COMPUTER SCIENCE

A Breadth-First Approach with Pascal

PAUL NAGIN

JOHN IMPAGLIAZZO

COMPUTER SCIENCE

A Breadth-First Approach with Pascal

Paul Nagin
John Impagliazzo
Hofstra University
Hempstead, New York

John Wiley & Sons
New York Chichester Brisbane Toronto Singapore

ACQUISITIONS EDITOR Steven Elliot
MARKETING MANAGER Susan Elbe
SENIOR PRODUCTION EDITOR Nancy Prinz
COVER DESIGNER Dawn L. Stanley
INTERIOR DESIGN Levavi & Levavi
COVER ART Otherworld Artyfax
MANUFACTURING MANAGER Susan Stetzer
ILLUSTRATION COORDINATOR Rosa Bryant

This book was set in ITC Garamond Light by Publication Services and printed and bound by R. R. Donnelley & Sons, Crawfordsville. The cover was printed by Phoenix.

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

Library of Congress Cataloging-in-Publication Data
Nagin, Paul A.
Computer science : a breadth-first approach with Pascal / by Paul Nagin and John Impagliazzo.
p. cm.
Includes index.
ISBN 0–471–31198–7 (paper : acid-free paper)
1. Computer science. 2. Pascal (Computer program language)
I. Impagliazzo, John. II. Title.
QA76.N273 1995
004—dc20 94-45368
CIP

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

Preface

The computing sciences are in their infancy. Although electronic computing machines have been in existence since the 1930s, this area of study was not formalized until the late 1960s when the Association for Computing Machinery (ACM) published its first curriculum recommendations for four-year programs in computer science. "Curriculum '68—Recommendations for Academic Programs in Computer Science"[1] encouraged the study of discrete mathematics and calculus and proposed a set of computing courses. "Curriculum '78—Recommendations for the Undergraduate Program in Computer Science"[2] updated the previous recommendations of 1968 in response to the rapidly changing field of computing. In 1984 and 1985 ACM published "Recommended Curriculum for CS1"[3] and "Recommended Curriculum for CS2,"[4] which encouraged use of data structures and software design and implementation in the early stages of a computer science curriculum.

The curricula recommendations in the computing sciences were becoming reactive rather than proactive as educators responded to the needs of a changing computing profession. In the late 1980s a special task force was created to address this issue. It published the "Report of the ACM Task Force on the Core of Computer Science,"[5] also called the "Denning Report." The report established topic areas that define the discipline this way:

> *The discipline of computing is the systematic study of algorithmic processes that describe and transform information: their theory, analysis, design, efficiency, implementation, and application. The fundamental question underlying all of computing is, "What can be (efficiently) automated?"*

A more general definition of computer science from the Denning Report incorporates the paradigms of *theory* (mathematical approach), *abstraction*

[1]ACM Curriculum Committee on Computer Science. "Curriculum '68—Recommendations for Academic Programs in Computer Science." *Communications of the ACM, 11,* 3 (March 1968), 151–197.

[2]ACM Curriculum Committee on Computer Science. "Curriculum '78—Recommendations for the Undergraduate Program in Computer Science." *Communications of the ACM, 22,* 3 (March 1979), 147–166.

[3]Koffman, Elliot B., et al. "Recommended Curriculum for CS1: 1984." *Communications of the ACM, 27,* 10 (October 1984), 998–1001.

[4]Koffman, Elliot B., et al. "Recommended Curriculum for CS2: 1984." *Communications of the ACM, 28,* 8, (August 1985), 815–818.

[5]Denning, Peter, et al. "Report of the ACM Task Force on the Core of Computer Science." ACM Press, New York, 1988. Also known as the "Denning Report." Reprinted in part in *Communications of the ACM, 32,* 1 (January 1989), and in *Computer* (February 1989).

(experimental approach), and *design* (engineering approach) within the following nine topic areas:

Algorithms and data structures
Programming languages
Architecture
Numerical and symbolic computing
Operating systems
Software methodology and engineering
Database and information retrieval
Artificial intelligence and robotics
Human-computer communication

The report also encouraged sensitivity to the social context of computing.

Previous curriculum recommendations promoted *depth* in the beginning stages of the curriculum through the early study of and concentration on programming. The Denning Report suggests *breadth* in the study of computing as a whole in the first stages of the discipline coupled with laboratory experiences, similar to the study of biology, chemistry, and physics.

