45
REM ONE-LINERS

One-liners, as single-line programs such as `10 PRINT` are known, predate home computing, the exchange of BASIC code in magazines, and even the BASIC programming language itself. These concise little programs were written at least as early as the beginning of the 1960s. The language that was most famous for writing such programs was APL, designed by Kenneth Iverson using special (non-ASCII) notation. APL was first described in his 1957 book *A Programming Language* and was first implemented at IBM beginning in 1960.

In APL there is no limit on the length of a line; anything that a programmer can express as a single statement counts. A report shows that "all 'practically' computable functions" can be written as APL one-liners (Lipton and Snyder 1977, 2), so perhaps one-liners in this language should not be particularly impressive. Programmers have nevertheless been impressed by them. APL one-liners have been published that solve the problem of placing N queens on an N × N chessboard (Selfridge 1977, 243) and that completely encode John Conway's Game of Life (McDonnell 1988, 6). The final Game of Life APL function presented is only nine tokens long. While not everyone involved with computing shares an enthusiasm for one-liners, or for APL, the exchange and academic publication of APL one-liners does demonstrate that interest in this form of program was not limited to amateurs or newcomers to computing.

In the early 1980s, magazines published one-line programs, sometimes regularly, to fascinate and intrigue home computer users and to help them explore programming. In the Commodore-specific magazine *RUN*, they appeared in a section near the front of the magazine entitled "Magic," which contained "Hints and tricks that will let you perform computing wizardry." Some of their one-liners and tips were clearly for amusement and educational purposes. Others were practical programming aids. Many were quite expressive and produced interesting visual effects.

Here is the first trick, numbered zero in hexadecimal, in the very first "Magic" section from the inaugural issue of *RUN*:

> **Trick $00.** This month's "one line special" is an antiquity—from the far-off days of 1978, when an 8K Commodore PET cost $795, and readable documentation was unheard of. There weren't any books, and the only magazines were newsletters produced by amateurs.
>
> *The PET Gazette* was one of them, and here is one of its early

```
{148}  10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

offerings, called "BURROW":

```
1A$="[up][down][left][right]":PRINTMID$(A$,RND(.5)*4+1,1)"
*[left]";:FORI=1TO30:NEXT:PRINT"[rvs on][space][left]";:GOTO1
```

It fits on one 40-column line, and it *does* get exciting. We'd like to see *your* one-line programs, and we want to print at least one good one each month. Programs can be fun, funny, useful or useless, as long as they fit in 40 columns or less. What do you have? (Sander 1984)

The program featured here (see figure 45.1) moves the cursor randomly either up, down, left, or right, prints an asterisk, moves left over it, turns on reverse video, prints a space, and moves left over that—and then repeats. This means that it will "dig" a reverse-video hole (green, not black, on a PET computer) haphazardly, although orthogonally, from its starting point to wherever it ends up around the screen. Its mazelike path involves both regularity (each move is directly along an axis) and randomness (which of the four directions it moves in is chosen at random), producing the promised excitement. The program has some affinity with `10 PRINT`, although `10 PRINT` creates a different sort of scrolling pattern and suggests a structure rather than traversing the screen a character at a time.

This printing of the "BURROW" program, already declared an antique, also shows an awareness of computing history and a willingness to rediscover older programs so they can be enjoyed by a new generation of programmers and users.

Here is another intriguing one-liner from *RUN* (Rapp 1985):

When he's not looking, run this on a friend's VIC or C-64. Then get him to type a line or two, and watch the fun as he scrambles for his warranty.

