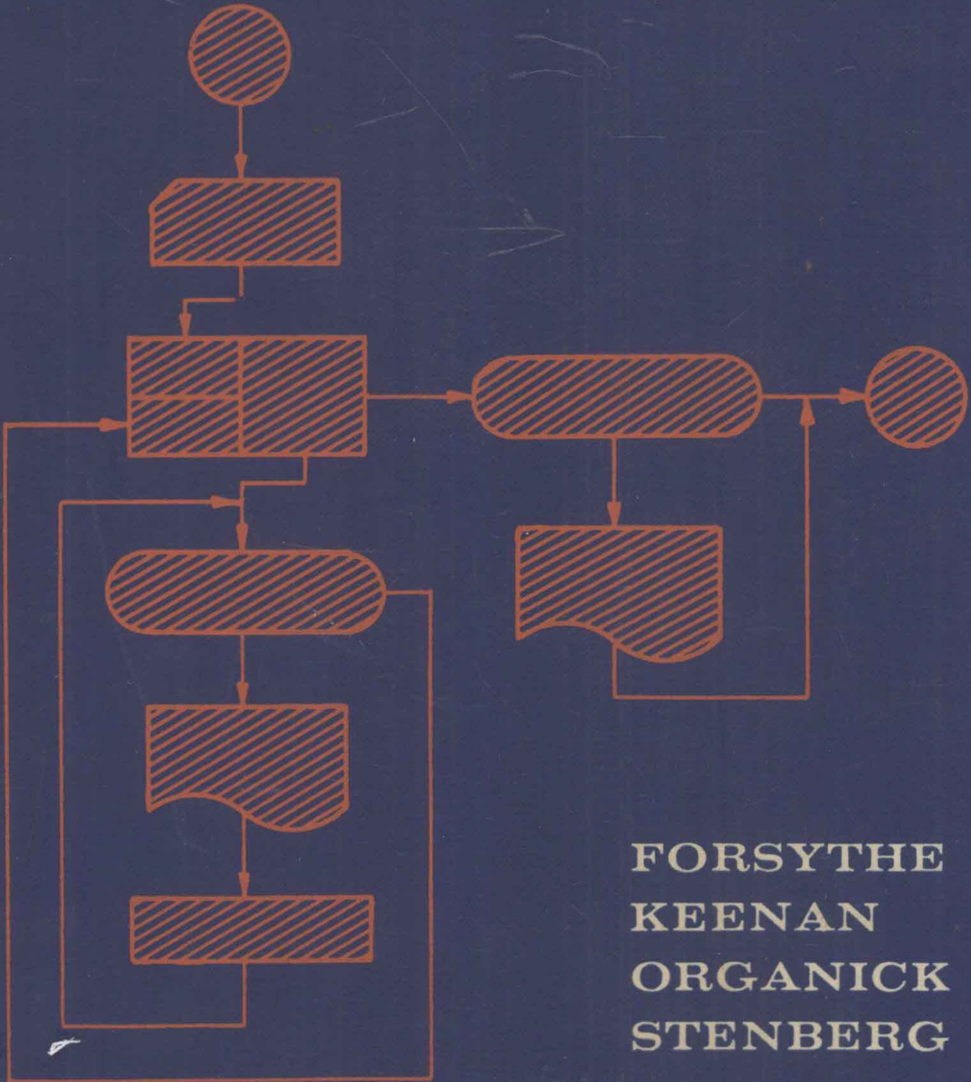
COMPUTER SCIENCE

A First Course



FORSYTHE
KEENAN
ORGANICK
STENBERG

ALEXANDRA I. FORSYTHE, M.A.

Department of Mathematics,
Gunn High School,
Palo Alto

THOMAS A. KEENAN, Ph.D.

Director, Educational Information Network,
EDUCOM

ELLIOTT I. ORGANICK, Ph.D.

Professor and Chairman, Department of Computer Science,
University of Houston

WARREN STENBERG, Ph.D.

Associate Professor of Mathematics,
University of Minnesota

JOHN WILEY & SONS, INC.

