The Use of Periods

All statements in the first three divisions end with a period. The PROCEDURE DIVISION entry and the module names end with a period, as does the last statement in each module. In our COBOL 85 programs, all other PROCEDURE DIVISION entries do not have periods. We will discuss punctuation in more detail later on.

Figure 1.9, then, represents an entire sample COBOL 85 program. It will run on any computer, although minor changes may be necessary in the SELECT statements because some compilers have specific requirements for the ASSIGN clause. You may want to key in and run this program on your system just to familiarize yourself with COBOL. Figure 1.11 is the computer-produced source listing of the same program. This program can

Figure 1.11 Computer listing of sample program.

```
01  IDENTIFICATION DIVISION.
02  PROGRAM-ID. SAMPLE.
03  ENVIRONMENT DIVISION.
04  INPUT-OUTPUT SECTION.
05  FILE-CONTROL. SELECT EMPLOYEE-DATA    ASSIGN TO EMP-DAT.
06                 SELECT PAYROLL-LISTING  ASSIGN TO PRINTER.
07  DATA DIVISION.
08  FILE SECTION.
09  FD  EMPLOYEE-DATA      LABEL RECORDS ARE STANDARD.
10  01  EMPLOYEE-RECORD.
11      05  EMPLOYEE-NAME-IN    PICTURE X(20).
12      05  HOURS-WORKED-IN     PICTURE 9(2).
13      05  HOURLY-RATE-IN      PICTURE 9V99.
14  FD  PAYROLL-LISTING    LABEL RECORDS ARE OMITTED.
15  01  PRINT-REC.
16      05                       PICTURE X(20).
17      05  NAME-OUT             PICTURE X(20).
18      05                       PICTURE X(10).
19      05  HOURS-OUT            PICTURE 9(2).
20      05                       PICTURE X(8).
21      05  RATE-OUT             PICTURE 9.99.
22      05                       PICTURE X(6).
23      05  WEEKLY-WAGES-OUT    PICTURE 999.99.
24  WORKING-STORAGE SECTION.
25  01  ARE-THERE-MORE-RECORDS  PICTURE XXX VALUE 'YES'.
26  PROCEDURE DIVISION.
27  100-MAIN-MODULE.
28      OPEN INPUT EMPLOYEE-DATA
29           OUTPUT PAYROLL-LISTING
30      PERFORM UNTIL ARE-THERE-MORE-RECORDS = 'NO '
31          READ EMPLOYEE-DATA
32               AT END
33                    MOVE 'NO ' TO ARE-THERE-MORE-RECORDS
34               NOT AT END
35                    PERFORM 200-WAGE-ROUTINE
36          END-READ
37      END-PERFORM
38      CLOSE EMPLOYEE-DATA
39            PAYROLL-LISTING
40      STOP RUN.
41  200-WAGE-ROUTINE.
42      MOVE SPACES TO PRINT-REC
43      MOVE EMPLOYEE-NAME-IN TO NAME-OUT
44      MOVE HOURS-WORKED-IN TO HOURS-OUT
45      MOVE HOURLY-RATE-IN TO RATE-OUT
46      MULTIPLY HOURS-WORKED-IN BY HOURLY-RATE-IN
47               GIVING WEEKLY-WAGES-OUT
48      WRITE PRINT-REC.
```

Figure 4.4 Solution to the Practice Program.

COBOL 85

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. SAMPLE.
*****************************************
*  sample  - updates a file with employee  *
*             names and salaries           *
*****************************************
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. VAX-6410.
 OBJECT-COMPUTER. VAX-6410.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT IN-EMPLOYEE-FILE ASSIGN TO DATA4E.
     SELECT OUT-SALARY-FILE  ASSIGN TO DATA4S.
*
 DATA DIVISION.
 FILE SECTION.
 FD  IN-EMPLOYEE-FILE
     LABEL RECORDS ARE STANDARD.
 01  IN-EMPLOYEE-REC.
     05  IN-EMPLOYEE-NAME          PIC X(20).
     05  IN-SALARY                 PIC X(5).
     05  IN-NO-OF-DEPENDENTS       PIC X(1).
     05  IN-FICA                   PIC X(5).
     05  IN-STATE-TAX              PIC X(6).
     05  IN-FED-TAX                PIC X(6).
     05                            PIC X(37).
 FD  OUT-SALARY-FILE
     LABEL RECORDS ARE STANDARD.
 01  OUT-SALARY-REC.
     05  OUT-EMPLOYEE-NAME         PIC X(20).
     05  OUT-SALARY                PIC X(5).
 WORKING-STORAGE SECTION.
 01  WS-WORK-AREAS.
     05  ARE-THERE-MORE-RECORDS    PIC X(3)  VALUE 'YES'.
*
 PROCEDURE DIVISION.
*************************************************************
*  100-main-module - controls opening and closing files  *
*                    and direction of program logic;     *
*                    returns control to operating system *
*************************************************************
 100-MAIN-MODULE.
     OPEN INPUT  IN-EMPLOYEE-FILE
          OUTPUT OUT-SALARY-FILE
     MOVE SPACES TO OUT-SALARY-REC
     PERFORM UNTIL ARE-THERE-MORE-RECORDS = 'NO '
       READ IN-EMPLOYEE-FILE
         AT END
           MOVE 'NO ' TO ARE-THERE-MORE-RECORDS
         NOT AT END
           PERFORM 200-PROCESS-RTN
       END-READ
     END-PERFORM
     CLOSE IN-EMPLOYEE-FILE
           OUT-SALARY-FILE
     STOP RUN.
```

The word FILLER is optional with COBOL 85 but required with COBOL 74

COBOL 74 Substitutions

