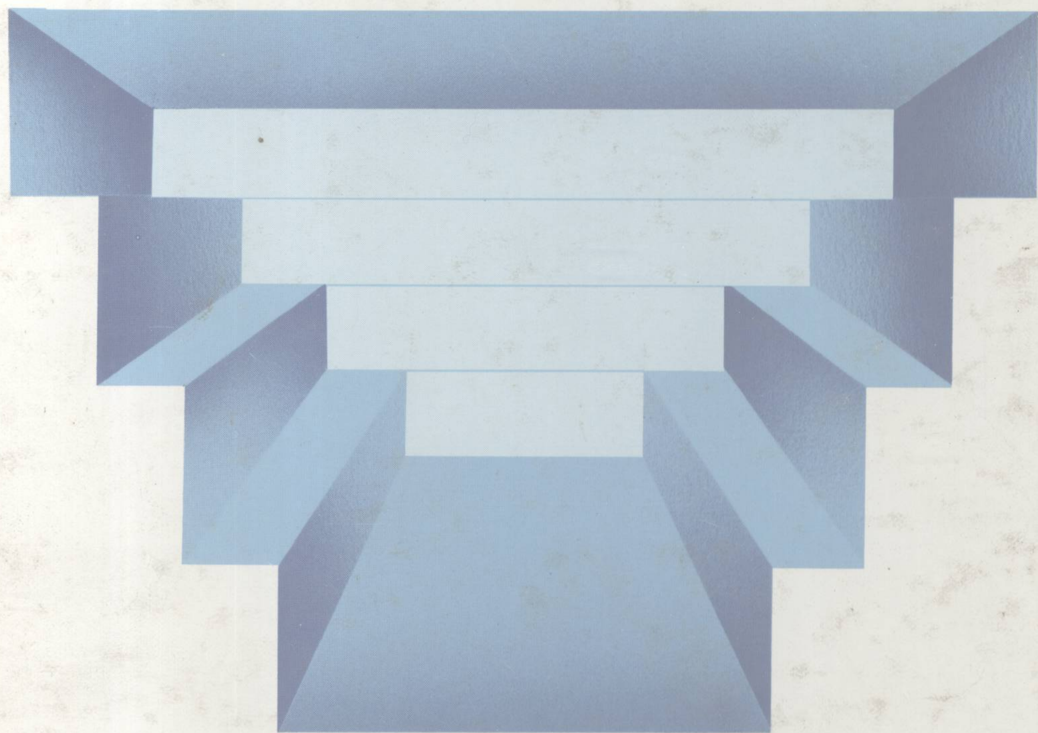
EDWARD YOURDON
CHRIS GANE
TRISH SARSON
TIMOTHY R. LISTER



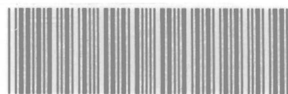
LEARNING TO PROGRAM IN STRUCTURED COBOL PARTS 1 AND 2

A YOURDON BOOK

8050981

Learning to Program in Structured COBOL

Parts 1 and 2


E8050981

Edward Yourdon
Chris Gane
Trish Sarson
Timothy R. Lister

Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging in Publication Data

Main entry under title:
Learning to program in structured COBOL.

(Prentice-Hall software series)
Includes indexes.
1.–COBOL (Computer program language)
2.–Structured programming. I.–Yourdon, Edward.
II.–Title. III.–Series.
QA76.73.C25L4 001.6'424 79-12427
ISBN 0-13-527713-2

PRENTICE-HALL SOFTWARE SERIES
Brian Kernighan, advisor

Production Supervision by Lynn Frankel
Cover Design by Suzanne Behnke
Manufacturing Buyer: Gordon Osbourne

©1979, 1978, 1976 by YOURDON inc., New York, N.Y.

All rights reserved. No part of this book
may be reproduced in any form or
by any means without permission in writing
from the publisher.

