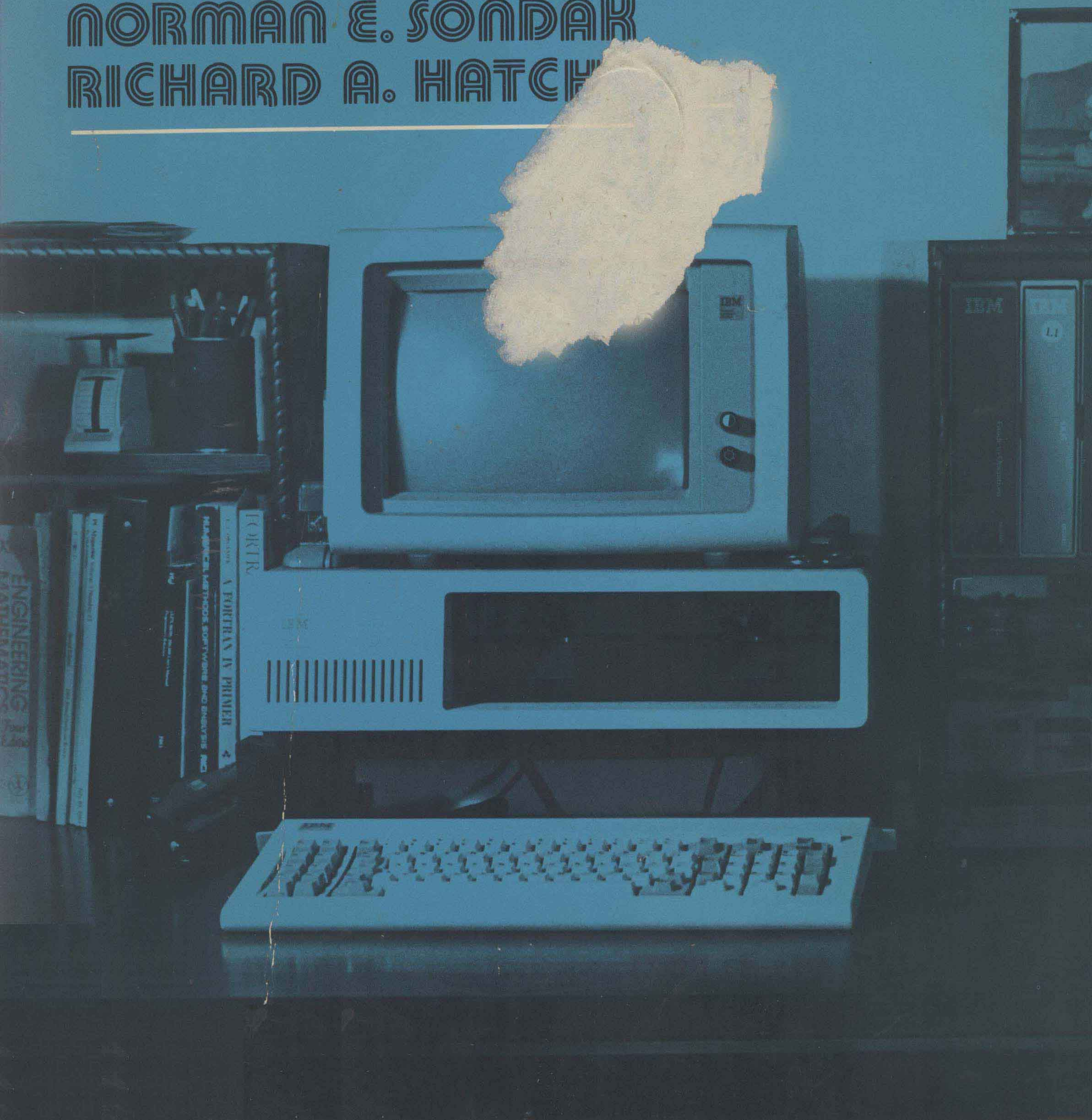
USING BASIC ON THE IBM PERSONAL COMPUTER

NORMAN E. SONDAK
RICHARD A. HATCH



USING BASIC ON THE IBM PERSONAL COMPUTER

Norman E. Sondak and Richard A. Hatch

Information Systems Department
San Diego State University

SRA®

SCIENCE RESEARCH ASSOCIATES, INC.
Chicago, Toronto, Henley-on-Thames, Sydney

A Subsidiary of IBM

Acquisition Editor	Pamela S. Cooper
Project Editor	Geoffrey T. Garvey
Copy Editor	Catherine Zeiss
Graphics and Composition	House of Graphics
Cover Design	Barbara Ravizza
Production Director	Arthur Kuntz

Credits

Reprinted by permission from BASIC (IBM Personal Computer Hardware Reference Library) © 1981 by International Business Machines Corporation:

Figure 2–3 (p. 13) Figures 10–1 and 10–2 (pp. 271 and 272)

