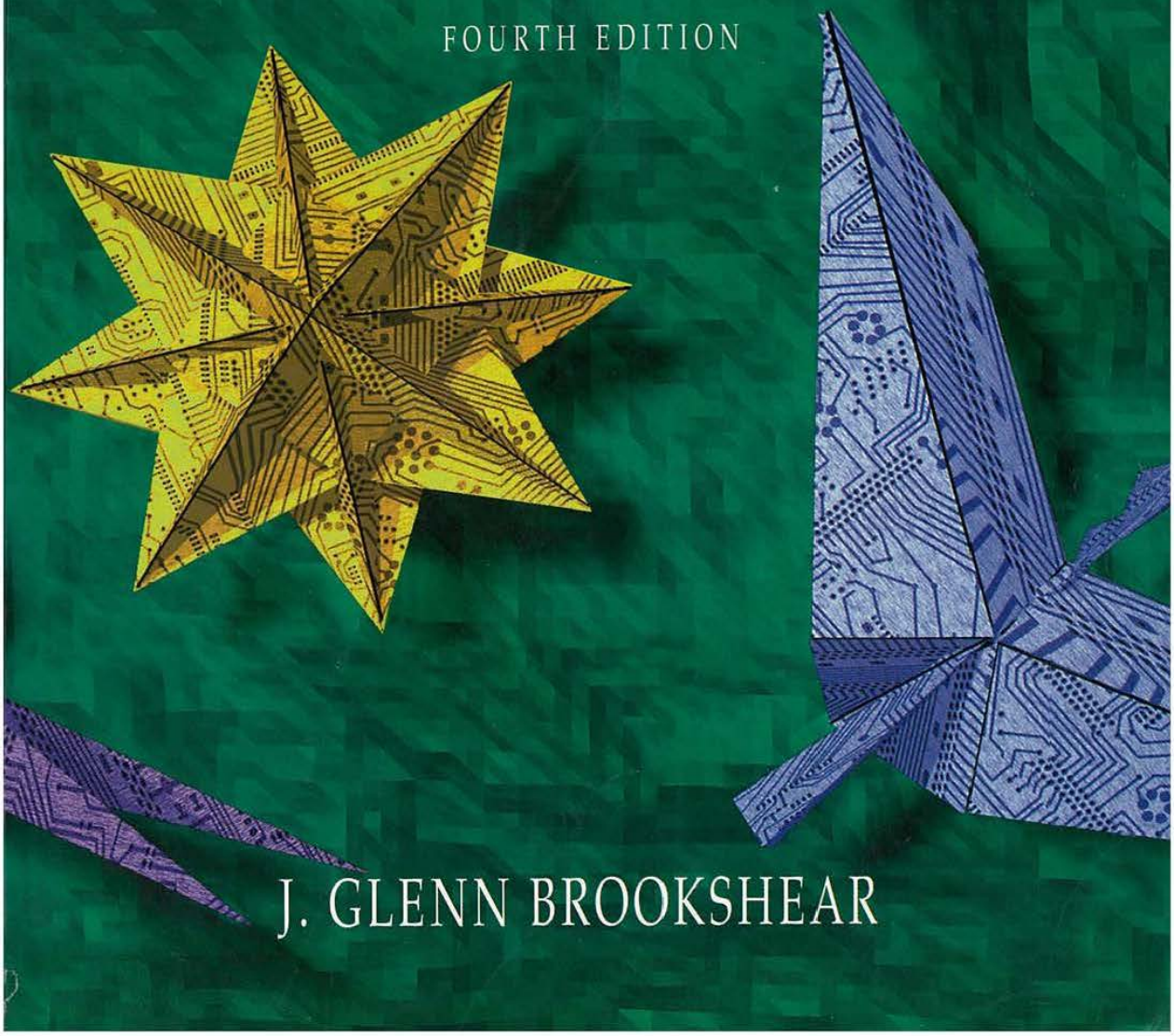
COMPUTER SCIENCE

AN OVERVIEW

FOURTH EDITION



J. GLENN BROOKSHEAR

FOURTH EDITION

COMPUTER SCIENCE

AN OVERVIEW

SELECTED TITLES FROM THE BENJAMIN/CUMMINGS SERIES IN COMPUTER SCIENCE

F. Carrano/P. Helman/R. Veroff
Data Structures and Problem Solving with Turbo Pascal: Walls and Mirrors (1993)

A. Kelley and I. Pohl
A Book on C: Programming in C, Second Edition (1990)

A. Kelley and I. Pohl
C by Dissection: Essentials of C Programming, Second Edition (1992)

I. Pohl
C++ for C Programmers, Second Edition (1994)

I. Pohl
C++ for Pascal Programmers (1990)

W.J. Savitch
Pascal: An Introduction to the Art and Science of Programming, Third Edition (1990)

W.J. Savitch
Turbo Pascal: An Introduction to the Art and Science of Programming, Fourth Edition (1993)

FOURTH EDITION

COMPUTER SCIENCE

AN OVERVIEW

J. Glenn Brookshear
Marquette University

The Benjamin/Cummings Publishing Company, Inc.

Redwood City, California • Menlo Park, California
Reading, Massachusetts • New York • Don Mills, Ontario • Wokingham, U.K.
Amsterdam • Bonn • Sydney • Tokyo • Madrid • Spain

To my parents
Garland and Reba Brookshear

Sponsoring Editor: Carter Shanklin
Editorial Assistant: Melissa Standen
Production Coordinator: Andy Marinkovich
Cover Design: Yvo Riezebos
Copyeditor: Barbara Conway
Proofreader: Holly Mclean-Aldis
Artists: Ben Turner Graphics
Composition: Graphic World

Library of Congress Cataloging-in-Publication Data
Brookshear, J. Glenn.
Computer science : an overview / J. Glenn Brookshear. — 4th ed.
p. cm.
Includes bibliographical references and index.
ISBN 0-8053-4627-9
1. Computer science. I. Title.
QA76.B743 1993 93-24520
004—dc20 CIP

12345678910-DOCR-97 96 95 94 93

The Benjamin/Cummings Publishing Company, Inc.
390 Bridge Parkway
Redwood City, CA 94065

PREFACE

I wrote this book to provide a comprehensive overview of computer science, one that presents a thought-provoking introduction to the key issues and concepts throughout the field. I have done this with two primary audiences in mind.

Computer Science Majors

The first audience consists of computer science majors and minors in the early stages of their college careers. Students at this stage tend to equate computer science with programming because that is essentially all they have seen. Yet computer science is much more than programming. In turn, beginning computer science students need to be exposed to the breadth of the subject in which they are