国外著名高等院校信息科学与技术优秀教材

PEARSON Addison Wesley

计算机科学概论（第11版）（英文版）

COMPUTER SCIENCE *An Overview*

11th Edition

人民邮电出版社
POSTS & TELECOM PRESS

〔美〕J. Glenn Brookshear 著

国外著名高等院校信息科学与技术优秀教材

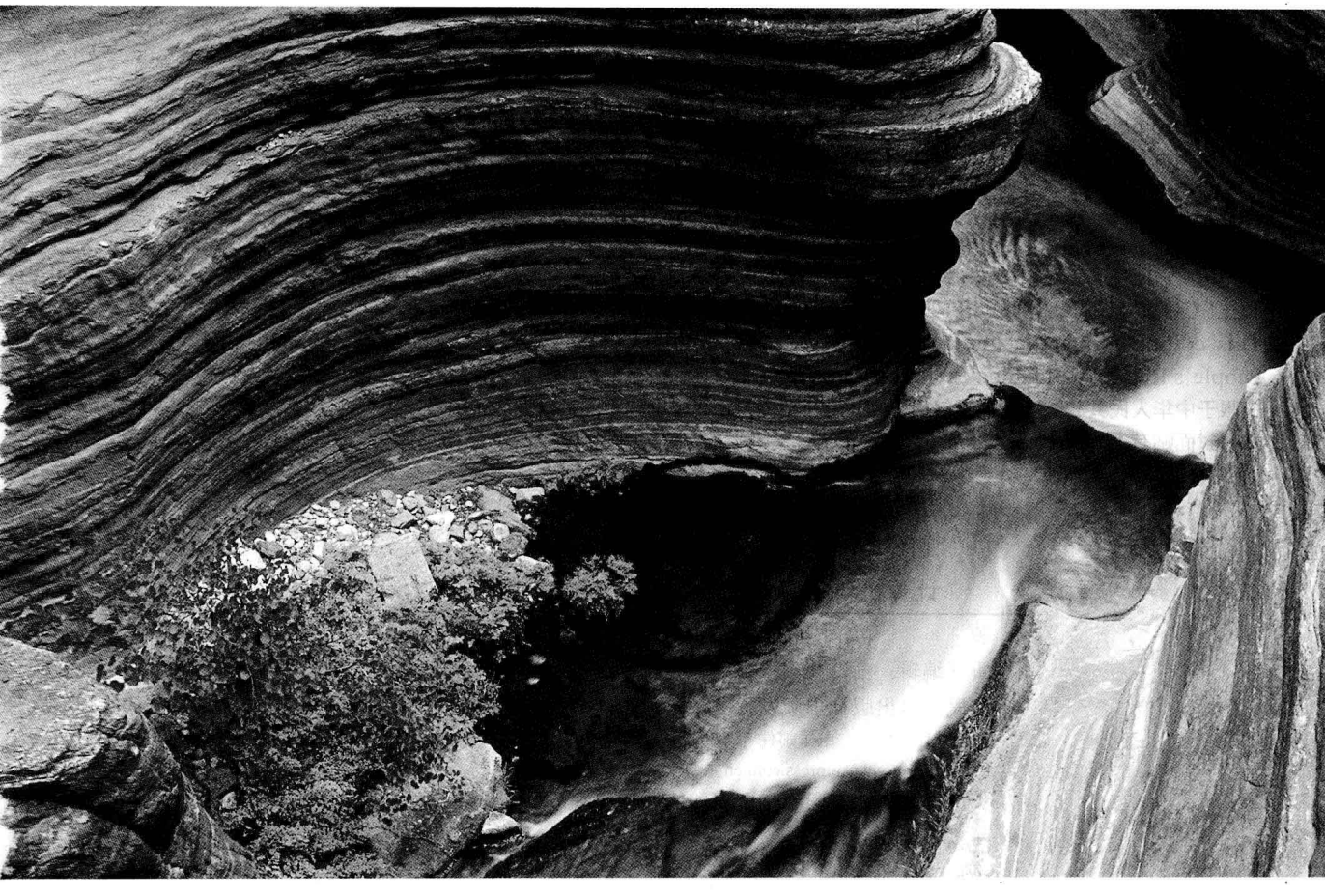
计算机科学概论

（第11版）

（英文版）

COMPUTER SCIENCE *An Overview*

11th Edition



〔美〕J. Glenn Brookshear 著

人民邮电出版社

北京

图书在版编目（CIP）数据

计算机科学概论 : 第11版 : 英文 / (美) 布鲁克希尔 (Brookshear, J.G.) 著. -- 北京 : 人民邮电出版社, 2012.5
国外著名高等院校信息科学与技术优秀教材
ISBN 978-7-115-27794-7

Ⅰ. ①计… Ⅱ. ①布… Ⅲ. ①计算机科学－高等学校－教材－英文 Ⅳ. ①TP3

中国版本图书馆CIP数据核字(2012)第046470号

版权声明

国外著名高等院校信息科学与技术优秀教材

计算机科学概论（第11版）（英文版）

◆ 著 [美] J. Glenn Brookshear
责任编辑 俞 彬

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
大厂聚鑫印刷有限责任公司印刷

◆ 开本：787×1092 1/16
