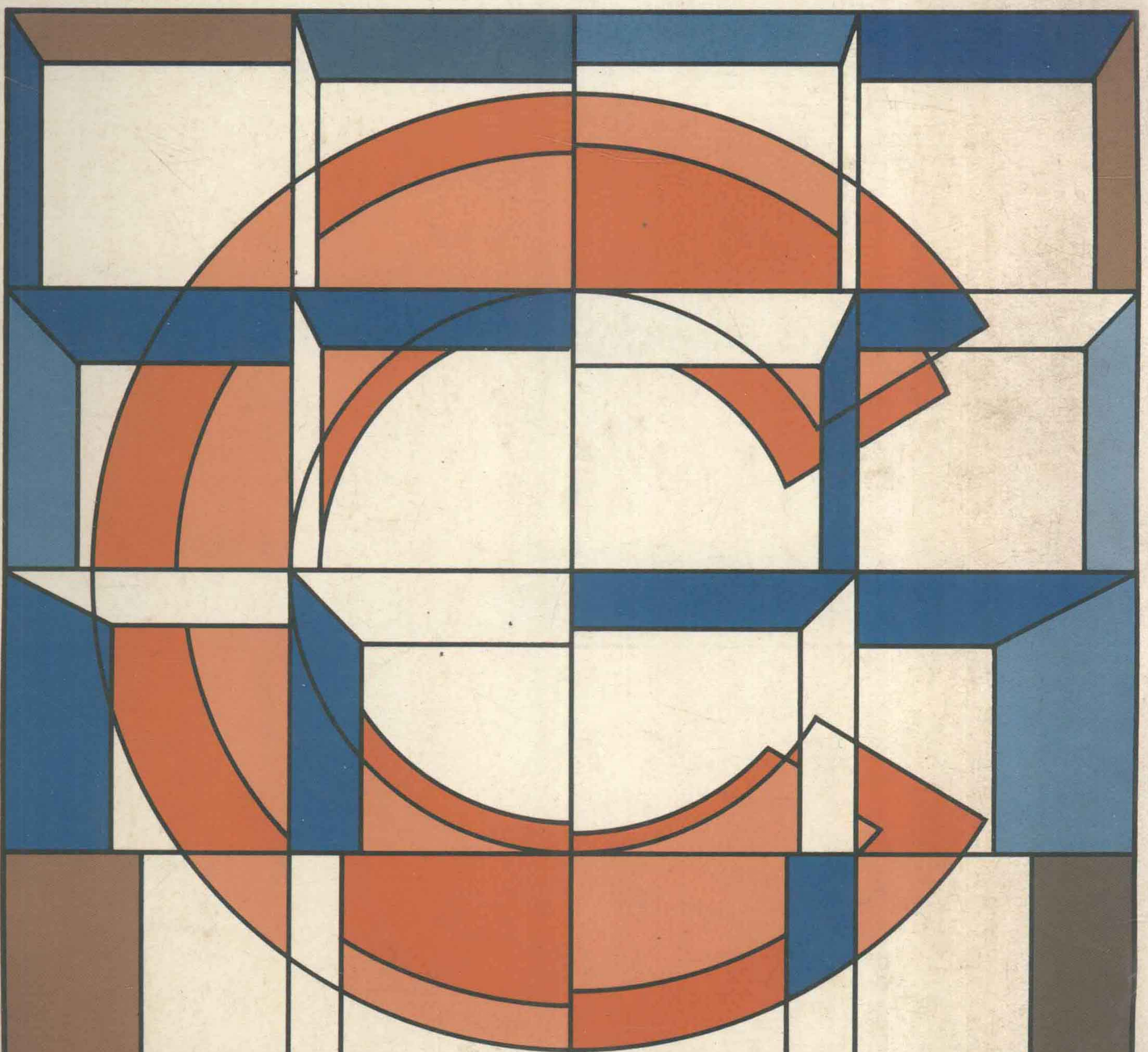
Robert T. Grauer
Marshal A. Crawford

Structured COBOL
A Pragmatic Approach



STRUCTURED COBOL: A PRAGMATIC APPROACH

ROBERT T. GRAUER

University of Miami

MARSHAL A. CRAWFORD

Programming Consultant

PRENTICE-HALL, INC., Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging in Publication Data

GRAUER, ROBERT T (date)
Structured COBOL: A Pragmatic Approach

(Prentice-Hall software series)
Previous ed. (c1978) published under title:
COBOL.
Includes bibliographical references and index.
1. COBOL (Computer program language)
2. Structured programming. I. Crawford,
Marshal A., 1943– II. Title. III. Series.
QA76.73.C25G7 1981 001.64'24 80–22283
ISBN 0–13–854455–7