Printed in the United States of America

10 9 8 7 6 5 4 3 2

PRENTICE-HALL INTERNATIONAL, INC., *London*
PRENTICE-HALL OF AUSTRALIA PTY. LIMITED, *Sydney*
PRENTICE-HALL OF CANADA, LTD., *Toronto*
PRENTICE-HALL OF INDIA PRIVATE LIMITED, *New Delhi*
PRENTICE-HALL OF JAPAN, INC., *Tokyo*
PRENTICE-HALL OF SOUTHEAST ASIA PTE. LTD., *Singapore*
WHITEHALL BOOKS LIMITED, *Wellington, New Zealand*

ACKNOWLEDGMENT

The following paragraphs are reprinted from *American National Standard Programming Language COBOL,* published in 1974 by the American National Standards Institute, New York:

> COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.
>
> No warranty, expressed or implied, is made by any contributor or by the CODASYL Programming Language Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.
>
> The authors and copyright holders of the copyrighted material used herein
>
>> FLOW-MATIC (trademark of Sperry Rand Corporation), Programming for the UNIVAC® I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form No. F 28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell
>
> have specifically authorized the use of this material in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.

FOREWORD

Learning to Program in Structured COBOL is intended for people with no previous knowledge of computers, who want to learn to program in COBOL, the most widely used computer language. Parts 1 and 2 of the *Learning to Program* series incorporate the methods and styles of "structured" programming, which have been shown to be more productive than traditional programming techniques.

Learning to Program in Structured COBOL, Part 1 can be used as a stand-alone introduction to structured programming or it can be used in conjunction with the more advanced concepts and features that are presented in Part 2. Both are complete texts and can be used effectively with other structured programming texts; however, Parts 1 and 2 were designed to be used together as a teaching unit and are complementary in content and approach as well as style and format.

The books can be used either for self-study or as the texts for an industrial or college course. If you are an instructor, please read the following Notes for Instructors, which discuss possible uses of this book and its companion volume as basic texts. If you plan to use either or both of the books to study COBOL on your own, you will find the frequent questions and exercises helpful, especially if you work them *before* checking the answers provided.

We have tried to make your learning easy, thorough, and fun. If you actually can run some of the exercise programs on a computer, you will enhance your learning, and find it fascinating to build a realistic commercial data processing system.

New York E.Y., C.G., T.S.

试读结束，需要全本PDF请购买 www.ertongbook.com

NOTES FOR INSTRUCTORS

One of the objectives of each book in this series is to serve as the text for a three-week full-time training course or a one-semester college course for people with little or no prior exposure to data processing.

Apart from teaching COBOL entirely in the context of structured programming, the course design incorporates several well-established educational techniques that have not, so far as we know, been applied in this area before. They are

- the concept of COBOL as a foreign language
- the concept of the spiral curriculum
- the concept of the "theory/practice sandwich"

COBOL as a foreign language

Teaching a foreign language using a grammar is not as effective as teaching via a set of syntactic structures. That is, it is better to learn a language by learning basic conversational exchanges, such as "Have you got an X? Yes, I have an X," rather than to learn "I have, you have, he has, she has. . . ." If we view COBOL in this light, we see that the standards manual and manufacturers' reference manuals are *grammars* of COBOL; they set forth the rules of the language in a formal way, exploring all the options of each statement, however obscure and rarely used. Many texts and courses explain the reference manual, but essentially follow the same pattern. In this text and in courses based on it, we use the four structures — process, decision, loop, and CASE — as the building blocks and teach the language with a structural rather than a grammatical orientation.

Regarding COBOL as a foreign language also suggests that we minimize the history and geography of the "country" concerned. While we do not question that a well-rounded professional should know the history of data processing from Hollerith to HISAM, we believe that history is irrelevant and confusing to the beginner, because it is of no help in performing his central task of solving problems with code. Likewise, while a profes-

sional COBOL programmer should know enough about the architecture of the hardware to appreciate the implications of alternative coding techniques, the beginner needs only a very simple model of main storage and common peripherals. We have taken pains to concentrate initially on the production of readable, changeable code, rather than on any considerations of run-time efficiency; for example, binary representation is not discussed until Chapter 10, in Part 2.

