[VA] SRC #017 - April 1st, 2024 Spring Special

1st April, 2024, 20:59 **Post: #1**

Valentin Albillo
Senior Member

Posts: 1,100
Joined: Feb 2015
Warning Level: 0%

[VA] SRC #017 - April 1st, 2024 Spring Special

Hi, all,

Welcome to my new *SRC #017 - April 1st, 2024 Spring Special*

Once again **April 1s**t is here and I want to celebrate both it and the new season with this ***Spring Special*** where I'm proposing a number of mini-challenges for you to tackle with your favorite *vintage HP* calc, plus interesting facts not widely known (if at all).

> ***Note****: No hard rules, post here whatever you want as long as it's on topic and absolutely* ***NO CODE PANELS*** *(lest I'll consider you a troll only fit for my* ***Ignore*** *list,) but I'd appreciate it if you'd use vintage HP calcs (physical/virtual,) unless you're clueless as far as programming them is concerned.*

As *April 1st* is also referred to as *April Fool* day, I'll take the last letter from *April* (**L**) and the two last ones from *Fool* (**OL**) to title each separate Section with a jocular prefix i.e. **LOL** !

1. LOL the First: Squares

Let's start nice and easy. Square numbers have been a source of beauty and admiration since ***Pythagoras*** did his thing with them millennia ago. Just look at these small (related) beauties !

$375{,}501^2 = 141{,}\mathbf{001{,}001{,}001}$

$751{,}002^2 = 564{,}\mathbf{004{,}004{,}004}$

Now, if these are cute just consider the sheer amounts of beauty you'll discover when dealing with their bigger relatives, so this mini-challenge asks you to calculate the following 2^2 squares:

$\mathbf{130834904430015239}^2 = ?$

$\mathbf{4712877147889716634938 99}^2 = ?$

$\mathbf{25781108305591628417975738}^2 = ?$

$\mathbf{14142208287606721994980505 0005}^2 = ?$

I know you can simply paste them in *Wolfram Alpha* or use the multiprecision library in your *RPL* model or mutiprecision canned software in any device or web site and get done with it, but it would be so ***lame*** that you'll risk *ridicule*. What I'm asking you to do is to program your own multiprecision squaring routine in your vintage *HP* calc and use it to get the results. It's not that difficult at all, you know ... that is, if you've got what it takes .. 🙂

I'll post my own squaring routine that I wrote for the **HP-71B** from scratch, just a mere *483 bytes* long and taking the form of a user-defined function so that it can be called right from the command line. It produces results like this, which you can profitably use for testing your code:

$31479267847267265637800300423742371877 51^2 =$

99094430419999466860630132079325624628516136149764941223248023037797 7224438001

Remember: if you decide to solve this mini-challenge, you *must* post both the *(beautiful) results* **and** the *code* which produces them, do not post results alone. Comments are most welcome, plus you might try getting more results like these.

2. LOL the Second: GCD

OK, full throttle now. You may remember the **GCD** function appearing in my recent ***SRC #016 - Pi Day 2024 Special***, where it was used in the *Second appearance* to count the number of co-prime pairs of random integers in a range and then this count was used to compute an approximation to π.

Sample **GCD** values: **GCD(35, 13) = 1**, **GCD(35, 14) = 7** and **GCD(35, 15) = 5**

In this mini-challenge you must write code for any *HP calc* to find out the answer to this simple question:

*As we have that **GCD(15, 4) = 1** and **GCD(15, 5) = 5**, for what value of **x** is **GCD(15, x) = 2** ?*

As always, you must post both results and code. Comments also most welcome. I'll post mine next Sat/Sun.

3. LOL the Third: Random

You may be aware that advanced 12-digit *HP* models of old incorporated an excellent **RNG** *(pseudo-Random Number Generator)* which could generate a trillion (i.e. 10^{12}) full 12-digit real (pseudo-)random numbers in the interval *[0-1)* before repeating. This *RNG* passes the Spectral test so it's extremely reliable for use in simulations and other advanced topics (e.g. multidimensional integration, Monte Carlo algorithms, etc.) without fear of any bias or short period degrading the results.

As far as I know, the implementations for the **HP-71B**, the *RPL* models, the **HP-42S** and ***Free42*** (and perhaps many other *HP* models) are functionally identical so they produce exactly the same sequence of **RND**s from the same seed. Now, this mini-challenge's question is:

*Two consecutive **RNDs** cannot be exactly equal or the generation of subsequent **RNDs** would be stuck in a loop. But **how close** can two consecutive **RNDs** actually be ?*

Use `RANDOMIZE 1` or the equivalent instruction to specify the value **1** as the seed and write code to output the list of ever closer consecutive **RND**s and their index **N** in the sequence, like this:

```
>RUN
      N       First RND         Next RND          |Difference|
 ---------------------------------------------------------------
      2     .731362440213     .77207218067       .040709740457
     13     5.64471991805E-2  6.30768172146E-2  6.6296180341E-3
    125     .805774019056     .803607575861      .002166443195
    ...
```

If you intend to go very deep into the sequence you'd better use a fast virtual model such as ***Emu71/Win*** or ***Free42*** for faster results and/or greater depths. You might even boldly examine the *entire* trillion-strong sequence to find out the absolute closest pair !

I'll post my *3-line*, *~90 byte* solution which produces the above, plus a *4-line*, *149-byte* variant which *stops* right *before* a given pair of consecutive **RND**s is generated so that you can generate them manually right from the command line (i.e. `>RND;RND`) and see for yourself how close they indeed are.

And remember, post code and results. What's your record ? Me, I've found a pair of consecutive **RND**s which are only *0.00000000001* apart ! Can you do better ?

4. LOL the Fourth: Logs

This is not a mini-challenge but a somewhat unexpected fact I've found, so you don't need to do anything but read on and eventually *comment*.

I've always thought that the two *logarithmic* functions available in the *10-digit* **HP-15C** (**LOG** and **LN**, base ***10*** and base ***e*** respectively) would have essentially the same accuracy overall, but lo and behold, I've found that **LOG** seems to be substantially more accurate in some cases than **LN**.

For instance, let's consider computing $\log_b(5^7) / \log_b(5)$, which should return exactly **7** for any base **b** ≥ 2. But the actual results are:

$LN(5^7) / LN(5)$ -> 7.00000000**4** *{ 4 ulp from the exact value }*
$LOG(5^7) / LOG(5)$ -> 7 *{ exact }*

and the same happens with other powers **N** of **5**, e.g. for **N** = **2** to **9** we have:

N	LOG	LN
2	2.00000001	2.00000001
3	3	3.00000001
4	4	4.00000001
5	5.00000001	5.00000001
6	6	6.00000002
7	7	7.00000004
8	8.00000001	8.00000002
9	9	9.00000001
Total *ulps*:	3	13

{ more than 400% larger error overall }

This *much* larger error puzzled me no end but I thought that perhaps the limited *10-digit* accuracy (*13-digit* internally) might be playing a role and decided to test the same computations using the *12-digit* (*15-digit* internally) **HP-71B**, which has **LN**, **LOG10** and **LOG2** (respectively base ***e***, base ***10*** and base ***2***.) These were the results for power **8**:

```
>STD
>LN(5^8) / LN(5)          -> 8.00000000004    { 4 ulp from the exact value }
>LOG10(5^8) / LOG10(5)   -> 8                 { exact }
>LOG2(5^8) / LOG2(5)     -> 7.99999999999     { 1 ulp from the exact value }
```

so we see that all three results differ among them, being in error by *4 ulp*, *0 ulp* and *1 ulp*, respectively, and again the same happens with other powers **N** of **5**, e.g. for **N** = **2** to **9** we have:

```
    N      LOG10            LN
 --------------------------------------
    2    2                2.00000000001
    3    3                3.00000000001
    4    3.99999999999    4.00000000001
    5    5                5.00000000001
    6    6.00000000001    6.00000000001
    7    7                6.99999999999
    8    8                8.00000000004
    9    8.99999999999    9.00000000002
---------------------------------------
  Total ulps:      3           12          { 400% larger error overall }
```

and again the **LN** error is much larger, so having two extra digits didn't help at all. For **LOG2** (not shown in the table above) the *Total ulps* for the range is *6 ulps*, <u>half</u> **LN**'s total error but <u>twice</u> **LOG10**'s total error. In the end, **$\mathbf{LOG_{10}}$** seems to be the most accurate logarithm available.

If you want to pursue the matter, you can try the above examples in your *HP* models, both *10-digit* and *12-digit*, from the first **HP-35** up to the latest *RPL* models, to see if **$\mathbf{LOG_{10}}$** and **LN** differ that much in their respective errors. Testing other range of values (here from 5^2 = **25** to 5^9 = **1,953,125**) could be revealing as well. Are there arguments with even larger errors ?

At any rate, comments and discussion would be most welcome.

5. LOL the Fifth: Gamma

Here's something *peculiar* I noticed while playing around with my **HP-71B** several decades ago, in *Experimental Mathematics* fashion.

Back then, I executed this loop directly from the command line to list the values of $\Gamma(10^{-1})$, $\Gamma(10^{-2})$, ..., $\Gamma(10^{-10})$, and got the following unexpectedly peculiar results:

```
>DESTROY ALL @ FOR N=1 TO 10 @ N;GAMMA(10^(-N))@ NEXT N
```

N	$\Gamma(10^{-N})$
1	9.51350769867
2	99.4325851191
3	999.423772485
4	9999.42288323
5	99999.4227942
6	999999.422785
7	9999999.42278
8	99999999.4228
9	999999999.423
10	9999999999.42

where the evergrowing *integer part* of each value is clearly **$\mathbf{10^N}$ - 1**, while the *fractional part* seems to be quickly converging to some limit around ***~0.42*** but the *12-digit* **HP-71B** lacks the accuracy needed to refine it further, so this mini-challenge is:

You must write code to find this limit to much greater accuracy (say *10-12* digits or more,) perhaps by simply using ***Free42*** *Decimal* to compute the values. Once done, answer these two questions:

- Can you *estimate* the *most accurate* value you got ? Because, as the integer part is exponentially growing, even the *34-digit* ***Free42*** *Decimal* will soon begin to *lose digits* in the fractional part, as it happened with the **HP-71B** results above. In that case, an *RPL* model with arbitrary *multiprecision* capability would surely help.
- Can you *identify* the *symbolic*, closed form of that numeric value ? My IDENTIFY program *v2.0 (unreleased)* sure can.

As always, post here your results *and* code (comments also welcome,) and please try to <u>steer clear</u> of references such as *Wolfram*, *MathWorld*, *Wikipedia*, ... as you'd just *ruin* the pleasure of doing a little *Experimental Mathematics* all on your own.