```
READ IN-EMPLOYEE-FILE
  AT END MOVE 'NO ' TO
     ARE-THERE-MORE-RECORDS.
PERFORM 200-PROCESS-RTN
  UNTIL ARE-THERE-MORE-RECORDS
     = 'NO '.
```

Figure 4.4 Solution to the Practice Program (continued).

```
*************************************************************
*  200-process-rtn - performed from 100-main-module        *
*                    moves employee information to output *
*                    areas, then writes the record        *
*************************************************************
 200-PROCESS-RTN.
     MOVE IN-EMPLOYEE-NAME TO OUT-EMPLOYEE-NAME
     MOVE IN-SALARY TO OUT-SALARY
     WRITE OUT-SALARY-REC.
```

COBOL 74 Substitutions

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

Sample Input Data

```
NANCY STERN          |09898|2|12300|098900|029900
ROBERT STERN         |10923|2|21000|098890|092830
CHRISTOPHER HAMMEL   |08437|1|38370|067373|073700
GEORGE WASHINGTON    |03383|2|39390|003920|039200
TOM JEFFERSON        |08383|8|32200|093830|039200
LORI STERN           |29339|1|63600|129290|029290
MELANIE STERN        |02384|1|02938|338382|023838
TEDDY SMITH          |10293|9|02239|328339|382839
JOHN DOE             |00338|2|29387|493038|330393
BILL FIXER           |08383|8|20028|303939|029383
```

NAME — SALARY — FICA — STATE TAX — DEPENDENTS

Sample Output

```
NANCY STERN          |09898
ROBERT STERN         |10923
CHRISTOPHER HAMMEL   |08437
GEORGE WASHINGTON    |03383
TOM JEFFERSON        |08383
LORI STERN           |29339
MELANIE STERN        |02384
TEDDY SMITH          |10293
JOHN DOE             |00338
BILL FIXER           |08383
```

NAME — SALARY — FEDERAL TAX

REVIEW QUESTIONS

I. True-False Questions

_____ 1. Files must be opened before they can be read.

_____ 2. A simple `PERFORM`, such as `PERFORM 300-CHECK-RTN`, will execute the named paragraph once and return control to the statement following the `PERFORM`.

_____ 3. COBOL paragraph-names end with a period.

_____ 4. Each module in a COBOL program is identified by a paragraph-name.

_____ 5. A `PERFORM UNTIL ...` will be executed repeatedly until the condition specified is met.

_____ 6. Consider the following statement: `PERFORM ... UNTIL ARE-THERE-MORE-RECORDS = 'NO '`. The data-name `ARE-THERE-MORE-RECORDS` must be defined in the `FILE SECTION`.

_____ 7. Suppose `EOF` were initialized at 1. It would be correct to use the following as an end-of-file test: `READ FILE-IN AT END MOVE 0 TO EOF`.

_____ 8. The `READ ... AT END ... NOT AT END ...` should end with an `END-READ` scope terminator.

_____ 9. The last instruction to be executed in a program should be a `STOP RUN`.

_____ 10. A paragraph-name may consist of all numbers, but it is best to use meaningful words to help identify the purpose of the paragraph.

II. General Questions

1. Indicate the `DIVISION` in which each of the following is coded and state its purpose.
   (a) `DATE-COMPILED`
   (b) `WORKING-STORAGE SECTION`
   (c) paragraph-name
   (d) `CONFIGURATION SECTION`
   (e) `FD`
   (f) level numbers
   (g) `LABEL RECORDS`
   (h) `FILE SECTION`
   (i) `SELECT`
   (j) `AUTHOR`
   (k) `STOP RUN`

The END-READ and END-IF are in color because they are permitted with COBOL 85 only. NOT AT END is a READ clause that can be omitted, as in the above.

As you proceed through this text, you will see how indentation is used to clarify the logic. You should use this technique in your programs as well. Note, however, that indentation does not affect the program logic at all. It is simply a tool that helps people *read* the program.

DEBUGGING TIP FOR COBOL 85 USERS

COBOL 85 programmers should always use scope terminators with the READ and IF statements, as well as others we will discuss. When scope terminators are coded, periods are not used to end statements except for the last statement in a paragraph. Scope terminators ensure that all clauses within a statement will be associated with the appropriate instruction, thereby minimizing logic errors.

AN INTRODUCTION TO INTERACTIVE PROCESSING

Interactive Processing. COBOL was originally developed to process files of data and is still widely used for that purpose. Many organizations, however, are using COBOL for interactive processing—where the user enters data using a keyboard of a PC or terminal, and output is displayed on the monitor at the user's desk. We use the ACCEPT verb for entering input from a keyboard and the DISPLAY verb for displaying output on a screen.

The instruction ACCEPT *identifier* enables the user to enter input data directly from a keyboard rather than from a disk file. The identifier is likely to be a WORKING-STORAGE entry. When input is entered using the ACCEPT verb, there is no need to establish a file.

To enter as input a DISCOUNT-AMT, for example, you can code:

