by Jeffreys Copeland and Haemer

MICHAEL AVETO

"Those who cannot remember the past are condemned to repeat it."
– George Santayana

"Those who do not understand UNIX are condemned to reinvent it, poorly."
– Henry Spencer

Software Ptools

This month, we take you to Hershey Heaven. In the flurry of irrelevancy with which the press inundates Nobel Prize winners, Alfred D. Hershey, the 1969 Nobel Laureate in Physiology or Medicine, was asked what he thought heaven would be like. He answered that in heaven, he thought he'd finally get an experiment that worked–and be able to do it over and over again.

In 1976, Brian W. Kernighan and P.J. Plauger wrote *Software Tools* (published by Addison-Wesley Publishing Co., ISBN 0-201-03669-X), a book we recommend to everyone. Here's a brief synopsis supplied by coauthor Peter Plauger:

Describes a number of small programs made popular by the UNIX operating system. Contains complete source code of all the programs in Ratfor, a structured dialect of FORTRAN that strongly resembles C. This classic pioneered the term 'software tools.'

This book, probably the clearest exposition of the UNIX tools philosophy, provides the source code for complete implementations of several UNIX tools, together with running commentary on every point of their design and implementation.

But why in Ratfor–something that "strongly resembles C"–and not C itself? The first edition of Kernighan and Ritchie's *The C Programming Language*, which introduced the world to C, was published in 1978. But in 1976, the two universally available programming languages were COBOL and FORTRAN-66 (often as the implementation FORTRAN IV). Kernighan and Plauger had a message to get out about how to program, to an audience who had never heard of UNIX or C, and who–at least the way it looked then–never would.

As a transition vehicle, they created a language that looked like C, but could be preprocessed into FORTRAN-66. (Most of you have probably never seen FORTRAN-66, but it lacks nearly everything you take for granted in programming languages: if-then-else, data structures, while loops, character I/O and even strings. The primary control-flow structures are the "logical if" and the "goto.")

Here's an example of relatively easy-to-read FORTRAN-66, taken from the original Bell Labs Ratfor documentation:

```
IF (X.LE.100) GOTO 10
   CALL ERROR(5HX>100)
   ERR = 1
   RETURN
10  ...
```

The equivalent in Ratfor?

```
if (X>100) {
   call error("x>100"); err = 1;
   return }
```

The idea was to use a language that would permit easy-to-read examples in a book that showed folks how to write code that improved their programming

environments. To close the loop, the final chapter of *Software Tools* designs and implements an entire Ratfor-to-FORTRAN preprocessor.

The book quickly spurred the formation of the Software Tools User Group (STUG). Formed at Lawrence Berkeley Labs, this group began distributing tapes that contained all of the code from the book, together with an ever-growing body of contributed tools, all of which could be installed on any computer with a FORTRAN compiler (which meant, at the time, pretty much any computer)–a UNIX-like environment, not just for non-UNIX systems, but for a world that had never heard of UNIX or C.

If you'd like to see what sorts of things were done, you can still find references to *Software Tools* on the Web. One such site is `http://www.geocities.com/SiliconValley/Lab/9247/#compilers`. An advantage of the UNIX "one tool, one job" philosophy is that you can attack each command separately, one at a time. Most are bite-size. Individual contributors can write a useful tool in a few days (or less) and make a real contribution to the larger whole. Yoked to this is the idea that when you write a little program, instead of a giant, monolithic one, you can really make it yours; you can get your arms around it and put in the work you need to get it just right.

The reason we can recommend a 25-year-old book, full of code for programs you'll never need to write, in a language you'll never use, is that it remains the clearest, best written, most entertaining and practical treatise we know on how to get programs just right. (Do not, by the way, be fooled into buying *Software Tools in Pascal*, written by the same authors and published in 1981. Kernighan's 1981 technical report, entitled, "Why Pascal Is Not My Favorite Programming Language," `http://cm.bell-labs.com/cm/cs/cstr/100.ps.gz`, gives great insights into both why this book didn't turn out the way it could have and why Pascal, once a hot contender for the programming language of choice, eventually lost out to C.)

It's hard to imagine, nowadays, just how revolutionary this book's approach was. It changed lives. Most folks reading this column have probably never even seen a punched card or written a FORTRAN program. To help put things in context, imagine working as a programmer in a world in which neither you nor anyone you know has ever heard of a filter or a "software tool," and the only tools available to you as a programmer are a compiler, an assembler and a linker. (And a world where nothing is off-the-shelf. We know someone who began his career writing a payroll system in FORTRAN for a movie studio.)

One chapter of *Software Tools* contains the complete design and implementation of an editor. Not a screen editor, mind you. After all, no one had cursor-addressable terminals back then.

What ever happened to STUG? One finds occasional fossils of STUG, such as Usenix's Software Tools User Group Award, `http://www.usenix.org/directory/stug.html`, but the group died of its own success. People who joined STUG learned about UNIX, helped popularize UNIX and the UNIX philosophy, eventually demanded UNIX and switched to UNIX when it became available.

