to refer to this file throughout the program. The name of the file on disk is custmast.dat, which should be located in the folder that's identified by this path:

```
c:\cobol\data
```

If the file isn't there when the program tries to open it, a run-time error occurs.

Similarly, the Select statement for the print file gives it a name of SALESRPT. When it is saved on disk, it will receive the name salesrpt.prn and it will be stored in the same folder as the customer file.

How do you print a report that's stored in a print file? You can use your word processing program or a text editor like NotePad. You'll learn more about that in figure 3-15.

The syntax of the Select statement for a sequential file on a PC

```
SELECT filename ASSIGN TO system-name
```

Typical Select statements for Micro Focus Personal COBOL

```
SELECT CUSTMAST ASSIGN TO "c:\cobol\data\custmast.dat".
SELECT SALESRPT ASSIGN TO "c:\cobol\data\salesrpt.prn".
```

The rules for forming a filename in standard COBOL

- Use letters, digits, and hyphens only.
- Don’t start or end the name with a hyphen.
- Use a maximum of 30 characters.
- Use at least one letter in the name.

Description

- A Select statement identifies a disk file or a print file used by the program.
- The system name for a disk file or print file consists of the path and filename for the file.
- Because Personal COBOL uses the Windows 3.1 interface, you need to keep the file name in the system name to a maximum of 8 characters. You should also use *dat* as the extension for data files and *prn* as the extension for print files.
- Although you can also code the system name for a print file so the output goes directly to a printer, it’s better to code the system name so it is saved in a disk file. Then, you can use your word processor or NotePad to print the output or to review the output on your monitor without ever printing it.

Figure 3-4 How to code Select statements for Micro Focus Personal COBOL

How to code Select statements on a mainframe

Figure 3-5 shows you how to code Select statements for sequential files when you're using COBOL on an IBM mainframe. In this case, the system name doesn't point to a specific file. Instead, the system name includes a *ddname* that gets associated with a specific file or printer when the program is run. This association is done by job control language, which you'll learn how to use in chapter 18.

When you code a Select statement for a mainframe, we recommend that the filename and the ddname in each statement be the same. That way, you won't have any trouble remembering which is which. By keeping the filename short, you also make it easier to code switch names like CUSTMAST-EOF-SWITCH that relate to the file. That's why all of the programs in this book use filenames that are eight or fewer characters.

As you can see in this figure, a mainframe provides for two types of sequential files: *VSAM* (*Virtual Storage Access Method*) files and non-VSAM files. Today, most sequential files are non-VSAM files, even though VSAM is the newer file organization method.

When a print file is prepared on a mainframe, it is stored in a queue (waiting list) with other print files. Then, the jobs in the print queue are printed one at a time as the printer becomes available. This is done automatically in a process called *spooling*. Alternately, you can hold the print file in the queue so it isn't printed until you release it. Then, you can review the print file on your screen and decide whether you want to print it. You can learn more about that in chapter 18.

The syntax of the Select statement for a sequential file

```
SELECT filename ASSIGN TO system-name
```

Typical Select statements for an IBM mainframe compiler

```
SELECT CUSTMAST ASSIGN TO AS-CUSTMAST.
SELECT SALESRPT ASSIGN TO SALESRPT.
```

The rules for forming a filename in standard COBOL

- Use letters, digits, and hyphens only.
- Don't start or end the name with a hyphen.
- Use a maximum of 30 characters.
- Use at least one letter in the name.

The syntax for a system name on an IBM mainframe

For a non-VSAM sequential disk file or a print file

```
ddname
```

For a VSAM sequential disk file

```
AS-ddname
```

Rules for forming a ddname

Use eight or fewer letters or digits, starting with a letter. For consistency, this should be the same as the filename.

Description

- The Select statement identifies a disk file or a print file used by the program.
- The ddname that's coded as part of the system name for a disk file is used in job control language when the program is run to assign the file in the COBOL program to a specific file on disk.
- IBM mainframes use two types of sequential disk files: VSAM and non-VSAM. Although VSAM is the newer file organization, non-VSAM sequential files are commonly used because they're easier to work with.
- The ddname that's coded for a print file is used in job control language when the program is run to assign the printed output to a specific printer or group of printers. Before the output is actually printed, though, it's written to a temporary disk file that's printed when a printer becomes available. This is called *spooling*.

Figure 3-5 How to code Select statements on a mainframe

How to code FD statements and record descriptions for disk files

Figure 3-6 shows how to write FD (File Description) statements for sequential disk files. In the FD statement itself, you identify the file using the filename from the Select statement for the file. After the FD statement, you code a record description for the file.

You also can code a RECORD CONTAINS clause in an FD statement. This clause specifies the number of characters (or bytes) in each record in the file. If you omit this clause, though, the compiler determines the number of characters in each record from the record description that follows. As a result, most programmers don't code this clause.

Because there's nothing in a disk file to indicate where one field in a record starts and ends, it's essential that the record description correspond with the records on the disk. If, for example, the record description calls for a two-digit branch number and the branch number on the disk is three digits, the branch number field in the program won't receive the right data when a record is read. Worse, none of the fields that follow that field in the disk record will receive the right data either. This usually leads to a run-time error.

How to code FD statements and record descriptions for print files

Figure 3-6 also shows how to code an FD statement and record description for a print file. In contrast to the FD statement for a disk file, individual fields aren't usually defined in the record description for a print file. That's because records of varying formats will be written to the print file, one for each type of line in the report. In programmer jargon, this record description area is often referred to as the *print area* so that's the name we use in all of our report-preparation programs.

