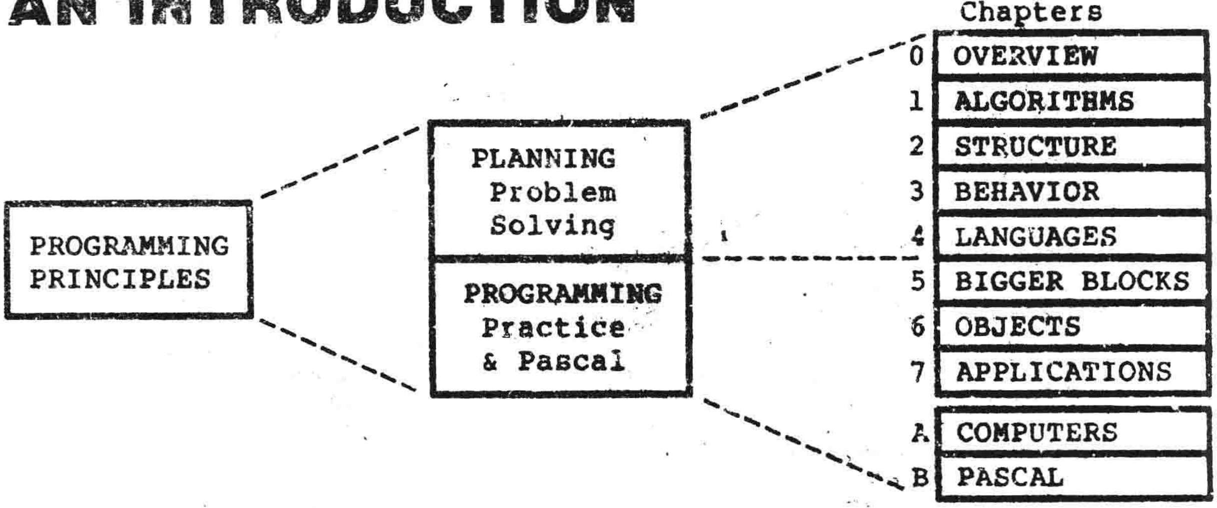
JOHN MOTIL

PROGRAMMING PRINCIPLES AN INTRODUCTION

PROGRAMMING PRINCIPLES AN INTRODUCTION



JOHN MOTIL
CALIFORNIA STATE UNIVERSITY
AT NORTHRIDGE

Allyn and Bacon, Inc.
Boston • London • Sydney • Toronto

Library of Congress Cataloging in Publication Data

Motil, John.
Programming principles.

Includes index.
1. Electronic digital computers--Programming.
I. Title.
QA76.6.M69 1983 001.64'2 83-11779
ISBN 0-205-08005-7

Printed in the United States of America

10 9 8 7 6 5 4 3 2 89 88 87 86 85 84

Table of Contents

of

PROGRAMMING PRINCIPLES

(detailed contents follow)

PREFACE
0. OVERVIEW: Top-down
1. ALGORITHMS: Representation
2. STRUCTURE: Form
3. BEHAVIOR: Dynamics
4. LANGUAGES: Communicating
5. BIGGER BLOCKS: More, etc.
6. OBJECTS: Data Structures
7. APPLICATIONS: Design

A. COMPUTERS: Low Level View
B. PASCAL: a High Level View

Preface

PROGRAMMING PRINCIPLES

CHAPTER OUTLINE.............1
GOALS AND NON-GOALS.........2
VIEWS OR BIASES.............4
USES OF THIS BOOK...........8
FORM OF THIS BOOK...........9
PRODUCTION OF THIS BOOK....10
SOURCES (Bibliography).....11
THANKS (Acknowledgements)..12

PREFACE:

PROGRAMMING PRINCIPLES

This book introduces the basic ideas of programming, and a structured way of thinking about planning and problem solving. It is intended for a first course in computing for both computer science majors and also non-majors.