6. LOL the Sixth: Miscellanea

1. *[HP-71B specific]* Try and deduce what result will be output by this expression *without* actually executing it: some regular **numeric** value ? Perhaps **Inf** or **NaN** ? An **error** message ? A long-running or indefinite internal **loop** ?

   NEIGHBOR(INTEGRAL(FNROOT(EPS,EPS,EPS),EPS,EPS,EPS),GAMMA(EPS))

   Once you've thoughtfully made your deduction, execute the expression and see what comes out. Was your deduction correct ? Can you explain the result obtained ?

   Please post your deduction & comments on the result.

2. *[HP-71B specific]* I executed this command-line expression on my **HP-71B** but it just resulted in `System Error`. What did I do wrong ? **Bad** addresses ? **Forbidden** range for `PEEK$` ? Can you find out what's wrong ?

   FOR I=64204 TO 64215 @ DISP CHR$(HTD(REV$(PEEK$(DTH$(2*I),2)))); @ NEXT I

3. Some nice results I got (you can check the √s with ***Free42*** *Decimal*):

   √ 95888 = 309.657875727390470**0000000**975517...
   √ 22008840 = 4691.358**0123456789**961013...

   (HP-71B specific, RPL/12-digit models too, maybe other 12-digit models) In **radians**:

   ```
   >EXP(ACOS(430/433))   ->  1.12500000006     { ~  9/8 }
   >EXP(ACOS(538/541))   ->  1.11111111113     { ~ 10/9 }
   ```

   Notice that **433** = **430** + **3** and **541** = **538** + **3**, i.e. both fractions are of the form **N** / (**N+3**) and produce results of the form **(N+1)** / **N** so perhaps there's some *hidden pattern* at large here. Or not !

4. *[HP-71B specific]* What does this **HP-71B** user-defined function compute ?

   ```
   10  DEF FNC(M,N)=M!/N!/(M-N)!
   ```

5. *[HP-71B specific]* The HP-71B does not allow for variable names beginning with a 2-letter or more prefix (e.g. `AB`, `CD7`, `MYTAXES`, etc.) as many other pocket computers do, but executing this assignment in a program or from the keyboard ...

   ```
   FORM=STOP
   ```

   ... *doesn't* result in a *Syntax Error* so what gives ?

6. Last but not least, you're probably acquainted with the number **42**, not only because it's the model number of the renowned *HP-**42**S*, later simulated by the *Free**42*** virtual version, but also because according to *The Hitchhiker's Guide to the Galaxy* **42** is "*The Answer to the Ultimate Question of Life, the Universe, and Everything*", which is saying a whole lot.

   However, this doesn't end **42**'s importance to this Universe and to prove the point you must write code to compute to full accuracy this simple function ***S(x)*** given the argument ***x***:

   $$S(x) = \frac{1}{\sum_{n=0}^{\infty} \binom{2n}{n}^3 \frac{nx+5}{2^{12n+4}}}$$

   Use your code to compute ***S(42)***. Astounding, isn't it ? 🙂

   Do *not* use any built-in summation capability but write and use instead your own code and post here both code and result. I'll post my simple *3-line* **HP-71B** user-defined function in a few days.

That's all. Waiting for your clever solutions and/or comments galore, I'll post mine next Saturday/Sunday so you've got plenty of time to concoct and polish your code. And no `CODE` pannels, please.

V.

WWW FIND EDIT QUOTE REPORT

2nd April, 2024, 10:29 (This post was last modified: 2nd April, 2024 10:36 by J-F Garnier.) **Post: #2**

J-F Garnier
Senior Member

Posts: 940
Joined: Dec 2013

RE: [VA] SRC #017 - April 1st, 2024 Spring Special

Valentin Albillo Wrote: (1st April, 2024 20:59)

As we have that ***GCD(15, 4) = 1*** *and* ***GCD(15, 5) = 5****, for what value of* ***x*** *is* ***GCD(15, x) = 2*** *?*

OK, let's try to solve GCD(15,X)=2 for X on the 71B w/ Math ROM.
Since GCD only accepts integer values, I had to cast the X variable to the INT type.
Choosing 0 and 10 as the initial guesses:

```
>FNROOT(0,10,GCD(15,INT(FVAR))-2)
12.9999999999
```

We can safely round the result to X=13.
An interesting result, isn't it?

Quote:

[*] *[HP-71B specific]* Try and deduce what result will be output by this expression *without* actually executing it: some regular **numeric** value ? Perhaps **Inf** or **NaN** ? An **error** message ? A long-running or indefinite internal **loop** ?

NEIGHBOR(INTEGRAL(FNROOT(EPS,EPS,EPS),EPS,EPS,EPS),GAMMA(EPS))

I miserably failed to predict the result !
But I learnt something about the 71B solver...

Quote:

[*] *[HP-71B specific]* I executed this command-line expression on my **HP-71B** but it just resulted in `System Error`. What did I do wrong ? **Bad** addresses ? **Forbidden** range for `PEEK$` ? Can you find out what's wrong ?

FOR I=64204 TO 64215 @ DISP CHR$(HTD(REV$(PEEK$(DTH$(2*I),2)))); @ NEXT I

Ah ! Nothing wrong, I even didn't need to try ...

Quote:

[*] *[HP-71B specific]* What does this **HP-71B** user-defined function compute ?

```
10  DEF FNC(M,N)=M!/N!/(M-N)!
```

It doesn't do much, I'm afraid.

Quote:

[*] *[HP-71B specific]* The HP-71B does not allow for variable names beginning with a 2-letter or more prefix (e.g. `AB`, `CD7`, `MYTAXES`, etc.) as many other pocket computers do, but executing this assignment in a program or from the keyboard ...

```
FORM=STOP
```

... *doesn't* result in a *Syntax Error* so what gives ?

Ah! another new "hidden feature" of the 71B ?

I well remember the undocumented MEMORY function, that accepts up to 2 parameters, but isn't that useful (do first DESTROY ALL to get consistent results):

```
>MEMORY(1,2)
1
```

More results later...

J-F

EMAIL PM WWW FIND QUOTE REPORT

2nd April, 2024, 21:38 **Post: #3**

John Keith
Senior Member

Posts: 1,027
Joined: Dec 2013

RE: [VA] SRC #017 - April 1st, 2024 Spring Special

Interesting challenges as always, Valentin!

I couldn't resist the last one, **S(42)**. Here is a quick and dirty RPL program using a **FOR** loop as a fake **DO** loop with a counter.

```
\<< 0. 0. 99.
FOR k OVER k * 5. +
k DUP 2. * SWAP COMB 3. ^ *
2. k 12. * 4. + ^ /
OVER + DUP ROT \=/
1. 99. IFTE
STEP INV SWAP DROP
\>>
```

I also noticed that using the bottom part of the expression (remove the **INV** from the last line), a value of about 8.45 will return a number close to 1/10 of **S(42)**.

EMAIL PM FIND QUOTE REPORT

2nd April, 2024, 21:41 (This post was last modified: 2nd April, 2024 21:51 by J-F Garnier.) **Post: #4**

J-F Garnier
Senior Member

Posts: 940
Joined: Dec 2013

RE: [VA] SRC #017 - April 1st, 2024 Spring Special

> **Valentin Albillo Wrote:** (1st April, 2024 20:59)
> I've always thought that the two *logarithmic* functions available in the *10-digit* **HP-15C** (**LOG** and **LN**, base ***10*** and base ***e*** respectively) would have essentially the same accuracy overall, but lo and behold, I've found that **LOG** seems to be substantially more accurate in some cases than **LN**.

I remember this has been reported a few times in the past, maybe in connection with some solutions of your challenges/SRCs, but I can't find any reference again (can you?).

I've never been convinced by this effect (decimal LOG better than natural LN), because internally the decimal log is computed by:
LOG(x) = LN(x) / LN(10) , using 3 extra guard digits.
So there is no reason for LOG to be more accurate, on the contrary it may be marginally less accurate.

Let's see your 15c example (a 10-digit machine) - with minor corrections:

N	$LOG(5^N)/LOG(5)$	$LN(5^N)/LN(5)$
2	2.000000001	2.000000001
3	3	3.000000001
4	4	4.000000001
5	5.000000001	5.000000001
6	6	6.000000002
7	7	7.000000004
8	8.000000001	8.000000002
9	9	9.000000001
Total *ulps*:	3	13

{ *more than 400% larger error overall* }

but **let's try another number: 11^N instead of 5^N**:

N	$LOG(11^N)/LOG(11)$	$LN(11^N)/LN(11)$
2	2	2
3	3	3
4	4.000000001	4
5	5.000000001	4.999999998
6	6.000000001	6.000000001
7	7.000000001	7
8	8.000000001	7.999999998
9	9.000000001	9.000000001
Total *ulps*:	6	6

{ ***now similar error overall*** }

Doing now the same test on the 71B (a 12-digit machine):

N	$LGT(11^N)/LGT(11)$	$LN(11^N)/LN(11)$
2	2	2
3	2.99999999999	3
4	3.99999999999	4
5	4.99999999999	5
6	5.99999999999	6
7	6.99999999999	7
8	7.99999999999	8

```
     9    8.99999999999    9
---------------------------------------
  Total ulps:       7           0
```
*{ **here, much larger error overall for the decimal LOG case** }*

So it's clear to me that we can't say that the decimal LOG provides more accurate results than the natural LN, overall.

The question we may ask is: why does it seem that LOG is better for expressions using a certain number such as 5^N, and why is LN better for other numbers such as 11^N ?

J-F

EMAIL PM WWW FIND QUOTE REPORT

3rd April, 2024, 20:13 (This post was last modified: 4th April, 2024 21:04 by C.Ret.) **Post: #5**

C.Ret
Member

Posts: 252
Joined: Dec 2013

RE: [VA] SRC #017 - April 1st, 2024 Spring Special

> **J-F Garnier Wrote:** (2nd April, 2024 10:29)
>
> > **Valentin Albillo Wrote:** (1st April, 2024 20:59)
> >
> > *As we have that **GCD(15, 4) = 1** and **GCD(15, 5) = 5**, for what value of **x** is **GCD(15, x) = 2** ?*
>
> OK, let's try to solve GCD(15,X)=2 for X on the 71B w/ Math ROM.
> Since GCD only accepts integer values, I had to cast the X variable to the INT type.
> Choosing 0 and 10 as the initial guesses:
>
> >FNROOT(0,10,GCD(15,INT(FVAR))-2)
> 12.9999999999
>
> We can safely round the result to X=13.
> An interesting result, isn't it?

This is a surprising result!

I believe that looking for *x* such that *GCD(15,x)= 2* is like looking for a hairy fish (a very common fish at the very beginning of the April River).

