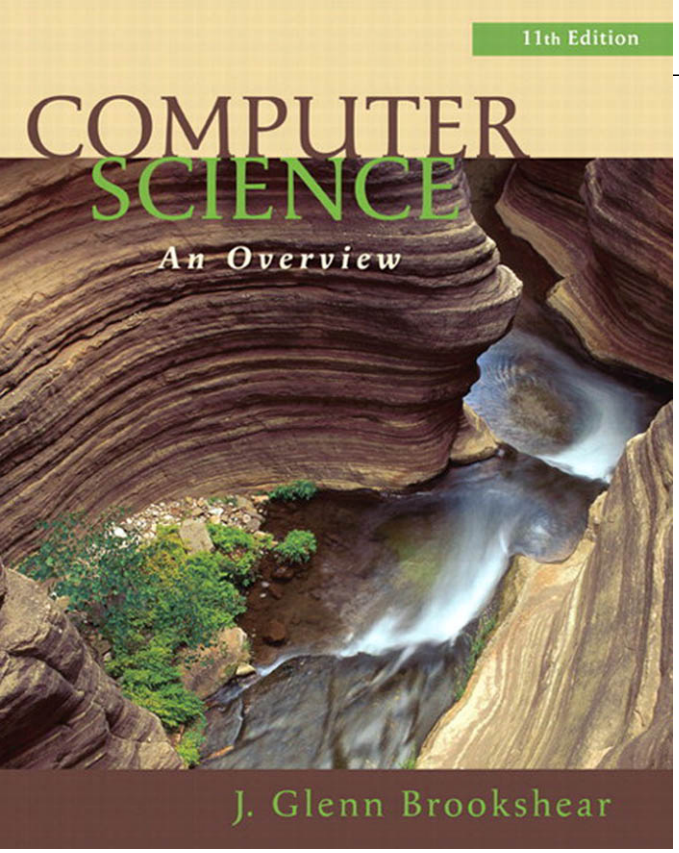

11th Edition
COMPUTER SCIENCE
An Overview
J. Glenn Brookshear

computer science

AN OVERVIEW

This page intentionally left blank

computer science

AN OVERVIEW

11th Edition

J. Glenn Brookshear

with contributions from

David T. Smith

Indiana University of Pennsylvania

Dennis Brylow

Marquette University

Addison-Wesley

Boston Columbus Indianapolis New York San Francisco Upper Saddle River
Amsterdam Cape Town Dubai London Madrid Milan Munich Paris Montréal Toronto
Delhi Mexico City São Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo

Editorial Director: Marcia Horton
Editor-in-Chief: Michael Hirsch
Editorial Assistant: Stephanie Sellinger
Vice President of Marketing: Patrice Jones
Marketing Manager: Yezan Alayan
Marketing Coordinator: Kathryn Ferranti
Vice President, Production: Vince O'Brien
Managing Editor: Jeff Holcomb
Production Project Manager: Kayla Smith-Tarbox
Senior Operations Supervisor: Lisa McDowell
Art Directors: Jayne Conte and Kristine Carney
Cover Designer: Rachael Cronin
Cover Image: "Slot Canyon" © gettyimages® Inc.
RF Media Editor: Dan Sandin and Wanda Rockwell
Project Management: GEX Publishing Services
Composition and Illustration: GEX Publishing Services
Printer/Binder: Edwards Brothers
Cover Printer: Lehigh-Phoenix Color/Hagerstown

Credits
Figure 0.3: "An abacus ". © Wayne Chandler. **Figure 0.4:** "The Mark I computer." Courtesy of IBM corporate archives. Unauthorized use is not permitted. **Figure 10.1:** "A photograph of a viral world produced by using 3D graphics (from *Toy Story* by Walt Disney/Pixar Animation Studios) © Disney/Pixar. **Figure 10.6:** "A scene from *Shrek 2* by Dreamworks SKG. © Dreamworks/ Picture Desk Inc./Kobal collection. **Figure 11.19:** "Results of using a neural network to classify pixels in an image." Inspired by www.actapress.com. **Chapter 11, Robots Making History feature: a.** "A soccer robot kicks a ball during the RoboCup German Open 2010 on April 15, 2010 in Magdeburg, eastern Germany." © Jens Schlueter/AFP/ Getty Images/ Newscom. **b.** "Tartan Racing's "Boss—winner of the Urban Challenge, a contest sponsored by DARPA to have vehicles drive themselves an urban environment." © DARPA. **c.** "One of NASA's rovers—a robot geologist exploring the surface of Mars." Courtesy of NASA/JPL-Caltech.