The goal is to develop an ability to create algorithms. A sub-goal is to communicate algorithms, as programs, in a language. This sub-goal is considerably easier to attain than the main goal. In fact the sub-goal, of programming in Pascal, is done in an appendix, whereas the rest of the book is devoted to the main goal. Hopefully, it conveys an attitude as well as a methodology.

The approach is very visual, using many diagrams (such as flow block diagrams, flowcharts, data flow diagrams, syntax diagrams, break-out diagrams). Each pair of pages is written as a unit (similar to a program module), with a page of graphics facing a corresponding page of text. The graphics were done with a computer.

Starting with many examples from everday life (involving cooking, business, change-making, calendars, games, etc.), this book creates an "algorithmic" logic of actions, showing how all algorithms can be created from only four building blocks.

Significant concepts (such as structured programming, top-down design, data flow, trees, arrays, records) are introduced simply, in terms of common examples, and then later in terms of computers. In this way, the important programming concepts are not confused with the details of computers or programming languages.

Practice of programming in the Pascal language is considered in a long appendix (of almost 100 pages). This can be studied in parallel with the more general part of the book, but preferably after chapter 3. Computing machines are also included briefly in another appendix.

Many problems, programs, and projects, of varying difficulty, are included after each chapter.

More detailed descriptions of the above ideas follow.

CHAPTER OUTLINE:

PROGRAMMING PRINCIPLES

0. OVERVIEW: Top-down
   provides a "bird's-eye" overview and introduction to the general ideas of programming (algorithms, languages, machines, and systems) showing how they are related.

1. ALGORITHMS: Representations
   provides many common examples of algorithms, along with many different ways of representing them, including formulas, tables, trees, flowcharts, flow block diagrams, data flow diagrams.

2. STRUCTURE: Form of Algorithms
   considers the basic building blocks and how they can be interconnected properly to form algorithms.

3. BEHAVIOR: Dynamics of Algorithms
   emphasizes the dynamic actions of programs, algorithms intended for computers.

4. LANGUAGES: Communicating Algorithms
   considers concepts common to most high level languages, emphasizing the similarities in semantics (meaning or actions) despite the differences in syntax (form or look).

5. BIGGER BLOCKS: More, Different, Deeper Concepts
   describes larger building blocks, more data types, deeper nests of loops, and more powerful sub-programs.

6. OBJECTS: Data Structures
   considers some compound objects (arrays and records) and some algorithms (such as sorting and searching) for operating on these data structures.

7. APPLICATIONS: Design, Systems
   considers some computer uses (including simulation and graphics) in business, engineering, and other areas. It also shows various design methods for larger programs.

Appendices

A. COMPUTERS: A Low Level Machine View
   introduces a simple but typical computer. Structuring is continued in a low level language.

B. PASCAL: Programming Practice
   introduces a simple high level programming language, Pascal, which is very similar to most modern programming languages. Many of the programs of the main text are shown written in this language. This appendix can be read in parallel with the text.

GOALS OF PROGRAMMING PRINCIPLES

WHAT The goal of this book is to introduce the basic concepts of programming and a structured way of thinking about planning, problem solving, and programming.

WHY The structured approach aims at plans and programs that are easy to read, understand, write, describe, improve, test, modify, extend, analyse and evaluate.

WHEN This book is intended for a first course in computing, at an early university level. It should be suitable for students majoring in computing science, as well as those majoring in any other area.

WHERE The concepts studied here are general, not restricted to any particular computer, or language, or system, or application area. There are a great many examples, mainly chosen from everyday life, involving paychecks, calendars, games, statistics, business, cooking, etc.

WHO This book is intended for the beginner. No previous background in computing is assumed (and in fact may be a disadvantage, for bad habits are hard to break). Extensive mathematical background is not a prerequisite, for most of the required mathematics and logic are developed as necessary.

