PART I

LEARNING TO CODE

1 How Important is Programming?

"To understand computers is to know about programming. The world is divided… into people who have written a program and people who have not."

Ted Nelson, *Computer Lib/Dream Machines* (1974)

How important is it for *you* to learn to program a computer?

Since the introduction of the first digital electronic computers in the 1940s, people have answered this question in surprisingly different ways.

During the first wave of commercial computing—in the 1950s and 1960s, when large and expensive mainframe computers filled entire rooms—the standard advice was that only a *limited* number of specialists would be needed to program computers using simple input devices like switches, punched cards, and paper tape. Even during the so-called "golden age" of corporate computing in America—the mid- to late 1960s—it was still unclear how many programming technicians would be needed to support the rapid computerization of the nation's business, military, and commercial operations. For a while, some experts thought that well-designed computer systems might eventually program themselves, requiring only a handful of attentive managers to keep an eye on the machines.

By the late 1970s and early 1980s, however, the rapid emergence of personal computers (PCs), and continuing shortages of computer professionals, shifted popular thinking on the issue. When consumers began to adopt low-priced PCs like the Apple II (1977), the IBM PC (1981), and the Commodore 64 (1982) by the millions, it seemed obvious that ground-breaking changes were afoot. The "PC Revolution" opened up new frontiers, employed tens of thousands of people, and (according to some enthusiasts) demanded new approaches to computer literacy. As Ted Nelson, a prolific inventor and computing advocate wrote, "You can and must understand computers NOW!" On learning to program computers, Nelson energetically compared programming to another American obsession—driving an

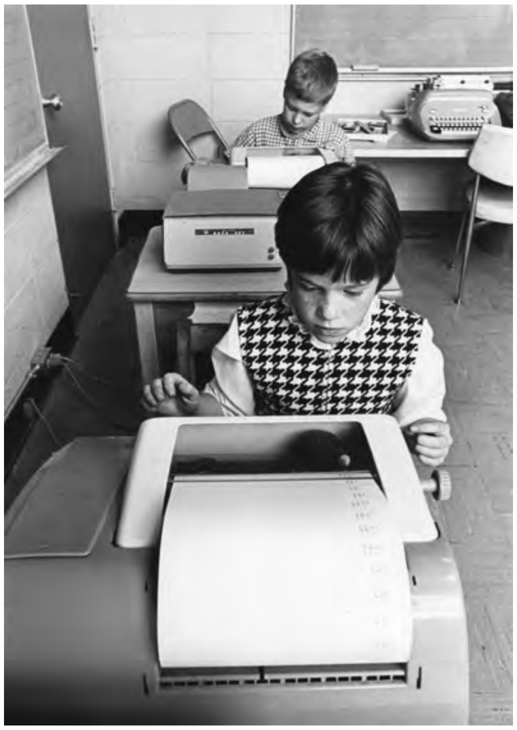


Figure 1.1 **American school children experiment with computer programming using teletype machines (1970s).** (Courtesy of the Computer History Museum)

automobile. "If you've never written a program, it's like never having driven a car," Nelson instructed. "You may get the general idea, but you may have little clear sense of the options, dangers, constraints, possibilities, difficulties, limitations, and complications."[1]

Ted Nelson was not alone. By the late 1970s, scores of programming advocates recommended that people of all ages learn to code as a way of understanding what the world's most intriguing devices were capable of. *Computer programming*—a process of formulating a problem for the computer to solve, writing instructions in a given computer language, loading instructions into the computer's memory, running the program, and correcting errors—had emerged as a major late-night pastime and (for some) a promising profession. In response to the mandate of Nelson and others, a surge of interest in programming developed, and the number of people who could write at least elementary programs grew from several thousand in

1. Ted Nelson, *Computer Lib Dream Machines* (Self-published, 1974; Microsoft Press revised edition, 1987), 40.

the early 1950s into millions by the early 1980s. (See Figure 1.1.) This sea change in computational literacy encouraged the widespread adoption of computers, boosted the global economy, and shaped the contours of the modern information age.

1.1 Programming Culture

This book is about the rise of computer programmers and the emerging social, technical, and commercial worldview that I call *programming culture*, which took a distinctive form during the early decades of microcomputers and personal computing, *c.* 1970–1995. It is a popular history of coding that explores the experiences of novice computer users, tinkerers, hackers, and power users, as well as the ideals and aspirations of computer scientists, educators, engineers, and entrepreneurs. A central part of this story is the *learn-to-program movement*, which germinated in government and university labs during the 1950s, gained momentum through counterculture experiments in the 1960s and early 1970s, became a broad-based educational agenda in the late 1970s and early 1980s, and was transformed by commercialization practices in the 1990s and 2000s. The learn-to-program movement sought to make computers more understandable, imprint useful technical skills, establish shared values, build virtual communities, and offer economic opportunities for technology enthusiasts. The movement also supported user communities, schools, and emerging commercial industries, many of which benefited from the utility and connectivity provided by digital electronic computers.

The learn-to-program movement had its ups and downs, but eventually set the stage for 21st century expressions of computational literacy, such as the Hour of Code, YouTube and Lynda courseware, certification programs, coding boot camps, and university degrees in disciplines such as computer science, software engineering, information technology, artificial intelligence, and (most recently) human–computer interaction. As the title of this book suggests, the learn-to-program movement fostered a groundswell of popular support for computing culture in America, resulting in what I call a *Code Nation*—a globally-connected society that is saturated with computer technology and enchanted by software and its creation.