Appendix C (p. 332)

Reprinted from USING BASIC ON THE CYBER by Norman E. Sondak and Richard A. Hatch. © 1982, Science Research Associates, Inc. Reprinted by permission:

Figure	Page	Figure	Page	Figure	Page
3–1	35	4–8	68	8–5	208
3–2	36	5–1	99	8–6	210
4–1	60	7–1	164	8–7	211
4–2	61	7–2	165	9–1	224
4–3	64	7–3	177	9–2	225
4–4	65	8–1	191	9–3	226
4–5	65	8–2	191	9–4	243
4–6	66	8–3	192		
4–7	67	8–4	193		

Library of Congress Cataloging in Publication Data

Sondak, Norman E.
Using BASIC on the IBM personal computer.

Includes index.
1. IBM Personal Computer—Programming. 2. Basic (Computer program language) I. Hatch, Richard A. II. Title III. Title: Using B.A.S.I.C. on the I.B.M. personal computer.
QA76.8.I2594S65 1985 001.64′2 84-13920
ISBN 0-574-21740-1

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

Preface to the Instructor

This book is an introductory text on programming in BASIC specifically designed for introductory data processing and information systems courses in colleges where student practice is done on IBM Personal Computers. Normally this text will provide the basis for about one-third of the student work in a standard course (3 or 4 units), or it will form the full content of an abbreviated programming course (1 or 2 units).

A complete description of the IBM Personal Computer BASIC language is provided. Students in all types of courses should study the core material comprising approximately the first half of the book. Instructors can select the most appropriate sections from the topics presented in the second half to customize text materials for specific courses. Ample material is provided to allow this book to be used for a full semester or two-quarter course in introductory programming in business, computer science, or engineering.

BASIC on the IBM PC provides an unusually powerful system for teaching students the rudiments of programming. The language meets the ANSI Minimal BASIC specifications, and it includes powerful extensions similar to those found in most microcomputer and minicomputer versions of BASIC.

This book is aimed at introductory programming students with little or no knowledge of computer systems or programming. It provides detailed support to the person who is using a computer for the first time, often a traumatic experience. It is designed to lead students through the first few sessions at the IBM PC keyboard without additional help from hard-pressed faculty or computer-center personnel.

The book helps students develop an understanding of the process of programming by stressing the functional design of computerized solutions rather than concentrating on simple statement syntax. It aims to develop a broad perspective on the uses of computers in business and science through personal experience in developing solutions to the kinds of problems actually faced by workers in these fields.

It consistently emphasizes and exemplifies structured, top-down program design, development, and implementation. It helps students manage their use of scarce computer resources by showing the advantages of planning before sitting down at the keyboard. A sample BASIC coding form and print spacing chart for use in planning is provided in Appendix D.

Concise and complete documentation is also exemplified and encouraged. The use of flowcharts in program planning is presented, and students are shown by example how to use pseudocode in planning and documentation.

Conceptual material is strongly supported by program examples. Most examples are complete routines or programs rather than isolated fragments of code. Each example is supported by extensive discussion showing how the procedure was designed. This approach promotes more intelligent understanding of computer use by showing students how to design procedures rather than just showing them how to code fragments of programs.

All examples of BASIC code and resulting displays have been transferred in digital form directly from an IBM Personal Computer into the recorded text from which this book was produced. Thus the problem of incorrect examples and examples containing typographical errors, very confusing to students, is avoided.

The book provides an unusually complete introduction to file processing, a topic typically ignored or given only cursory treatment in most other introductory programming books. Since file processing is the basis for most business data processing, the concepts of files, fields, and records are presented in the discussion of DATA and READ statements early in the text. The topic is then further developed throughout the book, with an extensive chapter on the details of IBM PC BASIC file input and output near the end of the book. An important advantage of a text devoted to a single version of BASIC is that file handling can be thoroughly treated in a practical, immediately applicable way. Material is provided to allow students to have some experience with external data files, even in abbreviated courses.

Each chapter in the text ends with a list of key terms and a comprehensive set of self-help review questions. Answers to selected review questions are provided at the end of the book to allow for student self-evaluation. In addition, each chapter provides an abundance of programming projects from which the instructor can select student assignments. Projects emphasize business data processing applications, but some scientific and engineering projects are also included in each chapter. All programming projects have been student-tested for suitability and realism.

With budget restrictions currently in effect, it is especially important that text materials provide as much support to the student as possible. Large class sections made necessary by personnel reductions limit the instructor's ability to provide personal assistance beyond the resources in the book. The material in this book is organized to allow its use in large sections if necessary. The approaches used here have been successfully tested in sections as large as 250 students.

The book is useful not only as an introduction to BASIC on the IBM PC, but also as a reference tool. Well-organized appendices cover DOS commands, BASIC statements and built-in functions, and error messages. These are keyed to the sections in the text providing fuller discussion of their applications.

