STERN & STERN

STRUCTURED COBOL PROGRAMMING

9th Edition

```
PROCEDURE DIVIS
100-MAIN-MODULE
    PERFORM 200
    PERFORM UNT
        READ ACC
          AT END
            MOVE
          NOT AT
            PERF
        END-READ
    END-PERFORM
    PERFORM 900
    STOP RUN.
200-INITIALIZAT
    OPEN INPUT
         OUTPUT
300-CALC-MODULE
    PERFORM 700
    PERFORM 400
        YEAR FR
400-COMPUTE-INT
    MOVE YEAR T
    COMPUTE WS-
        ** YEAR
```

for the year 2000 and beyond

STRUCTURED
COBOL
PROGRAMMING

NINTH EDITION

For the Year 2000 and Beyond

NANCY STERN
Hofstra University

ROBERT A. STERN
Nassau Community College

JOHN WILEY & SONS, INC.
New York Chichester Brisbane Toronto Singapore Weinheim

Acquisitions Editor Beth Lang Golub
Marketing Manager Peter Rytell
Development Editor Marian Provenzano
Full Services Manager Jeanine Furino
Production Service and Interior Design HRS Electronic Text Management
Cover Design Harry Nolan

This book was set in 10/12 Palatino by UG / GGS Information Services, Inc. and printed and bound by Donnelley/Crawfordsville. The cover was printed by Phoenix Color.

Recognizing the importance of preserving what has been written, it is a policy of John Wiley & Sons, Inc. to have books of enduring value published in the United States printed on acid-free paper, and we exert our best efforts to that end.

Library of Congress Cataloging in Publication Data
Stern, Nancy B.
Structured COBOL programming : for the year 2000 and beyond / Nancy Stern, Robert A. Stern.—9th ed.
p. cm.
ISBN 0-471-31881-7 (paper : alk. paper)
1. COBOL (Computer program language) 2. Structured programming.
I. Stern, Robert A. II. Title.
QA76.73.C25S75 2000
005.13'3 dc21 99-29008
CIP

Printed in the United States of America

10 9 8 7 6 5 4 3

PREFACE

INTENDED AUDIENCE

This book is intended for readers with no previous programming or computer experience as well as for those with some background in the computing field. It has been specifically designed for use in college courses on COBOL both in two-year and four-year schools.

OBJECTIVES OF THIS BOOK

1. To teach students how to design programs so that they are easy to read, debug, modify, and maintain.
2. To provide students with the ability to write well-designed elementary, intermediate, and advanced structured COBOL programs in their entirety. These include both batch and interactive programs.
3. To familiarize students with information processing and systems concepts that will help them interact with users and systems analysts when designing programs.
4. To focus on the key elements of the most recent COBOL standard, called COBOL 85, that facilitate and promote the writing of well-designed structured programs. We highlight where COBOL 85 features differ from COBOL 74, the previous standard. We also specify changes that will be incorporated in the next standard, which is currently referred to as COBOL 2000+.
5. To familiarize students with programming tools such as pseudocode and hierarchy charts that make program logic more structured, modular, and top-down. We also provide information on the flowchart, which is an older tool, but one that is still used by some for planning purposes.
6. To teach students useful techniques for maintaining and modifying older "legacy" programs.
7. To alert students to the Year 2000 (Y2K) Problem and to provide specific ways to correct the problem.

HOW THIS BOOK DIFFERS FROM *STRUCTURED COBOL PROGRAMMING*, EIGHTH EDITION

The ninth edition of *Stuctured COBOL Programming* includes the following revisions:

1. *A fully integrated explanation of the Year 2000 (Y2K) Problem and its remedies.*
   - In Chapter 4, where coding of complete COBOL programs is described in detail, we discuss the problem, why it occurred, and suggestions for making all COBOL code Y2K compliant.
   - Then, throughout the book, where programs with date routines are provided, we include an explanation of the coding required to avoid and eliminate this problem.