The learn-to-program movement (or more broadly, the software-maker movement) has inspired both disciples and critics. It has evolved over time and its advocates have traversed numerous professional boundaries and cultural institutions. The movement is historically distinct but also follows the patterns and rhythms of earlier socio-technical transformations, including the introduction of steam-powered technologies during the Industrial Revolution, the electrification of American businesses and homes, and the production of automobiles and "car culture" in the early 20th century.

Borrowing terminology from information science and the history of technology, the learn-to-program movement is identifiable as part of the "diffusion" and "domestication" phases that take place when a successful new technology is spread or "propagated" across society.[2] Scholars from the field of business and economic history also recognize this transition as a key period in which a new discovery or invention is widely adopted and made useful for the general public, resulting in new consumer behaviors and potential changes in the way that a market or the broader economy functions.[3] To achieve wide-spread diffusion, the movement often benefits from sustaining ideologies that strengthen the allegiance of followers and justify the time, resources, and commitment that are necessary for the movement's success.

Beyond hopes for material gain, America's expanding programming culture can also be viewed as a manifestation of the deep and abiding cultural tendency that many describe as "technological enthusiasm."[4] Technological enthusiasm is an upbeat, optimistic appraisal of new technical systems that not only stoke the engines of capitalism, but provide access to the ideals embedded in what is known as the American Project and the American Dream. The publishers of PC software systems readily participated in this vision, as each wave of entrepreneur–engineer strived to improve their software, best their rivals, and boost the productivity of corporations and the general public. By the 1980s, software creation had taken the form of a consensus ideology that united many Americans in a common, accessible dream of a better future through computing. As I will discuss in Chapter 2, this enthusiasm brought additional computing mythologies to the fore, and their collective use contributed to the positive view that American's held about PCs and software in the years to come.

2. See *Computerization Movements and Technology Diffusion: From Mainframes to Ubiquitous Computing*, edited by Margaret S. Elliott and Kenneth L. Kraemer (Medford, NJ: Information Today, Inc., 2008).

3. For a discussion of the phases that take place when a new consumer technology is introduced, see Joseph J. Corn, *User Unfriendly: Consumer Struggles with Personal Technologies, from Clocks and Sewing Machines to Cars and Computers* (Baltimore, MD: Johns Hopkins University Press, 2011). Also useful is Claude S. Fischer, *America Calling: A Social History of the Telephone to 1940* (Berkeley: University of California Press, 1994); and the essay collection *Does Technology Drive History? The Dilemma of Technological Determinism*, eds. Merritt Roe Smith and Leo Marx (Cambridge, MA: The MIT Press, 1994).

4. See Thomas P. Hughes, *American Genesis: A Century of Invention and Technological Enthusiasm, 1870–1970*, Second Edition (Chicago: University of Chicago Press, 2004); David A. Hounshell, *From the American System to Mass Production, 1800–1932: The Development of Manufacturing Technology in the United States* (Baltimore, MD: The Johns Hopkins University Press, 1984).

1.2 Learning a Language

By the late 1960s, programming emerged from America's research labs and government institutions to have a direct influence on universities, primary and secondary schools (K-12 in the U.S.), and the nation's businesses. But what type of mental activity did programming entail? How should students take their first steps when learning to program a computer? In search of an analogy, some specialists suggested that learning to program was a bit like learning to read or speak in a foreign language. Utilizing the comparison, some educators pressed for the inclusion of computer languages in their school's curriculum. Rather than taking a year or two of a spoken language, such as Spanish or German, a few innovative programs offered courses in computer language instruction, including FORTRAN, Logo, BASIC, and Pascal.

School administrators eager to provide practical job training (and to mollify prospective students and their parents) broadened the definition of "foreign language" to include instruction in computer languages, algorithms, and database management. The popular press advocated for coding instruction in newspapers and special reports, and computer book and magazine publishers released hundreds of titles to help students build simple applications for time-sharing systems and the first PCs.

No one argued that computer languages were the *same* as human languages, of course. But programming advocates pointed to the many parallels that they observed in the structure of spoken and computer grammars, and to the ways that basic logic and reasoning were gradually introduced to students. Instruction in programming seemed to permit access to the private world of a computer and its "brain" or central processing unit (CPU). Programming was also portrayed as a valuable exercise in logical thinking and problem solving. It was a mental activity that provided a conceptual introduction to how computers worked, as well as a deep dive into logic and syntax. For all these reasons, computer literacy advocates recommended that those who planned to use computers in the future should learn to code as soon as possible. "Even if you don't write programs yourself," Ted Nelson advised in 1974, "you may have to work with people who do."[5]

In the early years of the electronic computer revolution, it was the imposing image of the new *machines* that seemed to fascinate the public. The physicality of mainframe computers was reinforced by images of large devices whirring and blinking in popular films such as *Desk Set* (1957), *2001: A Space Odyssey* (1968), *Colossus: The Forbin Project* (1970), *Logan's Run* (1976), and *War Games* (1983). As computers became more reliable and better understood, however, the focus of

5. Nelson, *Computer Lib*, 43.

popular attention turned away from computing machinery to *software*, the programs that ran on computers, and the coding experts who wrote code in high-level languages like FORTRAN, COBOL, BASIC, and C. The computer industry went through many transitions in the 1960s and 1970s, adding minicomputers and other special-purpose machines. Gradually, the attention of the computing community shifted from scientific and military systems to the application software that powered new types of businesses and helped them manage information.