The syntax of the FD statement for a sequential file

```
FD  filename
    [RECORD CONTAINS integer CHARACTERS]
```

A typical FD statement and record description for a disk file

```
 FD  CUSTMAST.
*
 01  CUSTOMER-MASTER-RECORD.
     05  CM-BRANCH-NUMBER        PIC 9(2).
     05  CM-SALESREP-NUMBER      PIC 9(2).
     05  CM-CUSTOMER-NUMBER      PIC 9(5).
     05  CM-CUSTOMER-NAME        PIC X(20).
     05  CM-SALES-THIS-YTD       PIC S9(5)V9(2).
     05  CM-SALES-LAST-YTD       PIC S9(5)V9(2).
```

An FD statement that indicates the number of characters in each record

```
 FD  CUSTMAST
     RECORD CONTAINS 43 CHARACTERS.
```

A typical FD statement and record description for a print file

```
 FD  SALESRPT.
*
 01  PRINT-AREA      PIC X(132).
```

Description

- The FD statement provides a File Description. The filename used in this statement must be the same as the one coded in the Select statement in the Environment Division.
- Right after the FD statement for a file, you code a record description for the file. For a disk file, you often code a record description with a group item at the 01 level and all of the fields defined at the lower levels. For a print file, you usually code a one-line record description that gives the number of characters in the record.
- If you code the Record Contains clause, the number of characters you specify must be the same as the number of bytes in the record description for the file. If you omit this clause, the compiler determines the number of characters from the record description.

Figure 3-6 How to code FD statements and record descriptions for disk and print files

How to code the Procedure Division

To code the Procedure Division of a report-preparation program, you need to learn how to code just four new statements that work with files: Open, Close, Read, and Write. The figures that follow show you how to code these statements. After that, you'll learn how to use the Current-Date function so you can include the current date and time in your report headings. You'll also learn how to use the Accept statement to get the current date and time in case you're using a compiler that doesn't support the Current-Date function.

How to code Open and Close statements

Figure 3-7 shows how to code the Open and Close statements. Within these statements, you code the filenames that you defined in the Select statements. As you can see, you can open both disk files and print files with a single statement, and you can close them with a single statement.

What exactly do the Open and Close statements do? That varies from one platform to another. For an *input file* (a file the program reads), the Open statement at least checks to see whether the file is available. If it isn't, a run-time error occurs and the program ends. For an *output file* (a file the program writes to), the Open statement at least checks to make sure that there's enough available space for the file on the disk before opening it. In contrast, the Close statement releases the files so they can be used by other programs.

When you code a program that uses files, you have to make sure the files are opened before your program tries to read or write them. And you have to make sure they're closed before the program issues a Stop Run statement. Otherwise, run-time errors will occur.

You can, however, open and close a file more than once in a single program. You can do that when you want to read through the records in a file more than once. When you open a file the second time, you can read it again starting with the first record.

The syntax of the Open statement

```
OPEN INPUT  filename-1 ...
     OUTPUT filename-2 ...
```

The syntax of the Close statement

```
CLOSE filename ...
```

The Open and Close statements in the report-preparation program

```
OPEN INPUT  CUSTMAST
     OUTPUT SALESRPT.
CLOSE CUSTMAST
      SALESRPT.
```

An Open statement with two input files

```
OPEN INPUT  CUSTMAST
            BRCHMAST
     OUTPUT SALESRPT.
```

Description

- The filenames that you use in the Open and Close statements have to be the ones that you defined in the Select statements in the Environment Division.
- An Open statement has to be issued for a file before a program can issue a Read or Write statement for that file.
- A Close statement has to be issued for all open files before a program can issue a Stop Run statement.
- After you close a file, it is no longer available for processing. However, if you open an input file again, you can read the records again starting with the first record.

Figure 3-7 How to code Open and Close statements

How to code Read statements for sequential disk files

Figure 3-8 shows how to code Read statements. When a program issues a Read statement, it reads the next record in the file starting with the first record in the file. When there are no more records in the file, though, no record is read and the At End clause of the statement is executed. In this clause, you normally turn an EOF switch on to indicate that there are no more records in the file.

You can also code a Not At End clause on a Read statement. This is illustrated by the second example in this figure. Here, the Not At End clause contains a statement that adds 1 to a field named RECORD-COUNT.

How to code Write statements for print files

Figure 3-8 also summarizes the information you need for coding Write statements. The first thing you should notice here is that you code the name used in the record description for the file on this statement, not the filename as you do for a Read statement. The three Write statements shown in this figure all write the record description named PRINT-AREA.

The After Advancing clause of the Write statement determines how many lines are skipped before the next line is printed on the report. As you can see in the examples, you can code the reserved word PAGE in this clause to skip to the top of the next page before printing. You can code a literal value with or without the reserved word LINES (or LINE) to skip one or more lines before printing. And you can code a variable name with or without the reserved word LINES (or LINE) to skip the number of lines indicated by the variable value before printing.

If you're used to working with PC programs that print full pages at a time on a laser printer, you may have trouble imagining the paper skipping one or more lines before a line is printed. Remember, then, that COBOL was developed when all printers worked that way, and many printers today still do. That's why that type of printer is called a line printer. So even if you're using a laser printer, it helps to imagine the program printing one line at a time after the paper has been advanced one or more lines.

The syntax of the Read statement for a sequential disk file

```
READ filename [RECORD]
    AT END
        imperative-statement-1 ...
    [NOT AT END
        imperative-statement-2 ...]
```

Some typical Read statements for sequential disk files

```
READ CUSTMAST RECORD
    AT END
        MOVE "Y" TO CUSTMAST-EOF-SWITCH.

READ CUSTMAST
    AT END
        MOVE "Y" TO CUSTMAST-EOF-SWITCH
    NOT AT END
        ADD 1 TO RECORD-COUNT.
```

The syntax of the Write statement for a print file

```
WRITE record-name
                      { PAGE                          }
    AFTER ADVANCING   { integer [ LINE | LINES ]      }
                      { data-name [ LINE | LINES ]    }
```

Some typical Write statements for print files

```
WRITE PRINT-AREA
    AFTER ADVANCING PAGE.

WRITE PRINT-AREA
    AFTER ADVANCING 1 LINE.

WRITE PRINT-AREA
    AFTER ADVANCING SPACE-CONTROL LINES.
```

Description

- When the Read statement is executed for a sequential disk file, it reads the next record in sequence into the record description for the file. If there are no more records in the file, the At End clause is executed; otherwise, the Not At End clause is executed.
- Note that you can use only imperative statements in the At End and Not At End clauses. This means you can't code If statements in these clauses.
- When the Write statement is executed for a print file, one record is printed from the print area for the file and the paper is advanced the number of lines indicated by the After Advancing clause or to the top of the next page (PAGE).
- Note that you code the filename in a Read statement and the record name in a Write statement.

