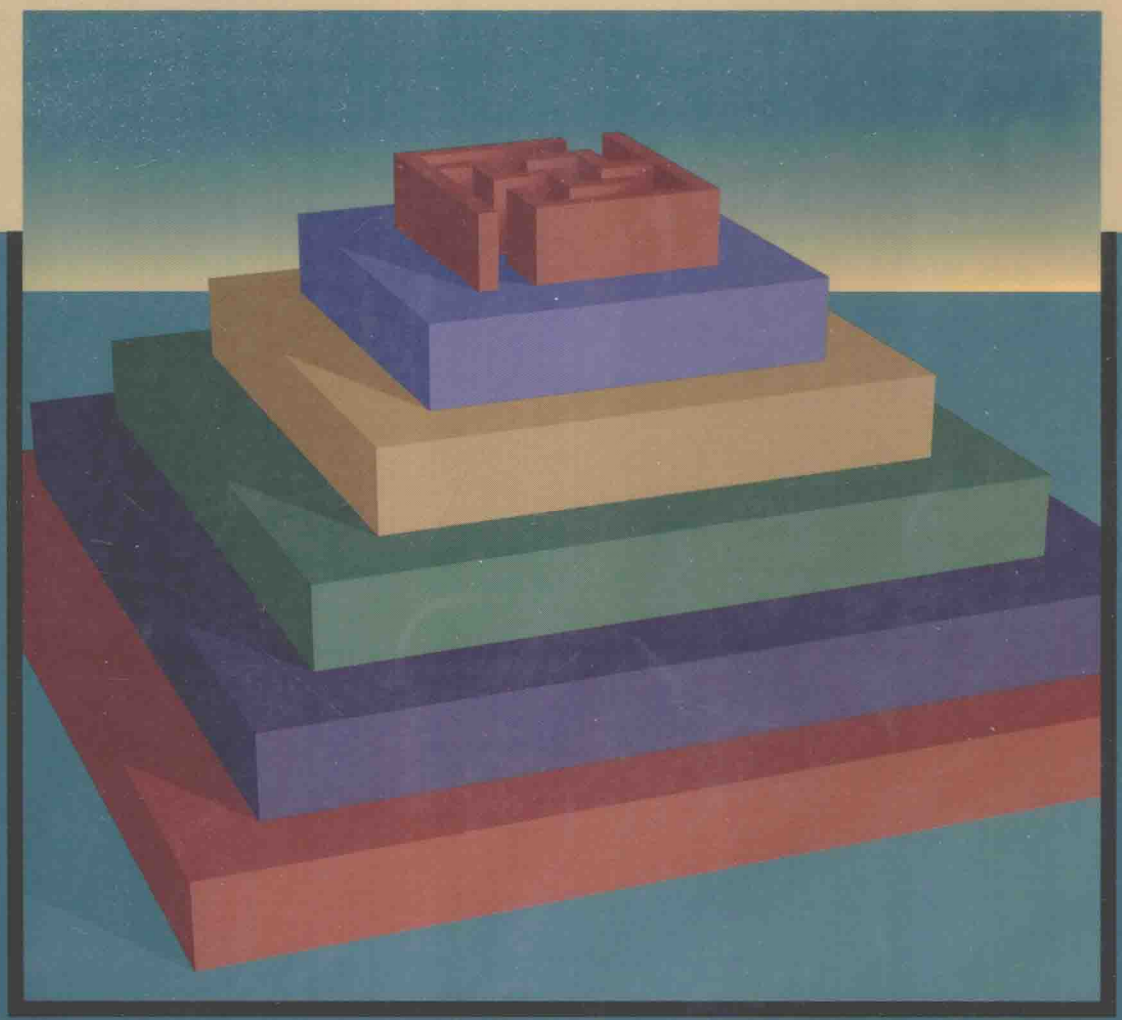
AN INVITATION TO COMPUTER SCIENCE



G. Michael Schneider • Judith L. Gersting

AN INVITATION TO COMPUTER SCIENCE

G. Michael Schneider
Macalester College

Judith L. Gersting
University of Hawaii–Hilo

Contributing author:
Sara Baase
San Diego State University

WEST PUBLISHING COMPANY
Minneapolis/St. Paul New York Los Angeles San Francisco

WEST'S COMMITMENT TO THE ENVIRONMENT

In 1906, West Publishing Company began recycling materials left over from the production of books. This began a tradition of efficient and responsible use of resources. Today, up to 95 percent of our legal books and 70 percent of our college and school texts are printed on recycled, acid-free stock. West also recycles nearly 22 million pounds of scrap paper annually—the equivalent of 181,717 trees. Since the 1960s, West has devised ways to capture and recycle waste inks, solvents, oils, and vapors created in the printing process. We also recycle plastics of all kinds, wood, glass, corrugated cardboard, and batteries, and have eliminated the use of Styrofoam book packaging. We at West are proud of the longevity and the scope of our commitment to the environment.

