70
CONCLUSION

10 PRINT has generated far more than a pattern that resembles an unending scrolling maze. It has generated talks, posts, papers, online conversation, demoscene productions, and now this book. But its most important product may be the countless programmers inspired by its concision, enticed by its output, and intrigued by its clever manipulation of two very simple symbols.

While 10 PRINT is a very particular, historically located object of study, it is not completely unique, precious, or rare. Whether or not new Commodore 64 owners realized it, a version of the program was included with every new computer, making it one of the most commonplace pieces of code of the era. There is no evidence to suggest that it was considered the best BASIC program, or even the best one-line BASIC program, for the Commodore 64. Rather, 10 PRINT is emblematic of the creative deluge of BASIC programming in and around the early 1980s. Many programmers at this time were home computer users who, in the years when the personal computer was just emerging as a household technology, seized on programming as a means of play, learning, and expression.

Yet, as this book has indicated, 10 PRINT *resonates*. It is more compelling than many similar Commodore 64 programs, works better than random-maze-generating programs on other platforms did, and can be varied and expanded in interesting and powerful ways. Still, it is only one example of how computers are used to explore computation and to create beautiful artifacts. 10 PRINT was selected as the focus of this book not because the program sits at the summit of all possible one-liners in any language and for any platform, but because the program can lead the way to appreciating code and the contexts in which it emerges, circulates, and operates.

Reading this one-liner also demonstrates that programming is culturally situated just as *computers* are culturally situated, which means that the study of code should be no more ahistorical than the study of any cultural text. When computer programs are written, they are written using keywords that bear remnants of the history of textual and other technologies, and they are written in programming languages with complex pasts and cultural dimensions, and they lie in the intersection of dozens of other social and material practices. Behind the ordinary features of a program—a call to produce random numbers, a printing mechanism, a repeating loop—lie ghostly associations with distant and forgotten forms of cultural activity and production whose voices echo from somewhere inside the labyrinth of

material history accumulated in a particular technology.

Code is not only a conventional semiotic system. At its essence, code also functions. Code runs. Code does something. Code executes on the computer and has operational semantics. But code means things to people as well, both implicitly and explicitly. What this book has done for a single line of code can be done for much larger programs as well, for programs of many other sorts. While other programs and other categories of program have been discussed in this book, the focus on a single short program has been productive rather than restricting. We hope this will encourage the detailed analysis of other short programs and suggest that it is worthwhile to focus on important subroutines, functions, and procedures within larger systems of code.

Looking at each token, each character, of a program is a helpful start, but only a foundation for the understanding of how code works for individuals and in society. It can show not only why a particular program functions the way it does but also what lies behind the computers and programs that are essential to the current world. In considering the `PRINT` keyword and the way it is used in `10 PRINT`, it is possible to see that `PRINT` invokes the CHROUT routine in the Commodore 64's KERNAL, that it provides the ability to append text at the current position (using ";") and to automatically scroll the screen upward when necessary. This particular behavior is a convenience in many cases and contributes to the visual effect of `10 PRINT`. At the same time, `10 PRINT` is a reminder of the history of computer output devices and of BASIC itself being developed on upward-scrolling Teletypes that literally printed.

To understand `10 PRINT`, it helps to identify the program as a one-liner and to note that it produces a seemingly random maze. Yet, a study of the code itself shows much more about BASIC, the Commodore 64, and the program itself than does a high-level categorization and description of function. This is true even though this code does not contain the easiest hooks for traditional interpretation, such as comments or variable names. `10 PRINT` shows that much can be learned about a program without knowing much of anything about its conditions of creation or intended purpose—or indeed, without it even having an intended purpose.

Today, some people who do not mainly identify as "programmers" nevertheless do program computers; they harness the ability of these machines to do provocative work. This is the case with designers who use

Processing, for instance, and with some who work in HTML, CSS, and JavaScript to create interesting programs on the Web. But the widespread access to programming that was provided by early microcomputers does not exist in the same form today as it did in the 1970s and 1980s. When people turn on today's computers, they do not see a "READY" prompt that allows the user to immediately enter a BASIC program.

The science fiction author David Brin wrote a few years ago on Salon.com about the difficulty of getting any form of BASIC running. He reported that he and his son "searched for a simple and straightforward way to get the introductory programming language BASIC to run on either my Mac or my PC," but could find none (Brin 2006). There are BASICs available now, including Microsoft Small Basic, explicitly intended to embrace the spirit of the original language. But in the early twenty-first century, such tools are still somewhat esoteric specialty items, not standard features of every home computer that make themselves available upon startup.

For popular programming, the early 1980s were certainly a special time. Computers were more difficult to use in some ways. The Commodore 64 required its users to issue complex commands to read a disk and run a program from it. But programming was easier. Over the past two decades, academic and industrial research labs have attempted to invent or apply simple programming tools for educational purposes, to teach anyone how to program at a rudimentary level. On the one hand, this book reminds us that a straightforward way for people to program their computers—either in BASIC or another simple language—is indeed possible, since it has already been achieved. But on the other hand, it also accentuates the many significant differences in the way computers are designed and used today compared to the heyday of the Commodore 64, differences that help explain why researchers can't simply recommend that interested parties buy an inexpensive home computer, turn it on, and experiment with it.

