



Comprehensive Structured COBOL

James Bradley

Comprehensive Structured COBOL



James Bradley

University of Calgary



Mitchell McGRAW-HILL

New York St. Louis San Francisco Auckland Bogotá Caracas
Hamburg Lisbon London Madrid Mexico Milan Montreal
New Delhi Oklahoma City Paris San Juan São Paulo
Singapore Sydney Tokyo Toronto Watsonville

Comprehensive Structured COBOL

Copyright © 1990 by Mitchell McGRAW-HILL. All rights reserved. Printed in the United States of America. Except as permitted under the United States Copyright Act of 1976, no part of this publication may be reproduced or distributed in any form or by any means, or stored in a data base or retrieval system, without the prior written permission of the publisher.

1 2 3 4 5 6 7 8 9 0 WEB WEB 9 3 2 1 0

ISBN 0-07-007078-4

This book was set in Syntax and Berkeley Old Style by TBH/Typecast. The editors were Karen M. Jackson and Raleigh Wilson; the designer was Randall Goodall, Seventeenth Street Studios; drawings were done by Joseph Maas, Advanced Presentations. Development and production management were provided by Cole and Associates. Webcrafters was the printer and binder.

Library of Congress Cataloging-in-Publication Data

Bradley, James, 1942-

Comprehensive structured COBOL / James Bradley.

p. cm.

ISBN 0-07-007078-4

1. COBOL (Computer program language) 2. Structured programming.

I. Title.

QA76.73.C25B73 1990
005.13'3--dc20

89-13743
CIP

PREFACE

COBOL REMAINS A MOST important programming language, with more than 75 percent of business application programs still being written in COBOL. COBOL does not, however, stand still, but continues to develop in light of experience and advances in programming theory. COBOL 85 compilers are now commonplace on both mainframes and microcomputers. While this book teaches COBOL 85 (COBOL 85 facilities are clearly marked throughout the book), it also provides a complete presentation of COBOL 74.

The major theme of *Comprehensive Structured COBOL* is that learning COBOL and the ideas associated with it should be both challenging and interesting. *Comprehensive Structured COBOL* emphasizes understanding COBOL and programming concepts as the key to becoming a successful COBOL programmer. Thus, the amount of explanatory material in the text is greater than would normally be expected, and many examples are given to strengthen the concepts and techniques presented. The student is taught to understand, in a commonsense manner, the “why” of COBOL concepts and programming techniques.

Explanations and analogies used in the book have been verified for complete accuracy. Having been an instructor in computer programming for more than 15 years and having been involved in computer-related research and commercial programming for some 20 years, I have experienced the ideas inherent in COBOL 85 from many different viewpoints.

This book was written with the goal of making it easy to read and understand and with the hope that the ideas will leap off the page at you. I have been encouraged by reviewers who are of the opinion that this writing goal has been successfully met. However, *Comprehensive Structured COBOL* does not neglect detail. The book includes a level of COBOL detail that should make learning COBOL an achievable goal of every student.

In addition to the movement from COBOL 74 to COBOL 85, exciting as that is, another transition is taking place. Computer file processing, the original domain of COBOL applications programming, is moving to relational data base processing using COBOL with embedded SQL.

Every student of COBOL must learn the difference between file processing and relational data base processing and must develop the ability to write COBOL/SQL programs for processing relational data bases. Relational data base systems that can interface with COBOL are now commonly available, and systems for use with supermini computers and workstations can be purchased at a nominal cost. For example, at the University of Calgary, where the relational system ORACLE runs on supermini computers, workstations, and personal computers, the mainstream COBOL 85 course includes assignments on the use of COBOL with SQL for data base processing. That relational data base processing using COBOL with embedded SQL will become a strong component of COBOL courses in the 1990s seems clear, and *Comprehensive Structured COBOL* contains the necessary COBOL/SQL material.