HOW The approach taken here could be called the "systems" view, or "top-down" view. It proceeds first from the top "big picture" view, then through intermediate refinements, and ultimately deals with the details.

NON-GOALS

It may be significant to realize what goals were not attempted here (although some of the non-goals may have been achieved).

This book is not intended as a language training manual,
although a language, Pascal, is introduced in some depth.

This book does not emphasize training in a skill,
although some skills are acquired in the learning process.

This book is not merely a collection of techniques or tricks,
although some techniques are encountered within the general unity.

This book is not a picture-book survey of computers,
although it does have considerable graphics.

This book is not a glorified glossary of computer "buzz words,"
although some vocabulary is introduced to describe the concepts.

This book is not a study of the hardware aspects of computers,
although some computer organization is briefly covered.

This book is intended for serious students to develop a view of planning, problem solving and programming using computers at a high human level.

VIEWS or Biases

This book is based on a few simple premises, views or beliefs (my biases, if you wish). They are written so as to seem obvious, but some have been argued considerably.

WE SHOULD LEARN TO READ BEFORE WE WRIGHT (RITE?)!

The creative activity of writing (be it prose, poetry, or programs) should begin by reading good writings of others. We then may analyse them, criticize them, appreciate them, and ultimately venture to write our own. Unfortunately, many people like to start writing programs very quickly, and they end up forming bad habits.

Here we delay the immediate creation and running of programs on computers. We start by reading already created ones, following (or tracing) them. Then we will create algorithms, study them, modify them, and ultimately code them into a language and run them on some machines.

A PICTURE IS WORTH 500 WORDS (OR IS IT 1024?)

Humans tend to understand graphic or diagrammatic structures more easily than written text. For that reason, this book uses many diagrams. In fact almost every second page here consists of diagrams, with a facing page of text (of about 500 words) describing the diagrams. The diagrams and text are equally important.

Diagrams often lead to immediate insight. Flowcharts and flow block diagrams help visualize flow of control. Data flow diagrams describe flow of data and sub-program interaction. Syntax diagrams define language structure. Data space diagrams illustrate parameter passing. Break-out diagrams aid in top-down design. Other diagrams include: state diagrams, trees, and two-dimensional traces.

In the past, many programming books have avoided graphics, often because the diagrams were difficult to draw, but with the aid of computers this difficulty is disappearing. All the diagrams in this book were done with computers.

FLOWCHARTS SHOULD FLOW (OR ELSE GO)

Flowcharts consist of boxes joined by arrows to indicate the flow of action (or control). These diagrams often work well for simple structures, but for more complex structures they could become confusing and prone to errors. It's just too easy to connect an arrow to a wrong box.

Here we insist that flowcharts must flow, and we develop a method and notation to enforce proper flow. Alternatively we develop a flow block diagram and pseudo-code, both of which show the flow structure diagrammatically in quite different but equivalent ways.

MORE VIEWS

LEAVE TO THE MACHINE WHAT BELONGS TO THE MACHINE
(dirty details, tedious work, repetition)

To use a computer (or car, or clarinet) properly, it is not necessary to know all the details of how it is made. A computer is a very low level device which communicates using the two symbols 0 and 1 only. Humans, however, communicate using the 26 characters of the alphabet, the ten decimal digits, punctuation, and various other symbols. Humans then put together characters into words, words into sentences, sentences into paragraphs, paragraphs into chapters, books, etc.

Here the emphasis will be at the higher, human levels, closer to our applications. Our programming languages will resemble English, mathematics, and symbolic logic. We will, however, consider the computer and its lower level language in sufficient detail to understand the basic concepts, but we will avoid great depth here. The computer, and its low level language, is not introduced too early, for one's first language (be it natural English language, or a programming language) forms the most powerful tool of thought, and if the first language is not at a proper high level, then creativity may be hampered.