To our families

Prentice-Hall Software Series
Brian W. Kernighan, advisor

Editorial/production supervision by Kathryn Gollin Marshak
Interior design by Nancy Milnamow
Cover design by Alon Jaedicker
Manufacturing buyer: Joyce Levatino

Printed in the United States of America

10 9 8 7 6 5 4

Prentice-Hall International, Inc., *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall of Canada, Ltd., *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*

PREFACE

We are indebted to the many instructors who have adopted our earlier work, *COBOL: A Pragmatic Approach,* and thereby inspired this new book. We further acknowledge the numerous reviews and thoughtful comments that became the basis for this new approach.

The change of greatest importance is that *every* illustrative program is structured. This contrasts with the earlier approach of beginning with an unstructured program in an effort to get students on the machine as quickly as possible. We still espouse the early "hands on" philosophy, but no longer introduce unstructured programs only to later reeducate the reader. This comes from our own as well as the (now) widespread acceptance of structured programming; in short, do it right from the beginning.

In addition, we are happy to report the following changes and/or additions that have been incorporated into *Structured COBOL: A Pragmatic Approach.*

1. Greater attention to pseudocode and hierarchy chart with decreased emphasis on the traditional flowchart.
2 Inclusion of material on top down testing, stepwise refinement, and the structured walk-through.
3. Strict adherence to the ANS 74 standard. (While some IBM extensions and/or deviations are included, these discussions *explicitly* state any deviation from the standard.)
4. Revision of most of the original COBOL illustrations to reflect our own increased awareness and attention to programming style.
5. Inclusion of over 20 additional programming projects (at the end of the appropriate chapters) for classroom assignment.
6. Inclusion of an appendix on Report Writer, as this once ignored facility appears to be on an upswing.
7. Expansion of Chapter 18 (OS JCL) to include material on SORT, subprograms, and MVS. Updating of Chapter 17 (DOS JCL) to include material on DOS/VS.
8. Expansion of Chapter 13 (ABEND debugging) to include the STATE and FLOW options.
9. Expansion of the discussion on indexed files to include COBOL coding differences between VSAM and ISAM implementation. The former adheres to the ANS 74 standard; the latter does not.

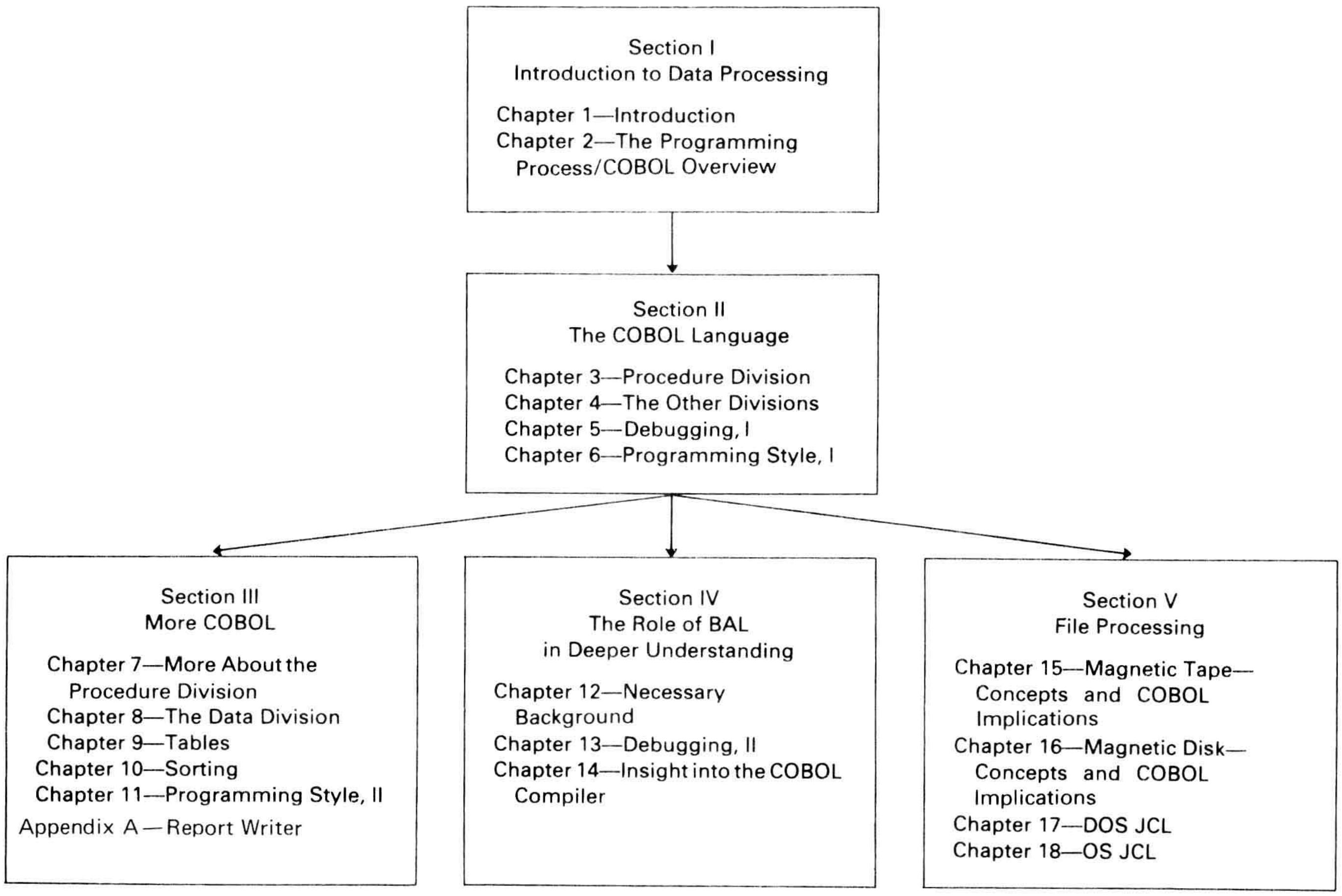
Why another book on COBOL? In recent years, universities have been criticized for failing to provide computer science and business graduates who are sufficiently versed in commercial data processing. This statement is partially justified for two reasons:

1. Most COBOL courses provide only "textbook" coverage and are sorely lacking in practical, i.e., commercial emphasis. Such subjects as JCL, file processing, debugging, structured programming, documentation, standards, and testing are glossed over or missed entirely.
2. College curricula traditionally treat COBOL and BAL in separate courses and do not provide an adequate link between the two. Although the COBOL programmer can and does exist without knowledge of Assembler, even a superficial understanding promotes superior capability to write efficient COBOL and is invaluable in debugging.

The primary objective of this book is to bridge the gap between traditional university curricula and the needs of industry. We address ourselves directly to the preceding statements and attempt to produce the well-rounded individual who can perform immediately and effectively in a third or fourth-generation environment.

The scope of the book is extensive, ranging from an introduction to data processing, to maintaining sequential and nonsequential files. A student may use this book without any previous exposure to data processing. Students with limited knowledge of COBOL can also use it since complete coverage will require two semesters. The text is modular in design so that Sections III, IV, and V may be covered in any order after Sections I and II are completed (see the accompanying diagram).

Modular Organization


Section I
Introduction to Data Processing
Chapter 1—Introduction
Chapter 2—The Programming Process/COBOL Overview
Section II
The COBOL Language
Chapter 3—Procedure Division
Chapter 4—The Other Divisions
Chapter 5—Debugging, I
Chapter 6—Programming Style, I
Section III
More COBOL
Chapter 7—More About the Procedure Division
Chapter 8—The Data Division
Chapter 9—Tables
Chapter 10—Sorting
Chapter 11—Programming Style, II
Appendix A—Report Writer
Section IV
The Role of BAL in Deeper Understanding
Chapter 12—Necessary Background
Chapter 13—Debugging, II
Chapter 14—Insight into the COBOL Compiler
Section V
File Processing
Chapter 15—Magnetic Tape—Concepts and COBOL Implications
Chapter 16—Magnetic Disk—Concepts and COBOL Implications
Chapter 17—DOS JCL
Chapter 18—OS JCL

In addition to the material on standard COBOL, there are two chapters devoted to debugging, two to JCL, two to programming style, two to file processing, and two on the role of BAL in better understanding COBOL. Although the JCL and BAL chapters pertain directly to IBM systems, ANS 74 COBOL is emphasized, so that the majority of the text relates to non-IBM installations as well. Inclusion of the IBM material, however, should enable the book to be used as a "programmer's guide" in that it contains a wealth of information that is not usually found in one place.

The authors wish to thank Karl Karlstrom of Prentice-Hall for making possible our entry into the world of publishing, Steve Cline, our editor, and Kathryn Marshak, our production editor. We thank our principal reviewers, Dr. Thomas DeLutis of Ohio State University, Dr. Jan L. Mize of Georgia State University, for their help and continued encouragement. Steve Shatz and Art Cooper are to be commended for their thoroughness in proofreading the galleys. We appreciate Ed Ramsey's help with some of the COBOL listings. We thank our many colleagues for their fine suggestions; these include Ken Anderson, Peter Baday, Jeff Borow, Les Davidson, Don Dejewski, Giselle Goldschmidt, Sam Ryan, Sue and Steve Wain, and anyone else whom we inadvertently omitted. Finally, we thank our typists, Francie Makoske and Deborah Miller, whose ability to interpret our scratches on yellow pads never ceased to amaze.