```
ACCEPT DISCOUNT-AMT
```

The format for the input is determined by the PIC clause for DISCOUNT-AMT.

Example 1

```
WORKING-STORAGE SECTION.
01  DISCOUNT-AMT          PIC 9(3).
```

Using a keyboard, the user would enter three integers into DISCOUNT-AMT.

Example 2

```
WORKING-STORAGE SECTION.
01  DISCOUNT-AMT          PIC 99V99.
```

Here, the user would enter four numbers that will be interpreted by the computer as a dollars-and-cents amount. That is, data entered as 1234 will be stored as 12^34.

To create an accounts receivable file interactively from data entered at a keyboard, we code:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  CREATE.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT ACCOUNTS-RECEIVABLE-FILE
        ASSIGN TO DISK1.
DATA DIVISION.
FILE SECTION.
FD  ACCOUNTS-RECEIVABLE-FILE
    LABEL RECORDS ARE STANDARD.
01  ACCOUNTS-RECEIVABLE-REC.
    05  AR-NAME           PIC X(20).
    05  AR-ADDRESS        PIC X(20).
    05  AR-BAL-DUE        PIC X(6).
```

```
PROCEDURE DIVISION.
100-MAIN.
    OPEN OUTPUT ACCOUNTS-RECEIVABLE-FILE
    PERFORM 200-PROCESS-RTN
        UNTIL AR-NAME = 'END'
    CLOSE ACCOUNTS-RECEIVABLE-FILE
    STOP RUN.
*****************************************************************
*  when user enters 'end' for ar-name the program terminates  *
*****************************************************************
200-PROCESS-RTN.
    ACCEPT AR-NAME
    IF  AR-NAME NOT = 'END'
        ACCEPT AR-ADDRESS
        ACCEPT AR-BAL-DUE
        WRITE-ACCOUNTS-RECEIVABLE-REC
    END-IF.
```

Only the output file, ACCOUNTS-RECEIVABLE-FILE, is defined with a SELECT statement. When the user enters a name of 'END', the program terminates.

The DISPLAY, which displays output on a screen, has the following basic format:

$$\underline{\texttt{DISPLAY}} \left\{ \begin{matrix} \text{identifier-1} \\ \text{literal-1} \end{matrix} \right\} \ldots$$

The format indicates that you can display fields or literals or some combination of the two.

Examples

1. To prompt for input at a keyboard:

```
DISPLAY 'ENTER NAME'
ACCEPT NAME
```

2. To display a message:

```
READ IN-FILE
    AT END DISPLAY 'END OF FILE'
END-READ
```

3. To display a result:

```
DISPLAY RECORD-COUNTER
```

4. To display a literal along with a result:

```
DISPLAY 'NO. OF RECORDS IS ', RECORD-COUNTER
```

Commas may be used to separate displayed data for ease of reading, but they are not required. To ensure that the message and result do not collide (e.g., 'NO. OF RECORDS IS100'), always leave a space at the end of the literal.

We will discuss ACCEPTs and DISPLAYs in more detail throughout the text. We introduce these topics here for PC users who want to focus on interactive processing early on.

Let us modify our preceding program to prompt for input and to prompt the user to specify when there is no more data. We also use a MORE-RECS indicator here to determine when there is no more data to be entered.

```
100-MAIN.
    OPEN OUTPUT ACCOUNTS-RECEIVABLE-FILE
    MOVE 'Y' TO MORE-RECS
    PERFORM 200-PROCESS-RTN
        UNTIL MORE-RECS = 'N'
    CLOSE ACCOUNTS-RECEIVABLE-FILE
```

The next version of COBOL will be produced during this decade and is currently referred to as COBOL 9x. At the end of each chapter we include changes that are likely to be included in that new standard.

WEB SITE

The Internet site www.ansi.org includes COBOL 9x updates.

THE FUTURE OF COBOL

The future of COBOL has been the subject of much discussion in recent years. It is, after all, a language that has been in existence for over 40 years while newer languages now have many more features that make them more suitable for PC-based applications.

