BEST OF THE PERL JOURNAL

Games, Diversions & Perl Culture

O'REILLY®

Edited by Jon Orwant

BEST OF THE PERL JOURNAL

Games, Diversions, and Perl Culture

Related titles from O'Reilly

The Best of the Perl Journal Series

Volume 1: Computer Science and Perl Programming

Volume 2: Web, Graphics, and Perl/Tk

Volume 3: Games, Diversions, and Perl Culture

Other Perl titles

Advanced Perl Programming

Beginning Perl for Bioinformatics

CGI Programming with Perl

Embedding Perl in HTML with Mason

Learning Perl on Win32 Systems

Learning Perl

Mastering Algorithms with Perl

Mastering Perl/Tk

Mastering Regular Expressions

Perl & LWP

Perl & XML

Perl Cookbook

Perl for System Administration

Perl for Web Site Management

Perl Graphics Programming

Perl in a Nutshell

Perl Pocket Reference

Perl/Tk Pocket Reference

Practical mod_perl

Programming Perl, 3rd Edition

Programming the Perl DBI

Programming Web Services with Perl

Regular Expression Pocket Reference

Writing Apache Modules with Perl and C

Also available

The Perl CD Bookshelf

BEST OF THE PERL JOURNAL

Games, Diversions, and Perl Culture

Edited by Jon Orwant

O'REILLY®
Beijing · Cambridge · Farnham · Köln · Paris · Sebastopol · Taipei · Tokyo

CHAPTER 43

The Zeroth Annual Obfuscated Perl Contest

Felix S. Gallo

Are you renowned for your excessive, belligerent tersity? Are you a misunderstood genius who writes Perl code that looks like it's been run through MD5? Or are you just plain evil and ornery enough to write code that's so grotesque that others pale in fear?

At last, you have a socially acceptable creative outlet: The Zeroth Annual Obfuscated Perl Contest! You are invited to participate in a contest to determine who can write the most incomprehensible, unreadable, confusing, horrific, amusing, and interesting Perl code.

There are four categories. You can enter as many as you like, but may only submit two pieces of code per category.

The Categories

Our inaugural contest has four categories:

Best Four-Line Signature
: This award is for the best piece of Perl code that fits into 4 lines of 76 characters of ASCII code (not counting end-of-line newlines).

Most Powerful
: This award goes to the piece of Perl code that does the most with the least. The limit on bytecount is 1024 characters, not including whitespace.

Most Creative
: This award goes to the most stunningly intriguing or ridiculously hilarious combination of obfuscation and functionality. The limit is 2048 bytes of Perl code, not including whitespace.

Best "The Perl Journal"
: In the fine "Just another Perl hacker" tradition, this award is given to the best code that generates the words `the perl journal`. Case and context are not important. The limit is 2048 bytes of Perl code, not including whitespace.

In addition to these four categories, the judges will award one applicant the coveted Best of Show award, a certificate suitable for framing or, imaginably, hiding.

How It Works

The judging will work in three phases.

In the first phase, the judges will examine the code carefully without running it in order to qualify its aesthetics. Any code that's completely understandable at this point will probably not win.

In the second phase, the judges will run the code, examine the output, and then look at the code again in light of its output.

In the third phase, the judges will dissect the code with filters, debuggers, and whatever else they can think of, attempting to determine how it works. Any code that's still incomprehensible at this point will probably win.

If the judges are stumped by the end of the third phase, they'll turn to the `SOLUTIONS` file you've helpfully included in your distribution and attempt to use that to reverse-engineer the code.

Hints and Suggestions

Judging obfuscation and what's "cool," "neat," or "best" is a subjective process. However, here are some general guidelines that might help you design your entry.

Overuse of one particular obfuscation method risks being tedious. Entries that demonstrate breadth, range, and knowledge are likely to beat entries that rely on repeated parlor tricks.

Being clever and humorous is good. As an example, a past winner in the Obfuscated C contest (our pale, weaker cousin) formatted his code in the shape of a maze; the program read its own source code in and implemented an ASCII 3D maze walking program.

Being surprising and deceitful is very good. Bonus points are awarded for obfuscated code that is not only syntactically obfuscated, but semantically obfuscated—code that appears to do one thing but does another is deemed extremely devilish.

Being poetic is also very good.

Entries needn't contain a `#!/usr/bin/perl` (or equivalent), unless they use nonobvious command-line switches. Neither the `#!/usr/bin/perl` nor the command-line switches will count toward the character limits.

Programs that purposefully crash machines or cause system problems tend to be unamusing, so please consider saving them for the Perl System Destroyers' contest.

Here's a tiny example that would not win in any of the categories, but which demonstrates a couple of now well-known obfuscation techniques. It's in the public domain, for what that's worth.

```
@^T = qw, 3( 2 9 6) ;
21_PENGUINS, and print map {(lc((split//=>\"\"=>")[$^T>1])..__)[$_]}@^T
```

Example analysis and judging thought process:

`@^T` is a small red herring; although it looks like some sort of Perl special variable, it's not (that would be `$^T`). Not particularly clever, but at least standardly so.