- All Y2K material is identified with an icon that designates it as a Y2K issue:

Y2K

2. *Updated Coding.*
   - We continue to emphasize COBOL 85 coding, while providing a limited discussion of the older COBOL 74 code, along with the changes—both approved and proposed—in the new standard. Students who will be working on legacy code to update COBOL programs so that they are Y2K compliant need to know COBOL 74.
   - To facilitate the reader's understanding of these three standards, we use icons to highlight a feature that specifically relates to a particular standard:

   COBOL 74 COBOL 85 COBOL 2000+

3. *Internet resources related to COBOL are provided thoughout the text.*
   - Because such resources are constantly changing, we provide discussions throughout the text on how students can use search engines and other tools to locate various COBOL sites. Throughout the text, Web sites that contain important COBOL information are cited.
   - References in the text that point to Web sites include a Net icon:

   The NET

4. *More material on indexed and relative disk file processing has been added.*
   - Discussions have been expanded on the `START` statement, `DYNAMIC` access of files, `FILE STATUS` codes, and `ALTERNATE RECORD KEYS`.
5. *Improved pedagogy.*
   - The basic pedagogic approach and teaching tools used in the eighth edition have been maintained.
   - In addition, we have added debugging tips and critical thinking questions to each chapter.
   - We have also eliminated some repetitive material—examples as well as narrative.
   - Each chapter now includes Internet assignments to familiarize students with sites that can be used to enhance their COBOL skills.
6. *Other content revisions.*
   - More material has been provided on multiple-level tables and arrays without inundating students with more material than they could possibly handle.
   - In-depth discussions have been provided on intrinsic functions.
   - Screen layouts and interactive programming have been emphasized. In Chapter 6, we introduce the `SCREEN SECTION` and illustrate how users can create fully interactive programs. From Chapter 6 on, every chapter includes interactive elements and at least one end-of-chapter Programming Assignment that requires interactive processing. Topics that specifically relate to interactive issues are highlighted with the use of an interactive icon:

FEATURES OF THE TEXT

FORMAT

The format of this text is designed to be as helpful as possible. Each chapter begins with:

1. **A detailed chapter outline.**
   Before beginning a chapter, you can get an overview of its contents by looking at this outline. In addition, after you have read the chapter, you can use the outline as a summary of the overall organization.
2. **A list of objectives.**
   This list helps you see what the chapter is intended to teach even before you read it.

The material is presented in a step-by-step manner with numerous examples and illustrations. Within each chapter there are self-tests, with solutions, that are designed to help you evaluate your own understanding of the material presented. We encourage you to take these tests as you go along. They will help pinpoint and resolve any misunderstandings you may have.

END-OF-CHAPTER MATERIAL

Each chapter ends with learning aids consisting of:

1. *Chapter Summary.*
2. *Key Terms List.* This is a list of all new terms defined in the chapter. Appendix D is a glossary that lists all key terms in the text along with their definitions.
3. *Chapter Self-Test*—with solutions so you can test yourself on your understanding of the chapter as a whole.
4. *Practice Program.* A full program is illustrated. We recommend you read the definition of the problem and try to code the program yourself. Then compare your solution to the one illustrated.
5. *Review Questions.* These are general questions that may be assigned by your instructor for homework. They include questions that require you to access the Internet for reference.
6. *Debugging Exercises.* These are program excerpts with errors in them. You are asked to correct the coding. The errors highlighted are those commonly made by students and entry-level programmers.
7. *Programming Assignments.* The assignments appear in increasing order of difficulty. They include a full set of specifications similar to those that programmers are actually given in the "real world." You are asked to code and debug each program using test data. You will need to either create your own test data or receive a set from your instructor.

   Programming Assignments in each chapter include at least one interactive program (designated with an icon) and a maintenance program that students will need to modify or update.

A syntax reference guide and a disk containing all programs illustrated in the book also accompany this text. The disk also contains data sets for all Programming Assignments. Alternatively, both the syntax reference guide and the disk can be downloaded from the Internet.

