Introduction to Standard COBOL Programming

FREDRIC STUART

Introduction to Standard COBOL Programming

FREDRIC STUART

Hofstra University

HARCOURT BRACE JOVANOVICH, INC.

New York Chicago San Francisco Atlanta

To Fleurette

ISBN: 0-15-545963-5

Library of Congress Catalog Card Number: 74-12870

Printed in the United States of America

Preface

The primary purpose of this book is to teach the student the rudiments of computer programming in the COBOL language in the standardized form sanctioned by the American National Standards Institute (ANSI). However, since many users of the book may find that the computers at their disposal are equipped with non-standard compilers, paragraphs entitled "Non-Standard Compilers" explain the most common deviations from ANSI usage.

Program exercises at the end of each chapter afford the student a great deal of practice, as indispensable here as in any other language course. The student is required to write (and run) complete programs of increasing complexity. There are thirty exercise programs in all. Textual material is illustrated with twenty-two complete COBOL programs, which can serve as models for the exercises. Since COBOL is designed primarily for business data processing applications, the sample program subject matter has been chosen accordingly: sales tax, mailing lists, inventory records, payroll, and so forth.

Most important, the student is introduced to COBOL gradually. The first sample and exercise programs (Chapter 1) do not employ files or input data; they are simple but complete table-producing programs, much less frightening to a programming novice than the usual textbook "introductory" samples. The DISPLAY verb is used in Chapter 1 samples and exercises; ACCEPT is added in Chapter 2 to introduce input-using problems. Organized data files make their first appearance in Chapter 3; output editing and printer carriage control are introduced in Chapter 4.

Although some language features have been omitted because of the introductory nature of the book, the essential language elements are covered. Arithmetic and conditional instructions, briefly introduced in the early chapters, are examined in detail in Chapters 5 and 6. PERFORM is presented in Chapter 7 and is combined with OCCURS usage in Chapter 8. The discussion of subscripting techniques and the input and storage problems associated with tables is unusually extensive.

Chapter 9 enlarges the focus from language to technique by examining in detail both the underlying logic and the programming methods for three significant procedures: sorting, merging, and searching. Chapter 10 explains the use of such special compiler features as the source program library (COPY verb) and the EXAMINE, SORT, and SEARCH subprograms.

Language format skeletons for all formats discussed in the text are gathered together in Appendix A as an aid to students in writing programs. Also included in this appendix is a complete reserved word list.

Appendix B is a detailed comparison of COBOL and FORTRAN intended primarily for students who have previously had FORTRAN programming experience. Organized by chapter, it may be treated as a supplementary COBOL-learning device by such "second-language" students. It may also be of interest, however, to students learning COBOL as a first programming language.

Appendix C contains running and debugging advice to facilitate independent student work on the end-of-chapter exercises:

1. A trial data set (Figure 18) is provided for use in testing twenty-six exercise programs. The ten-card set, which represents a bank's checking deposit records, may be gang-punched for distribution to students or, since it is a short data file, left to individual keypunching.
2. Correct output is given for all thirty exercise programs. Students may verify that their programs are running correctly by comparing their output to the Appendix C results. In addition, a glance at the expected output will often clarify exercise instructions.
3. The last section of Appendix C contains general advice on debugging. Together with explanations given in Chapter 1, this enables students to "debug" their own programs without "bugging" their instructor.

My thanks to Thomas DeLutis of The Ohio State University for his many helpful comments and suggestions. Thanks also to Armand DeAmbrosis, Claire Gittleson, Michael Goldberg, Eugene Ingoglia, Kenneth Nelson, and Charles Sorgie of the Hofstra University Computing Facility.

Fredric Stuart

Contents

CHAPTER 1

Computers, Programmers, and Languages

A Complete COBOL Program

With apologies to Pope,[1] "The proper study of the programmer is the program." Since this is so, before we engage in any general discussion of machines or objectives, let us examine a short, but complete, computer program written in the COBOL language.