The spiral curriculum

Usually, topics in a subject can be arranged in a linear order, one after another. However, this is difficult to do in teaching programming, because of the amount of interdependence between topics; the instructor is in the chicken-egg situation of not being able to teach topic A properly before the students know about topic B, and not being able to teach B before they know about A. The solution is to design a *spiral* curriculum in which all topics are treated several times at progressively increasing levels of detail. As you will see, Parts 1 and 2 develop five levels of the spiral:

Chapter 1:	brief explanation of the whole program development process, and a walkthrough of a simple COBOL program
Chapters 2,3,4,5,6:	establishment of the basic structures and language subset, with a thorough discussion of COBOL logic
Chapters 7,8,9,10:	use of auxiliary storage, a larger subset of the language, and internal data representation
Chapters 11,12:	use of tables and advanced input-output techniques, including buffering and blocking, and indexed and relative input-output

Chapters 13,14,15:	use of sorting and merging techniques, testing and debugging strategies, efficiency as well as optimization

The theory/practice sandwich

It is often a temptation for someone who is expert in a subject to teach theory at a more profound level than is desirable. This is partly because the more deeply one understands the theory behind a subject, the simpler it appears. So, the instructor may feel that the subject can be made simple to the learner by teaching the underlying theory at the same depth as the instructor understands it. This is a fallacy; the learner needs to start with familiar, concrete ideas and simple skills, and *then* learn abstract concepts. After a while, he can treat these abstract concepts as concrete things and then learn deeper-level concepts, and so on.

Introducing the subject of computer programming by teaching binary arithmetic is a case in point; it is true that, at a deep level, the computer is merely performing operations on binary strings, but that is no help to the beginner. The temptation to teach too much theory too early can be resisted by asking "What is the simplest act of mastery the learner can do next? What is the minimum theory he must know in order to do that act of mastery?" The idea behind the "theory/practice sandwich," then, is a curriculum that, within each spiral, has the structure

minimum theory

simple act of mastery

next item of minimum theory

next act of mastery

and so on.

The sequence of acts of mastery around which the book builds is

read a simple program

make a small modification to a program

write a card-to-print program

enhance the program to do some arithmetic

enhance the program to do complex logic

enhance the program to write a tape file

maintain the tape file

use the tape file to create an indexed disk file

use the indexed disk file in a simple accounting system

and so on.

Throughout the texts, specific program exercises, as well as the overall concepts, build on each other in such a way as to combine the maximum of learning with the minimum of coding and keypunching.

In addition, instructors may find it useful to refer to the suggested lesson plans and lecture notes for the first thirty sessions of a course (three hours per session), provided in Appendix B of Part 1.

Learning to Program in Structured COBOL Part 1

Ed Yourdon
Chris Gane
Trish Sarson

8060981

CONTENTS
Part 1

ACKNOWLEDGMENT ix

FOREWORD x

1: MAKING THE COMPUTER DO WHAT YOU WANT 1

1.1 Clerks, computers, compilers, and COBOL 1
1.2 Coding the data for the computer 4
1.3 Getting the data printed out 7

2: PROCESSES, DECISIONS, AND LOOPS 12

2.1 Moving data from field to field 12
2.2 Initializing fields 15
2.3 Qualified data-names 16
2.4 Literals 17
2.5 MOVEing numbers 18
2.6 Program logic: the IF statement 21
2.7 Repeating blocks of code with loop structures 24
2.8 Structuring a sample program 28
2.9 Coding a sample program (SAMPLE-2) 30

3: DEFINING DATA FOR THE COMPUTER 34

3.1 The COBOL character set 34
3.2 Group items and elementary items 35
3.3 File Definition 36
3.4 Names 39
3.4.1 Naming the same data in different places 41
3.5 Punctuation, layout, and comments 42
3.5.1 Avoiding confusion with handwriting 43
3.5.2 Comments and continuation 44
3.6 FILE SECTION and WORKING-STORAGE SECTION 45
3.7 Initializing Working-Storage 48
3.7.1 Initializing constants with VALUE 48
3.7.2 Initializing flags, counters, and other fields 49