```
10 POKE207,0:POKE204,0:WAIT198,1:GETA$: PRINT"{CTRL RVS OFF}"
CHR$(ASC(A$)+1.1*RND(0));:GOTO 10
```

This program is similar to `10 PRINT` in a few ways. It runs in an infinite loop; it also makes use of the `RND` function. These are true of "BURROW" as well. An additional similarity between this April Fool's program and `10 PRINT` is the use of `CHR$`. There is a significant difference, too. Rapp's program doesn't do anything obvious when run. After running it, the cursor sits blinking as if one were in the BASIC interpreter. Once run, this one-liner is actually in control of keyboard input and screen output and effectively

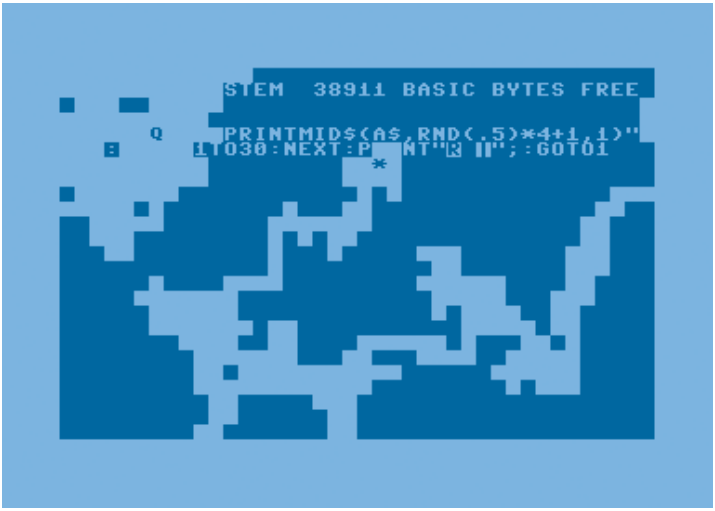

Figure 45.1
Screen capture from one-line "BURROW" program.

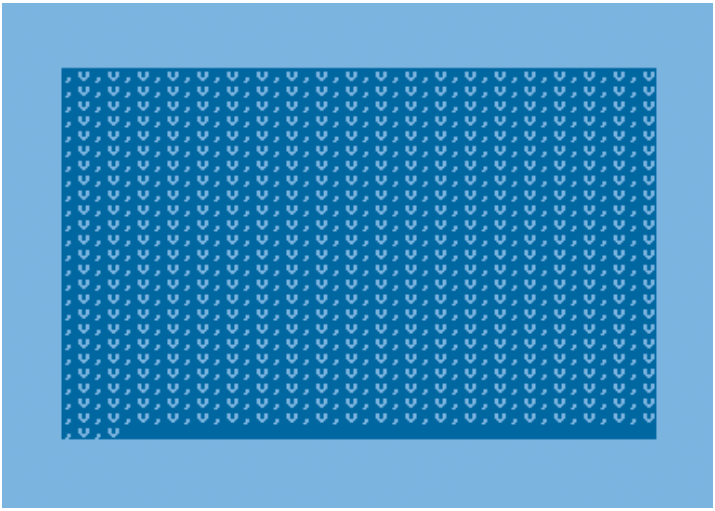

Figure 45.2
Screen capture from one-line program featured in *RUN* magazine to check monitor resolution.

```
(150)  10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

intercepts input from BASIC and the Commodore 64's operating system, the KERNAL, running atop them. Typing something will (often) cause the typed characters to appear on the screen as they usually would, but about one time in eleven, the next character in the PETSCII sequence will appear instead, possibly transforming the user's typed `10 PRINT "HELLO"` to the puzzling and frustrating `10 PRJNT "HFLLO"`.

`10 PRINT` uses `RND` differently, to draw from an even distribution of two characters. As the *Commodore 64 User's Manual* explains, this distribution can be skewed, as it is in the first variant of the program presented in the first remark, written in BASIC. Even with a somewhat unbalanced distribution, the larger impression is still mazelike. The "essential frustration" of the maze, on the one hand, is one that is evident and stems from its interlocking, larger structure to which the randomness contributes. Rapp's prank is tricky, on the other hand, because it is biased toward intermittent unpredictability and operates invisibly.

By the late 1980s, although the "Magic" section continued in *RUN* (through the magazine's last issue), it was handed off to other editors and filled with utility programs, in BASIC and assembly, that were significantly longer—often around twenty lines. This late one-liner from *RUN* (Hubbard 1987) offers help for the Commodore user looking for a new monitor and shows the utilitarian turn that programs took in later years (see figure 45.2):

```
10 PRINT CHR$(14):FOR A=1TO40*23:PRINT",V";:NEXT
```

Enter the program and run it. The screen will fill with 23 lines of commas and lowercase V's. To check the resolution, look at the single pixel that forms the point of the center of the v or the tail of the comma. On a monochrome monitor the pixels should be a single round point of light.

Some one-line utilities were compatible across BASIC machines such as the Commodore, Apple, and Atari home computers and might also run on the original Dartmouth BASIC—but many of the fun and exciting ones were specific to particular platforms. The numerous versions of BASIC included some which included commands such as `PLOT` and `SOUND` to facilitate making graphics and music. Of course, a one-liner in a BASIC of this variety could take advantage of these special commands. In these cases, one-liners were often teaching tools: programs that helpfully introduced

commands needed to perform higher-level tasks.

This practice continues in the contemporary practice of programming tutorials; one example is Peteris Krumins's "Perl One-Liners Explained" (Krumins 2009–2011). It introduces more than a hundred single-line pieces of code such as:

```
perl -MPOSIX -le'@now = localtime; $now[0] -= 7;
$now[4] -= 14; $now[7] -= 9; print scalar localtime
mktime @now'
```

Each of Krumins's examples includes a description—often a somewhat mysterious one, such as this program's: "Print date 14 months, 9 days and 7 seconds ago." The first question a non-coder might consider is "Why that? What was 14 months ago?" This sort of arbitrary program construction is not valuable as a utility in its given form. Rather, it is useful to try out because of what the programmer can accomplish by daring to change it and by inserting the code into a more complex program. That code snippet could be useful, for example, in a vacation scheduling or beer fermentation system. It resembles `10 PRINT` in that it unlocks the workings of higher-level functions (such as `localtime` and `mktime`). `10 PRINT` arouses interest not only from its visually active display with minimum code but because that code reveals an elegant means of accessing the higher-level video terminal system, which is an entry point vital to writing a diverse area of types of programs.