New York · London · Sydney · Toronto

10 9 8 7 6 5 4 3 2 1

Library of Congress Catalog Card Number 74-76053
SBN 471 26678 7
Printed in the United States of America

COMPUTER SCIENCE: A FIRST COURSE

COMPUTER SCIENCE: A First Course

PREFACE

Computer science is an entirely new course subject, although its concept has been developing for many years. Our purpose in this book, *Computer Science: A First Course,* is to help students understand today's world (and the world of tomorrow) where information of all kinds is the prime commodity. The processing of this information in business, science, government, and industry is rapidly becoming one of the nation's major endeavors. Computers are an indispensable tool in information processing. Students in this course will learn not only what computers are but what computers can and cannot do—they will learn to understand and appreciate the step-by-step methodical chain that begins with a problem, processes it through a computer, and ends with a satisfactory solution.

This book is the outgrowth of a School Mathematics Study Group (SMSG) program, begun in 1964, of which we were a part. This effort resulted in 1966 in a widely used experimental SMSG text called *Algorithms, Computation and Mathematics* which was directed to twelfth-grade mathematics students. From the beginning, we found that this material was equally instructive for college-level audiences. *Computer Science: A First Course* is based on the SMSG text and on our experience in teaching this text at high-school and college levels. Although the present volume draws significantly from the SMSG material and retains its spirit, we believe that this book and its supplementary texts comprise a major revision of the SMSG material with significant extensions in content. In short, we think that this work and its supplements are a distinct improvement worthy of independent use.

A recent report of the Curriculum Committee on Computer Science of the Association for Computing Machinery* has offered recommendations for a college-level course, entitled, "Introduction to Computing."

* "Curriculum 68, Recommendations for Academic Programs in Computer Science," *Communications of the ACM, Vol 11,* No. 3, March 1968, pp. 151–197.

We believe that this book is compatible with those recommendations and that it will be useful as a text for such a course.

Since a great deal of material is included here, it may be practical to study only portions of it. Another version of this textbook is available: *Computer Science: A Primer.* That version does not include Part III: "Nonnumerical Applications." Thus the primer may be more suitable for a short course or for one that concentrates exclusively on numerical computation.

As companion pieces to this volume and to the *Primer* version, several programming language supplements and a teacher's commentary are available. The programming language texts are especially useful because they are designed to dovetail, section for section, with the principal chapters of the basic text. The study of a computer programming language, such as FORTRAN, BASIC, or PL/1, from one of these supplements will help the student to convert the abstract algorithmic solutions of the problems from the basic text into actual solutions on the computer that is available to him.

The present volume centers around the study of comput*ing* rather than comput*ers*. Whereas many computer textbooks place significant emphasis on the design of complex networks of circuits and electronics that make up a computer, *Computer Science: A First Course* deals with the organization of problems so that computers can work them. Computing hinges primarily on the study of algorithms: not only learning to understand them but learning to construct and improve them.

Much thought has been given to the selection and ordering of the problems and exercises. The exercises are to be considered in the sense of five-finger piano exercises that test or strengthen some specific "local" learning. On the other hand, the problems require that the student *organize* local learning in order to reach a satisfactory solution. The solving of problems helps to synthesize knowledge into a more unified whole. The solution of at least some problems and exercises is considered vital to the progress of any student who uses this text.

The subject matter of this book constitutes a challenging first course in computer science for students with good high school-level preparation in mathematics.

Alexandra I. Forsythe
Thomas A. Keenan
Elliott I. Organick
Warren Stenberg

ORGANIZATION OF THE BOOK

For organizational purposes, the thirteen chapters of this book are divided into three parts. The first five chapters, which comprise Part I, form a basic introductory unit. Three fundamental ideas of computing are presented in the first chapter: the algorithm, its expression as a flow chart, and a conceptual model of a computer. In addition, the reader is introduced to a hypothetical but realistic digital computer capable of executing algorithms. By the end of this chapter, all of the fundamentals of flow-chart language have been introduced with appropriate discussions of assignment, branching, and looping. The next two chapters develop a more thorough explanation of the fundamentals and add some auxiliary concepts for *computation* and for *data organization* such as arithmetic expressions, compound conditions, multiple branching, vectors, and arrays. Chapter 4 integrates and refines the student's understanding of all these concepts by means of illustrative examples. An efficient shorthand for loop control, which simplifies the construction of many algorithms, is presented. Chapter 5, which does not depend on the specifics of the earlier chapters, alerts the student to pitfalls inherent in the use of approximations.

