STRUCTURED ANS COBOL PROGRAMMING

second edition

William M. Fuori
Stephen J. Gaughran

2ND
edition

Structured ANS COBOL Programming

WILLIAM M. FUORI, Ph.D., C.D.E.
STEPHEN GAUGHRAN, M.B.A., C.D.P.

Nassau Community College
New York

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

Library of Congress Cataloging in Publication Data
FUORI, WILLIAM M.
Structured ANS COBOL programming.

Rev. ed. of: Introduction to American national standard COBOL. 1975.
Includes index.
1. COBOL (Computer program language) 2. Structured programming. I. Gaughran, Stephen. II. Fuori, William M. Introduction to American national standard COBOL.
III. Title.
QA76.73.C25F86 1984 001.64'24 84-2100
ISBN 0-13-854430-1

Editorial/production supervision and interior design; *Aliza Greenblatt*

Cover design: *Celine A. Brandes*

Manufacturing buyer: *Gordon Osbourne*

The authors would like to thank IBM for reproduction of their business forms.

Previous edition published under the title of
Introduction to American National Standard COBOL

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-854430-1

PRENTICE INTERNATIONAL, INC., *London*
PRENTICE-HALL OF AUSTRALIA PTY. LIMITED, *Sydney*
EDITORA PRENTICE—HALL DO BRASIL, LTDA., *Rio de Janeiro*
PRENTICE-HALL CANADA INC., *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*

In memory
of my beloved mother
-William M. Fuori

To my wife Claire,
and to my children Stephen T., Maureen,
Kathleen, Patricia, Sheila, and Colleen
-Stephen Gaughran

Preface

WHAT IS COBOL?

With the introduction of the computer into the business world in the early 1950s came the need for a standardized business-oriented programming language. To ensure the universal applicability of this language, a committee (CODASYL, for the Conference On DAta SYstems Languages) comprised of computer manufacturers, government agencies, and commercial users was formed and charged with developing a standardized business-oriented language. In 1959 this committee began meeting to develop COBOL (COmmon Business Oriented Language), a language that would not be identified with any computer manufacturer and that therefore would offer advantages to both government and commercial users.

In 1968 the American National Standards Institute (ANSI) documented the COBOL specifications, agreed on by representatives from interested groups. In 1974 ANSI published a revised COBOL standard that incorporated the suggestions of computer manufacturers and users since the previously published standard was issued.

The 1974 changes have been implemented by most manufacturers. However, some manufacturers still have not completely implemented these specifications and most manufacturers are currently in the process of implementing the latest changes. It is therefore, important that the reader verify exactly which specifications are supported by his or her compiler before attempting to use them.

The latest set of changes proposed by ANSI in 1981 is currently in its final stages of acceptance and has been included in Appendix C.

INSTRUCTIONAL FEATURES

The entire format of this text is based on firsthand classroom experience and on the many suggestions made by COBOL instructors, students, and business people. The instructional features in *Structured ANS COBOL Programming, 2/E* are perhaps the most innovative and constructive ever presented in a COBOL textbook. They provide students with a new approach to learning COBOL by giving them practical application of what has often been considered difficult and abstract material.

Standard COBOL

The text includes a comprehensive discussion of each of the major COBOL language statements. Then to accommodate students or programmers who must prepare for the changes that will be required to convert from a COBOL compiler based on 1974 COBOL

standard to one based on the 1981 proposed revision of the standard, a summary of the changes is provided in Appendix C.

Straightforward Presentation

In this textbook **straightforward presentation** has two meanings. First, it means that the COBOL language statements are presented in a manner that makes them easy to understand. Second, it means that students are introduced very early to the basic requirements and procedures of the COBOL language. For instance, the students are introduced to the ACCEPT (for reading in data) and DISPLAY (for writing out data) clauses early in the text (Chapter 3). This early introduction will allow them to write almost immediately a simple but complete program.

