[VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" Special

05-04-2018, 10:39 PM **Post: #1**

Valentin Albillo
Senior Member

Posts: 636
Joined: Feb 2015
Warning Level: 0%

[VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" Special

.
Welcome to my *"Star Wars"*-themed *S&SMC#23 "May the 4th Be With You" Special*, young *padawan*.

Here a difficult challe.. erm, *"Jedi Trial"* awaits you. Should you complete its ***6 Steps*** to the complete satisfaction of your *Jedi Master* (that would be me..), you'll be a lowly *padawan* no more but you'll be promoted to be the ***71st Jedi Knight***.

Alas, it won't be easy ! Far from it ! A great reward requires great achievements. But should you succeed, you'll have learned some ancient, valuable techniques which will vastly enrich your skills and, dare I say it ? Yes ! You'll even have Great Fun !

Shall we begin ? But first of all, a little couple of

Notes: While the final *6th Step* is solvable to some degree or other with most any reasonably advanced HP calcs, the first *5 Steps* are intended specifically for the **HP-71B**. It might be the case that some of them aren't meaningless and might be solved using other HP models but I can't be sure.

When using an **HP-71B**, unless the particular task specifies otherwise (and most do), you can also use any of these ROMs: *Math, HPIL, JPC,* plus the *STRNGLEX* Lex file. No other ROMs or LEX files allowed.

In any case, you must use either a *physical* or *emulated/simulated* **HP calc**, solutions for other devices aren't allowed.

Step the First:

Write a program which accepts from the user an integer N from 1 to 19 and outputs the Nth digit of *Log(10)*, the natural logarithm of 10 (= *2.302585092994045684* to 19-digit accuracy).

Example:

- if the user specifies **1**, it must output **2** (the *1st* digit of Log(10))
- if the user specifies **2**. it must output **3** (the *2nd* digit of Log(10))
- if the user specifies **18**, it must output **8** (the *18th* digit of Log(10))
- if the user specifies **19**, it must output **4** (the *19th* digit of Log(10))

Requirements:

- the shorter and faster, the better
- it must run in a *barebones* HP-71B (no ROM/LEX files allowed)
- it must use *no variables at all*
- and needless to say, you can't supply the full 19-digit value of Log(10) to the program in any way or shape (e.g.: as a string in the program, DATA statements, reading from a file, input from the user, etc.).

I'll post my original solution, which is a 1-line program (48 bytes).

Step the Second:

If you succeeded with the previous *Step*, you'll find this one dead easy, namely:

Write a program which accepts from the user an integer N from 1 to 32 and outputs the Nth digit of *Pi/2* (= *1.5707963267948966192313216916397* to 32-digit accuracy).

Example:

- if the user specifies **1**, it must output **1** (the *1st* digit of Pi/2)
- if the user specifies **2**, it must output **5** (the *2nd* digit of Pi/2)
- if the user specifies **31**, it must output **9** (the *31th* digit of Pi/2)
- if the user specifies **32**, it must output **7** (the *32th* digit of Pi/2)

Requirements:

- the shorter and faster, the better
- it must run in a *barebones* HP-71B (no ROM/LEX files allowed)
- it must use *no standard math functions* and *no arithmetic operations* except **+** or **-**
- and again, you can't supply the full 32-digit value of Pi/2 to the program in any way or shape (e.g.: as a string in the program, DATA statements, reading from a file, input from the user, etc.).

I'll post my original solution, which is a 3-line program (152 bytes).

Step the Third:

Now for something different: you don't have to write a program but instead solve right from the command line the following equation (which actually is a polynomial equation in disguise):

$$\sqrt{x+1} + \sqrt{x+2} + \sqrt{x+3} + \ldots + \sqrt{x+98} + \sqrt{x+99} + \sqrt{x+100} = 700$$

Requirements:

- the faster and shorter (in that order), the better
- you can execute more than one command line in succession if need be
- you *can't use data files*
- you *can't run, call or use* any program code whatsoever
- for timing-comparison purposes, use **0** as any initial guess(es)

I'll post my original solution, which is 117 characters long (about 80 bytes). It's slightly longer (just 6 extra characters) but *much* faster (2.2x) than another shorter version which I'll also post.

Step the Fourth:

After that much trouble to complete the previous *Steps*, now for an easy one:

Write a program which accepts a positive integer N from the user and outputs both the number and its square for every value from N down to 0, both included, one pair per line.

Example: assuming the user supplied the number **1234**, the output would be like this, no more, no less:

```
1234    1522756
1233    1520289
1232    1517824
1231    1515361
    ...
4    16
3    9
2    4
1    1
0    0
```

subject to these

Requirements:

- the shorter the better
- it must run in a *barebones* HP-71B (no ROM/LEX files allowed)
- it must use *no variables at all* and no PEEK/POKE either

I'll post my original solution, which is either a 1-line, 49-byte program or a 2-line, 46-byte one.

Step the Fifth:

Enough with the easy stuff. Now we're getting tougher so you must ...

Write a program which accepts from the user a *single-digit* **Id**, then accepts from the user a **text** to scan for said *Id* and output the name associated with that *Id*.

The format of the text to scan (up to 80 characters long, say) is as follows:

(Id1):(Name1),(Id2):(Name2), ... , (IdN):(NameN)

where *(Id)* is a single digit and *(Name)* is a string of up to 30 characters A-Z & spaces. The *Id* aren't necessarily in numerical order in the *text*, but the *Id* sought for must appear somewhere within the *text*.

Example: suppose the user supplies to the program these *Id* and these *texts* to scan (all identical for this particular example):

Id = **1**,
Text = "*2:Yoda,1:Luke Skywalker,5:Obi Wan Kenobi,3:Darth Vader,4:R2D2*"

Output: ***Luke Skywalker***

Id = **5**, same *text* as before   Output: ***Obi Wan Kenobi***
Id = **2**, same *text* as before   Output: ***Yoda***
Id = **4**, same *text* as before   Output: ***R2D2***
Id = **3**, same *text* as before   Output: ***Darth Vader***

Requirements:

- the shorter the better
- it must run in a *barebones* HP-71B (no ROM/LEX files allowed)
- it must use *no variables at all* and no PEEK/POKE either

Sounds familiar, uh ? I'll post my original solution, which is a 2-line program (74 bytes).

Step the Sixth:

We'll call a *"Selfie"* to any positive N-digit integer number which has the property that if you sum its N digits raised to the Nth power you get the original number *backwards*. For instance, the **7**-digit number **5271471** is a *Selfie*:

5271471 => $5^7 + 2^7 + 7^7 + 1^7 + 4^7 + 7^7 + 1^7$ = **1741725**, which is 5271471 *backwards*

Write a program to find all *Selfies* from 1 to 9 digits long (for 10-digit HP calcs, 29 in all) or from 1 to 11 digits long (for 12-digit HP calcs, 37 in all). 0 is *not* a positive number so it's not a *Selfie*.

Requirements:

- the faster and shorter (in that order), the better

I'll post my original solution for the **HP-71B**, an 11-line program (398 bytes) which, when run in **Emu71**, finds in 3'20" all 37 *Selfies* up to 11 digits long. I'll also post a 12-line version which is 20% faster, in particular it finds all 11-digit *Selfies* in just over 80 seconds.

That's it, young *padawan*, your Ordeal has come to an end. I'll post my original solutions next Thursday so you've got plenty of time to develop your own. If you succeed in completing the ***6 Steps*** you will become ***the Most Honored 71st Jedi Knight***.

May the 4th Be With You !!

Regards.
V.
.

Find All My HP-related Materials here: **Valentin Albillo's HP Collection**

PM WWW FIND EDIT QUOTE REPORT

05-05-2018, 11:00 AM **Post: #2**

J-F Garnier
Senior Member

Posts: 461
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

Let's start with the easy part:

> **Valentin Albillo Wrote:** (05-04-2018 10:39 PM)
>
> Step the Fourth
> ...
> Write a program which accepts a positive integer N from the user and outputs both the number and its square for every value from N down to 0, both included, one pair per line.
> ...
> it must use no variables at all and no PEEK/POKE either

The challenge here is to use no variables (otherwise it's trivial).

Re-using a previous idea, here is my **41**-byte solution:

```
10 DISP VAL(DISP$);RES*RES
20 DISP SQRT(RES)-1;RES*RES @ IF RES THEN 20
```

To use it, type the number N, DON'T press ENTER but RUN.

J-F

EMAIL PM WWW FIND QUOTE REPORT

05-05-2018, 04:44 PM **Post: #3**

J-F Garnier
Senior Member

Posts: 461
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

> **Valentin Albillo Wrote:** (05-04-2018 10:39 PM)
>
> Step the First:
> Write a program which accepts from the user an integer N from 1 to 19 and outputs the Nth digit of *Log(10)*, the natural logarithm of 10 (= *2.302585092994045684* to 19-digit accuracy).
> ...
> Step the Second:
> ...
> Write a program which accepts from the user an integer N from 1 to 32 and outputs the Nth digit of *Pi/2* (= *1.5707963267948966192313216916397* to 32-digit accuracy).

I don't have solutions right now for the Steps the First and Second, but the solutions will be in some way related to the LOG(10) and PI values internally known by the HP71.

PI/4 is known with 31 digits, see for instance here.
and it's easy to make the digits 13-24 of PI visible with SIN(3.14159265358).

LOG(10) is known with 20 digits (2.30...56840) within the EXP function code:

```
Saturn Assembler     Math Routines - Part 1 <831213   Fri Dec 30, 1983   3:18 am
Ver. 3.39/Rev. 2306                                                      Page  66



                ********************
                *   |X| >= 1  case  *   Double Reduction by ln10 (x'=x-n'*ln10)
                ********************

     0CFC0 120   DXP200 AR0EX             R0=0...XPON(X); A=X_low
     0CFC3 AFC          ABEX   W          A=X_high; B=X_low

                *   Extra Precision ln10   *
     0CFC6 AF2          C=0    W
     0CFC9 2C           P=     12
     0CFCB 3348         LCHEX  5684       (low digits = 5684018- )
           65
     0CFD1 AF7          D=C    W
     0CFD4 7AB1         GOSUB  LNC10+     + LN10
     0CFD8 CE           C=C-1  A          (C,D)=2.3025 85092 99404 / 56840..0
     0CFDA AFD          BCEX   W
     0CFDD D2           C=0    A
                                          A=...X_high...
                                          B=0230258509299404
                                          C=...X_low..00000
                                          D=568400....00000

                                          R0=0--------xponX
     0CFDF 25           P=     5
```

Now, how to make ALL these extra digits visible in a Basic expression?
PEEKing the nibbles from the ROM would be cheating, no?

J-F

EMAIL PM WWW FIND QUOTE REPORT

05-05-2018, 06:53 PM **Post: #4**

Valentin Albillo
Senior Member

Posts: 636
Joined: Feb 2015
Warning Level: 0%

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

> **J-F Garnier Wrote:** (05-05-2018 04:44 PM)
> PEEKing the nibbles from the ROM would be cheating, no?

Thanks for your interest, J-F.

I'll comment extensively on each Step when I post my original solutions next week but as for your question, simply abide by the specified Requirements for each Step. For instance, if it doesn't specify that you can't pray for a miracle, then you can (and hope for the best) ... :-)

Best regards and have a nice weekend.
V.
.

Find All My HP-related Materials here: **Valentin Albillo's HP Collection**

PM WWW FIND EDIT QUOTE REPORT

05-05-2018, 07:54 PM **Post: #5**

rprosperi
Senior Member

Posts: 4,439
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

> **Valentin Albillo Wrote:** (05-05-2018 06:53 PM)
> ...simply *abide by the specified Requirements for* ***each Step***. For instance, if it doesn't specify that you can't pray for a miracle, then you can (and hope for the best) ... :-)

PEEK/POKE is prohibited for some steps, *but not for steps 1 & 2*.

That said, praying may also be in order here, as the hint implies...

--Bob Prosperi

EMAIL PM FIND QUOTE REPORT

05-08-2018, 05:16 PM **Post: #6**

J-F Garnier
Senior Member

Posts: 461
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

> **Valentin Albillo Wrote:** (05-04-2018 10:39 PM)
> Step the Third:
> ... you don't have to write a program but instead solve right from the command line the following equation ...
> ...
> - you can execute more than one command line in succession if need be
> - you *can't run, call or use* any program code whatsoever

The requirements put strong constraints on a possible solution:
the command line is limited to 96 characters,
FNROOT is not useable here (unless I missed something...),
it is very difficult (or impossible) to make a loop from the keyboard command line and include tests in it.
So I chose a dichotomic search (with a trick to avoid a test in the loop).
It's easy to find the maximum number of dichotomic steps needed, so no need of a termination test.

Here is my solution in two command lines and about 100 characters:
the first line initializes an array with two values defining an interval where the root is for sure:

```
DESTROY A @ A(0)=-1 @ A(1)=49
```

the second line drives a basic dichotomic search:

```
FOR J=1 TO 46 @ X=(A(0)+A(1))/2 @ S=0 @ FOR I=1 TO 100 @ S=S+SQR(X+I)-7 @ NEXT I @ A(S>0)=X @ NEXT J
```

(spaces added only for readability, don't type them in order to fit in a 96-char line).

The result is in X:

```
>X
3.28838856026
```

Any better/shorter/clever solution?

J-F

EMAIL PM WWW FIND QUOTE REPORT

05-09-2018, 11:53 PM **Post: #7**

Valentin Albillo
Senior Member

Posts: 636
Joined: Feb 2015
Warning Level: 0%

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

Hi all, and **hi J-F**:

To all, just a brief reminder that there's still 24 hours left to submit any further solutions to some or all of the 6 *Steps* of this challenge before I post my original solutions tomorrow, as stated in my original post.

As I did in previous challenges, I'll post my original solutions and comments only for those *Steps* for which code for one or more solutions (or attempted solutions) has been posted by the readers. The ones which receive no inputs will best be left for future use.

To **J-F**:

> **J-F Garnier Wrote:**
> Here is my solution in two command lines and about 100 characters:

The first time I tried it, it gave me a *"Subscript"* error because your first line assigns a value to *A(0)* and I had *OPTION BASE 1* set at the time.

This must be addressed, which is why I (nearly) always include *DESTROY ALL* as the first statement in any program or command line, followed by *OPTION BASE* if I use arrays. Adding those two statements to your first command line makes your solution about 15 characters longer or so. No big deal.

Your final *>X* to display the computed root can be included at the end of your second command line, it still fits, so you don't need a third mini-command-line just for it.

> **J-F Garnier Wrote:**
> It is very difficult (or impossible) to make a loop from the keyboard command line and include tests in it.

It's difficult but not impossible, it can be done as long as it all fits in a 96-char line. For instance, to search for and display the very *first* 4-digit square that includes *"444"*, you could issue a command line like this:

```
>FOR X=32 TO 99 @ X=X+100*(POS(STR$(X*X),"444")#0) @ NEXT X @ DISP X-101;RES^2

      38      1444
```

As you can see, it doesn't loop all the way to 99^2 but *stops* looping at the first square which fulfills the test condition and displays both the number and its square.

In a similar fashion, to search for the very first 4-digit *"square plus one"* that happens to be a *prime* number, this command line would do it:

```
>FOR X=32 TO 99 @ X=X+100*(PRIM(X*X+1)=0) @ NEXT X @ DISP X-101,RES^2+1

      36      1297
```

This last example uses the *PRIM* function from the *JPC ROM*, which also includes structures that can be used to implement this functionality. For instance, the last example could be rewritten like this:

```
>X=31 @ REPEAT @ X=X+1 @ UNTIL PRIM(X*X+1)=0 OR X>99 @ DISP X,X*X+1

      36      1297
```

which works as well except for the fact that if no X value would produce a prime then the last value (100) would be output instead , while the first version would output "-1", perhaps better indicating the *not-found* condition.

Best regards.
V.

Find All My HP-related Materials here: **Valentin Albillo's HP Collection**

PM WWW FIND EDIT QUOTE REPORT

05-10-2018, 09:23 AM (This post was last modified: 05-10-2018 09:28 AM by J-F Garnier.) **Post: #8**

J-F Garnier
Senior Member

Posts: 461
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

> **Valentin Albillo Wrote:** ⇒ (05-09-2018 11:53 PM)
>> **J-F Garnier Wrote:**
>> It is very difficult (or impossible) to make a loop from the keyboard command line and include tests in it.
>
> It's difficult but not impossible, it can be done as long as it all fits in a 96-char line. ...