Figure 3-8 How to code Read and Write statements

How to use the Current-Date function to get the current date and time

If you're using a COBOL-85 compiler that supports the 1989 addendum to the standards, you can use the Current-Date function to get the current date and time from your computer. Although most COBOL-85 compilers support the 1989 functions, VS COBOL II and COBOL for MVS on IBM mainframes don't. So if you're using one of these compilers, you have to use the Accept statement to get the date and time as shown in the next figure.

Figure 3-9 presents the Current-Date function. You can use this function as if it's a variable in any statement where it makes sense. When you use it in a Move statement, for example, the current date and time are moved to the receiving field.

In the data description in this figure, you can see the eight fields that you get from this function. Normally, though, you use only the first five or six fields in a program. The program in this chapter, for example, uses only the first five fields in the heading lines of the report. As a result, the last three fields are defined as FILLER.

The syntax of the Current-Date function

```
FUNCTION CURRENT-DATE
```

The data description for the data returned by the Current-Date function

```
01  CURRENT-DATE-AND-TIME.
    05  CD-YEAR                         PIC 9(4).
    05  CD-MONTH                        PIC 9(2).
    05  CD-DAY                          PIC 9(2).
    05  CD-HOURS                        PIC 9(2).
    05  CD-MINUTES                      PIC 9(2).
    05  CD-SECONDS                      PIC 9(2).
    05  CD-HUNDREDTH-SECONDS            PIC 9(2).
    05  CD-GREENWICH-MEAN-TIME-SHHMM    PIC X(5).
```

Two statements that use the Current-Date function

```
DISPLAY FUNCTION CURRENT-DATE.
MOVE FUNCTION CURRENT-DATE TO CURRENT-DATE-AND-TIME.
```

Description

- In 1989, an addendum was added to the COBOL-85 standards that provided for *intrinsic functions*. Although they were treated as an optional feature, most modern COBOL-85 compilers include these functions. You'll learn more about intrinsic functions in chapter 7.
- The Current-Date function gets the current date and time. The four-digit year that's returned by this function makes it easy to display years like 2000 in a form that makes sense.
- The Current-Date function can be used to represent a single variable in any statement where that makes sense.
- Before the Current-Date function became available, you had to use the Accept Date and Accept Time statements to get the date and time. You can learn more about these statements in the next figure.
- The five-character Greenwich Mean Time that is returned by this function indicates the number of hours and minutes that the current time is ahead or behind Greenwich Mean Time (+hhmm or –hhmm). If this information isn't available, this function puts zeros in this field.

Figure 3-9 How to use the Current-Date function

How to use Accept statements to get the current date and time

Figure 3-10 shows how to get the current date and time if your compiler doesn't support the Current-Date function. To get the current date, you use the Accept Date statement. To get the current time, you use the Accept Time statement. Once you get the date and time, you can move the components of these data items to the heading lines of a report just as you do when you use the Current-Date function.

In the syntax for the Accept Date statement, the shaded portion is an IBM extension that became available with the COBOL for MVS compiler. When this is coded (YYYYMMDD), the Accept Date statement gets the date with a four-digit year. Without that extension, the Accept Date statement gets the date with a two-digit year.

Since the VS COBOL II compiler doesn't provide for the YYYYMMDD extension, you can't get the current date with a four-digit year when using that compiler. That means you have to use special coding to determine what century a year belongs to now that we've reached the year 2000. Because of that limitation, most IBM mainframe shops upgraded to the COBOL for MVS or COBOL for OS/390 compiler during the last few years. That made it easier for them to upgrade their old programs so they are Y2K-compliant.

The syntax of the Accept statement for getting the date and time

```
ACCEPT data-name FROM { DATE [YYYYMMDD] }
                      { TIME            }
```

VS COBOL II code that gets the date with a two-digit year

```
01  CURRENT-DATE-AND-TIME.
    05  CD-CURRENT-DATE.
        10  CD-CURRENT-YEAR        PIC 99.
        10  CD-CURRENT-MONTH       PIC 99.
        10  CD-CURRENT-DAY         PIC 99.
    .
    .
    ACCEPT CD-CURRENT-DATE FROM DATE.
```

COBOL for MVS or OS/390 code that gets the date with a four-digit year

```
01  CURRENT-DATE-AND-TIME.
    05  CD-CURRENT-DATE.
        10  CD-CURRENT-YEAR        PIC 9999.
        10  CD-CURRENT-MONTH       PIC 99.
        10  CD-CURRENT-DAY         PIC 99.
    .
    .
    ACCEPT CD-CURRENT-DATE FROM DATE YYYYMMDD.
```

IBM mainframe code that gets the time

```
01  CURRENT-DATE-AND-TIME.
    05  CD-CURRENT-TIME.
        10  CD-CURRENT-HOURS       PIC 99.
        10  CD-CURRENT-MINUTES     PIC 99.
        10  CD-CURRENT-SECONDS     PIC 99.
        10  CD-CURRENT-HUNDREDTHS  PIC 99.
    .
    .
    ACCEPT CD-CURRENT-TIME FROM TIME.
```

Description

- If your compiler doesn't support the Current-Date function, you need to use the Accept Date and Accept Time statements to get the date and time.
- With the VS COBOL II compiler, you can't get the date with a four-digit year because the YYYYMMDD option isn't available. With the COBOL for MVS compiler, you can use the YYYYMMDD option to get the date with a four-digit year. With the COBOL for OS/390 compiler, you can use either the YYYYMMDD option or the Current-Date function to get the date with a four-digit year.

Figure 3-10 How to use Accept statements to get the current date and time

An enhanced version of the report-preparation program

If you study the program in figure 3-3, you should now be able to figure out how every statement in it works and why every statement is needed. Of course, there are other ways in which this program can be coded so it gets the same results, and you'll see some of these variations in the next chapter.