Part II is primarily concerned with numerical applications. In Chapter 6 the student is introduced to the *procedure,* the (isolated) program building-block unit from which complex systems can be formed. Procedures are explained in terms of a conceptual model that is easy to understand and to remember. Chapters 7 and 8 develop mathematical applications of computing that are selected from those called numerical methods. The solution of an equation by bisection is studied in Chapter 7, along with methods for finding the maximum or minimum of a function and for computing the area under a curve. The Gauss elimination method for solving a system of linear equations is given in Chapter 8, followed by an introductory treatment of averaging and linear regression. Part II is designed for students who have *not* studied calculus. However, students with calculus training may find that their understanding of mathematics and statistics has been strengthened after studying this material.

Part III is devoted to nonnumerical applications of computing (sometimes called symbol manipulation), representative of the newer areas of computer science research. Chapter 9 introduces the reader to an interesting representation of important classes of information: tree structures. Only the first four chapters of the book are prerequisite to this material. Certain decision processes (such as two-person games) and certain types of data (such as strings that represent arithmetic expressions) inherently possess or are best exhibited as tree structures. Tree-searching algorithms are introduced. Chapter 10 considers the subject of compiling, the process of translating from the familiar mathematical notation exhibited in various programming languages to forms that computers can execute more easily. Chapter 11 returns to the topic of trees. A level-by-level tree search method is applied to the problem of finding the best route or path on a map. Several new concepts for storing tree-structured data and for the pruning of such structures are introduced. Algorithms are developed that analyze the outcome of games. Chapter 12, on text editing and list processing, gives the student some insight into the kinds of problems involved in representing and operating on character "strings." For such data, an effective storage model is designed and algorithms are built up to transform strings in various ways. This chapter provides an excellent background for a student who will use one of the string-processing programming languages, such as SNOBOL. Chapter 13 takes another look at compiling, this time as an application of the string-operation concepts developed in Chapter 12. Combining this material with the material in Chapter 10, the student can follow the principal steps of compiling from the input (consisting of statements like those in a FORTRAN program) to the output (consisting of the equivalent machine language code).

The SAMOS Appendix amounts to an elementary programmer's reference manual for the hypothetical digital computer called SAMOS that was discussed briefly in Chapter 1. This appendix is suggested for collateral reading at various times in the course. SAMOS has been simulated on several actual computers,* making it possible for students to gain an easy initial exposure to machine-language programming through laboratory practice.

* Simulated on the IBM 1401 and 7090 computers and on the CDC 6400. (The simulator program written in FORTRAN was developed by and has been made available from the Florida State University Computer Center, E. P. Miles, Director.)

ACKNOWLEDGMENTS

Much of the credit for this book belongs to our many colleagues who participated in the organizational, planning, and computer text-writing sessions of the SMSG project and to the excellent supporting staff of SMSG at Stanford University. Below is a copy of the title page (and copyright) from the student text, which gives the names of the project's many contributors. To each of our colleagues on this list and to others who have offered contributions to this project, we express our sincere appreciation and the hope that our new book confirms the value of the initiating work.

ALGORITHMS,
COMPUTATION
AND
MATHEMATICS

Student Text

Revised Edition

The following is a list of all those who participated in the preparation of this volume:

Sylvia Charp, Dobbins Technical High School, Philadelphia, Pennsylvania
Alexandra Forsythe, Gunn High School, Palo Alto, California
Bernard A. Galler, University of Michigan, Ann Arbor, Michigan
John G. Herriot, Stanford University, California
Walter Hoffmann, Wayne State University, Detroit, Michigan
Thomas E. Hull, University of Toronto, Toronto, Ontario, Canada
Thomas A. Keenan, University of Rochester, Rochester, New York
Robert E. Monroe, Wayne State University, Detroit, Michigan
Silvio O. Navarro, University of Kentucky, Lexington, Kentucky
Elliott I. Organick, University of Houston, Houston, Texas
Jesse Peckenham, Oakland Unified School District, Oakland, California
George A. Robinson, Argonne National Laboratory, Argonne, Illinois
Phillip M. Sherman, Bell Telephone Laboratories, Murray Hill, New Jersey
Robert E. Smith, Control Data Corporation, St. Paul, Minnesota
Warren Stenberg, University of Minnesota, Minneapolis, Minnesota
Harley Tillitt, U.S. Naval Ordnance Test Station, China Lake, California
Lyneve Waldrop, Newton South High School, Newton, Massachusetts

The following were the principal consultants:

George E. Forsythe, Stanford University, California
Bernard A. Galler, University of Michigan, Ann Arbor, Michigan
Wallace Givens, Argonne National Laboratory, Argonne, Illinois

Like many texts that originate as a committee effort, the early drafts were revised a number of times. Two parts of the present work have passed the scrutiny of these repeated revisions without losing the character or content contributed by their originators. Chapter 5, "Approximations," still bears the distinctive imprint of Professor Walter Hoffman of Wayne State University who wrote the first draft. The SAMOS Appendix, which appears here in much the same form as in its first draft, is the work of the late Silvio O. Navarro, Professor of Computer Science at the University of Kentucky, who is fondly remembered by his friends and colleagues.

We also acknowledge the support of individuals who either recommended that the writing project be initiated or who played a part in organizing it. Among these were:

Dr. Arthur C. Downing, Control Data Corp.
W. Eugene Ferguson, Newton Mass. School System
Professor Robert Gregory, The University of Texas
George Heller, International Business Machines Corp.
Professor R. J. Walker, Cornell University

Four men were the "prime movers" of the SMSG project: Professor Edward G. Begle, SMSG's Director convinced the SMSG Executive Board and the National Science Foundation that the computer text project was a worthy undertaking and deserved the financial support of NSF. Professor George E. Forsythe of Stanford University, Professor Bernard A. Galler of the University of Michigan, and Dr. Wallace Givens of Argonne National Laboratories not only helped Professor Begle in his initiating efforts but provided the project with critical technical and organizational guidance during its initial stages.

Finally, we thank the National Science Foundation for its continued support of SMSG projects in the computer science area and Stanford University for giving us approval to incorporate the cited SMSG material in our text. SMSG, by giving this approval, does not endorse the current work in any way.

A.I.F.
T.A.K.
E.I.O.
W.S.

USING THE COMPUTER

For optimal learning, each beginner needs contact with a computer, primarily to verify and troubleshoot the algorithms that he constructs. What is meant by "contact" with a computer varies. One excellent form of contact is gained these days through the use of typewriter-like or keyboard/TV consoles that are connected to time-shared computer systems. Another form of desirable contact can be obtained by submitting programs and data that are prepared on punch cards and run on computers controlled by batch monitor operating systems. In any case, the computer contact should be held to the minimum that is necessary for adequate check-out of programs. Hands-on-the-computer experience is *not* a goal of this course.

At least as important as one's physical proximity to the computer should be the *programming language* that it requires and the special program that implements this language on the computer. Since the number of people using it is *not* necessarily a measure of how good a programming language is, the choice of a language, a computer, and a software system requires very careful professional study. ("Software" refers to the service programs used to operate and exploit the computer for the user's benefit.)

The software program that carries a language into effect on a computer is called a *compiler.* Compilers vary a great deal in the *rate* at which they cause programs to be translated or interpreted by the computer. A fast compiler is wanted. Compilers also vary widely in their handling of programming and language errors. For teaching purposes, the best compiler is one that detects, clearly identifies, and either reports or corrects these errors at the time the program is being *compiled.*

Some types of programming errors (such as dividing by zero) can only be detected and reported during the execution of the target (compiled) program. The quality of this detection is dependent on still another piece of software called the *executive system.* Furthermore, whether this executive system detects such problems as equipment malfunction during input or output and misuse of library subroutines will be very important to you, as the user.

