

Tiger Woods isn't Worried, and Neither is abigail: A duffer's first try at (perl) golf

Mike Giroux, rmgiroux@acm.org

Abstract

In March 2002, The Perl Review held its Second Perl Golf tournament, TPR(0,1). I decided to try my hand at the sport, and I chronicle my first attempt in this article. I describe each step I took, including the ones which are so verbose now that I have seen the leaders' solutions, including Ton Hospel's amazing winning entry. Even though I did not win, or even come close, I learned quite a bit about Perl.

1 The challenge

The Perl Golf website describes the rules. In this challenge, the program must distill an integer down to its "secret number" by taking every consecutive pair of integers and adding them together to form a new number. The program continues until it reduces the number to a single digit.

For example, the program should reduce 7654 to 429 like this:

```
7+6 => 13, 1+3 => 4
6+5 => 11, 1+1 => 2
5+4 => 9
```

The program repeats the process until a single digit number remains. At each iteration, the value is printed to standard output. The program reads from the command line the integer it should reduce, which is the only argument.

2 My solutions

I came up with my first, verbose version of the script to make sure I had a working approach. I added comments to make this clearer; otherwise, it is the code I started with.

Code Listing 1: The verbose version

```

1  #!/usr/bin/perl
2
3  $_=shift;           # place the command line argument into $_
4  while($_>9){       # while we have more digits to process
5      $pd=10;        # initialize "previous digit" to "none"
6      print "$_\n";  # print the current number
7      while(/(\d)/g){ # for each digit in $_
8          if($pd>9){ # if we don't have a previous digit,
9              $pd=$1; # save this one and then
10             next    # restart the loop
11         }
12         $nd=$pd+$1; # "next" is "previous" plus the current one
13         $pd=$1;     # save current digit as next "prev digit"
14         if($nd>9){  # if next digit is greater than 9
15             $nd=~/(\\d)(\\d)/ or die;
16             $nd=$1+$2; # repeat the addition
17         }
18         $nn.=$nd;   # concatenate next digit onto "new number"
19     }
20     $_=$nn;        # update $_ with new number
21     $nn='';
22 }
23 print "$_\n";     # print the final answer

```

The first try illustrates the main parts of the problem.

1. The program has to extract the command line argument from @ARGV, because \$ARGV[0] is too verbose for a golf challenge.
2. The program has to loop until the solution is complete, and print the result at each iteration
3. The program has to deal with the digits two by two, but /(.) (.) /g will not work since the second character of the first pair is the first character of the second pair. Given 1234, /(.) (.) /g will match 12 then 34, while we need 12, 23, and 34.
4. The program has to convert each pair in (00..99) to its corresponding secret digit between 0 and 9. For some pairs, there will be an intermediate step between 10 and 18.

3 Shaving strokes

Initially, I took the worst possible approach to all four of those problems, unfortunately. But the tpr01.pl test program reported that it passed all tests, scoring it at 368 strokes. I removed all whitespace, except newlines, and shrank all the two character variable names down to one character. This program still works, and I scored a much-improved 184 strokes.

Code Listing 2: Removing whitespace

```

1  #!/usr/bin/perl
2
3  $_=shift;
4  $p=10;
5  while($_>9){
6  print"$_\n";
7  while(/(\d)/g){
8  if($p>9){
9  $p=$1;
10 next
11 }
12 $x=$p+$1;
13 $p=$1;
14 if($x>9){
15 $x=~/(\\d)(\\d)/;
16 $x=$1+$2;
17 }
18 $n.=$x;
19 }
20 $_=$n;
21 $n='';
22 $p=10;
23 }
24 print"$_\n";

```

However, the scoring program counts newline characters in the score, so from here on, I write everything as one-liners. I had to break the lines for printing, but I submitted each of these scripts as a one-liner and I do not count the newlines in my score.

Code Listing 3: Removing newlines -- 162 strokes

```

1  #!/usr/bin/perl
2  $_=shift;$p=10;while($_>9){print"$_\n";while(/(\d)/g){if($p>9){$p=$1;next}$x=$p+$1;$p=$1;
3  if($x>9){$x=~/(\\d)(\\d)/;$x=$1+$2;}$n.=$x;}$_=$n;$n='';$p=10;}print"$_\n";

```

This is still the same approach as the commented version, essentially. I scored 162 strokes with this version, but still was not competitive, so I knew I had to find some way to cut down on the size of the program. My next idea was to try to find repeated strings, put them in variables, and use the string form of the eval function to expand the variables.

