HP Forum Archive 08

[Return to Index | Top of Index]

Short & Sweet Math Challenges for HP fans #4 [LONG!]

*Message #1 Posted by **Ex-PPC member** on 16 June 2002, 9:35 p.m.*

Welcome to Short & Sweet Math Challenges for HP fans #4. Not so "short" this time, but rather interesting, I hope ...

Foreword

Last week we were searching for integer numbers having some interesting property. This time, we depart from the integer realm and dive deep into the real numbers realm. As you know, many amazing things lurk in the depths, be it marine depths, interstellar space depths, whatever. Same thing for mathematics, once you stop scratching the surface and really go down, you may find lots of unexpected wonders waiting to be found. This week I'm challenging you to leave the comfortable built-in precission of your HP calculator, be it 10, 12, or 20 digits, and really go for multiprecission calculations, I'm talking 50 decimal digits ...

Sounds frightening ? Too difficult ? It shouldn't be. Back in the good old times of PPC, people were overcoming the built-in accuracy limitations of their HP-65s, HP-67s, and HP-41s, and writing wonderfully clever programs to perform computations to high-precission. Perhaps some of you will remember programs to compute the constant Pi to 1000+ digits and more on a 41C. Outside of PPC, you could find programs to compute multi-precission square roots and other functions in the HP User's Library. If people could do that with their 65s, 67s, and 41Cs back in the early eighties, there's no reason why you shouldn't try now, specially using the newer HP models, which boast much faster CPUs and much larger RAM.

So, I'm proposing you a challenge, were I will ask you to perform several computations to a prescribed high precission, then explain the amazing results you'll get. That will be the second half of the challenge. The first is you'll need to develop a multiprecission program, library or package to be able to perform said compuations on your HP.

You may beg, steal or borrow such a program, library or package, or you can write one yourself. For those of you wanting to try, I'll give some guidelines at the bottom of this posting. Be assured that this challenge is perfectly feasible, and even easy ! But you'll need a high performance HP calc. I think the barest minimum would be an HP-41CV, and you'll be much advised to use instead an HP-42S, HP-71B, or HP-48/49 for this task. FOr ease of presentation and explanation, all guidelines, examples and snippets of code will be given for an HP-71B's native BASIC language. They can be easily adapted/converted to RPN or RPL.

The Challenge (#4):

1. If you compute the square root of 308600 and 308699 to 41 decimal places, you'll get:

```
sqrt(308600) = 555.51777649324598368631537686142297732063495
sqrt(308699) = 555.60687540742330162311819242204090515640356
```

   which look as expected, long sequences of random-looking decimal digits, with no periodicity or pattern at all.

   But now compute the square root of 308642 to that same precission, 41 decimal places. Explain the amazing result. Can you find other numbers which behave like this ?

2. Compute 987654321/123456789 to 25 decimal places. Explain the result. In particular, is it just sheer coincidence that 729 (=9^3) which appears isolated at the 8th decimal position ?

3. Compute E^(PI*Sqrt(163)) to 12 decimal places (i.e: to 12 digits after the decimal point), where E = 2.718..., PI = 3.14159..., and Sqrt is the square root function. Explain the amazing result. Can you find any other values (instead of 163) behaving like this ?

4. This doesn't really require high precission, but it's funny: compute a real root for the following equation between 6 an 8 [Note: Log(x) is base-10 logarithm, not Ln(x)]. Give the root accurate to 12 decimal places:

```
Log(x) - x - Sqrt(x-2) + Sqrt(19*x) = PI
```

Optional: Guidelines for writing a multiprecission program/library/package for your HP calculator:

There are many ways to do it, but you can follow these simple guidelines. They aren't optimal, but will get the job done easily:

1. Chose a fixed-format representation for your multiprecission numbers, for instance

```
(sign)50digits.50digits
```

   i.e: 50 digits for the integer part, 50 digits for the decimal part, plus sign.

2. Store those digits internally in blocks of, say, 5 digits, in an array or consecutive storage registers. Such a number will occupy 20 elements, the position of the decimal point is implicitly assumed to be between blocks 10th and 11th.