印张：39
字数：737千字 2012年5月第1版
印数：1-3 000册 2012年5月河北第1次印刷
著作权合同登记号 图字：01-2012-2667号
ISBN 978-7-115-27794-7

定价：59.00元

读者服务热线：(010)67132692 印装质量热线：(010)67129223
反盗版热线：(010)67171154
广告经营许可证：京崇工商广字第0021号

前　言

本书是计算机科学的入门教材。在力求保持学科广度的同时，还兼顾深度，以对所涉及的主题给出中肯的评价。

读者对象

本书面向计算机科学以及其他各个学科的学生。大多数计算机科学专业的学生在最初的学习中都有这样一个误解，认为计算机科学就是程序设计、网页浏览以及因特网文件共享，因为这基本上就是他们所看到的一切。实际上计算机科学远非如此。因此，在入门阶段，学生们需要了解他们主攻的这门学科所涉及内容的广度，这也正是本书的宗旨。本书力图使学生对计算机科学有一个总体的了解，希望在这个基础上，他们可以领会该领域今后其他课程的特点以及相互关系。事实上，本书采用的综述方式也是自然科学入门教程的常用模式。

其他学科的学生如果想融入这个技术化社会，也需要具备这些宽泛的知识背景。适用于他们的计算机科学课程提供的应该是对整个领域很实用的剖析，而不仅仅是培训学生如何上网和使用一些流行的软件。当然这种培训也有其适用的地方，但本书的目的不在于此，而是用作计算机科学的教科书。

因此，在写这本书时，我们力求保持其对非技术类学生的可读性。因此，先前的版本已经被很成功地用作教科书，读者囊括了从高中生到研究生的各个教育层次众多专业的学生。这一版仍将贯彻这一目标。