planning to major. Providing this exposure is the purpose of this book. It gives students an overview of computer science—a foundation from which they can understand the relevance and interrelationships of future courses. Without such a perspective, students easily become immersed in the details of specialized courses and never understand the true scope and dynamics of the field. In short, this book represents the application of top-down methodologies, as taught within the curriculum, to the computer science curriculum itself.

A lot has happened since the first edition of this book. Today, the computer-science-is-much-more-than-programming philosophy is widely endorsed, as witnessed by the famous Denning Report and, more recently, the report of the ACM/IEEE-CS Joint Curriculum Task Force. Those who subscribe to this movement will be pleased that this fourth edition continues the tradition of the preceding ones in that its content conforms closely to the subject areas of computer science as identified in these reports. This text provides students of computer science with an accessible introduction to the breadth of their subject, all within a single volume.

Students of Other Disciplines

I also designed this book with majors of others fields in mind. Too often, these students are channelled into courses that teach them merely how to use some software packages or provide an elementary introduction to programming. Unfortunately, the subject matter of these courses is often time-sensitive, limited in portability, or not

developed to a depth to be useful outside the classroom. Any benefits from such courses dissipate quickly after the semester is over.

I believe that these students are seeking "computer literacy," which I loosely define as the ability to distinguish between computer science and science fiction. Providing this level of "literacy" in their respective fields is the purpose of such courses as general chemistry, biology, and physics. Students do not take these courses to develop specific skills. Rather, the major goal is to develop an understanding of the discipline—including its scope, major results and consequences, research techniques, and the current status of the field. The fact that a student might be required to develop certain skills while taking the course is merely a temporary consequence. The true benefit of the course—obtaining an overall picture of the subject—survives long after these specific items have been forgotten.