SAMPLE PROGRAM 1-1

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SALES-TAX.
REMARKS.     THIS PROGRAM PRODUCES A TABLE SHOWING SALES
             TAX DUE ON AMOUNTS FROM 25 CENTS TO 100
             DOLLARS.(BY QUARTER-DOLLAR INCREMENTS) ---
             SALES TAX RATE IS SEVEN PERCENT.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
77  AMOUNT    PICTURE 999V99.
77  TAX       PICTURE 9V99.
PROCEDURE DIVISION.
    MOVE 0.25 TO AMOUNT.
TAX-COMPUTATION.  MULTIPLY AMOUNT BY 0.07 GIVING TAX
                  ROUNDED.
    DISPLAY AMOUNT, "  TAX IS  ", TAX.
    ADD 0.25 TO AMOUNT.
    IF AMOUNT IS LESS THAN 100.25 GO TO TAX-COMPUTATION.
    STOP RUN.
```

A computer program is a set of instructions for the computer. The purpose of Sample Program 1-1 is evident from the programmer's comments in the REMARKS paragraph,

[1] Know thyself, presume not God to scan
The proper study of mankind is man.

—Alexander Pope, *Essay on Man*, 1733.

which is intended for the reader rather than the computer. While there is some degree of clutter caused by ritual terminology requirements in the first part of the program, you should be able to follow everything beneath "PROCEDURE DIVISION" without too much difficulty. The output produced by the program begins

```
00025  TAX  IS   002
00050  TAX  IS   004
00075  TAX  IS   005
00100  TAX  IS   007
etc.
```

and ends