Thanks for your comments, Valentin. What I meant is that it is not possible to include a IF THEN structure in a FOR NEXT loop, because NEXT is not allowed after THEN or ELSE.
So we have to find other ways to take decisions as I did in my solution and as you illustrated with your interesting examples.

For the OPTION BASE issue, here is an updated solution (I don't like changing global settings like OPTION BASE):

```
>DESTROY ALL @ A(1)=-1 @ A(2)=49
>FOR J=1 TO 46 @ X=(A(1)+A(2))/2 @ S=0 @ FOR I=1 TO 100 @ S=S+SQR(X+I)-7 @ NEXT I @ A(1+(S>0))=X @ NEXT J
>X
3.28838856026
```

Waiting for your own solution(s) !

J-F

EMAIL PM WWW FIND QUOTE REPORT

05-10-2018, 10:55 PM **Post: #9**

Valentin Albillo
Senior Member

Posts: 636
Joined: Feb 2015
Warning Level: 0%

[VA] Short & Sweet Math Challenges #23 - My Solutions & Comments

Hi, all:

Time to wrap up this **S&SMC#23**. Sadly, it seems it failed to grasp the attention of forum members judging by the scarcity of solutions/comments posted. I naively thought that there would be a considerable number of **HP-71B** and **Star Wars** fans here that would welcome this homage to the unofficial *Star Wars' Day,* *May the 4th*, but only **J-F Garnier** took notice and contributed his always valuable and interesting solutions and comments.

Also, although the very last *Step (Step the Sixth)* can be solved with every HP calc model out there from the *HP-10C* upwards (at least for 4-5 digits if not all 10-12), no one posted anything on it, not even *J-F*, a real pity. Anyway, these are my original solutions plus assorted comments:

Step the First:

"Write a program which accepts from the user an integer N from 1 to 19 and outputs the Nth digit of Log(10), the natural logarithm of 10 (= 2.302585092994045684 to 19-digit accuracy)."

My original solution is this 1-line program (48 bytes):

```
1 DISP (PEEK$("0CFCD",4)&"4"&PEEK$("0D19A",14))[20-VAL(DISP$)][1,1]
```

Let's see:

```
> 1 [RUN]       2
> 2 [RUN]       3
>18 [RUN]       8
>19 [RUN]       4
```

The *HP-71B* 64 Kb ROM which holds the entire operating system and *BASIC* language includes a number of explicit mathematical constants to a high precision and we can make use of that fact to obtain the pre-stored constants without the need to compute them anew, thus saving lots of time. It's just a question of knowing their address within the ROM.

My solution uses **PEEK** to assemble the 19-digit value of *Log(10)* from the ROM, in reverse order, and then uses the *string-slicing* operator applied *twice* in succession not to a string variable but directly to the resulting string expression to obtain the digits from the one specified by the user to the end of the string, and once more to isolated the precise digit. The digit position specified by the user isn't stored in a variable (via *INPUT*, for instance) but taken directly from the display via **DISP$**. This fulfills the *no-variables* requirement.

Lessons learnt:

- the use of **PEEK$** to extract mathematical constants from the **HP-71B** *System* ROM.
- the use of **DISP$** to obtain a value from the user without storing it in a variable
- the use of the *string-slicing* operator **[]** applied directly to a string expression
- the fact that the [] operator can be applied *more than once* in succession

Step the Second:

"Write a program which accepts from the user an integer N from 1 to 32 and outputs the Nth digit of Pi/2 (= 1.5707963267948966192313216916397 to 32-digit accuracy)."

My original solution is this 3-line program (152 bytes):

```
1 DESTROY ALL @ X=33-VAL(DISP$) @ A$=PEEK$("0DB0F",16)&PEEK$("0DA91",15) @ C=-1
2 FOR I=1 TO 31 @ N=VAL(A$[I,I]) @ N$=STR$(N+N) @ IF LEN(N$)=1 THEN N$="0"&N$
3 A$[I,I]=STR$(VAL(N$[2])+C) @ C=VAL(N$[1,1]) @ NEXT I @ DISP (A$&STR$(C))[X,X]
```

Let's see:

```
> 1 [RUN]       1
> 2 [RUN]       5
>31 [RUN]       9
>32 [RUN]       7
```

On the surface this seems exactly like the previous *Step*, only with another constant instead of *Log(10)*. The difference is that while the 19-digit value of *Log(10)* does appear in the ROM, the 32-digit value of *Pi/2* does *not*.

However, the value of *Pi/4* does so it's simply a matter of retrieving it from the ROM and multiply it times 2 within the requirements, in particular *not* using standard math functions and *no* arithmetical operator except **+** or **-**, which my solution does digit-by-digit. Once assembled, the required digit at the location specified by the user is retrieved and output.

Lessons learnt:

- some constants not in the ROM can be easily derived from the ones available there

Step the Third:

"Solve right from the command line the following equation: Sqrt(x+1) + Sqrt(x+2) + Sqrt(x+3) + ... + Sqrt(x+98) + Sqrt(x+99) + Sqrt(x+100) = 700"

My original solution is this 2-line command-line expression (117 characters in all):

```
>DESTROY ALL @ DIM T$[1400] @ FOR I=1 TO 100 @ T$=T$&"+SQR(FVAR+"&STR$(I)&")" @ NEXT I
>VAL("FNROOT(0,0,"&T$[2]&"-700)")

        3.28838856035
```
(correct 12-digit value: 3.28838856020)

The first command line dimensions a string to hold the full equation, which is then assembled into it in a simple loop, and then the second line simply completes the following expression which uses the **FNROOT** root from the *Math* ROM to solve the equation with identical initial guesses, **0** and **0**. Finally, the entire expression is evaluated using **VAL**, immediately obtaining the root, *3.28838856035*.

The assembled expression which *VAL* evaluates is this 1307-character string:

```
FNROOT(0,0,SQR(FVAR+1)+SQR(FVAR+2)+SQR(FVAR+3)+SQR(FVAR+4)+SQR(FVAR+5)+SQR(F
VAR+6)+SQR(FVAR+7)+SQR(FVAR+8)+SQR(FVAR+9)+SQR(FVAR+10)+SQR(FVAR+11)+SQR(FVAR
+12)+SQR(FVAR+13)+SQR(FVAR+14)+SQR(FVAR+15)+SQR(FVAR+16)+SQR(FVAR+17)+SQR(FVA
R+18)+SQR(FVAR+19)+SQR(FVAR+20)+SQR(FVAR+21)+SQR(FVAR+22)+SQR(FVAR+23)+SQR(FV
AR+24)+SQR(FVAR+25)+SQR(FVAR+26)+SQR(FVAR+27)+SQR(FVAR+28)+SQR(FVAR+29)+SQR(F
VAR+30)+SQR(FVAR+31)+SQR(FVAR+32)+SQR(FVAR+33)+SQR(FVAR+34)+SQR(FVAR+35)+SQR(
FVAR+36)+SQR(FVAR+37)+SQR(FVAR+38)+SQR(FVAR+39)+SQR(FVAR+40)+SQR(FVAR+41)+SQR
(FVAR+42)+SQR(FVAR+43)+SQR(FVAR+44)+SQR(FVAR+45)+SQR(FVAR+46)+SQR(FVAR+47)+SQ
R(FVAR+48)+SQR(FVAR+49)+SQR(FVAR+50)+SQR(FVAR+51)+SQR(FVAR+52)+SQR(FVAR+53)+S
QR(FVAR+54)+SQR(FVAR+55)+SQR(FVAR+56)+SQR(FVAR+57)+SQR(FVAR+58)+SQR(FVAR+59)+
SQR(FVAR+60)+SQR(FVAR+61)+SQR(FVAR+62)+SQR(FVAR+63)+SQR(FVAR+64)+SQR(FVAR+65)
```

```
+SQR(FVAR+66)+SQR(FVAR+67)+SQR(FVAR+68)+SQR(FVAR+69)+SQR(FVAR+70)+SQR(FVAR+71
)+SQR(FVAR+72)+SQR(FVAR+73)+SQR(FVAR+74)+SQR(FVAR+75)+SQR(FVAR+76)+SQR(FVAR+7
7)+SQR(FVAR+78)+SQR(FVAR+79)+SQR(FVAR+80)+SQR(FVAR+81)+SQR(FVAR+82)+SQR(FVAR+
83)+SQR(FVAR+84)+SQR(FVAR+85)+SQR(FVAR+86)+SQR(FVAR+87)+SQR(FVAR+88)+SQR(FVAR
+89)+SQR(FVAR+90)+SQR(FVAR+91)+SQR(FVAR+92)+SQR(FVAR+93)+SQR(FVAR+94)+SQR(FVA
R+95)+SQR(FVAR+96)+SQR(FVAR+97)+SQR(FVAR+98)+SQR(FVAR+99)+SQR(FVAR+100)-700)
```

and as you can see, both *FNROOT* and VAL *have* no problem in dealing with expressions of *any length* whatsoever, they're limited only by available RAM, *not* by the 96-character limit of the command-line itself. The same is true of *INTEGRAL*.

An alternative for the second command line would be the following:

```
>FNROOT(0,0,VAL(T$[2])-700)
```

which is a slightly shorter (just 6 characters less) but *much slower*. Matter of fact the first version is *2.2 times faster*, the reason being that the first expression parses the long expression to evaluate only *once*, then has *FNROOT* working with the parsed expression, while the second version has *FNROOT* executing *VAL* repeatedly as many times as needed to find and refine the root and thus parsing the long string *multiple* times.

J-F also managed to find a nice solution within the requirements (except for the fact that he uses initial guesses different from 0, which was also a requirement but that would be nitpicking) but as he uses a user-code loop within a loop it is much slower than using the assembly-language *FNROOT* and *VAL* applied to an expression parsed once into fast internal format. Anyway, congratulations to *J-F* for his very clever way to achieving the stated goal.

Just for fun and to demonstrate this *no-limits* capability even further, solving the following equation:

Sqrt(x+1) + Sqrt(x+2) + Sqrt(x+3) + ... + Sqrt(x+98) + Sqrt(x+99) + Sqrt(x+200) = 2018

entails executing these command lines (118 character in all):

```
>DESTROY ALL @ DIM T$[2800] @ FOR I=1 TO 200 @ T$=T$&"+SQR(FVAR+"&STR$(I)&")" @ NEXT I
>VAL("FNROOT(0,0,"&T$[2]&"-2018)")
```

`10.4122270141` *(correct 12-digit value: 10.4122270160)*

and this time the expression which *VAL* evaluates is the 2708-character string:

```
FNROOT(0,0,SQR(FVAR+1)+SQR(FVAR+2)+SQR(FVAR+3)+SQR(FVAR+4)+SQR(FVAR+5)+SQR(F
VAR+6)+SQR(FVAR+7)+SQR(FVAR+8)+SQR(FVAR+9)+SQR(FVAR+10)+SQR(FVAR+11)+SQR(FVAR
+12)+SQR(FVAR+13)+SQR(FVAR+14)+SQR(FVAR+15)+SQR(FVAR+16)+SQR(FVAR+17)+SQR(FVA
R+18)+SQR(FVAR+19)+SQR(FVAR+20)+SQR(FVAR+21)+SQR(FVAR+22)+SQR(FVAR+23)+SQR(FV
AR+24)+SQR(FVAR+25)+SQR(FVAR+26)+SQR(FVAR+27)+SQR(FVAR+28)+SQR(FVAR+29)+SQR(F
VAR+30)+SQR(FVAR+31)+SQR(FVAR+32)+SQR(FVAR+33)+SQR(FVAR+34)+SQR(FVAR+35)+SQR(
FVAR+36)+SQR(FVAR+37)+SQR(FVAR+38)+SQR(FVAR+39)+SQR(FVAR+40)+SQR(FVAR+41)+SQR
(FVAR+42)+SQR(FVAR+43)+SQR(FVAR+44)+SQR(FVAR+45)+SQR(FVAR+46)+SQR(FVAR+47)+SQ
R(FVAR+48)+SQR(FVAR+49)+SQR(FVAR+50)+SQR(FVAR+51)+SQR(FVAR+52)+SQR(FVAR+53)+S
QR(FVAR+54)+SQR(FVAR+55)+SQR(FVAR+56)+SQR(FVAR+57)+SQR(FVAR+58)+SQR(FVAR+59)+
SQR(FVAR+60)+SQR(FVAR+61)+SQR(FVAR+62)+SQR(FVAR+63)+SQR(FVAR+64)+SQR(FVAR+65)
+SQR(FVAR+66)+SQR(FVAR+67)+SQR(FVAR+68)+SQR(FVAR+69)+SQR(FVAR+70)+SQR(FVAR+71
)+SQR(FVAR+72)+SQR(FVAR+73)+SQR(FVAR+74)+SQR(FVAR+75)+SQR(FVAR+76)+SQR(FVAR+7
7)+SQR(FVAR+78)+SQR(FVAR+79)+SQR(FVAR+80)+SQR(FVAR+81)+SQR(FVAR+82)+SQR(FVAR+
83)+SQR(FVAR+84)+SQR(FVAR+85)+SQR(FVAR+86)+SQR(FVAR+87)+SQR(FVAR+88)+SQR(FVAR
+89)+SQR(FVAR+90)+SQR(FVAR+91)+SQR(FVAR+92)+SQR(FVAR+93)+SQR(FVAR+94)+SQR(FVA
R+95)+SQR(FVAR+96)+SQR(FVAR+97)+SQR(FVAR+98)+SQR(FVAR+99)+SQR(FVAR+100)+SQR(F
VAR+101)+SQR(FVAR+102)+SQR(FVAR+103)+SQR(FVAR+104)+SQR(FVAR+105)+SQR(FVAR+106
)+SQR(FVAR+107)+SQR(FVAR+108)+SQR(FVAR+109)+SQR(FVAR+110)+SQR(FVAR+111)+SQR(F
VAR+112)+SQR(FVAR+113)+SQR(FVAR+114)+SQR(FVAR+115)+SQR(FVAR+116)+SQR(FVAR+117
)+SQR(FVAR+118)+SQR(FVAR+119)+SQR(FVAR+120)+SQR(FVAR+121)+SQR(FVAR+122)+SQR(F
VAR+123)+SQR(FVAR+124)+SQR(FVAR+125)+SQR(FVAR+126)+SQR(FVAR+127)+SQR(FVAR+128
)+SQR(FVAR+129)+SQR(FVAR+130)+SQR(FVAR+131)+SQR(FVAR+132)+SQR(FVAR+133)+SQR(F
VAR+134)+SQR(FVAR+135)+SQR(FVAR+136)+SQR(FVAR+137)+SQR(FVAR+138)+SQR(FVAR+139
)+SQR(FVAR+140)+SQR(FVAR+141)+SQR(FVAR+142)+SQR(FVAR+143)+SQR(FVAR+144)+SQR(F
VAR+145)+SQR(FVAR+146)+SQR(FVAR+147)+SQR(FVAR+148)+SQR(FVAR+149)+SQR(FVAR+150
)+SQR(FVAR+151)+SQR(FVAR+152)+SQR(FVAR+153)+SQR(FVAR+154)+SQR(FVAR+155)+SQR(F
VAR+156)+SQR(FVAR+157)+SQR(FVAR+158)+SQR(FVAR+159)+SQR(FVAR+160)+SQR(FVAR+161
)+SQR(FVAR+162)+SQR(FVAR+163)+SQR(FVAR+164)+SQR(FVAR+165)+SQR(FVAR+166)+SQR(F
VAR+167)+SQR(FVAR+168)+SQR(FVAR+169)+SQR(FVAR+170)+SQR(FVAR+171)+SQR(FVAR+172
)+SQR(FVAR+173)+SQR(FVAR+174)+SQR(FVAR+175)+SQR(FVAR+176)+SQR(FVAR+177)+SQR(F
VAR+178)+SQR(FVAR+179)+SQR(FVAR+180)+SQR(FVAR+181)+SQR(FVAR+182)+SQR(FVAR+183
)+SQR(FVAR+184)+SQR(FVAR+185)+SQR(FVAR+186)+SQR(FVAR+187)+SQR(FVAR+188)+SQR(F
VAR+189)+SQR(FVAR+190)+SQR(FVAR+191)+SQR(FVAR+192)+SQR(FVAR+193)+SQR(FVAR+194
)+SQR(FVAR+195)+SQR(FVAR+196)+SQR(FVAR+197)+SQR(FVAR+198)+SQR(FVAR+199)+SQR(F
VAR+200)-2018)
```

which is quite a sight to behold.