INSTRUCTIONAL AIDS

INSTRUCTOR SUPPLEMENTS

1. INSTRUCTOR'S MANUAL by Connie Daniel, Dakota State University. This manual contains Chapter Objectives and Lecture Outlines. It also contains Solutions to Review Questions (which include True-False, Fill-in-the-Blank, General Questions, Interpreting Instruction Formats, and Suggestions for Validating Data), and Solutions to Debugging Exercises.
2. SOLUTIONS TO PROGRAMMING ASSIGNMENTS by James P. Ley, University of Wisconsin, Stout. This manual contains hard-copy solutions to all the Programming Assignments from the text. All the programs have been compiled and executed. The sample input and corresponding output are provided as well. The manual also comes packaged with a disk that contains solutions to the Programming Assignments, copies of the Practice Programs, and the data sets along with a README.TXT file.
3. TEST BANK by Diane Fischer, Dowling College. The Test Bank contains over 1000 questions. True-False, Fill-In, Short Answer, Multiple Choice, and Problem-Solving questions provide an excellent resource for instructors who create their own exams.
4. COMPUTERIZED TEST BANK. This supplement provides an electronic version of the printed Test Bank that enables instructors to customize material when creating exams for students.
5. POWERPOINT SLIDES by Richard Baum, DeVry Institute. The PowerPoint slides consist of figures and illustrations from the text combined with outlines of key concepts from each chapter. They are designed to enhance lectures and serve as a study tool for students.

STUDENT SUPPLEMENTS

1. STUDENT PROGRAM AND DATA DISK by James P. Ley, University of Wisconsin, Stout. This disk is packaged in the back of the textbook and contains the Practice Programs from the text along with the data for the Practice Programs and the Programming Assignments.
2. SYNTAX REFERENCE GUIDE by Nancy Stern and Robert A. Stern. This Syntax Guide is a brief, standalone booklet that students can use for quick and convenient reference. It is packaged at no cost in the back of all new copies of the text.

Four PC compilers are available with the text:

- Micro Focus Personal COBOL for Windows with a *Getting Started* manual prepared by Reed Doke.
- Micro Focus NetExpress 3.0 University Edition. The Micro Focus compilers are distributed by John Wiley & Sons only to colleges and universities in North America. To obtain these compilers *outside of North America,* please contact Merant Academic Programs at Micro Focus, 701 East Middlefield Road, Mountain View, CA 94043 or academics@merant.com.
- An educational version of Ryan McFarland's RM/COBOL-85. This is packaged with a student manual prepared by James Janossy.
- A full version of Fujitsu COBOL on CD with all documentation on the CD.

To obtain review copies of the compilers, please contact your local Wiley sales representative or call Wiley at 1-800-225-5945.

For online help, an email address (techhelp@wiley.com) is available for faculty only who may experience difficulty installing or using these compilers. If you are a student, please ask your instructor for help.

COBOL Web Site

Wiley's Web site for this book is www.wiley.com/cobol/; it contains late-breaking information as well as supplements that can be downloaded, and linkages to other COBOL sites.

The reviewers who provided many helpful suggestions throughout the development of this project are acknowledged on page x, along with all of those who helped bring this project to fruition.

We update our programming texts every few years and welcome your comments, criticisms, and suggestions. We can be reached c/o:

Nancy Stern
Robert A. Stern
BCIS Department
Hofstra University
Hempstead, NY 11550

Our Internet address is acsnns@hofstra.edu.

Nancy Stern
Robert A. Stern

ACKNOWLEDGMENTS

Our special thanks to (1) the following individuals at John Wiley & Sons: Beth Lang Golub, Editor; Joe Heider, Publisher; Ann Berlin, Vice President of Production and Manufacturing; Marian Provenzano, Development Editor; Jeanine Furino, Full Services Manager; Cynthia Rhoads, Assistant Editor; (2) Shelley Flannery, Copy Editor and Proofreader; (3) Jim Ley, University of Wisconsin at Stout, Supplements Coordinator, for solutions to the Programming Assignments and for invaluable suggestions; (4) Greg Wingerd, Senior Software Engineer at Logicon, Inc., for numerous useful suggestions; (5) Rosann Kelly, Hofstra University, for technical proofreading; (6) Ed Burke, Lorraine Burke, and Erica Smythe of HRS for design specifications and project management; and (7) Carol L. Eisen for her invaluable assistance in the preparation of the manuscript.