Library of Congress Catologing-in-Publication Data available upon request.

10 9 8 7 6 5 4 3 2 1—EB—14 13 12 11 10

Addison-Wesley
is an imprint of

PEARSON

ISBN 10: 0-13-256903-5
ISBN 13: 978-0-13-256903-3

preface

This book presents an introductory survey of computer science. It explores the breadth of the subject while including enough depth to convey an honest appreciation for the topics involved.

Audience

I wrote this text for students of computer science as well as students from other disciplines. As for computer science students, most begin their studies with the illusion that computer science is programming, Web browsing, and Internet file sharing 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 theme 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 survey approach is, in fact, the model used for introductory courses in the natural sciences.

This broad background is also what students from other disciplines need if they are to relate to the technical society in which they live. A computer science course for this audience should provide a practical, realistic understanding of the entire field rather than merely an introduction to using the Internet or training in the use of some popular software packages. There is, of course, a proper place for training, but this text is about educating.

Thus, while writing this text, maintaining accessibility for nontechnical students was a major goal. The result is that previous editions have been used successfully in courses for students over a wide range of disciplines and educational levels, ranging from high school to graduate courses. This eleventh edition is designed to continue that tradition.

New in the Eleventh Edition

The underlying theme during the development of this eleventh edition was to update the text to include handheld mobile devices, in particular smartphones. Thus, you will find that the text has been modified, and at times expanded, to

present the relationship between the subject matter being discussed and smartphone technology. Specific topics include:

- Smartphone hardware
- The distinction between 3G and 4G networks
- Smartphone operating systems
- Smartphone software development
- The human/smartphone interface

These additions are most noticeable in Chapters 3 (Operating Systems) and 4 (Networking) but is also observable in Chapters 6 (Programming Languages), and 7 (Software Engineering).

Other prominent changes to this edition include updates to the following topics:

- Software ownership and liability: The material in Chapter 7 (Software Engineering) pertaining to this topic has been rewritten and updated.
- Training artificial neural networks: This material, in Chapter 11 (Artificial Intelligence), has been modernized.

Finally, you will find that the material throughout the text has been updated to reflect the state of today's technology. This is most prevalent in Chapter 0 (Introduction), Chapter 1 (Data Storage), and Chapter 2 (Data Manipulation).

Organization

This text follows a bottom-up arrangement of subjects 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 information encoding, data storage, and computer architecture (Chapters 1 and 2); progresses to the study of operating systems (Chapter 3) and computer networks (Chapter 4); investigates the topics of algorithms, programming languages, and software development (Chapters 5 through 7); explores techniques for enhancing the accessibility of information (Chapters 8 and 9); considers some major applications of computer technology via graphics (Chapter 10) and artificial intelligence (Chapter 11); and closes with an introduction to the abstract theory of computation (Chapter 12).

Although the text follows this natural progression, the individual chapters and sections are surprisingly independent and can usually be read as isolated units or rearranged to form alternative sequences of study. Indeed, the book is often used as a text for courses that cover the material in a variety of orders. One of these alternatives begins with material from Chapters 5 and 6 (Algorithms and Programming Languages) and returns to the earlier chapters as desired. In contrast, I know of one course that starts with the material on computability from Chapter 12. In still other cases the text has been used in "senior capstone" courses where it serves as merely a backbone from which to branch into projects in different areas. Courses for less technically oriented audiences may want to concentrate on Chapters 4 (Networking and the Internet), 9 (Database Systems), 10 (Computer Graphics), and 11 (Artificial Intelligence).

On the opening page of each chapter, I have used asterisks to mark some sections as optional. These are sections that cover topics of more specific interest or

perhaps explore traditional topics in more depth. My intention is merely to provide suggestions for alternative paths though the text. There are, of course, other shortcuts. In particular, if you are looking for a quick read, I suggest the following sequence:

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	Operating systems
4.1–4.3	Networking and the Internet
5.1–5.4	Algorithms and algorithm design
6.1–6.4	Programming languages
7.1–7.2	Software engineering
8.1–8.3	Data abstractions
9.1–9.2	Database systems
10.1–10.2	Computer graphics
11.1–11.3	Artificial intelligence
12.1–12.2	Theory of computation

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 current state of affairs, and indicates directions of research. Another theme is the role of abstraction and the way in which abstract tools are used to control complexity. This theme is introduced in Chapter 0 and then echoed in the context of operating system architecture, networking, algorithm development, programming language design, software engineering, data organization, and computer graphics.