Production: Michael Bass & Associates
Copyediting: Linda Purrington
Text design: Linda M. Robertson
Cover design: R. Kharibian & Associates
Composition: American Composition & Graphics, Inc.
Illustration: Asterisk Group

Prepress, printing, and binding by West Publishing Company

British Library Cataloguing-in-Publication Data. A catalogue record for this book is available from the British Library.

Printed in the United States of America

02 01 00 99 98 97 96 8 7 6 5 4 3 2 1

Library of Congress Cataloging-in-Publication Data

Schneider, G. Michael.
An invitation to computer science / G. Michael Schneider, Judith L. Gersting.
p. cm.
Includes index,
ISBN 0-314-04375-6
1. Computer science, I. Gersting, Judith L. II. Title.
QA76.S3594 1995
004—dc20 94-39730
CIP

AN INVITATION TO COMPUTER SCIENCE

Preface

This is a text for a one-semester introductory course in computer science. It assumes no prior background or experience, and it is appropriate for use by either nonmajors or majors who want a broad overview of the field.

Introductory computer science service courses for nonmajors have undergone a number of changes in the last few years. In the 1970s and early 1980s, they were usually programming courses in FORTRAN, Pascal, or BASIC. At that time, it was felt that programming in a high-level language was the most important computing skill that students (usually science or engineering) could acquire. In the mid- and late-1980s the rapid growth in the use of computing caused the course to evolve into something called "computer literacy," where students learned about new applications of computers in such fields as business, medicine, law, education, and the arts. Finally, with the increased availability of personal computers and useful software packages, a typical early-1990s version of the computer science service course spends a semester teaching students how to use word processors, databases, spreadsheets, bulletin boards, and electronic mail.

Many people feel that the time is right for the introductory course in computer science to undergo yet another change. There are two reasons for this. First, many students coming to college today are quite familiar with personal computers and software packages. They have been writing with word processors since high school and have been using networks, e-mail, and bulletin boards for years. (In fact, it is not uncommon for students to know much more about using computer networks than faculty members.) A course that teaches how to use software packages will be of little interest. Second, a course that concentrates on only one aspect of computer science, whether it is programming, applications of computers, or software packages, can give students a highly misleading view of the discipline. It is not unusual for students completing a computer literacy course to view computer science as simply the study of programming or software packages, certainly a highly incorrect perception.

The feeling of many now is that the first course should be a breadth-first overview that introduces students to a wide range of topics in computer science. The material covered in this course could include such important and interesting topics as algorithms, hardware design, computer organization, system software, language models, programming, compilation, theory of computation, artificial intelligence, and social issues of computing. Students

would be introduced to the richness of ideas and problems addressed by professionals in computer science. A breadth-first approach would also bring us into line with most other scientific disciplines with respect to their survey course for nonmajors. For example, a chemistry service course introduces fundamental concepts (atoms, molecules, reactions) in addition to the uses and applications of chemistry. Similarly, a beginning physics course for nonmajors spends much of its time on such important theoretical concepts as elementary particles, force, matter, and energy.

That is exactly how this book is organized. It is a one-semester, breadth-first introduction to the discipline of computer science. It assumes absolutely no background in either computer science, programming, or mathematics. It is appropriate for use as a text for a service course for students not majoring in computer science. It would also be fully appropriate for use at schools where the first course for majors is an overview of the discipline rather than a programming course in Pascal, C/C++, or Scheme.

The text introduces a wide range of subject matter. However, it is not enough to simply present a mass of material, a wealth of facts and details. The discussion must be woven into some fabric, an organized theme that can unite the many topics covered. The book must create a "big picture" of computer science. Our big picture is to present the discipline of computer science as a six-layer hierarchy of abstractions, with each layer in the hierarchy building on ideas and concepts presented earlier. Just as the chemist builds from protons and electrons to molecules and then to compounds, so too will this text build from such elementary concepts as gates and circuits to higher-level ideas such as computer systems, virtual machines, languages, applications, and social, legal, and ethical problems of technology.