Code Listing 4: Remove repeated variables -- 206 strokes

```

1  #!/usr/bin/perl
2  $w='while('; $p='print"*_\n"'; $v='*p=10;'; $i='if('; $d='(\d)'; $a="*_=shift;$v$w*_>9)
3  {$p$w/$d/g){$i*p>9){*p=*1;next}*x=*p+*1;*p=*1;$i*x>9){*x=~/$d$d/;*x=*1+*2;}*n.=*x;}*_*=$n;
4  *n='';$v}$p"; $a=~s/\\*/\\$/g;eval$a;

```

Fortunately for my sanity, that turned out to be worse, and yielded 206 strokes. I had two problems with this approach. I had to replace all the \$ symbols in the string to be eval'ed with another symbol, so they

would not be prematurely evaluated during the double quoted interpolation at the `$a=""` assignment. Then I had to do a global substitution to replace those symbols with “\$”. And since I moved common operations out of line into the substitution variables, maintaining and improving this version would have been difficult. I found that the overhead required for this tack was too high for a low-par golf course like this one. I might have to revisit this idea in the future for longer problems.

By doing the “compression attempt” in code listing 4, I realized I was setting `$p=10` twice in my code, so I went back to the 162 stroke version and corrected that problem, which cut my score down to 156.

Code Listing 5: Removing repetition -- 156 strokes

```

1  #!/usr/bin/perl
2  $_=shift;while($_>9){$p=10;print"$_\n";while(/(\d)/g){if($p>9){$p=$1;next}$x=$p+$1;$p=$1;
3  if($x>9){$x=~/(\\d)(\\d)/;$x=$1+$2;}$n.=$x;}$_=$n;$n='';}print"$_\n";

```

At this point, I read the postmortem for *The Perl Review's* first golf tournament and after looking at Ton Hospel's winning entry, I realized that I could use the pop function just as easily as shift to get the command line argument, saving two strokes, and that by using the `-l` switch, I could save two newlines, saving one stroke net. I was now at 134 strokes.

Code Listing 6: Using pop -- 134 strokes

```

1  #!/usr/bin/perl -l
2  $_=pop;while($_>9){$p=10;print"$_" ;while(/(\d)/g){if($p>9){$p=$1;next}$x=$p+$1;$p=$1;$x-=9
3  if($x>9);$n.=$x;}$_=$n;$n='';}print"$_" ;

```

Of course, I forgot that `$_` is the default argument to print, and that I did not need the double quotes. I corrected that and removed some extra semicolons to get me down to 118 strokes.

Code Listing 7: Default arguments -- 118 strokes

```

1  #!/usr/bin/perl -l
2  $_=pop;while($_>9){$p=10;print;while(/(\d)/g){if($p<=9){$x=$p+$1;$p=$1;$x-=9if($x>9);$n.=$x}
3  $p=$1}$_=$n;$n=''}print

```

At this point, I could not squeeze my solution any farther. I needed a better approach to one of the four problems. I decided to focus on the problem of getting the pairs of digits.

I tried using `/(\d)(\d)/g`, the lookahead assertion, to match a second digit but to not consume it. That worked, but unfortunately, I found out that `(?=)` are not capturing parentheses, so I had to use `/(\d)(?=(\d))/`. I also figured out that I could sum the new digits with the ternary operator, `($1+$2)>9?($1+$2):($1+$2-9)`. I precomputed `$x=$1+$2`, so that became `$x>9?$x:$x-9`.

Code Listing 8: Ternary operator -- 87 strokes

```

1  #!/usr/bin/perl -l
2  $_=pop;while($_>9){print;s/(\d)(?=(\d))/x=$1+$2;$n.=$x>9?$x-9:$x/eg;$_=$n;$n=''}print

```

That got me down to 87 strokes. Switching to `$x-9*($x>9)` saved a stroke, and got me to 86.

Code Listing 9: No ternary operator -- 86 strokes

```
1 #!/usr/bin/perl -l
2 $_=pop;while($_>9){print;s/(.)(?=(.))/x=$1+$2;$n.=$x-9*($x>9)/eg;$_=$n;$n=''}print
```

I had another major breakthrough when I realized how stupid I was to build up `$n` all the time, when the substitution already modified `$_` almost exactly the way I wanted. However, the substitution left the last digit unchanged, but a simple chop corrected that. Now I was down to 75 strokes.