Once students have gained the confidence that comes from having successfully written a few simple but complete COBOL programs, they will be better prepared to understand the conceptually more difficult material that follows. In this "learning by doing" approach, students advance step by step from the simple to the complex.

After mastering the basic COBOL concepts, students are assigned more detailed and thought-provoking programming exercises. Some of these programming assignments include the more sophisticated aspects of COBOL programming, such as table handling, disk operations, and the SORT feature.

Abundant Exercise and Self-Testing Material

The exercises in this textbook fall into two categories. First, there are Self-Tests that are referenced throughout each chapter. These Self-Tests are carefully placed to allow the reader to test his or her understanding of the foregoing material.

Second, there is the exercise material which appears at the end of each chapter. These exercises, which include true–false questions, multiple-choice and completion questions, and items for discussion, cover all the material in their respective chapters, including those areas covered by the Self-Tests. In this manner, students are given a second opportunity to strengthen their knowledge of the subject.

In this end-of-chapter exercise material, there are other exercises deserving special attention. Perhaps, more than any one feature, these exercises, called "Problems" and "Programs," allow the student to apply his or her accumulated knowledge to comprehensive problems and in the solution of real-life business situations. Sample problems might ask the student to contrast the various file organizations while sample programs might require a student to write a program on current sales data for the sales representatives for a particular company or to write a program to handle an airline passenger reservation system. These are problems similar to those actual problems that the student will encounter in the business world.

All programming examples and assignments have been compiled and executed on a computer to ensure that they are "working" programs. The computer listings themselves are reproduced in the text for programming examples and in the instructor's manual and key for programming assignments.

Manufacturer-Independent COBOL

Structured ANS COBOL Programming, 2/E is written independent of any one computer system. However, there are sections in the text that refer to the IBM enhancements to the COBOL standard. These enhancements are identified as such, since they are not generally supported by the COBOL compilers of other computer manufacturers. Thus, students or programmers using an IBM computer system can take full advantage of that particular system, and students or programmers using another computer system can easily distinguish between the instructions pertaining only to the IBM system and those pertaining only to their system.

CONTENTS

Terminal or Batch-Oriented

As most colleges and businesses now support interactive COBOL facilitating the entry-,compilation, and execution of COBOL programs via terminal, the exercises in this text are designed so that they can be completed on either a batch system or an interactive system.

ORGANIZATION AND CONTENT

Structured ANS COBOL Programming, 2/E is organized into six text units, which are divided into 13 chapters, and a seventh unit which provides supplementary information in Appendixes A through E.

Unit I (Chapters 1 and 2) introduces the student to American National Standard COBOL and its basic structure. The student learns why the COBOL language came into being, how the language is structured, and what takes place during a COBOL compilation (the translation of the COBOL statements into machine-executable code).

Unit II (Chapters 3 and 4) introduces the student to those statements minimally necessary to write a simple COBOL program. In Chapter 3, the student learns to input and output discrete data utilizing the ACCEPT and DISPLAY statements. These statements are ideally suited to systems that input and output data via terminal. Chapter 4 follows up with the COBOL concepts and statements necessary to facilitate the input and output of files. The PERFORM statement is presented in this chapter so that structured programming can be introduced as early as possible.

Unit III (Chapters 5 and 6) are concerned with the arithmetic operations and branching statements, respectively.

Unit IV (Chapters 7 and 8) is concerned with some advanced COBOL statements and concepts. For example, Chapter 5 explains and illustrates the USAGE, SYNCHRONIZED, JUSTIFIED, BLANK WHEN ZERO, and advanced PICTURE clauses as well as introducing the EXAMINE, INSPECT, STRING, UNSTRING, and COPY statements. Chapter 6 introduces the table-handling features of COBOL.

Unit V (Chapters 9, 10, and 11) discuss the principally used file organizations and access methods both conceptually and from a practical programming point of view. Complete programs are provided to illustrate how to create, update, and process sequential, direct, and indexed (ISAM and VSAM) files.

