INTRODUCTION TO COMPUTER SCIENCE

THIRD EDITION

Neill Graham

Introduction to Computer Science

THIRD EDITION

Neill Graham

West Publishing Company
St. Paul New York San Francisco Los Angeles

Copyediting:
Deborah Drolen Jones

Interior design:
Biblio Book Design

Artwork:
Editing, Design & Production, Inc.

Composition:
The Clarinda Company

Cover photograph:
Floyd Rollefstad, Laser Fantasy Production

Printed in the United States of America

Library of Congress Cataloging in Publication Data

Graham, Neill, 1941–
Introduction to computer science.

Bibliography: p.
Includes index.
1. Electronic digital computers. 2. Electronic data processing. I. Title.
QA76.5.G658 1985 001.64 84-21940
ISBN 0-314-85240-9

Introduction to Computer Science

THIRD EDITION

Preface

This book is intended for a two-semester introductory course in computer science. No mathematical preparation is required beyond the usual elementary and high school courses. Although high school algebra is desirable, it is not necessary because the concepts that it might contribute—variables, functions, and expression evaluation—are developed in detail in this book. Most of the problems require no mathematical background. The occasional mathematically oriented problem, such as computing binomial coefficients, is intended only for students with sufficient mathematical background for the problem to be meaningful. The chapter on numerical methods requires somewhat greater mathematical sophistication than does the rest of the book.

The book has been completely rewritten for the third edition, with extensive changes in both content and presentation. Although I have not attempted to adhere rigidly to any one set of guidelines, the revision has been influenced by the course descriptions for CS1 and CS2 in the Association for Computing Machinery's Curriculum '78, the course description for the advanced placement test in computer science, currently available textbooks, and the recommendations of those who commented on the second edition. The following are some of the changes made in the third edition.

The first two chapters of the book present several algorithms for solving easily visualized problems such as sorting three names into alphabetical order and solving the Towers-of-Hanoi problem. Many instructors felt that the traditional Euclidean algorithm, with which the previous editions began, was too difficult for students without extensive mathematical backgrounds. These early algorithms are written in an Englishlike pseudolanguage to avoid all problems with the technicalities of an algorithmic language.

A new chapter on problems, problem-solving techniques, and their relation to the principles of algorithm construction has been included. The chapter on software design and testing has been expanded to discuss a number of concepts of modern software engineering, such as the software development life cycle, simulation of specifications, walkthroughs, correctness proofs, and program synthesis.

The use of assertions for understanding and verifying algorithms has been introduced from the outset and is referred to throughout the book. Although algorithm verification is discussed and illustrated, the emphasis is on assertions as an aid to understanding rather than as a tool for producing formal proofs of correctness.

Algorithms and programs are typeset in the boldface-italics form of Algol and Pascal rather than in the uppercase-letters-only form used in previous editions.

Data types, type definitions, and required variable declarations have been introduced into the algorithmic language. In the previous editions, types were treated informally and declarations were usually optional. This approach no longer seems appropriate in view of the importance of types and type checking in modern programming languages such as Pascal and Ada.

The discussion of parameter passing for procedures and functions has been completely revised. Parameters are classified as **in, out,** and **in out.** Parameter transmission by copy and by reference are both discussed. As in the second edition, the call-by-name mechanism is not discussed. Although call by name is of theoretical and historical interest, it plays no role in modern programming languages.

The discussion of global variable access—access to variables declared outside of a function or procedure—is expanded to include three possibilities: (1) **common** declarations as in FORTRAN and some other languages; (2) nested procedure declarations as in Algol and Pascal; and (3) modules encapsulating procedures, functions, and declarations, as in such languages as Modula-2, Ada, and UCSD Pascal. The third alternative is used throughout the remainder of the book to present many data structures as implementations of abstract data types. Emphasis is placed on the principle of information hiding as a guide to dividing a program into modules.

The discussions of data structures have been modified to emphasize depth rather than breadth; instead of trying to survey as many data structures and as many ways of representing a given structure as possible, fewer structures and representations are treated in greater detail.

The elements of algorithm analysis are introduced in the context of comparing the efficiency of different searching and sorting routines.

The two chapters on numerical methods have been combined into a single chapter, which provides sufficient coverage at the level of an introductory computer science course. Numerical integration is now illustrated by finding the area under a curve rather than by solving differential equations. Since it is assumed that students studying this chapter have a background in college algebra and trigonometry, standard mathematical concepts and notations are used more freely than in the other chapters. No previous knowledge of calculus is required; the chapter introduces the derivative as the slope of a tangent line and the definite integral as the area under a curve.

Part I introduces computers, information processing, computer applications, algorithms, and problem solving. Algorithms whose operation can be readily visualized

are presented in an Englishlike pseudolanguage. A sequential algorithm and an iterative algorithm are discussed in Chapter 1, and a recursive algorithm is discussed in Chapter 2. Since the recursive algorithm is somewhat more difficult to understand than the other two, some instructors may wish to postpone its discussion until Chapter 10, which also discusses recursion.