Why, then, do we insist that a computer science course for nonmajors emphasize skills? The goal should be to present an overall picture of the science, which is exactly what I have designed this book to provide. After taking a course based on this text, a student will have obtained an understanding of the science behind today's computerized society. This understanding will remain long after the details and skills "memorized" during the semester have dissipated. Indeed, the student will have been educated rather than trained.

The Fourth Edition

In addition to numerous minor changes designed to update, correct, or generally improve the text, this fourth edition differs from the third in the following, more significant ways.

- The role of abstraction and abstract tools is explicitly presenting as a recurring theme.
- Ties to social, ethical, and professional issues have been expanded.
- The material on networks in Chapter 3 has been expanded to include an introduction to the OSI reference model and its significance.
- The material on parallel computing has been expanded, including a new section on parallel programming using the Linda primitives in Chapter 5.
- A new section in Chapter 6 discusses the role of metrics in software engineering.
- The object-oriented paradigm has been given additional emphasis by means of specific examples using Ada and C++ in Chapter 7 and a new section on object-oriented databases in Chapter 9.
- Significantly more exercises have been added.
- Manuals for closed laboratories are now available in the languages Pascal and C.

Pedagogical Features

This text is the product of many years of teaching the material. As a result, it is rich in pedagogical aids. Paramount in this regard is the abundance of problems to enhance the student's participation. Each section within a chapter closes with several Questions/Exercises to challenge students to think independently. They review the material just discussed, extend the previous discussion, or hint at related topics to be covered later. These questions are answered in Appendix F.

Each chapter concludes with a collection of Chapter Review Problems. These problems are designed to serve as "homework" problems in that they call for specific answers, can be solved in a short period of time, and are not answered in the text.

Following the Chapter Review Problems are Problems for the Programmer. These problems are designed for students who already have a programming background and serve to enhance the student's problem-solving/program-development skills as well as provide additional insights into the material in the chapter. If desired, many of these problems can be expanded into programming projects. These problems are an excellent resource when the book is used as a text for a course following the traditional introductory programming course.

Another pedagogical aid is the use of optional sections. These sections are marked in the table of contents. The fact that a section is declared optional does not mean that its material is necessarily more difficult or should be skipped. It merely means that the material in later (nonoptional) sections does not rely on these sections. The purpose of identifying these sections is to allow students to reach later portions of the text more quickly than would otherwise be possible. For example, many instructors may wish to skip or postpone much of the material on machine architecture and operating systems in order to spend more time on algorithm development and representation as discussed in chapters 4 and 5. The use of optional sections allows for this change yet leaves the material available for the more inquisitive students or courses with different goals.

Laboratory Materials

Supplementary laboratory manuals that are coordinated with the text are available for courses with an introductory-level programming component. These manuals, one for the language Pascal and the other for C, are designed for a closed laboratory that meets once a week for approximately two hours. Each manual contains material for 16 laboratory sessions (many are optional) that teach the rudiments of the particular programming language and provide experiments that reinforce material in the parent text.

Each laboratory session consists of explanatory material, activities for the student

that are presented in a true experiment format that encourages investigation, and post-laboratory problems that ask students to apply their knowledge outside the closed laboratory environment.

The laboratory manuals are supported by software that is available from The Benjamin/Cummings Publishing Company via the Internet using ftp. The address is bc.aw.com. When asked for a name, respond by typing anonymous; when asked for a password, respond with your own address. From the directory in which you will be placed, the software is two directories down along the path bc/brookshear. (For non-UNIX readers, you get to this directory by typing cd bc/brookshear.) For more details, consult the 00README file in the brookshear directory. (Again, for the non-UNIX crowd, type get 00README to download this file to your local environment and type bye to terminate the connection.)