Unit VI (Chapters 12 and 13) introduce the COBOL SORT and MERGE features as well as the construction and accessing of subprograms in COBOL.

Unit VII (Appendixes A through E) contains supplementary materials that can be used throughout the text at the instructor's discretion. Included are detailed presentations on debugging of a COBOL program (Appendix A), documenting a COBOL program (Appendix B), the changes proposed by ANSI in 1981 (Appendix C), the General Formats of COBOL statements for quick student reference (Appendix D), and the ANS 1974, 1981, and IBM enhanced COBOL reserved word lists (Appendix E).

Because the emphasis of the text is on programming rather than on the COBOL language statements taken individually, there is a continuity from one chapter to the next. Within each unit the student comes to view the topics as a meaningful whole instead of a series of seemingly disjointed topics. The successful classroom field-testing of *Structured ANS COBOL Programming 2/E* with both business and data processing majors confirms that this textbook's approach to learning provides the COBOL student with an interesting, informative, and easy-to-read set of instructional materials and the COBOL programmer with a convenient, up-to-date reference on COBOL programming.

INSTRUCTOR'S MANUAL AND KEY

An instructor's manual and key is available to instructors in education and industry using the text. Particularly valuable as an aid in structuring the COBOL course to fit the interests and backgrounds of the students, this manual suggests methods of presentation re-

sulting from the classroom field-test of *Structured ANS COBOL Programming, 2/E*. The detailed lecture notes that are a feature of this manual include suggested discussion questions. The instructor's use of these materials will assure maximum coordination between the lectures and the reading and study assignments.

The detailed solutions provided for each end-of-chapter exercise include a computer listing for each exercise requiring the student to write a computer program. A *feature of special importance to the instructor is the inclusion of sample input data and the resultant output for each programming exercise. This relieves the instructor of having to make up test data and of having to write the program in order to be aware of the desired output.*

ACKNOWLEDGMENTS

Many instructors and students aided in the development of the manuscript for *Structured ANS COBOL Programming, 2/E*. I would like to thank, Professors Joseph Pacilio and Dominick Tedesco for their invaluable assistance and constructive criticism.

The following information is reprinted from *American National Standard Programming Language COBOL*, published by the American National Standards Institute, Inc.

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. F28-8013, copyrighted 1959 by IBM; FACT, DSI 27A52602760, 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.

For a complete record of the COBOL standard and specifications, the reader is referred to *American National Standard Programming Language COBOL*, American National Standards Institute, Inc., New York, 1974.

William M. Fuori
Stephen Gaughran

Contents

1

General Concepts

The introduction of the computer to the business world brought with it many new and complex problems. Prior to this time, computers had been used only for scientific purposes. Consequently, computer manufacturers and users were faced with developing a data processing system that would be usable on these existing computers and at the same time would be applicable to the newer, larger, and more powerful computers with minimal conversion, reprogramming, and retraining costs. They were also faced with developing a single business-oriented computer language that would be usable on most medium- and large-scale computers to replace diverse computer languages that varied extensively from one computer manufacturer to another. Also, the average programmer would have to be able to write a program in a reasonably short period of time using this language which, to some extent, would be self-documenting. As this language was to be heavily used in business, it would have to be one that noncomputer-oriented personnel, such as accountants and auditors, could read and understand with a reasonable amount of training.

To fulfill these needs, development began on a suitable and standardized commercial programming language. In May 1959, a committee of computer users was formed—consisting of computer manufacturers, representatives of the federal government, and other interested parties—and named **CODASYL** (**C**onference **O**n **DA**ta **SY**stem **L**anguages).

In April 1960, the committee produced a report titled COBOL (COmmon Business Oriented Language). A CODASYL COBOL Maintenance Committee was formed and charged with the responsibility for making needed modifications to the language. In order to make this language universally acceptable to the business community, this founding group granted unrestricted use of the language specifications to all users.