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. You will find that, although the text follows a plot, the topics are covered in a largely independent manner that allows you to pick and choose as you desire. The book is designed to be used as a course resource—not as a course definition. I suggest encouraging students to read the material not explicitly 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 feel obliged to say a few words about the bottom-up, concrete-to-abstract organization of the text. I think as academics we too often assume that students will appreciate our perspective of a subject—often one that we have developed over many years of working in a field. As teachers I think we do better by presenting material from the student's perspective. This is why the text starts with data representation/storage, machine architecture, operating systems, and networking. These are topics to which students readily relate—they have most likely heard terms such as JPEG and MP3; they have probably recorded data on CDs and DVDs; they have purchased computer components; they have interacted with an operating system; and they have used the Internet. By starting the course with these topics, students discover answers to many of the "why" questions they have been carrying for years and learn to view the course as practical

rather than theoretical. From this beginning it is natural to move on to the more abstract issues of algorithms, algorithmic structures, programming languages, software development methodologies, computability, and complexity that those of us in the field view as the main topics in the science. As I've said before, the topics are presented in a manner that does not force you to follow this bottom-up sequence, but I encourage you to give it a try.

We are all aware that students learn a lot more than we teach them directly, and the lessons they learn implicitly are often better absorbed than those that are studied explicitly. This is significant when it comes to "teaching" problem solving. Students do not become problem solvers by studying problem-solving methodologies. They become problem solvers by solving problems—and not just carefully posed "textbook problems." So this text contains numerous problems, a few of which are intentionally vague—meaning that there is not necessarily a single correct approach or a single correct answer. I encourage you to use these and to expand on them.

Another topic in the "implicit learning" category is that of professionalism, ethics, and social responsibility. I do not believe that this material should be presented as an isolated subject that is merely tacked on to the course. Instead, it should be an integral part of the coverage that surfaces when it is relevant. This is the approach followed in this text. You will find that Sections 3.5, 4.5, 7.8, 9.7, and 11.7 present such topics as security, privacy, liability, and social awareness in the context of operating systems, networking, database systems, software engineering, and artificial intelligence. Moreover, Section 0.6 introduces this theme by summarizing some of the more prominent theories that attempt to place ethical decision making on a philosophically firm foundation. 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.

Thank you for considering my text for your course. Whether you do or do not decide that it is right for your situation, I hope that you find it to be a contribution to the computer science education literature.

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 eleventh edition. These are classified as Questions/Exercises, Chapter Review Problems, and Social Issues. The Questions/Exercises appear at the end of each section (except for the introductory chapter). 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 material relating to the subject of the chapter. The Web sites identified in this preface, in the text, and in the sidebars of the text are also good places to look for related material.

Supplemental Resources

A variety of supplemental materials for this text are available at the book's Companion Website: www.pearsonhighered.com/brookshear. The following are accessible to all readers:

- Chapter-by-chapter activities that extend topics in the text and provide opportunities to explore related topics
- Chapter-by-chapter "self-tests" that help readers to rethink the material covered in the text
- Manuals that teach the basics of Java and C++ in a pedagogical sequence compatible with the text

In addition, the following supplements are available to qualified instructors at Pearson Education's Instructor Resource Center. Please visit www.pearsonhighered.com or contact your Pearson sales representative for information on how to access them:

- Instructor's Guide with answers to the Chapter Review Problems
- PowerPoint lecture slides
- Test bank

You may also want to check out my personal Web site at www.mscs.mu.edu/~glennb. It is not very formal (and it is subject to my whims and sense of humor), but I tend to keep some information there that you may find helpful. In particular, you will find an errata page that lists corrections to errors in the text that have been reported to me.

To Students

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 was 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 to be 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 appropriate for introductory students. Thus, I have not avoided topics. Instead I've sought better explanations. I've tried to provide enough depth to give you an honest picture of what computer science is all about. As in the case of spices in a recipe, you may choose to skip 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.

I should also 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.