```
09975  TAX  IS   698
10000  TAX  IS   700
```

Not very neat in terms of dollar signs, decimal points, and leading zeroes—but these problems are deferred until Chapter 4.

Equipment, Problems, and Problem Solvers

We are going to avoid technical questions on the electronic basis of computer performance, but it seems reasonable at least to ask: What is a computer expected to do for the user? For what kinds of problems is the computer really superior to other forms of equipment?

One answer to these questions is suggested by Sample Program 1-1: The computer is most useful for problems that require *repetition* of procedures. If, for example, we were interested only in the sales tax (at 7 percent) for an amount of exactly $37.65, it would hardly be worthwhile to go through the steps of writing a program, keypunching the written instructions, and running the program. The electric (or electronic) calculator would serve us better. In fact, even pencil-and-paper computation would be quicker than the computer solution.

In the area of business data processing, it would not be worthwhile to use the computer to figure one man's pay. But since the general method, once written as a computer program, may be used for unlimited numbers of pay records, computer execution of payroll computation is efficient. We shall see that all useful computer programs contain one or more program *loops*—sequences of instructions for which the programmer arranges repetitive execution. Such a loop is present in Sample Program 1-1, beginning and ending with the name "TAX-COMPUTATION."

We find ourselves, in the discussion above, emphasizing *speed* of execution. Certainly the most frequent response to the question "What is the advantage of the electronic computer over prior devices?" would be, "It is faster." But consider an analogy that has been quoted frequently by computer people (authorship unknown):

A human being is a *slow, error-prone genius.*
A computer is a *fast, accurate moron.*

There are three parts to this comparison—speed, accuracy, and mental ability—and each is important (particularly the last, as we shall shortly emphasize).

Computer Speed

As to speed, computer performance has been improved so rapidly that any precise quotation will always be proved an underestimate by next year. Let us simply summarize by saying that arithmetic operation times are now measured in microseconds (millionths of a second), or even nanoseconds (billionths of a second). I have just used a stopwatch to measure my solution time on an electronic calculator for

3827 X 4122 = 15,774,894

It took me five seconds; during this interval a modern computer could have performed well over a million such multiplications!

In terms of business data processing applications, these speeds permit execution in minutes of jobs formerly requiring hours or days with accounting machinery (e.g., large payrolls, inventory updates, etc.).

Computer Accuracy

The error-prone nature of human beings is easy to demonstrate; give ten people electronic calculators, have each work on the same problem requiring about three minutes of computations, and you are likely to have at least three different final answers. When calculating equipment requiring human data entry (i.e., key-pushing) is used in offices, it is common practice to have every calculation performed twice, by different operators.

Computer accuracy, on the other hand, is impressive—indeed, nearly flawless. Mechanical and electronic difficulties rarely lead to incorrect output; they are almost invariably diagnosed and signalled by the computer. Furthermore, both the likelihood of *human* error and the cost of such error are reduced when computers are used.

A computer program that has performed successfully once may be depended upon to do so over and over again. The error-prone human's work of planning the exact sequence of operations required to solve a problem (e.g., to produce a payroll from pay records) need not be repeated; and the possibility of incorrect execution of the sequence is eliminated.

Furthermore, the computer provides a permanent record of the input data used and the sequence of operations performed (program listing). In contrast, if an electric calculator operator keys in a number incorrectly or depresses the wrong operation button, no evidence of the mistake (or its location in the procedure) is left behind.

We have said also that the *cost* of human error is reduced when computers are used. Murphy's Law, which antedates Parkinson's Law and the Peter Principle, says,

"If anything can go wrong, it will."

A computer corollary states,

"There is at least one error in every program."

