大学计算机教育国外著名教材系列（影印版）

PEARSON
Addison Wesley

COMPUTER SCIENCE
AN OVERVIEW
SEVENTH EDITION

计算机科学导论
（第7版）

J. Glenn Brookshear 著

Pearson Education

清华大学出版社

大学计算机教育国外著名教材系列（影印版）

Computer Science: An Overview

Seventh Edition

计算机科学导论

（第 7 版）

J. Glenn Brookshear

清 华 大 学 出 版 社

北　京

图书在版编目(CIP)数据

计算机科学导论(第 7 版)= Computer Science: An Overview, Seventh Edition / 布鲁克希尔(Brookshear J.G.)著. —影印本. —北京：清华大学出版社，2003
（大学计算机教育国外著名教材系列）
ISBN 7-302-07792-4

Ⅰ. 计…　Ⅱ. 布…　Ⅲ. 计算机科学－高等学校－教材－英文　Ⅳ. TP3

中国版本图书馆 CIP 数据核字（2003）第 116053 号

出 版 者：	清华大学出版社	**地　址**：	北京清华大学学研大厦
	http://www.tup.com.cn	**邮　编**：	100084
	社总机：(010) 6277 0175	**客户服务**：	(010) 6277 6969

责任编辑：龙啟铭
印 刷 者：清华大学印刷厂
装 订 者：三河市新茂装订有限公司
发 行 者：新华书店总店北京发行所
开　本：185×230　**印张**：37
版　次：2004 年 1 月第 1 版　2004 年 1 月第 1 次印刷
书　号：ISBN 7-302-07792-4/TP・5678
印　数：1~5000
定　价：54.00 元

出 版 说 明

进入 21 世纪，世界各国的经济、科技以及综合国力的竞争将更加激烈。竞争的中心无疑是对人才的竞争。谁拥有大量高素质的人才，谁就能在竞争中取得优势。高等教育，作为培养高素质人才的事业，必然受到高度重视。目前我国高等教育的教材更新较慢，为了加快教材的更新频率，教育部正在大力促进我国高校采用国外原版教材。

清华大学出版社从 1996 年开始，与国外著名出版公司合作，影印出版了“大学计算机教育丛书（影印版）”等一系列引进图书，受到了国内读者的欢迎和支持。跨入 21 世纪，我们本着为我国高等教育教材建设服务的初衷，在已有的基础上，进一步扩大选题内容，改变图书开本尺寸，一如既往地请有关专家挑选适用于我国高校本科及研究生计算机教育的国外经典教材或著名教材，组成本套“大学计算机教育国外著名教材系列（影印版）”，以飨读者。深切期盼读者及时将使用本系列教材的效果和意见反馈给我们。更希望国内专家、教授积极向我们推荐国外计算机教育的优秀教材，以利我们把“大学计算机教育国外著名教材系列（影印版）”做得更好，更适合高校师生的需要。

清华大学出版社
2002.10

Preface

This book provides an introduction to the science of computing. It surveys the breadth of the subject while including enough depth to convey an honest appreciation for the topics involved.

Audience

I wrote this text for both computer science majors and students from other disciplines. As for computer science majors, most begin their studies with the illusion that computer science is programming and Web browsing since that is essentially all they have seen. Yet computer science is much more than this. In turn, beginning computer science students need exposure 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 appreciate the relevance and interrelationships of future courses in the field.

This same background is what students from other disciplines need if they are to relate to the technical society in which they live. A computer science course for nonmajors should provide a fundamental understanding of the entire field rather than merely an introduction to popular software packages. This survey approach is the model used for introductory courses in the natural sciences, and it is the model I had in mind as I wrote this text. Accessibility for nonmajors was one of my major goals. The result is that previous editions of this book have been used successfully in courses for students over a wide range of disciplines. This edition is designed to continue that tradition.

Organization

This text follows a bottom-up approach that progresses from the concrete to the abstract—an order that results in a sound pedagogical presentation in which each topic leads to the next. It begins with the fundamentals of computer architecture (Part 1), progresses to software and the software development process (Part 2), explores issues of data organization and data storage (Part 3), and closes by considering current and future applications of computer technology (Part 4).

