STRUCTURED BASIC PROGRAMMING

HARRY A. MORIBER

HARRY A. MORIBER

STRUCTURED BASIC PROGRAMMING

CHARLES E. MERRILL PUBLISHING COMPANY
A BELL & HOWELL COMPANY
COLUMBUS □ TORONTO □ LONDON □ SYDNEY

To Rositta, David, and Ann

Published by Charles E. Merrill Publishing Co.
A Bell & Howell Company
Columbus, Ohio 43216

This book was set in Goudy Old Style, Helvetica, and Universal Dot Matrix
Text design and production coordination: Ben Shriver
Cover design coordination: Cathy Watterson
Cover image: "Ray Tracing Spheres With Vases," animators Shaun Ho and Michael Collery for Cranston/Csuri Productions, Columbus, Ohio. The Ray Tracing algorithm simulates all the physics of light and shadow with reflective and refractive properties; the vases are created by a separate mapping technique. The image is an example of three-dimensional, solid-shaded raster graphics.

Library of Congress Catalog Card Number: 83-62900
International Standard Book Number: 0-675-20106-3
Printed in the United States of America
1 2 3 4 5 6 7 8 9 10—89 88 87 86 85 84

PREFACE

Structured BASIC Programming is designed for the first course in Basic Programming or as part of the programming component of an Introduction to Computers class. It can also be used by any individual outside the classroom who is seeking to learn and use BASIC programming. The text assumes no previous experience with BASIC or any other programming language. And while the text starts at a very elementary level, my goal has been to help people reach a level of proficiency where they can create useful and practical application programs.

This text has grown out of my many years of teaching the data processing courses at Greater Hartford Community College. What my students and I found lacking in other texts were programs that could be used outside the classroom. As instructors everywhere have found, teaching *motivated* students is the ideal—and abstract, mathematical examples simply failed to motivate my students. Therefore, *Structured BASIC Programming* has been built around a wide variety of realistic cases and problems which a student can use in the "real world." I hope that your students are as motivated by these examples as mine have been.

As an educator, I have tried to make this text as teachable as possible. I look forward to hearing about how well the text works in your class, as well as to receiving your suggestions for improving it. The highlights of *Structured BASIC Programming* are outlined in the following sections.

EMPHASIS ON STRUCTURED DESIGN PROBLEM-SOLVING

The text introduces the concepts of true structured problem solving and programming early in the text (chapter 2) and is consistent in reinforcing these principles throughout the text. This is done so that students will not pick up a programming style which must be corrected later in the course or in subsequent classes. Special attention to structured logic development is found throughout the text.

REALISTIC CASE STUDIES

These programs, included in every chapter from chapter 3 on, are designed to motivate students by showing them practical applications of the skills and concepts they are learning. Example cases include Inventory Control, Tax Calculations, Translating Languages, Charge Account Processing, Loan Amortization, Airline Reservations, and Sales Analysis. The cases are then used as a means for teaching those skills and principles so that students can create their own applied programs. Each case study is signalled by a "tab" visible at the top of the page.

REALISTIC END OF CHAPTER PROBLEMS

Problems for solution. Every chapter concludes with between seven and ten applied problems for solution, which are designed to test student retention of the chapter materials. These give students a chance to create their own practical programs, increasing their

problem-solving confidence. Examples include Carpet Costs, American to Metric Conversions, Computer Dating Evaluation, Salary Analysis, Mortgage Comparison, Break-Even Calculations, Appointment Calendar, and Job Placement, to name just a few.

Building Block problems. These additional five problems for solution (Inventory Control, Payroll, Personal Finances, Student Records Keeping, and Itemized Tax Deductions) are carried through every chapter of the text, each one growing finally to the size of a case study. By assigning one of these problems an instructor can reduce the amount of time spent keying in new codes. Students also learn by doing the important concepts of program modification and maintenance. The Building Blocks explicitly show the development and relationship among all the concepts and skills being taught throughout the text, and especially teach the advantage of structured design. Each Building Block—and its fellows in the other chapters—is signalled by a distinctively placed "tab" at the side of the page.