But COBOL is likely to remain an important language in the years ahead. Consider the following facts:

1. According to the Datapro Information Services Group, an estimated 150 billion lines of COBOL source code are currently in use, with programmers adding about five billion lines each year.
2. According to the Datapro Information Services Group, 42.7 percent of all applications development programmers in medium to large U.S. companies use COBOL.
3. International Data Corporation reports that revenues for COBOL desktop development are expected to be $176.4 million by 1998; these revenues have increased from $86.3 million in 1993 at an average growth rate of 15.4 percent a year.

These facts suggest that COBOL will remain an important language in the years ahead for two reasons: (1) older, mainframe-based ''legacy'' systems will need to be maintained by maintenance programmers who know COBOL and (2) COBOL is still being used by many organizations for new application development.

SELF-TEST Self-test questions are provided throughout the text, with solutions that follow, to help you assess your understanding of the material presented.

1. A program must be in _____ language to be executed or run by a computer.
2. Programs are typically written in a _____ language rather than in machine language because _____ .
3. Programs written in a language other than machine language must be _____ before execution can occur.
4. The process of converting a source program into machine language is called _____ .
5. The program written in a programming language such as COBOL is called the _____ program.
6. The object program is the _____ .
7. A _____ converts a source program into a(n) _____ program.
8. The errors that are detected during compilation denote _____ ; they are usually referred to as _____ errors.
9. Before executing a program with test data, the logic of the program can be checked manually using a technique called a _____ .
10. COBOL is an abbreviation for _____ .
11. COBOL is a common language in the sense that _____ .
12. (T or F) COBOL is ideally suited for scientific as well as business problems.

Solutions

1. machine
2. symbolic; machine languages are very complex
3. translated or compiled
4. compilation or translation

where identifier should have a `PIC 9(5)`. The day will be stored as yyddd, where ddd is a day number from 001–365 (001–366 in leap years). This value is commonly referred to as a **Julian Date**. 98002, then would be equivalent to a date of January 2, 1998 (the second day of the year).

Julian date is very useful for calculations, e.g., for determining accounts that are 30, 60, or 90 days past due. Subtracting 90 from a Julian date of purchase of 98190, for example, tells us that if a payment was made after 98100, then the account is *not* past due. The next chapter includes intrinsic date functions that enable the programmer to change the format for different types of dates.

WEB SITE

Until recently, COBOL programs have typically stored dates in mmddyy format, with yy being a two-digit year that assumed a twentieth-century date. That is, 010298 was assumed to be January 2, 1998. With the new century on the horizon, processing dates will become a big problem. Depending on the application, a two-digit date of 01 could mean 1901 (e.g., a person's year of birth) or 2001 (e.g., a year in a transaction).

This millenium change is likely to require extensive modifications to the billions of lines of existing COBOL code.

Our Web site (http://www.wiley.com/cobol/) as well as the Web sites http://www.flexus.com/cobol.html and http://www.year2000.com contain some interesting information about this and other problems. These Web sites also point to several shareware programs that can be used to solve the two-digit date problem.

Printing Quotation Marks

As noted, nonnumeric literals are enclosed in quotation marks (''), or apostrophes ('), depending on the computer system. Thus, the quotation mark (or apostrophe) itself cannot actually be part of a nonnumeric literal.

To print a quotation mark, then, we would use the COBOL figurative constant `QUOTE`. Suppose we wish to print a heading such as:

```
ITEM DESCRIPTION: 'SPECIAL'
```

This heading, which is to include quotation marks, is defined as:

The word `FILLER` is optional as a data-name with COBOL 85 but is required with COBOL 74

```
01  HEADING-1.
    05            PIC X(19)  VALUE SPACES.
    05            PIC X(18)  VALUE 'ITEM DESCRIPTION: '.
    05            PIC X      VALUE QUOTE.
    05            PIC X(7)   VALUE 'SPECIAL'.
    05            PIC X      VALUE QUOTE.
    05            PIC X(86)  VALUE SPACES.
```

The word `QUOTE` can be used to print a quotation mark as shown here; it cannot, however, be used to delimit a literal. Thus, `MOVE QUOTE SPECIAL QUOTE...` is not permissible.

CODING GUIDELINES FOR DESIGNING REPORTS

1. Include a heading that identifies the report.
2. Include the date and the page number in the report heading or on a separate page heading.
3. Include column headings for identifying fields to be printed.