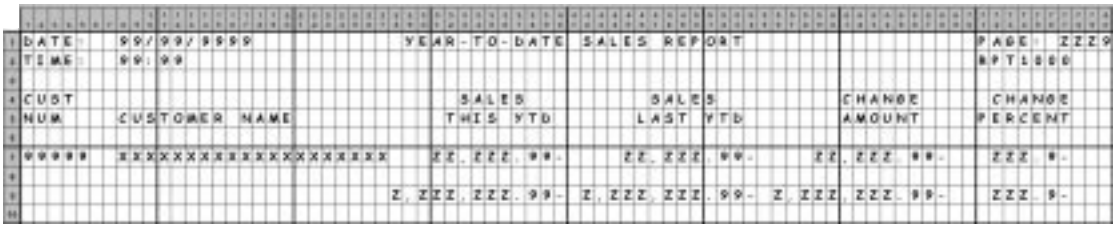
Now, if you feel confident that you understand the first version of the report-preparation program, it's time to make some enhancements to this program. These enhancements will make the program more realistic. They will also demonstrate some new coding techniques.

The enhanced report specifications

Figure 3-11 presents the report specifications for the enhanced program. Here, you can see that two columns have been added to the report. The first shows the change in sales from this year to last year; the second shows the change as a percent. Both of these are calculated from the input fields. In addition, this report should print a customer line only when the current year-to-date sales for the customer are greater than or equal to $10,000.

Before you turn the page, try to visualize the changes that need to be made to the original program. What needs to be added to the Data Division? What needs to be added to the Procedure Division?

The print chart for the enhanced report



The printed output for the test run

```
DATE:  01/27/2000          YEAR-TO-DATE SALES REPORT                  PAGE:    1
TIME:  13:52                                                          RPT1000

CUST                                  SALES         SALES         CHANGE     CHANGE
NUM    CUSTOMER NAME                THIS YTD      LAST YTD       AMOUNT    PERCENT

12345  CAREER TRAINING CTR         12,345.67     22,222.22      9,876.55-    44.4-
22222  HOMELITE TEXTRON CO         34,545.00          0.00     34,545.00    999.9
55555  PILOT LIFE INS. CO.         10,000.00      1,000.00      9,000.00    900.0
00111  DAUPHIN DEPOSIT BANK        14,099.00     19,930.00      5,831.00-    29.3-
12121  GENERAL SERVICES CO.        11,444.00     11,059.56        384.44     3.5
24680  INFO MANAGEMENT CO.         17,481.45     11,892.47      5,588.98    47.0

                                   99,915.12     66,104.25     33,810.87    51.1
```

Description

- Two columns of data have been added to the print chart. Both of these are calculated from the data in the input records.
- Instead of printing one line for each record in the file, this report lists only those customers with this year-to-date sales that are $10,000 or greater.
- If the last year-to-date field for a customer is zero, the program should print 999.9 in the change percent column. The program should also print 999.9 in this column if the result of the calculation is too large for the calculated field.

Figure 3-11 The report specifications for the enhanced program

The changes to the Data Division code

Figure 3-12 presents just the code that illustrates the changes that need to be made to the Data Division of the program. Here, the lines that have been added to the program are shaded along with any changes to the original lines.

In the Working-Storage Section, you can see a new group item called CALCULATED-FIELDS. It consists of an elementary item called CHANGE-AMOUNT that will receive the result of a calculation that's done for each customer line in the report. Because most programs require more than one calculated field, a group like this makes it easier to find those fields.

Then, in the print line descriptions, you can see that the required headings and fields have been added to the code. The locations of those headings and fields, of course, are based on the specifications in the print chart. To complete the program, the statements in the Procedure Division have to make the right calculations and move the right data to the new fields before the lines are printed.

The Data Division changes

```
 01  CALCULATED-FIELDS.
     05  CHANGE-AMOUNT           PIC S9(7)V99.
*
     .
     .
*
 01  HEADING-LINE-3.
     05  FILLER           PIC X(20)   VALUE "CUST                ".
     05  FILLER           PIC X(20)   VALUE "            SALES   ".
     05  FILLER           PIC X(20)   VALUE "      SALES         ".
     05  FILLER           PIC X(20)   VALUE "CHANGE     CHANGE   ".
     05  FILLER           PIC X(52)   VALUE SPACE.
*
 01  HEADING-LINE-4.
     05  FILLER           PIC X(20)   VALUE "NUM    CUSTOMER NAME".
     05  FILLER           PIC X(20)   VALUE "           THIS YTD ".
     05  FILLER           PIC X(20)   VALUE "     LAST YTD       ".
     05  FILLER           PIC X(20)   VALUE "AMOUNT    PERCENT   ".
     05  FILLER           PIC X(52)   VALUE SPACE.
*
 01  CUSTOMER-LINE.
     05  CL-CUSTOMER-NUMBER   PIC X(5).
     05  FILLER               PIC X(2)       VALUE SPACE.
     05  CL-CUSTOMER-NAME     PIC X(20).
     05  FILLER               PIC X(3)       VALUE SPACE.
     05  CL-SALES-THIS-YTD    PIC ZZ,ZZ9.99-.
     05  FILLER               PIC X(4)       VALUE SPACE.
     05  CL-SALES-LAST-YTD    PIC ZZ,ZZ9.99-.
     05  FILLER               PIC X(4)       VALUE SPACE.
     05  CL-CHANGE-AMOUNT     PIC ZZ,ZZ9.99-.
     05  FILLER               PIC X(3)       VALUE SPACE.
     05  CL-CHANGE-PERCENT    PIC ZZ9.9-.
     05  FILLER               PIC X(55)      VALUE SPACE.
*
 01  GRAND-TOTAL-LINE.
     05  FILLER               PIC X(27)      VALUE SPACE.
     05  GTL-SALES-THIS-YTD   PIC Z,ZZZ,ZZ9.99-.
     05  FILLER               PIC X(1)       VALUE SPACE.
     05  GTL-SALES-LAST-YTD   PIC Z,ZZZ,ZZ9.99-.
     05  FILLER               PIC X(1)       VALUE SPACE.
     05  GTL-CHANGE-AMOUNT    PIC Z,ZZZ,ZZ9.99-.
     05  FILLER               PIC X(3)       VALUE SPACE.
     05  GTL-CHANGE-PERCENT   PIC ZZ9.9-.
     05  FILLER               PIC X(55)      VALUE SPACE.
*
```

Figure 3-12 The Data Division changes for the enhanced program

The changes to the Procedure Division code

In figure 3-13, you can see the additions and changes that need to be made to the Procedure Division. In procedure 200, you can see that the If statement has been changed to a nested If statement. If the EOF switch is off, the nested If statement checks to see whether the value of sales this YTD (year-to-date) in the current customer record is greater than or equal to 10000. If it is, procedure 220 is performed to print a customer line on the report.