Here we will start at a very high diagrammatic level, with flow charts and flow block diagrams. Later we can use intermediate (or higher) levels involving languages such as Pascal, Basic, or Fortran. The computer then translates these languages into its lower level language.

STRUCTURE IS SOUND (So it works. So what?)

The structure, form or organization of any creative project is important. It is not sufficient that a program works; it should work well. It should be easy to read, write, describe, understand, test, analyse and evaluate. It must also do more, as follows.

Creating programs differs from most other creative activities (such as painting portraits or designing buildings). No one attempts to touch up someone else's completed painting or simply double the number of stories in an already existing building. But in programming there is always the potential for further modification, so programs should be created with such change in mind. Our masterpieces are made to be modified. They should be easy to extend, improve, expand, optimize or transport to other computers. Structured programs created in a top-down manner are usually better for such changes.

SIMPLICITY IS SUPREME

There are usually many ways to create anything, so ultimately a choice must be made. We will invariably choose the simplest, clearest structured way. Experience has shown that beginners often like to use very clever, devious, and difficult ways to program. Ultimately, however, when writing larger programs, the intellectual challenge of programming becomes too great, and only simplicity can keep programs at a manageable level.

OTHER VIEWS

There are many other personal beliefs that have been incorporated into this book. Some of them follow; others I am no longer aware of. Some views (such as "The best length of any unit is one page") correspond to good programming practice as well as good writing and teaching practice.

FIRST THE FAMILIAR

This book begins with common everyday algorithms, and only later gets into computer programs. This way the general fundamental concepts of algorithms are not confused with the particular details of computers or programming languages. The Devil hides in details.

SPIRAL IS SIMPLE

When any concept is introduced here, it is not treated exhaustively in its entirety at that one point. Instead it is first introduced simply; then at a later time is extended and returned to again and again. Each concept is revisited in ever-increasing refinement. This "spiralling" provides for increasing depth, but with breadth for proper "top-down" perspective. So if a concept is not clear to you at first, read on; it will get clearer as you proceed.

DUALITY IS DIVINE

Many concepts, not just programming ones, can be viewed in two ways which are complementary. Some examples are:

series vs parallel, general vs particular,
recursion vs iteration, depth vs breadth,
space vs time, wears tie vs forgets belt,
top-down vs bottom-up, intuitive vs logical,
hot vs cool, software vs hardware.

Although some areas of study seem to prefer a specialization, I believe programming requires a balanced view: an intuitive, holistic, spatial, subjective approach for creating programs, and also a rational, logical, defensive, objective approach for analysing, testing and optimizing programs. Programming is both an Art and a Science.

ALL THINGS ARE NOT EQUAL

Not all ideas, concepts, pages and chapters are of equal importance. The significance of each page is indicated in the table of contents preceding each chapter. The more important ideas are usually treated early in a chapter (top-down). Later in each chapter, there are extra challenging concepts (marked optional) which are not necessary for continuing to the next chapters. So if only two-thirds of this book is to be read, then it is best to read the first two-thirds of each chapter. Also, chapters are not equally significant, as shown below where asterisks indicate the relative importance.

*	0.	Overview	*	5.	Bigger Blocks
*	1.	Algorithms	**	6.	Objects
**	2.	Structure	*	7.	Applications
***	3.	Behavior		A.	Computers
*	4.	Languages	**	B.	Pascal

STILL MORE VIEWS

TWO DIMENSIONS ARE BETTER THAN ONE

Many concepts of computing appear to have one dimension, to be strung out in a line. For example, an algorithm seems to be a long list of instructions to be done step by step, and a computer program seems to be a long linear sequence of symbols. Actually, it is the structure behind these sequences that is important. It can be brought out by creating two-dimensional schemes such as break-out diagrams, trees and two-dimensional traces. Computers may have a single dimension; humans have many dimensions.

INDENTING IS IMPORTANT

One way of capturing the multi-dimensional form of prose, poetry, or programs is by indenting. In this text a reverse or hanging indent is used, with the most significant matters at the top and farthest to the left; less significant matters are indented farther to the right to show the lesser levels. A similar indentation is used for programs.