Figure 9.5 (continued)

```
                    PRINCIPAL TABLE

ACCT NO   59964
DEPOSITOR NAME DON WILLIS
PRINCIPAL $   500
RATE .14
NO OF YEARS  7

        YEAR        NEW BALANCE        ACCRUED INTEREST
          1         $    570.00        $    70.00
          2         $    649.80        $   149.80
          3         $    740.77        $   240.77
          4         $    844.48        $   344.48
          5         $    962.70        $   462.70
          6         $  1,097.48        $   597.48
          7         $  1,251.13        $   751.13
```

REVIEW QUESTIONS

I. True-False Questions

_____ 1. A PERFORM paragraph-name statement permanently transfers control to some other section of a program.

_____ 2. In-line PERFORMs are permitted with COBOL 85 as long as they end with END-PERFORM.

_____ 3. GO TO statements are generally avoided in structured programs.

_____ 4. EXIT is a COBOL reserved word that performs no operation.

_____ 5. Using a PERFORM UNTIL option, the condition is tested before the paragraph is executed even once.

_____ 6. PERFORM 400-LOOP-RTN N TIMES is only valid if N is defined as numeric.

_____ 7. Using PERFORM 400-LOOP-RTN N TIMES, N should not be altered within 400-LOOP-RTN.

_____ 8. It is valid to say PERFORM 400-LOOP-RTN N TIMES, where N = 0.

_____ 9. The PERFORM and GO TO statements will cause identical branching.

_____ 10. If several paragraphs are to be executed by a PERFORM statement, we may use the THRU option.

II. General Questions

1. Using a PERFORM statement with a TIMES option, write a routine to find N factorial, where N is the data item. Recall that N factorial = N × (N − 1) × (N − 2) × . . . × 1; e.g., 5 factorial = 5 × 4 × 3 × 2 × 1 = 120.
2. Rewrite the following routine using (a) a PERFORM statement with a TIMES option and (b) a PERFORM with a VARYING option:

```
    MOVE ZEROS TO COUNTER
    PERFORM 400-LOOP-RTN
        UNTIL COUNTER = 20.
    .
    .
400-LOOP-RTN.
    READ SALES-FILE
        AT END MOVE 'NO ' TO ARE-THERE-MORE-RECORDS
    END-READ
    ADD QTY OF SALES-REC TO TOTAL
    ADD 1 TO COUNTER.
```

3. Write two routines, one with a PERFORM ... TIMES and one with a PERFORM UNTIL ... to sum all odd integers from 1 to 1001, exclusive of 1001.
4. Write a routine to sum all odd integers between 1 and 99 inclusive.
5. Write a routine to calculate the number of minutes in a 24-hour day using the PERFORM ... TIMES option.

DEBUGGING EXERCISES

1. Consider the following coding:

```
        PERFORM 400-ADD-RTN
            VARYING X FROM 1 BY 1 UNTIL X > 50.
        .
        .
        .
    400-ADD-RTN.
        READ AMT-FILE
            AT END MOVE 'NO ' TO ARE-THERE-MORE-RECORDS
        END-READ
        ADD AMT TO TOTAL
        ADD 1 TO X.
```

   (a) How many times is AMT added to TOTAL?
   (b) Is the logic in the program excerpt correct? Explain your answer.
   (c) What will happen if there are only 14 input records? Explain your answer.
   (d) Correct the coding so that it adds amounts from 50 input records and prints an error message if there are fewer than 50 records.

2. Consider the following program excerpt:

```
        .
        .
        .
        PERFORM 200-CALC-RTN
            UNTIL NO-MORE-RECORDS
        .
        .
        .
    200-CALC-RTN.
        READ SALES-FILE
            AT END MOVE 'NO ' TO ARE-THERE-MORE-RECORDS
        END-READ
        MOVE 0 TO COUNTER
        PERFORM 300-LOOP-RTN
            UNTIL COUNTER = 5
        MOVE TOTAL TO TOTAL-OUT
        MOVE TOTAL-REC TO PRINT-REC
        WRITE PRINT-REC.
    300-LOOP-RTN.
        ADD AMT1 AMT2 GIVING AMT3
        MULTIPLY 1.08 BY AMT3 GIVING GROSS
        SUBTRACT DISCOUNT FROM GROSS
            GIVING TOTAL.
```

   (a) This coding will result in a program interrupt. Indicate why. What changes should be made to correct the coding?
   (b) Suppose COUNTER is initialized in WORKING-STORAGE with a VALUE of 0. Would it be correct to eliminate the MOVE 0 TO COUNTER instruction from 200-CALC-RTN? Explain your answer.
   (c) Code the three arithmetic statements in 300-LOOP-RTN with a single COMPUTE statement.

PROGRAMMING ASSIGNMENTS

1. **Interactive Processing.** Write a program to display a temperature conversion table on a screen. Compute and print the Fahrenheit equivalents of all Celsius temperatures at 10-degree intervals from 0 to 150 degrees. The conversion formula is Celsius = 5/9 (Fahrenheit − 32).
2. Write a program to produce a bonus report. See the problem definition in Figure 9.6.

   Notes:

   a. The payroll records have been sorted into ascending sequence by office number within territory number. There are three territories, two offices within each territory, and 10 employees within each office. We have, therefore, 60 records (3 × 2 × 10). Thus, all employees within office 01 within territory 01 will appear before employee records for office 02 within territory 01, and so on.
   b. Only employees who were hired before 1994 are entitled to a 10% bonus.

