STRUCTURED COBOL

RUTH ASHLEY

STRUCTURED COBOL

RUTH ASHLEY
Co-President of DuoTech
San Diego, California

John Wiley & Sons, Inc.
New York • Chichester • Brisbane • Toronto

Publisher: Judy Wilson
Editor: Irene Brownstone
Production Supervisor: Ken Burke
Artists: Carl Brown, Winn Kalmon
Makeup: Karla Savage

Library of Congress Cataloging in Publication Data:

Ashley, Ruth.
Structured COBOL.

(Wiley Self-Teaching Guides)
Includes index.
1. COBOL (Computer program language) — Programmed instruction. 2. Structured programming — Programmed instruction. I. Title.
QA76.73.C25A84 1980 001.64'24 79-27340
ISBN: 0-471-05362-7

Printed in the United States of America

80 81 10 9 8 7 6 5 4 3 2 1

How To Use This Book

This book has no special prerequisites. If you have taken a course or read a book on data processing, you should have no problem working through this Self-Teaching Guide. If this is your first venture into the world of computers, you may benefit from an introductory overview of the data processing environment. Another Self-Teaching Guide, Introduction to Data Processing (2nd edition) by Martin Harris, provides excellent background for this book.

This Self-Teaching Guide consists of thirteen chapters, each of which will bring you deeper into Structured COBOL, building on the previous information. Each chapter begins with a short introduction, followed by objectives that outline what you can expect to learn from your study of that chapter. Many chapters end with a Summary Exercise, a program for you to write, which lets you pull together and apply the new material you have learned. The Self-Test at the end of each chapter lets you assess how well you have met the objectives.

The body of each chapter is divided into frames—short numbered sections in which information is presented or reviewed, followed by questions that ask you to apply the information. The correct answers to these questions follow the dashed line in each frame. As you work through the Guide, use a card or folded paper to cover the correct answer until you have written yours. And be sure you actually write each response, especially when the activity is statement coding. Only by actually coding COBOL statements and checking them carefully (letter by letter, space by space) can you get the most from this Self-Teaching Guide.

As you code statements and programs throughout this book, you may use the forms provided or you may wish to use actual COBOL coding forms, available in most college bookstores or supply stores.

In the back of the book are three Appendixes. Appendix A lists COBOL reserved words, Appendix B shows the standard collating sequence, and Appendix C summarizes the formats of the Procedure Division statements discussed in this book. An index is also provided, so you can use this book for later reference.

Contents

Introduction

COBOL is a computer programming language that was designed to solve the data processing problems of business. A number of years ago, a national committee, under the auspices of the U.S. government but with representatives of most computer manufacturers, studied the many versions of COBOL. (Almost every large installation had its own version at that time.) The standardized COBOL that was adopted by that committee became American National Standard, or ANS COBOL. Many compilers today have variations, but they are extensions or modifications to the basic ANS COBOL. Reference manuals clearly indicate where the variations differ from the standard.

Structured COBOL deals with the COBOL language—the same COBOL that programmers and computers have been using for years. "Structured" here refers to programming, and, as such, is independent of the COBOL language. Structure is an approach to programming in which we are concerned with clarity as well as effectiveness. A structured approach to learning COBOL makes a complex subject easier to learn and will help you develop good coding habits automatically. Most computer installations ask all their COBOL programmers to use structure. Since Structured COBOL programs are clear and easy to read, they are much easier to modify than "traditional" COBOL programs. This is an important distinction since as much as fifty percent of programmers' time may be spent in modifying old programs.

Programs written in Structured COBOL make sense to all COBOL programmers and COBOL compilers. In fact, one of the major advantages of structured programming is that it is readable by human beings. A person who isn't a programmer at all can read a Structured COBOL program and figure out what is happening.

When you complete this Self-Teaching Guide, you will be able to write COBOL programs that will require no alterations to run on most systems, and only a few changes for others. The Environment Division in the COBOL program, because it describes the machines and equipment used, contains most of the material that varies among systems. When you begin actually running programs, therefore, you will have to find out what Environment Division entries are standard for the system you will be using. The majority of the COBOL program, however, is machine independent and will run equally well on almost any computer system.