We thank the following reviewers for their many helpful suggestions: Elizabeth R. Bajek, DeVry Institute of Technology; Anthony Basilico, Community College of Rhode Island; Richard H. Baum, DeVry Institute of Technology; Louis J. Berzai, University of Notre Dame; Annette Easton, San Diego State University; Kay H. Elliott, Angelo State University; Carolyn R. Harris, Midwestern State University; Dolores Kalayta, DePaul University; David F. Kephart, Illinois State University; Dennis Kirlin, Iowa Western Community College; Janice Knapp, University of North Carolina; James P. Ley, University of Wisconsin, Stout; Mark Meysenburg, Metropolitan Community College; Alan M. Przyworski, DeVry Institute of Technology, Atlanta Campus; Mohan P. Rao, Texas A&M University, Kingsville; Cindy Riemenschneider, University of Arkansas; Dr. Sam Ryan, Baruch College/CUNY; Roseanne Witkowski, Orange County Community College; Sheri Wright, Iowa Western Community College.

A special word of thanks to Hofstra University for giving us the opportunity to experiment with some new ideas and techniques, and to our students, whose interesting and insightful questions helped us improve our pedagogic approach.

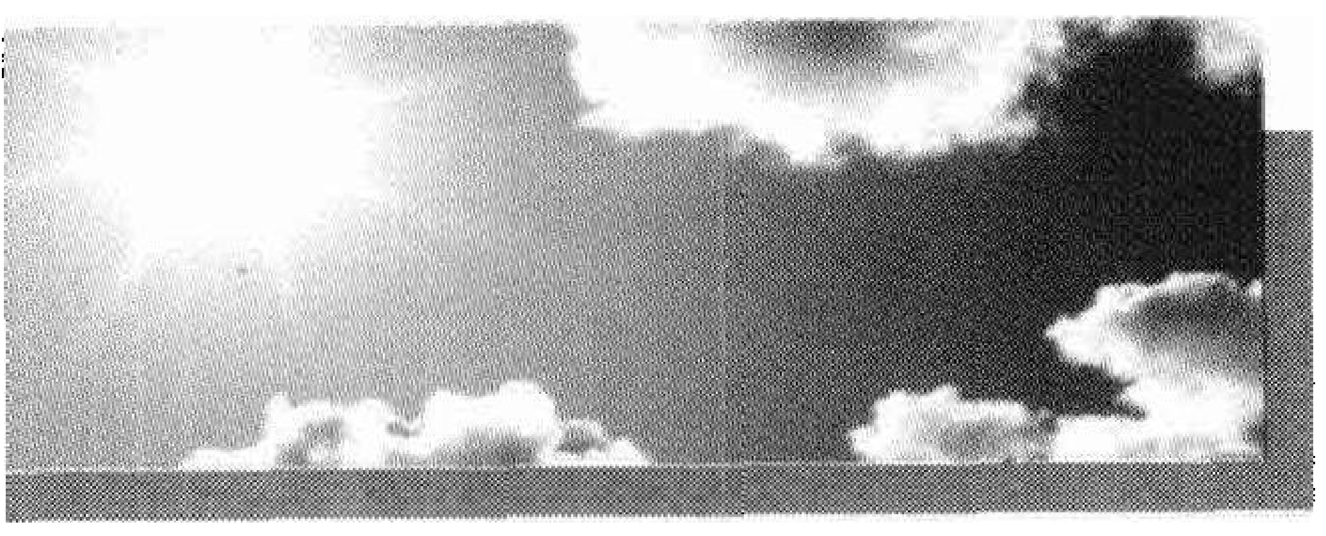
N. S., R. A. S.

CONTENTS

Accompanying this text are a COBOL Syntax Reference Guide, coding sheets, and Printer Spacing Charts.

This text is available with four compilers: (1) Fujitsu COBOL with on-line documentation; (2) Micro Focus Personal COBOL for Windows with a *Getting Started* manual; (3) Micro Focus NetExpress 2.0 University Edition; and (4) RM/COBOL-85 with a *Getting Started* manual.

UNIT I The Basics



CHAPTER 1

An Introduction to Structured Program Design in COBOL

OBJECTIVES

To familiarize you with

1. Why COBOL is such a popular business-oriented language.
2. Programming practices and techniques.
3. A history of how COBOL evolved and the use of the current ANS standard versions of COBOL.
4. An overview of the four divisions of a COBOL program.