Once you've got this program logic coded, it's relatively easy to add the statements that the other enhancements require. In procedure 220, for example, the first Compute statement calculates the change amount for each line, and a Move statement moves it to the print line. Then, an If statement checks whether the last YTD value is zero. If it is, the program moves 999.9 to the change percent field. If it isn't, the Compute statement calculates the change percent. Notice that this statement includes an On Size Error clause that sets the change percent field to 999.9 if the result is too large for the field. That could happen if last YTD sales are small compared to this YTD sales. After the If statement, the procedure continues as before and prints the customer line, which now includes the new fields.

Before you go on, you should understand why the Compute statement in this figure is coded the way it is. When we first wrote this program in Personal COBOL, we coded the Compute statement like this because that seemed to be the logical way to code it:

```
COMPUTE CL-CHANGE-PERCENT ROUNDED =
    CHANGE-AMOUNT / CM-SALES-LAST-YTD * 100
```

Although this worked right with Personal COBOL, this didn't get the right result when using the mainframe compilers. The problem has to do with the intermediate result field that the mainframe compilers use to hold the result of the divide operation. By switching the sequence of operations as shown in this figure, though, this Compute statement gets the right result for both PC and mainframe compilers. In chapter 7, you can learn how the intermediate result fields are defined by the mainframe compilers so you'll be better able to handle this type of problem.

Since procedure 300 requires similar calculations and coding, the best way to add those statements to procedure 300 is to copy them from procedure 220. Then, you can modify the statements so they use the appropriate data names.

Note that the CHANGE-AMOUNT field is used in both procedure 220 and procedure 300. It is also used with the Compute statement that calculates the change percent. That's why CHANGE-AMOUNT can't be defined within a print line as a numeric edited item like the change percent fields are.

The Procedure Division changes

```
 200-PREPARE-SALES-LINES.
*
     PERFORM 210-READ-CUSTOMER-RECORD.
     IF CUSTMAST-EOF-SWITCH = "N"
         IF CM-SALES-THIS-YTD >= 10000
             PERFORM 220-PRINT-CUSTOMER-LINE.
*

 220-PRINT-CUSTOMER-LINE.
*
     IF LINE-COUNT GREATER LINES-ON-PAGE
         PERFORM 230-PRINT-HEADING-LINES.
     MOVE CM-CUSTOMER-NUMBER  TO CL-CUSTOMER-NUMBER.
     MOVE CM-CUSTOMER-NAME    TO CL-CUSTOMER-NAME.
     MOVE CM-SALES-THIS-YTD   TO CL-SALES-THIS-YTD.
     MOVE CM-SALES-LAST-YTD   TO CL-SALES-LAST-YTD.
     COMPUTE CHANGE-AMOUNT =
         CM-SALES-THIS-YTD - CM-SALES-LAST-YTD.
     MOVE CHANGE-AMOUNT TO CL-CHANGE-AMOUNT.
     IF CM-SALES-LAST-YTD = ZERO
         MOVE 999.9 TO CL-CHANGE-PERCENT
     ELSE
         COMPUTE CL-CHANGE-PERCENT ROUNDED =
             CHANGE-AMOUNT * 100 / CM-SALES-LAST-YTD
             ON SIZE ERROR
                 MOVE 999.9 TO CL-CHANGE-PERCENT.
     MOVE CUSTOMER-LINE TO PRINT-AREA.
     WRITE PRINT-AREA AFTER ADVANCING SPACE-CONTROL LINES.
     ADD 1 TO LINE-COUNT.
     ADD CM-SALES-THIS-YTD TO GRAND-TOTAL-THIS-YTD.
     ADD CM-SALES-LAST-YTD TO GRAND-TOTAL-LAST-YTD.
     MOVE 1 TO SPACE-CONTROL.
*

 300-PRINT-GRAND-TOTALS.
*
     MOVE GRAND-TOTAL-THIS-YTD TO GTL-SALES-THIS-YTD.
     MOVE GRAND-TOTAL-LAST-YTD TO GTL-SALES-LAST-YTD.
     COMPUTE CHANGE-AMOUNT =
         GRAND-TOTAL-THIS-YTD - GRAND-TOTAL-LAST-YTD.
     MOVE CHANGE-AMOUNT TO GTL-CHANGE-AMOUNT.
     IF GRAND-TOTAL-LAST-YTD = ZERO
         MOVE 999.9 TO GTL-CHANGE-PERCENT
     ELSE
         COMPUTE GTL-CHANGE-PERCENT ROUNDED =
             CHANGE-AMOUNT * 100 / GRAND-TOTAL-LAST-YTD
             ON SIZE ERROR
                 MOVE 999.9 TO GTL-CHANGE-PERCENT.
     MOVE GRAND-TOTAL-LINE TO PRINT-AREA.
     WRITE PRINT-AREA AFTER ADVANCING 2 LINES.
```

Figure 3-13 The Procedure Division changes for the enhanced program

How to test a report-preparation program

When you develop a report-preparation program, testing is more difficult than when you test a simple interactive program for several reasons. First, you often have to prepare your own test files. Second, it's more difficult to review the data in input files and print files. Third, run-time errors are more likely. The topics that follow present some information that should help you do a better job of testing.