The instructional material and examples in this book are based on a programming system consisting of an IBM Personal Computer with 64K or more of memory, the monochromatic adaptor and monochromatic display, dual floppy disks, a printer, and the IBM Personal Computer Disk Operating System (PC-DOS) and IBM Personal Computer Disk BASIC by Microsoft. Where appropriate, special instructions are provided to users of single-disk systems. All the material in this book should be applicable to systems with high-resolution color displays, but no instruction is given in the special Advanced BASIC statements that can be used with that display. The material in this book may be applicable to non-IBM microcomputers that use the Microsoft Disk Operating System (MS-DOS) and GW BASIC, but the authors have not attempted to verify this material on each system that advertises functional equivalence to the IBM Personal Computer.

The Plan of This Book

This text provides a complete introduction to the facilities of IBM PC BASIC. The first eight chapters introduce the basic concepts of programming, the statements of ANSI Minimal BASIC, and the directly related statements and built-in functions of PC BASIC. The remaining three chapters discuss the powerful PC extensions to Minimal BASIC in file I/O, text processing, and low-resolution graphics. This material may be included in a course of study or not, at the option of the instructor.

Chapter One introduces the concepts of structured methodology and the basic techniques used in writing good programs.

Chapter Two provides an introduction to the IBM PC, including procedures for turning on and running the machine, maintaining floppy disks, and entering, modifying, and running a BASIC program.

Chapter Three is an introduction to the essential concepts and statements necessary to begin programming. The material in this chapter allows students to write meaningful interactive programs using only seven BASIC statements: PRINT, INPUT, LET, GOTO, IF/THEN, END, and REM. Students are able to begin practicing their programming skills without being overwhelmed with the details of a large number of statements.

Chapter Four is a practical, step-by-step discussion of the programming process. In this chapter, students learn an effective and efficient method of solving problems by computer. The chapter is constructed around a realistic case study involving the solution of a business problem through the design, development, coding, testing, and documentation of a well-structured modular program. A professional approach to programming is demonstrated.

Chapter Five introduces four major sets of techniques commonly used in business data processing programs: data files using DATA and READ statements; counting and accumulating; subprograms, including the built-in arithmetic functions of IBM PC BASIC, subroutines, and simple user-defined functions; and programmed processing of errors.

Chapter Six discusses techniques used in producing output reports. Summary techniques, including use of the control break, are presented. In addition, output formatting techniques provided in PC BASIC are thoroughly explained, including the TAB function and the PRINT USING statement.

Chapter Seven provides an extensive discussion of looping, including the indefinite loop, the DO WHILE loop, the DO UNTIL loop, and the FOR/NEXT loop. Techniques of table lookup and table searching are introduced using tables stored in DATA statements. Simulation using random numbers is explained and exemplified in the solution of a simple queuing problem.

Chapter Eight is a comprehensive introduction to the use of arrays. The techniques of table lookup and sequential and binary searching of arrays are presented in detail. The min-max sort and bubble sort are discussed, along with simple merge techniques.

Chapter Nine offers a thorough discussion supported by concrete examples of the use of external data files in business data processing on the IBM PC. This material is necessarily detailed, yet students in the authors' courses regularly are able to write programs using external data files.

Chapter Ten presents the string-handling and text-manipulation facilities provided in PC BASIC. String-handling functions and sub-string specifications in PC BASIC are conceptually similar to the approaches often used in other microcomputer and minicomputer versions of BASIC. This chapter provides a thorough discussion of text processing on the IBM PC, supported by clear illustrations.

Chapter Eleven is an introduction to the field of low-resolution computer graphics. The emphasis is on practical graphics for reporting.

The Appendices at the end of the book provide commonly used reference materials, including syntax diagrams of BASIC statements, tables of built-in PC BASIC functions and system variables, and explanations of PC error messages. Each of these references is keyed to the section of the text where further information may be found. A sample BASIC coding form and print spacing chart is also provided.

A list of DOS operating system and BASIC commands is included on the inside back cover for ready reference.

Acknowledgments

The authors are greatly indebted to Eileen Sondak and Ann Hatch, whose support made this project possible. We thank Ronald A. Lindsay for his help in developing the materials presented here.

We are also indebted to the following colleagues, whose comments were very useful in ensuring the quality of the finished book:

Wilbur P. Dershimer, Jr., Ed. D.
Seminole Community College

Eileen Bechtold Dlugoss
Cuyahoga Community College
COMSYSCO

Darrell Eubanks
Texas Tech University

Robert P. Hudson
Triton College

John F. Lehane, Ph.D.
San Jose State University

Preface to the Student

In this book, you will be learning about programming through personal experience. Whether you plan to be an information systems professional or a user of computer facilities, your programming experience as you study this book will provide excellent background.