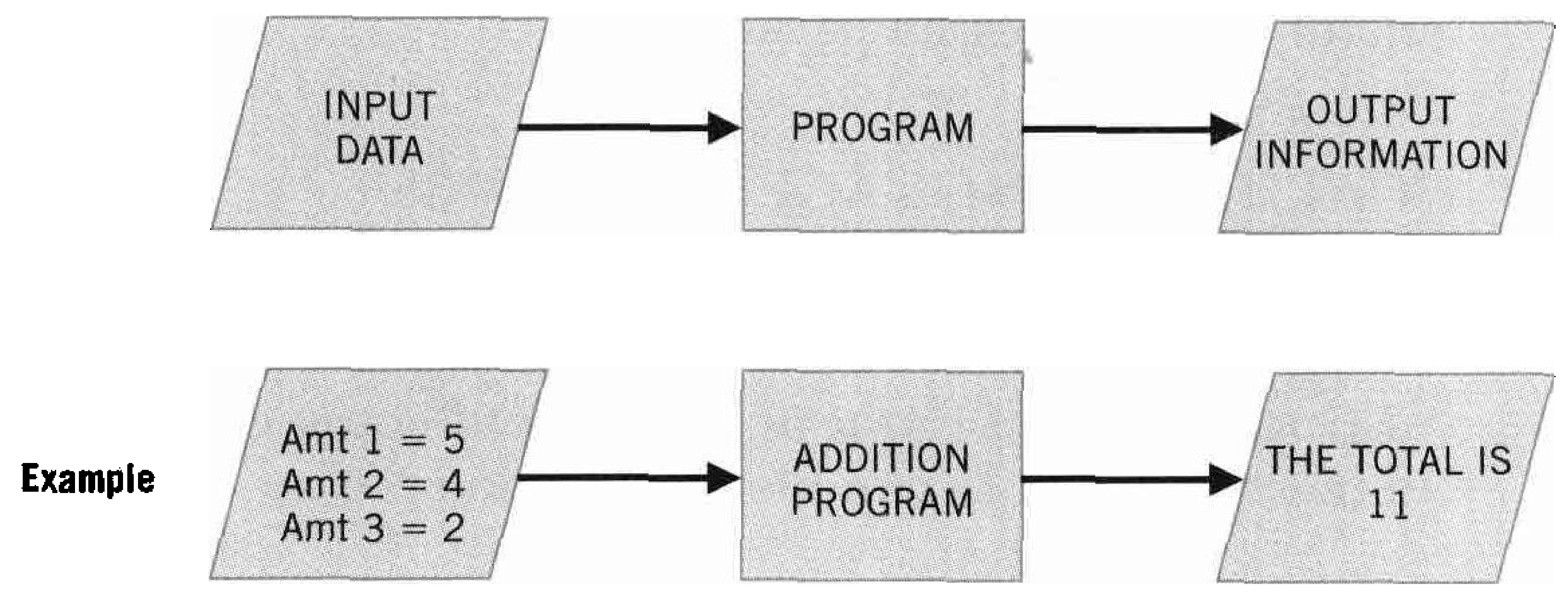
CONTENTS

COMPUTER PROGRAMMING: AN OVERVIEW

TYPES OF COMPUTER PROGRAMS

A **program** is a set of instructions that enable a computer to process data. There are two types of computer programs: **operating system programs**, which control the overall operations of the computer, and **applications programs**, which actually perform tasks required by users. The term used to describe all types of programs is called **software**.

An applications program operates on **input** data and converts it to meaningful **output** information. The following is an illustration of how a computer processes data:



A computer can process data only as efficiently and effectively as it is programmed.

The set of instructions in an applications program is written by a computer professional called an **applications programmer** or **software developer**.

Applications Programs

As noted, an applications program is written by an applications **programmer** or software developer to provide users with the information they need. Some applications programs are written to obtain a quick solution to a one-time problem; others are written to be run periodically on a regularly scheduled basis. An applications program to display the average grade for a set of exams entered as incoming data would be an example of a one-time job. An applications program to print student transcripts each semester would be an example of a program that is run periodically on a regular basis.

In general, applications programs read input, process it, and produce information or output that the user needs. Applications programs are generally used to computerize business procedures. A set of computerized business procedures in an application area is called an **information system**.