While writing the text, I actually thought in terms of developing a plot. Consequently, I am not surprised that many students have reported reading the text in much the same way that they normally read novels. On the other hand, the text is divided into largely independent chapters and sections that can be read as isolated units (see Figure 0.7 in Chapter 0) or rearranged to form alternative sequences of study. Indeed, the book is often used as a text for courses that cover the material in different orders. The most common of these alternatives begins with material from Chapters 4 and 5 (Algorithms and Programming Languages) and returns to the earlier chapters as desired. In contrast, I know of one case that starts with the material on computability from Chapter 11. (In still other instances the text has been used in "senior capstone" courses where it serves as a backbone from which students branch into projects in different areas.) I suggest the following sequence for those who simply want a condensed version of the novel:

Section	Topic
1.1-1.4	Basics of data encoding and storage
2.1-2.3	Machine architecture and machine language
3.1-3.3, 3.5, 3.7	Operating systems and networking
4.1-4.4	Algorithms and algorithm design
5.1-5.4	Programming languages
6.1-6.2	The field of software engineering
7.1-7.2	Elementary data structures
8.1-8.2	Elementary file structures
9.1-9.2, 9.6	Introduction to database technology
10.1-10.3	The field of artificial intelligence
11.1-11.2	Computability

In addition to the overall plot, there are several themes woven throughout the text. One is that computer science is dynamic. The text repeatedly presents topics in a historical perspective, discusses the state of the art, and indicates directions of current research. Another theme is the role of abstraction and the way in which abstract tools are used to control complexity.

Web Sites

This text is supported by a Web site at `http://www.aw.com/brookshear`. This is the text's official site maintained by Addison-Wesley. There you will find materials for both students and teachers such as supporting software (for example, simulators for the machine used as an example in Chapter 2 and described in Appendix C), laboratory manuals, links to additional topics of interest, an instructor's guide, and PowerPoint slides. You may also wish to check out my personal Web site at `http://mscs.mu.edu/~glennb`. It is not very formal (and it is subject to my whims), but I tend to keep some information there that you may find helpful.

To Students

I was introduced to the field of computing during my tour in the US Navy back in the late 1960s and early 1970s. (Yes, that makes me old—but it will happen to you also. Furthermore, being old makes me wise so you should listen to what I say.) I spent most of these Navy days maintaining the system software at the Navy's computer installation in London, England. After my tour was completed, I returned to school and finished my Ph.D. in 1975. I've been teaching computer science and mathematics ever since.

A lot has changed in computer science over the years, but a lot has remained the same. In particular, computer science was, and still is, fascinating. There are a lot of awesome things going on out there. The development of the Internet, progress in artificial intelligence, and the ability to collect and disseminate information in unheard of proportions are only some of the things that will affect your life. You live in an exciting, changing world, and you have the opportunity to be a part of the action. Take it!

I'm a bit of a nonconformist (some of my friends would say *more* than a bit) so when I set out to write this text I didn't always follow the advice I received. In particular, many argued that certain material is too advanced for beginning students. But, I believe that if a topic is relevant, then it is relevant even if the academic community considers it an "advanced topic." You deserve a text that presents a complete picture of computer science—not a watered-down version containing artificially simplified presentations of only those topics that have been deemed acceptable for introductory students.

Thus, I have not avoided topics. Instead I've sought better explanations. I have tried to provide enough depth to give you an honest picture of what computer science is all about. (There's a difference between the fact that launching the space shuttle makes a lot of noise and the realization that it rattles every bone in your body.) As in the case of spices in a recipe, you may choose to omit some of the topics in the following pages, but they are there for you to taste if you wish—and I encourage you to do so.

Finally I should point out that in any course dealing with technology, the details you learn today may not be the details you will need to know tomorrow. The field is dynamic—that's part of the excitement. This book will give you a current picture of the subject as well as a historical perspective. With this background you will be prepared to grow along with technology. I encourage you to start the growing process now by exploring beyond this text. Learn to learn.

Thank you for the trust you have placed in me by choosing to read my book. As an author I have an obligation to produce a manuscript that is worth your time. I hope you find that I have lived up to this obligation.

To Instructors

There is more material in this text than can normally be covered in a single semester so do not hesitate to skip topics that do not fit your course objectives or to rearrange the order as you see fit. I wrote the book to be used as a course resource—not as a course definition. You will find that, although the text follows a plot, the topics are covered in an independent manner that allows you to pick and choose as you desire (see Figure 0.7 for a pictorial summary of the entire text).