Organization and presentation

The book is organized into three sections:

I. COBOL Foundations

The first section deals with the foundations of COBOL and presents the divisions of COBOL programs, as well as the use of IF-statements and looping with the PERFORM verb. The first chapter assumes that the student has no prior programming experience and explains computer processing and elementary data processing concepts, together with elementary COBOL programming concepts. The student with programming experience can begin either in Chapter 1 with COBOL programming concepts or with Chapter 2.

Chapters 2 through 4 explain the common considerations in writing the four divisions of a COBOL program. In addition, Chapter 4, which deals with the PROCEDURE DIVISION, details the use of the common COBOL verbs. Many program examples are used throughout these three chapters to illustrate the concepts and COBOL constructs presented.

The IF-statement and the new EVALUATE statement are given an entire chapter (Chapter 5), with particular attention paid to understanding conditions, both simple and compound, and to nested IF-statements. Chapter 6 is devoted to looping, including nested loops, using the PERFORM . . . UNTIL and PERFORM . . . TIMES verbs. Structured programming is used throughout the first section, and principles are explained gradually at appropriate points in the text.

II. COBOL Processing Methods

The second section is devoted to COBOL programming methods. I believe that the order chosen is the best one, although it could easily vary

according to the wishes of the instructor or experiences of the students. The section begins with a chapter on printing simple reports, followed by a chapter on validation of input data. Then comes a chapter on the COBOL SORT verb as a preliminary to the next chapter, which presents the all-important topic of control-break processing for generating more complex reports. The final two chapters of the section cover programming techniques for single- and multiple-level arrays.

With the possible exception of the use of the Report Writer (in Chapter 20 in the third section), material presented in the first two sections is generally covered in many extensive one-semester introductory COBOL courses.

III. COBOL for File and Data Base Processing

The final section teaches file and data base processing using COBOL. Chapter 13 covers sequential file processing and file updating techniques in detail; Chapter 14 details indexed sequential files. VSAM is used in programming examples, although provision is made for the use of ISAM. Coverage of secondary key processing is also included. The elements of relative files are covered in Chapter 15. Students are taught the three different types of relative files, including hash files, that can be used. Coverage of the concept of hash files extends as far as chained progressive overflow for handling collisions. Full coverage of hash files with chained progressive overflow will be included in a forthcoming advanced COBOL book.

Relational data bases and SQL are covered in Chapters 16 and 17. Chapter 16 covers essential relational data base and SQL concepts, and Chapter 17 shows how COBOL and SQL together can be used to process a relation (or "file") in a relational data base.

Chapter 18 covers on-line processing using COBOL and CICS. Many instructors include CICS in a COBOL course because of its importance in business programming. The chapter material is sufficient for an explanation of the basic concepts involved and for some introductory COBOL/CICS programming. However, if you intend doing extensive CICS programming, you will need a supplementary CICS text.

Chapter 19 covers some remaining peripheral topics, including CALL and COPY statements, the USAGE clause, and STRING and UNSTRING statements. Chapter 20 covers the use of the COBOL Report Writer feature. The chapter includes basic Report Writer concepts and instructions for generating reports using programs that utilize this feature.

Supplementary material

An effective course is dependent on a team effort involving the instructor and students, as well as the teaching and learning materials they both share. The complexities of teaching COBOL to a varied student audience gives rise to the need for a truly useful instructional support package to assist the process. For this reason, I have spent considerable time talking to COBOL instructors about their course needs to determine what combination of

teaching materials would be most useful. With these conversations as a backdrop, I have developed a comprehensive instructor's manual to accompany *Comprehensive Structured COBOL* that satisfies a broad cross-section of course needs.

Included for each chapter are

- a chapter guide
- topics for classroom discussion
- solutions to chapter questions
- solutions to chapter programming assignments
- complete examination materials containing true/false, multiple-choice, fill-in, and debugging questions, and questions requiring students to write COBOL program excerpts

Also in the instructor's manual is a complete set of overhead transparency masters with art from the text and both full programs and program excerpts.