试读结束，需要全本PDF请购买 www.ertongbook.com

3.8 Condition-names 50
Review Quiz 52
3.9 Designing, structuring, and coding a sample program 53
3.10 Getting the program to run 71
3.10.1 Compiler diagnostics 72
3.10.2 Diagnosing diagnostics 74
3.10.3 Executing the program 75

4: DOING ARITHMETIC **76**

4.1 Arithmetic statements 76
4.1.1 ADD and SUBTRACT 76
4.1.2 MULTIPLY and DIVIDE 77
4.1.3 Rounding 79
4.1.4 COMPUTE 80
4.1.5 Dealing with result fields that are too small 82
4.1.6 Defining signed variables 82
Review Quiz 84

5: DEVELOPING PROGRAMS **85**

5.1 Using the Source Statement Library 85
5.1.1 The COPY statement 85
5.1.2 COPY REPLACING 87
5.2 Steps to follow in developing a program 88
5.2.1 Specifying test data 90
5.2.2 Program checks 91
5.2.3 Simple debugging 92
5.2.4 User acceptance 94
5.3 Enhancing SAMPLE-3 95

6: PROGRAM LOGIC **107**

6.1 Testing for conditions 107
6.1.1 Relational tests 107
6.1.2 Class tests 108
6.1.3 Condition-name tests 109
6.1.4 Sign tests 111
6.1.5 Complex tests 111
6.2 Nested IF statements 114
6.2.1 Block structures 120
6.2.2 The CASE structure 125
6.2.3 Simplifying nested IFs 127

6.3 Decision tables 129
6.3.1 Applying decision tables and nested IFs to a problem 134

7: GETTING DATA INTO AND OUT OF THE COMPUTER 158

7.1 Magnetic tape 158
7.1.1 FILE-CONTROL paragraph for tape files 160
7.1.2 OPEN and CLOSE for tape files 160
7.1.3 READ and WRITE for tape files 161
7.1.4 Creating a tape file 161
7.2 Maintaining a sequential file on tape 180
7.2.1 Changes to a record 181
7.2.2 Additions 182
7.2.3 Deletions 182
7.3 Random access devices 200
7.3.1 Organization of randomly accessed files 203
7.4 Indexed file organization 203
7.4.1 Creating an indexed file 205
7.4.2 Randomly accessing an indexed file 209
7.4.3 Updating an indexed file, adding and deleting records 211
7.4.4 Other facilities with indexed files 216
7.4.5 Disadvantages of indexed files 216
7.5 Relative file organization 217
7.6 Controlling printer spacing and paging 219
7.7 Top-down implementation of a useful accounting system 221
7.7.1 Functional specifications 221
7.7.2 Systems design 225
7.7.3 File design 227
7.8 Building the system with a skeleton and successive versions 227

APPENDIX A: Reserved Words 233

APPENDIX B: Lesson Plans and Lecture Notes 237

APPENDIX C: Answers to Review Questions 253

Part 2

PREFACE 261

8: PROGRAMMING FOR CHANGE 263

8.1 Characteristics of a good COBOL program 263
8.2 Hierarchies and structure charts 266
8.3 Cohesion and coupling in modular systems 270
8.4 COBOL modules and connections 280
8.5 Program-to-program linkage 285
8.6 The CALL statement 287
8.7 The LINKAGE SECTION 289
8.8 Using CALL to build systems 292
8.9 Structured program design methodologies 296

9: MORE POWERFUL FACILITIES 302

9.1 Traditional flowcharts 302
9.2 Details of the PERFORM statement 307
9.3 The PERFORM-UNTIL statement 310
9.4 CASE structures with GO TO DEPENDING ON 315
9.5 Literals, and why you shouldn't use them 319
9.6 The ALTER statement 321
9.7 MOVE, ADD, and SUBTRACT CORRESPONDING 323
9.8 The COMPUTE statement 326
9.9 The INSPECT/EXAMINE statement 329
9.10 The STRING/UNSTRING statement 339
9.11 The ACCEPT/DISPLAY statement 340