Acknowledgments

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

David T. Smith (Indiana University of Pennsylvania) and Dennis Brylow (Marquette University) played significant roles in the production this eleventh edition. David concentrated on Chapters 0, 1, 2, 7, and 11; and Dennis focused on Chapters 3, 4, 6, and 10. Without their hard work this new edition would not exist today. I sincerely thank them.

As mentioned in the preface to the tenth edition, I am indebted to Ed Angel, John Carpinelli, Chris Fox, Jim Kurose, Gary Nutt, Greg Riccardi, and Patrick Henry Winston for their assistance in the development of that edition. The results of their efforts remain visible in this eleventh edition.

Others who have contributed in this or previous editions include 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, and one anonymous. To these individuals I give my sincere thanks.

As already mentioned, you will find Java and C++ manuals at the text's Companion Website that teach the basics of these languages in a format compatible with the text. These were written by Diane Christie. Thank you Diane. Another thank you goes to Roger Eastman who was the creative force behind the chapter-by-chapter activities that you will also find at the Companion Website.

I also thank the people at Addison-Wesley who have contributed to this project. They are a great bunch to work with—and good friends as well. If you are thinking about writing a textbook, you should consider having it published by Addison-Wesley.

I continue to be grateful to my wife Earlene and daughter Cheryl who have been tremendous sources of encouragement over the years. Cheryl, of course, grew up and left home several years ago. But Earlene is still here. I'm a lucky man. On the morning of December 11, 1998, I survived a heart attack because she 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. 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

**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.

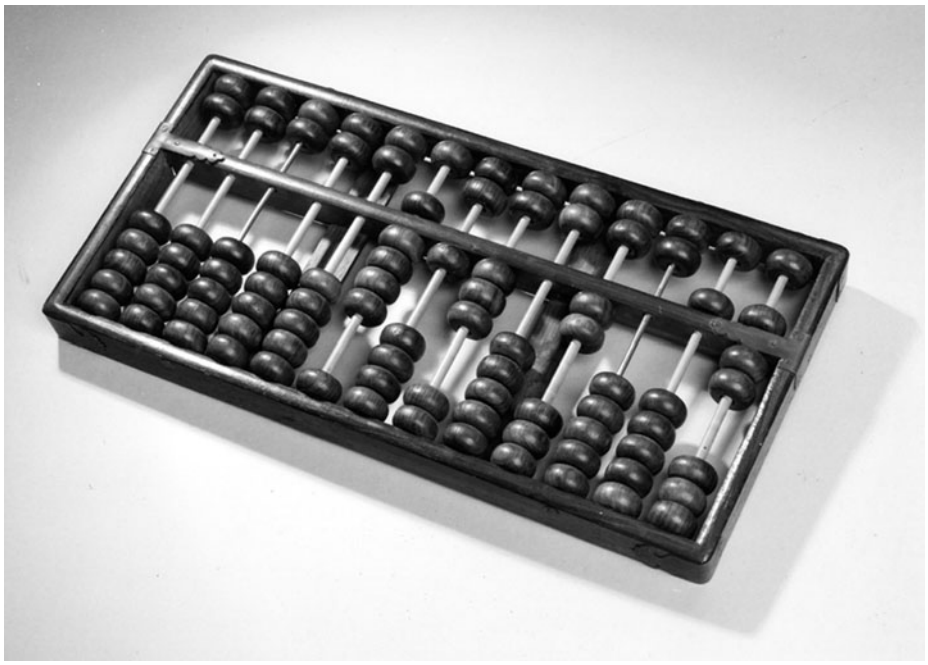
0.2 The History of Computing

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)



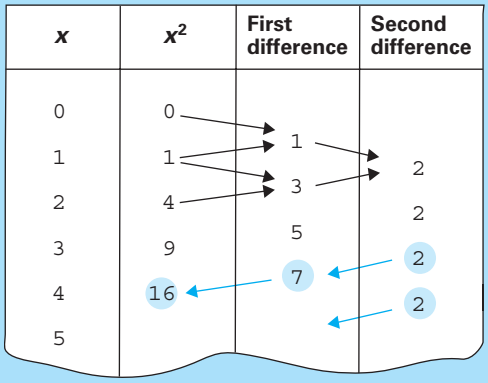
never received funding) was designed to read instructions in the form of holes in paper cards. Thus Babbage's Analytical Engine was programmable. In fact, Augusta Ada Byron (Ada Lovelace), who published a paper in which she demonstrated how Babbage's Analytical Engine could be programmed to perform various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