To bring COBOL into the lab, there is an accompanying data disk in ASCII for convenient up- or downloading of files in either mainframe or micro environments. Utilizing this data disk enables students to learn while doing, without the drudgery of having to rekey the assignments in the text.

Acknowledgments

The author is grateful for the hard work of the following people who reviewed several versions of the manuscript and contributed many constructive comments: Eileen Bobman, Spring Garden College, Philadelphia; Lawrence E. Jerald, Southern Illinois University, Carbondale; Dr. Mo Adam Mahmood, University of Texas, El Paso; Jeretta Horn Nord, Oklahoma State University, Stillwater; Karen Schnepf, Metro Community College, Omaha; Dr. S. Srinivasan, University of Louisville, Louisville; and Michael M. Werner, Wentworth Institute, Boston.

Dennis L. Varin, of Southern Oregon State University, Ashland, must be thanked for independently running and checking all the programs used in the text.

I would also like to warmly acknowledge my editors at Cole and Associates, Annette Gooch and Brete Harrison, and at McGraw-Hill, Karen Jackson and Raleigh Wilson, for their patience and commitment; the book designers, Seventeenth Street Studios, for a design that contributes significantly to the book as a learning tool; and my production editor at Cole, Lorna Cunkle, for her untiring efforts in bringing it to fruition. Developing a good book is a team effort; to those at Cole and McGraw-Hill who brought a successful team together, thank you.

Another essential aspect of this team effort is the continuing dialogue with adopters of a book, whose comments can be valuable in developing subsequent editions. For this reason, I'm always open to ideas and suggestions for improving the book. Please write to the author (Computer Science Department, University of Calgary, Calgary, Alberta, Canada T2N 1N4). Your contributions, if incorporated, will be acknowledged.

James Bradley
January, 1990

The following acknowledgment has been reproduced from COBOL Edition, U.S. Department of Defense, at the request of the Conference on Data Systems Languages.

Any organization interested in reproducing the COBOL report and specifications in whole or in part, using ideas taken from this report as the basis for an instruction manual or for any other purpose is free to do so. However, all such organizations are requested to reproduce this section as part of the introduction to the document. Those using a short passage, as in a book review, are requested to mention "COBOL" in acknowledgment of the source, but need not quote this entire section.

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.

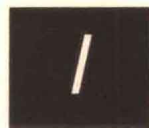
No warranty, expressed or implied, is made by any contributor or by the COBOL Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor or by the committee, in connection therewith.

Procedures have been established for the maintenance of COBOL. Inquiries concerning the procedures for proposing changes should be directed to the Executive Committee of the Conference on Data Systems Languages.

The authors and copyright holders of the copyrighted material used herein—FLOW-MATIC (Trademark of Sperry Rand Corporation), Programming for the Univac (R) I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form No. F28-8013, 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.

CONTENTS

1 *Elementary COBOL data processing concepts, 2*



COBOL FOUNDATIONS

- 1.1 Business data processing puts data into a more useful form, 3
 - Data for computer processing are stored on disk or tape, 3
 - Both instructions and data can be recorded in computer memory, 6
 - Instructions for computer processing can be written in COBOL, 6
 - There is a standard sequence for preparing and executing a COBOL program, 7
 - A computer has memory, input and output devices, and a processor, 8
 - COBOL goes back many years, 10
- 1.2 A simple example illustrates COBOL program development, 11
 - You need to structure COBOL programs carefully, 11
 - A pseudocode program is often written first, 11
 - Data are stored as computer files, 12
 - Working storage is a part of memory, 12
 - A detailed pseudocode program helps, 14
 - Most names in a program represent locations in memory, 15
 - The core of a COBOL program contains instructions, 16
 - Watch what happens as each instruction executes, 17
 - Space must be specified for items of data, 19