Lessons learnt:

- That **VAL** and **FNROOT** (and **INTEGRAL**) can deal with expressions *limited only by available RAM*, not by the 96-character limit of the command line.

Step the Fourth:

"Write a program which accepts a positive integer N from the user and outputs both the number and its square for every value from N down to 0, both included, one pair per line."

As **J-F** wrote in his nice solution, the task would be utterly trivial were it not for the *no-variables* requirement. My original solution is this 2-line (46 bytes) program:

```
1 DISP USING "#,^";(VAL(DISP$)+1)^2
2 DISP SQR(RES)-1,RES^2 @ IF SQR(RES) THEN 2
```

Let's see:

```
>15 [RUN]

 15       225
 14       196
 13       169
 ...

 3       9
 2       4
 1       1
 0       0
```

The trick to avoid using variables is first of all to accept a value from the user *directly from the command line* using **DISP$**, instead of inputting it to a variable, and then to store and handle it using **RES**, a system location which stores the value of the most recently evaluated **or displayed** numeric expression (whether real- or complex-valued but not string-valued).

Thus, the first line simply puts in *RES* an adequate initial value for the loop in the second line by simply displaying it. Normally this would result in this spurious initial value being displayed too, as a side effect, but this is avoided by the **USING** *image* which supresses both the value and the return carriage.

The second line then enters a simple loop where *SQR(RES)-1* retrieves and displays the next value to square and *RES^2* squares it and displays the result as well. As soon as *SQR(RES)* reaches 0, the loop (and the program) ends. Notice that *IF* does **not** update the value of *RES* to the expression being evaluated and tested.

By the way, there's another 1-line version which is 49 bytes instead of 46:

```
1 DISP USING "#,^";(VAL(DISP$)+1)^2 @ 'A': DISP SQR(RES)-1,RES^2 @ IF SQR(RES) THEN 'A'
```

It simply uses a *local label 'A' midline* to implement the loop instead of a line number, thus avoiding the use of a second line.

Lessons learnt:

- that **RES** can be used to occasionally replace a variable
- that **USING** a *image* we can store a value in *RES* without actually displaying it
- that a *local label midline* allows looping within part of a single line

Step the Fifth:

"Write a program which accepts from the user a single-digit Id, then accepts from the user a text to scan for said Id and output the name associated with that Id.

The format of the text to scan (up to 80 characters long, say) is as follows:

(Id1):(Name1),(Id2):(Name2), ... , (IdN):(NameN)

where (Id) is a single digit and (Name) is a string of up to 30 characters A-Z & spaces. The Id aren't necessarily in numerical order in the text, but the Id sought for must appear somewhere within the text."

This is similar in spirit to *Step 4* above but with the added difficulty of having to handle *two* inputs (the *Id* to search for and the *text* where to search for it) without using variables. My original solution is a 2-line program (74 bytes) but as no one posted any code or comment for this *Step*, it will be reserved for possible use at some future time.

Step the Sixth:

"We'll call "Selfie" to any positive N-digit integer number which has the property that if you sum its N digits raised to the Nth power you get the original number backwards. For instance, the 7-digit number 5271471 is a Selfie:

5271471 => 5^7 + 2^7 + 7^7 + 1^7 + 4^7 + 7^7 + 1^7 = 1741725, which is 5271471 backwards

Write a program to find all Selfies from 1 to 9 digits long (for 10-digit HP calcs, 29 in all) or from 1 to 11 digits long (for 12-digit HP calcs, 37 in all). 0 is not a positive number so it's not a Selfie."

Same here, no one posted any code or comments for this one either so I'll also save my original solution for a future article or something. As I said at the beginning of this post, it's a real pity as this Step was solvable (partially or in full) using most any HP calc model, and there are at least 3 ways to handle it, including brute force which will quickly become unfeasible as for 11-digit numbers there would be some 90 billion numbers to try, so more sophisticated techniques are required to reach that far in reasonable times (a few minutes).

By the way, I didn't ask for 12/10-digit solutions because for some 12/10-digit numbers the sum of their digits raised to the 12th/10th power exceeds the 12/10-digit range. Also, *there are no 12-digit Selfies* (but there's a *unique* 10-digit one).

That's all for now.
Regards.
V.

Find All My HP-related Materials here: **Valentin Albillo's HP Collection**

PM WWW FIND EDIT QUOTE REPORT

05-10-2018, 11:56 PM **Post: #10**

rprosperi
Senior Member

Posts: 4,439
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

> **Valentin Albillo Wrote:** (05-10-2018 10:55 PM)
> **Hi, all:**
>
> Time to wrap up this **S&SMC#23**.

Thanks for these painstakingly detailed notes about your solutions; these are always enjoyable even I've not had time, or sometimes the skills, to tackle them. I particularly like your use of "Lessons learned" to highlight the key techniques used for each solution, this helps a lot to drive these points in.

> **Valentin Albillo Wrote:** (05-10-2018 10:55 PM)
> - the fact that the [] operator can be applied *more than once* in succession

I had no idea one could apply the substring operator (I like 'slicer' better) multiple times. Though it appears to not be documented it does make sense following the 71b's syntax, etc. Very clever!

Thanks for another set of interesting and educational challenges. I suspect all the drama taking place in parallel distracted folks from playing along.

--Bob Prosperi

EMAIL PM FIND QUOTE REPORT

05-11-2018, 07:29 PM **Post: #11**

Valentin Albillo
Senior Member

Posts: 636
Joined: Feb 2015
Warning Level: 0%

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

Hi, Bob !

> **rprosperi Wrote:** (05-10-2018 11:56 PM)
> Thanks for these painstakingly detailed notes about your solutions; these are always enjoyable even I've not had time, or sometimes the skills, to tackle them. I particularly like your use of "Lessons learned" to highlight the key techniques used for each solution, this helps a lot to drive these points in.

Certainly these challenges I concoct *do* take a lot of time, first for getting a workable idea, then to find my own solutions to it, and finally to write, proofread and format the looong posts (the one for the challenge proper and the one for the solutions).

I recently added the feature you mention and I'm glad you like it. Though not a teacher by trade, I'm pretty fond of sharing knowledge with others, which can be considered kind of "teaching".

> **Quote:**
> I had no idea one could apply the substring operator (I like 'slicer' better) multiple times. Though it appears to not be documented it does make sense following the 71b's syntax, etc. Very clever!

I knew about the "slicer" many decades ago, back in the very early 80's, where I saw it available in **HP-85**'s BASIC. There, it could be applied only to string variables or elements of a string array but not to string expressions, and only once. The **HP-71B**'s BASIC version is much enhanced over the *HP-85*'s one and can be applied also to string expressions and any number of times in sequence.

In the first challenge I use it to save bytes, because `[20-VAL(DISP$)][1,1]` is shorter than other ways of achieving the same result with a single slicing operation.

> **Quote:**
> Thanks for another set of interesting and educational challenges. I suspect all the drama taking place in parallel distracted folks from playing along.

Thanks to you for your continued appreciation and for letting me know, it means a lot to me to ascertain that my efforts didn't go to waste.

Have a nice weekend and best regards.
V.

Find All My HP-related Materials here: **Valentin Albillo's HP Collection**

PM WWW FIND EDIT QUOTE REPORT

05-11-2018, 08:51 PM **Post: #12**

pier4r
Senior Member

Posts: 2,067
Joined: Nov 2014

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

> **Valentin Albillo Wrote:** (05-10-2018 10:55 PM)
> I naively thought that there would be a considerable number of **HP-71B** and **Star Wars** fans here

I think that some challenges requires just more time as one has to have the right conditions to attack them. Especially if the challenges are focused on few systems.

Also recently the forum went through some turbolences so this may have distracted the community as well.

In any case, always kudos for the extensive explanation!

Wikis are great, Contribute :)

EMAIL PM FIND QUOTE REPORT

05-11-2018, 08:57 PM **Post: #13**

Egan Ford
Member

Posts: 167
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

> **Valentin Albillo Wrote:** (05-10-2018 10:55 PM)
> no one posted any code or comments for this one either so I'll also save my original solution for a future article or something. As I said at the beginning of this post, it's a real pity as this Step was solvable (partially or in full) using most any HP calc model, and there are at least 3 ways to handle it, including brute force which will quickly become unfeasible as for 11-digit numbers there would be some 90 billion numbers to try, so more sophisticated techniques are required to reach that far in reasonable times (a few minutes).
>
> By the way, I didn't ask for 12/10-digit solutions because for some 12/10-digit numbers the sum of their digits raised to the 12th/10th power exceeds the 12/10-digit range. Also, *there are no 12-digit Selfies* (but there's a *unique* 10-digit one).

I put 3 hours into this and one sleepless night (could not stop thinking how to optimize). It'll continue to haunt me just like SSMC #20's last problem continues to do today. Coincidently I was reading Henry Ibstedt yesterday.

EMAIL PM FIND QUOTE REPORT

05-12-2018, 09:37 AM **Post: #14**

J-F Garnier
Senior Member

Posts: 461
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

Valentin, your solution of the Step the Third is really great!

I didn't known that it was possible to evaluate long expressions in that way.

I learnt something today (yes, it's still possible to learn new things, after more than 30 years of HP71 usage), thanks again:

> **Valentin Albillo Wrote:** (05-10-2018 10:55 PM)
> *Lessons learnt:*
>
> - That **VAL** and **FNROOT** (and **INTEGRAL**) can deal with expressions *limited only by available RAM*, not by the 96-character limit of the command line.

How did you come to discover this possibility? By chance or rational exploration?

Of course, it is only possible thanks to the VAL function of the HP71, that is actually an expression evaluation function, rather than the classic Basic VAL function that can only convert a number from its ASCII representation.
For instance, the solution is not applicable to the HP75.

The HP71 design team did a great job at the time.

J-F

EMAIL PM WWW FIND QUOTE REPORT

05-13-2018, 12:59 AM **Post: #15**

cortopar
Junior Member

Posts: 15
Joined: May 2018

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

Valentin,

I bet it doesn't feel like it, but even if this thread only had two posts, your original challenge and your answers, it would be tremendously valuable to many of us who observe more than interact.

Your posts over the years are about 98% of the reason for my interest in the 71b.

Thanks for continuing to carry the 71b torch, and thanks for all your posts on other models that have added to my RPN and math knowledge.

All the best,
Bob

PM FIND QUOTE REPORT

05-14-2018, 12:15 AM **Post: #16**

Valentin Albillo
Senior Member

Posts: 636
Joined: Feb 2015
Warning Level: 0%

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

.
Hi, **J-F Garnier** and **cortopar**:

> **J-F Garnier Wrote:** (05-12-2018 09:37 AM)
> Valentin, your solution of the Step the Third is really great!

Thank you very much, I'm glad you liked it.

> **Quote:**
> I didn't known that it was possible to evaluate long expressions in that way.
> I learnt something today (yes, it's still possible to learn new things, after more than 30 years of HP71 usage), thanks again:

You're welcome. The fact that you learned something new is proof enough that you're stilll pretty young, you know what they say about old dogs not being able to learn new tricks. :-D

> **Quote:**
> How did you come to discover this possibility? By chance or rational exploration?

I discovered that **VAL** can evaluate long expressions soon after I got me a pretty (and expensive!) **HP-71B** complete with **HP-IL** ROM, **Forth/Assembler** ROM and all the **IDS** volumes.

Afterwards, when I also got the **Math** ROM back inf 1985 (without its IDS, regrettably) I did test the limits of most of its functions and empirically discovered that the *funny functions* **FNROOT** and **INTEGRAL** were akin to *VAL* in that regard. I took notice of those facts at the time and then it was just a matter of finding the right occasion to share the knowledge.

> **Quote:**
> Of course, it is only possible thanks to the VAL function of the HP71, that is actually an expression evaluation function, rather than the classic Basic VAL function that can only convert a number from its ASCII representation. For instance, the solution is not applicable to the HP75.

Indeed, I've found no other version of *VAL* (or its equivalent in other languages) which can do that. The *HP-85/86/87* couldn't either, neither could the *HP9816/26/36* or the various versions of *Pascal* or *Visual This/Visual That*, etc.

> **Quote:**
> The HP71 design team did a great job at the time.

They did 95% great things and a few % not that great. For instance, I will never forgive the *<expletive>* who decided the inclusion of that abomination called *"CALC mode"*, which utterly wasted 5 Kb of the 64 Kb ROM which could've been put to much, much better use.

Or the *<milder expletive>* who implemented *string handling* in such a way that the system must have the whole string on the stack to do anything with it: say you need to extract or change the Nth character of a long string, the operating system cannot simply do it using pointer/address manipulations, no, it must copy the whole string to the stack, using lots of valuable RAM and time and thus making using long strings inefficient and slow. And there are more (meager string functions, missing complex functions, poor *PEEK/POKE* implementation, missing matrix functions in the *Math* ROM, etc) ...

> **cortopar Wrote:**

> I bet it doesn't feel like it, but even if this thread only had two posts, your original challenge and your answers, it would be tremendously valuable to many of us who observe more than interact.

Thank you very much, **Bob**, you're far too kind and I'm really glad you appreciate my *S&SMCs*.

> **Quote:**
> Your posts over the years are about 98% of the reason for my interest in the 71b. Thanks for continuing to carry the 71b torch, and thanks for all your posts on other models that have added to my RPN and math knowledge.

Thanks again. Though I tend to use the *HP-71B* more than other models, for obvious reasons, I *do like* and admire a lot the classic RPN ones and I have scores of programs written by myself for the *HP-11C, HP-15C, HP-25, HP-34C, HP-67/97, HP-41C, HP42S* and even the *HP35s*, most of them unpublished.

I intend to scan or type them anew as soon as I can find some way to put them online, which I'm actually looking into right now.

Best regards.
V.
.

Find All My HP-related Materials here: **Valentin Albillo's HP Collection**

PM WWW FIND EDIT QUOTE REPORT

05-14-2018, 02:59 PM **Post: #17**

John Keith
Senior Member

Posts: 615
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

> **Valentin Albillo Wrote:** (05-14-2018 12:15 AM)
> Indeed, I've found no other version of *VAL* (or its equivalent in other languages) which can do that. The *HP-85/86/87* couldn't either, neither could the *HP9816/26/36* or the various versions of *Pascal* or *Visual This/Visual That*, etc.

Except, of course, for RPL and other LISP-derived languages.

> **Quote:**
> They did 95% great things and a few % not that great. For instance, I will never forgive the *<expletive>* who decided the inclusion of that abomination called *"CALC mode"*, which utterly wasted 5 Kb of the 64 Kb ROM which could've been put to much, much better use.

Amen! They could at least have given us an RPN calculator mode. HP-41 compatible ideally, although I don't know if that would have fit into 5K.

Thanks from me also for a fun and invigorating challenge, although mostly "above my pay grade".

John

EMAIL PM FIND QUOTE REPORT

05-14-2018, 11:21 PM **Post: #18**

Valentin Albillo
Senior Member

Posts: 636
Joined: Feb 2015
Warning Level: 0%

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

.
Hi, **John**:

> **Quote:**
> Amen! They could at least have given us an RPN calculator mode. HP-41 compatible ideally, although I don't know if that would have fit into 5K.

It probably would, 5 Kb of optimized Saturn assembly language can reach very far. Just consider the many worthwhile functionalities of the basic HP-71B and it all fits in just 59 Kb, even still including some space/time-wasting garbage.

Also, the HP-71B operating system already includes essential RPN functionalities. Each numeric expression entered is parsed and tokenized into internal RPN form for storage as program code and/or execution, then decompiled as necessary for program listings, editing and debugging. It would simply be a matter of harnessing that existing functionality and exposing it to the user.

> **Quote:**
> Thanks from me also for a fun and invigorating challenge, although mostly "above my pay grade".

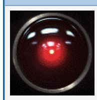
You're welcome, I'm truly glad that you liked it and greatly appreciate your kind feedback.

Regards.
V.
.