The six levels in our hierarchy are diagrammed in Figure 1.4 and listed here:

Level 1. The Algorithmic Foundations of Computer Science

Level 2. The Hardware World

Level 3. The Virtual Machine

Level 4. The Software World

Level 5. Applications

Level 6. Social Issues

Level 1 (Chapters 2 and 3) introduces the algorithmic foundations of computer science, the bedrock on which all other aspects of the discipline are built. It presents such important ideas as the design of algorithms, algorithmic problem solving, abstraction, pseudocode, iteration, and efficiency. It illustrates these ideas using such well-known examples as searching a list, finding the largest element, sorting a list, and pattern matching.

The discussion in Level 1 assumes that our algorithms are executed by something called a "computing agent," an abstract concept for anything that can carry out the instructions in our solution. Now, in Level 2 (Chapters 4 and 5) we say that we would like these algorithms to be executed by "real" computers to produce "real" results. This begins our discussion of hardware and computer organization. Chapter 4 presents the basic building blocks of computer systems—binary numbers, Boolean logic, gates, and circuits. Chapter 5 then shows how these elementary concepts are used to construct a real computer,

and it introduces the Von Neumann model of computing. It also presents a typical machine language instruction set and discusses how algorithms from Level 1 can be represented in machine language and run on the hardware of Level 2. It ends with a discussion of new directions in hardware design—massively parallel processors.

By the end of Level 2, students have been introduced to the basic concepts of logic design and computer organization, and they can appreciate the complexity of these subjects. This complexity is the motivation for Level 3 (Chapter 6), the virtual machine environment. This section describes how system software produces a user-oriented problem-solving environment that hides many of the hardware details discussed earlier. It presents the same problem discussed in Level 2, that of encoding an algorithm and running it, and shows how easy that is to do in a virtual environment that contains software tools such as text editors, assemblers, loaders, and an operating system. Level 3 also discusses the services and responsibilities of operating systems and introduces the different types of systems that can be created, such as real-time systems, embedded computers, time sharing, local-area networks, and distributed systems.

Now that we have a supportive problem-solving environment, what do we want to do? Most likely we want to write programs to solve problems. This becomes the motivation for Level 4 (Chapters 7–10), the world of software. Although the book should not be seen as a programming text, Chapter 7 contains an introduction to Pascal and some of its important concepts—variables, data types, assignment, conditional, iteration, and procedures. This will give students an appreciation for the task of the programmer and the power of the problem-solving environment created by a modern high-level programming language. However, we also want students to know that there are many other high-level language models. Chapter 8 shows a simple algorithm encoded in three other procedural languages, and also presents an overview of the functional, logic, and object-oriented language paradigms. Chapter 9 describes the design and construction of a compiler and shows how the languages of Chapters 7 and 8 must be translated into machine language for execution. This material ties together many ideas that have come before as it shows how an algorithm (Level 1) is coded into a high-level language and compiled (Level 4) into machine language and executed on a typical Von Neumann machine (Level 2) using the system software tools of Level 3. These frequent references to earlier concepts help to reinforce those ideas, and the use of "recurring themes" is a common teaching method in this text. Finally, in the last chapter of Level 4, we introduce the idea of computability and unsolvability. It describes a model of computing (a Turing machine) and uses that model to show that there are problems for which no general algorithmic solution can be found.

We now have a supportive software environment in which it is possible to write programs to solve important problems. The question now is, what problems should we solve? What are some of the most important applications of computers? In Level 5 (Chapters 11–12) we present a few important uses of computers, including spreadsheets and modeling, databases, numeric and symbolic computing, networks, electronic mail, and artificial intelligence. There is no way in two chapters to cover all the important applications of information technology, or to cover specific applications in minute detail. Our

goal is to show students that application software packages are not "magic boxes" but the result of the intelligent use of the computer science concepts developed in earlier chapters. We hope that this introduction to a few key applications will encourage readers to seek out information on applications that are specific to their own interests.

Finally, we reach the highest level of study, Level 6 (Chapter 13), which addresses social, ethical, legal, and professional issues raised by the applications discussed in Level 5. This section (written by contributing author Sara Baase) talks about such thorny problems as privacy concerns aggravated by the growth of on-line databases and security problems caused by the use of networks and telecommunication. This section introduces students to important social issues and makes them aware of the enormous impact that computer science and computer technology is having on society.