OTHER KEY FEATURES

Common errors and solutions tables. Concluding each chapter is a chart designed to help students avoid errors before programming or, if they occur, to quickly debug them afterwards.

Extensive coverage of tables, arrays, and files. Because of their importance in business, three full chapters and nine case studies are devoted to these key topics.

Appendix of BASIC commands. This chart is designed to assist the student in using one of several BASIC dialects, including IBM BASIC, Applesoft, Apple Business BASIC, TRS–80 BASIC, Vax 11 BASIC, minimal BASIC, Commodore BASIC and BASIC-PLUS (DEC), as well as the proposed ANSI BASIC.

Sound learning aids. Each chapter begins with a chapter outline and learning objectives and concludes with summaries and review questions as well as the problems already described. In addition, the text is heavily illustrated with flow charts, structure charts (VTOC's), and problem definition sheets, to provide as realistic a problem-solving environment as possible.

Computer program accuracy. All the case studies in the text were provided "camera ready" to the publisher. To ensure accuracy, they appear exactly as they did when I ran the programs.

Software available. Solutions to all the case studies are available in different formats. These will be provided free to adopters upon written request to the publisher.

TEACHING AIDS

The Instructor's Manual includes teaching objectives, solutions to all problems and Building-Block problems, and answers to end-of-chapter review questions. A separate test bank of 1,000 multiple-choice and true-false questions is organized according to type of question, number, chapter, learning objective, and level of difficulty. In addition to the print version, floppy disk versions are available which allow the instructor to retrieve test items according to the above criteria (as well as odd vs. even). This system is designed for the most flexible and pedagogically sound assembling of student exams. Also available to adopters and packaged with the test bank are a generous supply of transparency masters selected from the text.

ACKNOWLEDGMENTS

Many thanks are due to the original reviewers of the manuscript, who are responsible for so many refinements that appear in the text. These knowledgeable and helpful professionals are Jane Canty, of the DeVry Institute of Technology, Columbus, Ohio; Warren F. Buxton, from Maricopa Tech, Phoenix, Arizona; John B. Lane, of Edinboro State University, Edinboro, Pennsylvania; Herb Rebhun, from the University of Houston, Houston, Texas; and Jane Miley Danehy, State University of New York Agricultural & Technical College, Morrisville, N.Y.

Dr. Sukwant Sethi, Dr. Bob Heavilin, and Dr. Peter C. Loewenberg all provided help, support, and advice through the period in which the text was being prepared.

For their assistance in preparing the appendix, which compares and contrasts BASIC dialects, I would like to thank the following people and their respective companies: Ms. Sue Goodin, Apple Computer Company; Mr. Jim Totten, Digital Equipment Corporation; and Michael Epps, Commodore Computer Corporation.

Mr. Roy Levin of Farmington Valley Travel and Mr. Dennis Crean of Connecticut Poison Center deserve thanks for their help with the interesting airline reservations and poison control case studies and problems. Mr. Gene Sklandowski of Connecticut Bank & Trust was quite supportive with banking system principles.

Carolyn Muller, my unofficial research assistant skillfully wrote programs, drew VTOC charts, and offered a great deal of helpful advice and friendship. Karen Wasem and, again, Carolyn Muller helped write many of the solutions for the problems for solution which appear in the instructor's manual.

For Charles E. Merrill Publishing, Rich Wohl, my administrative editor, never lost confidence, and ever offered encouragement, great advice, and friendship. Also for Merrill, Jim Montgomery (St. Paul, Minnesota) my copy editor, my proofer Charles E. Cox (Pinole, California), and production coordinator Ben Shriver all earned a sincere vote of thanks.