Here is my code for any HP-71B to find *x*: `10 DISP "GCD( 15 , NaN ) = 2"` and what it display:

```
GCD(  15   ,  NaN  )  =  2
```

But, I may have start by the first apriL foOL :

Here is my program to compute a large square on the HP-71B:

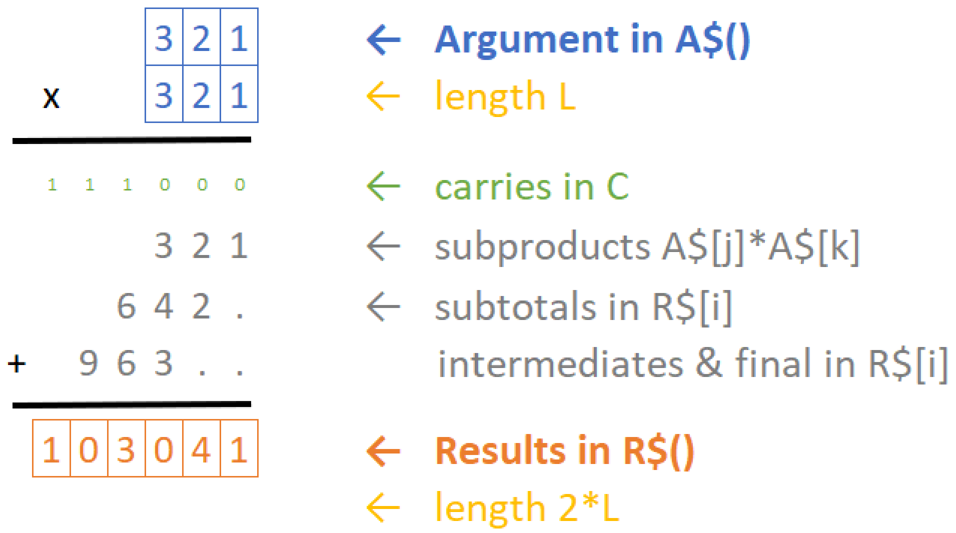
```
10 DESTROY ALL @ DIM A$[39] @ INPUT A$ @ L=LEN(A$) @ DIM R$[2*L] @ R$=SPACE$(48,2*L)
20 FOR K=L TO 1 STEP -1 @ C=0 @ FOR J=L TO 1 STEP -1 @ I=K+J
30 X=C+VAL(R$[I,I])+VAL(A$[J,J])*VAL(A$[K,K]) @ R$[I,I]=CHR$(48+MOD(X,10)) @ C=X DIV 10
40 NEXT J @ R$[K,K]=CHR$(48+C) @ DISP R$[K+L] @ NEXT K @ DISP R$ @ BEEP @ PAUSE
```

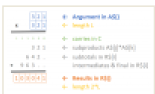
The calculation takes longer as the number is larger.
One can follow the progress of the computation as the figures are displayed as soon as they are determined. That is to say starting from the last towards the first.

```
[RUN]
? _
? 141422082876067219949805050005_ [END LINE]
                                                      (prgm)
> 5                                                  (prgm)
> 25                                                (prgm)
> 025                                                (prgm)
> 0025                                              (prgm)
> 00025                                              (prgm)
> 500025                                             (prgm)
. . .
> 25202222022205205000550500025                    (prgm)
> 225202222022205205000550500025                   (prgm)
~ biiiip ~
> 0200002055525005225202000505202222520222202205205000550500025         (susp)
```

You will note the presence of a leading zero which is important but which is not systematic. Try to calculate 99999999².

A small figure which shows how the calculation is done:





several edit to correct broken English, code syntax, insert illustrations and correct typos

PM FIND QUOTE REPORT

3rd April, 2024, 22:55 (This post was last modified: 3rd April, 2024 23:28 by Gerson W. Barbosa.) **Post: #6**

Gerson W. Barbosa
Senior Member

Posts: 1,580
Joined: Dec 2013

RE: [VA] SRC #017 - April 1st, 2024 Spring Special

LOL the First

HP-75C program:

```
10 OPTION BASE 0
15 INTEGER I,J,N
20 N=5
25 B=1000000
30 DIM A(9),B(5),C(12)
```

```
35 REM DIM A(N+4), B(N), C(2*N+2); N>1
40 FOR I=0 TO 2*N+2 @ C(I)=0 @ NEXT I
45 FOR I=0 TO N+4 @ A(I)=0 @ NEXT I
50 FOR I=1 TO N
55 READ A(I)
60 B(I)=A(I)
65 NEXT I
70 FOR I=N TO 1 STEP -1
75 T1=0 @ A1=0
80 FOR J=N-I+4 TO -1 STEP -1
85 C2=(T1+B(I)*A(J+1))/B
90 A1=FP(C2)*B
95 C(I+J)=C(I+J)+A1
100 T1=IP(C2)
105 NEXT J
110 NEXT I
115 FOR I=2*N TO 2 STEP -1
120 T=C(I)/B
125 C(I)=FP(T)*B
130 C(I-1)=C(I-1)+IP(T)
135 NEXT I
140 FOR I=0 TO 2*N-1
145 DISP C(I);
150 NEXT I
155 END
160 DATA 141422,82876,67219,949805,50005

>RUN
20000 205525 5225 202000 505202 222520 222202 205205 550 500025
```

That is,

```
1414220828760672199498050050005^2 =

2000020555250052252020005052022225202222022052050005505000025
```

LOL the Fifth:

I have digressed on this one and haven't done what has been asked (No cigar, I guess :-). I'll just say the fractional part tends to 1 minus a known mathematical constant to which the following is a pandigital appoximation in RPL algebraic expression format, good to twelve digits:

`'INV(√(3+7/((29/10)^8-INV(SQ((4^5)^INV(6))))))'`

On the HP 50g in approximate mode, `'1 - INV(√(3+7/((29/10)^8-INV(SQ((4^5)^INV(6))))))'` should return the same numeric result as `'1+Psi(1)'`.

Edited to fix a typo

EMAIL PM FIND QUOTE REPORT

4th April, 2024, 23:03 (This post was last modified: 4th April, 2024 23:09 by J-F Garnier.) **Post: #7**

J-F Garnier
Senior Member

Posts: 940
Joined: Dec 2013

RE: [VA] SRC #017 - April 1st, 2024 Spring Special

> **Valentin Albillo Wrote:** (1st April, 2024 20:59)
> Two consecutive **RND**s cannot be exactly equal or the generation of subsequent **RND**s would be stuck in a loop.

It's not correct. This would assume that the next RND values are fully determined by the last RND value, and this is wrong.
The RND value is determined by the internal seed, which is stored with 15 digits.
Or, in other words,we can't predict the next RND value from the last one (well, the possibilities for the next RND are limited).
To illustrate it, we can get the same RND value in two different sequences:

>RANDOMIZE .636248123586
>RND, RND, RND
.14159265359 .494478890124 .825547412541

>RANDOMIZE .90365957958
>RND, RND, RND, RND
2.71250347884E-2 **.14159265359** .416751399163 .130703385847

(done on a 71B, but as Valentin noted, the results are the same for many/all Saturn-based machines)

> **Quote:**
> But **how close** can two consecutive **RND**s actually be ?

As a consequence of the above analysis, it could be possible to get two consecutive RND values that are equal.
Matter of fact, I have one example. Can you find it? :-)

J-F

EMAIL PM WWW FIND QUOTE REPORT

6th April, 2024, 03:16 **Post: #8**

Juan14
Junior Member

Posts: 46
Joined: Jan 2014

RE: [VA] SRC #017 - April 1st, 2024 Spring Special

For the function S(X); notice that:
1). 2^(12N+4) = 2^(12N)*16, so 16 can go outside the summation.
2). 2^(12(N+1))/2^(12N) = 2^12 = 4096.
3). COMB(2N, N) = (2*N)!/(N!^2).
COMB(2(N+1), N+1)/COMB(2N, N) = ((2*(N+1))!/((N+1)!^2))/((2*N)!/(N!^2))
= (4*N+2)/(N+1) = 2*(2-1/(N+1)).
Having (COMB(2N, N)^3)/(2^(12N)) stored in register 03, you can get to (COMB(2(N+1), N+1)^3)/(2^(12(N+1)) by multiplying register 03 by ((2*(2-1/(N+1)))^3)/4096
= ((2-1/(N+1))^3)/512.
With X in register 01, N+1 in register 02 and summation in register 04, we have the next program for the free42:

PHP Code:
```
01 LBL “S”
02 STO 01
03 0
04 STO 02
05 1
06 STO 03
07 5
08 STO 04
09 LBL 00
10 2
11 1
```
S(42) = 3.141592653589793238462643383279504
wow, pi again!

EMAIL PM FIND QUOTE REPORT

6th April, 2024, 12:30 (This post was last modified: 6th April, 2024 19:09 by J-F Garnier.) **Post: #9**

J-F Garnier
Senior Member

Posts: 940
Joined: Dec 2013

RE: [VA] SRC #017 - April 1st, 2024 Spring Special

> **Valentin Albillo Wrote:** (1st April, 2024 20:59)
> ***how close*** *can two consecutive* ***RND****s actually be ?*
> Use **RANDOMIZE 1** or the equivalent instruction to specify the value **1** as the seed and write code to output the list of ever closer consecutive **RND**s and their index **N** in the sequence

OK, let's try the brute force on a super-fast HP-71B emulator:

```
10 RANDOMIZE 1 @ X=RND @ I=1 @ M=1
20 Y=RND @ I=I+1 @ IF ABS(X-Y)>=M THEN X=Y @ GOTO 20
30 M=ABS(X-Y) @ PRINT I,X,Y,M @ GOTO 20
```

2 .731362440213 .77207218067 .040709740457
13 5.64471991805E-2 6.30768172146E-2 6.6296180341E-3
125 .805774019056 .803607575861 .002166443195
316 .128424219936 .126476247276 .00194797266
378 .128629571043 .127765838222 .000863732821
1746 .657235932954 .658084547243 .000848614289
...
later, much later ...
38181163 .151576837566 .151576827517 1.0049E-8
... and even much later
157373808 .477539626899 .477539622968 3.931E-9
322950317 .325324468276 .325324470156 1.88E-9
... getting closer to the record:

431380423 .275319512003 .275319511289 7.14E-10
...[edit:]got it:
1838286534 .564079556829 .564079556839 **1.E-11**

[edit2:]
Sopped with index exceeding 5 x 10^9 with no better result.
No identical consecutive RNDs found, but only a small fraction of the RND period has been explored.