To learn about programming, you must take the time to practice writing programs yourself. There is simply no substitute for experience. Certainly reading a good textbook about programming will not make you a programmer unless you also practice programming yourself.

As you finish each chapter, answer the review questions at the end to ensure that you understand the material. Then select two or three of the programming projects and complete them on your own. Even if your instructor assigns programming exercises, do your own personal practice in addition.

In your personal practice, apply the principles of program development you are studying in this book. The first and most basic principle is this: Design and write your program <u>before</u> you sit down at the computer. The computer is a good place to enter your program and test it, but it is a very poor place to do the planning. If you plan at the computer, you will soon be tempted to jump ahead and enter some code. You can be sure that any coding you do before the planning is completed will be counterproductive; it will lead to unnecessary errors and a lot of extra work.

Once you finish planning a practice problem, enter the program on the IBM Personal Computer and test it if you can. Even if you do not have extra access to the PC, though, your own paper-and-pencil practice will pay off. Resolve right now to do it!

Explanation of Statement Notation Used in This Book

line#	Line number.
CAPITALIZED TERMS	Required words in statement; must be entered exactly as given.
| |	Alternatives; one of the terms contained in the list must be entered.
[ ]	Optional parts; may be entered if needed or omitted.
[...]	Optional repetition; the immediately preceding term may be repeated if necessary.

Other Special Notation Used in This Book

< >	Press key with this name.
____	Material entered by user at keyboard.

SPECIAL DOS KEY COMMANDS

The following commands are entered by pressing special keys or key combinations. Reference is to the page number in this book where discussion of related material begins. References to "Man" are to the IBM Personal Computer Disk Operating System manual and the IBM Personal Computer BASIC language reference manual.

COMMAND	ACTION	REFERENCE
<Alt>-<letter>	Enter a keyword into a BASIC statement.	19
<backspace>	Back up the cursor one display space and erase the character displayed in that position.	14
<Ctrl>-<Alt>-<Del>	Restart the computer. Any program in memory is lost.	14
<Ctrl>-<Break>	Interrupt execution of a BASIC program.	14
<Ctrl>-<Num Lock>	Freeze the screen display. Press any key to continue.	14
<Esc>	Discard current entry line. Must be pressed before <Return> on that line.	14
<shift>-<Prtsc>	Copy the current screen display on the printer.	13

DOS COMMANDS

DOS commands are entered at the DOS prompt of A>. The following descriptions provide the most common applications of these DOS commands. Many of these commands have additional forms and applications that are described in the IBM Personal Computer Disk Operating System manual. References are to the page in this book where discussion of related material begins. References to "Man" are to the IBM Personal Computer Disk Operating System manual.

COMMAND	ACTION	REFERENCE
A:	Set drive A as the default drive.	12
B:	Set drive B as the default drive.	12
BASIC	Enter the BASIC language system.	17
BASIC /F:n	Enter the BASIC system with a maximum of n files open at any one time.	231
CHKDSK [drive]	Display the number of bytes used for various purposes on a diskette and the number of bytes of space remaining.	16
CLS	Clear the display screen.	275
COMP filename drive	Compare a file on one diskette to the corresponding file on another.	17
COPY filename drive	Copy a file from one diskette to another.	17
COPY CON filename	Copy data entered at the keyboard to the specified file; terminate data entry by pressing <F6>.	237
DATE	Display and optionally change the system date.	Man
DEL filename	Delete a file from a diskette.	17
DIR [drive]	List the names and sizes of the files on a diskette.	16
DISKCOMP source-drive target-drive	Compare the entire contents of one diskette to the contents of another.	15
DISKCOPY source-drive target-drive	Format the target diskette and copy the entire contents of the source diskette to the target diskette.	14
EDLIN filename	Edit a file.	237
ERASE filename	Delete a file from a diskette.	17
FORMAT [drive] [/S]	Prepare a new or recycled diskette for use. The /S parameter formats a diskette containing the operating system.	15
PRINT filename	Transmit a file to the printer.	Man
PROMPT string	Change DOS prompt to string.	Man
REN old-name new-name	Change the name of a file on a diskette.	16
RENAME old-name new-name	Change the name of a file on a diskette.	16
TIME	Display and optionally change the system time.	Man
TYPE filename	Display the contents of the specified file.	235
VERIFY [ON or OFF]	Check after each disk write operation to ensure that the data can be read without error. Slows disk write operations.	Man

BASIC COMMANDS

BASIC commands are entered at the BASIC prompt of Ok. The following descriptions provide the most common applications of these BASIC commands. Many of these commands have additional forms and applications that are described in the IBM Personal Computer BASIC language reference manual. References are to the page in this book where discussion of related material begins. References to "Man" are to the IBM PC BASIC manual.