By the 1980s and 1990s, it became apparent that there were *not enough qualified programmers* to design, build, and maintain the software systems needed in the U.S. as the country expanded its computational interests into new areas. Although the academic discipline of computer science had taken shape in America's colleges and universities, these programs could not graduate enough scientists and engineers to satisfy the industry's needs. The situation was much the same in the rest of the computerized world, in schools and markets stretching from Europe to Asia. Journalist Clive Thompson has written about it this way: "If you look at the history of the world, there are points in time when different professions become suddenly crucial, and their practitioners suddenly powerful. The world abruptly needs and rewards their particular set of skills."[6] Computer programmers suddenly became this influential group.

The "big bang" of software construction that took place in the 1970s created waves of demand for qualified programmers that continue to expand up to the present. Even in the Internet age—when learning to manage websites, write blogs, and use social media tools has taken on great importance—learning to code has not lost its appeal. As this book goes to press, the leaders of technology companies such as Amazon, Google, Facebook, Apple, and Microsoft regularly complain to Congress that the U.S. does not have enough qualified software developers to meet its needs. According to these advocates, a special exemption is needed in our national immigration policies to allow more foreign high-tech workers into the U.S. to satisfy the demand for software developers and associated fields, such as hardware engineering, artificial intelligence, data mining, computer security, user interface design, audio engineering, cloud computing, product testing and verification, technical writing, product support, project management, and related fields. Programmers have become the lifeblood of our technical society.

1.3 New Ways of Thinking

Calls to learn coding techniques abound now from the leaders of our digital economy. So, too, are warnings that if a group does *not* heed the call, they will miss

6. Clive Thompson, *Coders: The Making of a New Tribe and the Remaking of the World* (New York: Penguin Press, 2019), 11.

out on all or part of what the global digital economy has to offer. But where did this urgency to learn programming come from? What has motivated schools and non-profit organizations to devote so many resources to preparing instructions for a computer? When did programming literacy emerge as a national priority? And what were the early experiences of programmers as they tinkered with mainframes, minicomputers, and the first microcomputers? How is this story connected to the development of successful platforms such as CP/M, MS-DOS, Microsoft Windows, the Apple Macintosh, and Unix-based systems?

Whether past or present, the invitation to become a software maker is an invitation to join a distinctive community within our global society and economy. This computing subculture was founded by a small group of research scientists and academics, but it has expanded into a diverse assortment of hobbyists, students, gamers, artists, musicians, hackers, engineers, career professionals, and part-time workers. Although each of these groups is distinct in socio-economic terms, there is discernable common ground in their understanding of computers and computing technology. Computer programmers share a basic orientation to the world that is shared, despite the differences that they experience in relation to hardware and software systems, learning tools, and historical context.

As a thought experiment, imagine that each subgroup within the programming collective can be conceived of as a concentric circle. In such a model of our programming culture, the entire assortment of circles would be drawn in close proximity to one another, and most of the circles would have points of intersection and overlap. A shared exposure to computational thinking is the overlap, even if the programming languages that people learn (and the tools they write programs with) change over time. Some computer programmers may take up professional work, and others will remain as hobbyists or late-night hackers. Still others may learn programming skills as part of a journey that leads to other types of fruitful work. Despite the differences, and there will be many, the entire set of circles is a model of our nation's programming culture.[7]

7. Georg Simmel first developed the idea of "cross-cutting social circles" to discuss how different groups meet at points of common interest, dispute, or compromise. See Georg Simmel, *Conflict and The Web of Group-Affiliations*, trans. Kurt H. Wolff and Reinhard Bendix, respectively (Glencoe, IL, 1955, original Berlin, 1908). For additional studies in the history of technology that have influenced my approach, see Joseph J. Corn, ed., *Imagining Tomorrow: History, Technology, and the American Future* (Cambridge, MA: The MIT Press, 1986); David E. Nye, *Narratives and Spaces: Technology and the Construction of American Culture* (Exeter, UK: University of Exeter Press, 1997); Nina Lerman, Arwen Mohun, and Ruth Oldenziel, "The shoulders we stand on and the view from here: Historiography and directions for research," *Technology and Culture* 38 (1997): 9–30; David E. Nye, *Consuming Power: A Social History of American Energies* (Cambridge, MA: The MIT Press, 1998); Joseph J. Corn, *The Winged Gospel: America's Romance with Aviation* (Baltimore, MD: Johns Hopkins University Press, 2002); Greg Downey, "Commentary: The Place of Labor in the

The call to join ranks with computer programmers is not just an invitation to new ways of thinking (learning computational logic) and new consumer behaviors (buying software and a programming primer), it is also a call to new social relationships, to new ways of seeing and knowing, and to participating in new personal and professional networks. The programming circles that collectively shape America's technical identity are as much expressions of a distinct subculture as are the ideas and values of Impressionist artists and their admirers in *Fin-di-siècle* Paris or jazz musicians and their fans during the Swing Era in New York City.