J-F

EMAIL PM WWW FIND QUOTE REPORT

7th April, 2024, 05:01 (This post was last modified: 8th April, 2024 00:25 by Valentin Albillo.) **Post: #10**

Valentin Albillo
Senior Member

Posts: 1,100
Joined: Feb 2015
Warning Level: 0%

RE: [VA] SRC #017 - April 1st, 2024 Spring Special

Hi, **all**,

I've got already fully formatted and *ready-to-post* my looong *Solutions* post (I might actually split it in three parts ... or not) so this is the **last chance** for those of you who still want to post something (code & results, comments) before I post my original solutions & comments revealing it all. The status of things done or still pending is as follows:

- **LOL the First**: Several people have posted code but so far only one of the 4 squares has been computed and posted, all other three are still pending, and no comments whatsoever on the nice pattern they all exhibit or on the code posted.

- **LOL the Second**: Only one entry so far. More detail or comments on it ? Any additional takers ?

- **LOL the Third**: This was well covered by **J-F**, interesting and revealing comments.

- **LOL the Fourth**: Again, this one has been well covered by **J-F** but no one else commented anything or offered any results for the questions asked.

- **LOL the Fifth**: Only **Gerson** offered some hints about this one but he posted neither code nor results, and no one else posted a thing.

- **LOL the Sixth**:

    1. Very little detail or comment and the result has not been posted.

    2. Ditto, only minimal detail and no explanation or comments posted.

    3. No one posted any comments or similar nice results, it's been totally ignored.

    4. Only the briefest of comments (like *Earth*'s entry in *Hitchhiker*: *"Mostly harmless"*) and no explanations.

    5. An additional mention of a similar case but no explanations for any of them.

    6. This has been more successful, with working code posted but only the one result.

That's all. If you need a few more days to finish something, just say it and I'll oblige. Frankly, I'm surprised at the lack of entries and comments, I deemed all sections interesting and perfectly within the reach of most calc programmers with minimal effort. Live and learn !

Waiting for your *11th-hour* productions,
V.
Edit: typo

WWW FIND EDIT QUOTE REPORT

8th April, 2024, 01:10 (This post was last modified: 8th April, 2024 05:12 by Gerson W. Barbosa.) **Post: #11**

Gerson W. Barbosa
Senior Member

Posts: 1,580
Joined: Dec 2013

RE: [VA] SRC #017 - April 1st, 2024 Spring Special

> **Valentin Albillo Wrote:** (7th April, 2024 05:01)
>
> [*] **LOL the Fifth**: Only **Gerson** offered some hints about this one but he posted neither code nor results, and no one else posted a thing.

Hello, Valentín,

Unusually busy week here. I'll try to fix that, even if only by a litte.

As I have suggested, that has to do with the Euler-Mascheroni constant, denoted by **γ** (lower-case gamma) .

The wp34s has **γ** built-in, but I guess Free42 should be more appropriate here. The following table, produced with help of one small RPN program, illustrates our attempt to get as many digits as possible on Free42, using that approach:

```
     γ = 0.5772156649015328606065120900824024

 1 - γ = 0.4227843350984671393934879099175976

      N      FRAC(Γ(10⁻ᴺ))          ABS(1 - γ - FRAC(Γ(10⁻ᴺ)))
    --------------------------------------------------------
      1  0.5135076986687318362 9        9.07233635703E-02
      2  0.4325851191506037135 3        9.80078405214E-03
      3  0.4237724845954661149 8        9.88149496999E-04
      4  0.4228832316241908057 4        9.89965257237E-05
      5  0.4227942255676734932 3        9.89046920635E-06
      6  0.4227853241535549892 7        9.89055087500E-07
      7  0.4227844340040575974 0        9.89055904580E-08
      8  0.4227843449890270019 3        9.89055986253E-09
      9  0.4227843360875231338 1        9.89055994420E-10
     10  0.4227843351973727389 2        9.89055995237E-11
     11  0.4227843351083576993 5        9.89055995288E-12
     12  0.4227843350994561953 9        9.89055997912E-13
     13  0.4227843350985660449 5        9.89055555121E-14
     14  0.4227843350984770299 9        9.89059651209E-15
     15  0.4227843350984681235          9.84106512090E-16
     16  0.4227843350984672 42          1.02606512090E-16
     17  0.4227843350984668 4           2.99393487910E-16
    --------------------------------------------------------
```

From the table, we notice that the constant approaches **1 - γ** as **N** increases. The maximum accuracy is reached when **N** = 16, when the first 15 digits are correct. From this point on, the accuracy degrades, as the Free42 precision is limited to 34 digits.

Here we also notice that the mantissas of the errors, the second column in the table, appear to tend to another constant. Regardless of any attempt to identify it, we can try to use it to get a few more correct digits, starting with **N** = 16 and down to **N** = 13, when the maximum accuracy is reached:

```
     16  0.422784335098467242 - 9.89055997912E-17 = 0.4227843350984671430944002088
     15  0.4227843350984681235 - 9.89055997912E-16 = 0.422784335098467134444002088
     14  0.42278433509847702999 - 9.89055997912E-15 = 0.42278433509846713943002088
     13  0.422784335098566044949 - 9.89055997912E-
14 = 0.42278433509846713934 92088
     12  0.4227843350994561953914 - 9.89055997912E-13 = 0.422784335098467139393488
```

Now we have about 50% more correct digits, 23. This scheme should give up to eight or nine correct digits on the HP-42S, but I have it to check it out.

Best regards,

Gerson.

```
00 { 58-Byte Prgm }
01▸LBL "L5th"
02 1
03 0.5772156649015328606065120900824024
04 -
05 X<>Y
06 +/-
07 10↑X
08 GAMMA
09 FP
10 -
11 LASTX
12 X<>Y
13 ABS
14 END
```

P.S.:

By using the constant **9.89055995288** we can get up to 7 correct digits of **γ** on the HP-42S and up to 23 on Free42:

```
00 { 38-Byte Prgm }
01▸LBL "E_M"
02 RCL ST X
03 +/-
04 10↑X
05 GAMMA
06 FP
07 X<>Y
08 NOT
09 10↑X
10 9.89055995288
11 ×
12 -
13 +/-
14 1
15 +
16 END

4 XEQ "E_M" ->
```

0.5772156756

```
11 XEQ "E_M" ->
```

0.57721566490153286060651

EMAIL PM FIND QUOTE REPORT

8th April, 2024, 10:34 **Post: #12**

JoJo1973
Member

Posts: 111
Joined: Apr 2016

RE: [VA] SRC #017 - April 1st, 2024 Spring Special

LOL the Sixth:

I've no HP-71B, but I had a Commodore 64 so I can throw a couple of guesses:

LOL 6.2:

A famous challenge in those naive days was to write a program resulting in the largest number of errors. Of course the easiest way to accomplish the task is to dump the ROM area where the messages are actually stored and embellish the output to make it look like a real error message...

LOL 6.5:

The Commodore 64's screen editor doesn't accept more than 80 characters when editing a program line: there are a lot of dirty tricks to circumvent this limitation, but of course one always starts with the cleanest ones.

For example you could omit useless blanks between instructions and gain a little space to squeeze that extra statement... but sometimes you end up with an ambiguous statement that leaves the parser (and you!) scratching its head in puzzlement:

FORM=STOP is a variable assignment or the beginning of a loop: FOR M = S TO P ?

Cheers!

EMAIL PM FIND QUOTE REPORT

9th April, 2024, 19:19 (This post was last modified: 10th April, 2024 03:50 by Gerson W. Barbosa.) **Post: #13**

Gerson W. Barbosa
Senior Member

Posts: 1,580
Joined: Dec 2013

RE: [VA] SRC #017 - April 1st, 2024 Spring Special

> **Gerson W. Barbosa Wrote:** (8th April, 2024 01:10)
> By using the constant **9.89055995288** we can get up to 7 correct digits of **γ** on the HP-42S and up to 23 on Free42:
>
> ```
> 00 { 38-Byte Prgm }
> ```

```
01▸LBL "E_M"
02 RCL ST X
03 +/-
04 10↑X
05 GAMMA
06 FP
07 X<>Y
08 NOT
09 10↑X
10 9.89055995288
11 ×
12 -
13 +/-
14 1
15 +
16 END

4 XEQ "E_M" ->

0.5772156756

11 XEQ "E_M" ->

0.57721566490153286060651
```

The constant actually goes like

K = **9.890559953279725553953956515…**

But there's a better way to get those extra digits. We only have to use the GAMMA function twice:

```
00 { 29-Byte Prgm }
01▸LBL "E_M"
02 1
03 -11
04 10↑X
05 GAMMA
06 LASTX
07 R↓
08 FP
09 -
10 R↑
11 +/-
12 GAMMA
13 FP
14 -
15 2
16 ÷
17 END
```

This will return

0.577215664901532860606565,

good to 22 digits, but nine bytes shorter.

Or we can add a couple of steps and get all 34 digits right, at the cost of another nineteen bites:

```
17 5.29099175976ᴇ-23
18 -
```

But then again the following should be more simple and two bytes shorter:

```
00 { 46-Byte Prgm }
01▸LBL "E_M"
02 5.772156649015328606065120900824024ᴇ-1
03 END
```

Update

If all we want to do is to obtain **γ** from **Γ** then this 42-byte Free42 program is better:

```
00 { 42-Byte Prgm }
01▸LBL "E_M"
```

```
02 -11
03 10↑X
04 GAMMA
05 LASTX
06 +/-
07 GAMMA
08 +
09 -2
10 ÷
11 5.29099175976E-23
12 -
13 END

XEQ "E_M" ->

5.772156649015328606065120900824024E-1
```


10th April, 2024, 22:51 **Post: #14**

Valentin Albillo
Senior Member

Posts: 1,100
Joined: Feb 2015
Warning Level: 0%

RE: [VA] SRC #017 - April 1st, 2024 Spring Special

Hi, **all**,

First of all, thanks to all of you who were interested in this *April 1st* thread and in particular to those who posted code, results and/or comments, namely **J-F Garnier**, **Juan14**, **C.Ret**, **John Keith**, **Gerson W. Barbosa** and **JoJo1973**. Thank you very much for your interest and continued appreciation.

However, I feel somewhat let down by the fact that only ***6*** people ***6*** took the trouble to post *anything* at all (out of the *~9,400* members currently registered,) but that's life and now it's time for my original solutions and comments, which for sheer message-length reasons I'll post **two *LOL* at a time**, so let's get the party started with the first two, i.e. ***LOL the First: Squares*** and ***LOL the Second: GCD*** ...