第 11 版新增的内容

第11版主要增加关于手持移动设备的内容，特别是智能手机。因此，这一版的内容经过了修改、扩充，目的便是呈现所述主题与智能手机技术之间的关系。具体主题包括：

- 智能手机硬件；
- 3G网络与4G网络的区别；
- 智能手机操作系统；
- 智能手机软件开发；
- 智能手机的人机交互界面。

以上新增内容主要出现在第3章（操作系统）和第4章（组网），第6章（编程语言）和第7章（软件工程）中也有提及。

这一版中的其他明显变动包括对以下内容的更新。

- 软件所有权和责任：这一版重写并更新了第7章（软件工程）中与此主题相关的内容。
- 训练人工神经网络：相关内容（见第11章，人工智能）已经实现了现代化。

最后，这一版更新了全书内容来反应当今的技术状况。这主要体现在第0章（绪论）、第1

章（数据存储）和第2章（数据操作）。

章节安排

本书主题由具体到抽象逐步推进——这是一种很利于教学的顺序，每一个主题自然而然地引导出下一个主题。首先介绍的是信息编码、数据存储及计算机体系结构的基本原理（第1章和第2章），进而是操作系统（第3章）和计算机网络（第4章），接着探讨了算法、程序设计语言及软件开发（第5章至第7章），然后探索如何更好地访问信息（第8章和第9章），第10章讲述计算机图形学技术的一些重要应用，第11章涉及人工智能，第12章通过对计算理论的介绍来结束全书。

本书编排顺序自然连贯，但各个章节具有很强的独立性，可以单独查阅，也可以根据不同学习顺序重新排列。事实上，本书通常作为各类课程的教材，内容选择的顺序是多种多样的。其中一种教法是先介绍第5章和第6章（算法和程序设计语言），然后按照需要返回到前面相应章节。我还知道有人是从第12章有关可计算性的内容开始的。这本书还曾作为深入不同领域项目的基础，用于“高级研讨班”的教科书。对于不需要了解太多技术的学生，教学中可以重点讲述第4章（组网及因特网）、第9章（数据库系统）、第10章（计算机图形学）和第11章（人工智能）。

每章开篇都用星号标出了选学章节。选学章节要么是讨论更专业的话题，要么是对传统内容作深入探究。此举仅仅是为那些想采取不同阅读顺序的人提供一点建议。当然，还有其他读法。尤其对于那些寻求快速阅读的读者，我建议采取下面的阅读顺序。

章　节	主　题
1.1～1.4	数据编码和存储基础
2.1～2.3	计算机体系结构和机器语言
3.1～3.3	操作系统
4.1～4.3	组网及因特网
5.1～5.4	算法和算法设计
6.1～6.4	程序设计语言
7.1～7.2	软件工程
8.1～8.3	数据抽象
9.1～9.2	数据库系统
10.1～10.2	计算机图形学
11.1～11.3	人工智能
12.1～12.2	计算理论

在本书中有几条贯穿始终的主线。主线之一是计算机科学是不断发展变化的。本书从历史发展的角度反复呈现各个主题，讨论其当前的状况，并指出研究方向。另一条主线是抽象的作用以及用抽象工具控制复杂性的方式。该主线在第0章引入，然后在操作系统、体系结构、组网、算法、程序设计语言、软件工程、数据组织和计算机图形学等内容中反复体现。

致教师

本教材所包含的内容很难在一个学期内讲授完，因此一定要果断地砍掉不适合自己教学目标的那些主题，或者根据需要重新调整讲授顺序。你会发现，尽管本书有它固有的结构体系，但各个主题在很大程度上是相对独立的，完全可以根据需要作出选择。我写本书的目的是把它

作为一种课程的参考书，而非圈定课程的内容。我希望你把某些主题留作阅读作业，鼓励学生自己学习，而不在课堂讲授。如果我们认为所有的东西都一定要在课堂上讲，那就低估学生的能力了。我们应该教会他们独立学习。

关于本书从具体到抽象的组织结构，我觉得有必要多言几句。作为学者，我们总以为学生会欣赏我们对于学科的观点，这些观点通常是我们在某一领域多年工作中形成的。但作为老师，我认为最好从学生的视角呈现教材。这就是为什么本书首先介绍数据的表示/存储、计算机体系结构、操作系统以及组网，因为这些都是学生们最容易产生共鸣的主题——他们很可能听说过JPEG、MP3这些术语，可能用CD和DVD刻录过资料，买过计算机配件，应用过某一操作系统，或者上过因特网。我发现，从这些主题开始讲授这门课程，学生可以为许多困惑他们多年的问题找到答案，并且把这门课看做是实践课程而不是纯理论的课程。由此出发就会很自然地过渡到较抽象的内容上，例如算法、算法结构、程序设计语言、软件开发方法、可计算性以及复杂性等，而这些内容就是我们本领域的人所认为的计算机科学的主要内容。正如我前面所说的，我并不是强求大家都按此顺序讲课，只是鼓励你们如此尝试一下。