3. Write a procedure to allow the user to input such a number, converting the user's input to the internal representation. Along the same line, write a procedure to convert a normal, 10- or 12-digit number as used in your HP calculator, to the internal representation for multiprecission.

4. Write a procedure to output a number in internal representation to a user-readable format. Similarly, write a procedure to convert a multiprecission value to the 10- or 12-digit format used natively by your calculator.

5. Write a procedure to change the sign of a multiprecission value

6. Write a procedure to add two multiprecission values and give the result as another multiprecission value. You just need to:

   - stablish a loop which will sum all corresponding blocks (taking into account the numbers' signs). As you are using 5-digit blocks, they will never overflow, as adding up two 5-digits numbers will only result in a 6-digit number at most.

   - stablish a second loop which will normalize all blocks to be 5-digits again, propagating carries from one block to the previous if its size was 6 digits.

7. Write a procedure to subtract two multiprecission values. You just need to call the procedure for changing the sign of the second one, then call the add procedure.

8. Write a procedure to multiply two multiprecission values. You just need to perform the multiplication as you would by hand, but using blocks of 5 digits at a time, and taking proper care of the carries, and of the final result. No intermediate overflows are possible, because multiplying two 5-digit numbers together will give a 10-digit result at most. Assuming your elements can hold 10 to 12 digits, there's no problem.

9. Write a procedure to divide one multiprecission number (A) by another (B). Here you can reduce the problem to multiplication, by using Newton's method to compute the reciprocal of the second number (B): assuming x0 to be a good approximation to 1/B then a better approximation is x1, computed as

   ```
   x1 = x0*(2-B*x0)
   ```

   which means we can compute reciprocals without division, just using multiplication. For instance, if x0=0.3 is an approximation to the reciprocal of B=3.7, then

   ```
   x1 = 0.3*(2-3.7*0.3) = 0.267
   x2 = 0.267*(2-3.7*0.267) = 0.2702307
   x3 = 0.2702307*(2-3.7*0.2702307) = 0.27027026
   ```

   are better approximations. The iterations converge quadratically, i.e: you get double the number of exact digits after each. As your initial approximation, you can use 1/B, which you can get to 10 or 12 digits by using the 1/X function of your HP calc ! Write a procedure to convert that value to the internal representation of multiprecission numbers, then compute 1/B using the iterative method above. As your initial 1/B is already accurate to 10 or 12 digits, just 2 or 3 iterations will give you 1/B accurate to in excess of 50 digits !

   Once you've got 1/B to 50-digit precission, then A/B reduces to A*(1/B), which you can compute with just one extra multiprecission multiplication.

10. Write a procedure to compute the square root of a multiprecission value (A). Just use Newton's method once again:

    ```
    x1 = (x0 + A/x0) / 2
    ```

    As your initial approximation, just use the value of SQRT(A) given by the square root built-in function. This will give you a value accurate to 10 digits, and then just 2 or 3 iterations of the procedure above will give you a square root accurate to in excess of 50 digits.

11. Include in your program the constant PI accurate to 50 decimal places. You can use this value:

```
PI = 3.14159265358979323846264338327950288419716939937511
```

12. Finally, write a procedure to compute E^X where X is a multiprecission value. One possible way is to use its Taylor's Series Expansion, like this:

```
E^X = 1 + X + X^2/2! + X^3/3! + X^4/4! + ... + X^n/n! + ...
```

where you should stop when the next term to add would be smaller than 1E-50. Of course, n! is the factorial function = 1*2*3*4*...*n

There are a number of tricks you could try to accelerate convergence. For instance, you could use the identity:

```
E^X = (E^(X/2))^2
```

i.e:, you can first divide X by 2, then compute E^(X/2) and square the result. As the argument X is smaller, convergence of the Taylor Series will be faster. Similarly, you could divide X by 8, then square the result three times in a row, etc.

Re: Short & Sweet Math Challenges for HP fans #4 [LONG!]

*Message #2 Posted by **katie** on 17 June 2002, 12:29 a.m.,*
in response to message #1 by Ex-PPC member

FYI,

I posted a program that I wrote (see the articles forum) some time ago to compute Pi to 99 digits on a 32SII. It uses many of the techniques that you mention for long precision arithmetic. Since it's only 99 digits the convergence time is pretty short (11 minutes), but the algorithm isn't particularly efficient just very compact (it needs to be given the severe memory limitation of the 32SII.)