COMMAND	ACTION	REFERENCE
AUTO [beginning-line-number],[increment]	Automatically generate program line numbers beginning with beginning-line-number and incrementing by increment. Enter <Ctrl>-<Break> to discontinue.	18
CLEAR	Set all numeric variables to zero and all string variables to null.	Man
CONT	Resume execution of a BASIC program interrupted by <Ctrl>-<Break>.	74
DELETE line# [-line#]	Delete one or more lines from the BASIC program in memory.	20
EDIT line#	Display a line of the BASIC program in memory for editing and position the cursor at the beginning of the line.	20
FILES "[drive]*.*"	List the names of the files on a diskette.	18
KEY OFF	Turn off the 25th line display of function key values.	18
KILL "filename"	Delete a file from a diskette.	18
LIST [line# [-line#]]	Display all or part of the lines of the BASIC program in memory.	19
LLIST [line# [-line#]]	Print a hardcopy listing of all or part of the lines of the BASIC program in memory.	19
LOAD "filename"	Erase any BASIC program lines currently in memory and copy the program named filename from a diskette into memory.	22
NAME "old-filename" AS "new-filename"	Change the name of a file on a diskette.	18
NEW	Erase any BASIC statements currently in memory.	18
RENUM [new-number],[old-number],[increment]	Renumber the lines of the BASIC program in memory beginning with existing line old-number. The new sequence begins with line new-number and is incremented by increment.	21
RESET	Close all open files.	Man
RUN	Execute the BASIC program currently in memory.	18
SAVE "filename"	Copy the BASIC program in memory to a diskette under the name, filename.	22
SYSTEM	Leave BASIC and return to DOS.	23
TRON	Enable tracing; display line number as each line is executed.	73
TROFF	Disable tracing.	

80L8605ABB 228 (260)
Using BASIC on the IBM Personal Computer
Narman E.Sondak and Richard A.Hatch
San Diego State University
在IBM个人计算机上的BASIC语言使用
1985 384 pp. $19.95
13-4740
Text, Soft Cover
0-574-21740-1
SRA P.O.Box 5380 Chicago, IL 60680-5380

ONE

What Is Structured Methodology and Why Is It Important?

We are living in the middle of a major revolution. Suddenly we can send spacecraft to the planets, manufacture products to tolerances unheard-of forty years ago, coordinate organizations of unparalleled size, and aim incredibly destructive weapon systems with pinpoint accuracy. These developments, and hundreds of others, all depend on one invention: the digital computer. The explosive changes that have taken place during the forty years since the first digital computer went into operation certainly justify calling this period the Computer Revolution.

The Power of the Computer

The power of the computer lies in the fact that it is a multi-purpose machine. One moment it can be producing the payroll check of an assembly-line worker, the next moment it can be calculating expected wear in a newly engineered bearing, and the next moment it can be producing a construction schedule for the 300,000 components of a battleship.

If a specific procedure can be devised for achieving an information processing goal, then a computer can, at least in principle, be instructed to execute that procedure. Of course, there are some limitations on the kinds of procedures a given computer can execute, but within broad limits a computer simply does what it is told to do.

The Problem with Computers

As tasks become complicated, however, it becomes difficult to devise correct procedures for achieving them. Space-shuttle guidance systems fail, paychecks get written for millions of dollars instead of hundreds, and projects do not get completed on time or within budget--not because the computer itself fails, but because of the human difficulty of defining precise procedures for achieving the desired goals.

In fact, the units of computer hardware--the physical components themselves--hardly ever fail. The software is, by any standard, the weak link in the system. Software is the collection of procedures, programs of instructions, manuals, and other documentation that defines the tasks the computer is to carry out. Our record in designing correct and reliable software over the past forty years has been spotty, at best.

Of all the steps in implementing a computerized solution to a problem, designing the program is the most difficult. Descriptions that people would find clear and obvious are seldom self-explanatory to a computer. We humans do not have any problem, for example, understanding the directions on a shampoo bottle: "lather, rinse, repeat." But a computer is so literal in its interpretation of instructions that these directions would send it into an endless cycle of repetition.

Historically, the larger and more complex the computer applications became, the more serious and costly the problems. By the mid-1970's the situation became so serious that a new and powerful design approach was under development, called structured methodology. "Structured" means using a preplanned set of working methods in solving problems. The purpose of structured methodology is to help people design programs that are reliable, easy to modify, cost effective, and completed on time and within budget.

The Cost of Programming

While the cost of hardware has been dropping rapidly in the past few years, the cost of software has been rising. It is now estimated that about 85% of the total cost of developing a new computerized processing system goes for software. Once the system is in operation, about two-thirds of the cost of maintaining it goes for software modifications. If the system is poorly designed, both initial costs and maintenance costs of software can be even higher proportions. The use of structured methods helps ensure the soundness of the design.