Find All My HP-related Materials here: **Valentin Albillo's HP Collection**

PM WWW FIND EDIT QUOTE REPORT

05-15-2018, 06:22 PM **Post: #19**



Jeff O.
Member

Posts: 185
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

> **Valentin Albillo Wrote:** ⇒ (05-10-2018 10:55 PM)
>
> **Hi, all:**
>
> Time to wrap up this **S&SMC#23**. ...
>
> *"We'll call "Selfie" to any positive N-digit integer number which has the property that if you sum its N digits raised to the Nth power you get the original number backwards. For instance, the 7-digit number 5271471 is a Selfie:*
>
> *5271471 => 5^7 + 2^7 + 7^7 + 1^7 + 4^7 + 7^7 + 1^7 = 1741725, which is 5271471 backwards*
>
> *Write a program to find all Selfies from 1 to 9 digits long (for 10-digit HP calcs, 29 in all) or from 1 to 11 digits long (for 12-digit HP calcs, 37 in all). 0 is not a positive number so it's not a Selfie."*
>
> Same here, no one posted any code or comments for this one either so I'll also save my original solution for a future article or something. As I said at the beginning of this post, it's a real pity as this Step was solvable (partially or in full) using most any HP calc model...

I'll confess to not having read through your challenge closely enough to see your note that this particular step might be potentially solvable by other models. I think the 71B is a wonderful machine and am glad to have an example, but have never attempted to master its use, so I assumed this challenge was not for me. Upon further review, Step the 6th is the kind of number manipulation challenge that I have enjoyed attempting on various models, along the lines of some of the HHC programming contests. With that said, reading that Egan put 3 hours and one sleepless night into it, and it will continue to haunt him, kind of scares me off a bit. My usual inclination with such problems is to just go ahead and try for a brute force method, then try to optimize. The DM42 is fast, I would like to see how many digits it could handle in a reasonable time by brute force, so if I get the time I may have a go at it.

In any case, thanks for your challenges, please don't be put off by a lower than hoped-for response. Next time I'll be sure to read through more carefully!

Dave - My mind is going - I can feel it.

PM FIND QUOTE REPORT

05-15-2018, 06:42 PM **Post: #20**

Maximilian Hohmann
Senior Member

Posts: 770
Joined: Dec 2013

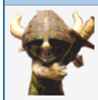
RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

> **Jeff O. Wrote:** ⇒ (05-15-2018 06:22 PM)
>
> In any case, thanks for your challenges, please don't be put off by a lower than hoped-for response.

Same here! Although I do have some 71Bs and even a Math ROM I am only superficially familiar with it's many functions and would not have been able to solve a single one of these challenges. Especially the ones which require PEEKing the digits of maththematical constants out of the ROM... (The last time I wrote the word "PEEK" before this reply must have been ca. 1983 when I did some machine language programming on my Sinclair ZX81). Nontheless these challenges and the answers are a pleasure to read and think about!

EMAIL PM FIND QUOTE REPORT

05-17-2018, 04:22 AM **Post: #21**



brickviking
Senior Member

Posts: 334
Joined: Dec 2014

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

> **Maximilian Hohmann Wrote:** ⇒ (05-15-2018 06:42 PM)
>
> ... (The last time I wrote the word "PEEK" before this reply must have been ca. 1983 when I did some machine language programming on my Sinclair ZX81). ...

PEEK and POKE have interesting results on a SHARP PC-1247. I got several somewhat unexpected results by poking instruction codes directly into program memory, as that was within the range of program listings. I don't have that calculator any more, I suspect I lost it in a move along with the dual-trace 1MHz oscilloscope.

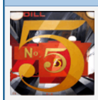
As the 1247 only had 3328 bytes of addressable memory, it wasn't considered "big iron" enough for what I thought I wanted out of a programmable calculator. It certainly wasn't in the same league as the 71B or 75C/D (but was probably considerably cheaper).

(Post 220)

Regards, BrickViking
HP-50g |Casio fx-9750G+ |Casio fx-9750GII (SH4a)

PM WWW FIND QUOTE REPORT

05-17-2018, 10:12 PM **Post: #22**



Valentin Albillo
Senior Member

Posts: 636
Joined: Feb 2015
Warning Level: 0%

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

.

Hi, Jeff O, **Maximilian Hohmann** and **brickviking**:

> **Jeff O Wrote:**
>
> Upon further review, Step the 6th is the kind of number manipulation challenge that I have enjoyed attempting on various models [...] With that said, reading that Egan put 3 hours and one sleepless night into it, and it will continue to haunt him, kind of scares me off a bit.

No need to be afraid, Egan was probably trying to scare would-be solvers, actually it's not that difficult. On hindsight, I probably goofed badly when I put this lovely problem last after a row of essentially 71B-only, tricky ones, so people failed to notice its generality (more or less solvable in every machine) and classic nature. My bad.

> **Quote:**
>
> My usual inclination with such problems is to just go ahead and try for a brute force method, then try to optimize. The DM42 is fast, I would like to see how many digits it could handle in a reasonable time by brute force, so if I get the time I may have a go at it.

Please do. I haven't created a solution for the *DM42* but I guesstimate that with correct programming it can solve the 11-digit version in 5 min. to 1 hour running time.

Quote:
In any case, thanks for your challenges, please don't be put off by a lower than hoped-for response. Next time I'll be sure to read through more carefully!

Thank you very much, you'll be welcome to try. I feel better now. :-D

Maximilian Hohmann Wrote:
Although I [...] would not have been able to solve a single one of these challenges. Especially the ones which require PEEKing the digits of mathematical constants out of the ROM... (The last time I wrote the word "PEEK" before this reply must have been ca. 1983 when I did some machine language programming on my Sinclair ZX81).

He he, same here ! I also had a *Sinclair ZX81* back then and also did *Z80A* machine language programming, most especially video games and graphics routines. I got many books dealing with the matter (which I still keep to this day), most of them truly excellent, and learned a lot. I remember writing my *Bombardier* game utterly by hand, with no compilers or any other tools, painstakingly computing the offsets for the jumps by sheer byte-counting, etc., and being ecstatic when it run fine the first time I tried it, not even a single bug or miscalculation. Those were the days ... !

Quote:
Nonetheless these challenges and the answers are a pleasure to read and think about!

Thank you very much for your kind words, much appreciated. See below for something on *PEEK* ... :-)

brickviking Wrote:
PEEK and POKE have interesting results on a SHARP PC-1247. I got several somewhat unexpected results by poking instruction codes directly into program memory, as that was within the range of program listings.

It's quite similar to the way we HP calc fans initially discovered synthetics in the *HP-41C*. We would enter data in storage registers and thanks to *Bug 2* it would appear in program memory as various synthetic functions, most notably *STO M, N,* and such. My first *HP-41C* was a very early model with all the bugs. Regrettably, I eventually sold it and the next *HP-41C* I got didn't have *Bug 2* anymore.

Quote:
As the 1247 only had 3328 bytes of addressable memory, it wasn't considered "big iron" enough for what I thought I wanted out of a programmable calculator. It certainly wasn't in the same league as the 71B or 75C/D (but was probably considerably cheaper).

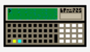
Much cheaper. And the *71B* was also much cheaper (and 5 times slower !) than the *75CD* (which I never liked, too bulky, bad keyboard layout, mediocre *BASIC*).

Best regards to all.
V.
.

Find All My HP-related Materials here: **Valentin Albillo's HP Collection**

PM WWW FIND EDIT QUOTE REPORT

05-20-2018, 03:50 PM **Post: #23**



J-F Garnier
Senior Member

Posts: 461
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

Valentin Albillo Wrote: (05-17-2018 10:12 PM)
... the *71B* was also much cheaper (and 5 times slower !) than the *75CD* (which I never liked, too bulky, bad keyboard layout, mediocre *BASIC*).

I do also prefer the HP71, but the HP75 is an interesting machine, too.
Especially the 16kB Math module is very good, much more powerful than the HP80 series Matrix ROM. It has matrix functions, complex number support (although not as nicely integrated than on the HP71), various utility math functions, the PROOT polynomial root finder, the Fourier Transform and more important the FNROOT and INTEGRAL functions.

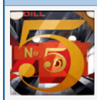
That is quite the same feature set than the 71.
Or we may better say that the HP71 Math ROM included all the previous HP75 Math ROM features, adding a better integration with the mainframe (e.g. complex number) and improvements such as the re-entrant FNROOT and INTEGRAL functions.

And with emu75 (very similar to emu71/DOS that you know very well), the first two drawbacks mentioned above are no more relevant :-)

J-F

EMAIL PM WWW FIND QUOTE REPORT

05-21-2018, 12:01 AM (This post was last modified: 05-21-2018 03:17 AM by Valentin Albillo.) **Post: #24**



Valentin Albillo
Senior Member

Posts: 636
Joined: Feb 2015
Warning Level: 0%

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

.
Hi, J-F:

J-F Garnier Wrote: (05-20-2018 03:50 PM)
I do also prefer the HP71, but the HP75 is an interesting machine, too.
Especially the 16kB Math module is very good, much more powerful than the HP80 series Matrix ROM.

I'd love to have a look at its *Owner's Handbook* to ascertain what it does and what it doesn't do. Just for instance, I think comparing it to the *HP80 series Matrix ROM* is quite unfair. The *Matrix ROM* does just that, matrix handling, so it's no surprise it doesn't delve with complex suppport, integrals, or root-finding, that's simply out of its scope.

For what it does, matrices, I wonder if the *HP-75 16 Kb ROM* had even a fraction of its functionality. I don't remember that it did, and that's why I'd love to see its manual. The *HP-71B Math ROM* is also quite inferior in that regard, having much less functionality in its matrix capabilities as compared to the *HP80 series ROM*.

> **Quote:**
> It has matrix functions, complex number support (**although not as nicely integrated than on the HP71**), [...]

Matter of fact, the complex number support isn't integrated at all. The *HP-75* mainframe has no provision whatsoever for complex number support (unlike the *HP-71B*, which does) and so there's no way to integrate it, nicely or not. I did try those capabilities at the time and found them severely lacking and awkward to use.

> **Quote:**
> [...]the PROOT polynomial root finder, the Fourier Transform and more important the FNROOT and INTEGRAL functions. **That is quite the same feature set than the 71**.

I seriously doubt it because the *HP-71B*'s is a **32Kb** ROM and the *HP-75C*'s is a **16 Kb** one. I don't think that *Capricorn* assembly language is 2 times more space-efficient than *Saturn* assembly language so I don't think that it could fit in 16 Kb what it takes 32 Kb in the *HP-71B*.

Back at the time I had an *HP-87XM* fitted with the *Assembler ROM* (among many others) ad 192 Kb RAM and I did tons of *Capricorn* assembly language *BIN* files, including a very large one implementing all kinds of matrix functionality (even *SORT*ing), printing utilities to speed raster graphics, CRT manipulation, the works, and I don't think the instruction set was *200%* more efficient space-wise.

> **Quote:**
> Or we may better say that the HP71 Math ROM included all the previous HP75 Math ROM features, adding a better integration with the mainframe (e.g. complex number) and improvements such as the re-entrant FNROOT and INTEGRAL functions.

The *Math ROM* article in the *HP Journal* says as much. It also says that it used the best algorithms from the *80 series Matrix ROM* and enhanced versions of the *HP-15C* algorithms.

> **Quote:**
> And with emu75 (very similar to emu71/DOS **that you know very well**), the first two drawbacks mentioned above are no more relevant :-)

Thanks for your kind words but I'm pretty sure you do know ***emu71/DOS*** better than me ... ;-D

Best regards.
V.
.
Edit to correct a mistake.

Find All My HP-related Materials here: **Valentin Albillo's HP Collection**

PM WWW FIND EDIT QUOTE REPORT

05-21-2018, 01:02 AM **Post: #25**

rprosperi
Senior Member

Posts: 4,439
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

> **Valentin Albillo Wrote:** (05-21-2018 12:01 AM)
> I seriously doubt it because the *HP-71B*'s is a **64 Kb** ROM and the *HP-75C*'s is a **16 Kb** one. I don't think that *Capricorn* assembly language is 4 times more space-efficient than *Saturn* assembly language so I don't think that it could fit in 16 Kb what it takes 64 Kb in the *HP-71B*.

The 75C/D has 48KB ROM in all, comprised of:

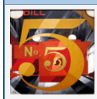
SYSROM - 24K
BASROM - 8K
ALTROM - 8K
MELROM - 8K

This is 'visible' by using the VER$ function which reports 'aaaaaa', 'bbbbbb', or 'dddddd', indicating the version letter for each of the 6 x 8K ROMs.

--Bob Prosperi

EMAIL PM FIND QUOTE REPORT

05-21-2018, 03:14 AM **Post: #26**



Valentin Albillo
Senior Member

Posts: 636
Joined: Feb 2015
Warning Level: 0%

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

.
Hi, **Bob**:

> **rprosperi Wrote:** (05-21-2018 01:02 AM)
> The 75C/D has 48KB ROM in all, comprised of:[...] This is 'visible' by using the VER$ function which reports 'aaaaaa', 'bbbbbb', or 'dddddd', indicating the version letter for each of the 6 x 8K ROMs.

My bad. I was referring to the sizes of the respective Math ROMs, not the System ones but I got the wrong 71B Math ROM size, it's 32 Kb, not 64 Kb, which I'll correct immediately in my post.

Thanks a lot. 4:15 a.m. here. Regards.
V.
.

Find All My HP-related Materials here: **Valentin Albillo's HP Collection**

PM WWW FIND EDIT QUOTE REPORT

05-21-2018, 03:42 AM **Post: #27**

rprosperi
Senior Member

Posts: 4,439
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

> **Valentin Albillo Wrote:** (05-21-2018 03:14 AM)
>
> .
> Hi, **Bob**:
>
> > **rprosperi Wrote:** (05-21-2018 01:02 AM)
> > The 75C/D has 48KB ROM in all, comprised of:[...] This is 'visible' by using the VER$ function which reports 'aaaaaa', 'bbbbbb', or 'dddddd', indicating the version letter for each of the 6 x 8K ROMs.
>
> My bad. I was referring to the sizes of the respective Math ROMs, not the System ones but I got the wrong 71B Math ROM size, it's 32 Kb, not 64 Kb, which I'll correct immediately in my post.
>
> Thanks a lot. 4:15 a.m. here. Regards.
> V.
> .

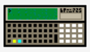
Happy to chat, and glad if it helped to clarify the discussion.

Though this probably could have waited until 6 or 7 am...

--Bob Prosperi

EMAIL PM FIND QUOTE REPORT

05-21-2018, 09:24 AM (This post was last modified: 05-21-2018 09:37 AM by J-F Garnier.) **Post: #28**



J-F Garnier
Senior Member

Posts: 461
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

> **Valentin Albillo Wrote:** (05-21-2018 12:01 AM)
> I'd love to have a look at its *Owner's Handbook* to ascertain what it does and what it doesn't do.

The handbook and QRG are available on the HP75 group (access to files for registered users only), together with the HP75 Math ROM **source file**!

> **Quote:**
> For what it does, matrices, I wonder if the *HP-75 16 Kb ROM* had even a fraction of its functionality. I don't remember that it did, and that's why I'd love to see its manual. The *HP-71B Math ROM* is also quite inferior in that regard, having much less functionality in its matrix capabilities as compared to the *HP80 series ROM*.

I dind't remember that the Series 80 Matrix ROM had such more functionalities. Maybe it's time for me to re-start my old HP85...

> **Quote:**
> Matter of fact, the [HP75] complex number support isn't integrated at all. The *HP-75* mainframe has no provision whatsoever for complex number support (unlike the *HP-71B*, which does) and so there's no way to integrate it, nicely or not. I did try those capabilities at the time and found them severely lacking and awkward to use.

Sure. Complex numbers are managed as 2-element arrays and, for instance, to add two complex numbers you must do something like:
MAT Z = CADD(Z1,Z2)
But at least it exists, if you need complex numbers, you don't have to write your own routines as you have to in the Series 80.

> **Quote:**
> > **Quote:**
> > [...]the PROOT polynomial root finder, the Fourier Transform and more important the FNROOT and INTEGRAL functions. **That is quite the same feature set than the 71**.
>
> I seriously doubt it because the *HP-71B*'s is a **32Kb** ROM and the *HP-75C*'s is a **16 Kb** one. I don't think that *Capricorn* assembly language is 2 times more space-efficient than *Saturn* assembly language so I don't think that it could fit in 16 Kb what it takes 32 Kb in the *HP-71B*.

This is surprising for me too. Even if the 71 Math LEX is only 27 kB long (the rest of the ROM is filled with 0), it makes a big difference of code size.
I intent to compare the features of the 71 and 75 Math ROM more in details. But it will be the subject of another thread.