-Katie

Begin of solution question #1

*Message #3 Posted by **thibaut.be** on 18 June 2002, 10:38 a.m.,*
in response to message #1 by Ex-PPC member

The inverse of 9, 90, 909009 and such numbers give interesting results : .11111, .011111, .000001100, ...

So a value of 555.5555578 is *almost* like 555.55555555, hence 5/9 * 1000 or 5000/9

If you square this value, you get 25 000 000 / 81 which is according to my CV 308 641.9753, or a very close value to 308 642.

Re: Begin of solution question #1

*Message #4 Posted by **Fernando del Rey** on 26 June 2002, 5:59 p.m.,*
in response to message #3 by thibaut.be

The square root of 308642 is:

555.555577777777333333351111110222222227199999 ...

But I can't figure out why such a curious pattern of digits.

Any ideas from the clever members of this forum?

Re: Short & Sweet Math Challenges for HP fans #4 [LONG!]

*Message #5 Posted by **Fernando del Rey** on 26 June 2002, 2:56 a.m.,*
in response to message #1 by Ex-PPC member

I obtained the solution to Question #4 on my 42S and the result is:

x = 7.00000001061

Real close to 7!

The solver converged very quickly given the interval 6 to 8 to start the search.

Why this interesting result? I have no clue. I guess it's just a numerical curiosity, but perhaps Mr. Ex-PPC Member will give us an explanation.

Still no answers have been given to questions 2 and 3. Anyone still working on it?

Re: S&SMC #4: Final Remarks [LONG!]

*Message #6 Posted by **Ex-PPC member** on 29 June 2002, 10:26 p.m.,*
in response to message #1 by Ex-PPC member

Foreword

Well, it seems S&SMC #4 didn't catch the fancy of HP lovers this time, perhaps the subject matter wasn't interesting enough, or it was deemed too difficult or time-consuming for a casual approach. Certainly, most of us are always too busy to be able to commit scarce free time to everything we would like to, so I'm particularly grateful to **Mr. Thibaut** and **Mr. Del Ray** for their kind contributions.

That said, here are some answers to the questions raised in S&SMC #4:

Question 1.

If you compute the square root of 308600 and 308699 to 41 decimal places, you'll get:

```
sqrt(308600) = 555.51777649324598368631537686142297732063495
sqrt(308699) = 555.60687540742330162311819242204090515640356
```

which look as expected, long sequences of random-looking decimal digits, with no periodicity or pattern at all.

But now compute the square root of 308642 to that same precission, 41 decimal places. Explain the amazing result. Can you find other numbers which behave like this ?

- **The raw facts:**

  Using your HP-41C and your multiprecision program, you first determine that, to 41 decimal places :

```
sqrt(308642) = 555.55557777777733333335111111022222227199999
```

  which indeed shows some *remarkable pattern*, unlike the other square roots above.

- **The explanation:**

  From the square root decimal expansion, it's clear that sqrt(308642) is very close to 555.5555... = 5000/9. So, with **d** being some small value, we have:

```
308642 = (5000/9)^2 + d
```

  where d = 308642-(5000/9)^2 = 2/81, small as expected. Thus, we have:

```
308642 = (5000/9)^2 + 2/81
```

  and factoring (5000/9)^2 out from the right side, we have:

```
308642 = (5000/9)^2 * ( 1 + (2/81)/(5000/9)^2) = (5000/9)^2 * (1 + 1/12500000)
```

  now, taking square roots in both sides:

```
sqrt(308642) = sqrt[(5000/9)^2 * (1 + 1/12500000)] = (5000/9)*sqrt(1 + 1/12500000)
```

Finally, our CAS-capable HP calc will give us the Taylor Series Expansion for sqrt(1+x):

```
sqrt(1 + x) = 1 + x/2 - x^2/8 + x^3/16 + ...
```

which for x = 1/12500000 yields:

```
sqrt(1 + 1/12500000) = 1 + 1/25000000 - 1/1250000000000000 + ...
                     = 1.000000039999999200000031999998400000008959999...
```

and thus we have:

```
sqrt(308642) = (5000/9) * sqrt(1 + 1/12500000)
             = 555.555555555555... *  1.000000039999999200000031999998400000008959999...
```

and as we are multiplying together two heavily patterned numbers, it is only to be expected that their product will feature some pattern too, at least initially:

```
             =  555.555577777777333333351111110222222227199999...
```

- **Finding more:**

This will be left as an (easy) exercice. Just a hint: inspired by this particular example, write a short program for your HP to try and find periodic fractions n/9 such that (n/9)^2 comes as close to being an integer as possible. The square root of that integer will probably be an interestingly patterned number.