The ability to run large numbers of small programming exercises in a reasonable amount of time and with satisfactory results is, as one can see, highly dependent on the software system. It is a disappointing but true fact that any particular programming language, such as FORTRAN, can still vary considerably in its implementation on different computers. It is even possible that the FORTRAN used on two copies of the same machine may differ materially in its ability to provide the desired service.

Many high schools and a great many colleges (also almost all of the large business or governmental institutions) already have computers that use a batch monitor type of operating system. For those who have no such facility of their own (and, sometimes, even for those who do), an arrangement with a nearby university or other institution for the use of their computer laboratory is urged. A computer 100 miles away and well equipped with software adapted to educational use may be better for the purpose than a small computer in the next room. A smooth educational software system on a machine, even if it is remote, allows students to focus on the construction of algorithms in a programming language. With this knowledge comes more rapid insight into the uses of computers in science and industry. Arrangements for remote use of time-shared computer systems (especially if one is favored with good-quality telephone service) via rented teletype or other keyboard consoles can be very effective.

USING A PROGRAMMING LANGUAGE

To increase the applicability of this book, the specific syntactic details of computer language have been separated from the main flow-chart text into a *language supplement*. The flow-chart language used in the main textbook deals only with concepts of central interest to *all* programming languages. Although embodying a set of general syntactic concepts, flow-chart language contains few such details. Having learned one basic programming language, it is easy to learn another. After the flow-chart language in this book has been studied, PL/1, BASIC, FORTRAN, COBOL, ALGOL, or any similar algebraic procedural language can be learned with ease as a second language. This organization enables a school to choose a programming language from among any of those just mentioned (or their equivalent) and still emphasize the fundamental concepts behind most computer usage. The great reward to the student from this separation of main concepts from syntactic details is the universal applicability of flow-chart language, which he learns first.

Those who read Chapter 12 (on strings) will find themselves well prepared for the string-oriented procedural languages like SNOBOL, and for special string-manipulating features of some of the other languages. For anyone studying Part III of this text in depth, a second programming language like SNOBOL is recommended in addition to (and complementary to) others such as PL/1, BASIC, or FORTRAN.

A brief comment on how the language supplements are organized will be helpful. We recommend that Chapter 1 of the language supplement should be studied only after completing the reading of Section 1-4 of Chapter 1 in the main textbook. The reason for this is to introduce the student to elements of the flow-chart language *before* he meets the programming language equivalent. In this way, a language like FORTRAN, BASIC, or PL/1 is already a *second* language.

After reading Chapter 1 of the supplement, small computer programs can be written as laboratory exercises. The instructor can be expected to supply *particularized information* to close the gap between the language supplement and *your* computer facility. Often, this type of special information is also available from local computer personnel. General-reference manuals are detailed technical publications that are seldom appropriate as an introduction and hardly ever specific to a facility. For instance, some computer facilities require that problems be submitted on special coding sheets. At other facilities, it may be necessary to key-punch cards or paper tape. If cards are used, there are nearly always particular details that differ among facilities regarding the preparation of card decks, including, for example, identification cards or job control cards. If typewriter-like terminals are used, each has its own method of operation and such details can only be supplied locally.

Each chapter of the supplement adds more language capability. Beginning with Chapter 2, any corresponding chapter can be read, section by section, along with the main textbook.

Some final remarks are in order concerning the handling of *input-output details*. To write a program in some languages, especially FORTRAN, one must learn certain data format details, but there is a risk of spending too much time learning these particulars at the expense of developing problem-solving experience. To avoid this, the FORTRAN supplement offers format details piecemeal, as needed, beginning with Chapter 1. The complete subject of format can be quite complex and, here again, one can profit from experienced assistance. Consult highly technical reference manuals only as a last resort.

Format in a language like PL/1 (or SNOBOL) need not be studied initially (or at all) because a set of "simplified" input-output statements is available. Thus the need to learn format codes and associated details is lessened and may be avoided entirely. Some FORTRAN implementations have available a very simple, easily learned I/O scheme involving a minimum of format control or none whatsoever.

COMPUTER SCIENCE: A FIRST COURSE

CONTENTS