Acknowledgments

With each new edition, the list of those who have contributed through their suggestions and comments continues to grow. Today this list includes J. M. Adams, D. C. S. Allison, P. Bankston, M. Barnard, K. Bowyer, P. W. Brashear, C. M. Brown, B. Calloni, M. Clancy, D. H. Cooley, F. Deek, M. J. Duncan, N. E. Gibbs, J. D. Harris, D. Hascom, P. Henderson, L. Hunt, L. A. Jehn, K. Korb, G. Krenz, T. J. Long, C. May, S. J. Merrill, J. C. Moyer, J. Paul Myers, Jr., G. Rice, N. Richert, J. B. Rogers, J. C. Simms, M. C. Slattery, J. Slimick, D. Smith, J. Solderitsch, L. Steinberg, J. Talburt, P. Tromovitch, and M. Ziegler. To these individuals I give my sincere thanks. A special thank you goes to Phil Bender and Jody Jung for writing the laboratory manuals.

As in the case of the earlier editions, I also thank my family, Earlene and Cheryl, for their support. They have seen how the development of a manuscript can expand to dominate an author's time. I thank them for their understanding and patience.

J.G.B.

CONTENTS

* Sections marked by an asterisk are optional in that they provide additional depth of coverage that is not required for an understanding of future chapters.

Sometimes the action requested by a program may be meaningful even though the data types involved are not the same. For instance, the above instruction makes sense if Price and Tax are integer but Total is real. In this case, the compiler uses the integer addition instruction, but the sum must be recoded into floating-point format before being assigned to Total. Such implicit conversion between types is called *coercion.*

Coercion is frowned on by many language designers. They reason that the need for conversion usually indicates an error in the program's design and therefore should not be accommodated by the translator. The result is that most modern languages are *strongly typed,* which means that all activities requested by a program must involve data of agreeable types without coercion. In turn, compilers for these languages report all type conflicts as errors.

Data Structure

Another major concept associated with data is *structure,* which relates to the conceptual shape of the data. Perhaps the simplest example of this occurs when using a string of characters to represent an employee's name or a part identification number. It is not sufficient to know that the data item is of type character, but one must also know how many characters make up the item. If a translator must generate the machine instructions to move an employee's name from one location in memory to another, it must know how many memory cells to move. Thus, a FORTRAN program might contain the phrase

```
CHARACTER(LEN = 8) Name
```

indicating that Name is to refer to a string of eight characters. The same information would be expressed Ada as

```
Name: STRING(1..8);
```

Another common example of structured data is an *array.* The term *array* refers to a block of values such as a list (often called a vector), a two-dimensional table (a matrix), or tables of higher dimensions (these do not have special names). Elements of an array normally are identified within a program through the use of indices. That is, the third entry in a vector named Sales is referenced by the expression Sales (3), and the entry from the second row and fifth column of a matrix named Scores is identified by Scores(2,5). (A minor exception is found in the C language, in which row and column numbers start at 0 rather than 1. Hence, in C, the entry in the second row and fifth column is identified by Scores [1][4].) Note that it is customary to list the row number before the column number.

To describe an array in the declarative part of a program, most languages use a syntax similar to that used for referring to the array later in the program's procedural

We have used the terms *virtual* and *conceptual* several times in reference to properties that, although appearing to belong to hardware, are actually simulated through a combination of hardware and software. For example, we saw that a single machine can appear to be many machines through the use of a time-sharing system or that a machine can appear to understand the words in a high-level programming language by means of an interpreter. This chapter is concerned with another conceptual feature, the structure (or organization) of data.

Recall that any information stored in a machine's memory must be organized to fit into a row of memory cells, even though this data may be more useful as a rectangular table of values. In this case, our problem is to simulate the rectangular shape using the tools provided by the machine. The goal is to allow the user of the data to think of the data as having this simulated shape without being concerned with the data's actual organization within the machine.

7-1 Arrays

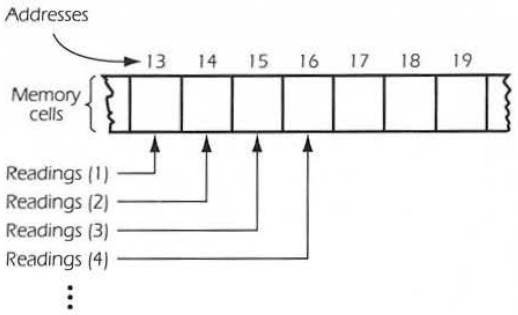
We begin our study of data structures by considering the organizations known as arrays encountered earlier in our discussion of high-level programming languages. There we saw that many high-level languages allow a programmer to express an algorithm as though the data being manipulated were stored in a rectangular arrangement; the programmer might refer to the fifth element in a one-dimensional array or the element in the third row and sixth column of a two-dimensional array. Since the array is actually stored in the memory cells of the machine, it becomes the job of either the translator or the interpreter to convert such references into the terminology of memory cells and addresses.