As a social historian with interests in the history of technology, business, and education, I am curious about the experiences of today's programmers and software creators, and where they received their training, inspiration, and cultural worldviews. (See Figure 1.2.) Although the Internet era has contributed much to the behaviors and identity of contemporary software developers, the core skills and thought patterns of modern programmers were influenced by even earlier commitments and achievements. These included the proliferation of high-level languages in the 1950s, the introduction of software engineering techniques in the 1960s, the idealism of educators, entrepreneurs, and authors in the 1970s and 1980s, and the diffusion of commercial programming techniques in the 1990s and 2000s.

My argument is that the learn-to-program movement gained momentum through each of these important transitions, as programmers, authors, and entrepreneurs created pathways through which Americans might learn programming skills and the fine-points of creating software for specific platforms. Computer book authors, magazine publishers, and influential programmer/educators played important, if overlooked, roles in the diffusion of these new skills. By establishing an ideological connection to the computer literacy movement, programmer/educators established a framework that made computer programming feel important, rewarding, and attached to the rituals of citizenship and corporate belonging. The learn-to-program movement took shape through the efforts of many unsung heroes, both women and men, and one of my goals with this book is to reacquaint historians and programmers with a cast of interesting actors and protagonists who have been left out of recent narratives. Part of this work involves using visual sources to

History of Information-Technology Revolutions," in *Uncovering Labour in Information Revolutions, 1750–2000 (International Review of Social History),* vol. 38 Supplement 11 (2003), 225–261; Lisa Gitelman, *Always Already New: Media, History, and the Data of Culture* (Cambridge, MA: The MIT Press, 2008); and Christopher Tozzi, *For Fun and Profit: A History of the Free and Open Source Software Revolution* (Cambridge, MA: The MIT Press, 2017).

Figure 1.2 **A middle school student learns computational thinking in a programming camp sponsored by the Tacoma/South Puget Sound MESA organization.** (Photo: Joshua Wiersma/Pacific Lutheran University)

unpack the social context of historic computing environments. (See Figure 1.3.) I will profile social reformers, writers, teachers, tinkerers, entrepreneurs, and hackers, as well as computer scientists, students, engineers, and the leaders of America's computing societies, including the Association for Computer Machinery (ACM). Predictably, most of the programmers that we meet will be members of more than one social or professional group.

To get a sense for the magnitude of the sea change that took place, consider some basic demographics. In 1957, there were approximately 15,000 computer programmers employed in the U.S., a figure that accounts for approximately 80% of the world's programmers active that year. The work of the first computing pioneers involved building and maintaining military systems, designing algorithms for scientific research, tracking census data, and implementing data-processing schemes for government bureaus and corporations.

In 2000, there were approximately 9 million professional programmers worldwide, with millions more who had been exposed to coding concepts as part of

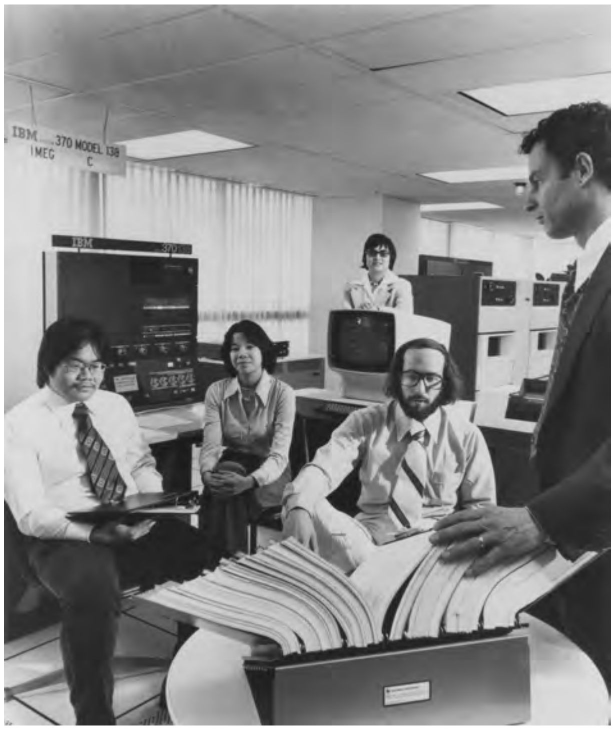


Figure 1.3 **Three men and two women gather for a meeting near an IBM 370 Model 138 Computer System in Berkeley, California. IBM's 1976 publicity photo emphasizes the value of teamwork and the extensive documentation that was prepared for programmers and administrators.** (Courtesy of the Computer History Museum)

their school curriculum or other experiences.[8] In addition to steady growth in military and scientific computing, the expanding digital economy has brought new opportunities for computer programmers in the fields of consumer software, video game programming, artificial intelligence, information publishing, digital communications, education, art, music, entertainment, medicine, and other areas that benefit from the use of computers.

The rising tide of opportunity for software developers has continued up to the present. In 2014, there were approximately 18.5 million software developers in the world, of which 11 million can be considered professional programmers and

8. Steve Lohr, *Go To: The Story of the Math Majors, Bridge Players, Engineers, Chess Wizards, Maverick Scientists, and Iconoclasts—The Programmers Who Created the Software Revolution* (New York: Basic Books, 2001), 6–7.

7.5 million can be considered as hobbyists.[9] Many programmers create or maintain software as part of their regular employment, while others write code for non-profit organizations that they support, and still others program at school, for recreation, or as an aspect of their personal or professional development.