`qw` uses comma as a quoting character, so the fake parentheses, semicolon, newline, and trailing `_PENGUINS` are an attempt to hide a numeric array composed of (`3, 2, 9, 6, 21`). The judges would instantly run pattern analysis software on this sequence, so this is not well hidden enough. Fairly clever, though.

The idea of starting a line with:

```
21_PENGUINS, and print map
```

is poetic, so that's worth something.

`split//=>` is famous by now, having been used on the comp.lang.perl.misc newsgroup, so this is mildly derivative and obvious.

The use of a scalar reference as a string in `\"\"=>>"` is fairly well hidden behind a wall of semantic misdirection. We'll call this clever, especially once we find out that we're using the letter `c` from that reference to prime an alphabet generator.

The use of `$^T` is humorous and misdirectional considering the use of `@^T`.

The semantics of `..__` are interesting, neat, and somewhat unexpected. Bonus points for the fact that `__` appears special and then unspecial but turns out to be special.

Disappointingly, the output is minimal, uninspiring, and extremely obvious. The program doesn't do anything remarkable or surprising.

So There You Have It

Please submit early and often, and encourage anyone you know who might be interested to do the same.

Although Perl has taken some hard knocks as being a write-only language (and this contest could be construed as an unabashed celebration of that fact), the intent of this contest is to demonstrate Perl's tersity and power, while at the same time giving the creative and demented minds of the Internet's legions a fun, intriguing playground. So have fun!

We look forward with some trepidation to seeing your code!

Results

First, I'd like to say that this contest was an amazing and therefore highly regrettable success. Yes, some members of the elite judging team are now trying to fit square pegs into round holes in a mental rehabilitation ward, and one of the judges was last seen washing his hands obsessively and muttering something about never being able to get clean again, but they gave themselves heroically in the line of duty.

The people to really *worry* about are the various entrants. These seriously warped individuals went far beyond what we thought we'd get with our pleasant little contest. No, they each created code so vicious, so grotesque, that if the U.S. State Department were to find out about obfuscated Perl code of this caliber, they'd immediately declare it an unexportable munition. Actually, some theorize this has already happened. Onwards.

In the spirit of true obfuscation, upon receiving the complete set of entries, we decided to modify the rules of the contest. Every entry was considered for every category—and in fact, some did well outside their chosen category.

Besides the best medicine for nausea, if there's one thing the judges learned from this contest, it's that there is much to learn from this contest. Each of the entries demonstrated from one to ten hideously powerful and educational Perl constructs; in terms of time spent, there can be few better methods of learning how Perl works than examining these expert-crafted hacks. We recommend that anyone who wants to advance their Perl knowledge download these entries and try to decipher them. The especially masochistic and adventuresome may wish to stop reading right here; some of the below passages will contain giveaways.

Last chance to stop reading before the awards are presented and the code is deconstructed...

Still with us? Great.

Best Four-Line Signature

3rd Place Tie: Krishna Sethuraman, Sriranga Veeraraghavan. Krishna did a good job of hiding the `index` function which extracted `Just another Perl hacker` from a string. Extra bonus points for using the `@bar{@array}` feature. Sriranga's code, studded with dollar signs, won high aesthetic formatting marks.

2nd Place: Poul Sørensen. Poul's code was fairly straightforward; at least one of the judges managed to understand what it would do without running it. However, it's a clever and cute hack which runs a capital letter up and down the lowercase string `the perl journal`.

1st Place: Robert Klep. Robert's code wasn't highly obfuscated; most of the difficulty of reading it came from the fact that it's got a lot of math in it. However, it won

big in the amusement department by calculating and printing out the Mandelbrot set in ASCII in only 2.5 lines. Here's Robert's entry:

```
#!/usr/bin/perl
$Y=-1.2;for(0..24){$X=-2;for(0..79){($r,$i)=(0,0);for(0..15){$n=$_;$r=($x=$
r)*$x-($y=$i)*$y+$X;$i=2*$x*$y+$Y;$x*$x+$y*$y>4&&last}print unpack("\@$n a"
,".,:;=+itIYVXRBM ");$X+=3/80}$Y+=2.4/25}
```

Most Powerful

3rd Place: Robert Klep, for his Mandelbrot set generator.

2nd Place: Gordon Lack. Here's Gordon's entire program:

```
#!/usr/bin/perl -l -w015l12pi.bak
```

Which, as you can plainly see, converts Mac-format text files into Unix-format text files.

1st Place: Russell Caton, who managed to squeeze a clever program that searches through your (optionally unordered) password file to find the first unused UID into only 1.5 lines of Perl. It's a genuinely useful piece of code for sysadmins disguised as line noise:

```
$-=100;while((($@)=(getpwent())[2])){push(@@,$@);}foreach(sort{$a<=>$b}@@){
(($_<=$-)||($_==($-+++1)))?next:die"$-\n";}
```

Most Creative

3rd Place: Stephen McCamant. Stephen's obfuscation is mostly in the math, but he gets great style points for having the last statement be `goto a` and for the execution of the program, which calculates and prints out π.

