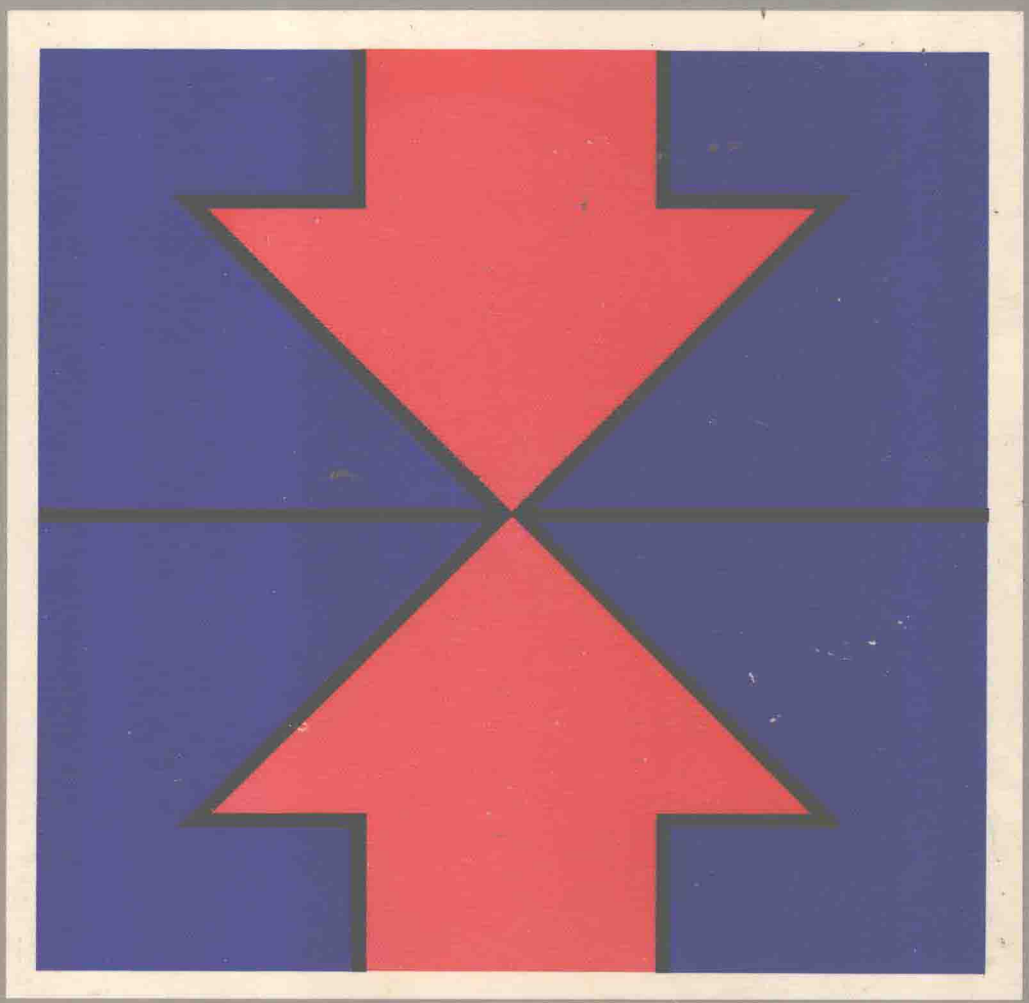
How to design and develop COBOL programs

A practical approach to design, coding, testing, and documentation



Paul Noll
Mike Murach

How to design and develop COBOL programs

A practical approach to design, coding, testing, and documentation

Paul Noll
Mike Murach

Mike Murach & Associates, Inc.
4222 West Alamos, Suite 101
Fresno, California 93711
(209) 275-3335

Development Team

Originator/author:	Paul Noll
Director/writer:	Mike Murach
Writer/editor:	Anne Prince
Production director:	Steve Ehlers
Cover design:	Michael Rogondino

Related program development products

A handbook called *The COBOL Programmer's Handbook*
Case Studies for *How to Design and Develop COBOL Programs*
Instructor's Guide for *How to Design and Develop COBOL Programs*
LISTMODS software for preparing structure listings from COBOL source programs

Related system development products

A textbook called *How to Design and Develop Business Systems*
Case Studies for *How to Design and Develop Business Systems*
Instructor's Guide for *How to Design and Develop Business Systems*

Related COBOL products

A textbook called *Structured ANS COBOL, Part 1*
A textbook called *Structured ANS COBOL, Part 2*
A textbook called *Report Writer*
Advisor's Guide for *Structured ANS COBOL*

Printed in the United States of America

20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

Library of Congress Catalog Card Number: 84-61556

ISBN: 0-911625-20-8

Contents

Section 4 Related Subjects

Appendixes

Preface

In 1977, we published a book by Paul Noll called *Structured Programming for the COBOL Programmer*. It presented a complete method of program development that included techniques for top-down design, structured coding, and top-down testing. Back then, we felt that this method would improve any programmer's productivity. We also felt that this method would help programmers improve the reliability and maintainability of their programs.