Many programmers of one-liners took advantage of the BASIC commands for high- and low-resolution graphics on computers contemporaneous with the Commodore 64, such as the Apple II. Apple II users enjoyed a rich culture of one-line graphics display programming as well as a tradition of "two-step" programs, which consisted of two lines instead of one. Apple II users also benefited from having a longer maximum line size than on the Commodore.

The two-line format came about because often one line was reserved to initialize the graphics display hardware and other variables. It did not and should not have run multiple times. The second line was a loop that, like `10 PRINT`, produced animated graphical output. In `10 PRINT`, an initialization line was unnecessary, because the default state of the Commodore 64 was a pseudo-graphics terminal: at power-up not only was the

computer in a state to immediately begin accepting and executing BASIC commands, it could also draw graphics characters from a set which was printed on the keyboard. Computers, such as those in the Atari and Apple series, had BASIC multimedia commands (`COLOR`, `GR`, `PLOT`, `HLIN`, `PSET`, to name a few) to access their platform hardware, and could be said to have led to more impressive one-liners that were not possible on the Commodore computers—something that only increased the value of the most impressive Commodore one-liners, including `10 PRINT`.

By way of example, consider the one-liner "Icicle Storm," developed for this book to demonstrate how the use of one-liners can communicate valuable details about a computer system. The program generates a simple multimedia display that looks like the sky filling with icicles, drawn using one of the diagonal graphics characters used in 10 PRINT (see figure 45.3):

```
10 POKE 1024+RND(1)*1000,78: GOTO 10
```

Although it is a minimal simulation, the code highlights several useful details about the Commodore 64 platform. First, the repeated calls to `POKE` the distribution `1024+RND(1)*1000` indicate setting values in a section of memory. This is the "direct route" or memory-mapped access to the text/graphics terminal on the platform. To experienced programmers of other computers, this one-liner communicates "This computer has a screen memory just like many others. This particular one begins at 1024 and is 1000 bytes long." The transfer of knowledge from platform to platform is a key part of the practice of programming; another key part is learning the differences among platforms. Sometimes knowing just a few details about a new system enables one to leverage a great deal of previous experience into competency of the new system.

While such addresses were not secret—they could be obtained simply by buying the *Commodore 64 Programmer's Reference Guide* that Commodore published (1982)—they held a certain value when printed material about programming was still sparse, in the early days of home computing. While commercial software empowered users within the realm of their applications, short programs in books and magazines illustrated how to make the computer do impressive things and empowered readers to program. They associated brief BASIC texts with sufficiently compelling title and graphical output or other effects to allow one to build up a catalogue of

Figure 45.3
Screen capture from the "Icicle Storm" one-liner. Characters are drawn at random positions on screen one at a time.

appropriately useful code segments. Thus "Icicle Storm," like `10 PRINT`, is not an effort to tell a story about weather. It is a cartoon that presents the physics of the virtual world it runs in, the text/graphics terminal, through the speed of the screen update and the properties of its regular grid.

The slow, ruthless instantiation of icicles mimics the dynamics of a mounting storm because the computer cannot draw them fast enough to fill the screen instantly. The pace of that experience is CPU-limited. It would be possible to *slow down* the drawing, but not to speed it up without resorting to something other than changes in BASIC code. Understanding the general pace or speed at which a platform executes code is useful information to programmers.

The kinetic movement of the storm, determined by the update rate of the screen, fulfills the purpose of the illusion sufficiently that the impression is uniquely identifiable and memorable. It thus invites and aids the programmer to remember the useful numbers in the program. As part of the cartoon illusion that the program conjures, the evoked scene also assumes the default foreground and background colors of the computer, producing blue ice crystals against an azure sky. This may even be a more appropriate

```
(154)  10 PRINT CHR$(205.5+RND(1)); : GOTO 10
```

and specific play on the default colors than `10 PRINT` provides.

A similar one-line program developed for this book, "Windy Day in Chicago," illustrates another feature of the VIC-II that is useful to programmers, smooth horizontal scrolling:

```
10 POKE 53248+22,INT(192+4+3*SIN((TIME*3.456+RND(1)
*.5))): GOTO 10
```

The program doesn't change any of the characters or colors on the screen; it simply causes everything on the screen to move back and forth semi-regularly as if the display were being blown around. This program demonstrates the relative simplicity of working with the side-scrolling register in the video chip, an advanced topic which is never taught explicitly in the Commodore 64 manuals.

Finally, another way to go about probing the capabilities of complex chips, including the Commodore 64's sound chip, the SID, is to simply write random values to their registers and attend to the result. Here is such a program for the SID, one which produces random sounds:

```
10 POKE 54272 + (0*TI+(RND(1)*25)),(RND(1)*256)
AND255:GOTO 10
```

Computers no longer power on to the READY prompt and the BASIC programming language, but, as the discussion of Perl one-liners shows, short, impressive, inviting programs live on in other languages and environments. There is more on the way that some Perl one-liners are apprehended and remembered in the next chapter, in the section "BASIC in Human Memory."