Code Listing 10: Losing `$n` -- 75 strokes

```
1 #!/usr/bin/perl -l
2 $_=pop;while($_>9){print;s/(.)(?=(.))/x=$1+$2,$x-9*($x>9)/eg;chop}print
```

I switched to a “do while” instead of the while loop which allowed me to get rid of the `>9` and one of the print statements. I saved two more characters by getting rid of the `()` around the while condition, shrinking the code to 68 strokes.

Code Listing 11: do {} while -- 68 strokes

```
1 #!/usr/bin/perl -l
2 $_=pop;do{print;s/(.)(?=(.))/x=$1+$2,$x-9*($x>9)/eg;chop}while$_
```

I realized that now that I was not using the ternary operator, `$x` cost me more than it gained, so I ditched it. That got me to 65 strokes.

Code Listing 12: Dropping the ternary operator -- 65 strokes

```
1 #!/usr/bin/perl -l
2 $_=pop;do{print;s/(.)(?=(.))/x=$1+$2-9*($1+$2>9)/eg;chop}while$_
```

From here, I tried all kinds of things to improve my solution, but this is the best answer I came up with, placing 18 strokes behind the winner and I am pretty sure this means I missed the cut. But I did try a couple of other avenues which were interesting.

I thought of precomputing the answers to the digit reduction. However, that only got 88 strokes. After the contest ended, by looking at one of the other solutions I realized I could have just assigned `@a=(0..9,1..9)` but that would only have gotten me to 72 strokes.

Code Listing 13: Precomputation -- 88 strokes

```
1 #!/usr/bin/perl -l
2 @a=map{int($_%10+$_/10)}0..18;
3 $_=pop;do{print;s!(.)(?=(.))!a[$1+$2]!eg;chop}while$_
```

I figured out that the calculation could be expressed as $(\$1+\$2-1)\%9+1$ for all $(\$1,\$2)$ except $(0,0)$. That only scored 68.

Code Listing 14: Another way to sum -- 68 strokes

```

1  #!/usr/bin/perl -l
2  $_=pop;do{print;s/(.)(?=(.))/\$1+\$2&&(\$1+\$2-1)\%9+1/eg;chop}while$_

```

I thought of using a subroutine for $\$1+\2 . I could only shrink that down to 71, though.

Code Listing 15: Using a subroutine -- 71 strokes

```

1  #!/usr/bin/perl -l
2  sub x{ \$1+\$2 }$_=pop;do{print;s/(.)(?=(.))/&x-9*(&x>9)/eg;chop}while$_

```

Next, I tried attacking the do-while loop, by rerunning the program with the exec function. I could only get this to 68 strokes, and it would not have worked in the tournament environment since the scripts are run with mode 0644, which means that I would have had to do something like `execX,0,$_`, which is even worse.

Code Listing 16: Using exec -- 68 strokes

```

1  #!/usr/bin/perl -l
2  $_=pop;print;s/(.)(?=(.))/\$1+\$2-9*(\$1+\$2>9)/eg;chop;$_&&exec$0,$_

```

I thought that maybe it was a *lack* of goto's that was harmful in this case. This was 67 strokes.

Code Listing 17: Using goto -- 67 strokes

```

1  #!/usr/bin/perl -l
2  $_=pop;a:print;s/(.)(?=(.))/\$1+\$2-9*(\$1+\$2>9)/eg;chop;$_&&goto a

```

Once the contest ended and I saw the other solutions, the biggest and most obvious improvement I could make to my script was to get rid of the first capturing parentheses, and use `$&`. That would have gotten me to 63 strokes.

Code Listing 18: Using \$& -- 63 strokes

```

1  #!/usr/bin/perl -l
2  $_=pop;do{print;s/.(?=(.))/\$1+$&-9*(\$1+$&>9)/eg;chop}while$_

```

I had a great deal of fun competing in the tournament. My experience of avoiding default arguments for clarity's sake in my scripting was a handicap in golf, and it was interesting to look at every cranny of the program trying to find one more character to squeeze.

I hope you will join me in the next round! Next time, I'm going to be brave, or foolhardy, and compete with the big kids in the expert ladder.

4 References

Perl Golf website – <http://perlgolf.sourceforge.net>

TPR(0,0) overview – http://www.theperlreview.com/Issues/The_Perl_Review_0.1.pdf

Background about the history of perl golf – <http://archive.developer.com/fw@perl.org/msg01949.html>

The post where the name appears to have originated – Greg Bacon, Message-ID: 7imntim.jh1@info2.uah.edu

5 About the author

Mike Giroux is a programmer originally from Quebec, now living in Connecticut. If you think the perl golf was ugly, you should see him try to play *real* golf.