Since 1977, more than 35,000 people have bought this book in more than 12,000 different companies throughout the country and the world. In addition, as an independent consultant and trainer, Paul has taught his development method to more than 5,000 programmers in 100 different companies. Based on a survey that we conducted earlier this year, we estimate that Paul's complete method is now used in more than 3,000 companies. And ideas taken from his method are used in thousands of other companies.

Since 1978, when we installed our first computer system at Mike Murach & Associates, we have used the method presented in *Structured Programming* in our own COBOL shop. We have used it with four different kinds of compilers on four different kinds of computer systems. We have developed both batch and interactive systems with it. We have developed about one million lines of COBOL code with it. We have developed complete operational systems for four different companies with it. And now, we can say from experience that the program-development method in *Structured Programming* works...and it works better than any other method we've used, read about, or heard about.

What this book does

This book is a revised edition of *Structured Programming for the COBOL Programmer*. Its purpose is to show you how to improve the quality of your programs as well as your programming productivity. Whether or not you are using structured development techniques right now, we believe this book will show you a better way to develop COBOL programs.

If you check the contents of this book, you can see that it consists of four sections. In general, the first section presents background information that is designed to convince you that our method is the right one for you. The next two sections present the procedures and techniques that make up the method. And the fourth section presents related subjects that can help you make more effective use of the development method.

In section 1, you will find three chapters that present the status of structured programming as we see it today. In our opinion, structured programming has failed in most COBOL shops, and to a large extent it has failed due to the bewildering array of techniques and methods that have been promoted under the name of "structured programming." But that doesn't mean it has to fail in your shop. In fact, chapter 3 presents eight principles that can lead you to dramatic improvements in programmer productivity and program quality.

In section 2, you will learn about the nine-task procedure for program development that we recommend for any COBOL shop. Chapter 4 presents the nine-task procedure. And chapters 5 through 11 present the techniques you should use for each of the tasks in the procedure. To keep this section manageable, the examples used to present the techniques are all taken from batch programs.

Then, in section 3, you'll learn how to apply the procedures and techniques of section 2 to interactive programs. You'll also learn how interactive compilers can force you to modify the standards recommended in section 2. When you finish this section, you should be able to apply the method of this book to interactive programs using any COBOL compiler.

Finally, in section 4, you'll find seven chapters on subjects related to the method of this book. For instance, chapter 15 shows you how the 198X COBOL standards may affect the way you develop programs. And chapter 16 presents other common methods of structured program development so you can compare them with the one in this book. Overall, this section presents many ideas that you can use in conjunction with the development techniques recommended in this book.

For those of you who read the first edition of this book, I think you'll find four major areas of improvement in this second edition. First, the development method of the first edition is refined and improved; it is presented in section 2 of this book. Second, the interactive material in section 3 of this book is new material that we've had dozens of requests for over the last few years. Third, section 4 presents much new material that gives you more perspective on program development. And, fourth, throughout this book, we've made more of an effort to explain *why* our

method is more effective than some of the other methods you are likely to be using. As a result, we expect this edition of the book to be more convincing than the first edition was.

As you can see from the contents, this is a lengthy book. But it doesn't have to be overwhelming. Keep in mind, then, that section 2 (chapters 4 through 11) presents the essential method of the book, while the other chapters only provide supporting information. As a result, you can skim or skip section 1 (chapters 1 through 3). You don't have to read section 3 until you're ready to apply the techniques of section 2 to interactive programs. And you only have to read the chapters in section 4 if they become of interest to you. If you read the book with this perspective, I think you'll find it quite manageable.

How we developed this book

When we wrote the first edition of this book, Paul Noll was the originator of the material and I was its writer and editor. Unfortunately, though, Paul Noll wasn't able to help with the development of this second edition of the book because he is teaching at Academia Sinica in Chengdu, China. However, Paul and I did meet for three days last February to compare notes and to agree on what the revision would contain.