1.4 Equity and Access

Despite the bright economic outlook for software developers, there are still numerous challenges in bringing programming proficiencies to the general population. In reality, only a small subset of the people who use computers actually go on to learn something about computational thinking or software development. Our modern economy requires many important job skills and personal investments. Considering the costs and the effort required, does it really matter who learns to program and who does not?

In the book *Stuck in the Shallow End: Education, Race, and Computing*, Jane Margolis et al. argue the "who" that learns to use technology matters a great deal, and that America has suffered throughout its history from inequities in access to computing.[10] Their research indicates that African-American and Latino children are *much less likely* to receive technology training in American schools than white or Asian children. When scholars analyze gender disparities and later professional outcomes, they find that only two out of ten information technology (IT) professionals are women in the current U.S. workforce.

Margolis and her contributors offer convincing evidence that the characteristics of programming culture matter tremendously to those who enter the subculture and to those who thrive in it (or recede from view). Understanding the long history of the learn-to-program movement and its cultural commitments and values reveals much about how people have interacted with computers in the past, and how we might expand computing opportunities in the future. Yasmin Kafai and Quinn Burke describe the challenge before us as working to better support "computational participation" in our schools and professional environments. In their important book, *Connected Code*, they recommend that thought leaders

9. International Data Corporation, *2014 Worldwide Software Developer and ICT-Skilled Worker Estimates* (Framingham, MA: International Data Corporation, 2014).

10. Jane Margolis, Rachel Estrella, Joanna Goode, Jennifer Jellison Holme, and Kimberly Nao, *Stuck in the Shallow End: Education, Race, and Computing*, Updated Edition (Cambridge, MA: The MIT Press, 2017). See also J. Margolis, J. Goode, and K. Binning, "Exploring computer science: active learning for broadening participation in computing," *Computing Research News* 27, no. 9 (October 2015).

Figure 1.4 **ACM member Professor Renzhi Cao teaches computer science to local middle school students in Tacoma, WA. Early engagement and outreach related to computational thinking has become a standard practice in many high-tech communities.** (Photo: John Froschauer / Pacific Lutheran University)

shape technology-centered cultures carefully, ensuring that *all* participants feel welcomed and included.[11] (See Figure 1.4.)

One important outgrowth of this research has been the rise of not-for-profit organizations that teach young people how to program, including Code.org, Black Girls Code, Girls Who Code, Native Girls Code, and The Hidden Genius Project. At the high school level, many organizations focus on introducing programming concepts and preparing students to take the College Board's AP Computer Science Principles examination. I evaluate the work of Code.org and the Hour of Code movement in the Afterword for this book. As a preview, I note here that Code.org has completed over 720 million introductory programming sessions since the organization began in 2013, with 46% female and 48% underrepresented minorities currently

11. Yasmin Kafai and Quinn Burke, *Connected Code: Why Children Need to Learn Programming* (Cambridge, MA: The MIT Press, 2016).

using the group's courseware.[12] These figures are clearly impressive, and they show one way that creative thinking and industry partnerships can support an education system that is struggling to find resources and leadership. However, the new initiatives were not created from whole cloth, but are simply the latest manifestations of a long programming literacy movement that has a fascinating history and is now being scaled to meet global needs. Just as in the past, there is an ongoing debate about the efficacy of wide-ranging computer literacy programs and the best way to deliver them.[13]

Importantly, this conversation about equity and access is connected to ethical considerations, and it will only move forward with input from many academic and industry partners. Our world is increasingly dependent on computers and technology, and it is imperative that we work together to understand the characteristics of technical communities and how they shape our hearts and minds. Computational thinking courses are among the most interesting places to attempt this work.

1.5 Personal Connections

I have wanted to write this book for a long time because I am fascinated with software development. PCs were an important starting place for me during my teenage years, and I first learned to write computer code on early microcomputers and PCs. Like many people of my age and social context, my first experiments with home electronics took place in the family rec room during the late 1970s. My extended family bought a Tandy TRS-80 and an Atari video computer system, and the young people in our circles used them to play video games like *Pong* and *Missile Command*. A bit later, I experimented with an early IBM Personal Computer when it was released in August 1981, just weeks before I entered college at Pacific Lutheran University (PLU) in Tacoma, Washington. I took an introductory computer programming course and declared as a Computer Science major at PLU, deferring my interests in history and education for graduate school. I learned BASIC, Pascal, C, and assembly language programming on the university's Digital Equipment Corporation (DEC) VAX 11-780 and DEC PDP-11 minicomputers. I also studied mathematics, data structures, algorithms, operating systems, digital logic, computer architecture, computer graphics, and networking theory. In 1985, I

12. See "Code.org 2018 Annual Report," February 12, 2019, 3. https://code.org/files/annual-report-2018.pdf. Accessed August 9, 2019.

13. For a summary of the current concerns and priorities in the computational literacy field, see Emmanuel Schanzer, Shriram Krishnamurthi, and Kathi Fisler, "Education: what does it mean for a computing curriculum to succeed?" *Communications of the ACM* 62, no. 5 (2019): 30–32.

Figure 1.5 **Michael Halvorson working in his office at Microsoft Corporation (1990).** (Photo courtesy of Michael Halvorson)

graduated from university and I was hired at Microsoft Corporation to work in one of their two Bellevue (Washington) office buildings, just before the company moved to its better-known Redmond campus. I was employee #850 in the rapidly expanding organization (see Figure 1.5), arriving when the best-selling products were MS-DOS, Microsoft Word for MS-DOS, and a few popular programming languages and development tools.