2nd Place: Steve Lidie (see the "The Perl Journal" category, below).

1st Place: Bob Sidebotham, whose submission was unbelievably hilarious. We recommend you go check out the original version. Needless to say, his program does not use π as it would have you believe, nor does it compute anything having to do with circles, nor are the comments true in the slightest. It's a big, majestic lie—its output is `THE PERL JOURNAL` spaced across the screen in five-character-high letters. Bob wins this category hands down, for apparently discovering that a higher power, à la Carl Sagan's *Contact*, has hidden the name of the best programming journal ever in a fundamental mathematical constant.

```
$maxerrors = 220; # needs tuning
$pi = reverse "3.14159265358979323846264338327950288419716939937510582097494459
30781640628620899862803482534211706798214808651328230664709384460955058223172535
94081284811174502841027019385211055596446229489549303819644288109756659334461284
75648233786783165271201909145648566923460348610454326648213393607260249141273724
58700660631558817488152092096282925409171536436789259036001133053054882046652138
41469519415116094330572703657595919530921861173819326117931051185480744623799627
49567351885752724891227938183011949129833673362440656643086021394946395224737190
70217986094370277053921717629317675238467481846766940513200056812714526356082778
```

```
5771342757789609173637178721468440901224953430146549585371050792279689258923542O
1995611212902196086403441815981362977477130996051870721134999999837297804995105 9
73173281609631859502445945534690830264252230825334468503526193118817101000313783
87528865875332083814206171776691473035982534904287554687311595628638823537820166
73231564231563231874231873231284231283236583236973236472239231011673231564231563
23287323187423128323128323158423158323197423197323147723923101167323656323187123
18722318742352832315842315832319742319732314772392310116732315642315632318722318
71231873231284231283231584231583231974231973231477239231011674234564231873232873
23628323158423158323697323147723923301723923101667323128423128323696823108823101
16742312832312882319682310882310116742352862339642350882310116732312842312882319
63231084231088231011673236283236963236088234016963231274231276231482231011963231
27423127623148223101923396323627623148223101196323127423127623148223101696323127
423127423548101";

while ($offset < length($pi)) {
        my($x) = substr($pi, $offset +++ 0, 2);
        my($y) = substr($pi, $offset +++ 1, 1); # XXX should be 3?
        my($z) = substr($pi, $offset +++ 2, 1);
        if ($x * cos($y) / cos($z)) {
                $dbg .= chr ($x) x $y;
                if (++$errors >= $maxerrors) {
                        # "cannot happen"
                        die("$dbg\n");
                }
        }
}

# passes sig test
print("ok!\n");
```

Best "The Perl Journal"

This award goes to the best program that produces the words `The Perl Journal`. There were some scintillating gems in this category, which made picking winners very difficult.

(Dis-)Honorable Mentions: Poul Sørensen for his neat streaming banner hack and Krishna Sethuraman for an elegant little haiku (reminiscent of the bad old days of Perl poetry) which compiles and runs.

3rd Place: Our own Steve Lidie, who threw the proverbial kitchen sink of obfuscation at the problem, including `__DATA__`, random numbers, cunningly commented code which has nothing to do with the solution, and a trashed out string which gets transformed into Perl code and evaluated. His effort was not only gorgeous, but awe-inspiring.

2nd Place: Bill Pollock, whose code contains a big `The Perl Journal` mural formatted prettily in comments. The code quickly reads itself in and uses characters from *inside* the comments to generate `The Perl Journal`. While the code isn't highly obfuscated, the idea of a program reading itself and then using a mural to make a string is pretty nifty.

1st Place: Gisle Aas. Gisle's entry only serves to strengthen our deepening suspicion that something is seriously amiss in Norway. Gisle's entry is so magnificently obfuscated that it's in a class by itself; in only 143 characters, Gisle manages to confuse Perl's namespace, Perl's notion of numbers, use the tenth day after the epoch began, and put together a tour-de-force substitution which one of the judges still doesn't understand. A hearty congratulations to both Gisle and his future therapist. With this entry, Gisle goes on to win the coveted Best of Show award and a mandatory seat on next year's judging committee. His entry:

```
*_=\$#;$/=q#(.)#;$#=10;$^X=~s|.*/||;$=chr;$#=gmtime$#;substr($#,$^F#^F
*$^F**$^F-1)=al;s$\$/( )\$/\$/$e\$2\u\$^X\$2\$3o\$1r$    && print time
```

Congratulations also go out to all the winners, who each richly deserve their titles and trophies, and also to all who participated. We fully expect that next year, not only will all of the judges return to a state of mental competency, but the contest will be even fiercer!

—Felix Gallo and Jon Orwant
The Official Highly Trained Zeroth Obfuscated Perl Contest Judges