In the mid-1980s, the UNIX tool set story was replayed more than once, to the great advantage of a new generation of computer users. Mortice Kern Systems (MKS) Inc., a Canadian company, rewrote the entire basic UNIX command set from scratch for MS-DOS, and later ported the same suite to various legacy systems.

In the same time frame, the Free Software Foundation (FSF) coordinated the contribution of an army of volunteers, who created freely redistributable versions of nearly all common UNIX tools, which eventually made up the bulk of the command-line utilities for Linux.

A third great source of rewritten UNIX tools are the BSD releases, coordinated by the University of California at Berkeley's Computer Science Research Group (CSRG), and found in a wide variety of freely available BSD-based UNIXes. As with STUG and the FSF, CSRG's work was the coordinated effort of an unruly army of individual volunteers.

Everyone eventually caught on to the software toolbox approach.

Tom Christiansen Becomes Irked

Well, everyone except the Windows world.

Those of you who read the `comp.lang.perl.misc` newsgroup know that Tom Christiansen, coauthor of many O'Reilly & Associates Inc. Perl books, is a frequent contributor. When a question irks Tom, he speaks right up, often chastising the questioner. Some people don't like this, so Tom is occasionally a source of discussion on the newsgroup in his own right.

Like grains of sand in an oyster, though, these irritants sometimes spur Tom to create something beautiful. (In the distance, we hear groaning; to quote Jo Haemer, "A cheap shot is a terrible thing to waste.")

Spurred on by irksome questions, Tom has written Perl man pages, FAQs, tools and even a series of Perl FMTYEWTK (Far More Than You Ever Wanted To Know) essays.

Several months ago, Tom was going through a stretch of irritation at people asking for complete Perl solutions to problems that could be solved with simple calls to basic UNIX utilities. Too often, his pointing this out didn't help the requester, because they were running some Microsoft Corp. platform that didn't have the basic utilities to solve the problem.

For a while, Tom's reaction was to declare that such lacunae were God's wrath visited upon anyone sinful enough to run something other than UNIX, and that we in the UNIX community had no obligation to help.

Those of you who've been reading the newsgroup, or this column, for some time will even remember a parody posting, by Nat Torkington–Tom's coauthor for *The Perl Cookbook* (published by O'Reilly, 1998, ISBN 1-56592-243-3).

```
Tim.Bunce@ig.co.uk (Tim Bunce) writes:
> The problem is to find the full list of
> names and the original order.

You INSTALL a FULL SET OF TOOLS, like THE LORD GOD
ALMIGHTY intended. REPENT, ye PRISONER of BILL!
The DAY of JUDGEMENT is AT PERL! Your MESSENGERS
are obviously just POOR substitutes for RELIABLE
PIPE COMMUNICATION which you'd have if you had a
REAL OPERATING SYSTEM and not a SCURRILOUS PIECE
OF TOOL-CHALLENGED COPROPHILIA!

Tom^WNat
:-)
```

This ultimately led us to write a Perl version of `tsort(1)` (see `http://sw.expert.com/C9/SE.C9.SEP.98.pdf` and `http://sw.expert.com/C9/SE.C9.OCT.98.pdf`).

More recently, though, Tom seems to have decided that the problem isn't going to go away, and has organized a project to rewrite all the basic UNIX utilities in Perl, so that any system with Perl can have the full, basic UNIX command set for free.

Note the word "organized." Tom has written some utilities himself but what he's really doing is coordinating contributions from all over the Perl world. Tom calls it the "Perl Power Tools" project. We prefer "Software Ptools": the "P" is psilent.

An Example: asa(1)

This looked like enormous fun, and we jumped in with both feet. We have both worked in the printer industry, so we decided to chip in by contributing a traditional UNIX utility that no one else would be silly enough to write: `asa(1)`, a program that interprets traditional FORTRAN carriage-control commands.

Here, a little history will help. A couple of decades ago, printers were all impact line printers that produced great stacks of accordion-folded, green-and-white-lined,14-inch-wide paper, which was actually 15-inches wide if you counted the tear-off strips on the sides that were perforated so teeth on the printer carriage could advance the paper.

These printers were simple; most couldn't even do graphics (graphics output was provided by another kind of printing device, called a plotter). Indeed, besides printing alphabetic characters, almost all they could do was to move to the top of a new page, backspace (in order to underline) and overprint lines (to produce bold characters).

But at the time, not even character sets were portable (non-IBM Corp. machines often used ASCII, but IBM machines, which were in the majority, used EBCDIC), which meant your programs couldn't assume that backspace was a ^H, form-feed was a ^F or carriage return was a ^M.

A convention was born: all printers agreed to look at the first character of each output line and interpret a small number of special characters as special carriage-control commands. For example, a "1" in the first column of an output line told the printer to eject the current page and move to the top of a new page. These conventions were made a part of the American Standards Association (ASA) FORTRAN standard. (You read that correctly: printer carriage controls were part of a programming language standard.) ASA was later renamed ANSI.

In the C/UNIX world, there are no such conventions. Moreover, neither contemporary terminals nor newer printers, such as laser printers, interpret output in this way. Old FORTRAN programs, ported from other operating systems, began finding themselves assuming these conventions on systems that didn't recognize them.