During my job interview at Microsoft, I was shown a testing (beta) version of Microsoft Windows 1.0. It was not very impressive at the time, but the new graphical operating environment for IBM PCs and compatibles would eventually become an exciting platform for many users, programmers, and commercial software publishers. My first work was at Microsoft Press, the book publishing division of Microsoft, founded by Bill Gates in 1983 to provide technical support for computer enthusiasts who were frustrated by the poor quality of software manuals. In the early days of personal computing, product documentation was often little more than printouts assembled in a three-ring binder, and there was not much in the way of computer-based help or training for PC users. From these humble beginnings, a major publishing industry took shape. It came to include bestselling magazines like *PC Magazine*, *Macworld*, and *Compute!*, as well as the computer book publishers Howard W. Sams, O'Reilly, Osborne McGraw-Hill, Que, Microsoft Press, Sybex, and IDG Books.

Our work at Microsoft Press was to help self-taught programmers and those who used Microsoft's business applications to get the most out of their software. I edited books, worked with independent authors, attended industry trade shows, and (beginning in 1986) started writing do-it-yourself (DIY) computer books about using operating systems and programming languages. I was lucky that my university training required a healthy dose of the liberal arts along with my computing classes. Both fields of study prepared me to tackle substantial research and writing projects in the years to come, and they were valued in the book publishing division.

The learn-to-program movement was something that I saw first-hand while working with Microsoft's customers and authors. In particular, there were fascinating people to learn from at computer industry trade shows, especially COMDEX and Macworld Expo. (See Figure 1.6.) In 1989, I co-authored the book *Learn BASIC Now* with my colleague and friend, David Rygmyr, and the book was carefully edited by Megan Sheppard and Dale Magee, Jr. (also employees of Microsoft Press). Our programming courseware included a full-featured version of the Microsoft QuickBASIC Interpreter for MS-DOS on three 5.25" disks, and Bill Gates wrote a Foreword to the book recalling his personal connection to Altair BASIC and his interest in using BASIC as a unifying language across computing platforms. (See Chapter 5.)

Learn BASIC Now sold many copies and it was selected as a finalist for a national book award in the computer book "How To" category. Our self-study guide clearly intersected with the powerful demand for programming instruction, and the low-cost QuickBASIC Interpreter made the product relatively inexpensive for newcomers. Over the years, I wrote another 15 books about software development, mostly for self-taught programmers and those who wanted to learn the newest features of popular products like Microsoft Visual Basic or Microsoft Visual Studio. Through the books, I was actively connected to publishers, software development teams, user groups, academics, journalists, literary agents, and a wide range of computer users—many of whom would write or email us directly for help.

1.6 Manifestos of the Movement

Despite my positive interactions with new programmers, I gradually learned that I was only a small part of the third or fourth wave of technical writers who had spread the message about computational literacy and learning to code in the years since the

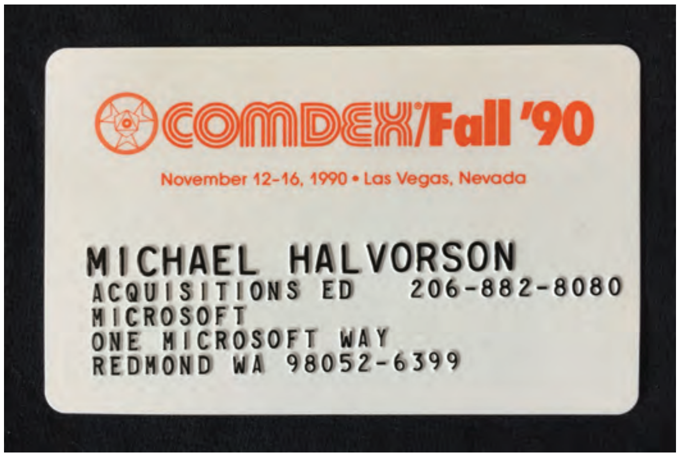


Figure 1.6 **An exhibitor badge from the COMDEX/Fall '90 trade show in Las Vegas, Nevada.** (Photo courtesy of Michael Halvorson)

introduction of the first computers. *Preparation of Programs for an Electronic Digital Computer* was published in 1951 by Maurice Wilkes, David Wheeler, and Stanley Gill to instruct readers on how to formulate machine code for the revolutionary EDSAC computer at the University of Cambridge.[14] Grace Mitchell, Daniel McCracken, and Elliott Organick also wrote creative programming primers for FORTRAN in the late 1950s and early 1960s, introducing non-specialists to programming.

In the era of time-sharing systems and early PCs, a new wave of programming advocates supported the movement. These were pioneers like Robert Albrecht and LeRoy Finkel, who participated in the People's Computer Company and the Homebrew Computer Club in Menlo Park, California. From the beginning, these visionaries understood that not only did people need to *buy computers* and start programming, but they needed to *learn how to program* through books, materials, and social interaction. These computing innovators wrote fascinating programs and produced several best-selling computer titles, but they have largely been neglected in the history of computing. A new book by Joy Lisi Rankin, *A People's History of Computing in the United States*, is an important exception to this lacuna, and Rankin demonstrates how Albrecht and his contemporaries inspired thousands of programmers to appreciate the benefits of BASIC.[15]

14. Maurice Wilkes, David Wheeler, and Stanley Gill, *Preparation of Programs for an Electronic Digital Computer* (Reading, MA: Addison-Wesley, 1951).