COBOL was received with tremendous success in the business field and it was apparent that this language was to have a long and bright future. Realizing this, USASI (United States of America Standards Institute) set out to produce COBOL specifications that were to be consistent with CODASYL specifications and were to be used as the standard COBOL by computer manufacturers. In August 1968, their efforts resulted in USASI COBOL or, as it is now known, **ANS** (**A**merican **N**ational **S**tandard) **COBOL**. A more recent revision to the ANS COBOL specifications published late in 1974 is shown in Appendix D compared with the enhanced specifications supported by IBM's OS Full ANS COBOL Compiler. The 1981 proposed revisions appear in Appendix C.

One of the most significant changes of concern to programmers in general was the introduction of top-down program design. This revolutionary program design concept changed the emphasis from how to code a program to how to design a program prior to the actual coding so that the program will be as error free, reliable, and easy to read, modify, and maintain as possible. This method of program design is discussed in greater detail later in this chapter and is used throughout this text.

To verify your understanding of the preceding material, turn to page 14 and take Self-Test 1-1.

COBOL ADVANTAGES AND FEATURES

The COBOL language is an easy-to-learn, easy-to-read, high-level programming language designed principally for use in business or commercial applications. Some of the advantages and features of this language are:

1. COBOL programs are written using precise, easily learned English words and phrases.
2. COBOL is usable on virtually every computer manufactured because it is a universally accepted standard language.
3. COBOL programs written for use on one computer are usable on other computers with a minimum of change.

4. COBOL programs are written utilizing common business terminology and are therefore easily read by nonprogrammer personnel such as accountants, auditors, or business executives with only a minimal background in data processing.
5. COBOL is easily learned by individuals who do *not* have extensive training in high-level mathematics.
6. COBOL facilitates program testing so that, if necessary, programs can be tested efficiently and thoroughly by individuals other than the original programmers.
7. Documentation of a COBOL program is relatively simple. In many cases, the COBOL program itself provides much of the total documentation required.
8. Provisions are provided for the updating of the COBOL language.
9. It is suited to top-down program design techniques.

A review of some relatively simple but important concepts will show what programmers must know and understand before they attempt to use a sophisticated language such as COBOL. These concepts may be reviewed by answering the following three questions:

1. What types of problems are suitable for computerized solution?
2. What general steps must programmers take when creating a COBOL program?
3. What happens after the COBOL program is written but before it is run on the computer to produce the desired output?

PROBLEMS SUITABLE FOR COMPUTERIZED SOLUTION

In today's automated world, when people are faced with the task of solving an analytical problem, they often turn to a computer for help. However, some problems lend themselves more to a computerized solution than others. Those problems for which a computer is ideally suited generally have the following characteristics:

1. Definable
2. Repetitive
3. Volume of storage or calculations
4. Cost-justifiable

Definable

It must be possible to clearly state the problem in terms of objectives that can be reached as the result of a series of logical and arithmetic steps. A computer, however, is limited in the types of operations that it can perform; the type of application to which it can be applied is also limited.

Repetitive

The application or task should be one that will be performed again and again. A typical example of a repetitive task is the generation of a company's weekly payroll; the same computations are required to produce the paycheck for each employee in the company, with only the actual numbers varying. This type of application is also repetitive in that it is executed many times in a given time span—52 times a year. For applications that are repetitive, the cost of preparing, writing, and documenting a program for a computerized solution is spread over the program's period of use. In addition, the boredom factor—experienced by people performing repetitive and monotonous tasks—is nonexistent for a computer.

Volume of Storage or Calculations

An application suitable for a computerized solution generally requires large quantities of data to be stored or processed and numerous logical or arithmetical calculations to be performed by the computer. Applications not requiring a volume of storage or processing can often be accommodated more easily and economically by a manual or other system.