How to prepare a test plan and test data

When you develop a report-preparation program, it often makes sense to develop a plan for testing the program. This *test plan* normally divides the testing into three or more phases as shown in figure 3-14. In phase 1 of this plan, just the main logic of the program is tested with "clean" data. That means that the data doesn't test for exceptional conditions like zero values in the last year-to-date field. As a result, the program should run to completion and print the report without any serious problems.

In phase 2 of this test plan, the limits of the input data are tested, including the maximum values that the fields can store, zero values, and negative values. When that testing is complete, the program should work correctly for all combinations of input data. Then, in phase 3, page overflow is tested.

Note that you don't need many records for the early phases of testing. For instance, the first phase in the plan calls for just three records with year-to-date values that test to make sure the right records are printed (those with values greater than or equal to $10,000). Similarly, it takes just a few more records to test the limits of the input data in a program like this. By keeping the number of records low, you make it easier to tell whether the program has worked correctly after each test run. Then, in the last phase, you can add as many records as you need for testing volume conditions like page overflow.

As you create your test plan, you should decide where the test run data will come from. Will you create the test files yourself using a word processor, a text editor, or the utility programs that are available on your system? Does another program in the system create test data that you can use? Can you copy some production data for your test files? For the early phases of testing, it's often best to create your own data. That way, you can be sure that you've tested all the conditions that the program should provide for.

For the exercises and projects in this book, you can get the test data from the CD ROM so you won't have to create a test plan or test data. When you develop your own programs, though, you should take the time to do a thorough job of testing. The test plan is a critical tool for making sure you do that right.

A test plan for the enhanced report-preparation program

Phase	Description	Data
1	Test main logic with clean data	Three records: one with this YTD sales < $10,000; one with it equal to $10,000; and one with it > $10,000
2	Limit test the calculations	Phase 1 data plus records that test the maximum and minimum values of the input fields, including at least one record with last YTD sales equal to zero
3	Page overflow	Phase 2 data plus enough records to cause page overflow

Sources of test data

- Enter your own data into a sequential file by using a text editor like NotePad, a word processor, or a utility program that's available on your system.
- If test data is available for other programs that use the same file, you can get a copy of that data.
- If other programs that use the same file are already in production, you can get a copy of that "live" data.

Description

- When you're ready to test a program, it often makes sense to develop a *test plan* for it. The test plan should describe the phases of testing as well as the data that should be used in each of the phases.
- For a large program, it makes sense to do the testing in at least three phases. These phases should test (1) the main logic of the program, (2) the maximum and minimum values that the program provides for, and (3) volume or overflow conditions like page overflow.
- For the first phase of testing, you usually need just a few records. For each later phase, you can add the records you need.
- When you prepare your own data, you have complete control of the data so you can test every input possibility. When you use live data or test data that's prepared by others, you need to list it or display it on your screen so you know for sure what the input data is.

Figure 3-14 How to prepare a test plan and test data

How to review input files and print files when using Personal COBOL

When you use a test file that has been prepared by someone else or by another program, you need to review the data so you know what the test results should be. To do that on a mainframe or mid-range system, you can use a utility program to display or print the data. To do that on a PC, you can use your word processing program or a text editor like NotePad.

In figure 3-15, for example, you can see how the test file for the enhanced report-preparation program looks when it's opened in Word. As you can see, nothing separates one field from another or one record from another. Instead, the entire file is one string of characters that is rolled over into lines due to the margin settings. When you open this file with NotePad, it is even harder to interpret because the entire file is displayed on one line.

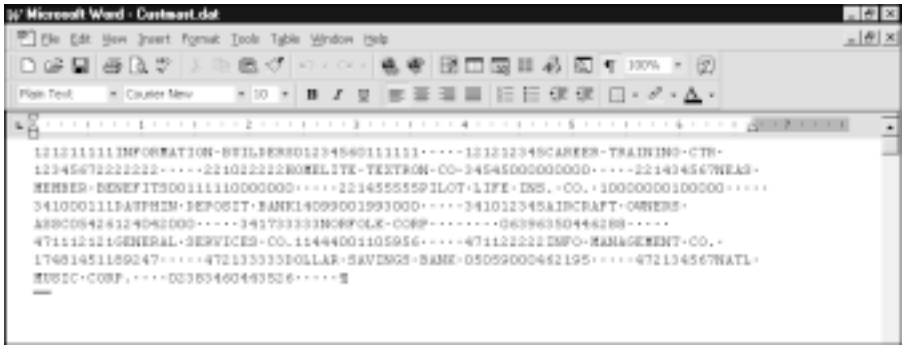
Nevertheless, you can review the disk data when you open it this way. Then, to make it more readable, you can move the cursor to the end of each record and press the Enter key. If you do that for all of the records in the file, the data will be displayed with one line for each record. At that point, you can print the data if you want to refer to it later. *But don't save the data in this form*, because your program won't be able to open the file if you do. Instead, you must close the file without saving the changes.

When your program writes a report to a print file on disk, you have to use a similar process to review the report. On a mainframe or mid-range system, utility programs are available for displaying or printing the report. But on a PC, you need to use your word processing program or a text editor. In this figure, for example, you can see the enhanced sales report after it has been opened by Word.