Programming is a labor-intensive activity, and the productivity of individual programmers varies enormously. Studies show that some programmers produce as much as ten times more finished lines of program than others in the same time period. Since programming is partly an art, part of the variance arises from differences in individual talents. But programming is also partly a science, and the application of basic principles of structured methodology can help the programmer make the best use of this science. Thus, structured methods can help increase programmer productivity and reduce the extreme productivity variations.

The Three Stages of Structured Methodology

Developing new computer applications involves three stages: (1) structured system design, (2) structured program development, and (3) structured programming.

The goal of structured system design is a clear, detailed description of the problem and how it will be solved. Parts of the problem are identified, and detailed solutions to each part are developed. Since few significant business problems can be solved by a single computer program, the system design specifies what programs will be required and exactly what each of the programs must do. System design is normally done by a systems analyst, who passes along the completed system design to programmers who will develop the actual programs.

The goal of structured program development is a plan for an efficient and economical program that meets the goals set by the systems analyst. Structured program development includes all the activities carried on from the time the programmer receives the analyst's design until the day (probably some years in the future) when the program is finally replaced by a completely new design for solving that problem.

The goal of structured programming is a program that is both correct and easy to modify. The actual writing of the program is one important aspect of structured program development. Techniques of structured programming will be discussed throughout this book.

STRUCTURED SYSTEM DESIGN

The purpose of <u>structured system design</u> is to aid human beings in dealing with complex problems. Because we humans have a limited ability to handle many details at once, human problem-solving methods must reduce complexity to manageable levels.

Our normal way of doing this is simply to ignore most of the details and concentrate on main lines of cause and effect. Since the human brain can readily perceive such patterns, it is fairly easy to teach others to solve problems by such methods. Computers, however, function only at the very detailed level. They cannot easily be instructed to perceive general patterns of cause and effect; they can only be instructed how to react to highly specific patterns of details. Thus, we cannot approach computerized problem-solving by ignoring the details.

Structured system design provides a workable alternative. It provides systematic guidelines for dividing large, complex problems into sets of smaller subproblems, then further dividing these subproblems into sub-subproblems, and so on. By taking apart a large problem into successively smaller subparts, eventually we reach a level at which we can understand each of the parts in full detail. At that point, we are in a position to devise detailed computerized procedures for dealing with the problem, part by part.

Input, Process, and Output Analysis

Through experience, we have learned that many system design problems can best be solved by dividing them into three steps. The first is input, in which detailed data is made available for processing. The second is processing, in which the data is transformed according to a specified procedure. The third is output, in which the results of the processing are recorded or displayed.

For example, in processing a payroll for hourly-wage employees, the input step involves assembling data about the number of hours each employee has worked during the pay period and each employee's rate of pay, and making this data available for processing. Processing consists of multiplying the number of hours times the hourly rate of pay. Output involves printing a paycheck for the employee and creating a record of the payment for accounting purposes.

If an actual payroll were as simple as this, of course, the systematic approach of structured system design would hardly be necessary. In a real payroll, many additional details must be handled, all correctly. A real payroll, for example, would include a section to calculate income-tax withholding. That section, too, can be analyzed into input (amount of gross pay and number of dependents declared by the employee), processing (calculating the tax), output (making the result available to the part of the program that prints the paycheck and creates the accounting reports).

Dividing problems into input, processing, and output stages is one of the most useful techniques of structured methodology. It is called IPO (Input, Processing, Output) analysis.

Top-Down Design

When we analyze a problem by dividing it into its parts and subparts, we must not only determine how each part works internally, but we must also determine how the various parts relate to each other. To see these relationships clearly, we must be very systematic. Experience shows that the analysis is most likely to be successful if we first divide the whole problem into its major parts, then divide each part into subparts, continuing toward smaller and smaller subparts as necessary. This analysis is the first step in top-down design.

Working systematically from the top (whole problem) down to the bottom (small subpart) provides substantial advantages over using a haphazard approach. Without such an approach, an analyst might divide a problem by first noting various obvious parts, then arbitrarily patching them together by devising additional parts to fit any gaps that remain. Although this method might result in a correct program, the systematic, top-down approach is far more likely to lead to a correct program time after time.

Once the problem has been analyzed into parts, the system to solve it is designed from the top down also. First, the input, processing, and output steps of the highest-level parts are designed. Then the input, processing, and output of subparts are designed to mesh with the requirements of the top-level parts. Finally, the input, processing, and output at the bottom levels are designed.

The principal advantage of top-down design over less systematic approaches arises from the necessity for parts and subparts to fit together, or interface, correctly. The input of subparts comes from higher-level parts, and the output of subparts becomes input to higher-level parts. If parts are designed in haphazard order, the danger is that these interrelationships may be designed for convenience in putting together some low-level part rather than to meet the needs of the system as a whole in the best way.

Top-down analysis and design are major steps toward increasing the influence of science in system design and reducing dependence on sheer personal talent in producing correctly working systems.