SUBS ARE SUPER

The concept of breaking up parts into sub-parts is extremely significant. For this reason, the sub-program concept should be introduced early, before other simpler, but inferior, methods (involving global variables). Through the use of break-out diagrams and data flow diagrams, this early emphasis on sub-programs is possible. Again, the lower details (stacks, parameter-passing methods, etc.) are postponed until later.

THERE IS A TIME

Arrays, while not terribly complex, are treated late in this book. They are not necessary at first, but we may still be tempted to use them because of their power. Also, the dual concept of the array, the record, may be more appropriate. There may be no optimal time to introduce some concepts, but there are non-optimal times.

NAMES ARE NICE

References may be made to things by either number or name. Here, names are preferred; all numbers look alike. Names are given to each chapter, sub-section, page, concept, and even homework problem. Giving good names makes things easier to remember, convenient to refer to, and helpful to manipulate.

BEWARE BEGINNINGS

This book is intended for beginning students, who are not expected to embark on a computing career immediately after reading this book. A surprising number of people believe that everything about programming can be learned in a single course. This book aims at providing a good foundation for continued growth.

Beginners, with their short algorithms and awkward typing, can be forgiven for using short variable names. After all, physics and mathematics have used single-character names for centuries. The advantage of long names becomes obvious to beginners after they start writing long programs. A compromise is used in this book: the smaller general algorithms involve short names, but the Pascal programs involve longer, meaningful names.

USES OF THIS BOOK

This book has been used in many different ways in various courses, and in different schools.

Typically, it has been used in an introductory course on algorithms and programming. It starts with chapters 0 to 3 emphasizing general algorithms, then "detours" to the Pascal appendix, and continues with chapters 4, 5 and 6, done in parallel with the remaining Pascal. Then some of the larger applications of chapter 7 are covered. Finally, the course ends with a brief description of computers and possibly a simulation of a simple computing machine.

Alternatively, this book has been used in a service course (COMP 101) to introduce many non-computer majors to programming. This is done by a shortened version of the above course, covering the general algorithms and some programming in Pascal. This course is followed (in a later semester) by a laboratory experience with emphasis on one particular language (Fortran, Cobol, Apl, PL/I, Basic, etc.). So, when a student wishes to learn more than one language, it is not necessary to repeat the basic principles and algorithms in each language laboratory.

This two semester sequence has much merit. The first course is a lecture (2 units) on fundamentals, with limited Pascal programming. The second course is a laboratory (1 unit) with extensive programming experience. This arrangement separates the principles (algorithms, problem solving) from the practice (language, syntax, system "incantations"), both of which require considerable amounts of time. This avoids the terrible compromises (premature programming, hectic pace, etc.) often encountered in a first programming course. Of course some "hackers" would be unhappy, but most students can abstain from the computer (for about a month), if they are convinced of the importance of this.

Originally, the book served as the basis for three courses: COMP 130SCE (for science and engineering students), COMP 130CSM (for computing majors) and also COMP 130GEN (for general education). The COMP 130SCE course, for example, tended to introduce the computer very early. Although this seemed natural (In the beginning was the computer ...), it led to unfortunate "bottom-up" habits, so now more algorithms are introduced before computers.

Also, the compartmentalizing of students according to the field of their interest tended to result in emphasis on that particular field, rather than on the computing topics.

Significantly, this book has also been used in a pure algorithms course, with no programming language or computer experience. There is sufficient intellectual content for such a course, but only the bravest of computing instructors will want to try that at present.

Additionally, this book has been used by individuals in various ways: as a course in "Pascal as a second language," as a "structured" refresher, as a survey of the "algorithmic method" and as a "top-down" companion to a first course in Pascal.

Undoubtedly other uses of this book are possible.