Babbage's Difference Engine

The machines designed by Charles Babbage were truly the forerunners of modern computer design. If technology had been able to produce his machines in an economically feasible manner and if the data processing demands of commerce and government had been on the scale of today's requirements, Babbage's ideas could have led to a computer revolution in the 1800s. As it was, only a demonstration model of his Difference Engine was constructed in his lifetime. This machine determined numerical values by computing "successive differences." We can gain an insight to this technique by considering the problem of computing the squares of the integers. We begin with the knowledge that the square of 0 is 0, the square of 1 is 1, the square of 2 is 4, and the square of 3 is 9. With this, we can determine the square of 4 in the following manner (see the following diagram). We first compute the differences of the squares we already know: $1^2 - 0^2 = 1$, $2^2 - 1^2 = 3$, and $3^2 - 2^2 = 5$. Then we compute the differences of these results: $3 - 1 = 2$, and $5 - 3 = 2$. Note that these differences are both 2. Assuming that this consistency continues (mathematics can show that it does) we conclude that the difference between the value $(4^2 - 3^2)$ and the value $(3^2 - 2^2)$ must also be 2. Hence $(4^2 - 3^2)$ must be 2 greater than $(3^2 - 2^2)$, so $4^2 - 3^2 = 7$ and thus $4^2 = 3^2 + 7 = 16$. Now that we know the square of 4, we could continue our procedure to compute the square of 5 based on the values of 1^2, 2^2, 3^2, and 4^2. (Although a more in-depth discussion of successive differences is beyond the scope of our current study, students of calculus may wish to observe that the preceding example is based on the fact that the derivative of $y = x^2$ is a straight line with a slope of 2.)



at Harvard University by Howard Aiken and a group of IBM engineers (Figure 0.4). These machines made heavy use of electronically controlled mechanical relays. In this sense they were obsolete almost as soon as they were built, because other researchers were applying the technology of vacuum tubes to construct totally electronic computers. The first of these machines was apparently the Atanasoff-Berry machine, constructed during the period from 1937 to 1941 at Iowa State College (now Iowa State University) by John Atanasoff and his assistant, Clifford Berry. Another was a machine called Colossus, built under the direction of Tommy

Figure 0.4 The Mark I computer (Courtesy of IBM archives. Unauthorized use is not permitted.)

Flowers in England to decode German messages during the latter part of World War II. (Actually, as many as ten of these machines were apparently built, but military secrecy and issues of national security kept their existence from becoming part of the "computer family tree.") Other, more flexible machines, such as the ENIAC (electronic numerical integrator and calculator) developed by John Mauchly and J. Presper Eckert at the Moore School of Electrical Engineering, University of Pennsylvania, soon followed.

From that point on, the history of computing machines has been closely linked to advancing technology, including the invention of transistors (for which physicists William Shockley, John Bardeen, and Walter Brattain were awarded a Nobel Prize) and the subsequent development of complete circuits constructed as single units, called integrated circuits (for which Jack Kilby also won a Nobel Prize in physics). With these developments, the room-sized machines of the 1940s were reduced over the decades to the size of single cabinets. At the same time, the processing power of computing machines began to double every two years (a trend that has continued to this day). As work on integrated circuitry progressed, many of the circuits within a computer became readily available on the open market as integrated circuits encased in toy-sized blocks of plastic called chips.

A major step toward popularizing computing was the development of desktop computers. The origins of these machines can be traced to the computer hobbyists who built homemade computers from combinations of chips. It was within this "underground" of hobby activity that Steve Jobs and Stephen Wozniak built a commercially viable home computer and, in 1976, established Apple Computer, Inc. (now Apple Inc.) to manufacture and market their products. Other companies that marketed similar products were Commodore, Heathkit, and Radio Shack. Although these products were popular among computer hobbyists, they