² **The IDENTIFICATION and ENVIRONMENT divisions, 44**

A complete COBOL program includes definitions, 20
COBOL programs have several divisions, 21
Record layouts and printer spacing charts are indispensable, 21
Value clauses are used to place initial values in memory locations, 26
A and B margins must be observed with COBOL programs, 26
Some files have labels, 27
You may need to change the SELECT statements, 28
Names of memory locations are called identifiers, 28

1.3 Diagrams can help you write correct programs, 28

You can draw diagrams of programs, 29
Flowcharts are used to display structure and logic, 29
File navigation diagrams relate input and output records, 30

1.4 Programs have to be tested thoroughly, 30

You can play computer: desk checking, 31
You must detect and correct errors, 34
Extensive testing of programs is often necessary, 36
You can always improve a program, 36

1.5 You have learned some COBOL instructions, 37

2.1 The IDENTIFICATION DIVISION identifies a program, 47

The basic syntax for the IDENTIFICATION DIVISION is brief, 47
There is also a longer syntax for the IDENTIFICATION DIVISION, 48

2.2 The ENVIRONMENT DIVISION links files to hardware, 48

Essential ENVIRONMENT DIVISION coding involves the SELECT statement, 49
The SELECT statement has a simple syntax, 49
There are several types of physical files, 51
SELECT implementor names depend on the computer system, 51
IBM has its own system for SELECT implementor names, 52
You can still make common errors with SELECT, 54

2.3 A program development example, 54

Input files do not have spaces between fields, 54
The input file has four fields, 56
Decimal points are not stored on disk or tape, 56
Decimal point markers do not print, 56
Edit symbols are often printed, 56
You cannot use identifiers with edit symbols in arithmetic operations, 58
An output file usually has headers, 59
Make spacing charts for input and output data, 60
A program that processes one record after another in a simple loop, 60
You need AFTER ADVANCING with every WRITE, 61
You can use ADD, SUBTRACT, MULTIPLY and DIVIDE statements, 61

3

The COBOL data division, 68

3.1 You specify file details and buffer record identifiers in the FILE SECTION, 70

The FD entry has a file label entry, 73

What are file labels?, 74

You can create a file with a label, 74

The file labels you specify vary, 75

There are other useful FD entries, 75

Blocks of records increase efficiency, 76

Buffer record identifiers can be elementary or group, 77

Identifiers with contents that will be printed should be 133 characters, 78

Identifiers are used with both buffer record and working-storage locations, 79

The rules for specifying identifiers are the same for buffers and working storage, 79

When naming identifiers consider both rules and style conventions, 79

Every identifier must be defined with a level number, 80

You can collect identifiers into subsidiary group identifiers, 81

Use level numbers 01, 05, 10, 15 . . . , 82

You can collect 01 elementary identifiers into a group identifier, 82

PICTURE specifications give the size and data type of a memory location, 83

Alphanumeric identifiers cannot be used for arithmetic operations, 83

Data are left justified in alphanumeric identifiers, 84

Simple numeric specifications are used for identifiers participating in arithmetic operations, 85

3.2 The WORKING-STORAGE SECTION has definitions of elementary and group identifiers, 86

A VALUE clause initializes the value in an identifier, 87

Literals are simply values, 87

Literals are limited in size, 88

Large alphanumeric literals can span several lines, 88

Avoid extensive use of literals with MOVE and arithmetic statements, 89

You can also initialize with figurative constants, 89

You can also use INITIALIZE for initialization, 90

FILLER specifications are commonly used for specifying fields in output records, 90

A FILLER field can contain characters other than blanks, 91

Edited identifiers are used to print numeric quantities including non-numeric symbols, 92

An edited identifier cannot be used for arithmetic operations, 93

Edit items take up space in memory, 93

Edit identifiers are versatile, 93

There are two ways to specify plus and minus signs, 94

3.3 A program development example, 97

You can display the input file so it can be understood, 97

Processing complies with some simple business rules, 98

The output data is a printer report on the state of each account, 98

The data in input records are highly condensed, 99

4

The essentials of the PROCEDURE DIVISION, 112

The calculations involve simple interest, 100
Auxiliary identifiers are needed for calculations, 100
The calculations are repeated for each input record, 102
Use **ROUNDED** to have a computed result rounded off, 102
Intermediate results require sufficient decimal places, 102
Specify complex headers one item at a time using **FILLER** fields, 105