(Still at Merrill) Jan Hall, Ann Mirels, Cathy Watterson, Marylou Zschach, Bev Small, Alan Borne, George Owen, and George McCann all supported the text through the many stages of production and promotion. Finally, thanks to John Hayes, Merrill's New England sales representative, who indirectly started this project back in May of 1979 by asking me what kind of programming text I would like to see on the market.

Debbie and Martin Page earned my undying gratitude by typing the many versions of the manuscript. A final thanks goes to my family, Rositta, David, and Ann, for their great patience and support.

THE CHARLES E. MERRILL INFORMATION PROCESSING SERIES

INTRODUCTION

Spencer (1983): *Illustrated Computer Dictionary*
Spencer (1983): *An Introduction to Computers: Developing Computer Literacy*
Spencer (1982): *Data Processing: An Introduction (2d Edition)*
Spencer (1982): *Data Processing: An Introduction with BASIC (2d Edition)*
Spencer (1981): *Introduction to Information Processing (3d Edition)*
Spencer (1976): *Computer Science Mathematics*

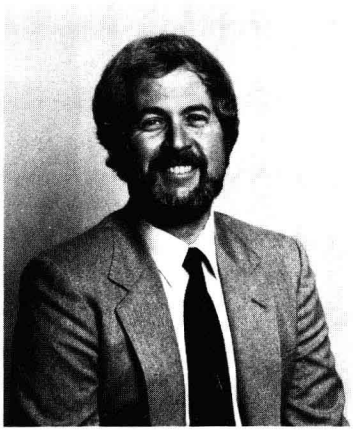
LANGUAGES

Moriber (1984): *Structured BASIC Programming*
Thompson (1981): *BASIC—A First Course*
Richards and Cheney (1981): *COBOL—A Structured Approach*

SYSTEMS/MIS

Thierauf (1984): *Effective Management Information Systems: Accent on Current Practices*
Reynolds (1984): *Introduction to Business Telecommunications*
Thierauf and Reynolds (1982): *Effective Information Systems Management*
Thierauf and Reynolds (1980): *Systems Analysis and Design*

ABOUT THE AUTHOR



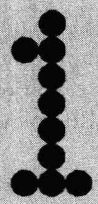
Harry A. Moriber has been Assistant Professor of Data Processing at Greater Hartford Community College since 1974, where he has taught courses in BASIC, COBOL, FORTRAN, Assembler, Systems Analysis and Design, and Microcomputers in Business, among other subjects.

Prior to teaching, the author accumulated a wealth of experience in the business sector and as a company commander in the United States Army. Moriber was a full-time operations research consultant for the Aetna Life and Casualty from 1971–74; from 1968–71 he was supervisor of the Nonmetallic Materials Group and statistician for Avco Lycoming Division; from 1965–68 he was a chemical engineer for Mobil Oil Corporation.

Educated at the University of Bridgeport, New York University, and the Polytechic Institute of New York, Moriber holds degrees in chemical engineering as well as business administration.

Structured BASIC Programming is the first in a series of four programming texts that the author has contracted to do with the Charles E. Merrill Publishing Company.

CONTENTS

CHAPTER 

INTRODUCTION TO COMPUTER PROGRAMMING

OUTLINE

LEARNING OBJECTIVES

Upon completion of Chapter 1, you will be able to

- ☐ Use standardized methods to define and develop computer programs.
- ☐ Understand the 5 steps in preparing a computer program.
- ☐ Understand the difference between instructions for a computer and instructions for humans.
- ☐ Use programmer tools and supplies to prepare VTOC charts and flowcharts.
- ☐ Interpret and use worksheets for input specifications, decisions and formulas specifications, output specifications, error recovery routine specifications, and validation procedure specifications.
- ☐ Use standard methods of program documentation.
- ☐ Understand, detect, and correct some system command errors, BASIC syntax errors and logic errors.
- ☐ Understand the differences in BASIC dialects.
- ☐ Enter and run the example BASIC programs on your own computer system.