The ACM and the Computer Society of the Institute for Electrical and Electronic Engineers (IEEE-CS) embraced the Denning Report. In 1991 both societies jointly published "Computing Curricula 1991,"[6] which serve a variety of computing programs. These recommendations do not prescribe courses for study. Instead they decompose the nine topic areas into knowledge units that can be arranged for an individual program of study. Such a program should consist of the nine topic areas and reflect one or more of the paradigms of theory, abstraction, and design. Many of these concepts are also reflected in other ACM publications such as *Computing Curricula Guidelines for Associate Degree Programs*[7] and the *Model High School Computer Science Curriculum.*[8]

GOALS OF THIS TEXT

This text arose from the emerging demand for an introductory, broad-range, or *breadth-first* book on computer science. We believe that using the breadth-first

[6]Tucker, Allen B., et al. "Computing Curricula 1991—Report of the ACM/IEEE-Computer Society Joint Curriculum Task Force," ACM Press, New York, 1991. Reprinted in summary in *Communications of the ACM* (June 1991), 68–84.

[7]ACM Two-Year College Computing Curricula Task Force. "Computing Curricula Guidelines for Associate Degree Programs—Computing Sciences," ACM Press, New York, 1993.

[8]ACM Task Force of the Pre-College Committee. "ACM Model High School Computer Science Curriculum," ACM Press, New York, 1993.

approach will not interfere with the remainder of the typical undergraduate computer science curriculum. Rather it should increase class motivation since students will have an overall sense of the field instead of merely a narrow-band knowledge of programming.

Breadth-First versus Depth-First The traditional, depth-first approach to teaching computer science generally consists of a two- or three-semester exposure to a particular procedural language like Pascal with emphasis on syntax, algorithm design, and data structures. Other topics in computer science are generally ignored. This *programming approach* reinforces the misconception that computer science is the study of programming syntax and applications. Although it is true that programmers do (mostly) programming, computer scientists deal with *computing*, using programming as a tool with which to explore and develop ideas.

The breadth-first approach, on the other hand, gives exposure to the essential elements of computing. Selected topics include: machine architecture, algorithms, data communications, complexity theory, database design, artificial intelligence, information retrieval, and software engineering. Programming is integrated throughout the topics as a tool for exploring these aspects of the field. Depth of knowledge is relegated to other courses taken in the remaining years of undergraduate and graduate study.

We believe that students should be exposed to the various aspects of computing early in their education and sample the breadth of the discipline so that they will have a clearer understanding of what the field comprises. It is for this reason that we adhere to the principles promoted in the "Denning Report" and in "Computing Curricula 1991."

NOTE TO THE PROFESSOR

Philosophy For many instructors, teaching a breadth-first course in computer science is a novelty. The range of topics may even seem intimidating, especially when they are intended for an audience of first-year majors. Experience has shown us, however, that students gravitate to the breadth topics of computer science, which stimulate new ideas for them and generate meaningful discussions.

We do not mean to teach beginning students all of computer science or all the intricacies of the Pascal language in a one-year course. It is more important to let students sample areas of the discipline in small doses and develop a basic understanding of Pascal. Some topics are taken to a greater level of detail to give instructors more flexibility in the presentation of the material. This detail can be omitted without loss of continuity.

Implementation The material in the text is designed for first-year undergraduate majors and minors in computer science for presentation over two semesters. Accelerated programs can also use the text for a one-semester course. Upon

completion, students will have developed a broad foundational knowledge of the principal elements in the science of computing and a working knowledge of Pascal.

The following table provides suggested coverage for a one-year sequence with three possible tracks. The A-track is the most complete, needing full coverage

CHAPTER	SECTIONS A-TRACK	SECTIONS B-TRACK	SECTIONS C-TRACK
1	1, 2, 3	1, 2, 3	1, 2, 3
2	1, 2	1, 2	1, 2
3	1, 2, 3	1, 2, 3	1, 2, 3
4	1, 2, 3	1, 2	1
5	1, 2, 3	1, 2, 3	1, 2, 3
6	1, 2	1, 2	1
7	1, 2, 3	1, 2, 3	1, 2, 3
8	1, 2	1, 2*	1
9	1, 2	1, 2	1, 2
10	1, 2	1, 2	1
11	1, 2, 3	1, 2, 3	1, 2, 3*
12	1, 2	1	1
13	1, 2, 3	1, 2	1, 2*
14	1, 2	1, 2	1
15	1, 2	1, 2*	1
16	1, 2	1, 2*	1
17	1, 2, 3*	1, 2*	1
18	1, 2	1*, 2	2*
19	1, 2	1, 2*	1
Required sections	45	36	29

with one optional section. The B-track offers more flexibility, with 10 sections that are optional. The C-track is the minimum necessary coverage for a breadth-first approach. Asterisks (*) show optional sections. All tracks ensure sufficient coverage of programming methodology and design.

Students completing this text will have gained a general understanding of the significant topic areas of study in the field of computer science. In addition, they will have a substantial preparation in Pascal programming. We believe that the integrated breadth-first approach gives students a strong foundation in the subject and the skills to become knowledgeable and effective computer scientists. This knowledge will be a solid platform on which students can build a career in computing.

SUPPLEMENTS

The following materials are available to accompany the text.

1. An instructor's manual with suggested lesson plans and solutions to most exercises.
2. A computer disk for instructors containing program samples and suggested questions for examinations.