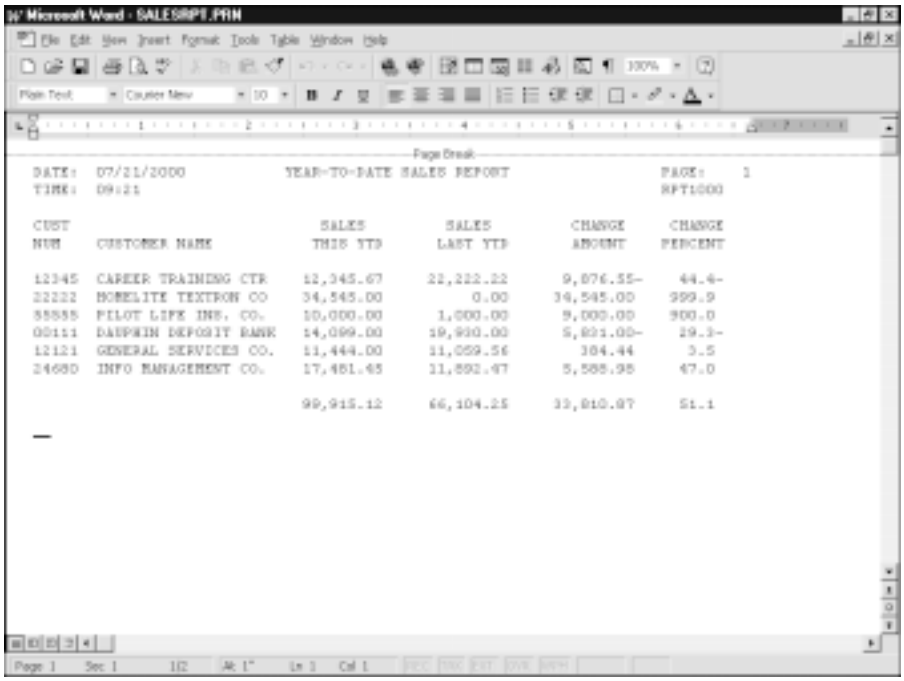
If the print file has long lines, you may need to format the report before it looks the way you want it to. For instance, you may want to change the layout from Portrait to Landscape or reduce the size of the font. Once you've got it the way you want it, you can review it on the screen or print it.

In this figure, you can see that the report starts with a page break, because the program skips to the next page before it prints the first line. To save paper, then, you can delete the page break before you print the report. For many of your unsuccessful test runs, though, you won't need to print the report at all.

The disk file when it's opened in Word



The print file when it's opened in Word



Description

- To review the data in an input file, you can open the file with your word processing program or NotePad. Note, however, that the data isn't neatly aligned so it is difficult to read.
- To review or print the data in a print file, you can open the file with your word processor or with NotePad. To make the data more readable, you may want to change the page layout to landscape, change the margins, or change the font or font size. Then, you can review the output without printing it, or you can print it.

Figure 3-15 How to review input files and print files when using Personal COBOL

Common run-time errors

When you develop a report-preparation program, you're likely to encounter some common run-time errors. Since you'll be better able to avoid these errors by knowing what they are and what causes them, figure 3-16 presents a brief overview.

The first group of errors are file handling errors. These occur when a serious error occurs while the program tries to open or close a file or read or write a record in a file. On a mainframe, this type of error is called an *operation exception*. If, for example, a program tries to open a file that it can't find on the disk, an operation exception occurs. This figure also lists three other common causes of operation exceptions, and there are others that you'll learn about in section 3.

The second group of errors are data errors, and you should already be familiar with them. If, for example, a Compute statement tries to operate upon a field that contains invalid data, a run-time error occurs. On a mainframe, this is known as a *data exception*.

Although there are many causes of data exceptions, this type of error commonly occurs when the COBOL description for a disk record doesn't match the data. If, for example, you define one of the input fields in your program as S9(4) and the field is actually S9(3), that can lead to a data exception later on. So if a data exception occurs during the first test run for a report-preparation program, be sure to check this possibility.

The other two data errors aren't specifically related to report-preparation programs, but they often occur when you're not familiar enough with the input data to provide for all possible data conditions in your program. That happens when you don't take the time to review the test data or live data that you have been given.

The first of these errors occurs when the result of an arithmetic operation is too large for the receiving field and On Size Error hasn't been coded. On a mainframe, this is called an *overflow exception.* The best prevention for this is to know the data and define the receiving fields so they are large enough to handle the results.

The second one occurs when an arithmetic operation tries to divide by zero. On a mainframe, this is called a *divide exception.* Here again, the best prevention is to know the data so you won't be surprised when one of the fields turns up with a value of zero.

Note that the last two conditions don't cause run-time errors with Personal COBOL. Instead, Personal COBOL makes adjustments so the statements can execute successfully. In this case, though, the results are incorrect so this is still a programming error.

File handling errors

- The file can't be opened.
- A Read or Write statement is executed before the file is opened.
- The program refers to a field in the record description area for a file after the At End clause has been executed or before the first Read statement has been executed.
- A Stop Run statement is executed before all opened files are closed.
- On a mainframe, errors like these are called *operation exceptions*.

Data errors

- A statement tries to perform an arithmetic operation or a numeric comparison on invalid numeric data. This type of error can occur when the data in an input record doesn't match the record description. This causes a run-time error on a mainframe and with Personal COBOL. On a mainframe, this type of run-time error is called a *data exception*.
- An arithmetic operation gives a result that is too large for the receiving field. On a mainframe, this causes a run-time error called an *overflow exception*. With Personal COBOL, the result is truncated instead of causing a run-time error.
- A divide operation tries to divide by zero. On a mainframe, this causes a run-time error called a *divide exception*. With Personal COBOL, the zero is treated as a one instead of causing a run-time error.

Figure 3-16 Common run-time errors when developing report-preparation programs

Perspective

The program in this chapter has many of the characteristics of a production program in business. In the real world, a report will have more columns and require more calculations. And many reports will require summary information, which will increase the logical complexity of the program. But the program in this chapter gives you a good idea of how a report-preparation program is developed in business.

Curiously, the enhanced version of the program requires only 11 different statements in the Procedure Division. So the difficulty in writing this kind of program isn't mastering the syntax of the COBOL statements. Instead, the difficulty is controlling the logic of the program.

If you had any trouble understanding the logic in this program, consider that the program is only 211 lines long and requires only seven procedures. Now imagine a program that gets data from two or more files, prints totals at two or more levels, and consists of 1500 lines or more, which is typical of a report-preparation program. To handle the logical complexity of a program of that size, you need to take a highly structured approach to its development. And that's what you'll learn how to do in the next chapter.