To handle this, early UNIX systems included a program called `asa(1)`, which translated FORTRAN carriage controls. Listing 1 shows our implementation of `asa(1)`.

And now for our dramatic reading: The meat of the program lies in 10 lines of code, lines 10 through 20. Everything else is professionalism.

Listing 1. asa(1)

```
1   #!/usr/local/bin/perl -w
2   # $ID: asa,v 1.1 1999/05/31 22:03:15 jsh Exp jsh $

3   use strict;

4   exit 1 if grep {!-r} @ARGV;   # traditional

5   if (grep /-/, @ARGV)  {
6    $0 =~ s(.*/)();
7    warn "usage: $0 [filename ...]\n";
8    exit 2;   # traditional
9   }

10  while (<>)  {
11   chomp;
12   s/^$/ /;
13   s/^[^10+-]/\n/;
14   s/^1/\f/;
15   s/^\+/\r/;
16   s/^0/\n\n/;
17   s/^-/\n\n\n/;
18   print
```

Continued on Page 42

```
19     or exit 1;   # traditional
20  }

21  =head1 NAME

22  asa - interpret ASA/FORTRAN carriage-controls

23  =head1 SYNOPSIS

24  asa [I<filename> ...]

25  =head1 DESCRIPTION

26  =over 2

27  Traditional FORTRAN programs put carriage-control characters
28  in the first columns of their output,
29  which were interpreted by older line printers
30  according to the ASA vertical format control standard.
31  (ASA was the American Standards Association -- now ANSI.)

32  Under this standard, the first character of each printable record (line)
33  determines vertical spacing, as follows:

34  =over 2
35  I<blank>    carriage return
36  0           two carriage returns
37  1           formfeed
38  +           overprint
39  -           three carriage returns (IBM extension)

40  =back

41  All other characters are discarded, and empty lines behave as though
42  they have a leading blank.

43  B<asa> interprets these characters.

44  =back

45  =head1 EXIT VALUES

46   =over 2

47  0 normal exit

48  1 inability to write on stdout or to read an input file

49  2 bad argument

50  Exit status values chosen from MKS toolkit.

51  =back

52  =head1 AUTHOR

53  Jeffrey S. Haemer

54  =head1 BUGS

55  Currently, B<asa> just looks at the readability of its input files
56  at startup time. It should really do it a file at a time,
57  but that makes the code look gross.

58  The carriage-control '-' is an IBM extension.
59  Perhaps the default should ignore it
60  and there should be a '-i' option to interpret it.

61  =head1 SEE ALSO
62  I<Communications of the ACM>, Vol 7, No. 10,
63  p. 606, October 1964.

64  NWG/RFC 189, Appendix C

65 =cut
```

Lines 1 through 3 are our usual boilerplate. The shebang line (line 1) invokes the Perl interpreter and gives it the `-w` flag, which queries various questionable usages. The third line requires the still more picky `strict` pragma. As long as we're going to write a utility, we might as well catch as many silly errors as we can. The second line says we're keeping our code under revision control.

Lines 4 through 9 do argument parsing. The comment "traditional" means that it's traditional for this command to exit with an exit status of "2" if the arguments are misspecified.

Lines 21 through 65 are documentation. Perl lets you keep your documentation in the same file as your code, so they don't get out of sync.

The meat of the program is the loop begun on line 10 and finished on line 20, which reads and prints the file one line at a time. Carriage control is specified entirely by the first character in the line, so line 11 begins by removing any special ASCII carriage controls at the ends of lines. The printer will never see them. The standard says that a line beginning with anything except one of the special ASA carriage-control characters should trigger a new line and a carriage return, and lines 12 and 13 give us that. Line 12 prints blank lines as blank lines. Line 13 consumes any other character that begins a line and performs the default action: terminating the preceding line. (Yes, that's right. If you want a character at the beginning of a line, you have to precede it with something else. The first character is always interpreted as carriage control.)

Lines 14 through 17 interpret the ASA, beginning-of-line carriage-control codes:

`1`	Form-feed
`+`	One carriage return (for overprinting)
`0`	Two carriage returns/line-feeds (for double-spaced lines)
`-`	Three carriage returns/line-feeds (for triple-spaced lines)

There it is. A 10-line program. Just an oddly shaped brick in the ziggurat of free UNIX tools for the non-UNIX world.

Want to chip in? It's fun. Go to `http://language.perl.com/ppt` and take a look at what's done and what's not. Make the world a better place by spending a few hours in Hershey Heaven.

Until next time, happy trails.

Jeffrey Copeland *(*`copeland@alumni.caltech.edu`*) lives in Boulder, CO, and works at Softway Systems Inc. on UNIX internationalization. He spends his spare time rearing children, raising cats and being a thorn in the side of his local school board.*

Jeffrey S. Haemer *(*`jsh@usenix.org`*) works at QMS Inc. in Boulder, CO, building laser printer firmware. Before he worked for QMS, he operated his own consulting firm and did a lot of other things, like everyone else in the software industry.*

Note: The software from this and past Work columns is available at `http://alumni.caltech.edu/~copeland/work` *or alternately at* `ftp://ftp.expert.com/pub/Work`.