Cost-Justifiable

The end result—the knowledge gained, the data or output created, the money or time saved—must justify or substantiate the cost of preparing, writing, and executing a program for computing the solution. Employing a computer to solve a problem will not always result in a saving of time or money. Many menial tasks can be performed more economically by noncomputerized devices.

Once a given application is found to be suitable for a computerized solution, the preparation for the construction of a computer program can begin.

To verify your understanding of the preceding material, turn to page 14 and take Self-Test 1-2.

THE PROGRAMMING PROCESS

A substantial amount of preparation is necessary before the actual programming of an application can begin. The programmer must be concerned with such items as what functions are to be performed by the program; what input data will be provided; what the form of the input data will be; what output medium will be used; and what data is to be output from the program. These questions represent some of the many questions to be answered before the programmer can begin to write down the first instruction in the program. The programmer must not only be able to view the problem as a whole but also possess a detailed understanding of each of its component parts. This is done with a constant awareness of both the capabilities and the limitations of the computer. In general, the programmer subdivides the programming task into four areas:

1. Problem Analysis
2. Program Design
3. Coding, Executing, and Debugging the Program
4. Documentation

Problem Analysis

The first step the programmer takes in preparing an application for programming is to analyze the problem. This analysis begins with the studying and clarifying of the statement of the problem. Initial problem statements are generally somewhat vague and imprecise for a number of reasons. Sometimes the terms used in the statement of the problem cannot themselves be stored or maintained by computer (temperature, pressure, debits, etc.); other times the problem statement tacitly relies on the capabilities and common sense of the human reader; while still other times, the exact requirements of the problem have not been completely formulated by the user.

In any event, the programmer with the help of the originator of the problem must eliminate any vagueness, imprecision, or ambiguity in the problem statement so that he or she may formulate a precise list of program objectives. On other than the most trivial of problems, a systematic analysis of the requirements, and design of the overall structure of the program, should precede any attempt to write program statements. It is convenient to view this process as a **top-down** or **level-by-level** analysis of the problem. The top level is the initial statement of the problem; the bottom level is the complete program; the number of intervening levels depends on the complexity of the problem. The levels occurring after the initial level are generally designed to:

1. Break up big problems into smaller ones—which in turn are broken up into even smaller ones.
2. Reduce the program objectives from English to programming terms—convert "find," solve," and so on, to "read," "print," "repeat," and then eventually to READ, WRITE, PERFORM—the statements of a programming language.

This process can be carried out informally or in accordance with certain established techniques such as HIPO (**H**ierarchy plus **I**nput-**P**rocessing-**O**utput). Although this proc-

ess may often be carried out informally, we shall briefly consider HIPO for purposes of illustrating one of the formal procedures used.

HIPO, originally designed as a documentation tool, has become a useful tool of program design as well. The major objectives of techniques such as HIPO in program design and documentation are:

1. State the functions to be accomplished by the program.
2. Provide an overall structure or hierarchy by which the individual functions of the program can be understood.
3. Provide a visual description of the input to be used and the output to be produced by each function.

To accomplish these ends, HIPO utilizes two types of charts: a hierarchy chart and an input-process-output chart. The **hierarchy chart** pictorially illustrates how each program function is divided into subfunctions or modules, and the **input-process-output chart** expresses each module in the hierarchy in terms of its input and output considerations. Examples of these charts are shown in Figures 1-1 and 1-2.

Figure 1-1 deserves a little explanation. What this figure illustrates is how the program function "Calculate Pay" is subdivided into subfunctions or modules. Specifically, this chart shows us that "Calculate Pay," designated as hierarchy level 1.0, is divided into subfunctions "Calculate Gross Pay" and "Calculate Net Pay." Thus, in order to "Calculate Pay," one must first "Calculate Gross Pay" and "Calculate Net Pay." Since "Calculate Gross Pay" is at level 2.0, it must be completed before proceeding to level 3.0. To complete level 2.0 one must complete sublevels 2.1, 2.2, and 2.3. Each of these is completed as the result of completing its sublevels. For example, to complete level 2.2, one must complete levels 2.2.1, 2.2.2, 2.2.3, and 2.2.4. Thus, before level 2.0 can be considered complete, all sublevels beginning with 2.XXX must have been completed.