You will see, however, that such errors are more easily detected and corrected than mistakes made by calculator operators. Many programmer errors (in language syntax) are successfully diagnosed by the computer (actually, by the *compiler program*, discussed below) and may be corrected by altering one or two punched cards. Furthermore, the kind of human oversight that frequently requires complete reworking of lengthy procedures ("My God, we forgot to change the tax rates!") is far less costly when a few minutes of computer rerun, rather than hours or days of human computational labor, are required. (Logical errors committed by the programmer can, however, be quite costly in commercial data processing applications.)

Computer Imbecility

Human beings are geniuses and computers are morons, says the third part of the comparison. This is why the art of *programming* is at the heart of successful computer operations. A *program* is a step-by-step sequence of instructions that the computer is to follow. The computer can and will do nothing except what has been explicitly called for. In most instances it will follow its program blindly, regardless of what kinds of logical disasters may occur on the way. A logical error by the programmer (for example, calling for multiplication by 7 to compute a 7 percent sales tax) will induce the computer to produce wrong answers with its customary great speed and "accuracy."

Newspaper stories and accounting department excuses that cite "computer error" are really shifting the blame unfairly. The "error" has always actually been committed by a human being, usually the programmer (but sometimes an operator). When a genius commands a moron, it is ignoble to pin the mistakes on the moron.

Computer Hardware

Every computer system has three essential kinds of component equipment: a central processing unit, input devices, and output devices.

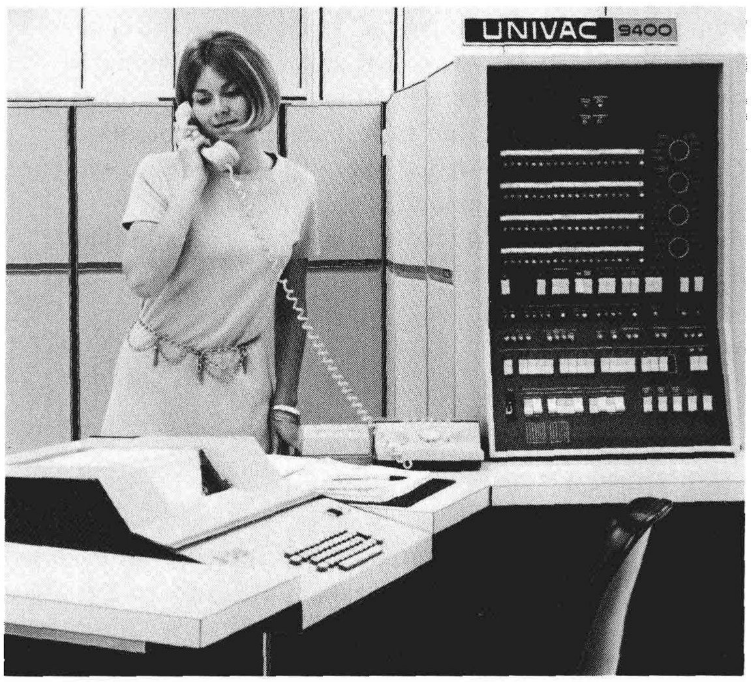


Figure 1 A Central Processing Unit (CPU)
(Courtesy of Sperry Univac, a division of Sperry Rand Corporation)

The central processing unit ("CPU"; see Figure 1) contains the primary storage areas (the main "memory"), in which both instructions and data relating to a specific job reside when that job is being executed. This unit houses the essential control circuitry and usually also has an external control panel (with switches and lights) and a console typewriter, both of which serve as low-volume input/output devices for the system operator.

Two common input/output devices are the card reader (input only; Figure 2) and the high-speed printer (output only; Figure 3). The smallest computer systems are frequently restricted to these two units. The card reader may be accompanied by a card punch, which serves as an output device.

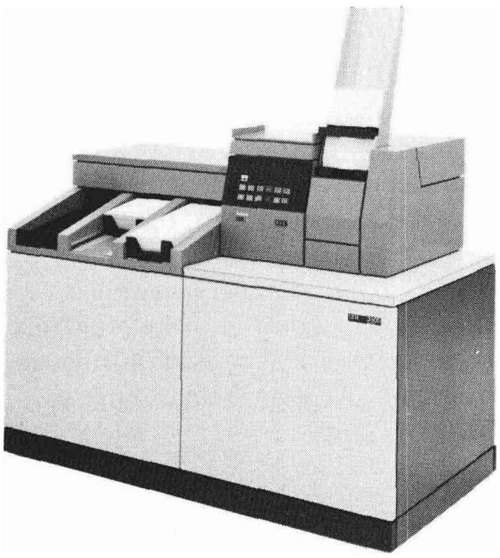
Two other media of great importance for high-volume input/output operations are magnetic tapes and magnetic disks. These each serve as both input and output medium and also provide significant *secondary storage* areas. That is, programs and data stored on tapes and disks are accessible to the CPU, which transfers them to its *primary* storage area as needed.

Magnetic tape is handled by units called tape drives (Figure 4), using reels on which the tape is wound like motion-picture film. Access to material on tapes is *sequential*—that is, to reach information in the middle of a tape it is necessary to wind through the first part of the tape. By contrast, disks are *direct access* media; any part of the magnetic disk may be reached with approximately equal speed. Each disk drive (Figure 5) contains one or more spindles on which are stacked several platters resembling large phonograph records.

Input and output equipment may also include optical scanning devices, remote terminals, and graphic plotters. However, COBOL programs are usually concerned principally with cards, printer, tapes, and disks.

Computer Software and Computer Languages

All of this sophisticated "hardware" must be directed by written programs, the "software" provided by both the manufacturer and the local programmers. The ultimate



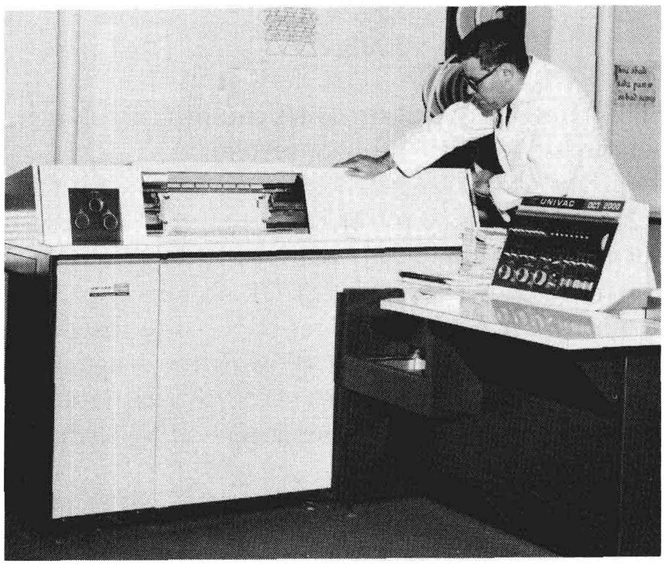


Figure 2 (left) A Card Reacer (Courtesy of IBM Corporation)
Figure 3 (right) A High-Speed Printer (Courtesy of Sperry Univac, a division of Sperry Rand Corporation)

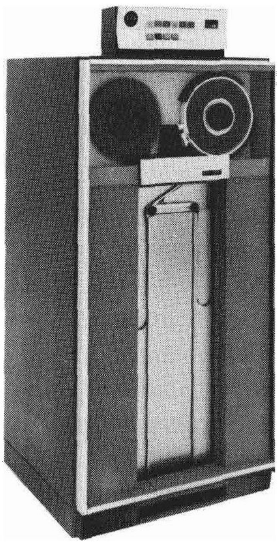
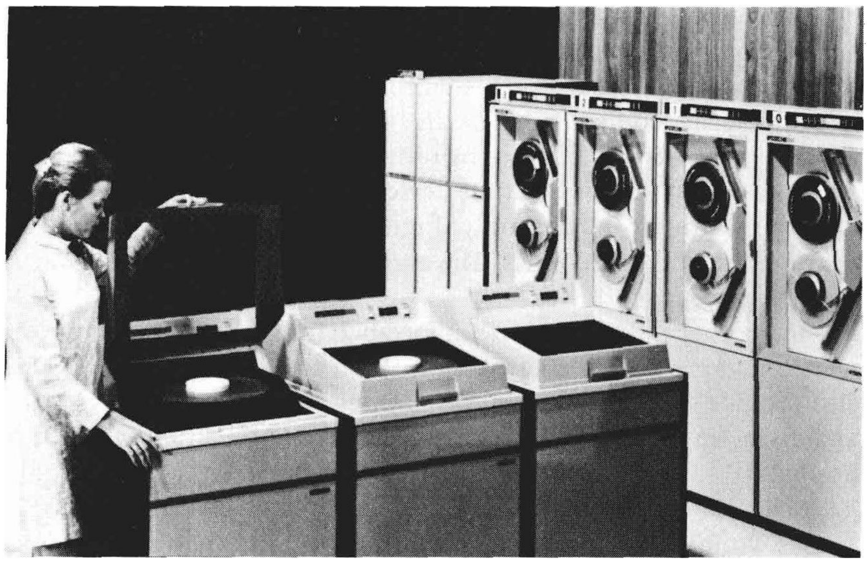



Figure 4 (left) A Magnetic Tape Unit (Courtesy of IBM Corporation)
Figure 5 (right) Magnetic Disk Drives (Courtesy of Sperry Univac, a division of Sperry Rand Corporation)

form of any program is actually a strange dialect called *machine language*, which is the only set of instructions directly executable by the computer. A sequence of machine language instructions looks like this (UNIVAC System 70/7):