One-Dimensional Arrays

Suppose an algorithm for manipulating a series of 24 hourly temperature readings is expressed in a high-level language. The programmer would probably find it convenient to think of these readings arranged as a one-dimensional array, that is, a list called Readings whose various entries are referenced in terms of their position in the list. This position is often called an index. The first reading might be referenced by Readings[1], the second by Readings[2], and so on.

The conversion from the conceptual one-dimensional array organization to the actual arrangement within the machine can be rather straightforward. The data can be stored in a sequence of 24 memory cells in the same order envisioned by the programmer. Knowing the address of the first cell in this sequence, an interpreter or translator can then easily convert terms such as Readings[4], into the proper memory terminology. In this case, to find the actual address, one merely subtracts one from the position of the desired entry and then adds the result to the address of the first

Figure 7-1 The array of Readings stored in memory starting at address 13



cell in the sequence. (If the first cell in the sequence is at address 13, the reading referenced by Readings[4] is located at location 13 + (4 − 1) = 16, as shown in Figure 7-1.)

Multidimensional Arrays

The conversion is not quite so simple with multidimensional arrays. Consider, for example, a record of the sales made by a company's sales force during a one-week period. We can think of such data arranged in tabular form, with the names of the sales personnel listed down the left side and the days of the week listed across the top. Hence we can think of the data being arranged in rows and columns; the values across each row indicate the sales made by a particular employee, while the values down a column represent all the sales made during a particular day. Extracting information from the table therefore involves finding the value common to both a given row and a given column.

A machine's memory is arranged not in a rectangular fashion but rather as a row of memory cells; thus, the rectangular structure required by the sales table must be simulated. To do this, we first recognize that the size of the array does not vary as updates are made. We can therefore calculate the amount of storage area needed and reserve a block of contiguous memory cells of that size. Next, we store the data in the cells row by row. That is, starting at the first cell of the reserved block, we copy the values from the first row of the table into consecutive memory locations; following this we copy the next row, then the next, and so on (Figure 7-2, on the following page). Such a storage system is said to use ***row major order*** in contrast to ***column major order,*** in which the array is stored column by column.

With the data stored, the problem now becomes locating particular entries as they are requested. Recall that because the user will be thinking in terms of rows and columns, a request will be in the form of wanting, for example, the value of the entry in the third row and fourth column (that is, the sales made by the third employee on

Computer Science/General

The conceptual framework beginners need—proven successful through three editions!

COMPUTER SCIENCE: AN OVERVIEW, Fourth Edition • J. Glenn Brookshear, *Marquette University*

The new Fourth Edition of *Computer Science: An Overview* builds on its success with up-to-date coverage of today's advances in computer science and technology. Offering a chapter-by-chapter, course-by-course overview of the computer science curriculum, Brookshear communicates the central ideas of complex topics through clear, crisp explanations that relate to a student's intuition. He also emphasizes important, unifying themes throughout, such as abstraction and the role of abstract tools, to illustrate the conceptual connections among the major topics in the field.

NEW IN THIS EDITION:

- Expanded coverage of object-oriented concepts, including new material on object-oriented programming and databases.
- Expanded coverage of networks, including an explanation of the OSI reference model that's accessible to introductory students.
- Coverage of recent advances in such areas as parallel programming, artificial intelligence, and software engineering.
- Integrated discussion of social, ethical, and legal issues related to computer science.

Computer Science: An Overview may be used as a primary text or as a supplement to any programming text. New laboratory manuals, *Experiments in Computer Science*, available in Pascal or C versions, teach the fundamentals of the chosen language and provide hands-on experience with topics covered in the text.

THE BENJAMIN/CUMMINGS PUBLISHING COMPANY, INC.