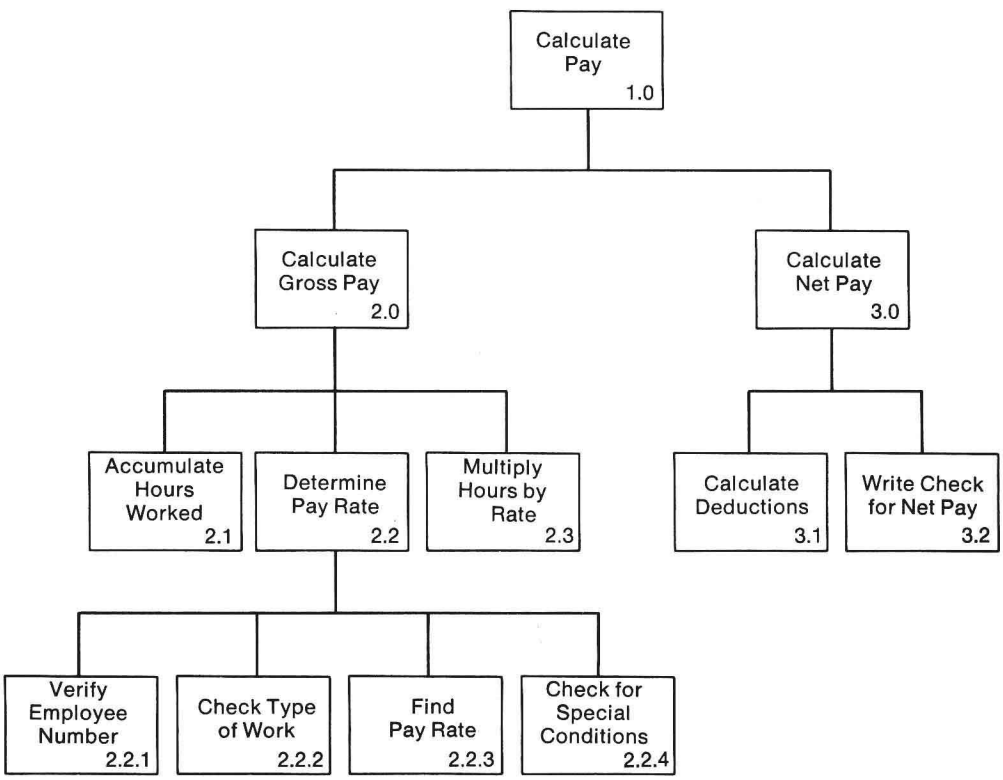

FIGURE 1-1 Example of a HIPO hierarchy chart.