This, then, is the hierarchical structure of the text. It begins with the algorithmic foundations of the discipline, and works its way upward from low-level hardware concepts through virtual machine environments, languages, software, and applications programs, to the social issues raised by computer technology. This organizational tool is one of the most important aspects of the book, as it allows us to present computer science as a unified, integrated, and coherent discipline of study.

Because of the structure of this text as a hierarchy of ideas, a sequential progression through the chapters is most suitable. However, material from Level 6 on social issues can be introduced throughout the course as appropriate. Also, small sections along the way can be omitted if desired, for example, 2.3.3 and 3.5.4 (the pattern matching algorithm), 4.4.3.2 (the full-adder circuit), 7.7 (software engineering), 8.3.4 (parallel programming), or 9.2.4 (code optimization). The Instructor's Manual associated with this text provides suggestions on how to organize and present the material based on the time and the resources available for the course.

Another important development in the field is the realization that, like physics, chemistry, or biology, computer science is an empirical, laboratory-based discipline in which learning comes not only from listening but also from doing and trying. Many complex ideas in computer science cannot be truly understood until they are visualized, observed, and manipulated. Today, most computer science faculty view a laboratory component as an essential part of every introductory course. This important development is fully reflected in our approach to teaching computer science. Associated with the text is a laboratory manual and custom-designed laboratory software that allows students to experiment with theoretical concepts presented in this text.

The manual contains 19 separate laboratory experiences that build on and amplify the ideas presented. Each laboratory assignment includes software that visualizes an important concept and gives students the chance to observe, study, analyze, and modify. For example, associated with Level 1 (the algorithmic foundations of computer science) are labs that animate the algorithms presented in Chapters 2 and 3, and that analyze the running time of these algorithms for different-sized data sets. Associated with Level 2 (the hardware world) are labs that allow students to design and analyze logic circuits and program a simulated Von Neumann machine identical to one presented in the text. Similarly there are projects for virtually all the concepts discussed in the text. Each of the 19 laboratories includes an explanation of how to use the

software, a description of how to conduct the experiment, and discussion questions and problems for students to complete. Students should be able to work on their own or in collaborative teams to run these lab experiments, in either a closed lab or open lab setting. The exercises in lab experience 12 (12.12–12.17) involve Pascal programming for which the students may need a bit more help. Lab experience 1 serves as an introduction to the software suite, and also provides the students with a useful glossary-building tool that they can use with the text (and with other courses as well).

Both PC and Macintosh versions of the software are available. Most of the lab experiences require only the software specifically designed to accompany the text. A few, however, depend upon student access to additional software. Lab 13 requires Scheme or some other LISP-based interpreter. Lab 17 assumes that the students are using Microsoft Excel or some other spreadsheet package. Lab 18 is based upon Mathematica, and Lab 19 assumes that students have access to electronic mail and the Internet.

In addition to the laboratory software, there are a number of other teaching aids available to support the text material, including transparency masters, videotapes, a test bank, and an instructor's manual containing chapter outlines, answers to exercises, and suggestions for course organization.

Computer science is a young and exciting discipline, and we hope that the material in this text, along with the laboratory projects, will convey this feeling of newness and excitement. By presenting the field in all of its richness—algorithms, hardware, software, applications, social issues—we hope to give students a deeper appreciation for the many diverse and interesting areas of research and study within the discipline of computer science.

G. Michael Schneider
Judith L. Gersting

Acknowledgements

We would like to thank the many people who contributed greatly to the successful completion of this project. First, a huge thank you to Ken Lambert and Tom Whaley for their cooperative spirit and outstanding work on the laboratory manual and the associated software package. Many thanks to Jerry Westby, our editor at West Publishing, for "hanging in there" through the painstaking cycles of revisions and rewrites. Many other individuals at West were also helpful during the completion of this project, most notably Dean De Chambeau and Stephanie Buss. We would like to acknowledge the contributions of all the referees who read the early drafts of the text and made so many helpful suggestions and comments. Finally, we wish to thank our spouses, Ruthann and John, for always being supportive of our work.