FORM OF THIS BOOK

The form (layout, format, or syntax) of this book differs from that of most other books. I have tried to reflect the "top-down" view in the design of the book. It has some of the same form that large programs have. I realize that I have not been completely successful in achieving this goal, but, as with most programs, there is room for modification. I am grateful to the publisher, Allyn and Bacon, Inc., for cooperation in making this possible.

BOOKS, in general, consist of chapters which are broken into sub-chapters (sections, parts, etc.) which ultimately are "cut" into equal-sized segments, called pages. This arbitrary cutting into pages often causes pages to start and finish in mid-sentence, a practice that I find annoying (except in novels). It's like breaking up one long program into sub-programs at arbitrary points (every 50 lines).

CHAPTERS in this book are broken into sections, and this break-out is shown in front of each chapter. A chapter is "sandwiched" between a preview at the front and a review following it. A set of problems follows each chapter.

SECTIONS are further split into smaller segments, called page-pairs. There may be two to seven of these segments in a section. Seven (plus or minus two) is a "magic" number often encountered in programming.

PAGE-PAIRS are simply the two pages we face when a book is opened up. The two pages are created to be complementary; one page consists of graphics, and the facing page consists of printed text. Each pair is devoted to a single, isolated concept.

CONCEPTS do not always fit a page easily. When a concept is small, the paragraphs are spaced out to fit the page, so forming blocks of text, adding to the readability. When the concept is large, it is split into sub-concepts, which in turn have their own page.

PAGES consisting of graphics are found at the left of a page-pair, and those consisting of text are at the right. This is consistent with the "split-brain" theory, which states that left and right sides of a brain process information differently.

TEXT PAGES are laid out in a structured way. They consist of "blocks" of text, separated by gaps of white space. Titles are not centered, but instead are allowed to "hang over," out of the blocks of text. Centering hides structure; indentation emphasizes it. Page numbers also hang over the top right of each page for easy reference.

PARAGRAPHS have a reverse indentation, like programs, the better to show the structure. The first sentence is the main or topic sentence, and the first word in it is also made the main or key word. This key word, often capitalized, protrudes into the left margin, so serving as a handle for this concept. This page is an example of such structure. Notice all the blocks, gaps, white space, indentation and capitalization.

PRODUCTION OF THIS BOOK

Computers were very widely used in the production of this book. They ranged from a microcomputer (Apple II) to a "monster" computer (CDC Cyber 170-750).

Graphics were produced interactively on a screen, and then run off on a printer. A student, Luis Castro, created the software for the Apple computer, and then lived with his creation to produce most of the graphics for this book ... almost 200 pages! Marie Alanen took part in the initial stages. My son Jan used this graphic system to create the syntax diagrams. Joe Kwan modified this graphics tool, making it more "friendly" to use.

Text portions of the book were produced originally on the Cyber by Ruth Horgan, and modified over many versions. The Runoff system was then re-created by Luis Castro to conveniently produce proportionately spaced text. Joe Kwan adapted the system to the Apple computer.

Portability between such diverse systems was accomplished mainly because all the tools (for graphics, text, indexing, etc.) were created in the Pascal language. The graphics compatibility among the Spinwriter, Diablo, and Dataproducts printers was a pleasant surprise.

Power of this computer production was rather awesome. I had total control over every aspect. For example, I could specify a diagram roughly, and Luis would enter it into the system, making improvements. Then I could preview the diagram on a screen, and refine it further. I could "tweak" any line, moving it as little as a sixtieth of an inch. Finally, I could get a "hard" copy on paper. To have a human artist re-draw a diagram for any slight change would be prohibitive; to specify a computer to do it was simple. This graphing tool (GRAPH), for creating flow block diagrams, is available from the author.