1. LOL the First: Squares

This mini-challenge asks you to calculate the following 2^2 squares. You must program your own multiprecision squaring routine in your vintage HP calc and use it to get the results.

$$130834904430015239^2 = ?$$
$$4712877147889716634938899^2 = ?$$
$$25781108305591628417975738^2 = ?$$
$$14142208287606721994980505000 5^2 = ?$$

My original solution is this *10-line*, *415-byte* **HP-71B** user-defined function, which accepts the number to be squared as a *string* and returns its square as another *string*:

```
 1  DEF FNS$[200](P$) @ STD @ DIM M,L,P,I,J,U,A$[206]
 2  OPTION BASE 1 @ A$=FNL$(P$) @ U=LEN(A$) DIV 6 @ K=10^6
 3  DIM A(U),C(2*U),C$[12*U] @ MAT A=ZER @ MAT C=ZER
 4  FOR I=1 TO U @ A(U+1-I)=VAL(A$[I*6-5,I*6]) @ NEXT I @ FOR I=1 TO U
 5  M=A(I) @ L=I @ FOR J=1 TO U @ P=M*A(J) @ C(L)=C(L)+P @ P=RES
 6  IF P>=K THEN C(L)=RMD(P,K) @ C(L+1)=C(L+1)+P DIV K
 7  L=L+1 @ NEXT J @ NEXT I @ I=2*U+1 @ REPEAT @ I=I-1 @ UNTIL C(I)
 8  C$=STR$(C(I)) @ FOR I=I-1 TO 1 STEP -1
 9  C$=C$&FNL$(STR$(C(I))) @ NEXT I @ FNS$=C$ @ END DEF
10  DEF FNL$[206](A$) @ P=RMD(LEN(A$),6) @ FNL$=RPT$("0",6*(P#0)-P)&A$
```

Note: Branching isn't needed thanks to the keywords provided by the **JPC** *ROM* (**REPEAT..UNTIL**). It also uses **MAT..ZER** from the **Math** *ROM* and **RPT$** from the ***STRNGLEX*** *LEX* file for speed and convenience, all of them replaceable by slower *BASIC* code if necessary.

- Lines *1-3* do the initialization.
- Line *4* splits the input string into an array of 6-digit numeric values.
- Lines *5-7* perform the multiprecision squaring and locate the leftmost *nonzero* element.
- Lines *8-9* conjoin the result array elements into a result string and return it.
- Line *10* is an auxiliary user-defined function which adds needed leftmost *0*s, if any.

My program chops the input into *6-digit* values (multiplying two 6-digit numbers never exceeds the *12-digit* range of the **HP-71B**) instead of doing the squaring digit by digit, which would be about *36x* more processing and so *~30x* slower.

Also, using strings for input/ouput is extremely convenient for multiprecision values, as the user simply enters the value between quotes and the result can be directly stored in a string variable for further processing e.g. combined with other similar multiprecision *UDF*s. For instance, raising a multiprecision value to the fourth power is as simple as **FNS$(FNS$(*value*))**

Finally, there's no limit to the size of the result squares other than maximum string length (*65,500+* characters i.e. digits). As listed above, it allows for up to *200-digit* squares but that limit can be changed by simply replacing the constants ***200*** and ***206*** by the desired maximum size.

Now let's use **FNS$** from the command line to quickly compute the requested squares:

>DESTROY ALL
>FNS$("130834904430015239")

171177722172112212111172177772227121

>FNS$("471287714788971663493899")

222112110111011100020110111110102200012010222201

>FNS$("257811083055916284179757738")

66466554546464564566564664466556465454664555664464 4

>FNS$("1414220828760672199498050500 05")

2000020552500522520200050520222252022220220520500055050 0025

As you can see, these beautiful large squares contain solely the digits *(1,2,7)*, *(0,1,2)*, *(4,5,6)* and *(0,2,5)*, respectively. Alas, they aren't the only such squares, see if you can find some more !

Additional comments: No one posted all four squares so three of them were left missing (though once you've created your multiprecision squaring program it's all too easy to use it to compute the squares and post them) and no one posted any more examples of this pattern so here you are, a few more:

$77470059130002034719700749^2$ =

600161006160601161661106000601066100061610011116 1001

$94257542957794332698779 8^2$ =

88844844044404440008044044444040880004804088880 4

2. LOL the Second: GCD

*You may remember the **GCD** function. In this mini-challenge you must write code for any HP calc to find out the answer to this simple question:*

As we have that ***GCD(15, 4) = 1*** *and* ***GCD(15, 5) = 5****, for what value of* ***x*** *is* ***GCD(15, x) = 2*** *?*

In the question above there's no mention of the **GCD** <u>*keyword*</u> from the ***JPC*** *ROM*, which is only available for the **HP-71B** and is limited to *integer* arguments, while this mini-challenge (which can be solved using almost *any* programmable scientific *HP* calc, from the **HP-19C/29C** upwards) deals with the **GCD** *(Greatest Common Divisor)* <u>*math function*</u>.

Mathematically, the **GCD** of two *integers* is the largest positive *integer* that divides each of them. But as it happens with the *factorial*, which can be extended to non-integer arguments via the Γ function, or with the Harmonic series, for which the digamma function does likewise, the **GCD** function can also be extended to *non-integer* arguments, and even to *complex* ones. This is done using this simple formula (which you can find in *Wikipedia*, no need to hunt obscure scholar papers for it):

$$\mathrm{GCD}\,(n,m) = \log_2 \prod_{k=0}^{n-1} (1 + e^{-2i\pi km/n}) \qquad \mathrm{odd}\, n \geq 1$$

where ***n*** has to be odd and ≥ **1** but ***m*** can be *complex*.

This *2-line, 89-byte* **HP-71B** user-defined function accepts a *complex* argument ***K*** and a positive *odd integer* argument ***N*** and returns ***GCD(K,N)*** as per the above formula:

```
5  DEF FNG(K,N) @ COMPLEX P,R @ P=1 @ R=-2*PI*K/N @ FOR M=0 TO N-1
6  P=P*(1+RECT((1,R*M))) @ NEXT M @ FNG=LOG(P)/LOG(2) @ END DEF
```

We can check that it works fine by using this *4-line*, *106-byte driver* program to test it for 100 pairs of random integer arguments, excluding *co-prime* pairs from the listing. The first integer is passed as a *complex* value and the second integer is forced to be *odd*:

```
1  DESTROY ALL @ RANDOMIZE 1 @ RADIANS
2  FOR I=1 TO 100 @ A=INT(RND*50)+1 @ B=2*INT(RND*25)+1
3  G=IROUND(REPT(FNG((A,0),B))) @ IF G#1 THEN DISP USING "4X,4D,4D";A,B,G
4  NEXT I @ END
```

```
>RUN
       6   3       3
      38  19      19
      22  33      11
       9  21       3
      30  45      15
        ...
```

so it seems to be working Ok. Of course we can also call the *UDF* from the keyboard, e.g.:

```
>FNG((30,0),45)  ->  (15.000000002, 3.81114434334E-9), i.e. 15
>FNG((1/2,0),3)  ->  (1.79248125035, -2.26618007092)
```

Now, for real arguments we can use ***cos x = ½·(e^{ix} + e^{-ix})*** to reduce the formula to the real product:

$$\mathrm{GCD}(n,m) = n + \log_2\left(\prod_{k=1}^{(n-1)/2} \cos\frac{km\pi}{n}\right)^2 \qquad \text{odd}\, n \geq 1$$

The user-defined function now becomes simpler (*71 bytes*, no comples variables or functions) and faster:

```
5  DEF FNG(N,M) @ P=1 @ R=PI*M/N @ FOR K=1 TO (N-1)/2
6  P=P*COS(R*K) @ NEXT K @ FNG=N+LOG2(P*P) @ END DEF
```

> **Note**: do *not* "optimize" **LOG2(P*P)** to **2*LOG2(P)**

Let's apply it to this mini-challenge's question. First, let's evaluate ***GCD(15, x)*** for ***x*** in *[4..5]* at steps of *0.2*:

```
>DESTROY ALL
>FOR I=4 TO 5 STEP .2 @ DISP USING "3X,D.D,2X,S2D.7D";I,FNG(15,I) @ NEXT I

  x     GCD(15, x)
------------------
 4.0    +1.0000000  <- GCD(15,4) = 1
 4.2    +3.0660268
 4.4    +1.6288312
 4.6    +2.2305063
 4.8    +5.0211098
 5.0    +5.0000000  <- GCD(15,5) = 5
```

and a cursory inspection reveals that there are at least *three* solutions to the original question *"For what value of **x** is **GCD(15, x) = 2** ?"*. Let's find one of them using this *1-line* main program together with the *UDF*:

```
1  DESTROY ALL @ RADIANS @ STD @ DISP FNROOT(4,5,FNG(15,FVAR)-2)
5  DEF FNG(N,M) @ P=1 @ R=PI*M/N @ FOR K=1 TO (N-1)/2
6  P=P*COS(R*K) @ NEXT K @ FNG=N+LOG2(P*P) @ END DEF
```

```
>RUN
```

x_3 = **4.59247568122**

Now we can check that this value of ***x*** really makes ***GCD(15, x) = 2*** :

```
>FNG(15,RES)  ->  2
```

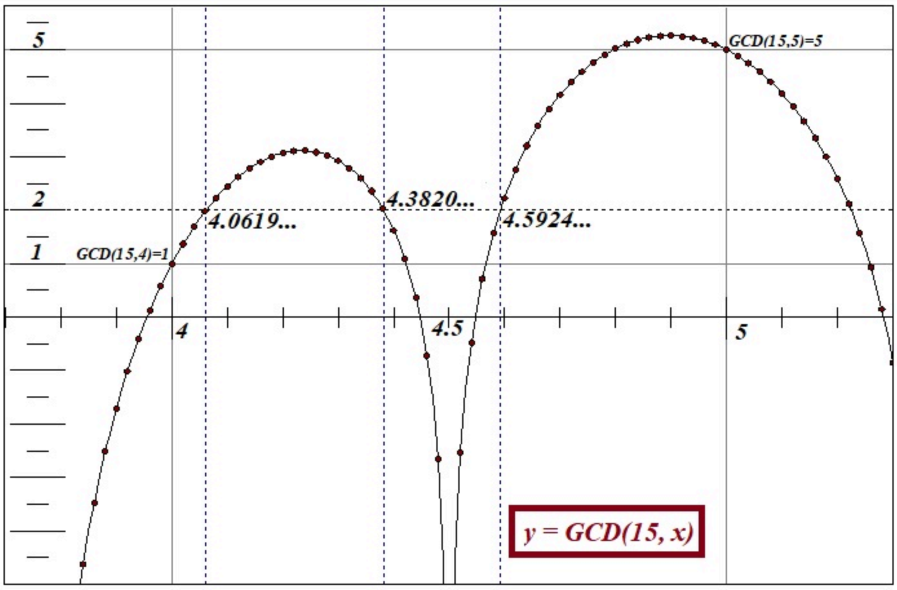
In the table above we can see that there are likely *two* other solutions for ***x***, one in *[4.0, 4.2]* and another in *[4.2, 4.4]*, namely:

FNROOT(4.0, 4.2 ...) -> x_1 = **4.06191388928**, FNG(15,RES) = 2.0000000002

FNROOT(4.2, 4.4 ...) -> x_2 = **4.38207921034**, FNG(15,RES) = 1.9999999996

and that completes all three solutions to this mini-challenge in the interval *[4, 5]*, which you can see listed with 20 decimal digits in the comments below.