我们都知道，学生能学到的东西要远远多于我们直接传授的内容，而且潜移默化传授的知识更容易被吸收。当要“传授”问题的解决方法时，就更是如此。学生不可能通过学习问题求解的方法变成问题的解决者，他们只有通过解决问题——还不仅仅是那些精心设计过的“教科书式的问题”，才能成为问题的解决者。因此我在本书中加入了大量的问题，并特意让其中一些问题模棱两可——意味着正确方法或正确答案可能不只一个。我建议你们采用并充分拓展这些问题。

另一类适合“潜移默化学习”的主题还有职业精神、伦理和社会责任感。我认为这种内容不应该独立成章，而是应该在有所涉及时讨论，而这正是本书的编排方法。你们会发现，3.5节、4.5节、7.8节、9.7节和11.7节分别在操作系统、组网、软件工程、数据库系统和人工智能的上下文中提及了安全、隐私、责任和社会意识的问题。此外，0.6节就通过总结一些比较著名的理论而引入这一主题——这些理论都企图把伦理上的决断建立在哲学的坚实基础上。你还会发现，每一章都包含了“社会问题”小节，这些问题将鼓励学生思考现实社会与教材内容的关系。

感谢你对本书感兴趣。无论你是否选用本书作为教材，我都希望你认同它是一部好的计算机科学教育文献。

教学特色

本书是多年教学经验的结晶，因此在辅助教学方面考虑较多。最主要的是提供了丰富的问题以加强学生的参与——这一版包含1000多个问题，分为“问题与练习”、“复习题”和“社会问题”。“问题与练习”列在每节末尾（除了第0章外），用于复习刚刚讨论过的内容、扩充以前讨论过的知识，或者提示以后会涉及的有关主题。这些问题的答案可以从图灵社区（www.ituring.com.cn）本书网页免费注册下载。

“复习题”列在每章的末尾（第0章除外）。它们是课后作业，内容覆盖整章，在书中不给出答案。

“社会问题”也列在每章的末尾，供思考讨论。许多问题可以用来开展课外研究，可要求学生提交简短的书面或口头报告。

在每章的末尾还设有“课外阅读”，它列出了与本章主题有关的参考资料。同时，前言以及正文中所列的网址也非常适合查找相关资料。

补充材料

本书的许多补充材料可以从配套网站www.pearsonhighered.com/brookshear上找到。以下内容面向所有读者。

- 每章的实践项目帮助加深理解本教材的主题，并可以帮助了解其他相关主题。
- 每章的“自测题”帮助读者复习本书中的内容。
- 介绍Java和C++基本原理的手册，它在教学顺序上与本书是兼容的。

除此之外，教师还可以登录Pearson Education的教师资源中心（www.pearsonhighered.com）网站申请获得下面的教辅资料。

- 包含“复习题”答案的教师指南。
- PowerPoint幻灯片讲稿。
- 测试题库。

你也许还想看一下我的个人网站www.mscs.mu.edu/~glennb，不是很正式（体现了我某一时的灵感和幽默），但你或许能找到些有用的信息。特别值得一提的是，你将在此找到本书的勘误。

致学生

我有一点点偏执（一些朋友说我可远不是一点点），所以写本书时，我很少接受他人的建议，其中许多人认为一些内容对于初学者过于高深。我相信即使学术界把它们归为“高级论题”，但只要与主题相关就是合适的。读者需要的是一本全面介绍计算机科学的教科书，而不是“缩水”的版本——只包括那些简化了的、被认为适合初学者的主题。因此我不回避任何主题，而是力求寻找更好的解释。我力图在一定深度上向读者展示计算机科学最真实的一面。就好比对待菜谱里的那些调味品一样，你可以有选择地略过本书的一些主题，但我全部呈现出来是为了在你想要的时候供你“品尝”，而且我也鼓励你们去尝试。