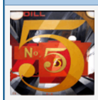
> **Quote:**
> Back at the time I had an *HP-87XM* fitted with the *Assembler ROM* (among many others) ad 192 Kb RAM and I did tons of *Capricorn* assembly language *BIN* files, including a very large one implementing all kinds of matrix functionality (even *SORT*ing), printing utilities to speed raster graphics, CRT manipulation, the works, and I don't think the instruction set was *200%* more efficient space-wise.

If you still have some material of that time, we at the HP Series 80 group would love to see it, and preserve it if you permit.

J-F

EMAIL PM WWW FIND QUOTE REPORT

05-21-2018, 03:48 PM **Post: #29**



Valentin Albillo
Senior Member

Posts: 636
Joined: Feb 2015
Warning Level: 0%

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

> **J-F Garnier Wrote:** (05-21-2018 09:24 AM)
> (... lotsa lotsa things ...)
> J-F

This is deviating too far from my S&SMC#23 so time for another dedicated thread.

I suggest you create it with some adequate Subject (say, "Math ROMs for 71B, 75C and Series 80") and include as its first post what you say in your latest here, then I'll comment on each of your points.

Regards.
V.
.

Find All My HP-related Materials here: **Valentin Albillo's HP Collection**

PM WWW FIND EDIT QUOTE REPORT

05-24-2018, 07:14 AM **Post: #30**

PeterP
Member

Posts: 101
Joined: Jul 2015

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

Dear Valentin, thank you for providing yet another of your most wonderful teaching exercises! While it was more focused on 71B but you - thankfully - left one to be tackled by other calculators as well. I had a long flight (actually two) so I tried by luck with it but alas, upon finishing a working code discovered that I had found the original thread after the deadline had past (I did not read the fine print nor any of the comments lest I spoil my pleasure).

My code takes a few minutes to deliver the first 9 digit 'Selfie' on my i41CX, is of course entirely clumsy and could use a true masters hand, but I wanted to ask for your opinion about posting it or not given that it is indeed past your suggested deadline.

In any case, I am very thankful for spending yet again an incredible amount of time and effort in concocting, creating, testing, and then wrapping in a nice story one of your wonderful S&SMC.

Many more you make, I hope.

Cheers

PeterP

Cheers,

PeterP

PM FIND QUOTE REPORT

05-24-2018, 06:10 PM (This post was last modified: 06-02-2018 07:07 PM by Jeff O..) **Post: #31**

Jeff O.
Member

Posts: 185
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

> **Valentin Albillo Wrote:** (05-17-2018 10:12 PM)
>
> > **Jeff O Wrote:**
> > My usual inclination with such problems is to just go ahead and try for a brute force method, then try to optimize. The DM42 is fast, I would like to see how many digits it could handle in a reasonable time by brute force, so if I get the time I may have a go at it.
>
> > **Valentin Albillo Wrote:** (05-17-2018 10:12 PM)
> > Please do. I haven't created a solution for the *DM42* but I guesstimate that with correct programming it can solve the 11-digit version in 5 min. to 1 hour running time.
>
> **PeterP Wrote:** (05-24-2018 07:14 AM)
> ...but I wanted to ask for your opinion about posting it or not given that it is indeed past your suggested deadline.

Peter,
Based on Valentin's "please do" stated above, I'm going on the assumption that it is OK to post. If interested, see below, which details my "clumsy" solution.

As a start, I went ahead and created a brute-force program with which I identified the 30 selfies from 1 to 10 digits:

1
2
3
4
5
6
7
8
9
173
351
704
4361
4749
8028
48039
72729
84745
438845
5136299
5271471
7180089
8180124
15087642
77439588
351589219
579533274
638494435
802115641
4777039764

For the sake of completeness, here are the 11-digit selfies, determined using a revised program as described in my post below:
15 694 046 123
52 249 382 004
30 609 287 624
97 653 680 744
60 605 588 394
87 561 939 628
41 919 540 249

I implemented this program on Free42 with the intent to download to my DM42 to see how fast it might go on that machine. The above results were obtained running Free42 on my desktop PC. Output was to the virtual printer. It produces the first 19 (i.e., 6-digit or fewer selfies) nearly instantaneously, then slows down considerably. Takes about maybe 2.5 minutes to get the 7-digit, then maybe 25 minutes for the 8-digit, and *awhile* for the 9-digit. I let the program run most of yesterday and then overnight, and this morning was rewarded with the lone 10-digit selfie. It looks like finding the seven 11-digit selfies would take about 20 days, so I think I'll probably wait until I figure out some optimized method to find those rather than continuing to run my program.

My brute force method does not look directly for selfies, it looks for numbers which have the property that if you sum its N digits raised to the Nth power you get the original number back (let's call them inverse selfies), then it simply reverses those to create the selfie. I quickly found that not all inverse selfies will be a selfie when reversed. Specifically those that end in zero will not, since when an N digit number ending in zero is reversed, it becomes an N-1 digit number. I thought that perhaps filtering out numbers ending in zero, i.e., not summing their digits to the Nth power to see if they were inverse selfies and so reducing the quantity of numbers to be checked by 10%, might speed things up. Unfortunately, performing that check on every number seemed to take longer, or at least was no quicker, than summing the digits to the Nth power for all numbers and then checking only inverse selfies to see if they end in zero. (I found two such numbers, 370 and 24678050, before I revised the program to eliminate them.) In any case, a 10-fold or more increase in speed is really needed to make this practical.

I can see some ways to determine that some numbers need not be checked, for example, no 10-digit number with three or more nines need be checked since all those will sum to an 11 digit number, no 10 digit number with six as the largest digit can be a selfie since thoese will not sum to a 10 digit number. I'll keep thinking about the problem to see if I can develop a method that will be much quicker - hopefully it won't keep me awake at night.

Here is the brute force program listing:

```
00 { 90-Byte Prgm }
01▸LBL "VA6_6"
02 CLA
03 CF 29
04 FIX 00
05 RCL 02
06 ARCL ST X
07 ALENG
08 STO 00
09 0
10▸LBL 04
11 ATOX
12 X=0?
13 GTO 05
14 48
15 -
16 RCL 00
17 Y↑X
18 +
19 GTO 04
20▸LBL 05
21 CLX
22 RCL 02
23 X≠Y?
24 GTO 08
25 10
26 ÷
27 FP
28 X=0?
29 GTO 08
30 ARCL ST Y
31 0
32 STO 01
33▸LBL 06
34 ATOX
35 X=0?
36 GTO 07
37 48
38 -
39 RCL 01
40 10↑X
41 ISG 01
42 DEG
43 ×
44 +
45 GTO 06
46▸LBL 07
47 R↓
48 PRX
49▸LBL 08
50 ISG 02
51 DEG
52 GTO "VA6_6"
53 END
```

edited to correct typo and add clarity to one item
edit no. 2 - added the 11-digit selfies to the list

Dave - My mind is going - I can feel it.

PM FIND QUOTE REPORT

05-24-2018, 10:13 PM **Post: #32**

Valentin Albillo
Senior Member

Posts: 636
Joined: Feb 2015
Warning Level: 0%

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

.
.
Hi, **PeterP** and **Jeff O.**:

> **PeterP Wrote:** ⇒ (05-24-2018 07:14 AM)
> Dear Valentin, thank you for providing yet another of your most wonderful teaching exercises!

Long time no see, **PeterP** !! I missed you ! ... Glad to see you taking part in one of my recent *S&SMC*'s as you so frequently did in the past.

> **Quote:**
> While it was more focused on 71B but you - thankfully - left one to be tackled by other calculators as well.

Yes, I was fearing that it was too 71B centric and regrettably the **HP-71B** seems placed in a *no-man's land* being the only calc-sized BASIC model, not RPN, not RPL, so it seems it's got very few fans and thus I provided a *plan B* by including last a challenge that could be solved in most calcs. My fears proved real so I'm glad I did.

> **Quote:**
> upon finishing a working code discovered that I had found the original thread after the deadline had past (I did not read the fine print nor any of the comments lest I spoil my pleasure).

Actually, there are no hard deadlines, most especially for a challenge that no one posted anything about and thus for which I also posted no solution. In such a case anyone is welcome to post anything at any time.

> **Quote:**
> My code takes a few minutes to deliver the first 9 digit 'Selfie' on my i41CX, is of course entirely clumsy and could use a true masters hand, but I wanted to ask for your opinion about posting it or not given that it is indeed past your suggested deadline.

See above. I'm eager to see your code so post it here at your earliest convenience, and if possible include timings.

> **Quote:**
> In any case, I am very thankful for spending yet again an incredible amount of time and effort in concocting, creating, testing, and then wrapping in a nice story one of your wonderful S&SMC. Many more you make, I hope.

You're welcome, thanks a lot for your everlasting appreciation. I have several *S&SMC* plus assorted *Mini-Challenges* ready to post at a moment's notice.

> **Jeff O. Wrote:**
> As a start, I went ahead and created a brute-force program with which I identified the 30 selfies from 1 to 10 digits:

I'm glad that you decided to give it a go, as you say you would. Brute-force or not your results are correct so congratulations, you're the first one (and so far the only one until **PeterP** posts his code) to solve it so my most sincere congratulations.

> **Quote:**
> The above results were obtained running Free42 on my desktop PC. It produces the first 19 (i.e., 6-digit or fewer selfies) nearly instantaneously, then slows down considerably. [...] It looks like finding the seven 11-digit selfies would take about 20 days, so I think I'll probably wait until I figure out some optimized method to find those rather than continuing to run my program.

Wise decision. Brute-force usually takes (generic) you so far, then you must think harder in order to beat the exponential curse.

> **Quote:**
> In any case, a 10-fold or more increase in speed is really needed to make this practical. [...] I'll keep thinking about the problem to see if I can develop a method that will be much quicker - hopefully it won't keep me awake at night.

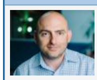
Please do. I'll post my original solution at 23:xx (GMT+1) next Sunday so you've got plenty of time to refine and expand your own.

Thanks to both and best regards.
V.
.

Find All My HP-related Materials here: **Valentin Albillo's HP Collection**

PM WWW FIND EDIT QUOTE REPORT

05-25-2018, 02:57 AM **Post: #33**



PeterP
Member

Posts: 101
Joined: Jul 2015

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

Thank you VA for your kind encouragement to post, it was a pleasure to work on for a flight (and a night) so I'm glad I get to share.

Another way to define the selfie is an n-digit number which is identical to the sum of the n-the power of its digits, but does not end in a 0.

To constrain the search space, I used two features:
(1) One can always rank order the digits of a number in a monotonously falling fashion (each digit is smaller or equal than the prior one
(2) As soon as one has a sum of n-th power of digits which is larger than 10^n, one can stop as all combination of digits to the right will yield unqualified results.

The combined application of the above allows to cut of quite substantial swaths of the search tree.

The implementation is based on the specific limitations of the HP41, namely:
1) It can only deal with at most 6 subroutine levels. This makes a recursive approach unfortunately not possible on the 41, yet I would not be surprised if this is possible, indeed advisable for the 71b
2) The 41 is very very slow in dealing with direct number entries. As such, virtually all numbers are stored in registers for faster processing
3) Akin to the JPC rom, my version uses my beloved Sandbox module, for the use of INCX, DECX, and AINT
4) Once a number has been found with a sum of the n-th powers of its digits between 10^n and 10^(n-1), the number is then checked to see if it is a selfie, aka the sum of *its* digts raised to the n-th power, using AINT and ATOX.

The code below takes as an entry the number of digits and then runs until all n-digit selfies are found. Adding a loop over all digits would be trivial but was not done for the purpose of easier exploration of the results of the code, timing, etc.

Code:

```
LBL 'VASSMC'
CLRG
STO 00        ''store n-digit
10^x
STO 40        ''store upper limit
10
/
STO 42        ''store lower limit
48
STO 48        ''store const for fast conversion from ASCII to number
```

Very much looking forward to comments, in particular ways to make it smarter.

Cheers,

PeterP

PM FIND QUOTE REPORT

06-02-2018, 07:01 PM Post: #34

Jeff O.
Member

Posts: 185
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

> **Valentin Albillo Wrote:** (05-24-2018 10:13 PM)
>
> > **Jeff O. Wrote:**
> > In any case, a 10-fold or more increase in speed is really needed to make this practical. [...] I'll keep thinking about the problem to see if I can develop a method that will be much quicker - hopefully it won't keep me awake at night.
>
> Please do. I'll post my original solution at 23:xx (GMT+1) next Sunday so you've got plenty of time to refine and expand your own.
>
> Thanks to both and best regards.
> V.

So as to not use anyone else's concepts for the time being, I have not reviewed Peter's work. Should anything I say or present below be a repetition of concepts presented by Peter, I fully acknowledge his priority.

In an effort to reduce execution time, I developed a method which basically checks 10 numbers at a time. I guess a better way to describe it would be to say that it determines if there is a solution between successive numbers that end in zero.

Since it is still basically just brute force, i.e., does not break any ground in attacking the problem in a super-efficient manner, I won't explain the procedure in detail. If anyone wants an explanation, I can provide it.

Since it effectively checks 10 numbers at a time, it seemed like a program based on the procedure should theoretically be up to 10 times faster than the previous program, as it checks all N+1 digit numbers by counting up to N. In practice, it appears that the various manipulations required to implement it eliminate some of the time saving. The speed-up seems to be more like a factor of 4 to 5. But with that improvement, I went ahead and turned it loose to find the 11-digit selfies. After (only) several days (again, on Free42 running on a couple of PCs), it found the seven 11-digit selfies:

15 694 046 123
52 249 382 004
30 609 287 624
97 653 680 744
60 605 588 394
87 561 939 628
41 919 540 249

The new program listing is presented below. Unfortunately, this program does not technically meet the original challenge. It cannot determine the single digit selfies from 1 through 9 since if you start counting at 1, it is checking candidates starting at 10. I could add code to just print out 1 through 9 at the start, but that seems unnecessary.

As noted, this is still essentially just brute force, and would be totally impractical on a DM42. Running on a physical machine in minutes or hours would require a massive speed-up factor. I'll keep trying to think of some other method to attack the problem that would be faster, but there's not much time until Sunday. Then I'll have to decide if I want to admit defeat and review Valentin's solution, or let it haunt me the rest of my days...

```
00 { 112-Byte Prgm }
01▸LBL "SELF"
02 CLA
03 CF 29
04 FIX 00
05 RCL 02
06 ARCL ST X
07 10
08 ×
09 ALENG
10 1
11 +
12 STO 00
13 R↓
14▸LBL 04
15 ATOX
16 X=0?
17 GTO 05
18 48
19 -
20 RCL 00
```

```
21 Y↑X
22 -
23 X<0?
24 GTO 08
25 GTO 04
26▸LBL 05
27 R↓
28 RCL ST X
29 RCL 00
30 1/X
31 Y↑X
32 1
33 +
34 IP
35 STO 11
36 RCL 00
37 Y↑X
38 RCL- 11
39 X≠Y?
40 GTO 08
41 10
42 RCL× 02
43 RCL+ 11
44 ARCL ST X
45 0
46 STO 01
47▸LBL 06
48 ATOX
49 X=0?
50 GTO 07
51 48
52 -
53 RCL 01
54 10↑X
55 ISG 01
56 DEG
57 ×
58 +
59 GTO 06
60▸LBL 07
61 R↓
62 PRX
63▸LBL 08
64 ISG 02
65 DEG
66 GTO "SELF"
67 END
```

Dave - My mind is going - I can feel it.

PM FIND QUOTE REPORT

06-02-2018, 11:37 PM **Post: #35**

Valentin Albillo
Senior Member

Posts: 636
Joined: Feb 2015
Warning Level: 0%

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

.
Hi, **Jeff O.**:

> **Jeff O. Wrote:** (06-02-2018 07:01 PM)
>
> In an effort to reduce execution time, I developed a method which basically checks 10 numbers at a time. I guess a better way to describe it would be to say that it determines if there is a solution between successive numbers that end in zero. [...] the speed-up seems to be more like a factor of 4 to 5. [...] After (only) several days (again, on Free42 running on a couple of PCs), it found the seven 11-digit selfies:

Congratulations, it's quite an achievement and I thank you for your efforts on this last part of my *S&SMC#23*. A factor of 400-500% faster over what sheer brute force produces is certainly a remarkable improvement.

