High Level COBOL Programming

Gerald M. Weinberg
Ethnotech, Inc.

Stephen E. Wright
Richard Kauffman
Martin A. Goetz
Applied Data Research, Inc.

Winthrop Publishers, Inc.
Cambridge, Massachusetts

Library of Congress Cataloging in Publication Data
Main entry under title:

High level COBOL programming.

(Winthrop computer systems series)
Includes bibliographical references.
1. COBOL (Computer program language) I. Weinberg, Gerald M.
QA76.73.C25H53 001.6'424 77-599
ISBN 0-87626-329-5

17 Dunster Street, Cambridge, Massachusetts 02138

10 9 8 7 6 5 4 3 2

High Level COBOL Programming

Preface

SHOULD YOU BUY THIS BOOK?

If you are reading this preface, chances are you are trying to decide whether this book is worth reading, or buying, or using as a textbook. We shall try to give you the information you need to make that decision.

The title is of some help. COBOL means this book is aimed at the community of people who solve business problems using COBOL in some way. "High level" means that it is not another introduction to COBOL. If you have never written a COBOL program, or managed the writing of one, this book is not for you—at least, not yet. Take an introductory COBOL course, or read one of the hundreds of introductory texts; after that, we shall be most pleased to have you back.

The one other thing you need to know about this book is captured in one phrase used above—"solve business problems." Though we have perhaps lost sight of it in the muck and mire of day-to-day programming, what computers are really useful for is *solving business problems*. The computer, the COBOL language, and each individual program are merely *means* we employ to solve business problems. Unhappily, they often seem to be a means of *creating* business problems.

If you are attempting to solve business problems with the aid of computers and, more especially, COBOL, this book is for you. If you are, in the process, creating more business problems than it seems to be worth, this book is *especially* for you. If you are a student, hoping to someday solve business problems with the aid of computers and COBOL, this book is for you, too—though you may not appreciate the full nature of the problems that it addresses. If you have a good teacher, he or she will be able to fill you in about why such a book is necessary—that is, why there is more to solving business problems than writing COBOL programs.

That's really all there is to deciding whether to use this book, other than taking a look at the way it is written to see whether you like our approach. Inside, you'll find that we are not trying to tell you *what to think*, but are giving you some ideas about *how to think* when you are trying to solve business problems with COBOL. We're also trying to show you what to think *about*. We'll direct your attention to some factors you may have been excluding from your thinking about programming. As we do this, we'll draw upon thoughts that others have had and put into practice with considerable success. No doubt you may have already tried some of them and can learn from your own experiences. Once you've added what we have to say to your own store of experience, you may find that you want to change the way you go

about solving business problems. That's what we hope, and we'd like to hear from you as it happens.

HOW TO USE THIS BOOK

This book has an unusual organization—you must start reading at the front and continue page by page to the back!

While this structure initially may not seem particularly curious, we have to warn those nontechnical people who might be tempted to crack the book somewhere in the middle and quickly decide it is not for them. This book is for *all* people involved in the development of COBOL programs—from top management on down. Although not all people will read all the way through from front to back, everyone does need to start in the front. As the going gets more technical, some may drop out along the way—but we hope not before they learn that there is an *inescapable technical dimension* to managing successful COBOL programming.

Conversely, there is an *inescapable business dimension* to writing successful COBOL programs—so the technical people should not skip the first part of the book. In a way, the entire book is about the *interweaving of technical and business questions*. Managers must be willing to bear with us through some demonstrations of why their best-laid plans can be sabotaged by poor attention to technical details. On the other hand, programmer/analyst types are going to have to learn more about how their work fits into a larger world than MOVE CORRESPONDING or GO TO DEPENDING ON.

Everyone, therefore, will benefit from reading this book as we have said—from the front to the back. They might benefit even more if they get together and discuss what they have read.

On the other hand, we have no illusions about the importance of our book in the eyes of the typical reader. Ou business has always been characterized by "busyness," so we have added a few features for the harried programmer or executive. First of all, we have provided a generous diet of subtitles to assist the reader in skipping over parts of no immediate interest. The subtitles will also help the reader who is interested, but cannot spend enough time at one stretch to read an entire chapter, or even a section. Usually, the material under one subtitle is readable on its own, without strong reference to the preceding material.