Interactive vs. Batch Processing

Some applications are processed interactively, as the data is transacted, while other data is collected and processed later, in batches. Interactive applications typically accept input data from a PC, workstation, or terminal. The input is processed immediately and the output is displayed on a screen and/or printed. This type of interactive processing is used when data must be current at all times and output is required immediately after processing. A Point-of-Sale system in which receipts are computer generated when sales are transacted is an example of an interactive application; both the accounts receivable and inventory files are updated as a result of a transaction.

Other applications process large volumes of input at periodic intervals. Payroll procedures used to update the master collection of payroll information prior to printing pay checks are often performed in batch mode at periodic intervals. We will see that COBOL is ideally suited for both interactive and batch processing applications.

When an applications program is written for a specific user it is called a **customized program**. COBOL is ideally suited as a language for writing customized applications programs. Although we will focus on COBOL for applications programming in this text, we now provide a brief description of another type of applications software called an applications package.

If the tasks to be performed by a program are relatively standard, such as preparing a budget, an **applications package** might be purchased as an alternative to writing a customized program in a language such as COBOL. Such packages are sold by software vendors. Typically, documentation is provided by the manufacturer in the form of a user's manual, which explains how to use the package. For example, Excel is a widely used package for applications such as budgeting, scheduling, and preparing trial balances.

If a package exists that can be used *as is* for an application, purchasing it will almost always be cheaper and easier than writing a customized program. But if an application has special requirements, then writing a customized program may be preferable to modifying an existing package.

Machine Language Programs

All programs to be executed by the computer must be in **machine language**. It would be very tedious and cumbersome for the programmer or software developer to code instructions in this form. He or she would need to reference actual addresses or locations in memory and use complex instruction codes.

Symbolic Programs

Since programming or software development in machine language is so difficult, programming languages have evolved that enable the programmer to write English-like or symbolic instructions. However, before symbolic instructions can be executed or run, they must be translated or **compiled** by the computer into machine language. The computer itself uses a translator program or **compiler** to perform this conversion into machine language.

There are numerous **symbolic programming languages** that can be translated into machine language. COBOL is one such language that is used extensively for commercial applications. Other symbolic programming languages include Visual Basic, Pascal, C, and C++.

THE APPLICATIONS PROGRAM DEVELOPMENT PROCESS

The process of developing programs is similar for all applications regardless of the symbolic programming language used. An overview of the steps involved in the program development process follows. Each of these steps will then be discussed in detail.

PROGRAM DEVELOPMENT PROCESS

1. Determine Program Specifications
   Programmers, along with systems analysts who are responsible for the overall computerized design of business procedures, work with users to develop program specifications. Program specifications include input and output layouts describing the precise format of data along with the step-by-step processing requirements for converting input to output.
2. Design the Program Using Program Planning Tools
   Programmers use design or program planning tools such as flowcharts, pseudocode, and hierarchy charts to help map out the structure and logic of a program before the program is actually coded.
3. Code and Enter the Program
   The programmer writes and then keys or enters the source program into the computer system using a keyboard.
4. Compile the Program
   The programmer makes certain that the program has no rule violations.
5. Test the Program
   The programmer develops sample data, manually "walks through" the program to see that it generates the correct output, and then uses the program to operate on the data to ensure that processing is correct.
6. Document the Program
   The programmer writes procedure manuals for users and computer operators so they can run the program on a regularly scheduled basis.

Most novices believe that computer programming begins with coding or writing program instructions and ends with program testing. You will find, however, that programmers who begin with the coding phase often produce poorly designed or inadequate programs. The steps involved in programming should be developmental, where coding is undertaken only *after* the program requirements have been fully specified and the logic to be used has been carefully planned.

Moreover, there are steps required *after* a program has been coded and tested. Each program must be documented with a formal set of procedures and instructions that specify how it is to be used. This **documentation** is meant for (1) those who will be working with the output, (2) computer operators who will run the program on a regularly scheduled basis, and (3) maintenance programmers who may need to make modifications to the program at a later date.