CHAPTER 18

An Introduction to Object-Oriented Programming

CONTENTS

THE ADVANTAGES OF OBJECT-ORIENTED PROGRAMMING

We have emphasized throughout this text the importance of writing programs that are easy to read, debug, maintain, and modify. As the need for computer applications increases, the need for well-designed, well-written, and well-documented programs increases as well. Some of the concepts we have focused on that improve the quality of programs include the use of:

1. Structured techniques
2. Top-down design
3. Comments to help document the program
4. Scope terminators with COBOL 85 to delimit instructions
5. Indentation and other COBOL coding techniques for improving readability
6. `COPY` and `CALL` statements to minimize duplication of effort
7. Data validation techniques to minimize input errors

The above techniques are used with most third-generation languages for improving the overall design of programs. But because there is an increasing need for quality

Summary of Terms

Object—an integrated unit of data and procedures or methods that operate on that data.
Encapsulation—hiding the data and procedures in an object behind a user interface so that the user program need not be concerned about the details and so that security can be maintained.
Class—a set of objects that share attributes (data and procedures).
Inheritance—the ability of objects to share attributes held by other objects in the same class.
Polymorphism—the ability for objects in a class to respond to methods differently from other objects in the class.
Persistence—the ability of a user program to operate on class objects and retain the changes made.
Instantiation—establishing a new instance of an object in a class.

TUTORIAL FOR OBJECT-ORIENTED COBOL

The single most important change in the COBOL 9x standard will be its expected inclusion of an official version of object-oriented COBOL. In anticipation of the release of COBOL 9x, several software developers have already incorporated object-oriented options into their compilers:

Software Developer	Product
Micro Focus	Object COBOL
IBM	Visual Age for COBOL
Computer Associates	Visual Realia COBOL
Netron, Inc.	Netron/Fusion
TechBridge Technology	TechBridge Builder

Micro Focus has been a leader not only in including object COBOL extensions, but also in including graphical user interfaces and client-server features in their products. The following is a brief tutorial, consisting of three lessons, that has been reprinted with permission from Micro Focus's Personal Object COBOL manual.[1] Personal Object COBOL is a version of COBOL available with this text.

LESSON 1: "HELLO WORLD" AND PROCEDURAL COBOL

With Micro Focus's Object COBOL option, you use the key programming tool, the Class Browser, along with some of the fundamental concepts of object-oriented COBOL programming. In going through the tutorial, you'll learn:

- The basic parts of a class.
- The key elements needed to use an object.
- Passing a parameter to an object method.
- Passing a parameter to an object method and getting back a return value.

Along the way, you'll get a feel for Micro Focus's Class Browser and learn how to load, compile, and run code with it. By the time you finish (and it doesn't take long), you should feel familiar with the Browser and its basic controls.

We start with a programming classic: *Hello world*.

[1] *Getting Started*, Micro Focus Personal COBOL for Windows 3.1 with Object Orientation, 1995.

Example 1 Let us consider first a function to calculate the square root of a number. To calculate the square root of X and place the result in Y, we may code:

```
COMPUTE Y = FUNCTION SQRT (X)
```

The word FUNCTION is required, followed by the specific intrinsic function, in this case the square root or SQRT function. The field or argument to be operated on is enclosed in parentheses. Arguments can be identifiers or literals. We use the COMPUTE instruction to return to the field called Y the value of the square root of X.

Example 2 Suppose we want to convert an alphanumeric field called NAME-IN to all uppercase letters. There is an intrinsic function for this as well:

```
MOVE FUNCTION UPPER-CASE (NAME-IN) TO NAME-OUT
```

UPPER-CASE is an intrinsic function that is used to move to NAME-OUT the uppercase equivalent of whatever value is in NAME-IN. Characters that are not letters will be moved as is, and lowercase letters will be converted to uppercase. John O'Connor 3rd, for example, will be returned to NAME-OUT as JOHN O'CONNOR 3RD.

We have divided the 42 intrinsic functions that have already been approved into several categories: calendar functions, numerical analysis and statistical functions, trigonometric functions, financial functions, and character and string functions.

CALENDAR FUNCTIONS

1. CURRENT-DATE—If you code MOVE FUNCTION CURRENT-DATE TO CURRENT-DATE-AND-TIME, the latter should contain 21 characters and have the following components:

```
01  CURRENT-DATE-AND-TIME.
    03  THIS-DATE.
        05  CURRENT-YEAR    PIC 9999.
        05  CURRENT-MONTH   PIC 99.
        05  CURRENT-DAY     PIC 99.
    03  THIS-TIME.
        05  HRS             PIC 99.
        05  MINUTES         PIC 99.
        05  SECONDS         PIC 99.
        05  HUNDREDTHS      PIC 99.
        05  OFFSET-VALUE    PIC X.
        05  OFFSET-HOUR     PIC 99.
        05  OFFSET-MINUTE   PIC 99.
```