Additionally, they are duly located and labeled in my graph of ***y = GCD(15, x)*** right below, where you can also see the perhaps *unexpected* fact that ***GCD(15, 4.5)*** tends to **-∞** because in this case the *cosine product* includes one cosine which is ***0***, thus so is the product itself and its $\log_2$ tends to **-∞**. Notice theres's a *fourth* solution just outside the *[4..5]* interval:



Additional comments: These are the three solutions computed using the **HP-42S** *Solver* via the 34-digit ***Free42*** *Decimal* simulator:

```
01  LBL "GCD15"   13   x          01  LBL "LOL2"
02  PI            14  DSE 00      02  MVAR "X"
03   x            15  GTO 00 ►    03  RCL "X"
04  15            16  X^2         04  XEQ "GCD15"
05   ÷            17  LOG         05   2
06   7            18   2          06   -
07  STO 00        19  LOG         07  END
08  SIGN          20   ÷
09 ►LBL 00        21  15
10  RCL 00        22   +
11  RCLx ST Z     23  END
12  COS

[SOLVER]  ->  Select Solve Program
Touch [LOL2], 4.0, [X], 5.0, [X], [X]  ->  4.59247568122 (...121 559541826...)
              4.0, [X], 4.2, [X], [X]  ->  4.06191388926 (...926 216056306...)
              4.2, [X], 4.4, [X], [X]  ->  4.38207921034 (...034 278186774...)
```

You can press **[SHOW]** after each solution to see up to 34 digits (at least 30 will be correct.) The fourth "solution" just outside the *[4..5]* interval is at x_4 = **5.22459905977** *(222552325...)*.

That's all for now. I'll post the next two **LOL**s in a couple' days or so. Meanwhile, let's see your comments.

Regards.
V.

WWW FIND EDIT QUOTE REPORT

15th April, 2024, 22:17 (This post was last modified: 15th April, 2024 22:48 by Gerson W. Barbosa.) **Post: #15**

Gerson W. Barbosa
Senior Member

Posts: 1,580
Joined: Dec 2013

RE: [VA] SRC #017 - April 1st, 2024 Spring Special

> **Valentin Albillo Wrote:** (10th April, 2024 22:51)
>
> **Additional comments:** No one posted all four squares so three of them were left missing (though once you've created your multiprecision squaring program it's all too easy to use it to compute the squares and post them) and no one posted any more examples of this pattern so here you are, a few more:
>
> $77470059130002034719700749^2$ =
>
> **6001610061606011616611060000601066100061610011116100 1**
>
> $94257542957794332698779 8^2$ =
>
> **8884484404440444000804404444404088000480408888 04**

Here's a less sloppy HP-75 program with all four squares plus one found at OEIS. When I was editing it, I noticed there was a typo in line 115 of my previous program, current line 125: the second **2** should be **1**. The numbers to be squared are placed in DATA lines, divided in chunks of six digits with no leading zeros, preceded by the number of chunks.

```
10 OPTION BASE 0
15 INTEGER I,J,K,L,N
25 B=1000000
30 DIM A(9),B(5),C(12)
35 REM DIM A(N+4), B(N), C(2*N+2); N>1
40 FOR K=1 TO 5
45 READ N
50 FOR I=0 TO 2*N+2 @ C(I)=0 @ NEXT I
55 FOR I=0 TO N+4 @ A(I)=0 @ NEXT I
60 FOR I=1 TO N
65 READ A(I)
70 B(I)=A(I)
75 NEXT I
80 FOR I=N TO 1 STEP -1
85 T1=0 @ A1=0
90 FOR J=N-I+4 TO -1 STEP -1
95 C2=(T1+B(I)*A(J+1))/B
100 A1=FP(C2)*B
105 C(I+J)=C(I+J)+A1
110 T1=IP(C2)
115 NEXT J
120 NEXT I
125 FOR I=2*N TO 1 STEP -1
130 T=C(I)/B
135 C(I)=FP(T)*B
140 C(I-1)=C(I-1)+IP(T)
145 NEXT I
150 L=1
155 IF C(0)<>0 THEN DISP STR$(C(0));ELSE DISP STR$(C(1)); @ L=2
160 FOR I=L TO 2*N-1
165 C$=STR$(C(I)) @ C$=RIGHT$(C$,LEN(C$))
170 IF 6-LEN(C$)=0 THEN GOTO 185
175 C$="0"&C$
180 GOTO 170
185 DISP C$;
190 NEXT I
195 DISP
200 NEXT K
205 END
210 DATA 3,130834,904430,15239
215 DATA 4,471287,714788,971663,493899
220 DATA 5,25,781108,305591,628417,975738
225 DATA 5,141422,82876,67219,949805,50005
230 DATA 5,100,990098,979999,970099,500001

>RUN
171177722172112212111172177722227121
2221121101110111000201101111101022000120102222 01
```

Edited to remove an unnecessary BASIC line

EMAIL PM FIND QUOTE REPORT

16th April, 2024, 08:07 **Post: #16**

EdS2
Senior Member

Posts: 582
Joined: Apr 2014

RE: [VA] SRC #017 - April 1st, 2024 Spring Special

Oh, a nice implied pair of challenges at a page linked from that OEIS entry - no squares known consisting only of digits 0,1,3 or 6,7,8.

EMAIL PM FIND QUOTE REPORT

17th April, 2024, 03:47 **Post: #17**

Valentin Albillo
Senior Member

Posts: 1,100
Joined: Feb 2015
Warning Level: 0%

RE: [VA] SRC #017 - April 1st, 2024 Spring Special

Hi, all,

> **VA (i.e. me) Wrote:**
>
> I'll post the next two LOLs in a couple' days or so. Meanwhile, **let's see your comments**.

Well, it seems that I was being overoptimistic, as usual, because after *5 days 5* elapsed only the illustrious **Gerson W. Barbosa** bothered to post an interesting new **HP-75C** program to solve ***LOL 1*** and *(drum roll)* also explicitly listed all four beautiful squares I wanted everyone to behold, plus a bonus *fifth* square in the same fashion. Thanks a lot, **Gerson**, and here you are, another bonus square featuring π no less than three times i.e. at the very beginning, in the middle and near the end, a worthy apropos appearance:

$177324875114669443080086908188^2$ =

314441113344331143341411331434444443134341114311131414443344

Now, these are my original solutions and comments for the next two sections, i.e. ***LOL the Third: Random*** and ***LOL the Fourth: Logs***. The show must go on ...

3. LOL the Third: Random

This mini-challenge's question is:

> ***How close*** *can two consecutive* ***RND****s actually be ? Use* **RANDOMIZE** *1 as the seed and write code to output the list of ever closer consecutive* ***RND****s and their index* ***N*** *in the sequence.*

My original solution is this *3-line*, *89-byte* never-ending program which will produce the goods:

```
1  DESTROY ALL @ RANDOMIZE 1 @ STD @ N=0 @ X=99 @ L=1
2  N=N+1 @ Y=RND @ D=ABS(X-Y) @ IF D<L THEN DISP N;X;Y;D @ L=D
3  X=Y @ GOTO 2
```

```
>RUN
       N     First RND          Next RND          |Difference|
 ------------------------------------------------------------
         2  .731362440213     .77207218067      .040709740457
        13   5.64471991805E-2  6.30768172146E-2  6.6296180341E-3
       125  .805774019056     .803607575861     .002166443195
       316  .128424219936     .126476247276     .001947972660
       378  .128629571043     .127765838222     .000863732821
      1746  .657235932954     .658084547243     .000848614289
      1864  .724574750035     .724711386925     .000136636890
      3091  .804652037305     .804530031878     .000122005427
      4983  .900183907166     .900119502104     .000064405062
      5002  .185041013513     .184988311770     .000052701743
      5964  .350363678669     .350333411003     .000030267666
     33971  .800461443203     .800483558857     .000022115654
     56943  .322507676917     .322505089514     .000002587403
    144113  .468047416778     .468049379480     .000001962702
    192237  .771445619886     .771443980639     .000001639247
   1606781  .176400853304     .176399416567     .000001436737
   1732702   3.20935575951E-2  3.20944940649E-2  9.364698E-7
   1840905  .862173808769     .862174478752     .000000669983
   1969683  .573449101200     .573448667982     .000000433218
   2143212  .948380454319     .948380365276     .000000089043
  10684317  .555880267196     .555880325796     .000000058600
  38181163  .151576837566     .151576827517     .000000010049
 157373808  .477539626899     .477539622968     .000000003931
 322950317  .325324468276     .325324470156     .000000001880
 431380423  .275319512003     .275319511289     .000000000714
1838286534  .564079556829     .564079556839     .000000000010
```

at which time I interrupted the search, with my record being the last line shown in the output, i.e. two *consecutive RND*s, namely **0.564079556829** and **0.564079556839** , which differ by just *0.00000000001*.

Of course this finding took *waaaay* long to run even on a very fast emulator, so eventually I had to stop the search without looking at the *entire* one-trillion-long sequence. Thus, I can't confirm whether *closer* consecutive *RND*s are possible or not in *this* specific sequence generated by the seed ***1***, in particular consecutive ones *identical* to *12-digit* accuracy (i.e. difference = ***0***).

Also, I feel that there's something *eerie (IMHO)* in seeing two consecutive *RND*s come out as the almost identical value (or even *identical* just on screen if you use **FIX 4** or **FIX 6**, say), so if you want to experience this feeling yourself try this *4-line*, *149-byte ad-hoc* variant of the above program:

Note: *Yes, I know that you can go directly to the vicinity of the two close consecutive RNDs using* **PEEK$** *and* **POKE** *so saving much running time but I like it better this way.*

```
1  DESTROY ALL @ RANDOMIZE 1 @ STD @ N=0 @ X=99 @ L=1 @ INPUT "N=";M
2  IF N=M-2 THEN DISP @ DISP "Execute RND ; RND ..." @ PAUSE
3  N=N+1 @ Y=RND @ D=ABS(X-Y) @ IF D<L THEN DISP N;X;Y;D @ L=D
4  X=Y @ GOTO 2
```

Now, in the list above we have this line:

1864 .724574750035   .724711386925   .000136636890

so to see by yourself that those two very close *RND*s are indeed produced consecutively at that point in the sequence, do the following:

>RUN -> **N=** *{ key in **1864** and [END LINE] }*
-> *{ normal output as above, until ... }*
-> **Execute RND ; RND ...** *{ the program stops; execute the following: }*

>FIX 4 @ RND;RND -> **0.7246** **0.7247**

or for a *much closer* pair of consecutive *RND*s:

>RUN -> **N=** *{ key in **56943** and [END LINE] }*
-> *{ normal output as above, until ... }*
-> **Execute RND ; RND ...** *{ the program stops; execute the following: }*

>FIX 6 @ RND;RND -> **0.322508** **0.322505**

and last, still closer so even more impressive:

>RUN -> **N=** *{ key in **10684317** and [END LINE] }*
-> *{ normal output as above, until after a really, really long while ... }*
-> **Execute RND ; RND ...** *{ the program stops; execute the following: }*

>FIX 7
>RND -> **0.5558803**
>RND -> **0.5558803**

and *le voilà!*. Reminds me of Groundhog Day, it looks like **RND** is broken ...