9.12 The STOP RUN statement 341

10: INTERNAL CODING AND THE DATA DIVISION 342

10.1 Representing numbers with ones and zeroes 342
10.2 Representing characters: USAGE DISPLAY, COMP, COMP-3 343
10.3 Alignment of fields: SYNC, JUSTIFIED 348
10.4 Negative numbers and the SIGN clause 350
10.5 Editing fields with PIC 353
10.6 Figurative constants 362
10.7 The RENAMES clause 363
10.8 The REDEFINES clause 365
10.9 Example 367

11: USING TABLES 370

11.1 Introduction 370
11.2 Defining related data items 370
11.3 Defining tables 374
11.4 Indexes and the SET statement 378
11.5 Sequential searches 379
11.6 The SEARCH and SEARCH ALL statements 382
11.7 Multidimensional tables 384
11.8 Variable length tables 387
11.9 Precautions with tables 388

12: ADVANCED INPUT-OUTPUT TECHNIQUES 390

12.1 IDENTIFICATION DIVISION options 390
12.2 ENVIRONMENT DIVISION options 391
12.3 Options of the OPEN and CLOSE statements 392
12.4 Buffering and blocking 394
12.5 Indexed input-output 397
12.6 Relative input-output 399
12.7 File status 400
12.8 The USE statement and DECLARATIVES section 403

12.9 Introduction to Report Writer and Data Communications 404
12.10 Benefits of an operating system 405

13: SORTING AND MERGING **407**

13.1 Introduction 407
13.2 The SORT statement 407
13.3 The SORT description (SD) 409
13.4 INPUT and OUTPUT PROCEDURES 409
13.5 The MERGE statement 414

14: TESTING AND DEBUGGING **415**

14.1 Introduction to testing 415
14.2 Walkthroughs 416
14.3 Top-down testing 418
14.4 Common bugs 422
14.5 Debugging strategies and techniques 427

15: EFFICIENCY AND OPTIMIZATION **431**

15.1 Introduction 431
15.2 Strategy for optimization 433
15.3 Measuring the inefficiency in your program 435
15.4 Programming techniques for efficiency 437
15.4.1 Avoid Unnecessary Internal Data Conversions 438
15.4.2 Organize Searches Efficiently 439
15.4.3 Organize IF-ELSE-IF Constructs Efficiently 441
15.4.4 Arrange Blocking and Buffering for Efficiency 442
15.4.5 Change CALL statements to PERFORM statements 443

AFTERWORD **445**

APPENDIX **447**

GLOSSARY **457**

INDEX **467**

8060981

1 Making the Computer Do What You Want

1.1 Clerks, computers, compilers, and COBOL

You probably have heard a lot about computers before picking up this book. Some of it may be alarming — for example, how computers are invading and taking over our lives. Some of it may be optimistic, as in the predictions of computers doing all of the boring work, leaving people a life of ease and leisure. Neither of these statements is true, of course, and by the end of the book we hope you will be in a position to make up your own mind about the meaning of computers (from a position of strength), because *you* will be giving the orders.

That is what being a programmer is all about: giving the orders to computers. Think of the computer as a clerk without any common sense, and think of yourself as the clerk's boss. Whatever you tell the clerk to do, he will do *exactly* that, incredibly fast, all the time drawing on a vast memory of what you and others have told him in the past. But, if you tell the computer to send out a check for $100,000 when you mean only $100, the computer will blindly obey and pay the $100,000.

The key requirement of your job as a programmer is to understand in practical terms what work people need done by the computer, and then to translate exactly those needs into code the computer can read and obey. Computers work by streams of coded electronic pulses, which we shall discuss in detail later in the book. Since these pulses, of course, are meaningless to humans, a variety of computer *language translators* have been developed, to transform commands in an English-like language

试读结束，需要全本PDF请购买 www.ertongbook.com