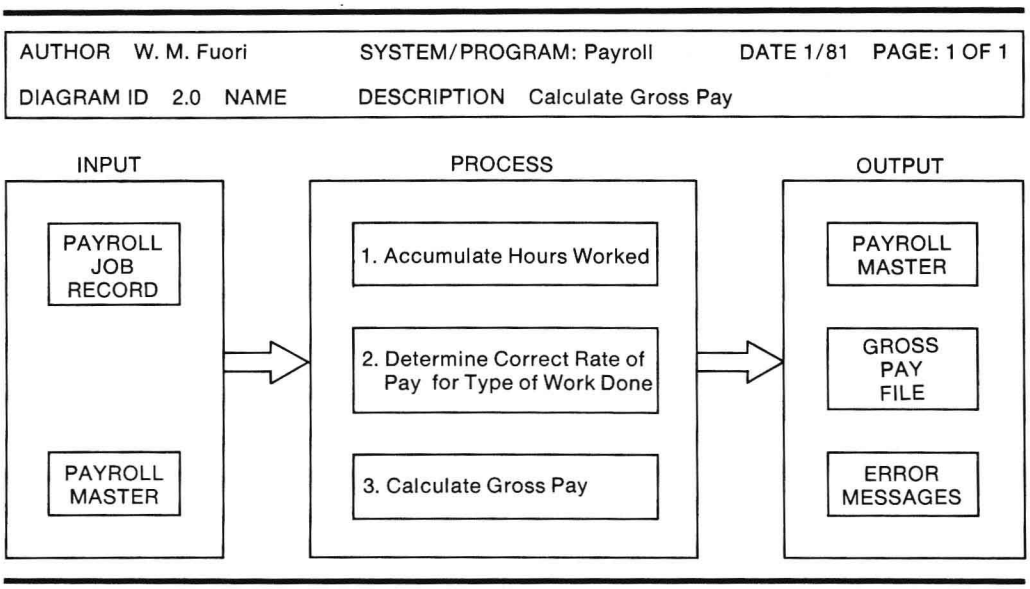

FIGURE 1-2 Example of a HIPO input-process-output chart.

Once level 2.0 has been completed, level 3.0 can commence, and so on, until all levels in the hierarchy have been completed.

HIPO diagrams are constantly being refined as the development of the program continues. The initial emphasis is on ensuring that program functions are clearly understood by both the designers and originators so that the resulting program will meet the needs for which it is being designed. The major functions of the program are listed at the top level and expanded to lower-level modules as more detail is required.

During the process, certain basic guidelines should be observed:

1. All modules in the hierarchy chart should logically relate to one another, with control passing from the top module down to the next lower module, and so on.
2. Each module in the hierarchy chart should represent only one program function and contain a single entry point and a single exit point (only one way into the module and only one way out of the module).
3. The functions of each module should be specific enough so that it can be implemented with a minimum of one page or approximately 50 lines of coding. This will greatly enhance the texting and debugging of the module.

Once this phase of the program design has been completed using a formal procedure such as HIPO, or informally as described earlier, the next can step begin.

Program Design

Once the hierarchy and input-processing-output charts have been completed, the detailed program design can commence. This can be accomplished with the aid of a program flowchart or pseudocode.

The **flowchart** is a graphic representation of the nature and order of the arithmetic and logical operations required to solve a problem. **Pseudocode,** on the other hand, is an imitation computer code. It is used in place of symbols or a flowchart to describe the logic of a program. Pseudocode is intended to overcome the two principal disadvantages of the flowchart: The flowchart is time consuming to create and is difficult to modify without redrawing it completely.

Pseudocode employs the basic structures employed in structured programming (**SEQUENCE, IF-THEN-ELSE, DO-WHILE** or **DO-UNTIL,** and the **CASE** structure). Figure 1-3 illustrates these structures in both flowchart and pseudocode form. Figure 1-4 illustrates a program segment to determine a salesperson's pay as it would appear in a flowchart and in pseudocode.

Programmers and systems analysts still disagree on whether they prefer the program flowchart or pseudocode. Both techniques are currently in use. Only the future will tell which technique, if any, will gain universal acceptance. For our purposes in this text, we will employ the program flowchart.

As stated above, the flowchart is a graphic representation of the nature and order of the arithmetic and logical operations required to solve a problem. It illustrates the sequential order of the steps to be taken to process the input data and provide meaningful results. The programmer's flowchart is similar to an architect's blueprint. The architect draws a blueprint before starting construction of a building; the programmer constructs a flowchart before starting the programming of an application. Figure 1-5 illustrates an IBM template that can be used for the construction of structured program flowcharts. Note that the basic programming structures are shown on the template for ready reference.