Additional comments: J-F Garnier did his best to try and solve this mini-challenge. He used a *"super-fast"* **HP-71B** emulator with his own 3-line *BASIC* program which was very similar to my own original solution above (although he forgot to include the nearly-mandatory **DESTROY ALL** statement at the very beginning) and let it run for presumably large amounts of time until at last he *exactly matched* my own record, namely:

```
1838286534   .564079556829     .564079556839     1.E-11
```

but although he let it run for *5 billion*-deep values in the sequence before stopping it for good, he was unable to get two *identical* (to 12-digit) consecutive random numbers in this particular sequence created by the original seed ***1***, as he theorized to be possible. He adds:

"but only a small fraction of the RND period has been explored"

and I agree, as *5 billion* is only ***0.5%*** of the whole *1-trillion*-strong sequence so *99.5%* of it remains unexplored.

Last but absolutely not least, **J-F** left some intriguing observations which he never fully developed. For example, he said:

*"The RND value is determined by the internal seed, which is stored with 15 digits. Or, in other words,we can't predict the next RND value from the last one (well, **the possibilities for the next RND are limited**)."*

but he *never* explained the underlying algorithm used and *why* and *how* the possibilities for the very next *RND* value are limited. Knowing that, perhaps it'd be much easier to see if two consecutive values can indeed be identical to 12-digit accuracy or not, without having to generate and check tens or hundreds of *billions* of *RNDs*.

He also posted two sequences generated by different seeds which include the *same 11-digt* (not *12-digit*) value *.14159265359* but he didn't tell how he found those particular seeds, nor did he explain why the very next *RND* values after the *.14159265359* do differ from their *second* decimal digit on (namely *.**4**94478890124* and *.**4**1675139916*,) while it would seem that, as the possibilities are *"limited"*, they should be much closer and not differ so markedly.

Adding to the mysteries, **J-F** also said:

*"[...] It could be possible to get two consecutive RND values that are equal. Matter of fact, **I have one example**. Can you find it? :-)"*

and again he never bothered to post that *"one example"* he found nor to explain how did he find it.

In short: for sure **J-F** is under no obligation whatsoever to share or post anything at all but I've always thought that one of the goals of these mini-challenges is to provide entertainment while also introducing useful math concepts & programming techniques to learn from. So, having at hand the solutions by me and by others serves as an effective way of *sharing* knowledge, but if you won't share what you know or what you found, then *what's the point ?* I don't give prizes, you know.

Frankly, I was expecting **J-F** to eventually provide answers to the above matters so that me and other interested people would learn something new and be enlightened in the process, but to my big surprise he never did. At least so far, several weeks *(as of 2024-04-17)* since he posted his message. As *Chloe B.* would say: *"C'est décevant, totalement décevant"*. 🙂

4. LOL the Fourth: Logs

This wasn't a mini-challenge *per se* but simply me reporting an *unusual* finding, namely that $\mathbf{LOG_{10}}$ *seemed* to be the most accurate logarithm available in some *HP vintage* calcs (namely the *10-digit* **HP-15C** and the *12-digit* **HP-71B**). I then suggested the following for interested people to explore and post their findings:

- *Try the above examples in your HP models, both 10-digit and 12-digit, from the first **HP-35** up to the latest RPL models, to see if $\mathbf{LOG_{10}}$ and **LN** differ that much in their respective errors.*

- *Test other range of values (here from 5^2 = **25** to 5^9 = **1,953,125**).*

- *Find out if there are arguments with even larger errors.*

Regrettably, only **J-F Garnier** investigated the matter in this excellent post of his, coming to the conclusion that *"we can't say that the decimal LOG provides more accurate results than the natural LN, overall"*, conclusion with which I mostly agree.

He also asks a related question which is pending further investigation but this ***LOL the Fourth*** seems to have been largely *ignored* so far (**J-F** excepted, of course,) though I think it's an interesting topic related to the innards of *HP* algorithms and even perhaps to their architecture. And answering **J-F**'s question, no, I couldn't find any references in older threads either nor do I remember this topic having been discussed here ever. I took my observations from ancient *(20-25 year-old or more)* notes which I wrote at the time but never published before.

Enough for now. I'll post my original solutions and comments to the two *final* ***LOL 5*** and ***LOL 6*** *sections* either in an unspecified number of days *(a few, a week, a month ...)* or right after the next comment(s), whichever comes first.

In the meantime, you might want to have a look at some of my previous ***April 1st*** challenges, they feature a true plethora of interesting topics, solutions and comments which are sure to keep you entertained while you wait:

SRC #011 - April 1st, 2022 Bizarro Special
SRC #007 - 2020 April 1st Ramblings
SRC #005- April, 1st Mean Minichallenge
Introducing APRIL microchallenge
Short & Sweet Math Challenge 20 April 1st Spring Special
Short & Sweet Math Challenge 18 April 1st Spring Special
Short & Sweet Math Challenge 15 April 1st Spring Special

Regards.
V.

WWW FIND EDIT QUOTE REPORT

19th April, 2024, 23:51 **Post: #18**

Valentin Albillo
Senior Member

Posts: 1,100
Joined: Feb 2015
Warning Level: 0%

RE: [VA] SRC #017 - April 1st, 2024 Spring Special

Hi, **all**,

Well, at long last these are my original solutions and comments for the last two sections, i.e. ***LOL the Fifth: Gamma*** and ***LOL the sixth: Miscellanea***. The party's almost over ...

5. LOL the Fifth: Gamma

Executing this loop to list the values of $\Gamma(10^{-1})$, $\Gamma(10^{-2})$, ..., $\Gamma(10^{-10})$, gives:

```
>DESTROY ALL @ FOR N=1 TO 10 @ N;GAMMA(10^(-N))@ NEXT N

  N     Γ(10^-N)
 -------------------
  1  9.51350769867
```

```
  2  99.4325851191
        ...
  9  999999999.423
 10  9999999999.42
```

where the fractional part seems to be quickly converging to some limit around ***~0.42****, so you must write code to find this limit to much greater accuracy (say 10-12 digits or more). Once done, answer these questions:*

- *Can you estimate the most accurate value you got ?*
- *Can you identify the symbolic, closed form of that numeric value ?*

As the table above demonstrates, the *12-digit* **HP-71B** lacks the precision to resolve the fractional part to at least 12 digits so we use the *34-digit* ***Free42*** *Decimal* with this small *15-step*, *27-byte RPN* program which displays just the fractional part of $\Gamma(10^{-N})$ for ***N***= ***-1***, ***-2***, ...

```
01  LBL "GAM10"     09  FP
02  ALL             10  STOP
03   1              11  R↓
04 ►LBL 00          12  ISG ST X
05  ENTER           13  LBL 00
06  +/-             14  GTO 00 ►
07  10^X            15  END
08  GAMMA
```

```
XEQ "GAM10" ->

 { I in ST Y, fractional part in ST X, [R/S] to continue }

   1   .513507698669     { using SHOW from now on }
   2   .432585119151
   3   .423772484595     12   .4227843350994561953914
   4   .422883231624     13   .42278433509856604949
   5   .422794225568     14   .422784335098477702999
   6   .422785324154     15   .4227843350984681235
   7   .422784434004     16   .4227843350984672 42    { best estimate }
   8   .422784344989     17   .42278433509846684
   9   .422784336088     18   .4227843350984742      { error grows, insufficient accuracy }
  10   .422784335197
  11   .422784335108
```

and we see that the fractional part seems to be converging to **0.4227843350984672** for ***N*** = ***16***, truncating at *16 digits* by ignoring two *"guard"* digits, and the values for **N** = **17** and **N** = **18**, as the accuracy is clearly worsening (the error begins to grow). Thus, my *estimate* for the most accurate value obtained is:

0.4227843350984672

Can this value be *identified* ? Well, yes, we can try any of the programs out there online or using your own or your calc's identification software. I did use my **HP-71B**'s `IDENTIFY` program which identified the value as **`1-EulerGamma`**.

If proceeding manually, one can notice that 1 - **0.4227843350984672** = **0.5772156649015328** , which is immediately recognizable but if not, just searching this value online immediately reports it as the Euler–Mascheroni constant, *aka* γ constant. So, the fractional part's limit is *symbolically identified* as:

1 - γ

which is as can be expected because the *Laurent series expansion* of the Γ function near zero is:

$$\Gamma(z) = \frac{1}{z} - \gamma + \frac{1}{12}\left(6\gamma^2 + \pi^2\right) z + O\left(z^2\right)$$

and you can see the **-** γ term there.

Additional comments: *Solver extraordinaire* **Gerson W. Barbosa** tackled this mini-challenge extensively, correctly produced the limit to great accuracy and identified it as the *Euler-Mascheroni* γ constant, but he also tried to significantly improve the accuracy and said (my **bold**):

> *"Here we also notice that the mantissas of the errors, the second column in the table, appear to tend to* ***another constant****. Regardless of any attempt to* ***identify it****, we can try to use it to get a few more correct digits, [...]* ***Now we have about 50% more correct digits****, 23"*

but I think that his reasoning is *circular* because he's using the *actual*, *externally-obtained* 34-digit γ value (*0.5772156649015328606065120900824024*, as seen in his **"L5th"** program,) to compute the second column of the table in order to obtain the *second* constant *(0.98905599...,)* and use it to improve the result. In other words, he's using the *real* γ to improve the accuracy of his *computed* γ, i.e. a circular way of proceeding.

As for the second constant's *(0.98905599...)* identification, it's mainly the ***O(z)*** term in the *Laurent* expansion above, namely:

$$\frac{1}{12}(6\gamma^2+\pi^2)z$$

6. LOL the Sixth: Miscellanea

1. *Try and deduce what result will be output by this **HP-71B** expression without actually executing it:*

   NEIGHBOR(INTEGRAL(FNROOT(EPS,EPS,EPS),EPS,EPS,EPS),GAMMA(EPS))

   Was your deduction correct ? Can you explain the result obtained ?