我还要指出的是，在任何与技术有关的课程中，当前学到的详细知识未必就适合以后的需要。这个领域是发展变化的——这正是使人兴奋的方面。本书将从现实及历史的角度展现本学科的内容。有了这些背景知识，你们就会和技术一起成长。我希望你们现在就开始行动起来，不要局限于课本的内容，而要进行大胆探索。要学会学习。

感谢你们的信任，选择了我的这本书。作为作者，我有责任创作出值得一读的作品。我希望你们觉得我已经尽到了这份责任。

致谢

首先我要感谢那些支持本书（阅读并使用本书前几个版本）的人们，我感到很荣幸。

David T. Smith（宾夕法尼亚州印第安纳大学）和Dennis Brylow（马凯特大学）对这一版的制作给予了很大帮助。David的贡献主要集中在第0章、第1章、第2章、第7章和第11章，Dennis的贡献主要集中在第3章、第4章、第6章和第10章。如果没有他们的工作，这一版将不会出现。我衷心地感谢他们。

我在第10版的前言中已经提到过，要特别感谢Ed Angel、John Carpinelli、Chris Fox、Jim Kurose、Gary Nutt、Greg Riccardi和Patrick Henry Winston在第10版中所做的贡献。他们的努力

成果在第11版被保留了下来。

其他对这一版和之前版本作出贡献的人包括J. M. Adams、C. M. Allen、D. C. S. Allison、R. Ashmore、B. Auernheimer、P. Bankston、M. Barnard、P. Bender、K. Bowyer、P. W. Brashear、C. M. Brown、H. 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. Ezekiel、S. Fox、N. E. Gibbs、J. D. Harris、D. Hascom、L. Heath、P. B. Henderson、L. Hunt、M. Hutchenreuther、L. A. Jehn、K. K. Kolberg、K. Korb、G. Krenz、J. Liu、T. J. Long、C. May、J. J. McConnell、W. McCown、S. J. Merrill、K. Messersmith、J. C. Moyer、M. Murphy、J. P. Myers、Jr.，D. S. Noonan、W. W. Oblitey、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. Tonellato、P. Tromovitch、E. D. Winter、E. Wright、M. Ziegler，还有一位匿名的朋友。我向他们中的每一位致以最真诚的谢意。

如前所述，本书的配套网站上有Java和C++手册来讲述这两种语言的基础知识，与本书的内容相得益彰。它们是由Diane Christie撰写的，在此表示感谢。另外，还要感谢Roger Eastman，他是本书配套网站上每章实践项目的出题人。

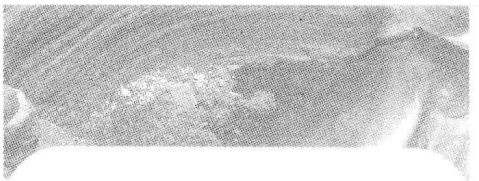
我同时要感谢为本项目作出贡献的Addison-Wesley的员工。他们不仅是很好的合作伙伴，而且是很好的朋友。如果你们打算写一本教材，可以考虑交给Addison-Wesley出版。

我还要感谢我的夫人Earlene和我的女儿Cheryl，感谢她们这么多年对我的鼓励。当然Cheryl已经长大，几年以前已经离家开始独自生活。Earlene还陪在我身边。我是一个幸运的人。1998年12月11日的早晨，我突发心脏病，是她及时把我送到了医院，让我逃过了一劫。（对于年轻一代的你们，我有必要解释一下，躲过心脏病的一劫有点像你们又获准延期提交课后作业。）