4.1 The PROCEDURE DIVISION is structured in paragraphs, 114

An instruction is called a statement, 118
Distinguish imperative and conditional statements, 118
Always use paragraphs, 119
Paragraphs are structured hierarchically, 119
A boxed comment should be used with each paragraph, 120
Hierarchical structure is the essence of structured programming, 120
The **PERFORM** statement is crucial for writing structured programs, 122
Numbering paragraphs helps you to find them, 124
Paragraphs can be grouped into sections, 124
PERFORM . . . UNTIL is used with repeated and conditional paragraph execution, 125
Be aware of how the condition affects the **PERFORM . . . UNTIL** statement, 126
THRU allows looped execution of several paragraphs, 127
THRU is sometimes used with a paragraph containing only **EXIT**, 128

4.2 The common non-computational COBOL verbs are MOVE, READ, WRITE, PERFORM, OPEN, and CLOSE, 129

The **OPEN** verb opens files, 129
The **CLOSE** verb closes files, 130
The **READ** verb makes the next input record available for processing, 131
The **WRITE** statement sends records to output, 132
Literals can be used with many statements, 134
Avoid literals in the PROCEDURE DIVISION, 135
The figurative constants **SPACES** and **ZEROS** are useful, 135
There is more to the **MOVE** statement than you might think, 135
A group **MOVE**, with group identifiers, is alphanumeric, 138
A table is useful for checking if a **MOVE** is allowed, 140
Recall numeric identifiers with the **SIGN LEADING SEPARATE** attribute, 140
Avoid ambiguous identifiers, 142
A multiple (“shotgun”) **MOVE** is allowed, 142
There is a special **MOVE** called **MOVE CORRESPONDING**, 142
You can use **ACCEPT** and **DISPLAY** for input and output of small amounts of data, 143

4.3 COMPUTE is mainly used with more complex computations, 144

The **COMPUTE** verb is powerful and flexible, 144
Avoid the need to use **ON SIZE ERROR**, 145

Avoid complex COMPUTE expressions, 146
Use parentheses with COMPUTE expressions, 147
Use ROUNDED to have a result rounded, 148
Subtle truncation errors are possible with arithmetic operations, 149

4.4 ADD, SUBTRACT, MULTIPLY, and DIVIDE are used with simple computations, 150

It is better not to use edit identifiers in the GIVING clause, 156
You can use individual scope terminators with arithmetic verbs, 157
It is poor practice to use more than one identifier after GIVING, 157
COMPUTE and other arithmetic verbs are used in different situations, 157

5 Selection of alternatives with conditional statements, 166

5.1 The IF-statement selects between alternatives, 168

Learn the IF-statement syntax thoroughly, 170
Learn the exact semantics for the IF-statement, 172
IF-statements are useful for handling various possibilities, 172
A program using IF-statements, 173
Pseudocode IF-statements are useful in planning programs, 176

5.2 You must understand conditions thoroughly, 177

A simple condition compares two quantities, 177
Compound conditions are more complex, 178
There are some special conditions you must know about, 181
Sign conditions can be handy, 182
Class conditions are useful for validating input data, 182
Condition names permit flexibility, 183

5.3 Nested IF-statements are used for multiple alternatives, 184

Learn the semantics of nested IF-statements, 184
A decision table should be used with every nested IF-statement, 190
A program example using nested IFs, 192
Other scope terminators are sometimes needed within an IF-statement, 194