The COBOL program you write is called a source program. This source program is then fed into a source computer, where it is compiled or translated into a machine-language program, the object program. One COBOL statement may be translated into as many as fifty consecutive object statements, since extremely detailed instructions must be given to the computer in terms it understands. At the compilation stage, many of the errors (or bugs) in a program become apparent. Errors in spelling of the special COBOL words, omission of required spacing or punctuation, and use of incorrect formats are just some of the factors that can hinder the compilation of your source program into the machine-language object program. Learning to program in machine language does not circumvent these problems; machine-language format, syntax, and sequencing require even more attention to the details of both the language and the system. When you program in a higher level language, such as COBOL, the compiler provides error messages from the source computer, which will help you correct your program. The object computer is used then to execute your mechanically correct program.

In this guide, we shall stick as closely as possible to ANS COBOL, using a structured approach. Structured COBOL is self-contained, including all the materials necessary for you to study ANS COBOL; you do not need access to a computer to learn how to write a program in COBOL. Of course, you will need access to a computer to develop what you learn here into actual programming.

This book includes an introduction to the problems you'll encounter when you first begin to compile, test, and run your Structured COBOL programs. Until you begin that process, you may not really appreciate the benefits of Structured COBOL, even though you know the statements and can code the control structures. Structured coding makes the testing and debugging part of programming much more manageable.

Since we don't know what system is used in your installation, we can't tell you exactly how to go about it. You will probably need guidance from an instructor, a fellow student, or a congenial programmer. And before you run any programs, you may find it useful to review the COBOL reference manual for your installation. Beyond the basics presented in this book, you will find that COBOL has many more options, some more statement types, and various aids for testing your programs, which will be useful as you actually begin to apply what you have learned in Structured COBOL.

CHAPTER ONE

The Structure of Programming

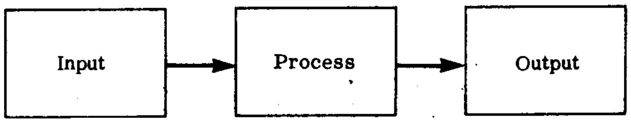
The first chapter of Structured COBOL deals not with COBOL, but with structured programming in general. Before we can begin to apply coding rules, we need to get an overview of the structure of programming in a business environment. Virtually all programming problems can be seen as combinations of three structures. We will see how these three simple structures can be combined to solve most business programming problems. In this chapter we'll look at a few ways to describe the general structure of a problem.

All of this will help you to begin thinking about programming as a method of creating procedures, called programs, to solve business problems.

When you have completed this chapter, you will be able to:

- state two advantages of structured programming over traditional programming;
- identify the type of programming problem represented by an example;
- identify examples of sequence, selection, and repetition in a program structure;
- interpret a simple hierarchy chart; and
- interpret a simple pseudocode design.

1. The basics of programming can be diagrammed like this:



When you write a program, you specify an exact procedure the computer will follow to solve a problem. In essence, you tell the computer what it will get as input data, what to do with it, and what you want as output data. Everything between the input data and the output data is processing. In a business environment, a great deal of data is handled. Problems such as inventory control, personnel file maintenance, and payroll processing all involve input of data, processing, and output of the results.

Consider a payroll problem—any payroll problem.

(a) Give one example of input information that is needed.

____

(b) Give one item of processing.

____

(c) Name one piece of possible output data.

____

- - - - - - - - - - - - - - - - - - -

(a) hours worked, rate per hour, person's name, etc.
(b) multiply hours times rate, figure tax, etc.
(c) a paycheck, records, etc.

2. Consider a situation in which a department store has a large file (or collection) of customer account records. Every day the accounting department receives several hundred payments in the mail. Clerks keypunch a set of data cards, each with customer number and amount paid. These cards become the input data for a program that looks up each customer from whom a payment was received, and subtracts the payment from the current balance. Finally, it prints out names of any customers who paid too much, and it also prepares a daily "income statement" of the total amount received that day by mail.

   (a) Name two items of input to this problem.

   ____________________ ____________________

   (b) Name two elements of processing.

   ____________________ ____________________

   (c) Name two items of output.