So you're not confused when you read this book, I want you to realize now that Paul and I don't agree on everything related to the development of COBOL programs. Yes, we agree on the essential techniques of program design, structured coding, and top-down testing, but we don't agree on some of the related techniques like the use of HIPO diagrams and walkthroughs. This is understandable because Paul's experience has been primarily in large businesses and mine has been in small businesses. Throughout this book, then, I'll try to make clear who believes in what so you can draw your own conclusions. Remember, though, that I'm writing the book, and I'm trying to do my best to present both Paul's opinion and my opinion whenever they differ.

Who this book is for

At the present time, there are thousands of COBOL programmers who are *not* writing structured programs. And this is a preposterous waste of programming talent. So if you're one of these programmers, this book is for you.

But this is also for those programmers who are writing "structured" programs, but who aren't writing reliable programs that are easy to read and maintain and who aren't producing their programs at a professional rate of productivity. Unfortunately, as you'll learn in section 1, we feel that these programmers represent the majority of the programmers working today.

If you are a programming manager, you probably have more personnel problems than you care to think about. So this book is for you too. If you enforce the method in this book throughout your shop, we believe you'll experience dramatic improvements in programmer productivity and program quality in your shop. So a year from now, you should have fewer personnel problems than you have today.

If you read the first edition of this book, we think you'll find that it's worth reading this one too. In brief, this book presents improvements to the method of the first edition; it shows you how to apply the method to interactive programs; and it gives you more background and perspective on program development.

Why this book is effective

We think this book is effective for five main reasons. First, Paul Noll is a practical man who emphasized the practical side of program development in the first edition of this book. And that's what we've tried to do in the second edition too. So this is not a book of theory; it is a book of proven techniques.

Second, Paul Noll has presented the method of this book to hundreds of programmers over the last ten years. Since 1978 Paul has been an independent consultant and trainer who has taught more than 5000 programmers how to program more effectively. Before that, he was the director of technical training for Pacific Telephone in San Francisco at which time he developed more than 50 courses for a staff of more than 120 programmers. As a result of these experiences, Paul has been able to refine his educational approach and to increase its effectiveness. And we've used Paul's approach in this book.

Third, this book was developed with the belief that a method should never be presented without showing its application. In contrast to other books, then, this book presents complete and practical solutions for actual programming problems. Once you see how the development method is applied to these problems, you can see how it can improve your effectiveness.

Fourth, the fact that this book presents structured programming in the context of COBOL increases its effectiveness. In contrast to the many books that present structured programming without relating it to a specific language, you won't be told how to design a program without seeing how the design relates to the COBOL code. You won't learn what the three valid coding structures are without seeing them coded in COBOL. You won't be told what a program stub is without seeing several in COBOL. And you'll see all of the structured coding techniques in COBOL.

Finally, this book contains dozens of illustrations taken from all phases of structured programming...more than twice as many as there were in the first edition of the book. You'll see structure charts for the four types of batch programs common to most business systems. You'll see

three different structure charts for the same interactive program as it's implemented using three different COBOL compilers. You'll see pseudocode for the key modules of these programs. You'll see many examples of COBOL coding. You'll see examples of other design and coding methods. And you'll see much more. In our opinion, these illustrations, more than any other factor, determine whether or not a methods course is effective...and they are the missing ingredient in other books on program development.

Some related products

If you want to run a course on the program-development method of this book, we offer a complete training package. It consists of this book, *Case Studies* that go with this book, and an *Instructor's Guide* that makes it easy for an instructor to administer a course using the case studies and text. This package not only teaches a student how to apply modern programming methods to the development of COBOL programs, but it also motivates experienced programmers to use these methods.

If you decide to adopt the method of this book throughout your shop, we offer a reference book called *The COBOL Programmer's Handbook*. This handbook presents standards for program development that conform to the recommendations made in this textbook. It also presents seven model programs, four batch and three interactive, that you can use as models for the development of your own programs. In addition, the *Handbook* can be used as a training aid in a course on program development.

If you're concerned about system design and development as well as program development, we offer a training package for system development too. It is called *How to Design and Develop Business Systems*. And it too consists of text, case studies, and advisor's guide. If you get this package, you will see that its methods for designing and developing business systems complement this book's methods for designing and developing COBOL programs.

Finally, if you train programmers in COBOL, we offer a complete package in *Structured ANS COBOL* that supports the development method presented in this book. It consists of three textbooks and an advisor's guide. If you get this package, you will see that it is closely coordinated with our package for designing and developing COBOL programs. So your COBOL students will learn to develop programs the right way from the start.