The text part of the book was developed similarly. I would specify the text initially, and Ruth would edit it before entering it. We had many discussions, leading to refinements which were very easily made to the "soft" copy. The text format was also possible to control in detail. I specified the distance between successive characters, the "leading" or space between the lines, and the space (lots of it), between paragraphs (forming blocks of text). This total control led to a novel indentation scheme, similar to that used for programs. Incidentally, the text in the appendices was not spaced proportionately; you may wish to compare it to the rest of the book.

Complexity of details in such a book could easily become overwhelming. Computers again made the complexity manageable, enabling the dozens of programs, hundreds of pages of text, and thousands of graphics to be readily retrieved, modified, and again stored.

SOURCES (Bibliography)

In this book I have borrowed from many sources. Most of these sources are at a higher level; I have tried to present them at an introductory level. For this reason, most of the following references should be read after this book, not along with it.

Programming, in general, is not covered in many books. One of the few language independent books is "FOUNDATIONS OF COMPUTER SCIENCE", written by M.S. Carberry, H.M. Khallil, J.F. Leathrum, L.S. Levy, and published by Computer Science Press.

Pascal is covered in many books, which often purport to cover problem solving and program design. Most go with very great detail into the Pascal language, and much less detail in developing the ability to create algorithms. A concise coverage of Pascal can be found in the classic "PASCAL USER MANUAL AND REPORT" by K. Jensen and N. Wirth, published by Springer-Verlag.

Data structures are most appropriately covered after this book. Niklaus Wirth has also written a book: "ALGORITHMS + DATA STRUCTURES = PROGRAMS" published by Prentice-Hall. It is concise but at a rather high level.

Machines are well treated in a book by J. Ullman: "FUNDAMENTAL CONCEPTS OF PROGRAMMING SYSTEMS" published by Addison-Wesley. It also links machines to languages.

Languages in general are treated in a unified way using contour diagrams by E.I. Organick, A.I. Forsythe and R.P. Plummer in their book entitled "PROGRAMMING LANGUAGE STRUCTURES" published by Academic Press.

Programming principles of a more theoretical nature are further developed by W. Wulf, M. Shaw, P. Hilfinger and L. Flon in their book "FUNDAMENTAL STRUCTURES OF COMPUTER SCIENCE" published by Addison-Wesley.

Design of programs in a structured way is developed by J.D. Warnier in "LOGICAL CONSTRUCTION OF PROGRAMS". The diagrams developed there have evolved into the break-out diagrams of this text.

Problems, 777 of them, can be found in the text "GRADED PROBLEMS IN COMPUTER SCIENCE" by A.D. McGettrick and P.D. Smith, and published by Addison-Wesley.

Notation for representing algorithms in flow block form evolved from a paper by I. Nassi and B. Shneiderman in the ACM SIGPLAN Notices (Volume 8, Number 8).

Mathematical and engineering systems concepts underlying many discrete structures (Boolean algebra, discrete probability, sequential machines and stochastic systems) are developed in the book "DIGITAL SYSTEMS FUNDAMENTALS" written by John Motil, and published by McGraw-Hill (hardcover) and Ridgeview Press (paperback).

THANKS (Acknowledgments)

Many persons have contributed to this book, and I wish to thank them all. None, however, is responsible for any errors. I had total control over that, and I accept any blame.

First, I must thank Ruth Horgan and Luis Castro, two very hard-working people, without whom I could not have completed this book. Their work has been described on a previous page.

Faculty at California State University, Northridge (CSUN) have taught from many early versions, and offered suggestions, changes, and encouragement. They include: Jack Alanen, Morteza Anvari, Ronald Colman, Raymond Gumb, Robert Henderson, Gary Hordemann, Ruth Horgan, Dorothy Landis, Diane Schwartz, Linda Stanberry, and Wigberto Yu. Raymond Davidson, Steve Gadomski, Philip Gilbert, Steven Stepanek, and Rein Turn suffered through very early versions. Other faculty who did not teach this course also contributed, including Russell Abbott, Shan Barkataki, Michael Barnes, Fred Gruenberger, Kenneth Modesitt, Peter Smith, and David Salomon.