2. WHEN-COMPILED—This calendar function returns the same 21 characters of date and time information indicating when the program was compiled.
3. INTEGER-OF-DATE—This is another calendar function that returns the number of days since January 1, 1601. The starting date of January 1, 1601 was chosen somewhat arbitrarily as a base from which integer values for more recent dates could be calculated. This numeric value is often used to determine the days that have elapsed from one date to another:

```
COMPUTE NUMBER-OF-DAYS-SINCE-LAST-PURCHASE =
    FUNCTION INTEGER-OF-DATE (THIS-DATE)
    - FUNCTION INTEGER-OF-DATE (PURCHASE-DATE)
```

or

```
SUBTRACT FUNCTION INTEGER-OF-DATE (PURCHASE-DATE)
    FROM FUNCTION INTEGER-OF-DATE (THIS-DATE)
        GIVING NUMBER-OF-DAYS-SINCE-LAST-PURCHASE
```

The INTEGER-OF-DATE function assumes that the argument or date field is in yyyymmdd format (e.g., January 1, 1998 would be 19980101). Note that any

Example

```
DISPLAY 'ENTER CODE'
    LINE 2 { POSITION 10 }
           { COLUMN 10   }
```

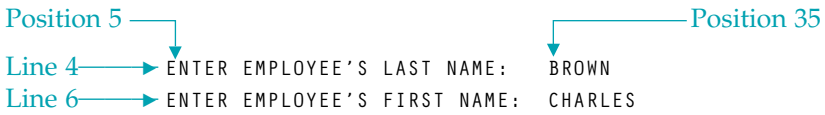
Use COLUMN for Micro Focus COBOL, VAX, and IBM AS/400 compilers. Use POSITION for the Ryan McFarland compiler.

If input is also entered interactively with the ACCEPT verb, then the input, as well as the output, can be positioned on the screen.

Example

```
200-REQUEST-FOR-EMPLOYEE-INFORMATION.
    DISPLAY 'ENTER EMPLOYEE'S LAST NAME:'
        LINE 4 COLUMN 5
    ACCEPT E-LAST-NAME
        LINE 4 COLUMN 35
    DISPLAY 'ENTER EMPLOYEE'S FIRST NAME:'
        LINE 6 COLUMN 5
    ACCEPT E-FIRST-NAME
        LINE 6 COLUMN 35.
```

The screen display would appear as follows:



Using LINE and COLUMN (or POSITION), you can design your interactive screen displays any way you like. Without the LINE clause, the default would be the next sequential line. Without the COLUMN or POSITION clause, the default would be column 1 of the next line.

THE SCREEN SECTION FOR DEFINING SCREEN LAYOUTS USING PCS

Interactive Processing. Because the need for interactive processing is greatest with microcomputers, PC versions of COBOL tend to have the most advanced features. PC versions of COBOL usually have a SCREEN SECTION as the last one in the DATA DIVISION for defining screen layouts.[1] They also have enhancements to the ACCEPT and DISPLAY verbs that are very helpful for solving some of the problems we have described.

We will discuss some of the elements of the SCREEN SECTION that are common to many compilers. Keep in mind that because this section is not part of the standard, the available options will differ, depending on both the COBOL compiler and the hardware you are using. Most of the options presented here are available in one form or another. We focus on RM/COBOL-85 and Micro Focus COBOL for illustrating the SCREEN SECTION, because (1) they are so popular and (2) student editions of these compilers are available with this text.

The SCREEN SECTION of the DATA DIVISION describes the format of a screen so that (1) the DISPLAY statement can more dynamically display literals and data and (2) the ACCEPT statement can enter data in a more user-friendly way.

The following are some of the features that can be provided by a SCREEN SECTION:

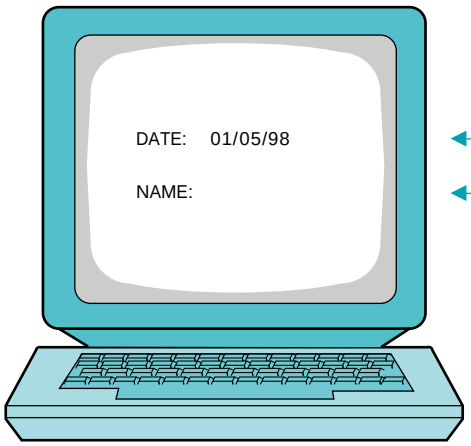
Highlighted display—in boldface (dual intensity) or blinking.
Reverse video—the background and foreground colors are reversed.
Underlining.
Color display—compiler dependent.

[1]See Stern, Stern, and Janossy, *Getting Started with RM/COBOL-85* and *Getting Started with Micro Focus Personal COBOL*, which are available with this text.

Sounding a bell to signal that input is to be entered.
Clearing the screen after each interaction.
Designating the specific positions in which data is to be entered on a screen.