STRUCTURED PROGRAM DEVELOPMENT

Structured program development is based on the normal life cycle of a program. A program's life cycle normally includes the following eight stages:

1. Problem definition
2. Program design and logic development
3. Coding

4. Keyboarding and translation
5. Testing and debugging
6. Documentation
7. Operations
8. Maintenance

Each of these stages can be enhanced by applying the techniques of structured program development.

Problem Definition

Although the programmer's work normally begins after a systems analyst has written a general description of the program's input and output, the programmer must begin by developing a thorough understanding of the problem. At this stage, definition of inputs and outputs must be complete and fully detailed. In addition, the required processing must be spelled out in full detail.

Program Design

Program design is the process of devising a strategy for solving the problem once it is thoroughly understood. The complexity of individual data processing programs usually requires that they be segmented into modules, each performing a particular function. Techniques of structured program development include methods of systematically charting the functions allocated to each module and the relationships among modules.

Programs should be designed top-down, just as whole systems are. First, the plan for the main program module is fully developed, then the plans for other subordinate modules are developed.

Coding

Coding is the process of writing the program in instructions that can be recognized and executed by the computer. The computer can only execute instructions coded as electronic pulses called machine language. Early in the history of computers, people had to translate instructions into machine language laboriously by hand. By the early 1960's, however, computers themselves began to do this job, using computer programs called higher-level languages.

One of the five or six most widely used languages is BASIC, Beginners' All-purpose Symbolic Instruction Code, developed in the mid-1960's by John G. Kemeny and Thomas E. Kurtz at Dartmouth College. The BASIC language system allows programmers to enter sequences of instructions in a way people can easily understand, and it translates those instructions into the machine language the computer can execute.

Each language prescribes a highly defined form in which instructions must be coded. The coding stage involves formulating the sequence of instructions into this form. Structured methodology provides powerful techniques for coding programs correctly and efficiently. Most of the rest of this book is about these subjects.

Keyboarding and Translation

Before instructions can be executed by the computer, they must be recorded in a form the computer can read and they must be translated into machine language. In the early days of computing, instructions were keypunched onto cards for entry into the computer. Now, most programs are entered directly into computers through their keyboards. There are two principal strategies in the translation process: interpretation or compilation. An interpreter is a machine language program that reads one instruction at a time and immediately carries out the step described in that instruction. The interpreter includes the machine language routines needed to carry out the instructions. A compiler, on the other hand, reads all the instructions in the program and translates the whole set into a corresponding machine-language program; only when the translation is completed does the execution of the instructions begin.

Each type of translator has its advantages and disadvantages. An interpreter can be run on a very small computer, and it is capable of executing a programmer's instructions almost immediately. It is best suited to program development, in which changes are frequently made and immediately tested. It is also well suited to programs that will be quickly written and executed only once. BASIC on the IBM Personal Computer is implemented as an interpreter.

A compiler requires more time to produce an executable program each time a change is required, and its operation requires greater computer resources. However, the program it produces is actually executed much faster--perhaps as much as 100 times faster--once it is fully correct. Thus a compiler is best suited to producing operational programs that may be used frequently without change.

Testing and Debugging

Testing and debugging* involve two steps. The first is to ensure that the program is coded in accordance with the language used (e.g., that all the statements are acceptable to the BASIC interpreter). The second is to ensure that the program performs its planned function correctly.

Structured program development provides a technique, top-down testing, for ensuring the correctness of a complex program. Using this approach, the highest-level module of the program is first tested alone. Then one next-level module is added and tested, then another, and so on. This method ensures that any errors that arise at each stage must be caused by the addition of the one new module. The alternative approach, testing each low-level module separately, then putting all the modules together and testing the whole program, often results in extremely persistent errors that are difficult to find because they result from the subtle mismatch of parts.

Documentation

Documentation is written material explaining what the program does and how it works. Documentation includes explanatory notes written directly into the program, separate descriptions with charts and tables to aid in later modification of the program, directions for computer operators who must run the program, and manuals for those who must prepare input data and use the output reports.

Much of the documentation in program development should be done as the project progresses, stage by stage. Many of the charts used in documentation, for example, are actually prepared in the problem-definition and program-design stages. Internal program documentation is written as the program is coded. Once the program is fully tested, these materials are reviewed and the additional documents are readied in final form.

Because programs, once completed, are often used and continually modified for as many as ten or twenty years, it is extremely important to document them completely and clearly. The person who originally wrote the program may be long gone before the program is finally retired from service, so that person's memory of how the program works will not be available when modifications are needed. Even if that person is available to make modifications, his or her memory of the details may soon fade. Documentation is vital, and this book will emphasize the use of good documentation techniques.

---

*In the early days of computing, electromechanical relays (mechanical switches opened and closed by electromagnets) were used instead of vacuum tubes or transistors. Tracking down a persistent error in a program one day, some programmers found that a small moth had crawled between the contacts of a relay and been crushed there, preventing the switch from closing. The case was widely publicized--in fact, that bug is preserved in the Naval Museum in Norfolk, Virginia--and the term bug, meaning an error in a program, was born.