   ____________________ ____________________

- - - - - - - - - - - - - - - - - - -

(a) customer file, payment cards; (b) look up accounts, subtract payments, add up all payments, check if balance is less than zero; (c) new balances, total of payments, list of overpaid customers

3. Most business problems fall into four general categories. <u>Update problems</u> involve modifying master sets of data. <u>Summary problems</u> use input data to find specific information or totals. <u>Report problems</u> use input data to produce printed reports in a specific format. <u>Editing problems</u> require a detailed "edit" or verification of input data, often as a preparation for its use as input to still another program.

Indicate whether each problem below is a summary, update, report, or editing problem.

(a) Go through a set of patient records and print each patient's name, length of hospital stay, and total bill. ____________

(b) Process a set of patient records to find out the average cost per day per patient. ____________

(c) Process a set of patient records to change attending physician "Sweeney" to "McDowell". ____________

(d) Process a set of payment records to make sure each is in correct format and includes an eight digit patient number. ____________

- - - - - - - - - - - - - - - - - - -

(a) report; (b) summary; (c) update; (d) edit

4. Many business problems contain elements of all four general problem types. In the situation described in frame 2, which aspect of the problem, if any, is of each type?

(a) Summary ____________

(b) Update ____________

(c) Report ____________

(d) Editing ____________

- - - - - - - - - - - - - - - - - - -

(a) income statement (total amount in); (b) changing customers' balances; (c) list of overpaid customers; (d) none indicated in the problem statement

5. Each problem, no matter what type, involves input, processing, and output. Every program—which is the procedure for solving the problem—must include at least one input function, one processing function, and one output function. A given program may include many of each type of function, depending on the complexity of the problem. In addition, every program has an overall control function which calls in any of the other functions, as needed. Structured programming is based on these control functions. They control the sequence in which all other functions are executed.

(a) What type of program would require a control function? ____________

(b) What type of function is involved in the tax computation in a payroll problem solution? ____________

(c) What type of function would call on a tax computation function?

____

- - - - - - - - - - - - - - - - - -

(a) all types; (b) processing; (c) control

6. In a program, flow of control dictates which instruction the computer considers next. Most high-level languages, including COBOL, offer various ways to specify flow of control. Traditional programming is based on flow of control, and takes full advantage of all the ways of modifying flow that are available. Some of these ways require very little internal storage and/or very little time. Some are "elegant" and allow the programmer to code fewer lines. The result may be an efficient program, but it often communicates very poorly with human readers. In structured programming, we restrict ourselves to a very few ways to control sequence of execution in programs. The enhanced clarity of coding that results means programs can be written more quickly, debugged and tested more quickly, and revised much more easily in the future.

(a) Which generally uses more ways to control sequence of execution—traditional or structured COBOL? ______________________

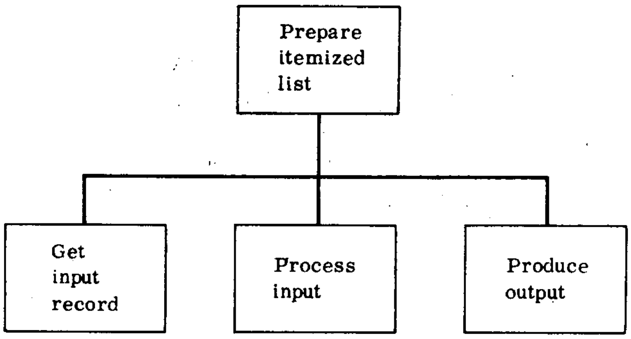
(b) Which are generally easier for you to read—traditional or structured COBOL programs? ______________________

- - - - - - - - - - - - - - - - - -

(a) traditional; (b) structured

7. Data processing problems have been programmed using traditional techniques for many years. You may be familiar with flowcharts, for example. Let's look more closely at the special features of structured programming.

Structured programming looks at a problem as a hierarchy of functions to be performed. Higher level functions are control functions; they control the execution of lower level functions, which may be more control functions, input functions, processing functions, or output functions. Each function, sometimes called a module, is invoked only by higher level control functions.



This example shows a high-level module with three lower level modules. What modules shown could be invoked by each of these?