Question 2.

Compute 987654321/123456789 to 25 decimal places. Explain the result. In particular, is it just sheer coincidence that 729 (=9^3) which appears isolated at the 8th decimal position ?

- **The raw facts:**

Using your HP42S and your multiprecision program, you find that:

```
987654321/123456789 = 8.000000072900000663390006036849O...
```

which is *very nearly* 8, except for the isolated 729, 66339, 6036849 ... Now, 729 = 9^3. This isn't a coincidence, as we will see.

- **Explanation:**

Using our prime factor finder program or built-in function, we find that:

```
   729 = 9^3         = 9^3 * 91^0
 66339 = 9^3 * 91    = 9^3 * 91^1
6036849 = 9^3 * 91^2  = 9^3 * 91^2
```

so we can *conjecture* that:

```
987654321/123456789 = 8 + 9^3*1E-10*SUM[N=0,N=INFINITE,(91*1E-10)^N]
```

where the sum goes from N=0 to N=infinite. This can be proved very easily using the well-known summation formula for geometric progressions, as this happens to be one.

Question 3.

*Compute E^(PI*Sqrt(163)) to 12 decimal places (i.e: to 12 digits after the decimal point), where E = 2.718..., PI = 3.14159..., and Sqrt is the square root function. Explain the amazing result. Can you find any other values (instead of 163) behaving like this ?*

- **The raw facts:**

  Using your HP-71B and your multiprecision program, you first find that, to 12 decimal places:

```
E^(PI*Sqrt(163)) = 262537412640768743.999999999999...
```

  which comes *extremely close* to being an integer.

- **The explanation:**

  This time the explanation is more involved, has to do with the theory of modular functions, and can't be given here in full without making this already long post much longer. Interested people can have a look at this very interesting URL:

  http://membres.lycos.fr/bgourevitch/mathematiciens/approx/approx.html

  It's in French, but never mind, the numbers and formulas speak for themselves.

- **Finding more:**

  It's quite easy to write a program on your HP calc to test if Z=E^(PI*Sqrt(N)) is nearly an integer for various values of N. If you do write and run it, it should find these values:

```
N =  25  -> Z =           6635623.9993+
  =  37  -> Z =         199148647.99997+
  =  43  -> Z =         884736743.9997+
```

```
=  58  -> Z =        24591257751.9999998+
=  67  -> Z =       147197952743.999998+
=  74  -> Z =       545518122089.9991+
= 148  -> Z =  39660184000219160.0009+
= 163  -> Z = 262537412640768743.9999999999992+
```

Other such values include also 232, 268, 522, 652, and 719, among infinitely many others, but 163 is still the most amazing of them all.

As for question 4, the solution was already given by Mr. Del Rey. It was a simple curiosity, a coincidence, no explanation necessary or possible, and it didn't require multiprecision at all, anyone could have solved it using any HP model with a solver or a very simple root finding program. I was somewhat surprised that nobody posted the solution much sooner.

Conclusion

Well, it's a pity that no one produced any code to do even basic multiprecision arithmetic in their favorite HP calc, despite the simple guidelines I offered. Anyway, just to show that it isn't as difficult or time-consuming as it seems, I'm including here a short snippet of sample code I wrote for this section. The code is intended to be used as a basis for your own ideas and it isn't optimized at all.

Sample Code

- This sample code is written in the native HP-71B BASIC language and implements a user-defined string function, FNM$, which accepts as input two strings representing multiprecision integer values and returns a string result representing the integer value of their product.