5.4 The EVALUATE statement can be used with multiple alternatives, 195

The EVALUATE verb has two formats, 196
A program example using EVALUATE, 201

6 Looped instruction sequences, 210

6.1 A PERFORM . . . UNTIL loop is commonly used to process files, 211

Watch the position of READ with looped processing, 212
Looped sequences are very common in COBOL programs, 213
The PERFORM . . . UNTIL statement uses the same type of condition as IF-statements, 215
Pseudocode is often used when planning looped sequences, 216

Compound conditions are often used with PERFORM . . . UNTIL, 216
Creating an infinite loop is easier than you might think, 217

6.2 The PERFORM . . . TIMES statement is used with a fixed number of iterations, 219

6.3 Nested loops are important, 225

6.4 The in-line PERFORM can be convenient, 233
There is a PERFORM with TEST AFTER option, 234



COBOL PROCESSING METHODS

7 Printing reports, 244

7.1 There are different types of reports, 246
Detail reports have a line for each input record, 247
Exception reports cut down on output, 247
Summary reports have one line for many input records, 248
The parts of a report have names, 248

7.2 Edit symbols are used extensively with numeric output, 249
Decimal points need edit symbols to be printed, 250
Commas also require edit symbols, 250
There is an edit symbol for the dollar sign (\$), 250
The Z symbol suppresses leading zeros, 251
The check protection asterisk (*) is an edit symbol, 251
The plus (+) and minus (-) edit symbols can be tricky, 252
Credit (CR) and debit (DB) work like the minus symbol, 253
The blank edit symbol (B) puts blanks between digits, 253
The slash symbol (/) can be used in dates, 253
The zero symbol (0) is handy with megadollar quantities, 254
Watch out for the floating symbols (\$, +, -), 254
You can edit non-numeric pictures with B and 0 symbols, 255
The group MOVE is an alphanumeric MOVE, 255
Use QUOTE to print quotation marks, 255
The justified right option is handy, 256

7.3 Special COBOL facilities are needed for the current date, 256
There is an ANS method for extracting the date, 256
You can use REDEFINES with the ANS method of extracting the date, 258
You can MOVE CORRESPONDING to extract the date, 260
There is an easy IBM enhancement for extracting the date, 260

7.4 Printing headers is detailed but straightforward, 261
The WRITE . . . ADVANCING statement is the key to printing headers, 263

Be clear about WRITE . . . AFTER ADVANCING semantics, 264
WRITE . . . BEFORE ADVANCING semantics are important too, 264
Avoid mixing WRITE . . . BEFORE and WRITE . . . AFTER, 265
WRITE . . . PAGE is used to control printing of pages, 265

7.5 COBOL report program examples, 267
Use printer spacing charts to lay out reports, 267
A program for a detail report, 268
There is a convention for organizing paragraphs, 274
Simple summary report program, 275

8 *Validation of input data, 290*

8.1 Incorrect input data can often be detected, 292
Use a checklist for input error detection when writing a program, 292
Use the class test for detecting incorrect data type, 294
Use the sign test for incorrectly signed data, 294
Check for blanks as a missing data test, 295
A reasonableness test involves checking ranges and limits, 295
Bad data are repaired by counting and replacing characters, 296
Condition codes must be checked, 299
Checking for correct sequence is often necessary, 300
Check digit computations help detect errors in numeric fields, 302

8.2 A program can react to input errors in different ways, 303
Use a checklist of possible program reactions to input errors when
writing a program, 303
Good error messages make the operator's life easier, 304

8.3 Master files are created with built-in validation, 304
Detecting errors is the basis of master file creation, 304
Validity checks account for the complexity of master file creation
programs, 307

9 *Sorting and merging, 318*

There are two ways to sort a file, 320
9.1 The COBOL SORT verb is used to sort a file, 320
The straight sort is a simple use of the SORT verb, 321
The SORT verb can be used with an input procedure, 326
SORT can be used with an output procedure, 333
SORT can be used with both input and output procedures, 338
The order of duplicates can be specified with SORT, 339