15. Joy Lisi Rankin, *A People's History of Computing in the United States* (Cambridge, MA: Harvard University Press, 2018), 68–69, 94–100.

Also important in the 1960s and 1970s were the pioneering efforts of the educational theorists Arthur Luehrmann, Seymour Papert, Cynthia Solomon, and Wally Feurzeig, all active in the computing hotbeds of Cambridge, Massachusetts and Greater Boston. Luehrmann coined the term “computer literacy” and encouraged students to learn structured programming with BASIC and Pascal. Papert, Solomon, and Feurzeig co-developed the Logo programming system at the Massachusetts Institute of Technology (MIT), and they wrote about its potential to teach computational thinking to children. Also, from the era of time-sharing systems, David Ahl, an early DEC employee, published tutorials that advocated for the use of computer games to teach programming concepts. My favorite of Ahl’s titles is *101 BASIC Computer Games*, published by DEC in 1973. This book is filled with mimeographed program listings that Ahl received in the mail from BASIC users across the U.S. It was one of the first bestselling computer programming titles, selling tens of thousands of copies to novice computer users, hobbyists, academics, and working professionals.

Many of the earliest manifestos of the learn-to-program movement were sold out of VW vans and dusty boxes in computer clubs. However, this DIY world was also on the fringes of the professional software development community, which took its energy from debates within the nascent software engineering movement and the emerging discipline of computer science. The standard-bearers in this field created the computers, operating systems, and programming languages that would fuel the academic and commercial worlds of software development in the years to come. Readers learned about their important discoveries through conferences and influential computer books such as Donald Knuth, *The Art of Computer Programming* (1968 and later); Kathleen Jensen and Niklaus Wirth, *The Pascal User Manual and Report* (1971); Brian Kernighan and Dennis Ritchie, *The C Programming Language* (1978); and Rodnay Zaks, *Programming the Z80* (1979). Although these authors did not always publish programming primers, they helped experienced programmers understand the cadence of computer languages, taught people to devise data structures and algorithms, and explored the advanced features of operating systems and computer architecture. The introduction of professional and commercial programming practices is a crucial stage of the learn-to-program movement.

1.7 A New History of Personal Computing

Code Nation explores the social, technical, and commercial changes that took place in the U.S. as computer programming became a regular part of life for so many. The trials and triumphs of PC programmers are featured on these pages, as well as the negative consequences that came to people who were denied the opportunity to code based on their location, gender, ethnicity, or economic circumstances. My emphasis is not on high-tech leadership strategies or the tactics that generated corporate wealth, but on the stories of lesser-known programmers, authors, academics, and entrepreneurs. Some were successful, and some have been mostly forgotten. But this is itself a lesson in the history of innovation, business, and technology.

To tell this tale, *Code Nation* presents a new history of personal computing in the U.S. I present a detailed analysis of early computer platforms, a discussion of important compilers and development tools, a "behind-the-scenes" look at application and operating-system programming, the origins of corporate and "enterprise" computing strategies, the rise of user's guides and computer books, and early attempts to market and sell PC software. Writing a fresh history of personal computing involves significant challenges, in part because the most recent storytelling emphasizes the roles that famous "pioneers" and "founders" have played in narratives about Silicon Valley, the Greater Boston area, and the Pacific Northwest. There has been no shortage of popular books about Apple Computer, Microsoft, Amazon, Google, and Facebook—usually emphasizing the rise of the stereotypical "computer nerds" to positions of wealth and influence in the companies that benefited from personal computing and Internet-based technologies.[16]

It is often difficult to move beyond these perspectives because of a curious lack of sources that document early personal computing and its broader impact on American society. Most of the earliest PC hardware and software companies have merged or gone out of business, leaving little in the way of historical materials to study. IBM is a noteworthy exception to this trend, recently releasing some of its materials to historians of computing.[17] But Apple Computer's corporate records have been carefully edited by their legal teams and are only partially available. Microsoft has also been reluctant to open its corporate archives to scholars and the general public. Beyond the personal narratives of former employees and product enthusiasts, how are historians to study the history of personal computing? What sources can we use to understand how corporate identities were shaped, hardware

16. An example of this work is Walter Isaacson, *The Innovators: How a Group of Hackers, Geniuses, and Geeks Created the Digital Revolution* (New York: Simon and Schuster, 2014). An intriguing new approach is Margaret O'Mara's history of Silicon Valley, which connects the technical and business development of the region to local and national politics. See Margaret O'Mara, *The Code: Silicon Valley and the Remaking of America* (New York: Penguin Press, 2019).

17. See James W. Cortada, *IBM: The Rise and Fall and Reinvention of a Global Icon* (Cambridge, MA: The MIT Press, 2019), especially chapter 14. Cortada was well positioned to write this history because he is a former IBM employee as well as a professional historian.