   The expression returns the value **10**, exactly. **EPS** is the constant **1E-499** and ***GAMMA(EPS)*** is thus **1E499**, which is just used to indicate **NEIGHBOR** the *direction* used to return the nearest machine-representable number to its first argument (the value of **INTEGRAL**,) so its *value* isn't relevant as long as it's ≥ **10**. The breakdown is:

   ```
   >FNROOT(EPS,EPS,EPS)              ->  -9.99999999999E499 = -MAXREAL
   >INTEGRAL(-MAXREAL,EPS,EPS,EPS)  ->   9.99999999999
   >GAMMA(EPS)                       ->   1.E499
   >NEIGHBOR(9.99999999999,1E499)   ->  10
   ```
   so:
   ```
   >NEIGHBOR(INTEGRAL(FNROOT(EPS,EPS,EPS),EPS,EPS,EPS),GAMMA(EPS) -> 10
   ```

2. I *executed this command-line expression on my **HP-71B** but it just resulted in **System Error**. Can you find out what's wrong ?*

   FOR I=64204 TO 64215 @ DISP CHR$(HTD(REV$(PEEK$(DTH$(2*I),2)))); @ NEXT I

   A trick question. The expression above just peeks characters from the specified addresses in the *System ROMs*, which happen to be the ones which form the error message ***"System Error"*** so that's what you get. The expression itself is perfectly correct and returns the pertinent text.

   Additional comments: **JoJo1973** had the right intuition here. He wrote:

   > *"[...] to write a program resulting in the largest number of errors. Of course the easiest way to accomplish the task is **to dump the ROM area where the messages are actually stored** [...]"*

   which goes to the gist of the matter.

3. *Some nice results I got:*

   $\sqrt{95888}$ = 309.657875727390470000000975517...
   $\sqrt{22008840}$ = 4691.3580123456789961013...

   *In **radians**:*

   ```
   >EXP(ACOS(430/433))  ->  1.12500000006     { ~  9/8 }
   >EXP(ACOS(538/541))  ->  1.11111111113     { ~ 10/9 }
   ```

   [...]so perhaps there's some hidden pattern at large here.

   None that I know of, perhaps just a coincidence. Regrettably, no one commented on those nice results I found, which is a pity as the last two seem to me rather remarkable. Another nice one I also found is:

   $$\Gamma\cosh\left(\sqrt[6]{\frac{250}{57}}\phi\right)$$

   where ϕ is the *Golden Ratio* and which evaluates to ***6.2000000002*** on my **HP-71B**.

   Additional comments: The first square root (*$\sqrt{95888}$*) can be used to generate an integer perfect square nearly as awesome as the ones shown in ***LOL the First***, namely

   30965787572739047^2 = 95887***999999999999999999***3958468209

   with no less than ***18*** consecutive *'**9**'*s in the middle ! I'd say this must be the *smallest* perfect square with that many consecutive *'**9**'*s (or any other diigit) in the middle, by far.

4. *What does this **HP-71B** user-defined function compute ?*

```
10  DEF FNC(M,N)=M!/N!/(M-N)!
```

Another trick question. At first sight it would seem that the *"**!**"* symbol stands for the *factorial* (which is the case for standard mathematical notation,) so this user-defined function would seem to be computing ***m!/n!/(m-n)!*** , which is the number of *combinations* of ***m*** things taken ***n*** at a time without repetition.

However, this is not the case here because the **HP-71B** uses the keyword **FACT** for the factorial while *"**!**"* is the ***comment*** delimiter, so *everything* in the line after the first *"**!**"* is considered a *comment* and of course it won't be executed at all.

This means that the line reduces to `10 DEF FNC(M,N)=M` , which just returns the value passed in the **M** argument. As **J-F Garnier** said: *"It doesn't do much, I'm afraid."*

5. *The **HP-71B** does not allow for variable names beginning with a 2-letter or more prefix, but executing this assignment in a program or from the keyboard ...*

```
FORM=STOP
```

*... doesn't result in a **Syntax Error** so what gives ?*

Yet another trick question. The **HP-71B** parser doesn't generally care for spaces between keywords, etc., so it interprets this line as **FOR M=S TO P** , which just begins a `FOR-NEXT` loop using scalar numeric variables **M**, **S** and **P**, all of it perfectly legal syntax.

Additional comments: JoJo1973 nailed this one, despite not owning a **71B**. His experience with a non-*HP*, non-calc machine (a *Commodore 64* no less) served him well to understand what was happening. As he rethorically asked: *"FORM=STOP is a variable assignment or the beginning of a loop: FOR M = S TO P ?"*.

6. *Write code to compute to full accuracy this function **S(x)** given the argument **x**. Use your code to compute **S(42)**:*

$$S(x) = \frac{1}{\sum_{n=0}^{\infty} \binom{2n}{n}^3 \frac{nx+5}{2^{12n+4}}}$$

My original solution is this *3-line* **HP-71B** user-defined function:

```
1  DEF FNS(X) @ S=0 @ T=0 @ N=0 @ REPEAT @ S=S+T
2  T=COMB(2*N,N)^3*(X*N+5)/2^(12*N+4) @ N=N+1
3  UNTIL S=S+T @ FNS=1/S @ END DEF

   >DESTROY ALL
   >FNS(42)

       3.14159265358      { fully correct save 1 ulp }
```

This is yet another marvelous Ramanujan's series. The correct value of this series is exactly π and has the awesome *BBP formula*-like property that, as the denominators are exactly **16 · 2^{n}**, this can be used to compute the *second* block of ***n*** binary digits of π without having to compute the *first* ***n*** binary digits.

Additional comments: John Keith provided a nice *RPL* solution for this mini-challenge and he also added an interesting observation but he *never* posted the *value* produced by his program and that's not Ok. Remember: you *must* mandatorily post both code AND results, one without the other won't do, certainly not when also posting the result is so extremely little extra work.

Juan14 provided algebraic manipulations to achieve extra speed, a working *35-step RPN* program to be run in ***Free42*** *Decimal* and the result accurate to *34 digits* save *1 ulp*. Not bad. Plus he was amazed by yet another unexpected appearance of π.

That's all, the party's over now and it was enjoyable even if few people actually attended so let's call it a wrap. Thanks to everyone who viewed this thread and/or contributed to it.

This will be my last *challenge*-oriented ***SRC*** for an indefinite period of time so I really hope you enjoyed it while it lasted. *All good things ...*

Bye.
V.

WWW FIND EDIT QUOTE REPORT

20th April, 2024, 20:27 **Post: #19**

Gerson W. Barbosa
Senior Member

Posts: 1,580
Joined: Dec 2013

RE: [VA] SRC #017 - April 1st, 2024 Spring Special

> **Valentin Albillo Wrote:** (19th April, 2024 23:51)
>
> ...So, the fractional part's limit is *symbolically identified* as:
>
> **1 -** γ
>
> which is as can be expected because the *Laurent series expansion* of the Γ function near zero is:
>
> $$\Gamma(z) = \frac{1}{z} - \gamma + \frac{1}{12}\left(6\gamma^2 + \pi^2\right) z + O\left(z^2\right)$$
>
> and you can see the **-** γ term there.
>
> **Additional comments:** *Solver extraordinaire* **Gerson W. Barbosa** tackled this mini-challenge extensively, correctly produced the limit to great accuracy and identified it as the *Euler-Mascheroni* γ constant, but he also tried to significantly improve the accuracy and said (my **bold**):
>
> > *"Here we also notice that the mantissas of the errors, the second column in the table, appear to tend to **another constant**. Regardless of any attempt to **identify it**, we can try to use it to get a few more correct digits, [...] **Now we have about 50% more correct digits**, 23"*
>
> but I think that his reasoning is *circular* because he's using the *actual*, externally-obtained 34-digit γ value (*0.5772156649015328606065120900824024*, as seen in his **"L5th"** program,) to compute the second column of the table in order to obtain the *second* constant *(0.98905599...,)* and use it to improve the result. In other words, he's using the *real* γ to improve the accuracy of his *computed* γ, i.e. a circular way of proceeding.
>
> As for the second constant's *(0.98905599...)* identification, it's mainly the ***O(z)*** term in the *Laurent* expansion above, namely:
>
> $$\frac{1}{12}(6\gamma^2 + \pi^2)z$$

Hello, Valentín,

Yes, indeed that was a circular procedure. But then my interest had shifted to using these values of the **Γ** function to get **γ** on the calculator to full accuracy using less bytes than just writing the constant itself. Perhaps I should have used another label instead **"L5th"** for that one.

But I have managed to do it also in a non-circular way, as you can see from this later post:

> **Gerson W. Barbosa Wrote:** (9th April, 2024 19:19)
>
> But there’s a better way to get those extra digits. We only have to use the GAMMA function twice:
>
> ```
> 00 { 29-Byte Prgm }
> 01▸LBL "E_M"
> 02 1
> 03 -11
> 04 10↑X
> 05 GAMMA
> 06 LASTX
> 07 R↓
> 08 FP
> 09 -
> 10 R↑
> 11 +/-
> 12 GAMMA
> 13 FP
> 14 -
> 15 2
> 16 ÷
> 17 END
> ```

This will return

```
0.57721566490153286060651
```

Anyway, it is possible to get about the same accuracy with only one evaluation of the **Γ** function near zero without resorting to circular proceeding. We can ignore the higher-order terms and easily isolate **γ** in the Laurent series expansion you have mentioned above:

$$\gamma \approx \frac{1-\sqrt{1-\frac{z^2\pi^2}{6}+2\left(z\Gamma\left(z\right)-1\right)}}{z}$$

```
 N              (1-√(1-z²π²/6+2(zΓ(z)-1)))/z;  z=10⁻ᴺ
-------------------------------------------------------
 1              0.585903128471302463502247124180713
 2              0.577305962052745649649509365511113
 3              0.577216571923409504524791859837 1
 4              0.57721567397586577623297727055 4
 5              0.5772156649922803103048289349 3
 6              0.57721566490244033922467674 65
 7              0.5772156649015419353968150 27
 8              0.57721566490153295135441953
 9              0.57721566490153286151399 45
10              0.577215664901532860615600
11              0.5772156649015328606069 1
12              0.577215664901532860603 9
-------------------------------------------------------
```

Best regards,

Gerson.

```
00 { 42-Byte Prgm }
01▸LBL "gamma"
02 +/-
03 10↑X
04 PI
05 RCL× ST Y
06 X↑2
07 6
08 ÷
09 RCL ST Y
10 GAMMA
11 RCL× ST L
12 LN
13 E↑X-1
14 STO+ ST X
15 -
16 1
17 X<>Y
18 -
19 SQRT
20 +/-
21 1
22 +
23 X<>Y
24 ÷
25 END
```