On the opening page of each chapter I have used asterisks to indicate those sections that I suggest as optional—they delve more deeply into material or branch in directions you may not wish to pursue. But these are merely suggestions. In particular, you will find that the condensed version of the text outlined earlier in this preface omits more than merely the "optional" sections. To clarify, consider Chapter 7, Data Structures. Depending on your course objectives, you may handle this chapter in any of the following ways—all of which I, myself, have done at one time or another. First, in a "computer literacy" course, you may choose to skip the entire chapter. If you merely want to introduce the subject of data structures, you would probably cover only Sections 7.1 and 7.2 (as suggested in the condensed version). If, in addition, you want to present the basic structures themselves, you will need to cover Sections 7.1 through 7.6. Finally, if you want to extend the study to include customized data types or pointers in machine language, you will want to include the "optional" Sections 7.7 and/or 7.8.

I also suggest that you consider covering some topics as reading assignments or encourage students to read the material not included in your course. I think we underrate students if we assume that we have to explain everything in class. We should be helping them learn to learn on their own.

I have already explained that the text follows a bottom-up, concrete-to-abstract organization, but I want to expand on this a bit. As academics we too often assume that students will appreciate our perspective of a subject—often one that we have developed over years of working in a field. As teachers we do better by presenting material from the student's perspective. This is why the text starts with data representation/storage, machine architecture, and machine language. These are topics to which students readily relate—they can see the computer's components, they can hold them, and most students will have bought and used them. By starting the course with these topics, I see the students discovering answers to many of the "why" questions they have been carrying for years and learning to view the course as practical rather than theoretical. From this beginning it is natural to move on to the more abstract issues of algorithm discovery, design, representation, and complexity that those of us in the field see as the main topics in the course.

We are all aware that students learn a lot more than we teach them directly, and the lessons they learn indirectly are often better absorbed than those that are studied explicitly. This is significant when it comes to "teaching" problem solving. Students do not learn to solve problems by studying problem-solving methodologies as an isolated subject. They learn to solve problems by solving problems. So I have included numerous problems throughout the text. I encourage you to use them and to expand on them.

Another topic that I place in this same category is that of professionalism, ethics, and social responsibility. I do not believe that this material should be presented as an isolated subject. Instead, it should surface when it is relevant, which is the approach I have taken in this text. You will find that Sections 0.5, 3.7, 6.1, 6.8, 9.6, 10.1, and 10.7 present such topics as security, privacy, liability, and social awareness in the context of networking, database systems, software engineering, and artificial intelligence. You will also find that each chapter includes a collection of questions called *Social Issues* that challenge students to think about the relationship between the material in the text and the society in which they live.

Pedagogical Features

This text is the product of many years of teaching. As a result, it is rich in pedagogical aids. Paramount is the abundance of problems to enhance the student's participation—over 1,000 in this seventh edition (1,010 to be precise). These are classified as *Questions/Exercises*, *Chapter Review Problems*, and *Social Issues*. The *Questions/Exercises* appear at the end of each section. 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.

The *Chapter Review Problems* appear at the end of each chapter (except for the introductory chapter). They are designed to serve as "homework" problems in that they cover the material from the entire chapter and are not answered in the text.

Also at the end of each chapter are the questions in the *Social Issues* category. They are designed for thought and discussion. Many of them can be used to launch research assignments culminating in short written or oral reports.

Each chapter also ends with a list called *Additional Reading* that contains references to other materials relating to the subject of the chapter. The Web sites, identified earlier in this preface, are also good places to look for related material.

Seventh Edition

Although this seventh edition maintains the same chapter-by-chapter structure as previous editions of this text, topics have been added, some have been deleted, and much of the remaining material has been rewritten to provide an up-to-date and relevant picture of the science of computing.

The most significant distinctions between the sixth and seventh editions are pedagogical in nature. Much of the material has been reorganized and rewritten to improve clarity and simplify explanations. For example, Sections 2.1 and 2.2 (Computer Architecture and Machine Language) have been reorganized, the formal introduction to algorithms in Section 4.1 has been softened, the introduction to data structures (Section 7.1) has been reorganized, Section 7.7 (Customized Data Types) has been streamlined, the material on sequential and text files has been combined into a single section (8.2), the material on computability (Sections 11.1-11.3) has been rewritten. Moreover, numerous figures have been added and the artwork has been improved extensively.