Conclusion

Quite frankly, we were disappointed with the sales of the first edition of this book even though we sold more than 35,000 copies of it. In fact, because the need for a better method of program development is so great,

we see our experiences with the first edition as a failure in marketing. We were even more disappointed to discover through a survey that less than 50 percent of the people who bought the first edition actually adopted the method it presented. And we know from 17 years in the publishing business that it's much easier to sell a book on a specific skill like CICS or TSO than it is to sell a book on a method for developing systems or programs. It seems, in fact, that most programmers will do everything they can to get the technical training they need, but they won't take the time to get the methods training that is critical to their effectiveness.

As you can see, though, that hasn't stopped us from revising the first edition of this book. I guess we just felt compelled to revise it because we've learned so much in the last eight years. Maybe this time the book will be more convincing. And certainly the development method is more complete than it was in the first edition. Maybe this time more people will join us in reaching the high levels of programming fulfillment that are possible when you use an effective method of program development.

If you have any comments or questions about the method or this book, we'd love to hear from you. So please use the comment form in the back of this book. And thanks for being our customer.

Mike Murach
Fresno, California
October, 1984

Section 1

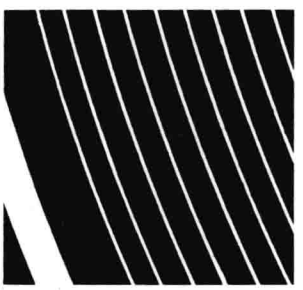
Introduction

This section is intended to give you some perspective on structured programming and the structured-programming movement. Specifically, this section tries to answer three questions that are important to any programmer or programming group: Has structured programming failed? If it has failed, why did it fail? And what can you do to make sure structured programming doesn't fail in your shop?

If by chance you're not interested in this introductory information, you can skip this section. In this case, you can go on to sections 2 and 3, which present the procedures and techniques you should use if you want structured programming to succeed in your shop. Section 2 presents the procedures and techniques you should use when you develop any business program using batch programs as examples. Section 3 shows you how to apply these procedures and techniques to the development of interactive programs.

Chapter 1

Has structured programming failed?



By the mid-1970s, it was clear that most companies that had computers were not doing an adequate job of developing COBOL programs. A study done in 1975 showed that the average COBOL programmer produced only 10 to 12 lines of tested code per day. In addition, the programs in a typical COBOL shop were generally unreliable, often requiring emergency repairs in order to get basic jobs done. And each year the cost of maintaining old programs went up so it wasn't unusual for a company to spend more than half of its programming budget on program maintenance rather than on new program development.

With these problems, it was natural that the users of most computer systems were something less than satisfied. In a typical shop, more than half the new programs were delivered behind schedule and over budget. And if a user made a request for a new program, he was likely to wait many months before the programming department even started it.

Into this situation, growing more desperate each day, came the promise of "structured programming." If we were to change our methods of developing programs, we were told, we could make dramatic improvements in both programmer productivity and program quality. Then, the computer users would be satisfied because programs would be delivered on time and within budget. And the programming backlog, the jobs waiting to be done by the programming department, would eventually be reduced to reasonable proportions.

By the late 1970s, the structured programming movement had gained considerable momentum. For several years, it was one of the primary

subjects of conversation and training. Countless articles and several books were published on the subject. Almost all COBOL training took on a structured look. And most COBOL shops made a serious effort to "get structured."

But now it's the mid-1980s, and you don't hear much about structured programming anymore. If you read the major trade magazines, you rarely find an article on structured programming. Only an occasional book is published on the subject. And, overall, the subject seems quite dead. Microcomputers, networking, and fourth-generation languages seem to be far more important to the average data processing manager.

The questions we have to ask, then, are these: Has structured programming fulfilled its promise? Is that why the subject is dead? Or, has structured programming failed? After the articles were published, after the seminars were given, after the development standards were revised...did things improve or did most COBOL shops simply replace one set of ineffective development standards with a new set?

Before I try to answer these questions, I'm going to review the major changes in program development from 1965 to the present. Next, I'll point out the shortcomings of unstructured development methods and the promised benefits of structured methods. Then, I'll present the effect of the structured programming movement on programmer productivity and program quality. Last, I'll summarize the results of a COBOL survey we recently conducted. When you complete this chapter, I think you'll be able to decide for yourself whether or not structured programming has failed. Keep in mind, though, that the important question is not so much whether structured programming has failed, but whether structured programming has failed in *your* shop.