9.2 Merging files requires use of either SORT or MERGE verbs, 339
The SORT verb is more flexible for merging, 340
There are file format restrictions for merging with SORT, 343
The MERGE verb is less flexible for merging, 344
There are also file restrictions for merging with MERGE, 348
COBOL can process a file containing records with different formats or
variable length records, 349

10 ***Control-break processing, 354***

10.1 Single-level control-break processing is used with summary reports, 355

A control break generates a summary line, 356

When a control field changes, a summary line is printed, 358

End-of-file processing involves forcing a final control break, 360

10.2 A single-level control-break program, 363

Program for simple control-break processing, 363

Coherence and independence concepts are used with COBOL paragraphs, 370

10.3 Two-level control-break processing is used for reports with two summary levels, 372

Identify the primary and secondary sort fields, 372

Major and minor control fields trigger control breaks, 373

With two-level control-break logic, the major control field is tested first, 376

10.4 A two-level control-break program, 381

COBOL program for two-level control-break processing, 382

Higher-level control-break processing is common, 387

10.5 Debugging techniques with complex programs need to be mastered, 388

The READY TRACE command lists executed paragraph names, 389

DISPLAY outputs contents of one or more identifiers, 390

EXHIBIT functions similarly to DISPLAY, 392

11 ***Single-level arrays, 398***

11.1 The array concept is illustrated by repeated processing of a short input file, 401

You use OCCURS to define an array of identifiers, 401

A subscript distinguishes the different identifiers of an array, 402

You put data into an element by using the subscript, 402

Elements of an array are processed one by one using the subscript, 403

Subscripts can be used with elementary identifiers of an array of group identifiers, 404

COBOL program using an array for repeated processing of a short input file, 404

The PERFORM . . . VARYING verb is used to scan an array, 409

To use PERFORM . . . VARYING you must know how many array elements to process, 410

An in-line PERFORM . . . VARYING is also available, 411

Watch out for a wrong UNTIL condition when scanning the elements of an array, 412

11.2 There are array technicalities to be mastered, 413

There are rules for specifying arrays, 413

There are several ways to initialize an array, 415
There are rules for referring to subscripted identifiers, 417

11.3 An array can be used for histogram generation, 418
Histograms and arrays go together, 418
Histogram development involves a specific technique, 419
COBOL program for developing a histogram, 421
There is a problem with a zero subscript, 423

11.4 An array is used with repeating groups of fields, 425
Move the repeating groups into an array for manipulation, 426
A program for manipulation of repeating groups, 429

11.5 Arrays can be used as look-up tables, 433
Example of a look-up table for product data, 433
A look-up table has to be initialized with data in a specific order, 436
You can use either linear or binary searches, 438
You can use PERFORM . . . VARYING for a linear table search, 440
The SEARCH verb is used for a linear search of an array, 441
An index identifier contains a memory address offset value, 444
You must use SET to manipulate an index value, 445
The differences between an index identifier and a subscript identifier are very important, 446
You can also use an index identifier with PERFORM . . . VARYING, 447
You use SEARCH ALL for a binary search of a table, 448
You must define a search key and an index to use SEARCH ALL, 449

12 MULTIPLE-LEVEL ARRAYS, 460

12.1 The technicalities of two-level arrays must be learned, 461
Two subscripts are used with a two-dimensional array, 463

12.2 Accumulating totals is a common two-dimensional array activity, 464
First you initialize a two-dimensional array (step 1), 466
Then accumulate totals in the two-dimensional array (step 2), 468
Then print contents of the two-dimensional array as a report (step 3), 468
Then generate column totals from the two-dimensional array (step 4), 470
Finally sum the entire two-dimensional array (step 5), 471
A complete program with a two-dimensional array, 472

12.3 Two-level arrays sometimes have to be searched, 476
You can search a two-level array using PERFORM . . . VARYING, 477
You can search a two-level array using both PERFORM . . . VARYING and SEARCH, 478
Sometimes you retrieve just the position of a target element, 479

12.4 Three-level arrays are needed too, 480