Part II covers computer hardware, software, and information representation. Instructors differ as to how deeply they wish to go into the technicalities of information representation. To make it as easy as possible to cover only the material desired, the topics in Chapter 4 are arranged in order of increasing difficulty. Section 4.1 and the discussion of binary addition and subtraction in Section 4.2 are elementary. The rest of Section 4.2 and Sections 4.3 and 4.4 are of intermediate difficulty. Sections 4.5 and 4.6 are somewhat more difficult still. Section 4.5 is not prerequisite to Section 4.6, so one can study the hypothetical computer without taking up floating-point-number representations.

Part III on algorithm construction is the heart of the book. Data types, constants, and variables are introduced, along with the fundamental manipulations of input, output, assignment, and expression evaluation. This part is organized around the basic control structures: sequencing, selection, repetition, and function and procedure invocation. Repetition is taken up before selection since much more interesting and realistic examples can be presented once repetition has been mastered. Part III ends with a chapter on program design and testing. This chapter reviews and systematizes the algorithm construction techniques that have been used so far and discusses the additional problems that arise in large programming projects.

In Part IV the emphasis switches from control structures to data structures. Type definitions and structured types are introduced, and records, files, arrays, strings, and linked structures are explored. Chapter 13 introduces arrays and describes searching and sorting of one-dimensional arrays. Since different searching and sorting algorithms can differ dramatically in efficiency, this chapter seems a good place to introduce the elements of algorithm analysis. Chapter 14, on stacks and queues, introduces these data structures not only for their own interest but as examples of abstract data types defined by means of the operations that can be carried out on their values.

Part V provides a brief introduction to numerical analysis. Included are discussions of roundoff and truncation errors, solving nonlinear equations, solving systems of linear equations, and numerical evaluation of definite integrals.

I wish to thank the following persons for their comments on the second edition and their suggestions for the third edition: Dian Lopez, University of Minnesota (Morris Campus); John Leeson, University of Central Florida; Jean Rogers, University of Oregon; and David Weldon, Winona State University.

Contents

Part I
Computers, algorithms, and problem solving

1

Computers, information, and algorithms

The idea of the computer can be traced back to the early nineteenth century, when the British mathematician Charles Babbage proposed a mechanical "Analytical Engine" to carry out mathematical calculations under the control of punched cards. The first electronic computers were built in the early 1940s, about a century after Babbage's original proposal.

For many years, the cost, size, and power consumption of computers restricted their use to large organizations, and it was difficult or impossible for individuals to gain access to them. In the mid 1970s, however, progress in microelectronics led to the development of small, low-cost, *microcomputers* or *personal computers*. The same developments gave rise to *embedded computers,* which can be incorporated in consumer products such as microwave ovens, television sets, cameras, automobiles, and video games. Today most people living in developed countries have probably used a computer, although in the case of embedded computers they may not be aware that they have done so.

The first computers were built to carry out the complex numerical calculations of science, engineering, and mathematics. People quickly realized, however, that computers are not limited to numbers, but are general-purpose machines for storing and manipulating information. In principle, any information-processing task can be carried out by a computer if we can code the information to be processed in symbols the computer can manipulate and if we can describe precisely the manipulations to be carried out. In practice, factors such as the memory (information storage) capacity of a computer, the speed with which it operates, and the means by which it com-

municates with the outside world determine whether a particular computer is suitable for a particular information-processing task.

A computer operates under the control of a set of detailed, step-by-step instructions called a *program*. Program control is responsible for the computer's enormous versatility: by changing the program we can drastically change the information-processing task the computer carries out. With one program a computer might control a robot; with another it might print workers' paychecks; with still another it might play a game with the user. The other side of this coin, however, is that we must write or purchase* a program for every job we want the computer to do for us. The programs, or *software,* for a computer system may cost far more than the computing machinery, or *hardware*.

Programs are to computers what phonograph records are to phonographs. The program determines the behavior the computer will exhibit, just as the record determines the sounds that the phonograph will produce. Changing the program changes the computer's behavior, just as changing the record makes the phonograph play a different tune. What's more, we tend to attribute the behavior exhibited to the program rather than to the computer, just as we attribute the music to the record rather than to the phonograph. For example, we might say that a particular program plays chess and that we enjoy a particular record, instead of making the more precise statements that the computer plays chess under the control of the program and that we enjoy the sounds produced by the phonograph when a particular record is played.

To summarize, no matter how much computers differ in size, cost, and internal construction, they all have the following two characteristics in common:

1. A computer is a general-purpose information processor.
2. A computer works under the control of a program—a set of detailed, step-by-step instructions for the processing to be done.

These two characteristics of computers determine the subject matter of computer science. Since computers process information, computer science studies *data structures*—means for representing information in a form suitable for computer processing. And since a computer works under the control of a detailed set of instructions, computer science studies *algorithms,* sets of instructions for carrying out particular information-processing tasks.

The discipline that we know as computer science in the United States is known in many other countries as *information science* or *informatics*. The well-known Dutch computer scientist E. W. Dijkstra advocates the term *computing science*. These alternate names emphasize that computer science is more concerned with the techniques of information processing than with the technical details of the machines that carry out that processing.

---

*Some programs may be permanently built into a computer, some may be included in the purchase price of the computer, and some may be available through *users' groups*—associations formed by persons who use a particular kind of computer. Mostly, however, computer users must write or buy the programs they need.