Second, we have enclosed some of the most technical material in *boxes* to indicate that it can be skipped without any real loss of continuity—unless you are interested in the technical points in themselves. Skipping the boxes should shorten the reading time and raise the interest level for the nontechnical reader. Even the technical reader may choose to skip the boxes on the first pass.

Third, the organization of the book should make it easy to refer back to some important point. The Contents is a good guide, but don't

forget the Index in the back. Also in the back are some appendixes containing particularly technical material for those who want to go one level deeper into technical matters.

ON THE USE OF ADR SOFTWARE PRODUCTS

In order to demonstrate the effective interweaving of technical and business dimensions in the development of COBOL programs, we need to discuss the role of software tools such as a source library system and a COBOL preprocessor. We particularly need a macropreprocessor to attain our higher level without worrying about burdensome details that might tax the patience of the less technical reader.

Because several of the authors are from Applied Data Research (ADR), MetaCOBOL® and The LIBRARIAN® are available to us, and have been reasonably well tried. We do not wish to imply that there are no other preprocessors or source library systems, nor that others are without merit. The choice of tools—made or purchased—is beyond the scope of this book, and should be left to the individual installation. What we have done is demonstrate the use of tools in ways that can reasonably be adapted to similar tools—or even (though with somewhat more clerical effort) to systems without such tools at all.

A COBOL macropreprocessor enables us to experiment with and evolve new logical structures and introduce them to the COBOL community without waiting five years before the next language revision. Because we use software tools, we separate out the purely clerical factors in changing to new structures, so we present our readers with a vision of what they may attain by pursuing these directions. We believe that these directions should be pursued, and that tools will help. That is the main emphasis of this book.

Because of our ADR orientation, there may seem to be another emphasis—to buy or license ADR products. Although we would be happy if that should happen, we must not let that pleasant possibility intrude on our main message. Consequently, we have attempted to minimize direct reference to ADR products by name, except in the appendixes that describe those products explicitly. In the text, we shall refer to "our preprocessor," "our development library system," and so forth. We have taken liberties with the precise syntax and other features of these ADR products whenever we felt that:

1. precision would interfere with reader comprehension; or
2. the particular features were not, perhaps, as we would like them to be.

In short, we hope that the use of ADR software products contributes to, rather than detracts from, the attainment of our major goal—which is to help our readers improve all aspects of *their* COBOL programming.

Acknowledgments

Material for this book has come from too many sources to name individually. Each of the COBOL examples—good and bad—has come from a real COBOL program from a real business, either an ADR user or an Ethnotech client, except where otherwise noted. We wish to thank all these anonymous programmers, as well as the hundreds of other programmers and managers with whom we have discussed the problems in this book. Essentially, all the ideas presented here are distilled from their practical experience at solving problems on the job.

In order to ensure the quality of the book, all the material has been submitted to extensive editorial review. Of particular help to us were the reviews of members of the ADR and Ethnotech technical staffs. The painstaking editorial work of Chris Gibbs, Mary Hutchinson, and Julie Wilson must be mentioned in a separate category—"above and beyond the call of duty." In Julie's case, an even stronger commendation is in order, perhaps, for risking her sanity in trying to index all this diverse material.

Special commendation also must go to Herb Nolan of Winthrop, who picked up the pieces upon the untimely departure of another member of the staff, and to Mike Meehan, whose vision inspired this project and kept it going in the darkest hours.