ROBERT T. GRAUER
MARSHAL A. CRAWFORD

CONTENTS

Section I

INTRODUCTION TO DATA PROCESSING

Chapter 1

INTRODUCTION

OVERVIEW

This book is about computer programming. In particular it is about COBOL, a widely used commercial programming language. Programming involves the translation of a precise means of problem solution into a form the computer can understand. Programming is necessary because, despite reports to the contrary, computers cannot think for themselves. Instead they do *exactly* what they have been instructed to do, and these instructions take the form of a computer program. The advantage of the computer stems from its speed and accuracy. It does not do anything that a human being could not if he or she were given sufficient time.

All computer applications consist of three phases: input, processing, and output. Information enters the computer, it is processed (i.e., calculations are performed), and the results are communicated to the user. Input can come from punched cards, magnetic tape or disk, computer terminals, or any of a variety of other devices. Processing encompasses the logic to solve a problem, but in actuality all a computer does is add, subtract, multiply, divide, or compare. All logic stems from these basic operations, and the power of the computer comes from its ability to alter a sequence of operations based on the results of a comparison. Output can take several forms. It may consist of the ubiquitous 11 × 14⅞ computer listing or printout, or it may be payroll checks, computer letters, mailing labels, magnetic tape, punched cards, etc.

We shall begin our study of computer programming by describing punched card input and printed output in some detail. We shall consider the structure of a computer and contrast machine- and problem-oriented languages. We shall pose a simple problem and develop the logic and COBOL program to solve it. The rapid entrance into COBOL is somewhat different from the approach followed by most textbooks, but we believe in learning by doing. There is nothing very mysterious about COBOL programming, so let's get started.

PUNCHED CARD INPUT

The punched card (Figure 1.1) has been around a long time. Its development was motivated by the U.S. Constitution (that is not a typographical error). Our Constitution requires that a federal census be taken every ten years. As the country expanded, processing of census data consumed increasing amounts of time (three years for the 1880 census), and the government needed a faster way of tabulating data. Herman Hollerith introduced the 80-column card in the late 1880s, and it has been with us ever since. (Mechanical devices, which were not computers, were available in