As noted, many microcomputer versions of COBOL, like Micro Focus COBOL and RM/COBOL-85, include a SCREEN SECTION of the DATA DIVISION for screen design. This section follows the FILE and WORKING-STORAGE SECTIONs. The DISPLAY statement in the PROCEDURE DIVISION causes the screen to be displayed according to the specifications in the SCREEN SECTION; similarly, the ACCEPT statement enables users to enter input according to the same SCREEN SECTION specifications.

The SCREEN SECTION, like the other sections of the DATA DIVISION, consists of group items that themselves are subdivided into elementary items. Consider the following, in which the date is generated by the computer and a name is entered by the user:



The SCREEN SECTION elements that will enable the literals to print and the user to enter the date on the same line is as follows:

```
SCREEN SECTION.
01  SCREEN-1.
    05  BLANK SCREEN.
    05  LINE 1 COLUMN 1    VALUE 'DATE:'.
    05  COLUMN 8   PIC X(8)  FROM STORED-DATE.
    05  LINE 3 COLUMN 1    VALUE 'NAME:'.
    05  COLUMN 8   PIC X(20) TO NAME-IN.
```

STORED-DATE is a field already stored in the DATA DIVISION.

The name is typed in columns 8-27. It will be transmitted to a NAME-IN field in the DATA DIVISION.

To obtain the computer-generated date and then to have the computer prompt for name, we code:

```
DISPLAY SCREEN-1
ACCEPT  SCREEN-1
```

Let us consider the SCREEN-1 entries in the SCREEN SECTION.

1. The first entry within SCREEN-1 clears the screen.
2. The following two entries refer to LINE 1:

```
05  LINE 1 COLUMN 1    VALUE 'DATE:'.
05  COLUMN 8   PIC X(8)  FROM STORED-DATE.
```

Since the line beginning with COLUMN 8 does not have a LINE number, it is assumed by the computer that column 8 refers to LINE 1 specified in the previous entry.

STORED-DATE is a DATA DIVISION entry that must already have a value before

Print the average GPA for all students at the college.

8. **Interactive Processing.** Write an interactive program for a bank. Users will enter on a PC or terminal a customer name, principal amount to be invested (P_0), rate of interest (R), and number of years that the principal will be invested (N). Display the user's name and the value of the investment after N years. The formula is:

   Principal after N years $= P_0 (1 + R)^N$

   Do not forget to include prompts before accepting input.

9. **Maintenance Program.** Modify the Practice Program in this chapter to print the overall class average at the end of the report.

Figure 8.5 Input record layout for Programming Assignment 8.

MEDICAL-FILE Record Layout			
Field	**Size**	**Type**	**No. of Decimal Positions (if Numeric)**
PATIENT-NAME	20	Alphanumeric	
LUNG-INFECTION	1	Numeric: 1 = present 0 = absent	0
TEMPERATURE	1	Numeric: 1 = high 0 = normal	0
SNIFFLES	1	Numeric: 1 = present 0 = absent	0
SORE-THROAT	1	Numeric: 1 = present 0 = absent	0

10. **Interactive Processing.** The LENDER Bank offers mortgages on homes valued up to $500,000. The required down payment is calculated as follows:

    4% of the first $60,000 borrowed
    8% of the next $30,000 borrowed
    10% of the rest

    The amount borrowed cannot exceed 50% of the value of the house.

    Write a program to accept input from a keyboard that specifies for each borrower the amount that he or she wishes to borrow along with the price at which the house is valued. Display (1) a message that indicates if the amount the user wishes to borrow is acceptable (no more than 50% of the value) and (2) the required down payment if the amount to be borrowed is acceptable.

11. **Maintenance Program.** Modify the Practice Program in this chapter as follows. Every time a record is created on the master customer file, print the contents (edited) of that record as well.

Figure 9.9 (continued)

Sample Input

FLOOR PADS	010^00	05

Sample Output

```
ITEM FLOOR PADS
COST 10
YEAR    DEPRECIATION TO DATE    REMAINING VALUE
 1                 2                    8
 2                 4                    6
 3                 6                    4
 4                 8                    2
 5                10                    0
```

8. **Interactive Processing.** Write a program to enable users to enter:

   Name
   Street Address
   City, State, and Zip

   on three lines of a screen. Prompt the user for the number of mailing labels needed:

   ```
   DISPLAY 'ENTER NO. OF MAILING LABELS'
   ACCEPT NO-OF-LABELS
   ```

   Display all accepted input on the screen again, asking the user to verify that it is correct (e.g., `'IS THE DATA CORRECT (Y/N)?'`). If the user responds by keying Y for 'yes' (upper- or lowercase Y should be acceptable), then print the required mailing labels on the printer. If you are using a PC, make the interactive screen display as interesting as possible (e.g., use different colors, highlighting, etc.).

9. **Maintenance Program.** Modify the Practice Program in this chapter so that the interest earned each year appears on each detail line printed.

   Hint: For the first year, subtract the initial principal amount from the new balance at the end of the first year. Thereafter, subtract the previous year's balance from the new balance at the end of the corresponding year.