> **Quote:**
>
> Unfortunately, this program does not technically meet the original challenge. It cannot determine the single digit selfies from 1 through 9 since if you start counting at 1, it is checking candidates starting at 10. I could add code to just print out 1 through 9 at the start, but that seems unnecessary.

It certainly is. The meat of the challenge is to produce the many-digit selfies, not the trivial ones. Insisting on that would be nitpicking on my part, which I don't usually indulge in.

> **Quote:**
>
> As noted, this is still essentially just brute force, and would be totally impractical on a DM42. Running on a physical machine in minutes or hours would require a massive speed-up factor

Indeed. As far as I know, there are at least *three* ways to attack this problem. The first and most obvious is sheer brute force (*very* brute), but that stumbles at 10 or 11 digits at most and even then it takes excessively long times.

The second and third ways depend on the same idea but implement it differently and both reduce exponentially the required times, to the point that 11 digits can be reached in *Emu71* in a few minutes, and using somewhat slow multiprecision software running on a decidedly slow old PC they can still find all the solutions with 10, 20, 30 and more digits in a few hours at most.

> **Quote:**
>
> I'll keep trying to think of some other method to attack the problem that would be faster, but there's not much time until Sunday. Then I'll have to decide if I want to admit defeat and review Valentin's solution, or let it haunt me the rest of my days...

Time is not a problem, I'm also no nitpicker with deadlines and such. If you need more time, I'll glady postpone posting my solutions. Also, if you'd accept a *hint* or two in order to be able to implement the 2nd or 3rd ways, I'll gladly obligue. Perhaps it would spoil somewhat the pleasure of finding the idea by yourself but in the other hand it's also quite pleasurable to create a markedly non-trivial working solution starting from a little hint instead of sorely admitting defeat after so much effort and time.

Your choice ... :-D

Again, thanks for your interest and your great efforts, have a nice weekend.
V.
.

Find All My HP-related Materials here: **Valentin Albillo's HP Collection**

PM WWW FIND EDIT QUOTE REPORT

06-03-2018, 05:52 AM **Post: #36**

Warbucks
Junior Member
Posts: 15
Joined: Mar 2018

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

The original problem is a BBP problem using PSLQ as a tool (integer relations finding) is it not?

EMAIL PM FIND QUOTE REPORT

06-03-2018, 05:34 PM **Post: #37**

Jeff O.
Member
Posts: 185
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

> **Valentin Albillo Wrote:** (06-02-2018 11:37 PM)
> Time is not a problem, I'm also no nitpicker with deadlines and such. If you need more time, I'll glady postpone posting my solutions. Also, if you'd accept a *hint* or two in order to be able to implement the 2nd or 3rd ways, I'll gladly obligue. Perhaps it would spoil somewhat the pleasure of finding the idea by yourself but in the other hand it's also quite pleasurable to create a markedly non-trivial working solution starting from a little hint instead of sorely admitting defeat after so much effort and time.
>
> Your choice ... :-D
> Again, thanks for your interest and your great efforts, have a nice weekend.
> V.

Thanks for your kind words about my efforts. Sure, I'll take a little more time and a hint or two. I'll consider it a learning experience rather than a failing.

Dave - My mind is going - I can feel it.

PM FIND QUOTE REPORT

06-04-2018, 10:19 PM **Post: #38**

Valentin Albillo
Senior Member
Posts: 636
Joined: Feb 2015
Warning Level: 0%

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

.
Hi, **Jeff O.**:

> **Jeff O. Wrote:** (06-03-2018 05:34 PM)
> Thanks for your kind words about my efforts. Sure, I'll take a little more time and a hint or two. I'll consider it a learning experience rather than a failing.

In 5 minutes, check your Private Messages. :-)

Regards.
V.
.

Find All My HP-related Materials here: **Valentin Albillo's HP Collection**

PM WWW FIND EDIT QUOTE REPORT

06-06-2018, 02:48 AM **Post: #39**

Jeff O.
Member
Posts: 185
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

> **Valentin Albillo Wrote:** (06-04-2018 10:19 PM)
> In 5 minutes, check your Private Messages. :-)
>
> Regards.
> V.
> .

Thank you, message received. I'll see what I can do with the information.

Dave - My mind is going - I can feel it.

PM FIND QUOTE REPORT

06-07-2018, 04:10 AM **Post: #40**

Valentin Albillo
Senior Member

Posts: 636
Joined: Feb 2015
Warning Level: 0%

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

> **Warbucks Wrote:** (06-03-2018 05:52 AM)
> The original problem is a BBP problem using PSLQ as a tool (integer relations finding) is it not?

I don't get you. What *"original problem"* are you referring to ?

V.
.

Find All My HP-related Materials here: **Valentin Albillo's HP Collection**

PM WWW FIND EDIT QUOTE REPORT

06-22-2018, 09:20 PM **Post: #41**

Jeff O.
Member

Posts: 185
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

With some coaching and insights from Valentin, I attacked the problem from a more efficient direction and managed to create a program which is at least three orders of magnitude faster than my previous brute-force efforts. The programs below calculate the 37 selfies from 1 to 11 digits in approximately 1 minute, 6 seconds on my probably not too fast desktop. Extrapolating run times, I calculated that it could run in something less than 3 hours on my DM42. So I plugged it into a USB power source and set it off. Two hours, 34 minutes later it completed the task, so it apparently is a realistic challenge for a physical machine.

The most useful information that Valentin provided was that it is not necessary to check and sum every number. While there are 89,999,999,999 11 digit numbers from 10,000,000,000 to 99,999,999,999, many, many, many of these contain the same digits and so will have the same sums to the 11th power. So if a way can be found to sum each combination just once, that will greatly reduce the quantity of numbers needing to be checked. (e.g., 54321 is the same as 43215 is the same as 12345 etc., so only one of these need be summed.) I think I actually realized this while working on the brute-force methods, but saw no easy way to generate only the minimal set of combinations of digits. With Valentin's encouragement that this was the way to go, I was able to develop a method to sequentially generate the smallest combination of the digits for each set of n digits. The method may be difficult to decipher from the program listing (it happens in steps 10 through 42). I doubt that the method is particularly insightful, so in an effort to keep this post as short as possible, I won't describe it. I can do so if there is any interest.

Once the sum for a given set of n digits is created, you just have to see if its sum of digits to the nth power contains the same set of digits as the input number. If so, the reverse of that sum is a selfie. I did this in what I am sure is a very clunky fashion:

- Break the number into its separate digits
- Sort the digits
- Reassemble into a number
- Compare to input number
- if equal, reverse the sum, that is the selfie
- if not equal, not a selfie, go back to create the next combination to check

Due to the way I "manufacture" the numbers to check, having to do with how I handled numbers with zeros in them, my program generates the selfies in the following order:

1
2
3
4
5
6
7
8
9
704
351
173
8028
4361
48039
4749
7180089
72729
84745
8180124
438845
5271471
5136299
15087642
802115641
77439588
351589219
52249382004
30609287624
579533274
638494435
4777039764
60605588394
15694046123
41919540249
97653680744
87561939628

The program is far from optimized, it would probably be possible to improve on the performance. But I believe that Valentin plans to wrap this one up soon, and I'm not sure I will have the time to try for further improvement. I wanted to get an "entry" in, so I'll go ahead and post this version. If I do find time and am able to improve the program before Valentin finalizes things, I'll provide an update.

```
00 { 249-Byte Prgm }
01▸LBL "NM5"
02 11
03 STO 00
04▸LBL 01
05 0
06 STO IND 00
07 DSE 00
08 GTO 01
09▸LBL 02
10 1.011
11 STO 00
12▸LBL 03
13 9
14 RCL IND 00
15 X≠Y?
16 GTO 04
17 ISG 00
18 GTO 03
19 STOP
20▸LBL 04
21 RCL 00
22 IP
23 STO 00
24 1
25 RCL+ IND 00
26▸LBL 05
27 STO IND 00
28 DSE 00
29 GTO 05
30 11
31 STO 00
32 0
33▸LBL 06
34 RCL 00
35 1
36 -
37 10↑X
38 RCL× IND 00
39 +
40 DSE 00
41 GTO 06
42 STO 12
43 CLA
44 CF 29
45 FIX 00
46 ARCL ST X
47 ALENG
48 STO 00
49▸LBL 09
50 0
51▸LBL 07
52 ATOX
53 X=0?
54 GTO 08
55 48
56 -
57 RCL 00
58 Y↑X
59 +
60 GTO 07
61▸LBL 08
62 R↓
63 STO 14
64 LOG
65 IP
66 1
67 +
68 RCL 00
69 X≠Y?
70 GTO 10
71 RCL 14
72 10
73 ÷
74 FP
75 X=0?
76 GTO 10
77 ARCL 14
78 RCL 00
79 20
80 +
81 1000
82 ÷
83 21
84 +
85 STO 16
86 STO 15
87▸LBL 11
88 ATOX
89 X=0?
90 GTO 13
91 48
92 -
93▸LBL 13
```

```
94 STO IND 15
95 ISG 15
96 GTO 11
97 ISG 13
98 DEG
99 XEQ "SORT"
100 20.02
101 RCL+ 00
102 STO 15
103 0
104▸LBL 12
105 RCL 15
106 21
107 -
108 IP
109 10↑X
110 RCL× IND 15
111 +
112 DSE 15
113 GTO 12
114 RCL 12
115 X≠Y?
116 GTO 10
117 RCL 14
118 STO- 14
119▸LBL 14
120 FP
121 STO+ 14
122 LASTX
123 IP
124 0.1
125 STO÷ 14
126 ×
127 X=0?
128 GTO 15
129 GTO 14
130▸LBL 15
131 RCL 14
132 PRX
133▸LBL 10
134 11
135 RCL 00
136 X=Y?
137 GTO 02
138 RCL 12
139 ARCL ST X
140 ISG 00
141 DEG
142 GTO 09
143 STOP
144 END
```

```
00 { 51-Byte Prgm }
01▸LBL "SORT"
02 RCL 16
03 SIGN
04▸LBL 21
05 LASTX
06 LASTX
07 RCL IND ST L
08▸LBL 22
09 RCL IND ST Y
10 X<Y?
11 GTO 23
12 X<>Y
13 LASTX
14 +
15▸LBL 23
16 R↓
17 ISG ST Y
18 GTO 22
19 X<> IND ST L
20 STO IND ST Z
21 ISG ST L
22 GTO 21
23 RTN
24 END
```

Credit to Gamo for the reversing integer routine in steps 117 through 129.
Credit to Jean-Marc Baillard for the SORT subroutine.

Dave - My mind is going - I can feel it.

PM FIND QUOTE REPORT

06-24-2018, 03:57 PM **Post: #42**

Thomas Klemm
Senior Member

Posts: 1,447
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

Here's a Python program for the sixth step:

Code:

```
def selfie(m, i, k, n, s):
    if i == 0:
        if n == ''.join(sorted(str(s))):
            print str(s)[::-1]
```

```
    else:
        for j in range(k, 10):
            selfie(m, i - 1, j, n + str(j), s + j ** m)

for m in range(12):
    selfie(m, m, 0, '', 0)
```

The following 41 selfies are listed:

Code:
```
0
1
2
3
4
5
6
7
8
9
073
```

It takes about 3 seconds to run:

Code:
```
real    0m2.805s
user    0m1.942s
sys    0m0.054s
```

Four of them start with 0:

Code:
```
0
073
05087642
05694046123
```

They might not be considered selfies which agrees with:

> **Quote:**
> all 37 Selfies up to 11 digits long

PM FIND QUOTE REPORT

06-25-2018, 01:42 PM **Post: #43**

John Keith
Senior Member

Posts: 615
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

> **Thomas Klemm Wrote:** (06-24-2018 03:57 PM)
> Here's a Python program for the sixth step:
>
> **Code:**
> ```
> def selfie(m, i, k, n, s):
>     if i == 0:
>         if n == ''.join(sorted(str(s))):
>             print str(s)[::-1]
>     else:
>         for j in range(k, 10):
>             selfie(m, i - 1, j, n + str(j), s + j ** m)
>
> for m in range(12):
>     selfie(m, m, 0, '', 0)
> ```

I gather that Valentin's original definition of a selfie as the *reverse* of the sum of powers is aimed at eliminating numbers which end in zero. I'm not a Python expert, but if you amend the third line of your program to test that the last digit of s is not zero, it will return Valentin's original 37 numbers (at some cost in execution speed).

John

EMAIL PM FIND QUOTE REPORT

06-25-2018, 02:41 PM **Post: #44**

Thomas Klemm
Senior Member

Posts: 1,447
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

> **John Keith Wrote:** (06-25-2018 01:42 PM)
> I gather that Valentin's original definition of a selfie as the *reverse* of the sum of powers is aimed at eliminating numbers which end in zero.

Not sure about that. Might as well be to make it a bit more difficult.

> **Quote:**
> I'm not a Python expert, but if you amend the third line of your program to test that the last digit of s is not zero, it will return Valentin's original 37 numbers (at some cost in execution speed).

Or then you simply filter them out after the computation:

Code:
```
python selfie.py | grep -v ^0
```

07-18-2018, 01:17 AM **Post: #45**

Valentin Albillo
Senior Member

Posts: 636
Joined: Feb 2015
Warning Level: 0%

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

.
Hi, all:

This is my **200th** post so it's perfectly fitting to wrap up here and now my ***S&SMC#23*** by posting my very own solution to the final *6th Step*, namely:

Step the Sixth:

- *"We'll call "Selfie" to any positive N-digit integer number which has the property that if you sum its N digits raised to the Nth power you get the original number backwards. For instance, the 7-digit number 5271471 is a Selfie:*

  5271471 => 5^7 + 2^7 + 7^7 + 1^7 + 4^7 + 7^7 + 1^7 = 1741725, which is 5271471 backwards

  Write a program to find all Selfies from 1 to 9 digits long (for 10-digit HP calcs, 29 in all) or from 1 to 11 digits long (for 12-digit HP calcs, 37 in all). 0 is not a positive number so it's not a Selfie."

My Solution:

This is my solution for the **HP-71B**, a 12-line program (437 bytes) which finds and outputs all *Selfies* from 1 up to 11 digits long in less than 3 min on my *POPS* system.:

```
 1  DESTROY ALL @ OPTION BASE 0 @ DIM R(9) @ L=48 @ FOR K=1 TO 11
 2  DIM F(K),S(K),D$(K) @ DISP K;": "; @ M=10^K @ T=M/10
 3  H=0 @ FOR I=1 TO 9 @ R(I)=I^K @ NEXT I
 4  H=H+1 @ F(H)=F(H-1)
 5  I=F(H) @ N=S(H-1)+R(I) @ IF N<M THEN S(H)=N @ D$(H)=D$(H-1)&CHR$(I+L) ELSE 11
 6  IF H#K THEN 4 ELSE IF N<T THEN 10 ELSE B$=STR$(N) @ A$=D$(H)
 7  IF SPAN(B$,A$) THEN 10 ELSE IF SPAN(A$,B$) THEN 10
 8  FOR I=1 TO K @ P=POS(A$,B$[I,I]) @ IF P THEN A$[P,P]="" ELSE 10
 9  NEXT I @ B$=REV$(B$) @ IF B$=TRIM$(B$,"0") THEN DISP B$;" ";
10  IF F(H)#9 THEN F(H)=F(H)+1 @ GOTO 5
11  H=H-1 @ IF H THEN 10
12  DISP @ NEXT K
```

Notes:

- We only search for numbers having from 1 to 11 digits because for 12-digit numbers there are some whose sum of the 12th-powers of their digits exceeds 10^12 and thus cannot be checked correctly with 12-digit integer arithmetic. For instance:

  888888888888 -> Sum = 12*8^12 = 824633720832, which is still within range and can be correctly checked
  888888888889 -> Sum = 11*8^12+1*9^12 = 1.03834378058E12, which *exceeds* the integer range and *can't* be checked properly

  thus the search is limited to 11-digit numbers or less (there are no 12-digit Selfies anyway). The same applies to 10-digit calcs, which can only search for numbers up to 9-digit long.
- *Line 7* uses the *SPAN* keyword from *STRNGLEX* to help speed the search but it's use is optional and can simply be deleted. Without it the program is *8% shorter* (11 lines, 371 bytes) but *18% slower* (200 sec. vs. 170 sec.)