An Overview

Many scientists and historians regard the late 1940s as the beginning of a sociological and cultural revolution as significant as the Industrial Revolution and the harnessing of electricity. This new revolution is due to electronic devices called computers. Scarcely one of our daily activities is not influenced in one way or another by a computer or computer technology.

The introduction in the early 1950s of large-scale computer sytems to big business and their subliminal effect on our lives is well documented. But the affordable, compact, and inexpensive microcomputer is destined to change the way we live, work, and play. Since the midseventies the computer has become an accepted and often vital part of many homes and small businesses. Today the estimated number of small-computer owners is in the tens of millions. Daily newspapers and weekly magazines include special sections on computers and their use, and publishers market hundreds of computer publications for hobbyists, business owners, scientists, and the general public.

Computers were for more than thirty years the prerogative of large businesses mainly because computer systems were prohibitively expensive to nearly everyone else. But today cost is practically no object. For example, a computer that rented five years ago for $50,000 to $100,000 per year can now be replaced by equivalent or better microcomputers that cost less than $15,000 to $20,000 to buy outright. Computer technology now brings such rapid cost benefits that people searching for the right computer for their needs must weigh the advantages of purchasing a suitable computer against waiting for a future machine that will do what they want even better and a lot cheaper. Scientists estimate that comparable computer systems decrease in price by one-half every two years, so that a system that sells for $5,000 today (a reasonable price for a complete functional computer system for home or business) will cost only about $1,000 in five years. These estimates are probably very conservative.

More important than the reduction in price are the enhanced capabilities of newer systems, brought about by improvements in hardware and especially software. Operation in many new systems has been so simplified that even a novice without experience or understanding of how a computer works can learn a few codes and get the computer to accomplish the most sophisticated accounting, mathematical, word processing, or business tasks. The computer has become a tool accessible to everyone—much the way the TV (whose electronics few of us understand) has become part of our way of life.

This development is no accident. Computer manufacturers long realized the potential market for machines that everyone (not only computer scientists or programmers) could use. With price reductions of computer system components—memory, logic units, input and output devices—the computer became affordable for many small businesses and individuals. The next and obvious step was to make the computer simple to operate, like the automobile. Few of us fully understand (or care to understand) how an electronic ignition system functions, how an automatic transmission works, or the difference between disk brakes and drum brakes. Is the product dependable? Will it get us where we want to go? Is the price right?

Computer manufacturers considered the same issues, and the result is the current revolution in microcomputer marketing.

The correct name for these computers is still a marketing problem. A "home" or "personal" computer connotes a limited machine for simple or fun applications. "Small" or "micro-" computers convey a lack of capacity or capability, which is certainly not the case. The right adjective will probably come in time.

Whatever we call them, one thing is certain: they are revolutionizing not only small businesses, which can now enjoy the cost reductions associated with automation, but the way we live and work.

Clearly, the microcomputer revolution frees many workers from boring, repetitious, and unpleasant tasks found in business, education, and industry; and yes, as you may have guessed, it displaces many workers from their jobs. But this revolution creates many

more jobs than it replaces, and many people find these new jobs more challenging, enjoyable, and financially rewarding.

More important, the microcomputer gives a long-needed boost to owners and operators of small businesses. It frees them from routine paperwork, records keeping, and manual accounting, and it permits tighter control of inventory management so crucial to the success of retail and manufacturing businesses. The microcomputer now gives management time: time to plan marketing strategies, to develop new products, to supervise production, and to investigate new and better manufacturing techniques.

Independent and federal government studies show that small business owners spend from one-third to one-half of their time in routine managerial functions, especially in preparing accounting and tax reports required by most states, municipalities, and the federal government. Small business failure ratios are tragic, with more than half of all new businesses failing before the end of their fifth year. Although sufficient statistics are not yet available to establish a trend, failure ratios should drop as microcomputers relieve owners and managers from nonproductive activities and allow them to implement the money-making and money-saving techniques used in big businesses for thirty years.