Gerald M. Weinberg
Stephen E. Wright
Richard Kauffman
Martin A. Goetz

Contents

Preface xvii

Should You Buy This Book • How to Use This Book • On the Use of ADR Software Products • Acknowledgments

A. THE PROGRAMMING BUSINESS 1

A.1. What Is High Level COBOL Programming? 2

Categorizing "High Level" • Programming as a High Level Responsibility • Equality in Programming Languages

A.2. What Is Programming? Why Is It Difficult? 5

Programming Is Foresight • Programming Is Writing the Recipe, Not Baking the Cake • The "If-so-then-so" Paradigm • The Programmer's Recipe Must Be

Unambiguous • The Programmer's Recipe Must Be Complete • The Programmer's Recipe Must Terminate • The Programmer's Recipe Must Be Widely Applicable • The Programmer's Recipe Must Be Verifiable • The Programmer's Recipe Must Be Created in a Controlled Way • Large Programs Are Mastered by Successive Refinement • The Programmer Has Many Tools, Some of Them Programs • Programming Is More than Communicating with Computers

A.3. Can Programming Be Treated in a Businesslike Manner? 9

The Big Switch to Small Computers • Some Indirect Programming Costs • Programs as Assets • Good Management in Programming Pays

A.4. Is a Businesslike Manner Enough? 12

Appearance Isn't Everything • Coding Isn't Everything • Titles Aren't Everything • Realizing the "High Level" Vision • Functions Are More Important than Definitions

Questions 16

For Managers • For Technical People

Supplementary Readings 17

B. MANAGING HIGH LEVEL PROGRAMMING 19

B.1. How Do You Get People to Produce Quality Programs? 20

The Right Thing for the Wrong Reason • The Wrong Thing for the Right Reason • Communicate, Not

Tell • Goals and Trade-offs • Better Never than Late • Haste Makes Waste

B.2. What Is the Value of Quality? 23

Little Errors Cost Big Money • Little Money Becomes Big Money • Disasters Become Rules • Rules Become Counterproductive

B.3. Monitoring Quality During Development 26

Finding Bugs Before They Eat Too Much Money • The Development Library • The Program Activity Reports • The Module History Report

Questions 29

For Managers • For Technical People

Supplementary Readings 30

C. CRITICAL PROGRAM READING 31

C.1. Case Study: SERVICING 32

Good Writing for Easier Reading • Paying Attention to Details • Programming Isn't Baloney • A Lot Can Be Learned from a Little • Writing with the Reader in Mind • Learning to Make Something out of Nothing • Looking at Literals

C.2. Abbreviation Through Shorthands 36

The Importance of Meaningful Names • Separating Reading and Writing • Installation-standard Shorthands • What We Have Accomplished • Box: Defining Shorthands for the Preprocessor

C.3. Case Study: Service Reporting 41

The Harm in the GO TO • What Clarity Buys • Symptoms of Poor Programming • Eliminating the Residue of Poor Programming

Questions 44

For Managers • For Technical People

Supplementary Readings 46

D. HIGH LEVEL CONTROL 47

D.1. Control Structure Through Macros 48

The Effect of Small Problems on Big Programs • The All-powerful Period • A Preprocessor IF-THEN-ELSE Structure • An Extensible IF-THEN-ELSE • Why Eliminate the GO TO? • Where Has the GO TO Gone?

D.2. The Process Concept 53

Bricks or Stones? • Building with Processes • Getting in "Structured" Trouble • Making Assertions As We Build

D.3. Eliminating the "GO TOO FAR" 56

Four Ways to Keep Control • Style in Control • Standards in Control • New Language in Control— SELECT EVERY • SELECT FIRST • The SELECT Postscript • Verifying the Control

D.4. Controlling the Source Code 61

Benefits of a Development Library System • The Changeover to a Development Library • Updating

the Source Library • Outputs from the Development Library • Box: Implementing Changes to a Source Module

Questions 65

For Managers • For Technical People

Supplementary Readings 67

E. STYLE AND WORKMANSHIP 69

E.1. Style Through Feedback 70

The Feedback Principle • "Backward" Thinking • Difficulty with Negation • DeMorgan's Law • Compound Conditionals • "Redundant" Parentheses • What Is "Natural"?