Let's run it:

```
>RUN
         1 : 1 2 3 4 5 6 7 8 9
         2 :
         3 : 704 351 173
         4 : 8028 4361 4749
         5 : 48039 72729 84745
         6 : 438845
         7 : 7180089 8180124 5271471 5136299
         8 : 15087642 77439588
         9 : 802115641 351589219 579533274 638494435
        10 : 4777039764
        11 : 52249382004 30609287624 60605588394 15694046123 41919540249 97653680744 87561939628
```

There are ***37*** *Selfies* up to 11 digits long in all (for 12-digit calcs) and ***29*** *up to 9 digits long* (for 10-digit calcs). There are no 12-digit *Selfies*.

The program uses my *generalized loops* (first featured in ***S&SMC#21***) to perform an exhaustive search for *Selfies*. The key to speed the search enormously is to check as few numbers as necessary (this will reduce the search time *exponentially*), then to implement minor but welcome optimizations which will further reduce the seach time by a significant *linear* factor.

The procedure goes like this: for N-digits numbers, there's no need to check every number from *00...00* to *99...99*, we only need to check the *smallest* N for each permutation of its N-digits. This alone reduces the search exponentially, as stated. Let's see an example with 2-digit numbers:

For 2-digit numbers, in a naive exhaustive search we would simply compute the sum of the 2nd power (squares) of *every* number from *00* to *99* and see if the sum has the *Selfie* property, i.e., it equals the original number in reverse. If so, it is displayed as a solution and we would then proceed to the next 2-digit number and so on. We would check *10^2 = 100* numbers in all, the 100 values from *00* to *99*.

However, we might notice that *12* and *21* have the exact same sum of their squared digits because addition is conmutative: ***Sum(12)*** *= 1^2+2^2* ***= 5*** *= 2^2+1^2* ***= Sum(21)***, so we don't actually need to compute and check the sums for every digit permutation, checking just the first one *(12)* will do, no need to also check *21*.

This, combined with the fact that the sum must be 2-digit too (so we need to check only those numbers which have 2-digit sums, i.e.: *10 >= Sum(N) <= 99*) means that instead of the ***100*** numbers from *00* to *99* we only need to check these ***40***:

```
04 05 06 07 08 09 13 14 15 16 17 18 19 23 24 25 26 27 28 29
33 34 35 36 37 38 39 44 45 46 47 48 49 55 56 57 58 66 67 77
```

and thus the search has been reduced from *100* to just *40* numbers, i.e.: by a factor of *2.5x*. For greater number of digits the reduction factor increases *exponentially* and thus the time for the full search decreases exponentially as well.

What do we need to do to implement this concept ? Two things:

- **first**, we need to generate and check only the *lowest* value for every permutation of digits, i.e: for N = 2 digits we'll generate and check *17* but not *71*. For N=3 digits, we'll generate and check *047* but not its five permutations *(074, 407, 470, 704, 740)*, as they all have the same sum of the 3rd powers of their digits. This means that for N=10 digits, you'll only generate and check **one** of the *3,628,800 permutations* possible, thus speeding up the search by a factor of *~ 3.6 million*.
- **second**, for each generated number (say *047*) we have to check if the sum of the Nth powers of its digits is a permutation of the number being checked, i.e.; the sum of the cubed digits of ***047*** is *0^3+4^3+7^3* = ***407***, which indeed *is* a permutation of *047*, the number being checked, so the reverse of the sum, ***704***, is a *Selfie* and we display it as one of the 3-digit *Selfies*.

  This will reduce the search time exponentially. Combined with other optimizations (generating efficiently the numbers to check, computing efficiently the sums, skipping N-digit numbers with sums > or < N-digits, checking efficiently if an N-digit number is or not a permutation of another, etc.) will help to further reduce the time by a *linear* factor which can be *~ 10x* or more, i.e., further reducing an already fast 30 min. search to *less than 3 min*.

This is not the only efficient approach possible. There's another way to conduct the search by keeping *tallies* but my **HP-71B** implementation of that second approach isn't included here as it runs *slower* than the first (though for some non-*HP-71B* implementations it can run more than 2x faster).

Thanks for your interest and nice solutions, see you all in ***S&SMC#24*** next Autumn.
V.
.

Find All My HP-related Materials here: **Valentin Albillo's HP Collection**

PM WWW FIND EDIT QUOTE REPORT

07-18-2018, 02:10 AM **Post: #46**

rprosperi
Senior Member

Posts: 4,439
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

> **Valentin Albillo Wrote:** (07-18-2018 01:17 AM)
> This is my **200th** post so it's perfectly fitting to wrap up here and now my ***S&SMC#23*** by posting my very own solution to the final *6th Step*, namely...

Thanks for the original challenge but also for the several earlier, and now this detailed solution for the final section. Though most often I am not up these challenges of yours, I thoroughly enjoy pondering them (until I get frustrated) and then reading the various members' solutions. And your detailed explanations (like this one above) are excellent learning tools as they not only explain how you solved it, but also why you chose that technique and how it works, generally in a clear enough manner to see how to apply similar techniques in the future to other problems.

Also, I've noted that your very detailed solutions have often inspired others to also explain their solutions in a similar, detailed manner, which is far more helpful to subsequent readers than a (possibly well thought-out, but still) obscure code listing and a brief note claiming this solution is '3 bytes shorter' or some other similar claim to fame.

Thanks to all that participated and shared their answers here, it's quite interesting to many of us quiet readers.

--Bob Prosperi

EMAIL PM FIND QUOTE REPORT

10-24-2019, 02:26 PM (This post was last modified: 11-05-2019 01:47 PM by Jeff O..) **Post: #47**

Jeff O.
Member

Posts: 185
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

I'm guessing no one was expecting an update regarding further work on this, but a couple of recent things brought it back to my attention. First, the HHC 2019 programming contest was related, and second, Valentin's posting of his web site led me to re-read his challenge and my input. For posterity's sake, I decided to go ahead and add my new work to the original thread. (Apologies in advance for the length of this post, brevity is not a strength of mine.)

As noted in my old post above, I ended up solving the sixth part of Valentin's challenge, with optimizations to reduce the search space, but with the knowledge that other portions of the program were decidedly not optimized. I was particularly dissatisfied with my method to determine if the digits in the sum of the digits to the n-th power were a permutation of the digits in the n-digit input number:

> **Jeff O. Wrote:** (06-22-2018 09:20 PM)
> I did this in what I am sure is a very clunky fashion:
>
> \- Break the number into its separate digits
> \- Sort the digits
> \- Reassemble into a number
> \- Compare to input number
> \- if equal, reverse the sum, that is the selfie
> \- if not equal, not a selfie, go back to create the next combination to check

As I said, I found the above dissatisfying, but the program worked, Valentin wrapped up the challenge, and I returned to the mundane tasks of everyday life.

After the HHC 2019 programming contest was presented, I realized that it was related to Valentin's challenge, limited to 3-digit numbers and eliminating the "selfie" constraint, i.e., just looking for 3-digit numbers whose digits to the 3rd power summed to the input number, not the reverse. After creating a brute-force method to solve the programming contest, I revisited my "selfie" program to see how it might be used. First I removed the few lines that did the "selfie" part, and it quickly found the four answers to the contest. Then somewhere along the line I re-read my old post, remembered my dissatisfaction and decided to see if I could improve on the previous version, especially the permutation identification part quoted above. When I originally worked on the problem, I tried to think of a better way to do it. Doing it manually, I envisioned something like the following process:

1. Write down both numbers
2. Pick a digit in the first number, look for it in the second
3. If a is match found, mark out the digit from each number, go back to step 2. If all digits get checked and matched, go to step 5. If they do not match, go to step 4.
4. If a match for any digit in the input number cannot be found in the second, the numbers cannot be permutations of each other.
5. All digits found and uniquely matched, so the numbers are permutations of each other.

For example, check 12345 against 34521

Pick the 5 from 12345, matches the 5 from 34521, strike out both, leaving 1234X vs. 34X21. These might be re-written as 1234 and 3421.

Now check the 4, leaving 123X vs. 3X21 or 123 vs. 321, and so on until checking 1 vs. 1, and thus finding the two starting numbers to be permutations of each other.

Now check 12745 vs. 34521.

As above, 5 matches 5, leaving 1274X vs. 34X21 (or 1274 vs. 3421), 4 matches 4, leaving 127XX vs. 3XX21 (127 vs. 321). The next time around, 7 is not found in 321, so we abort the check and declare that the originals are not permutations.

(I'm sure the above described procedure is quite simple and obvious to MoHPC Forum members, but I find it useful to spell out even the simple things, so I may remember them more readily in the future should the need arise.)

For the purposes of the following discussion, the "first" number is the sum of the digits of the input number to the n-th power, and the "second" number is the input number.

I think I considered the above method back in 2018 but got hung up on how I might check each digit and delete matches. It sounded like a lot of breaking, shifting and re-storing that did not immediately seem to be much if any better than my dissemble-sort-reassemble method. That's when I dropped further work. When I returned to the problem recently, I revisited the above described manual procedure, and realized that rather than delete the digits as checked and repack to create new numbers, if I could find a way to identify that a match had been found and no longer check those digits, I would not need to reassemble at each step. It is relatively easy to extract single digits from the first number and create a number with one less digit, so each digit is eliminated from further checking at each step. For checking against the second, I determined that I did not need to eliminate the matches and reconstitute the second number without that digit, I just needed to make sure that a digit could not be matched again. So if a match is found, rather than delete and re-assemble, I changed that digit to a value that could not be a match to any further digits extracted from the original number. My input numbers were generated by the method I used (described below) as individual digits in sequential storage registers. So if a match was found, I stored pi in the register that held the matching digit. I was then free to check all future digits against all registers that represented the second number with no possibility that that digit could be matched again.

With a bit of work, I was able to implement the above method, and pleased to find that it worked quite well. The revised program is reduced to 120 steps compared to 143 of the original, plus the need for the separate 23 step SORT routine is eliminated. With the above major change and a couple of other improvements, the new version runs in less than half the time of the original. On my DM42, all selfies are found in 1 hour and 9 minutes vs. 2 hours 34 minutes of the original. With Free42 on my desktop, it finds them all in 18 seconds vs. 45 seconds for the original program.

I'm fairly happy with the improvement, but I may continue looking for refinements, as it is an enjoyable pastime. (edit - see below for an update to this post and the following post for a new method.)

Here is the code:

Code:

```
000       { 209-Byte Prgm }
001       LBL "Σd↑n"
002       11                  Steps 2 through 9...
003       STO 00              ...clear registers  1 through 11 and 20
004       0
005       STO 20              initialize register 20 which keeps track of number of digits in number
006       LBL 01
007       STO IND 00
008       DSE 00
009       GTO 01
010       LBL 02              Label 02, begin main routine
```

My number generator generates the following sequence of numbers to check (read top to bottom, left to right):

Code:

```
 1    13    26     44     79    118    226    399     ...    3333        ...      111111111          ...
 2    14    27    ...     88    119    227    444     799     ...       9999            ...  39999999999
 3    15    28     49     89    122    228    ...     888    4444      11111     1111111111  44444444444
 4    16    29     55     99    ...    229    499     ...     ...        ...            ...          ...
 5    17    33    ...    111    188    233    555     899    5555      99999    11111111111  49999999999
 6    18    34     59    112    189    ...    ...     999     ...     111111            ...  55555555555
 7    19    35     66    113    199    288    599    1111    6666        ...    19999999999          ...
 8    22    36    ...    114    222    289    666     ...     ...    1111111    22222222222  59999999999
 9    23    37     69    115    223    299    ...    1999    7777        ...            ...  66666666666
11    24    38     77    116    224    333    699    2222     ...   11111111    29999999999          ...
12    25    39     78    117    225    ...    777    2999    8888        ...    33333333333  99999999999
```

In case it is not immediately apparent what is going on with the above sequence of numbers to check, a description of the procedure used to create the sequence is as follows:
1. Start with all 11 registers representing the (up to) 11 digit number set to zero.
2. Recall the digit in the ones position
3. Is it 9?
4. If not, increment it, the 11 registers now represent the next number. Check for sum d^n being a permutation, then return to step 2.
5. If it is equal to 9, check the digit to the left in the tens position.
6. Is it 9?
7. If not, increment it and also set the value in the ones position to that new value, the 11 registers now represent the number to be evaluated.
8. If it is equal to 9, check the digit to the left in the hundreds position.
9. Is it 9?
10. If not, increment it and also set the values in the tens and ones positions to that new value, the 11 registers now represent the next number to be evaluated.
11. If it is equal to 9, check the digit to the left in the thousands position.
12. And so on.

To be honest, I'm not quite sure how I arrived at the above method to create the sequence of numbers. It took me a while to figure out how it worked and what I was doing in my program again after 15 months. Probably some well-known technique that I stumbled upon.

The full listing of all numbers generated by the above procedure would include 167,959 numbers. The interested reader will likely notice that the above listing contains no numbers that include any zeros. Yet there are several numbers containing at least one zero which satisfy the goal that the digits of the n-digit number raised to the n-th power sum to the n-digit number itself. I developed and checked those as follows. After generating and checking each of the above numbers, I then padded them with zeros. So for a single digit number, say 5, I checked 5, 50, 500, 5000,..., 50 000 000 000. I did not raise the zeros to the higher powers, I only raised the original non-zero digits to the higher powers and summed those. This requires checking 11 numbers for every single-digit number in the above list, 10 numbers for every 2-digit number, 9 numbers for each 3-digit, etc. on up to 1 number for each 11-digit number in the above list. That increases the total count of the numbers needing to be checked to 352,704. Still quite a small fraction (0.0003527%) of the 99,999,999,999 numbers which would have to be checked via a brute-force method.

Last but not least, attached is a zipped raw file in case you would like to play with the program in Free42 or your DM42. If you would like to simply determine those n-digit numbers whose digits raised to the n-th power sum to themself, i.e., remove the selfie constraint (and who wouldn't?), delete steps 93 through 109.

Edit - as expected, I continued to attempt to improve my program. In an effort to speed it up, I developed a new way to determine the number of digits in the current number developed by my above described method. It appears to be only about 1% faster, so not as much improvement as I had hoped. But it is 6 steps shorter, so I would call it a better effort. I replaced the program listing above with the new version, and have attached a new zipped raw file. The original raw file is still attached for posterity.

Attached File(s)

Sumd^n.zip (Size: 335 bytes / Downloads: 0)
Sumd^n V2.zip (Size: 335 bytes / Downloads: 0)

Dave - My mind is going - I can feel it.

PM FIND QUOTE REPORT

11-04-2019, 08:18 PM (This post was last modified: 11-05-2019 02:58 PM by Jeff O..) **Post: #**

Jeff O.
Member

Posts: 185
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

> **Jeff O. Wrote:** (10-24-2019 02:26 PM)
> ...but I may continue looking for refinements, as it is an enjoyable pastime. (edit - see below)

Obviously, I cannot leave well enough alone. Late last week, another approach to identify the “selfies” after summing the digits of the input number occurred to me. The new method replaces the “disassemble-sort-re-assemble” method of my much prior work, and the “check-and-mask-digit” method of my latest work. It occurred to me that after summing the digits to the nth power of my input number, all I had to do was sum the digits of that result to the n-th power and compare to the first sum of digits to the nth power. If they match, then I have found a selfie. (Actually a reverse of the selfie, which is then reversed to give the selfie after checking for a trailing zero.). My logic for this was as follows:

1. Let the digits^n of n-digit number A sum to n-digit number B
2. Let the digits^n of n-digit number B sum to n-digit number C
3. If B=C, then the digits^n of n-digit number C must also sum to n-digit number B
4. Therefore, number C (and B, of course) must contain the same digits as number A, i.e., is a permutation of the digits in A.

No. 4 is a conjecture, not an assertion. I guess I hoped that there was some well-known proof that the sum of n digits to the nth power is unique if it sums to an n-digit number. I had no reason to think there was such a proof, but a guy can hope, right? I exchanged a PM with Valentin, he quickly found some cases where a given number can be generated by more than one combination of digits to a power, so I assume my hoped-for proof does not exist.

But, absent the desired proof, I believe the worst that could happen is that my algorithm would find a given solution more than once, i.e. I don't think any solutions would be missed. That is based on the following reasoning. I’m fairly confident that my method checks one (and only one) example of each possible combination of n digits. So let’s say that an n-digit number produces a sum of d^n that is a selfie but is not composed of the digits of the input number. That result is still a selfie, so it will be reported. When the digits of that selfie are checked (before or after that instance), it would again produce that selfie.