```
48 00  23C0
4C 00  23C2
40 00  2310
D2 01  2310  2310
48 00  2310
4E 00  2028
F3 17  3002  2028
96 F0  3003
```

This contains operation codes and primary storage addresses, which are commonly expressed in the hexadecimal number system.

This sort of language is (1) rather difficult for programmers to write and/or read and (2) usually unique to the computer make and model being used. In the 1950s, the usefulness of electronic computers was greatly enhanced by the development of *compiler languages*, such as FORTRAN and ALGOL, followed in 1960 by COBOL.[2] A compiler language is problem-oriented in form, as opposed to machine-oriented. It is designed to make programs easy to write and read, by allowing the programmer to use combinations of English language and simple (decimal system) arithmetic notation. For example, all eight machine language statements shown above are generated by writing in COBOL merely

[2] COBOL was originated by a committee formed in 1959 (several years after FORTRAN, developed at IBM, had become widely used for "scientific" applications programming), which included government and industry computer users, manufacturers, and university representatives. The earliest version of the language, designed for business applications, became available in 1960.

```
MULTIPLY 3 BY 4 GIVING ANSWER.
```

Instructions written in compiler language comprise a *source program,* which must be translated into an *object program* in the computer's machine language prior to execution. The translation is accomplished by a *compiler program* provided by the computer manufacturer.[3]

As we have seen, during the translation process each of the programmer's source statements may produce many machine-language instructions. However, the programmer need not be concerned with the details of movement of data into and out of registers, recording of numeric storage addresses, conversion from decimal to other number systems, and so forth. He is free to concentrate on the logic of his own problem.

A compiler program attends to these details as it translates the source language, which is standardized, into a form acceptable to the particular computer being programmed. In order to make their machines accessible to programmers in many fields and thereby increase their marketability, computer manufacturers generally provide compiler programs for a number of the more common compiler languages for each model of computer. Thus, a typical "software" package includes compiler programs for FORTRAN ("FORmula TRANslation"), COBOL ("COmmon Business Oriented Language"), RPG ("Report Program Generator"), BASIC (a remote terminal language), and perhaps others as well.

ANSI COBOL

The American National Standards Institute (Formerly American Standards Association) has coordinated efforts to standardize several of the leading compiler languages. In this text we shall follow the language rules of ANSI COBOL. In the few instances in which manufacturer-supplied compiler programs tend to depart from this standard, we shall point out the essential differences so that you may adjust your programs if necessary.

COBOL Coding Forms

This hardware/software discussion has taken us away from our principal concern: writing programs in the COBOL language. In case you have forgotten what a COBOL program looks like since Sample Program 1-1, let's tackle another problem.

To help its field staff compute premium payments, an insurance company requires a table showing, for each number of days from 1 to 365, what *proportion* of the year has elapsed, to the nearest hundredth. A program designed to produce the table appears in Figure 6.

The general form of this program is close to that of Sample Program 1-1; in fact, you will note that six identical statements appear in both programs. Before discussing general COBOL

[3]This discussion skips an intermediate type of programming language, the *assembler* languages. Assemblers substitute mnemonic symbols for the digital operation codes and storage addresses used in machine languages. Though the substitution of alphabetic codes simplifies life somewhat for the programmer, the required translation to machine language is usually on a one-to-one basis. Thus, for example, eight assembler language statements would probably be required to duplicate the sequence shown on page 6.