The BASIC Programming Language

BASIC, an acronym for Beginner's All-Purpose Symbolic Instruction Code, was developed at Dartmouth College between 1962 and 1965 as a simple teaching language for beginning programmers. It is usually provided as an interpretative language. That is, BASIC code is not translated as a complete program and then converted into machine code, but is interpreted line by line during execution, making it slightly less efficient than compiler languages. The simple syntax permits programs to be written with little memorization of rules and conventions. Also, BASIC is programmed and executed in an interactive time-sharing mode, usually on CRT terminals or microcomputers. Therefore both syntax and some logic errors are detected immediately and can be corrected quickly. The later, compiler versions of BASIC allow for faster program execution. In its early years BASIC was not efficient or powerful, with few built-in functions, limited array- (table) and alphabetic data- (string) handling capabilities, and interpreters that were slow compared with high-level compiler languages.

Within the last few years, however, BASIC has become one of the leading programming languages in business, science, and education because of the widespread popularity and proliferation of microcomputers, nearly all of which now have BASIC interpreters or compilers. See Figure 1-1. The language has been standardized somewhat,* so that programs written for one computer can usually run, with simple modifications, on another manufacturer's computer. Probably most significant, the BASIC instruction set has been expanded to accommodate most high-level mathematics, array processing, and alpha-numeric data processing. Built-in mathematic and array-processing functions in recent versions of BASIC rival those in so-called sophisticated compiler languages like FOR-TRAN. Combine this with data file manipulation to rival COBOL, and almost any scientific or business problem can be programmed in BASIC. For business applications BASIC, combined with excellent operating systems, permits character manipulation and file-handling methods including direct access, sequential access, and indexed sequential access (available in many manufacturers' versions). With these enhancements and newly included "structures" (for DO WHILE, DO UNTIL, and case control), BASIC has become a very versatile general-purpose language, easily learned and programmed. At many schools and colleges it is the introductory language, and for many business software systems it is the foundation language.

More powerful versions of BASIC still vary significantly from one manufacturer to another. When the ANSI standardizes BASIC, as it has FORTRAN and COBOL, these

*The American National Standards Institute (ANSI) expects to release specifications in 1984 for an enhanced BASIC.

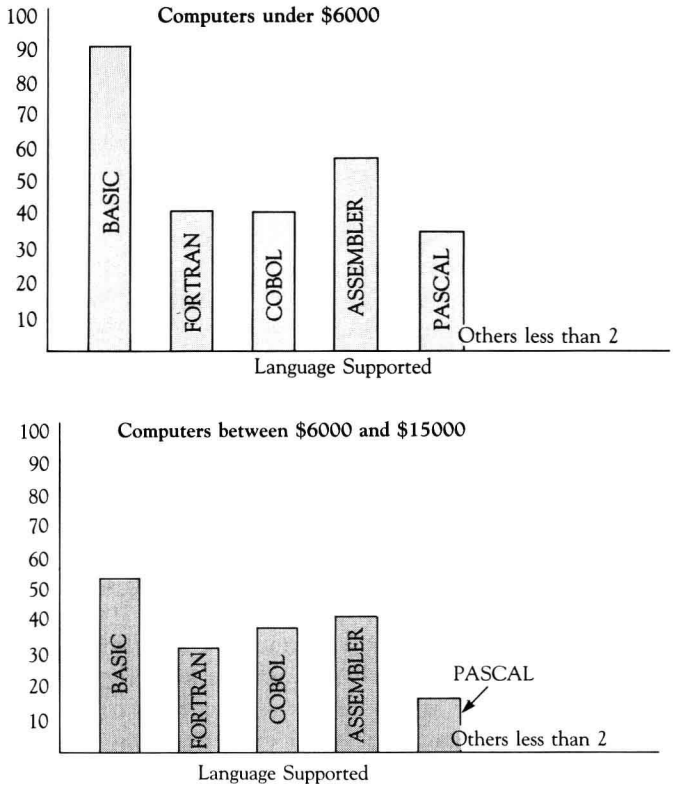


FIGURE 1-1.
Popularity of various computer languages.