There are of course numerous additions of topics as well. These include techniques of encoding sound (Chapter 1), expanded coverage of networking (Chapter 3), open-source development (Chapter 6), additional material on copyrights and patents (Chapter 6), XML (Chapter 8), and associative memory (Section 10.4). Moreover, numerous sidebars that expand the material in the text have been added throughout.

You will also find that this seventh edition has a new design and art program that gives the book a more "open" appearance than its predecessors. The goal is to make the material in the text more accessible and less daunting to beginning students. I hope you like it.

Acknowledgments

I first thank those of you who have supported this book by reading and using it in previous editions. I am honored.

With each new edition, the list of those who have contributed to the book as reviewers and consultants grows. Today this list includes J. M. Adams, C. M. Allen, D. C. S. Allison, B. Auernheimer, P. Bankston, M. Barnard, P. Bender, K. Bowyer, P. W. Brashear, C. M. Brown, B. Calloni, M. Clancy, R. T. Close, D. H. Cooley, L. D. Cornell, M. J. Crowley, F. Deek, M. Dickerson, M. J. Duncan, S. Fox, N. E. Gibbs, J. D. Harris, D. Hascom, L. Heath, P. B. Henderson, L. Hunt, M. Hutchenreuther, L. A. Jehn, K. Korb, G. Krenz, J. Liu, T. J. Long, C. May, W. McCown, S. J. Merrill, K. Messersmith, J. C. Moyer, M. Murphy, J. P. Myers, Jr., D. S. Noonan, S. Olariu, G. Rice, N. Rickert, C. Riedesel, J. B. Rogers, G. Saito, W. Savitch, R. Schlafly, J. C. Schlimmer, S. Sells, G. Sheppard, Z. Shen, J. C. Simms, M. C. Slattery, J. Slimick, J. A. Slomka, D. Smith, J. Solderitsch, R. Steigerwald, L. Steinberg, C. A. Struble, C. L. Struble, W. J. Taffe, J. Talburt, P. Tromovitch, E. D. Winter, E. Wright, M. Ziegler, and one anonymous. To these individuals I give my sincere thanks.

I also thank the people at Addison-Wesley, Argosy Publishing, and Theurer Briggs Design whose efforts are reflected within these pages. A team that goes though the process of publishing a book becomes a family. My publishing family has grown to include more wonderful people during the production of this seventh edition.

My wife Earlene and daughter Cheryl have been tremendous sources of encouragement over the years. I thank them for putting up with the author in me. They've seen that "the

book" can truly make an absent-minded professor absent minded. It's comforting to be able to drift off into such academic pursuits as writing a book knowing that someone else is keeping in touch with the real world. In particular, on the morning of December 11, 1998, I survived a heart attack because Earlene got me to the hospital in time. (For those of you in the younger generation I should explain that surviving a heart attack is sort of like getting an extension on a homework assignment.)

Finally, I thank my parents, to whom this book is dedicated, for instilling in me the importance of education. I close with the following endorsement whose source shall remain anonymous: "Our son's book is really good. Everyone should read it."

J. G. B.

Contents

CHAPTER 0

Introduction

Computer science is the discipline that seeks to build a scientific foundation for such topics as computer design, computer programming, information processing, algorithmic solutions of problems, and the algorithmic process itself. It provides the underpinnings for today's computer applications as well as the foundations for tomorrow's applications. This breadth means that we cannot become knowledgeable in computer science by studying only a few topics as isolated subjects or by merely learning how to use the computing tools of today. Rather, to understand the science of computing, we must grasp the scope and dynamics of a wide range of topics.

This book is designed to provide such a background. It presents an integrated introduction to the subjects that constitute a typical university computer science curriculum. The book can therefore serve as a foundation for beginning computer science students or as a source for other students seeking an introduction to the science behind today's computer-oriented society.

0.1 The Study of Algorithms

We begin with the most fundamental concept of computer science—that of an algorithm. Informally, an **algorithm** is a set of steps that defines how a task is performed.[1] For example, there are algorithms for constructing model airplanes (expressed in the form of instruction sheets), for operating washing machines (usually displayed on the inside of the washer's lid), for playing music (expressed in the form of sheet music), and for performing magic tricks (Figure 0.1).