Changes in COBOL program development from 1965 to the present

Although the COBOL language was developed in the late 1950s, it wasn't used much until third-generation computer systems were introduced in the mid-1960s. As a result, the history of COBOL programming is a short one. The significant history runs from 1965 to the present.

In general, you can divide COBOL programming into three styles or eras. First, a typical shop in 1965 simply adapted assembler language techniques to the development of COBOL programs. The result was what I call *GOTO programming*. Second, in an attempt to improve programmer productivity and program quality, a typical COBOL shop experimented with *modular programming*. Finally, in an attempt to improve on modular programming, a typical COBOL shop converted to *structured programming*.

More important than this general history, though, is the history of program development in each COBOL shop. For instance, a well-run COBOL shop may have started with GOTO programming in 1965, moved to modular programming by 1970, and established structured

programming by 1975. In contrast, a poorly-run shop may still be developing programs using GOTO programming.

If you've been in the data processing business for more than a short time, you probably know what I'm talking about when I mention GOTO, modular, and structured programming. But in case you don't, here's a brief introduction to each of these programming styles. Once you understand them, you'll have a better idea of why the promise of structured programming was so alluring.

GOTO programming In the GOTO era of program development, a programmer designed a program by developing a detailed program flowchart. Figure 1-1, for example, is a flowchart for a simple file-to-printer program that prepares a wage report. As you can see, the flowchart goes into so much logical detail that the programmer has used a flowcharting form to make it somewhat easier to follow. If you work in a well-run COBOL shop, it may be hard for you to believe that people ever created flowcharts like this. But ten years ago, they were quite common.

After a programmer completed a detailed flowchart, he began coding the program using the flowchart as a guide. When he reached a decision block like block Jl, he continued with the processing for one of the conditions and coded the routines for the other conditions later on. After all routines were coded, a good programmer checked the code for completeness using the flowchart as a reference.

One of the characteristics of traditional COBOL coding was the extensive use of GOTO statements, because at least one GOTO was used for every decision block in a flowchart. To code the wage-report program based on the flowchart in figure 1-1, for example, a programmer would probably have used a dozen or more GOTO statements. But it is this branching from one paragraph to another within a COBOL program that makes the code difficult to read and modify. In actual practice, a traditional COBOL program that used 1000 different verbs in the Procedure Division was likely to use anywhere from 100 to 250 GOTO statements, so the code was extremely difficult to follow.

Modular programming As you can see in figure 1-1, a flowchart from the GOTO era had no identifiable structure. As a result, it was difficult to create, read, and maintain. And the resulting program was difficult to code, test, debug, and maintain. By the late 1960s, then, programmers in the better COBOL shops began to think in terms of modular programming.

The idea of modular programming was to divide a program into a number of independent modules—one mainline module and one or more subroutine modules. By so doing, the logic of a program was simplified, which in turn simplified coding and testing.