- **Listing:**

```
 10 DEF FNM$[200](A$,B$) @ OPTION BASE 1 @ DIM M,N,L,P,I,J,N1,N2,N3 @ N1=LEN(A$)
 20 N2=LEN(B$) @ N3=N1+N2 @ DIM A(N1),B(N2),C(N3),C$[N3]
 30 FOR I=1 TO N1 @ A(N1+1-I)=VAL(A$[I,I]) @ NEXT I
 40 FOR I=1 TO N2 @ B(N2+1-I)=VAL(B$[I,I]) @ NEXT I
 50 FOR I=1 TO N2 @ M=B(I) @ IF NOT RES THEN 100
 60 L=I @ FOR J=1 TO N1 @ N=A(J) @ IF NOT RES THEN 90
 70 P=M*N @ C(L)=C(L)+RES @ P=RES
 80 IF RES>9 THEN C(L)=RMD(P,10) @ C(L+1)=C(L+1)+P DIV 10
 90 L=L+1 @ NEXT J
100 NEXT I
110 FOR I=N3 TO 1 STEP -1 @ IF C(I) THEN 130
120 NEXT I
130 C$="" @ FOR I=I TO 1 STEP -1 @ C$=C$&STR$(C(I)) @ NEXT I @ IF C$="" THEN C$="0"
140 FNM$=C$ @ END DEF
```

- **Examples:**

```
Given:   A$="31415926535897932384"
         B$="2236067977499"


We find:


   FNM$(A$,B$) = "702481473103824548857867354276l6"
   FNM$(A$,A$) = "9869604401089358618440891014462359234 56"
   FNM$(B$,B$) = "499999999999646837029 5001"


and to top it all:


   FNM$(FNM$(A$,B$),FNM$(A$,B$)) =


   = "4934802200541193730412497902865991365066527558841386130375443456"
```

- **Some notes:**

  1. We are representing multiprecision numbers as strings (instead of arrays) and multiprecision operations as user-defined functions. This makes it much more easy for the user to use them interactively, from the command line, no need to write a small BASIC program to execute them.

  2. Line 10 defines FNM$ as a user-defined string function that can return a string up to 200 characters in length. So the maximum result is limited to 200 digits.

  3. The string parameters are meant to represent multiprecision integer values only, not floating point ones. The result will be an integer as well.

  4. The multiprecision integers represented as strings are stored internally in full-precision numeric arrays, one single digit per array element. Then, pairs of single digits are multiplied at a time in a doubly nested loop. This is *extremely* inefficient: the HP-71B can hold integers up to 12 digits long on an array element, so your improved version should store and multiply pairs of 5 or 6 digits at a time. This would increase speed by a factor of 20 or more

  5. Lines 30 and 40 store each digit in their own numeric array element

  6. The commands "IF NOT RES" in lines 50 and 60 skip digits equal to 0. By the way, RES returns the "LASTX" value, i.e.: the last result in a calculation. It's faster than using a variable, and saves memory as well.

  7. Lines 70 and 80 multiply each pair of single digits and take care of the carry if the result exceeds one digit.

8. Lines 110 and 120 avoid outputting leading zeroes

9. Line 130 reassembles the result back to a string

Re: S&SMC #4 & #3 Final Remarks...

*Message #7 Posted by **Andrés C. Rodríguez (Argentina)** on 30 June 2002, 9:06 a.m.,*
in response to message #6 by Ex-PPC member

Thank you again for an interesting challenge and for the enlightning final remarks. While this time I have had not enough time or interest to participate trying to solve the questions, I think the two-week period is more appropriate for the S&SMCs. Let us hope that the reduced participation in the last episode will not induce Mr. Ex-PPC Member to quit challenging us. It remains to be proved that the participation is larger in odd-numbered S&SMCs than in even-numbered cases. :-)

BTM, Mr. Ex-PPC, while I am fully aware I was late with my final response to S&SMC #3, and also do know that the examples you give in your final remarks were not optimized (so no comparisons are to be made), still I would had liked some comments about my answer (of course, you are not obliged at all to provide such); since I think the program I submitted have a couple of interesting features to discuss. Of course, this could be done by email, but not having your name or email address only left this option for me to voice this.

Of course, I mean no critique or complaint, and please disregard any idiomatic expression which may suggest that.

[Return to Index | Top of Index]

Go back to the main exhibit hall