最后，我要感谢我的父母，本书即是给他们的献礼。我用下面一句赞美的话作为结束，就不说是他们哪一位说的了："我们儿子的书真的非常好，人人都应该阅读。"[①]

J. G. B.

① 本书前言部分由刘艺翻译。

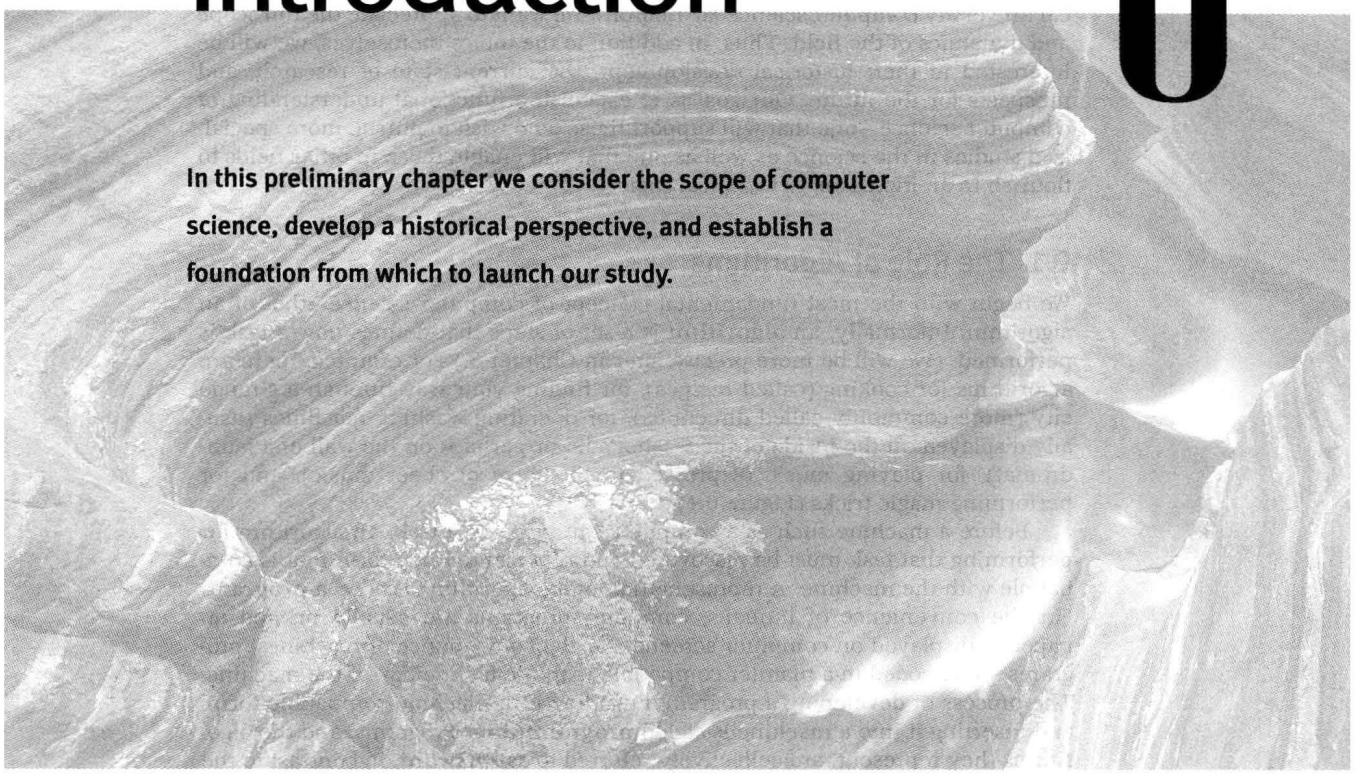


contents

*Asterisks indicate suggestions for optional sections.

CHAPTER

0 Introduction



In this preliminary chapter we consider the scope of computer science, develop a historical perspective, and establish a foundation from which to launch our study.