Augusta Ada Byron

Augusta Ada Byron, Countess of Lovelace, has been the subject of much commentary in the computing community. She lived a somewhat tragic life of less than 37 years (1815–1852) that was complicated by poor health and the fact that she was a nonconformist in a society that limited the professional role of women. Although she was interested in a wide range of science, she concentrated her studies in mathematics. Her interest in "compute science" began when she became fascinated by the machines of Charles Babbage at a demonstration of a prototype of his Difference Engine in 1833. Her contribution to computer science stems from her translation from French into English of a paper discussing Babbage's designs for the Analytical Engine. To this translation, Babbage encouraged her to attach an addendum describing applications of the engine and containing examples of how the engine could be programmed to perform various tasks. Babbage's enthusiasm for Ada Byron's work was apparently motivated by his hope that its publication would lead to financial backing for the construction of his Analytical Engine. (As the daughter of Lord Byron, Ada Byron held celebrity status with potentially significant financial connections.) This backing never materialized, but Ada Byron's addendum has survived and is considered to contain the first examples of computer programs. The degree to which Babbage influenced Ada Byron's work is debated by historians. Some argue that Babbage made major contributions whereas others contend that he was more of an obstacle than an aid. Nonetheless, Augusta Ada Byron is recognized today as the world's first programmer, a status that was certified by the U.S. Department of Defense when it named a prominent programming language (Ada) in her honor.

were not widely accepted by the business community, which continued to look to the well-established IBM for the majority of its computing needs.

In 1981, IBM introduced its first desktop computer, called the personal computer, or PC, whose underlying software was developed by a newly formed company known as Microsoft. The PC was an instant success and legitimized the desktop computer as an established commodity in the minds of the business community. Today, the term *PC* is widely used to refer to all those machines (from various manufacturers) whose design has evolved from IBM's initial desktop computer, most of which continue to be marketed with software from Microsoft. At times, however, the term *PC* is used interchangeably with the generic terms *desktop* or *laptop*.

As the twentieth century drew to a close, the ability to connect individual computers in a world-wide system called the **Internet** was revolutionizing communication. In this context, Tim Berners-Lee (a British scientist) proposed a system by which documents stored on computers throughout the Internet could be linked together producing a maze of linked information called the **World Wide Web** (often shortened to "Web"). To make the information on the Web accessible, software systems, called **search engines,** were developed to "sift through" the Web, "categorize" their findings, and then use the results to assist users researching particular topics. Major players in this field are Google, Yahoo, and Microsoft. These companies continue to expand their Web-related activities, often in directions that challenge our traditional way of thinking.

At the same time that desktop computers (and the newer mobile laptop computers) were being accepted and used in homes, the miniaturization of computing machines continued. Today, tiny computers are embedded within various devices. For example, automobiles now contain small computers running Global Positioning Systems (GPS), monitoring the function of the engine, and providing voice command services for controlling the car's audio and phone communication systems.

Perhaps the most potentially revolutionary application of computer miniaturization is found in the expanding capabilities of portable telephones. Indeed, what was recently merely a telephone has evolved into a small hand-held general-purpose computer known as a **smartphone** on which telephony is only one of many applications. These "phones" are equipped with a rich array of sensors and interfaces including cameras, microphones, compasses, touch screens, accelerometers (to detect the phone's orientation and motion), and a number of wireless technologies to communicate with other smartphones and computers. The potential is enormous. Indeed, many argue that the smartphone will have a greater effect on society than the PC.

The miniaturization of computers and their expanding capabilities have brought computer technology to the forefront of today's society. Computer technology is so prevalent now that familiarity with it is fundamental to being a member of modern society. Computing technology has altered the ability of governments to exert control; had enormous impact on global economics; led to startling advances in scientific research; revolutionized the role of data collection, storage, and applications; provided new means for people to communicate and interact; and has repeatedly challenged society's status quo. The result is a proliferation of subjects surrounding computer science, each of which is now a significant field of study in its own right. Moreover, as with mechanical engineering and physics, it is often difficult to draw a line between these fields and

Google

Founded in 1998, Google Inc. has become one of the world's most recoginzed techology companies. Its core service, the Google search engine, is used by millions of people to find documents on the World Wide Web. In addition, Google provides electronic mail service (called Gmail), an Internet based video sharing service (called YouTube), and a host of other Internet services (including Google Maps, Google Calendar, Google Earth, Google Books, and Google Translate).

However, in addition to being a prime example of the entrepreneurial spirit, Google also provides examples of how expanding technology is challenging society. For example, Google's search engine has led to questions regarding the extent to which an international company should comply with the wishes of individual governments; YouTube has raised questions regarding the extent to which a company should be liable for information that others distribute through its services as well as the degree to which the company can claim ownership of that information; Google Books has generated concerns regarding the scope and limitations of intelectual property rights; and Google Maps has been accused of violating privacy rights.