Part-time faculty also contributed, especially Richard Kaplan, Robert Lingard, Georgia Lulovics, Robert McCoard, and Albert Pierce.

Visiting faculty members also taught from this book and provided differing insights. Thanks to John Van Iwaarden of Hope College, Michigan, Paul Tavolato from the University of Vienna, and David Brailsford from the University of Nottingham.

Professors from other departments often audited this course, and provided unusual feedback. Thanks to Ernest Scheuer of Management Science, Donald Bianchi of Biology, Edward Hriber and Virgil Metzler of Engineering, Felix Jumonville and Larry Krock of Physical Education, Donald Wood of Radio-TV-Film, and Jerrold Gold and Joel Zeitlin of Mathematics. Richard Truman of Management Science read some early chapters and eliminated many errors. Jeffrey Sicha of Philosophy was particularly helpful in many ways.

Administrators at CSUN also helped in their own ways despite severe resource problems. Thanks to two deans, Charles Sanders and A. F. Ratcliffe, and to five department chairs (named above) in the last few years. Sandra Metzger was instrumental in finding equipment from various local industries. Use of donated equipment from Dataproducts and Marketron was appreciated.

Participants in a Faculty Development Program were very helpful, especially about pedagogical matters and the instructor's guide. They include George Lorbeer of Secondary Education, Nathan Weinberg of Sociology, Thomas Bader, Thomas Maddux, Alexander Muller, and Michael Patterson of History. Other participants were George Douglas of Religious Studies, Richard Smith and James Torcivia of Psychology and William Vincent of Physical Education. Additional faculty included Beverly Grigsby of Music, John Miller of Management Science, and Robert Noreen from the department of English.

MORE THANKS

Secretaries LuAnne Rohrer and Sally Gamon were very helpful, along with many student assistants. Nick Dalton spent many hours copying the graphics. The School technicians, Jack Siano, Bob Allen, Jan Berreitter, Herb Petzold, and Dennis Tibbetts, provided prompt and cheerful aid.

Other colleges have tried preliminary versions, and feedback was appreciated from Margaret Brennan of Glendale Community College, and Ken Stevens of The College of the Canyons. Graduate and senior students Darel Roberts and Ken Clark also contributed ideas and programs.

Reviewers of the manuscript had very helpful advice (some of which I did not take). Thanks to Gary Ford of Arizona State University, Madeleine Bates of Bolt Beranek and Newman, and David Boswell of the University of Waterloo. Very early reviews by Brian Hansche of Arizona State University were useful. Michael Meehan, my original editor, was particularly insightful. Other helpful people from Allyn and Bacon were Gary Folven, Doug Hinchey, Nancy Murphy and Paul Solaqua.

Students from many classes were very good at detecting errors. Encouraged by a small reward for each error or inconsistency, they helped improve the book. I found it interesting that some errors and inconsistencies were not detected by hundreds of students over the years. Some of these imperfections have not been removed in the spirit of "shaboui" - nothing is perfect unless it has some imperfection.

Computer Center staff, directed by Jerry Boles, were often very helpful. Thanks to Mona Clark, Gary Cohen, Jeff Craig, Ann Fuller, Joyce Hayes, J.P. Jones, Nancy Murry, Dave Sansom, Dave Thompson, and Kurt Webb.

Computer store owners Russ and Gene Sprouse from Rainbow Computing provided good information and service when it was badly needed. Lee Castile of California Press was very helpful with the final graphics.

I must also acknowledge Stan Rifkin, who (among other things) initially introduced me to the concept of Structured Programming, by arranging for me to attend a short course given by Edsger Dijkstra. There I was "born again" to programming, which I had previously come to dislike. I also feel privileged to have attended a course taught by Niklaus Wirth.

Finally, I must thank my family and friends for bearing with me during this "obsession."

John Motil