Source for data: Popular Computing, September 1982. (Total of 56 manufacturers.)

problems will diminish. For now, most BASIC texts, including this one, treat the dialectal differences of BASIC by pointing out revisions needed to run programs on different computers. This is troublesome and distracting, but regard it as a short-term inconvenience and not as a serious drawback to using an otherwise powerful and comprehensive language.

BASIC's unstandardized history led to other problems. Students who learned BASIC as their first language often developed habits incompatible with good programming practice, and some carried these bad habits into their programming with other languages. FORTRAN and COBOL programmers are expected to write clear, logical, well-documented programs because business and industry standards dictate it. Industry standards for structured design were implemented quickly in COBOL programming environments after studies proved that COBOL programmers were averaging only 10 lines of useful code per day. Structured language standards improved this average tremendously, but the impact on BASIC is not yet apparent.

Only in recent years, with businesses buying and using microcomputers by the millions, did the need for logical, easily understood and modifiable BASIC programs develop. Many BASIC programs were formerly written with results as the only objective. In many cases programming consultants tried unsuccessfully to debug accounting and inventory control programs written by a staff programmer in BASIC. After hours of sorting through a maze of random GO TO statements, the consultants finally gave up and suggested the program be rewritten in a logical structured design. Such programming perpetuates BASIC's bad reputation. BASIC can conform to the principles of sound logical structured design, however, even in its less sophisticated versions.

In the words of Sigmund Freud, "Only when one understands the problem, can one solve it," we have the key to writing successful computer programs. We must know what we wish the computer to do before we instruct it.

Imagine giving a speech on a subject you knew little about, or directing a motorist to a part of town you've never been to or even heard of. Many new programmers nevertheless attempt to instruct the computer to solve problems they don't really understand. The results in wasted time and effort and money are disastrous.

STEPS IN PREPARING A COMPUTER PROGRAM

Some simple rules allow efficient computer problem-solving. These guidelines are summarized in the following list and detailed in this section. For those new to computers and programming, this section clarifies many misconceptions about what a computer can and cannot do.

1. *Define the problem* by studying and comprehending the input data, organization (structure, content, etc.), the outputs desired, and the equations or decisions (or both) needed to go from input to output. Cover every possibility. Assume nothing.
2. *Develop a method of solution*, called an algorithm, by drawing a pictorial or graphic representation of all the logical steps in their proper sequence, using standard forms and symbols.
3. *Code the algorithm* in a programming language chosen on the basis of the type of problem, the computer languages available, the computer equipment available, and the needs of your environment.
4. *Test the program* to ensure that the results are valid and accurate. Find and remove computer language errors, called bugs. Verify output results with hand-processed test data whenever possible.
5. *Document the program* so that others, even non-data processing people, can understand what the program does and how it does it. This is an important and frequently overlooked step, because many programs must be modified or used by someone other than the author.

Step 1: define the problem. Let's use this definition for a computer program:

DEFINITION

A computer program is a series of instructions that directs the computer to use available input data to solve a particular problem and produces useful information called output. This output may be a single numerical answer, a complex report, pictures or graphs, voice or music, or information on a variety of media.

Before we write these instructions, we must be clear about the following:

1. What input is available to the program? What data is needed? What form does the data take (alphabetic, numeric, alphanumeric)? From where is the data obtained (e.g., tape, disk, voice, terminal)? Is the data reliable, or will we need to include error routines in the program so that it will not terminate or produce useless output?
2. What output is desired from the program? Is it a simple numerical result? Is it a lengthy table or report? What information aside from the computer's results is needed to complete the output (titles, headings, special forms, etc.)? What are the possible output media and how will the optimum be selected?
3. What calculations (formulas and equations) or data manipulations are needed to obtain the output desired? Which calculations or routines can be performed by predefined functions, and which must be written by the programmer? Do the solutions require extreme accuracy or unusual precision? Or can good approximations through iterative