Before a machine can perform a task, an algorithm for performing that task must be discovered and represented in a form that is compatible with the machine. A machine-compatible representation of an algorithm is called a **program.** Programs, and the algorithms they represent, are collectively referred to as **software,** in contrast to the machinery itself, which is known as **hardware.**

The study of algorithms began as a subject in mathematics. Indeed, the search for algorithms was a significant activity of mathematicians long before the development of today's computers. The major goal was to find a single set of directions that described how all problems of a particular type could be solved. One of the best known examples of this early research is the long division algorithm for finding the quotient of two multiple-digit numbers. Another example is the Euclidean algorithm, discovered by the ancient Greek mathematician Euclid, for finding the greatest common divisor of two positive integers (Figure 0.2).

Once an algorithm for performing a task has been found, the performance of that task no longer requires an understanding of the principles on which the algorithm is based. Instead, the performance of the task is reduced to the process of merely following directions. (We can follow the long division algorithm to find a quotient or the Euclidean algorithm to find a greatest common divisor without understanding why the algorithm works.) In a sense, the intelligence required to solve the problem at hand is encoded in the algorithm.

It is through this ability to capture and convey intelligence by means of algorithms that we are able to build machines that display intelligent behavior. Consequently, the level of intelligence displayed by machines is limited by the intelligence that can be conveyed through algorithms. Only if we find an algorithm that directs the performance of a task can we construct a machine to perform that task. In turn, if no algorithm exists for solving a problem, then the solution of that problem lies beyond the capabilities of machines.

The development of algorithms is therefore a major goal within the field of computing, and consequently a significant part of computer science is concerned with issues relating to that task. In turn, we can gain an understanding of the breadth of computer science by considering some of these issues. One deals with the question of how algorithms are discovered in the first place—a question that is closely related to that of problem solving in general. To discover an algorithm for solving a problem is essentially to discover a solution for the problem. It follows that studies in this branch of

[1] *More precisely, an algorithm is an ordered set of unambiguous, executable steps that define a terminating activity. These details are discussed in Chapter 4.*

computer science draw heavily from such areas as the psychology of human problem solving and theories of education. We consider some of these ideas in Chapter 4.

Once an algorithm for solving a problem has been discovered, the next step is to represent the algorithm so it can be communicated to a machine or to other humans. This means that we must transform the conceptual algorithm into a clear set of instructions and represent these instructions in an unambiguous manner. Studies emerging from these concerns draw from our knowledge of language and grammar and have led to an abundance of algorithm representation schemes, known as **programming languages**, which are based on a variety of approaches to the programming process, known as programming paradigms. We consider some of these languages and the paradigms on which they are based in Chapter 5.

As computer technology has been applied to more and more complex problems, computer scientists have found that the design of large software systems involves more than the development of the individual algorithms for performing the required activities. It entails designing the interaction among these components as well. To deal with such complexities, computer scientists have turned to the well-established field of engineering in hopes of finding tools for handling such problems. The result is the branch of computer science known as software engineering, which today draws from such diverse

Figure 0.1

An algorithm for a magic trick

Effect: The performer places some cards from a normal deck of playing cards face down on a table and mixes them thoroughly while spreading them out on the table. Then, as the audience requests either red or black cards, the performer turns over cards of the requested color.

Secret and Patter:

Step 1. From a normal deck of cards, select ten red cards and ten black cards. Deal these cards face up in two piles on the table according to color.

Step 2. Announce that you have selected some red cards and some black cards.

Step 3. Pick up the red cards. Under the pretense of aligning them into a small deck, hold them face down in your left hand and, with the thumb and first finger of your right hand, pull back on each end of the deck so that each card is given a slightly *backward* curve. Then place the deck of red cards face down on the table as you say, "Here are the red cards in this stack."

Step 4. Pick up the black cards. In a manner similar to that in step 3, give these cards a slight *forward* curve. Then return these cards to the table in a face-down deck as you say, "And here are the black cards in this stack."

Step 5. Immediately after returning the black cards to the table, use both hands to mix the red and black cards (still face down) as you spread them out on the tabletop. Explain that you are thoroughy mixing the cards.

Step 6. As long as there are face-down cards on the table, repeatedly execute the following steps:

> **6.1.** Ask the audience to request either a red or a black card.
>
> **6.2.** If the color requested is red and there is a face-down card with a concave appearance, turn over such a card while saying, "Here is a red card."
>
> **6.3.** If the color requested is black and there is a face-down card with a convex appearance, turn over such a card while saying, "Here is a black card."
>
> **6.4.** Otherwise, state that there are no more cards of the requested color and turn over the remaining cards to prove your claim.