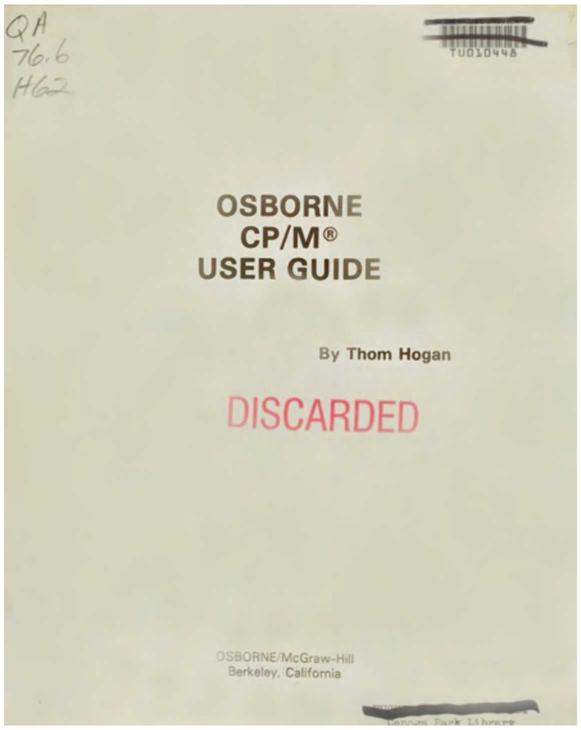
and software products were created, and whether computing initiatives succeeded or failed? Just as important, how did the *users* of PCs experience new products and come to understand their features? Can we assess how regular people accepted, accommodated, or rejected the plans and proposals of industry elites?

Code Nation proposes a publication-centered way of examining the early history of microcomputing and personal computing, from experiments with time-sharing systems, to the mail-order kits of early enthusiasts, to book and magazine publications for platforms like MS-DOS, the Apple Macintosh, Microsoft Windows, and Unix/Xenix. I evaluate the history of personal computing using hundreds of programming primers, textbooks, manuals, magazines, user's guides, and trade show catalogs from the early 1950s to the late 1990s. These neglected sources have allowed me to explore the challenges presented by the first PC systems, the content of computer literacy debates, the methodology of early programming primers, the strategies of successful (and unsuccessful) entrepreneurs and corporations, and the way that computing has impacted the daily life of Americans. To support this analysis, I include technical descriptions of hardware and software systems, code snippets from historic programming languages, the biographies of little-known programmers and entrepreneurs, and a product-based assessment of early hardware and software systems. I also present over 80 historic photographs selected from relevant archives, museums, corporations, and private collections.

I have learned that printed materials related to computers and software—once a common feature of many offices, homes, and schools—have been discarded at an alarming rate. When discussing the issue of "disappearing sources" with a local college librarian, I learned that older computer books and magazines are especially vulnerable to being categorized as *ephemera*, or transitory sources of information about outdated methods or technologies. (See Figure 1.7.) With new computer books and periodicals arriving on a monthly basis, and shrinking budgets, how important is it to maintain an historic collection of FORTRAN, BASIC, and C primers? Especially in locations where shelf space is at a premium? My source's questions are legitimate, of course. But the comment points out how vulnerable technical sources are to abandonment. "Often, they are simply recycled," my informant conceded.

But, if we cannot study issues like computer literacy in the past, how can we hope to evaluate it in the present?

For the purpose of this study, I was able to find many older computer books and periodicals in private collections, as well as the technical libraries of larger public universities. For example, I have spent many weeks in the engineering library at the University of Washington in Seattle, which has a good collection. I also found many books, newsletters, and software packages in the Computer History Museum



QA
76.6
H62

[redacted]

OSBORNE
CP/M®
USER GUIDE

By Thom Hogan

DISCARDED

OSBORNE/McGraw-Hill
Berkeley, California

[redacted]

Figure 1.7 **The title page of Thom Hogan's *Osborne CP/M User Guide*. Published by Osborne/McGraw-Hill in 1981, this book was one of the most important operating system primers of the microcomputer era. Like many older computer publications, however, it has been widely discarded by libraries.** (Photo courtesy of Michael Halvorson)

in Fremont, California. But like the chapbooks and "street literature" of earlier eras, historic computer books and materials can easily be lost if historians are not sensitive to the many treasures that they contain. In particular, they reveal the teaching strategies used to introduce new technical systems, and the opinions and practices of regular people who are learning new technologies. I hope that this publication-centered approach will be of interest to future historians of computing. There are still many fascinating sources that slumber in our nation's technical collections.

I begin *Code Nation* with a comparative analysis that examines computing in the 1960s and 1970s, emphasizing the era's sense of crisis about how software was being created and its multilayered hopes for renewal. My survey presents four overlapping computing mythologies, each representing a different aspect of the period's professional, cultural, and technical traditions. These narratives introduce early advocates for software engineering practices, countercultural idealists who promoted widespread access to tools, creative scholars from the emerging discipline of computer science, and the designers of the first personal computers. In the 1980s and 1990s, American programmers drew on many of these motifs, creating a worldview that bundled hopes, anxieties, and dreams about the new platforms.

Author's Biography

Michael J. Halvorson

Michael J. Halvorson, Ph.D., is Benson Chair of Business and Economic History at Pacific Lutheran University, where he teaches courses on the history of business, computing, and technology. He has written widely on European history, application software, and programming personal computers, including the popular series *Microsoft Visual Basic Step by Step*, Pearson (2013). To learn more about the Code Nation project, visit www.thiscodenation.com.

Index