Computer programs can be representational; they can depict worldly things and ideas, and they can resonate with related figures, images, and designs. In the case of `10 PRINT`, the program's mazelike output is not a neutral pattern, but one wrapped up in numerous contradictory Western ideas about the notion of a maze. Whether a program's representations are incidental or very deliberate, they have a meaning within culture. The cultural history of the maze demonstrates that there are more and less obvious associations with this type of structure, some wrapped up with the history

of science in the twentieth century and others emerging from computing itself. Although a program's output is only one of its aspects, a reading of code should certainly take into account what a program does and what texts, images, and sounds it produces.

While `10 PRINT` is a text, it exists in the material context of computing. It was printed (in different versions) first in a spiral-bound manual and later in a glossy magazine. It ran on a particular taupe unit, the Commodore 64, the components of which were influenced by economic circumstance and the physical possibilities of chip design and selection. The BASIC programming language in which `10 PRINT` is written was shaped by the sharing of programs in print and in human memory, and by the specific technical aspects of the Altair 8800 and the Dartmouth Time-Sharing System. Our discussion of `10 PRINT` has tried to account for these relevant material qualities while also attending to the formal, computational nature of the code—what it does—and how that interacts with material, historical, and other cultural aspects of the program.

All programs are written in particular settings (a corporate office, a computer clubhouse, a university, a coffeehouse) and are influenced by the means by which they are written. Whenever code is considered, it is worthwhile to investigate how it was written and what material factors came into play as it was transmitted, shared, and elaborated. As with the Teletypes that preceded computers like the Commodore 64 and the laptops that eventually replaced them, the physical makeup, cost, contexts of use, and physical form of computers have significant effects on how they are put to use.

People tend to imagine computer programs as largely static, frozen masses of code. To the extent that this view is valid at all, it makes sense only within a small slice of computing history. It is true, for instance, that the retail market for shrink-wrapped software and the sale of videogames on cartridges tend to support the view that a program is a particular, stable sequence of code and nothing else.

Of course, this era has passed. Software of all sorts, including videogames, is distributed on systems that can and frequently do patch and update programs. Download a mobile phone app or even a Playstation 3 game that is initially free of advertisements and, after running an update, the program can start downloading and displaying ads while it runs. People now think little of modifications of their software, even those that are intrusive and annoying. At the same time, today's operating systems

are easily patched online to prevent security problems and to add new features, bringing benefits to users.

The view of programs as static is even less tenable when one considers the writing, running, and distribution of programs throughout the history of computing. Custom software written for businesses has long been maintained and updated—for half a century. The BASIC programs people keyed in from magazines invited users to modify them. In educational and software development settings programs have typically been converted to other programs by elaboration and modification.

`10 PRINT` is not just a line of code; it defines a space of possible variations (some of which were explored in the remark Variations in BASIC), possible ports (see the remark Ports to Other Platforms and other ports throughout the book), and possible elaborations (such as the one described in the remark Maze Walker in BASIC). `10 PRINT` can simply be run, but it can also be considered as an instant in the process of programming, a process that can lead to a better understanding of and relationship with computation, in addition to leading to other aesthetically interesting and differently functioning programs. This book has tried to establish `10 PRINT` not just as a program, but also as part of the process of learning about and developing programs—something that can be said about almost any code.

Since programs are dynamic, and some of them explicitly invite modification, and since modifying programs is a way to better understand them, the platform, and computing generally, why not modify a program as part of a scholarly investigation of the program? This is one approach taken in this book. The variations, ports, and elaborations in this volume set some of the qualities of the canonical `10 PRINT` into relief in an interesting and informative way.

To see what is special about different platforms, and how platforms differ from one another, we have produced ports of `10 PRINT` during our investigation of it and the writing of this book. Porting a specific program makes for a very different and more revealing comparison than does simply lining up the technical specs of the two systems for side-by-side comparison. It shows what specific qualities of a platform are important for particular effects and for the functioning of particular programs. Similarly, developing variations allows programmers to explore the space of possibility within a platform. In all of these cases, programming is not a dry technical

 `10 PRINT CHR$(205.5+RND(1)); : GOTO 10`

exercise but an exploration of aesthetic, material, and formal qualities.

Whether one is studying a videogame, some other aesthetic object, or code that runs a voting machine or defines a climate change model, writing programs can help us comprehend the code and its cultural relevance. In the case of large systems, it could be unwieldy to re-implement and modify the entire program. This approach, however, is being tried with the story generator MINSTREL (Tearse, Mateas, and Wardrip-Fruin 2010), explicitly for the purpose of better understanding that influential system and how it tells stories. It is possible to reimplement and modify important routines, functions, and procedures, to grasp more firmly what a program does as opposed to what it could have been written to do. Analyzing the code by interacting with it, revising it, and porting it is one of the main critical activities this book contributes to critical code studies and related fields in digital media.

Early on, a variant of `10 PRINT` was presented to novice readers to hint at the tremendous potential of the computer using a simple but elegant technique. This book is meant to serve a similar function. Through its many approaches to a one-line program, the book is meant to unlock the potential for analyzing digital objects and culture through code.