(a) Get input record ______________

(b) Prepare itemized list ________________________________

- - - - - - - - - - - - - - - - - - - -

(a) none (no function is shown at a lower level than Get input record);
(b) Get input record, Process input, and Produce output (all of the lower level functions)

8. Structured programming enables you to follow a structured design and produce a program more quickly than would traditional programming. And the structured program will be easier to test and, most important, considerably easier to read than a traditional program. In a typical business computer installation, more programmer time is spent maintaining (which includes modifying) old programs than is spent writing new ones. And programmer costs are the largest expense category in many computer installations. Computer time costs are coming down, while salaries go up. Therefore many installations have decided against micro-efficiencies of "elegant" programming in favor of the more efficient use of human resources available through structured programming.

Which of the following factors contribute to the demand for structured programming?

_____ (a) Computer time is more expensive these days.

_____ (b) Computer personnel are more expensive these days.

_____ (c) Efficient use of machine resources is critical.

_____ (d) Readable programs are easier to maintain.

- - - - - - - - - - - - - - - - - - - -

b, d

9. Each module of a structured program has one entry point and one exit point. This makes it relatively easy for a programmer to read, write, or understand a program. Names given to modules and pieces of data are easier to read if they are meaningful. Instructions that have some meaningful relationship to one another can be grouped together by spacing or indentation. You'll be using all these techniques in later chapters as you code structured COBOL programs.

   Name two characteristics that make a structured program easy to understand.

   ____

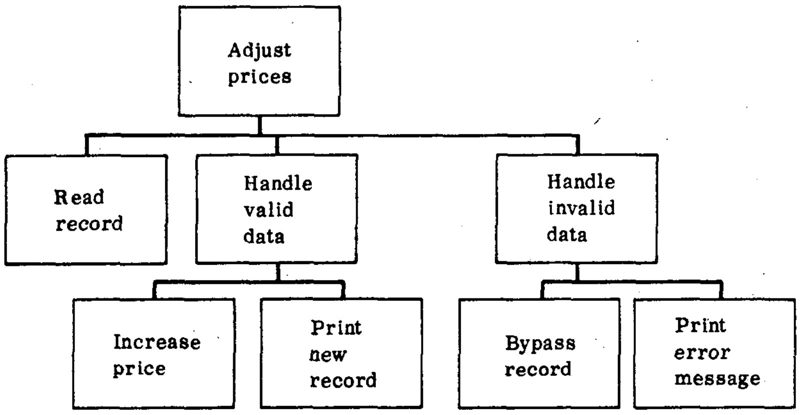
   ____

- - - - - - - - - - - - - - - - - - - -

(any two of these) meaningful names for data and modules; indentation and spacing to group instructions; separate modules for different functions

10. Traditional programming relies on the flowchart to show the "flow of control" in a program. Although flowcharts can be (and often are) adapted for structured programming, flowcharts focus on control rather than on functions so we will not use them in this book. We will use narrative descriptions along with structured hierarchy charts and a pseudocode (which you'll see shortly) to express program designs in this book.

    Examine the following hierarchy chart.



Which narrative below fits the structure shown in the chart?

_____ (a) Records that contain valid data will be bypassed.

_____ (b) Records will be processed under the control of either Handle valid data or Handle invalid data.

_____ (c) Prices in valid records only will be adjusted.

- - - - - - - - - - - - - - - - - - -

b, c (both fit the chart)

11. Hierarchy charts show the general structure of a problem. Recall our earlier discussion of types of programming problems.

    (a) Which type of problem does the hierarchy in the last frame represent—summary, update, report, or editing? ______________________

    (b) Name the control functions in that chart. ______________________

    ____

- - - - - - - - - - - - - - - - - - -

(a) update (we're changing something in the record)
(b) Adjust prices, Handle valid data, Handle invalid data

CONTROL STRUCTURES

Sequence Structure

12. In the normal way of operating, the computer executes one statement after another, in sequence, unless it receives an instruction to the contrary. The sequence in which the instructions are given to the computer determines the sequence in which they are executed—we can call this a sequence structure. For example, suppose you want the computer to print a line with your name. Your instructions (in everyday English) might be

    1. Here is my name
    2. Print it

    No other sequence for these two instructions would make sense. Below are some English instructions to find and print the total price. Number them (from 1) in the correct sequence.