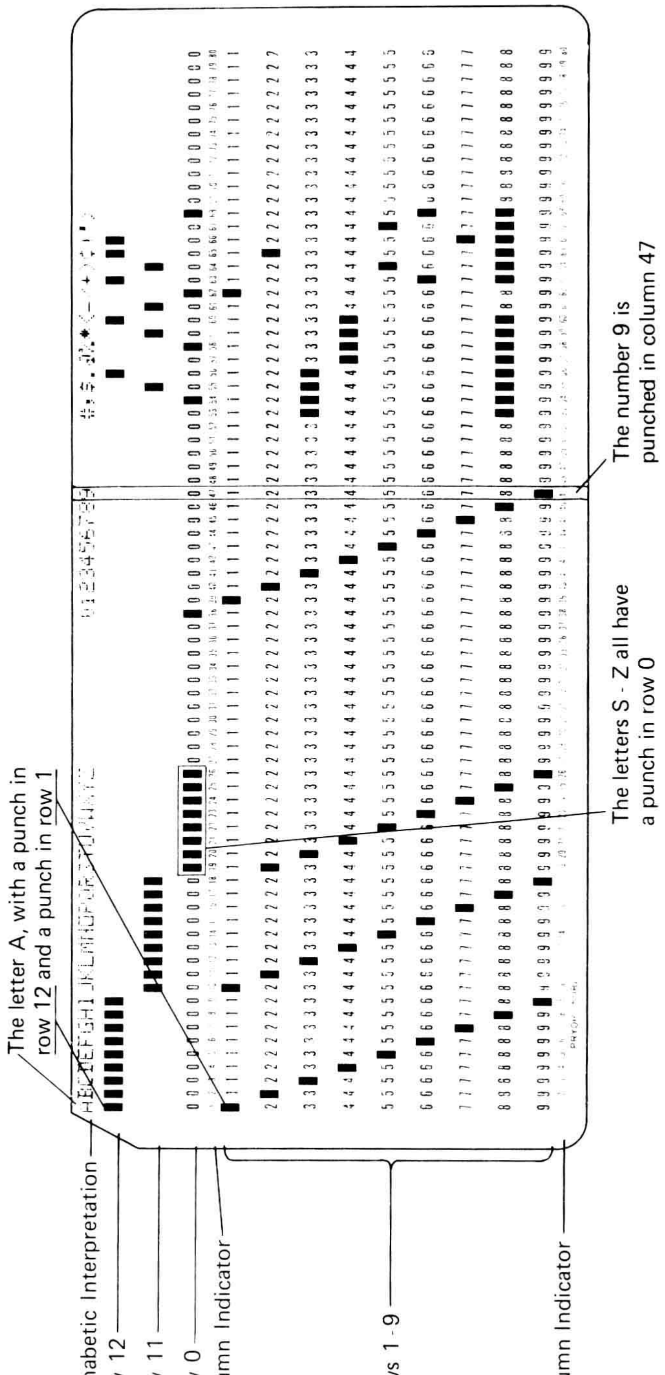

Alphabetic Interpretation
Row 12
Row 11
Row 0
Column Indicator
Rows 1 - 9
Column Indicator
The letter A, with a punch in row 12 and a punch in row 1
The letters S - Z all have a punch in row 0
The number 9 is punched in column 47

FIGURE 1.1 *The 80-Column Card*

the nineteenth century to process these cards.) Another interesting sidelight is the size of the punched card; it has the dimensions of the dollar bill of 1880.

The punched card has 80 vertical columns, each of which consists of 12 rows and contains a single character. Each character has its own unique combination of row punches. The letter A, for example, has a punch in row 12 and row 1 (see Figure 1.1). The letter B has a punch in row 12 and row 2. In Figure 1.1, A and B appear in columns 1 and 2, respectively. The upper three rows of the card (rows 12, 11, and 0) are known as zones, and the other rows as digits. Every letter has two punches: one zone and one digit. The letters A through I all have the same zone, i.e., row 12. The letters J through R all have a punch in row 11 and S through Z in row 0. The numbers 0 through 9 contain a single punch in the appropriate row. The bottom edge of the card indicates the column. The column indicator, in conjunction with the alphabetic information at the very top, indicates what is punched where; e.g., column 47 contains the number nine.

As a test of your understanding, what character is punched in column 26 of Figure 1.1? (Answer: Z.) What punches are required for the letter X? (Answer: row 0 and row 7.) In what column does X appear? (Answer: column 24.)

The computer is colorblind, and hence it does not matter if information is punched on red, white, or blue cards. Some installations, however, require that the first or last card in a deck be a specified color. This is to delineate one deck from another and is totally for human convenience. The very top edge of the card in Figure 1.1 interprets the information on the card. Again this is for human convenience only. The card reader senses the holes that are punched and does not refer to the interpreted information. Indeed, the latter need not be present at all.

PRINTED OUTPUT

The most widely used medium for computer output is the 11 × 14⅞ printout (alias listing, readout, etc.). Just as a computer must be told which card columns contain incoming data, it must also be told where to print its output. The 11 × 14⅞ form typically contains 132 print positions per line and 66 lines per page. Figure 1.2 contains a print layout form commonly used by programmers to plan their output. As can be seen from Figure 1.2, "THIS IS A PRINT LAYOUT FORM" is to appear on the tenth line from the top, beginning in column 15 and extending to column 41.

It makes no difference to the computer if printing is on wide or narrow paper, single- or multiple-part forms, mailing labels, payroll checks, etc. The machine is only interested in knowing what information is to appear and where. This is accomplished via instructions in a program.

STRUCTURE OF A COMPUTER

A computer can be thought of as a collection of electronic devices that (1) accept data, (2) perform calculations, and (3) produce results. A functional representation is shown in Figure 1.3.

We have already spoken about input and output. Main storage, i.e., the computer's memory, stores instructions and data while they are being processed. The size of a computer is measured by the capacity of its memory. Large modern machines have memory capacities of several million characters.

The central processing unit (CPU) is the "brain" of the computer. It consists of an arithmetic and logical unit (ALU) and a control unit. The ALU actually executes instructions; i.e., it adds, subtracts, multiplies, divides, and compares. The control unit monitors the transfer of data between main storage, input/output (I/O) devices, and the ALU. It decides which instruction will be executed next and is the "boss" of the computer.

Speed of execution is another way in which computers are measured. The ALU of a modern machine can execute millions of instructions per second. The time to execute a single instruction is expressed in microseconds (millionths of a second) or nanoseconds (billionths of a second).

MACHINE VERSUS HIGHER-LEVEL LANGUAGES

Each computer has its own unique machine language tied to specific locations in its memory. Human beings, however, think in terms of problems and use quantities with mnemonic significance,