E.2. Style and Choice in Decision Structures 75

Levels of Programming Risk • SELECT LEADING ACTIONS • SELECT FOR • Editing with the IS FALSE Suffix

E.3. Which Switch to Ditch? 78

The Many Meanings of MOVE • A Standard Flag • Standards and Style

E.4. To Qualify or Not to Qualify? 81

The Effectiveness of Name Standards • Qualification versus Renaming • Prefixing by Preprocessor

E.5. Aids to Workmanship 84

Debugging versus Design • Methods of Controlling Typographical Errors • File and Data Definition Libraries • Box: COBOL COPY and the Preprocessor • Automatic Detection of Error-prone Forms • Box: Preprocessor Assistance for Stylistic Improvements

Questions 88

For Managers • For Technical People

Supplementary Readings 89

F. MODULARITY 91

F.1. The Module Concept—A Case in Point 92

Some Muddled Logic and Its Consequences • Indications of Poor Structure • A New Top Level Design • The Preprocessor DO • The Preprocessor LOOP • A Serious Error Arising from Poor Structure • The HEADER Module • Functional Division into Modules • Testing the Functional Division • The Too Small Module • Expansion Without Contamination • Prevention versus Cure

F.2. Modules and Modification 99

Testing a Design through Virtual Modifications • Reasonable and Unreasonable Modifications • Building Control Counts • Interspersing Batch Cards • Altering the Header • Vulnerability to Punching Errors

F.3. Creating a Modular Test Environment 104

Testing versus Production • Box: Explanation of Modular Test Run • Advantages of the Modular

Test Environment • Monitoring Modular Testing • The Transition to Production • Transition to Maintenance

Questions 107

For Managers • For Technical People

Supplementary Readings 109

G. REFINEMENT 111

G.1. Development Through Refinement 112

A Practice Problem—Counting Duplicates • The Top Level Conception—An Output-driven Program • Splitting the Problem • WRITE-NEXT-OUTPUT • GET-NEXT-OUTPUT • INITIAL-PROCESSING and FINAL-PROCESSING • Naming and Placing Flags

G.2. A Test Environment for Refinement 118

The Metamodule Concept • The Structure of the Metamodule • Stubs to Simulate Modules • Proceeding in Verifiable Steps • Box: The Structure of Two Stubs • What is Reliable Development?

G.3. Development Through Continued Refinement 123

What Each Level Must Accomplish • The Need for an Image of Lower Levels • NEXT-RECORD-TO-OUTPUT-AREA • Should We Improve It?

G.4. Backing Up and Going On 126

UPDATE-OUTPUT-AREA • Good Reasons for Backing Up • FLAG Conventions with Common

Subroutines • Completing the Refinement • Characteristics of Bottom Levels

Questions 131

For Managers • For Technical People

Supplementary Readings 132

H. VERIFICATION 133

H.1. Verifiable Assertions 134

A Small Number of Testable Assertions • Assertions Developed Top Down • An Ounce of Prevention or a Pound of Excuses • Assertion of General Properties • Exposing Ambiguous Specifications

H.2. Verification and Refinement in Parallel 137

Verifying the Replacement of Stubs • Determining the Input Set Needed for Verification • Simulating Files with Tables • What Happened to Debugging?

H.3. Verification With Generated Input Files 141

Eternal Vigilance • Verification with Feedback • Verifying Input Modules • A Data Generator Program • Box: Building a Test Data Generator Miniprogram

Questions 146

For Managers • For Technical People

Supplementary Readings 148

I. SHELTERING 149

I.1. Sequence Checking 150

To Do It or Not to Do It? • Where to Do It • How to Do It • A Question that Prevents Patching • Verifying the Modification • Can Bugs Be Found?

I.2. Internal Coordination of Attributes 155

Troubles Come Not Singly • Linking Related Attributes • Linking Procedural Code to Table Size • Eliminating Literals

I.3. Protecting the Interface With Closed Subroutines 159

The CALL—Solution or Problem? • Verbalizing the Interface • Advantages of the Preprocessor Interface

I.4. Counting for Control 162

Must Input Checking Always Be Sticky? • Impulses Can Be Dangerous • Impulses Can Be Useful • Inpulses Can Be Right • Impulses Must Be Verified

I.5. Defending Against Things That "Can't Happen" 166

A Sad, Funny, Special Story • Positive Programming • Tools Encourage Sheltering

Questions 168

For Managers • For Technical People