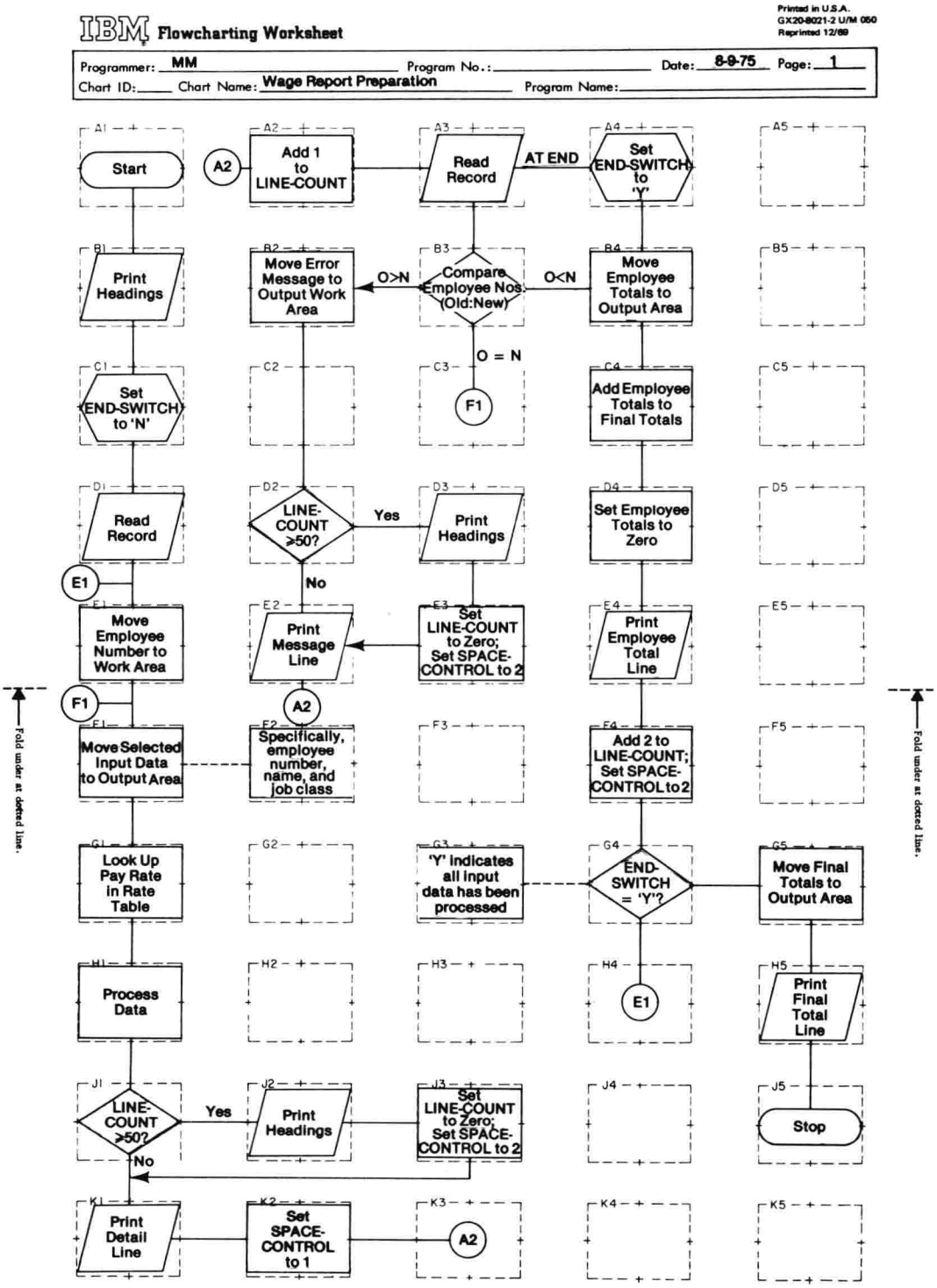


Figure 1-1 A program flowchart for a program that prepares a wage report

Figure 1-2, for example, is a flowchart for the mainline module of the same wage-report program that is flowcharted in figure 1-1. This module indicates all the major processing modules as well as the logical decisions required to direct the program to these modules. As you can see, each module has a stripe at the top of the symbol to indicate the name of the module. So the wage-report program consists of the mainline module plus eight other modules named HOUSEKEEPING, READ-RECORD, and so on.

After a programmer completed the flowchart for the mainline module, she flowcharted each of the subroutine modules, or at least the more complex modules. To continue the concept of modularity, she might divide a subroutine module into additional, more specific, modules, depending on the length and complexity of the module. Since the idea was to make each programming module manageable, the programmer tried to keep each module between 50 and 200 statements in length.

To code a modular program, the programmer started with the mainline module. Next, she coded each of the subroutine modules. In contrast to GOTO programming, then, the modular program was coded in a somewhat predictable sequence.

By using modules, the COBOL programmer cut down on her use of GOTO statements. Instead, to control the execution of subroutine modules, she used PERFORM statements. For instance, the code in figure 1-3 represents the mainline module flowcharted in figure 1-2. Notice that all the PERFORM statements are written with the THRU EXIT clause so the mainline routine won't have to be changed no matter how many paragraphs the subroutines contain. Notice also, however, that GOTO statements are still used extensively. In this 22-statement routine, five GOTO statements are used.

Structured programming When structured programming started to become popular in the mid-1970s, it was somewhat revolutionary. It said throw away your flowcharts and your GOTOs and do things differently. Because many people were aware of the problems they experienced using unstructured techniques, they were ready to make this revolutionary change. Then, when success stories were published showing the dramatic improvements possible when you used structured programming, more people became convinced that they should make this change to new methods. Eventually, so many people had changed their methods, were in the process of changing them, or were considering changing them, that it was clear structured programming was here to stay. In fact, so many people changed that it was fair to describe the change as an industry-wide movement, the structured programming movement.

When you develop programs using structured programming, you no longer use a program flowchart (although some people tried). Instead, you design a program using a design document like a structure chart, a