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 computing infrastructure.

This book provides a comprehensive introduction to this science. We will investigate a wide range of topics including most of those that constitute a typical university computer science curriculum. We want to appreciate the full scope and dynamics of the field. Thus, in addition to the topics themselves, we will be interested in their historical development, the current state of research, and prospects for the future. Our goal is to establish a functional understanding of computer science—one that will support those who wish to pursue more specialized studies in the science as well as one that will enable those in other fields to flourish in an increasingly technical society.

0.1 The Role 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. (We will be more precise later in Chapter 5.) For example, there are algorithms for cooking (called recipes), for finding your way through a strange city (more commonly called directions), for operating washing machines (usually displayed on the inside of the washer's lid or perhaps on the wall of a laundromat), for playing music (expressed in the form of sheet music), and for performing magic tricks (Figure 0.1).

Before a machine such as a computer 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 representation of an algorithm is called a **program.** For the convenience of humans, computer programs are usually printed on paper or displayed on computer screens. For the convenience of machines, programs are encoded in a manner compatible with the technology of the machine. The process of developing a program, encoding it in machine-compatible form, and inserting it into a machine is called **programming.** 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 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.

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.

Figure 0.2 The Euclidean algorithm for finding the greatest common divisor of two positive integers

Description: This algorithm assumes that its input consists of two positive integers and proceeds to compute the greatest common divisor of these two values.

Procedure:

Step 1. Assign M and N the value of the larger and smaller of the two input values, respectively.

Step 2. Divide M by N, and call the remainder R.

Step 3. If R is not 0, then assign M the value of N, assign N the value of R, and return to step 2; otherwise, the greatest common divisor is the value currently assigned to N.

It is through this ability to capture and convey intelligence (or at least intelligent behavior) by means of algorithms that we are able to build machines that perform useful tasks. Consequently, the level of intelligence displayed by machines is limited by the intelligence that can be conveyed through algorithms. We can construct a machine to perform a task only if an algorithm exists for performing that task. In turn, if no algorithm exists for solving a problem, then the solution of that problem lies beyond the capabilities of machines.

Identifying the limitations of algorithmic capabilities solidified as a subject in mathematics in the 1930s with the publication of Kurt Gödel's incompleteness theorem. This theorem essentially states that in any mathematical theory encompassing our traditional arithmetic system, there are statements whose truth or falseness cannot be established by algorithmic means. In short, any complete study of our arithmetic system lies beyond the capabilities of algorithmic activities.

This realization shook the foundations of mathematics, and the study of algorithmic capabilities that ensued was the beginning of the field known today as computer science. Indeed, it is the study of algorithms that forms the core of computer science.