To conclude, this new version is down to 100 steps, finds **all 37 selfies** for 1 to 11 digit numbers, and runs about 25% faster than the prior program presented in the post immediately above. Free42 completes the task in about 12 seconds, and my DM42 did so in 52 minutes. (It also finds no duplicates, so for up to 11 digits, my conjecture appears to be correct?)

Here is the code:

Code:

```
000        { 167-Byte Prgm }
001        LBL "Σd↑n"
002        11                  Steps 2 through 8 clear registers 1 through 11 and 20
003        0
004        STO 20              initialize register 20 which keeps track of number of digits in number
005        LBL 01
006        STO IND ST Y
007        DSE ST Y
008        GTO 01
009        LBL 02              Label 02, begin main routine
```

A zipped raw file is attached at the bottom. Here is the program listing outside of a "code" box to allow easy pdf creation:

```
000 { 167-Byte Prgm }
001 LBL "Σd↑n"
002 11            Steps 2 through 8 clear registers 1 through 11 and 20
003 0
004 STO 20        initialize register 20 which keeps track of number of digits in number
005 LBL 01
006 STO IND ST Y
007 DSE ST Y
008 GTO 01
009 LBL 02        Label 02, begin main routine
010 1.011         Store 1.011...
011 STO 00        ...for ISG to check digits
012 LBL 03        Label 3, check digits to see if they equal 9
013 9             Enter 9
014 RCL IND 00    recall digit
015 X≠Y?          is it not equal 9?
016 GTO 04        if not, go to routine to increment
017 ISG 00        if equal to 9, increment counter to...
018 GTO 03        ...loop back to check next digit
019 STOP          if register 11 equal 9, all unique combinations up to 99,999,999,999 have been checked
020 LBL 04        routine to increment registers
021 RCL 20        recall number of digits
022 RCL 00        recall loop counter
023 IP            take integer part since it has ISG target coded
024 X>Y?          is current digit pointer greater than current number of digits
025 STO 20        if so, store new number of digits
026 STO 00        store back in counter for DSE to increment registers
027 1             Enter 1
028 RCL+ IND 00   Recall and increment digit pointed to by counter
029 LBL 05        label for loop to set all digits up to counter to new value
030 STO IND 00    store new value
```

```
031 DSE 00        decrement counter
032 GTO 05        loop back to store new value in next lower digit position
033 RCL 20        recall number of digits in current input number
034 STO 00        store for power to raise digits to form sum d^n
035 LBL 07
036 RCL 00        Recall number of digits in input number, may be padded for inclusion of zeroes, use for power to raise digits to form sum d^n
037 RCL 20        recall number of digits in current input number for loop index to form sum d^n. Only sum digits greater than 1.
038 0
039 LBL 09        Label to sum d^n
040 RCL IND 18
041 RCL 00
042 Y↑X
043 +
044 DSE 18
045 GTO 09
046 STO 14        store sum d^n in register 14
047 LOG           take common log
048 IP            take integer part
049 1             Enter 1
050 +             Add 1 to determine number of digits
051 RCL 00        recall number of digits
052 X≠Y?          digits in original not equal digits in sum d^n?
053 GTO 10        if not equal, cannot be selfie, skip all checks
054 CLA           clear alpha register
055 FIX 00        Fix 0 to eliminate zeros after decimal for integers
056 CF 29         clear flag 29 to eliminate radix symbol display for integers, else it would get copied to alpha register
057 ARCL 14       copy sum d^n into alpha register
058 0             enter zero
059 LBL 08        Label for loop to sum d^n of sum d^n of input number
060 ATOX          move char# of leftmost digit of sum d^2 into X
061 X=0?          is it zero?
062 GTO 11        if zero, all digits shifted out, go to label to see if sum d^n of sum d^n equals original sum d^n
063 48            if not zero...
064 -             ...subtract 48 to convert char# to actual digit
065 RCL 00        recall number of digits
066 Y↑X           raise digit to nth power
067 +             add to running sum of d^n
068 GTO 08        loop back to sum next d^n
069 LBL 11        recall number of digit being checked
070 X<>Y          swap
071 RCL 14        recall originla sum d^n
072 X≠Y?          does sum d^n of sum d^n not equal original sum d^n?
073 GTO 10        if not equal, cannot be selfie, skip to check next number
074 10            enter 10
075 MOD           determine rightmost digit
076 X=0?          is FP zero, i.e., was rightmost digit zero?
077 GTO 10        if so, cannot be selfie since reverse will be fewer digits, skip all checks
078 RCL 14        if not, recall sum d^n
079 STO- 14       store in reg 14 to clear it. (steps 78 through 89 reverse the digits in the number to produce selfie)
080 LBL 16
081 FP            take fractional part
082 STO+ 14       add to register holding reversed sum d^n
083 LASTX         recall input number
084 IP            take integer part
085 0.1           enter 0.1
086 STO÷ 14       divide current reversed number by 0.1 to shift left one digit
087 ×             multiply input number by 0.1 to shift one digit right
088 X≠0?          is input number not zero, indicating more digits to shift?
089 GTO 16        if not zero, loop back to shift and add again
090 RCL 14        if zero, number has been reversed, recall reversed sum d^n
091 PRX           print number for which d...d=reverse of sum d^n
092 LBL 10        routine to check if original number padded with zeroes to 11 digits have been checked
093 11            enter 11
094 RCL 00        recall number of digits in original number (or padded with zeroes previously)
095 X=Y?          are they equal
096 GTO 02        if so, done checking original number and all padded with zero to 11 digits, go generate the next number
097 ISG 00        increment the number of digits
098 DEG           no op
099 GTO 07        go back to sum d^(n+1)
100 END
```

edit - shaved a couple of steps by using stack registers for a couple of loops instead of a storage register.

Attached File(s)

Sumd^n V3.zip (Size: 286 bytes / Downloads: 2)

Dave - My mind is going - I can feel it.

11-06-2019, 12:17 AM (This post was last modified: 11-06-2019 12:21 AM by Albert Chan.) **Post: #49**

Albert Chan
Senior Member

Posts: 1,226
Joined: Jul 2018

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

Jeff O. Wrote: (11-04-2019 08:18 PM)

My logic for this was as follows:

1. Let the digits^n of n-digit number A sum to n-digit number B
2. Let the digits^n of n-digit number B sum to n-digit number C
3. If B=C, then the digits^n of n-digit number C must also sum to n-digit number B
4. Therefore, number C (and B, of course) must contain the same digits as number A, i.e., is a permutation of the digits in A.

No. 4 is a conjecture, not an assertion ...

Hi, Jeff O.

I checked all (88) Armstrong numbers, and see how many A's can produce it.

If B is an Armstrong number (B=C), A is unique → A,B has same sets of digits

Confirmed (by brute force) that your conjecture is correct.

EMAIL PM FIND QUOTE REPORT

11-23-2019, 11:17 PM (This post was last modified: 12-01-2019 07:17 PM by Jeff O..) **Post: #50**

Jeff O.
Member

Posts: 185
Joined: Dec 2013

RE: [VA] Short & Sweet Math Challenges #23: "May the 4th Be With You !" ...

One final (hopefully) update: With some encouragement and advice from Valentin, I developed yet another version of my program. Valentin suggested that I eliminate excessive use of Y^X and instead pre-compute the digits to the powers (i.e., 0^N, 1^N, …9^N) and sum them according to the digits in the number. So instead of calculating 5^8, for example, merely sum the pre-computed value of 5^8 if there is a 5 in the 8-digit number under evaluation.

I was initially unsure if I could apply it due to the manner in which prior versions create and check numbers containing zeroes. See above for how I did that, suffice it to say that it would require creating and storing the values of 0 through 9 to the Nth power for each number, because my prior method changes the number of digits at each number checked. So pre-computing 0 through 9 to Nth power for each new number of digits at each number would not be expected to save much if any time.

But I liked the idea, so I decided to see if I could alter my method of generating input numbers to allow it to be used. The goal would of course be to generate all one-digit numbers, then all two-digit numbers (including those with one trailing zero), then all three-digit (including those with one or two trailing zeroes), etc., on up to all 11 digit numbers. This would enable me to pre-compute 0^N, 1^N, …9^N only when the number of digits is increased by one, i.e., only 11 times for the entire run of the program, which is I'm sure what Valentin intended. With a little effort, I was able to do just that. My prior method of generating numbers would go like this:

19999999999
22222222222
22222222223
22222222224
etc.

The new method inserts numbers with zeroes like this:

19999999999
20000000000
22000000000
22200000000
22220000000
22222000000
22222200000
22222220000
22222222000
22222222200
22222222220
22222222222
22222222223
22222222224
etc.

Incorporating the pre-computation method was a bit tricky, as it requires storing those values for later recall. The most natural registers to store them would be 0 through 9, so that the digit value could point to the register containing that digit to the Nth power. But all of my prior versions stored the digits of my input numbers in register 1 through 11, so I chose to store the values of 0^N, 1^N, …9^N in registers 20 through 29. This requires adding 20 to the recalled digit value before indirectly recalling the appropriate d^N value to create the sum. I was able to handle this, and the latest version is indeed the quickest yet. Using Free42 on my laptop PC, this latest version completes the task in approximately 35.6 seconds, compared to 51.7 seconds for the prior version, an approximate 30% decrease in execution time. On my DM42, the timings are 35 minutes and 57 seconds for the latest version vs. 51 minutes and 40 seconds for the prior version, also essentially a 30% reduction.

On a desktop PC that I also use at times, I get anomalous results. Initially, the timings were 12.9 seconds vs. 11.7 seconds. Then that PC received a Windows update, and now Free42 runs much more slowly. And, oddly enough, the new version is actually slower than the prior version, 49.9 seconds vs. 48.5 seconds. I am running the same version of Free42 on both PCs (the decimal version of 2.5.11), so I don't know why the timings are not consistent on one PC vs. the other, nor why the windows update would slow Free42 down by a factor of 4.

For timing results, I guess I put the most faith in the DM42 results, so I believe that the method is demonstrably faster. Below is the program listing, and a zipped raw file is attached. I'd be interested in seeing timings using Free42 in other platforms if anyone has the time or inclination.

```
000 { 212-Byte Prgm }
001 LBL "Σd↑n"
002 FIX 00          Fix 0 to eliminate zeroes after decimal for integers
003 CF 29           clear flag 29 to eliminate radix symbol display for integers, else it would get copied to alpha register
004 20              Steps 4 through 9 clear registers 1 through 20 (need to clear 20 for 0^N for all N's, not done in subroutine)
005 0
006 LBL 01
007 STO IND ST Y
008 DSE ST Y
009 GTO 01
010 1
011 STO 01          initialize register 1 to 1
012 XEQ 12          execute subroutine to calculate 1^N....9^N for N=1
013 GTO 98          go to label to calculate sum d^N for 1
014 LBL 02          Label 02, begin main routine
015 RCL 01          recall rightmost digit
016 X=0?            is it zero?
017 GTO 99          if so, prior number had trailing zero(es), go to routine to copy latest incremented number down to next lower register
018 1.011           Store 1.011...
019 STO 00          ...for ISG to check digits
020 LBL 03          Label 3, check digits to see if they equal 9
```

```
021 9                 Enter 9
022 RCL IND 00        recall digit
023 X≠Y?              is it not equal 9?
024 GTO 04            if not, go to routine to increment
025 ISG 00            if equal to 9, increment counter to…
026 GTO 03            ...loop back to check next digit
027 STOP              if register 11 equal 9, all unique combinations up to 99,999,999,999 have been checked
028 LBL 04            routine to increment registers
029 RCL 12            recall number of digits
030 RCL 00            recall loop counter, which has counted up to register number containing first non-9 digit
031 IP                take integer part
032 STO 19            store for use in storing new incremented values in lower registers
033 STO 19            store back in register zero without .011 rarget for use as DSE later
034 X>Y?              is register in which first non-9 found greater than current number of digits?
035 XEQ 12            if so, new number of digits established, go to subroutine to calculate 1^N, 2^N,…9^N
036 1                 enter 1
037 STO+ IND 00       add one to register in which first non-9 found
038 DSE 00            decrement register 0
039 DEG               no operation
040 0                 enter 0
041 LBL 05            label for loop to store zeroes in registers N-1 to 1
042 STO IND 00        store zero
043 DSE 00            decrement register zero
044 GTO 05            loop back to store next zero
045 GTO 98            go to label to calculate sum d^N
046 LBL 99            label to store incremented value in lower registers after storing zeroes
047 RCL IND 19        recall new incremented value
048 DSE 19            decrement loop counter
049 STO IND 19        store new incremented value in next lower register
050 LBL 98            label to calculate sum d^N
051 RCL 12            recall current number of digits
052 0                 enter zero
053 LBL 09            loop label for summing d^N
054 20                enter 20
055 RCL+ IND ST Z     add to value of digit in register N, N-1,…1
056 RCL IND ST X      recall value in register 20 + the value of the digit
057 RCL+ ST Z         add to running sum
058 R↑                roll up…
059 X<>Y              … and swap to get stack in proper order
060 DSE ST Y          decrement register containing N, N-1…1
061 GTO 09            loop back to recall and sum next d^N
062 STO 14            store sum d^N in register 14
063 CLA               clear the alpha register
064 ARCL 14           copy sum d^N into alpha register
065 ALENG             length of alpha register equals how many digits in number
066 RCL 12            recall current number of digits
067 X≠Y?              digits in original not equal digits in sum d^N?
068 GTO 02            if not equal, cannot be selfie, skip all checks
069 RCL 14            if equal, recall sum d^N
070 +/-               negate
071 LBL 08            Label for loop to sum d^N of sum d^N of input number
072 ATOX              move char# of leftmost digit of sum d^N into X
073 X=0?              is it zero?
074 GTO 11            if zero, all digits shifted out, go to label to see if sum d^N of sum d^N equals original sum d^N
075 28                if not zero…
076 -                 ...subtract 28 to convert char# to actual digit plus 20
077 RCL IND ST X      recall d^N of value of digit
078 RCL+ ST Z         add to negated sum d^N
079 GTO 08            loop back to recall next digit
080 LBL 11            Label to see if sum d^N of sum d^N equals original sum d^N
081 X≠Y?              zero will be in x, if sum d^N of sum d^N equals original sum d^N, zero will be in y
082 GTO 02            if not equal, cannot be selfie, go back to create and check next number
083 RCL 14            recall sum d^N
084 10                enter 10
085 MOD               determine rightmost digit
086 X=0?              is FP zero, i.e., was rightmost digit zero?
087 GTO 02            if so, cannot be selfie since reverse will be fewer digits, skip all checks
088 RCL 14            if not, recall sum d^N, steps 88 through 99 reverse the digits in the number to produce selfie
089 STO- 14           store in reg 14 to clear it.
090 LBL 16            label for loop to reverse digits
091 FP                take fractional part
092 STO+ 14           add to register holding reversed sum d^N
093 LASTX             recall input number
094 IP                take integer part
095 0.1               enter 0.1
096 STO÷ 14           divide current reversed number by 0.1 to shift left one digit
097 ×                 multiply input number by 0.1 to shift one digit right
098 X≠0?              is input number not zero, indicating more digits to shift?
099 GTO 16            if not zero, loop back to shift and add again
100 RCL 14            if zero, number has been reversed, recall reversed sum d^N
101 PRX               print number for which d…d=reverse of sum d^N
102 GTO 02
103 LBL 12            Subroutine to calculate 1^N, 2^N,...9^N and store in registers 21 through 29
104 STO 12            store new N
105 9                 enter 9 for DSE loop
106 LBL 13
107 RCL ST X          duplicate X register
108 RCL 12            recall number of digits
109 Y↑X               raise 0…9 to Nth power
110 20                enter 20
111 RCL+ ST Z         add to 0…9
112 X<>Y              swap
113 STO IND ST Y      store d^N in register 20+d
114 RCL ST Z          recall current digit
115 DSE ST X          decrement
116 GTO 13            loop back to calculate next
```

117 RTN subroutine return
118 END

edits - revised program now three steps shorter, added commented program listing, fixed a mistake, clarified some things.

Attached File(s)

Sumd^n V4.zip (Size: 326 bytes / Downloads: 0)

Dave - My mind is going - I can feel it.

User(s) browsing this thread: Valentin Albillo*, 1 Guest(s)