The program flowchart serves the programmer in many ways, some of which are the following:

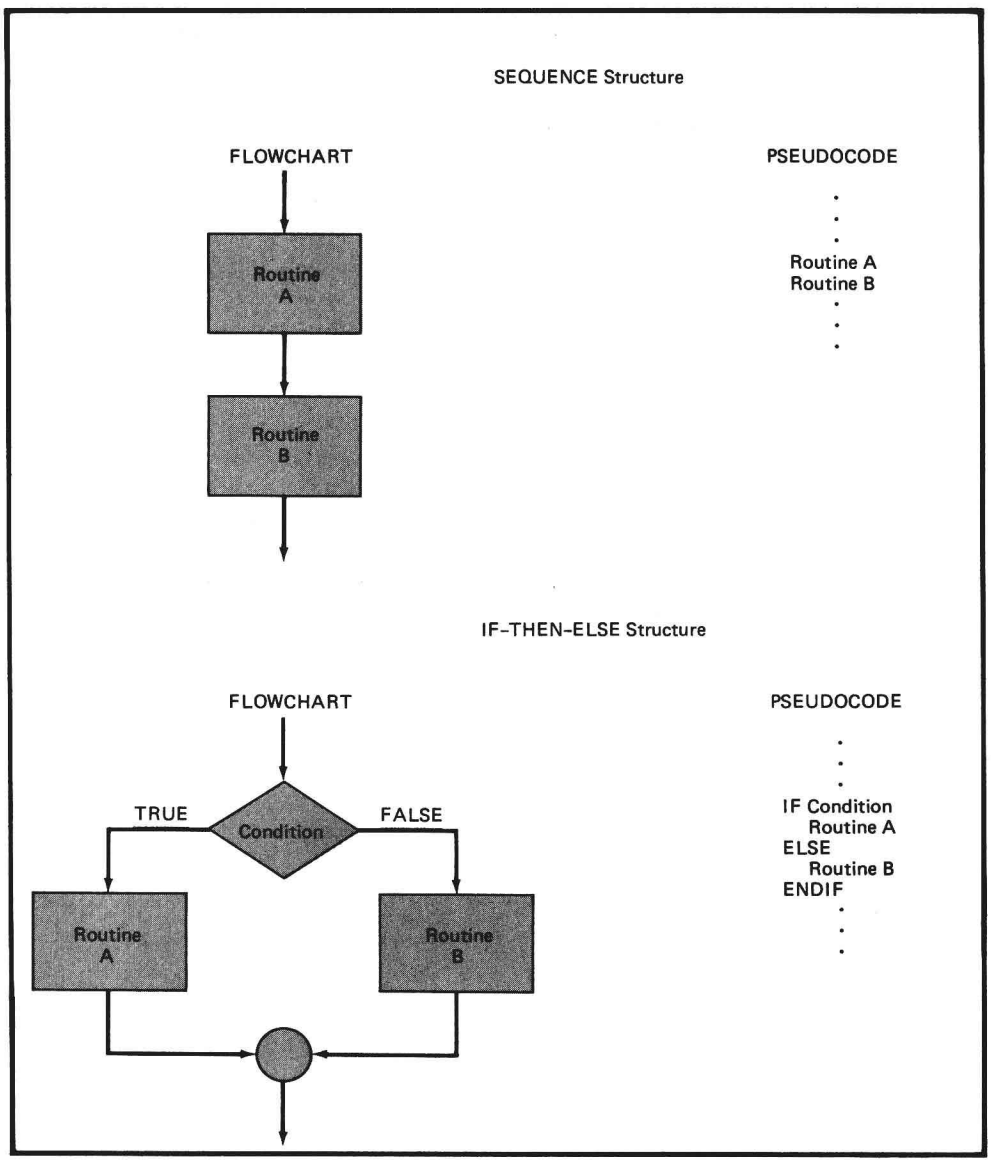

FIGURE 1-3 Basic structures shown in flowchart and pseudocode forms.