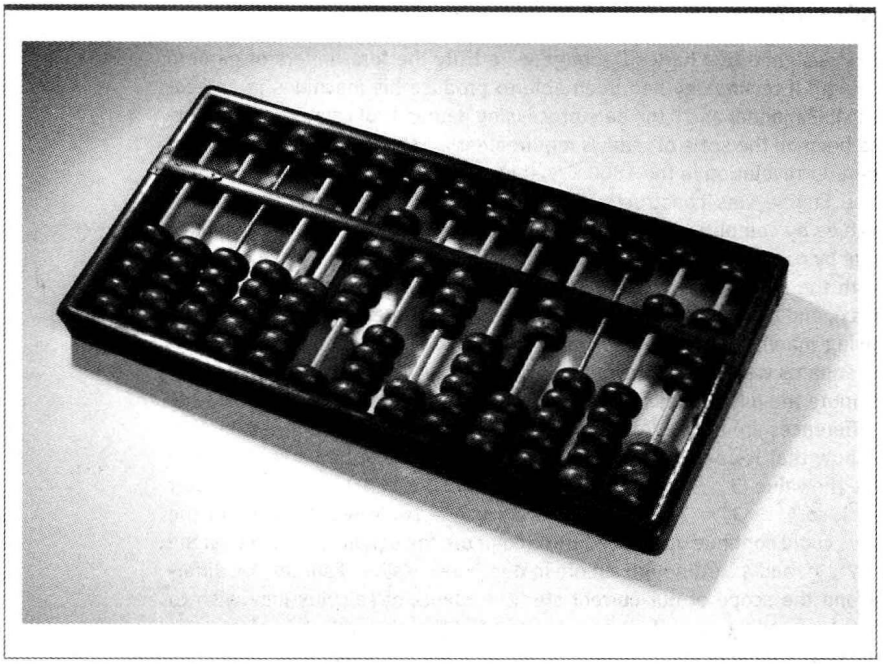
0.2 The History of Computing

Today's computers have an extensive genealogy. One of the earlier computing devices was the abacus. History tells us that it most likely had its roots in ancient China and was used in the early Greek and Roman civilizations. The machine is quite simple, consisting of beads strung on rods that are in turn mounted in a rectangular frame (Figure 0.3). As the beads are moved back and forth on the rods, their positions represent stored values. It is in the positions of the beads that this "computer" represents and stores data. For control of an algorithm's execution, the machine relies on the human operator. Thus the abacus alone is merely a data storage system; it must be combined with a human to create a complete computational machine.

In the time period after the Middle Ages and before the Modern Era the quest for more sophisticated computing machines was seeded. A few inventors began to experiment with the technology of gears. Among these were Blaise Pascal (1623–1662) of France, Gottfried Wilhelm Leibniz (1646–1716) of Germany, and Charles Babbage (1792–1871) of England. These machines represented data through gear positioning, with data being input mechanically by establishing initial gear positions. Output from Pascal's and Leibniz's machines was achieved by observing the final gear positions. Babbage, on the other hand, envisioned machines that would print results of computations on paper so that the possibility of transcription errors would be eliminated.

As for the ability to follow an algorithm, we can see a progression of flexibility in these machines. Pascal's machine was built to perform only addition. Consequently, the appropriate sequence of steps was embedded into the structure of the machine itself. In a similar manner, Leibniz's machine had its algorithms firmly embedded in its architecture, although it offered a variety of arithmetic operations from which the operator could select. Babbage's Difference Engine (of which only a demonstration model was constructed) could be modified to perform a variety of calculations, but his Analytical Engine (the construction for which he

Figure 0.3 An abacus (photography by Wayne Chandler)



never received funding) was designed to read instructions in the form of holes in paper cards. Thus Babbage's Analytical Engine was programmable. In fact, Augusta Ada Byron (Ada Lovelace), who published a paper in which she demonstrated how Babbage's Analytical Engine could be programmed to perform various computations, is often identified today as the world's first programmer.

The idea of communicating an algorithm via holes in paper was not originated by Babbage. He got the idea from Joseph Jacquard (1752–1834), who, in 1801, had developed a weaving loom in which the steps to be performed during the weaving process were determined by patterns of holes in large thick cards made of wood (or cardboard). In this manner, the algorithm followed by the loom could be changed easily to produce different woven designs. Another beneficiary of Jacquard's idea was Herman Hollerith (1860–1929), who applied the concept of representing information as holes in paper cards to speed up the tabulation process in the 1890 U.S. census. (It was this work by Hollerith that led to the creation of IBM.) Such cards ultimately came to be known as punched cards and survived as a popular means of communicating with computers well into the 1970s. Indeed, the technique lives on today, as witnessed by the voting issues raised in the 2000 U.S. presidential election.

The technology of the time was unable to produce the complex gear-driven machines of Pascal, Leibniz, and Babbage in a financially feasible manner. But with the advances in electronics in the early 1900s, this barrier was overcome. Examples of this progress include the electromechanical machine of George Stibitz, completed in 1940 at Bell Laboratories, and the Mark I, completed in 1944