Determine Program Specifications

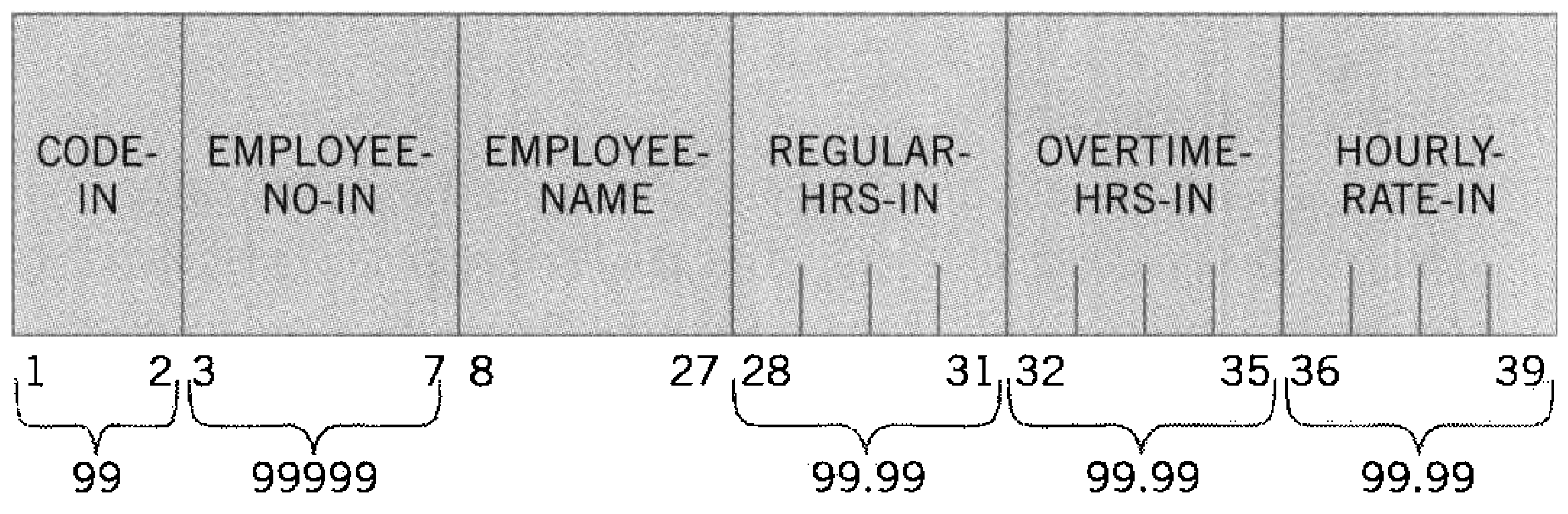
When a company decides to computerize a business application or information system such as payroll or accounts receivable, a systems analyst or a software developer is typically assigned the task of designing the entire computerized application. This systems analyst works closely with users to determine such factors as output needs, how many programs are required, and input requirements. A **user** is the businessperson who, when the application is computerized, will depend on or use the output.

When a systems analyst decides what customized programs are required, he or she prepares **program specifications** to be given to the programmers or software developers so that they can perform their tasks. Typically, the program specifications consist of:

1. **Record layout forms** to describe the formats of the input and output data on disk or other storage medium. Figure 1.1 illustrates two examples of record layouts. (We will use version (b) for most of our illustrations.) They indicate:
   a. The data items or field names within each record.
   b. The location of each data item within the record.
   c. The size of each data item.
   d. For numeric data items, the number of decimal positions. For example, 99.99 is a four-digit field with two integer and two decimal places. See Figure 1.1*a*.
   e. In some organizations, standard names of the fields to be used in a program are specified on the record layouts. In other organizations, names of fields are assigned by the programmer or software developer. In Figure 1.1 we use fields whose precise COBOL names have been defined by the programmer.
2. **Printer Spacing Charts** for printed output. Printed output requires a format not typically needed for other types of output:
   a. Headings are usually printed that contain report and page titles, dates, page numbers, and so on.
   b. Data must be spaced neatly across the page, allowing for margins.
   c. Sometimes additional lines for error messages or totals are required.

Figure 1.1 Sample record layouts.

(a)



(b)

Employee Record Layout			
Field	**Size**	**Data Type**	**No. of Decimal Positions (if Numeric)**
CODE-IN	2	Numeric	0
EMPLOYEE-NO-IN	5	Numeric	0
EMPLOYEE-NAME	20	Text	
REGULAR-HRS-IN	4	Numeric	2
OVERTIME-HRS-IN	4	Numeric	2
HOURLY-RATE-IN	4	Numeric	2