Sara Baase thanks the following people who assisted in the preparation of Chapter 13 by providing leads and information, answering questions, and/or reading drafts:

Robert C. Balling, Jr. (Director of the Office of Climatology, Arizona State University), Tim Barnett (Scripps Institution of Oceanography, University of California, San Diego), Leland Beck (Computer Science Division, San Diego State University), Jim Bennett (president, Center for Constitutional Issues in Technology), John L. Carroll (Computer Science Division, San Diego State University), Michael B. Cline (Azon, Keuffel & Esser), Jeffrey W. DeMarre (Seattle-Tacoma International Airport), Williamson Evers (Hoover Institution), Chuck Charman (General Atomics), Chuck Cooper (Biology Department, San Diego State University), Pat Feuerstein, Bence Gerber (Livermore Software Technology Corporation), Mike Godwin (staff counsel, Electronic Frontier Foundation), James L. Hoover (Law School Library, Columbia University), the staff of the Interlibrary Loan Department at San Diego State University, Jim Johnson (General Motors Research Labs), Tawfik Khalil (General Motors Research Labs), Darrell Long (Computer Science Department, University of California, Santa Cruz), Bernard Mandanca (National Oceanic and Atmospheric Administration), John F. B. Mitchell (United Kingdom Meteorological Office), Steve Napear (San Diego Supercomputer Center), Peter G. Neumann (SRI International), Jack Resnick, MD (Radiology Department, Sharp Rees-Stealy Medical Group), Jack Revelle, Carol Sanders, Jack Sanders (attorney), Bernard Siegan (University of San Diego School of Law), Robert Ellis Smith (publisher, *Privacy Journal*), Shari Steele (staff attorney, Electronic

Frontier Foundation), Jacob Sullum (associate editor, *Reason* magazine), Vernor Vinge (Computer Science Division, San Diego State University), Baja Bob Vinton, Steve Wampler (Lawrence Livermore National Laboratory), Jim Warren (AutoDesk, Inc.), Robert Whirley (Lawrence Livermore National Laboratory), and any others I may have omitted.

We would also like to thank the following reviewers for their valuable comments:

Ernest C. Ackermann
Mary Washington College

Elizabeth S. Adams
Hood College

Virginia T. Anderson
University of North Dakota

William N. Anderson, Jr.
Fairleigh Dickinson University

James D. Arthur
Virginia Polytechnic Institute

Douglas Baldwin
SUNY-Geneseo

Adrienne G. Bloss
Roanoke College

Anselm Blumer
Tufts University

Kim B. Bruce
Williams College

Jim Carter
University of Saskatchewan

Lillian N. Cassel
Villanova University

Darrah Chavey
Beloit College

John Cigas
Rockhurst College

David Cordes
University of Alabama

Lee D. Cornell
Mankato State University

Michelle Wahl Craig
University of Toronto

Grace Anne Crowder
Towson State University

Fadi Pierre Deek
New Jersey Institute of Technology

Herbert L. Dershem
Hope College

Maurice L. Eggen
Trinity University

Henry A. Etlinger
Rochester Institute of Technology

Daniel J. Falabella
Albright College

John E. Howland
Trinity University

Mary Kolesar
Utah State University

Ken Lambert
Washington & Lee University

Rickard A. Lejk
University of North Carolina

Jimmie M. Purser
Millsaps College

Samuel A. Rebelsky
Dartmouth College

Jane M. Ritter
University of Oregon

Larry F. Sells
Oklahoma City University

Cliff Shaffer
Virginia Polytechnic Institute

Angela B. Shiflet
Wofford College

Ted Sjoerdsma
Washington & Lee University

Jeff Slomka
Southwest Texas State University

Gordon A. Stegink
Hope College

Paul Stephan
Case Western Reserve University

Bill Taffe
Plymouth State College

Robert J. Wernick
San Francisco State University

Tom Whaley
Washington & Lee University

Craig E. Wills
Worcester Polytechnic Institute

Carol W. Wilson
Western Kentucky University

Finally, we wish to acknowledge the use of the following trademarks in the text: Macintosh, Hypercard (trademarks of the Apple Computer Corp.), T3D and X-MP (Cray), CM-5 (Thinking Machines Co.), VAX 4000-400, VAX-VMS (Digital Equipment Corp.), MS-DOS, Windows, Windows NT, Visual BASIC, Visual C++, Excel (Microsoft), UNIX (AT&T Corp.), Toolbook (Asymmetrix Corp.), and Mathematica (Wolfram Associates).

Brief Contents

Contents