Summary

- A *sequential file* on disk consists of *records* that are read in sequence. Each record consists of *fields*.
- To plan a report, you create a *print chart* that shows the content and format of each line to be printed.
- A *switch* is a field that can have a value of either yes or no (or on or off). All business programs require one or more switches.
- An *input file* is a file that is read by a program, and an *output file* is a file that is written to. A *print file* is an output file that contains a report that can be printed.
- Each file that's used by a program must be identified with a Select statement in the File-Control paragraph of the Environment Division. In addition, the File Section of the Data Division must contain an FD statement and record description for each file.
- Before you can read from or write to a file, you must open it. Before a program ends, you must close all open files.
- A Read statement for a sequential disk file reads the next record in a file, starting with the first record. A Write statement for a print file writes one record for each line of the report.
- If you're using a compiler that supports the *intrinsic functions*, you can use the Current-Date function to get the current date and time from the system. Otherwise, you can use the Accept Date and Accept Time statements.

- A *test plan* lists the test phases and describes the test data that will be used in each phase.
- Four common types of run-time errors on a mainframe are *operation exceptions*, *data exceptions*, *overflow exceptions*, and *divide exceptions*.

Terms

sequential file
record
field
print chart
print file
switch
filename
system name
ddname
spooling
print area
input file
output file
intrinsic function
test plan
operation exception
data exception
overflow exception
divide exception

Objectives

- Given the specifications for a report-preparation program like the ones in this chapter, develop a program that prepares the report.
- Given the specifications for a report-preparation program, prepare a test plan for it.
- Describe the characteristics of a sequential file on disk.
- Describe the use of a print chart.
- Describe the use of a switch.
- List two ways that you can get the current date and time into a program.
- Describe each of these types of errors: operation exception, data exception, overflow exception, and divide exception.
- If you're using Micro Focus Personal COBOL, use your word processing program to open, display, and print an input file or a print file.

Exercise 3-1 Test the sales report program

This exercise will guide you through the process of testing the sales report program in this chapter when you're using Micro Focus Personal COBOL. It will also show you how to use your word processor to review the input data and the report that's prepared from it.

Compile and test the program

1. Start Personal COBOL, and open for edit the program named rpt1000.cbl that's in the c:\cobol folder. Then, compile the program (it should have a clean compile).
2. Close the document window, and open the program for execution. Then, run the program (it should end with a return code of zero).

Review the sales report and the test data

3. Start your word processor and open the print file named salesrpt.prn that's in the c:\cobol\data folder. If it looks like the one in figure 3-15, it means the program worked correctly. If you want to print the file, delete the page break at the start of the file, adjust the font size and page setup (if necessary), and print. Then, close the file.
4. If necessary, you can also use your word processor or NotePad to review the test data for the program. To do that, open the file named custmast.dat that's in the c:\cobol\data folder. If you're using Word, the data should look the way it does in figure 3-15. Then, you can move the cursor to the end of the first record and press the Enter key to display that record on one line by itself. If you repeat that process for the next record, you get a better view of what the input data looks like. When you're through experimenting, close the file…*but don't save it or your program won't be able to open it*.
5. To test whether page overflow works, change the LINES-ON-PAGE field in your program to a value of 10, recompile, and rerun. Then, open the print file in your word processor to see whether the report prints with 10 lines per page. If it does, close the print file, reset the LINES-ON-PAGE value to 55, and recompile.

Step through the program to see how it works, then close the program

6. If you have any doubts about how this program works, use the skills you learned in chapter 2 to step through the program from the start. Or use a breakpoint to step through any parts of the program that you don't understand. As you step, display the contents of the fields that are being operated upon.
7. When you're satisfied that you understand exactly how this program works, close the document window for the program.

Exercise 3-2 Compile and test a flawed program

To give you some practice with compiling and testing a report-preparation program, this exercise forces you to correct some compile-time errors and debug some run-time errors.

Correct the compile-time errors

1. Open the program named rpt1000x for edit, and compile it. Since we put a few errors in it, it doesn't compile cleanly.
2. Correct the errors and recompile until you get a clean compile. You should be able to do that without much trouble.

Test the program

3. Close the document window and open the same program for execution. Then, run it. The first time, you should see a message box for a run-time error like this:

   Then, when you click on the OK button, you return to the Animator with the Open statement highlighted. This indicates that there's either a problem with the Select statement or the file isn't where it's supposed to be.

4. Fix the problem, recompile, and rerun the program. This time, another run-time error message is displayed: "Illegal character in numeric field." Then, when you click on the OK button, a Move statement in procedure 220 is highlighted. The clue here is that the statement moves a field from the customer master record to a numeric edited field. This indicates that there's either a problem with the data or with the FD statement or record description that defines the data, but you already know that the data is okay.
5. To fix the problem, you need to make sure that the FD and record description match up with the input specifications. When you find the problem, correct it, recompile, and rerun the program. This time the program should run to a normal termination.
6. Use your word processor to check the report that was printed. At a glance, you can see that the report is double-spaced when it should be single-spaced. Don't stop there, though; do a thorough review of the report to make sure it works correctly. Check that the data values in each column are what you expected. Then, close the word processing document and return to the Animator. Fix the problems, recompile, and retest until you've got everything working right.

Close the program

7. Close the word processing document and the program. Now, you've got two programs that work the same: rpt1000 and rpt1000x.

Exercise 3-3 Enhance the sales report program

To show you how easy it is to enhance a well-written program, this exercise guides you through the process of enhancing the sales report program.

Enhance the original program

1. Open the program named rpt1000 for edit, change the Program-ID to RPT2000, and save the program as rpt2000. Then, close and re-open the program.
2. Make the enhancements to the program that are specified in figure 3-11. To learn the most from this exercise, try to make these enhancements without referring to the code in figures 3-12 and 3-13.
3. Compile and test the enhanced program until it works correctly.

Modify the enhanced program

4. Change the Program-ID to RPT2100, save the program as rpt2100, and close and re-open it.
5. Instead of printing a customer line on the report only when this year's YTD sales are greater than or equal to $10,000, modify this version of the report so it prints a customer line when the change amount is greater than or equal to $5,000. Then, compile and test the program until that works right.
6. If that was easy enough to do, modify the report again so it prints a customer line only when the change percent is positive and greater than or equal to 25.0%. Then, compile and test the program until it works right.

Close the program

7. Close the program, and exit from Micro Focus Personal COBOL.