Operations

The operations stage is the use of the program during its normal life of ten to twenty years. The main operational activities include data entry by various departments, running of the program by computer operators, and use of the resulting information by various people throughout the organization. In each case, clear and complete documentation is essential to successful operation of the program.

Maintenance

For a typical program, as much money will be spent on maintenance--continuing modification of the program during its lifetime--as was spent to plan and develop it in the first place. Therefore, providing for ease of maintenance is an important way of reducing the program's overall cost. If structured techniques were used in the planning and development stages, the program should be easy for the maintenance programmer to understand and modify. Thus, maintenance costs should be significantly reduced.

STRUCTURED PROGRAMMING

Structured programming provides a set of techniques used in the coding stage of program development. The goal of structured programming is a program that works correctly and is easy to understand and modify.

The truth of the matter is that coding is the easiest part of program development. As a beginner you may find it challenging to figure out how to state instructions in the limited vocabulary of BASIC, but even at the beginning the difficult part of programming is planning the sequence of steps of the program--the logical flow. Once that sequence is established, then translating the steps into BASIC is not very difficult. Coding normally takes less than one-fourth of the total time spent on a programming project.

Structure and Simplicity

Well-structured programs are simple to read and understand. This apparent simplicity is achieved only at the cost of considerable effort, but the effort is worthwhile. If a program is easy to understand, then you can easily see whether its logical sequence of steps is correct. When the program must be modified someday, it will be easy for that programmer to figure out how to make the necessary changes in it without introducing errors. Both you and that future programmer benefit.

Structured programming provides a small set of simple patterns (structures) and requires that all the code in a program conform to one of those patterns. Experience shows that such programs are likely to be simple to understand.

Another technique of structured programming involves applying the familiar top-down approach. First, the main module is coded, then the subordinate modules. Working top-down ensures that the main module is structured simply and clearly, since it is the most important part of the program.

Finally, structured programming includes the technique of the structured walkthrough, a systematic group inspection of the coded program by other programmers before testing begins. This walkthrough can detect costly errors that otherwise might not be discovered until the operations stage.

Structured Programming and BASIC

Because BASIC was developed before the advent of structured programming, standard BASIC does not include the types of instructions necessary to implement structured programming directly. The American National Standards Institute (ANSI) has developed a standardized Minimal BASIC, which specifies a core set of instructions included in nearly all versions of BASIC.

Most manufacturers have added to this core set of instructions. Some have added all the instructions needed to implement structured programming. Programs written in these versions of BASIC can achieve the maximum simplicity.

BASIC on the IBM Personal Computer is an intermediate version in this respect. The manufacturer has added many useful types of instructions in addition to the ANSI Minimal BASIC set, but it did not add all the instructions that would be useful in structured programming.

The result is that while programs in IBM PC BASIC cannot take full advantage of the simplicity offered by structured programming, they can approach this goal. In cases where ideal single instructions are not provided, these can be simulated by brief sequences of instructions. You may find that you arrive at a clearer understanding of structured programming through working out these problems than if instructions for full structure had been provided.

What Makes a Program Good?

Structured methodology, in summary, is a set of techniques for producing good programs as inexpensively as possible. When you are writing programs for other people to use, good programs are ones that apply the Golden Rule: Do unto others as you would have them do unto you. To help you apply this maxim, here are basic criteria:

1. Correctness
2. Ease of use
3. Readability
4. Maintainability

Above all, a good program works correctly. It produces correct results from the data supplied to it. This is the most important criterion of a good program.

A good program is easy to use. Data entry should be structured so it is as easy as possible, both for convenience and to minimize data-entry errors. The output should be provided in forms that make it as clear and useful as possible. The actions of the program should be ones the user expects, so there are no surprises. When the user makes an error, the program′s response should be politely phrased and helpful in indicating how to correct the error.

A good program and its documentation are easy to understand. The program itself should be carefully structured for simplicity, and it should include ample explanatory comments. Charts and other aids to understanding should be provided. Documentation should be clearly written for each group of readers, avoiding the use of computer jargon and other unnecessary complications.

A good program will have a long lifespan, and it will have to be changed from time to time. In fact, the changes will probably cost as much as the program did to begin with. Good programmers recognize this and make their programs easy to modify. This means that the programs are easy to understand and that small changes in one part of the program will not create unexpected errors in other parts. Systematic use of structured methodology will lead to programs of this kind.

Terms You Should Understand

ANSI Minimal BASIC
bug
coding
compiler
debugging
documentation
hardware
higher-level language
input
interface
interpreter
IPO analysis
machine language
maintenance
module
operations
output
processing
software
structured methodology
structured program development
structured programming
structured system design
structured walkthrough
top-down design
top-down testing