    _____ (a) Multiply quantity by unit price to get total price

    _____ (b) Unit price is 7.00, quantity is 5

    _____ (c) Print total price

- - - - - - - - - - - - - - - - - - -

(a) 2; (b) 1; (c) 3

Selection Structures

13. Sometimes we don't want every instruction executed in sequence, but want to select some instructions to be executed only under certain conditions.

For example, if an item is not food, we may have to add tax to its price. In this case, we use a selection structure. The standard selection structure is the IF-THEN-ELSE, often simply called the IF structure. Here is how we can show a selection control structure:

```
IF item is not food
THEN
      add tax
ELSE
      don't add tax
ENDIF
```

We use IF to specify a selection criterion—a condition. When the condition is true, we want the computer to do whatever we have written under THEN. When the condition is false, we want the computer to do whatever we have written under ELSE. The ENDIF marks the end of the IF control structure.

The condition, whether it is true or false, determines whether the THEN or ELSE action will be done. In no case will they both be done. As you'll see later, either can be omitted, however. The selection structure above will have the same functional effect if it is written like this:

```
IF item is food
THEN
      don't add tax
ELSE
      add tax
ENDIF
```

See if you can write a selection structure (condition and general actions) based on the chart in frame 10. We've included the special words for you.

```
IF ____________________________
THEN

   ____________________________
ELSE

   ____________________________
ENDIF
```

- - - - - - - - - - - - - - - - - - -

There are two ways you might have answered this. Both are equally correct.

```
IF data is valid                IF data is not valid
THEN                            THEN
   handle valid data               handle invalid data
ELSE                            ELSE
   handle invalid data             handle valid data
ENDIF                           ENDIF
```

14. The way we write the instructions here is not really computer code, so we generally call it pseudocode. It simulates the sequence of statements as you will code them in a computer program. A pseudocode segment can be translated into COBOL, or almost any computer language.

Another item worth noting in the pseudocode selection structure is that the THEN and ELSE sections can each contain many actions. For example, the THEN action "handle valid data" from the last frame could be replaced with:

```
THEN
      increase price
      print new record
ELSE
.
.
.
```

Referring to frame 10, indicate here what sequence of actions would replace "handle invalid data" in the selection structure.

____

- - - - - - - - - - - - - - - - - - -

bypass record; print error message

15. The next action in sequence after an ENDIF will be executed whether the condition was true or false. The pseudocode below combines the sequence and selection structures. Examine it, then answer the questions that follow.

```
Get a card
IF employee is full-time
THEN
      add 1 to full-time count
ELSE
      add 1 to part-time count
ENDIF
Print name from card
```

(a) Suppose the result of "Get a card" indicates a part-time employee.

Which instruction will be selected—THEN or ELSE? ______________

(b) What instruction will be executed after "add 1 to full-time count"?

____

(c) What is the purpose of ENDIF? ______________________

____

- - - - - - - - - - - - - - - - - - -

(a) ELSE (the condition—employee is full-time—is not true); (b) print name from card; (c) it marks the end of the selection control structure

Repetition Structure

16. Besides sequence and selection, we often want to specify repetition, or iteration, to have instructions executed repeatedly. We can specify that a statement or group of statements will be executed repeatedly until a condition becomes true. The control structure is called PERFORM UNTIL. We specify that some actions will be performed until a condition becomes true. Here is a simple COBOL repetition structure.

```
PERFORM UNTIL no more cards
    add 1 to card-counter
    print line
    get card
ENDPERFORM
```

The structure specifies that the action statements will be executed repeatedly until no more cards are in the file. Here "no more cards" represents a condition. The ENDPERFORM marks the end of the PERFORM control.

(a) How many times will the statements be executed if the input deck has no cards at all? ____________

(b) How many times will the statements be executed if there are 12 cards in the deck? ____________

(c) What two control structures are represented in this example?

____

- - - - - - - - - - - - - - - - - -

(a) none; (b) 12; (c) sequence and repetition

17. The following pseudocode uses all three control structures—sequence, selection, and repetition.

```
Get a card
PERFORM UNTIL no more cards
    IF employee is full-time
    THEN
        add 1 to full-time count
    ELSE
        add 1 to part-time count
    ENDIF
    Print name from card
    Get a card
ENDPERFORM
Print count totals
```