ACKNOWLEDGMENTS

We wish to acknowledge the contributions made to the manuscript by our students, colleagues, reviewers, and editors. We thank the many students who participated in the class testing of the material and in particular Alan P. Baker, Andrew Botwinick, Richard N. Gruenfelder, Jeanette R. Sones, David J. Stecher, and Frank P. Tufano for their assistance in developing and testing some of the programs. A special thanks to our Hofstra colleagues, Thomas B. Steel, Jr. and Olga Salizkiy, for their comments and suggestions in the early stages of manuscript development.

The reviewers of the manuscript made fine suggestions that were incorporated in this text. We extend our heartfelt thanks to:

Lillian Cassel
Villanova University

Marsha Moroh
College of Staten Island

Karl J. Klee
Jamestown Community College

Vaidy S. Sunderam
Emory University

Ann Ford
University of Michigan

Ken Collier
Northern Arizona University

Maria Petrie
Florida Atlantic University

Robert D. Campbell
Manatee Community College

John D. McGregor
Clemson University

Eleanor Quinlan
Ohio State University

Linda Werner
University of California: Santa Cruz

Clifford Shaffer
Virginia Tech

Elizabeth Adams
Richard Stockton College

Dennis J. Frailey
Southern Methodist University

Our work was made easier because of their comments and contributions. To these professionals we are most grateful.

We also wish to thank those associated with John Wiley & Sons. We are much appreciative of the support given by our publisher, Wayne Anderson, our

executive editor, Charity Robey, and by Nancy Prinz, Christopher Curioli, Dawn Stanley, Susan Elbe, and Lisa Passmore. A very special thanks to our development editor Judith Goode for her insightful comments and her meticulous professional skill. We are most grateful to our editor, Steven Elliot, for his many hours of personal dedication to this work.

Last but not least, we would like to thank our families for their untiring support.

To the Student

You are probably beginning this course with some exposure to computers and introductory programming. This exposure can manifest itself in many forms, from casual self-taught knowledge to formal courses. Additionally, you are expected to have experienced rudimentary problem solving through the study of high school mathematics and science.

It would be beneficial for you to study discrete mathematics concurrently with this book. That branch of mathematics presents many concepts such as sets, logic, graphs, and recursion that are essential to your understanding of computing. Although some introductory discrete mathematics is included in the book, we believe that these concepts are important enough for you to study them in depth in a separate setting.

PEDAGOGICAL FEATURES OF THE TEXT

The text contains the following pedagogical features to help you learn and apply basic concepts.

Chapter Opener Each chapter begins with a *photograph* related to the theme of the chapter. The chapter opener contains two features: *Learning Goals* and *Chapter Activity*. *Learning Goals* are a series of statements that highlight what you learn from the chapter. *Chapter Activity* is some endeavor or problem that you should be able to do after completing the chapter. This is intended as a "sneak preview" of the level of the material and provides a framework for the rest of the chapter.

Section Activities Each section contains activities to engage the reader. These include:

> ***Perspective***—*short vignettes with historical perspectives and curiosities, social and ethical issues, mathematical reminders, programming tips, human-computer communication issues, and other topics of interest. These exposures add different views of the subject and raise the social consciousness of those entering the profession.*
>
> ***Before You Go On***—*short, on-the-spot exercises whose solution can be drawn from the text immediately preceding them.*

Exercises Each section ends with a set of exercises of increasing difficulty in a variety of formats. The exercises are arranged as follows.

A Concept Check *consisting of 12 short questions grouped into true-false, multiple choice, and fill-in formats. Answers to these questions can be easily extracted from the text.*

The *Concept Check* is followed by three sets of open-ended questions of increasing difficulty.

Set A questions *are relatively easy and all students should be able to find the right answers with a minimum of effort.*

Set B questions *are moderate to challenging, occasionally needing substantial time and effort.*

Set C questions *often include laboratory exercises to be done using the computer. The suggested laboratory activities in Set C are designed to motivate and engage you in closed as well as open laboratory activities.*

Answers to many of these exercises are given after the appendixes.

The appendixes include references to the Pascal language, reserved words, standard names, operators, ASCII and EBCDIC character codes, and a derivation of Simpson's rule. An index is also supplied.

ACTIVE LEARNING

Computer science can only be mastered by active participation on the part of the student. This means carefully reading the material, doing as many of the exercises as possible, participating in classroom discussions, and dedicating sufficient time to prepare and properly execute computer programs. There are no shortcuts.

Computer science is filled with thorny technical and theoretical issues needing analytical, nonintuitive thinking. Many of the real-world problems currently under investigation may take years or decades to solve, if ever! This new and rich area of study needs patience and perseverance. Try not to get too frustrated if things do not fall into place the first time you encounter them. Be persistent and ask questions. Become an active student!

PAUL NAGIN
JOHN IMPAGLIAZZO

1995 January

Brief Contents

Contents