HP Forum Archive 16

Short & Sweet Math Challenges #16: Thinking square !

*Message #1 Posted by **Valentin Albillo** on 11 May 2006, 4:45 a.m.*

Hi all,

A month-and-a-half has elapsed since I posted my S&SMC#15 (April 1st Special) and it's about time for me to offer you yet-another-challenge for your potential enjoyment and dusting off your alleged HP-programming skills.

As #16 is both the square of 4 and the square of the square of 2, I feel it's only appropriate that this time we think squarely so here you are, a couple of square challenges for you to try and solve with your HP.

As always, I'll post my original solutions and comments within a few days, so if you want to try your hand at them you'll have the whole weekend at the very least to try and solve them for good.

In all cases, *you **must** write a program* for your chosen HP handheld preferably, but other brands are also acceptable as long as they're handhelds (but no PDAs). Emulators/simulators for said models are also welcome, of course. Any programming language or paradigm is acceptable as long as it actually runs in your chosen handheld. Now for

---

The Challenge

:

Scroll 1: Being yourself ...

Write a program that finds out all 10-digit integers (base 10, of course) that upon being squared, the result's rightmost digits *exactly reproduce the original number*. For example, 9573921746 would be a solution if we had

9573921746² = 91659977599573921746

which, alas, it's not the case. Take note that 10-digit solutions beginning by one or more 0's obviously aren't 10-digit numbers, of course, and thus aren't valid solutions.

I'll give both a 4-line, 101-byte BASIC solution for the HP-71B, which takes negligible time to find all solutions, as well as a 44-step RPN solution for the HP-15C, which takes slightly over a minute to do likewise.

Scroll 2: ... Big Time !

Now that you've conquered the previous challenge, you're up to the next summit, where we'll *square* the conditions, i.e.: write a program to find and orderly output all solutions with the same requirements as above, but this time *for 10-squared, i.e. 100-digit integers !*

Don't worry, it isn't nearly as insurmountable as it seems, given the correct approach and algorithm. To prove it, I'll provide a short (600+ bytes) solution for the HP-71B in a mere 18 lines of code, which finds out and outputs all 100-digit solutions fairly fast.

Scroll 3: All included

Write a program to find and output all pairs of 5-digit integers such that taken *together* they do include *all digits from 0 to 9* and each of their respective 10-digit squares also include all digits from 0 to 9 as well.

For example, 54903 and 12786 would be a solution pair if their squares were, respectively, 2570184639 (which includes all digits from 0 to 9) and 1540783926 (ditto) but, unfortunately, they aren't.

I'll give a 13-line, 393-byte solution for the HP-71B which finds all solutions in less than 9 seconds in Emu71 @ 2.4 Ghz, and less than 25 min. in a real HP-71B.

Caveats:

As always, the following caveats *strictly* apply, namely:

Please refrain from posting just the solutions, *actual code* written by yourself ***is*** *mandatory*. Googling solutions and posting them out is pretty lame and only serves to spoil the challenge for others and to make blatantly public your lack of programming skills and/or utter disrespect for fair rules and this forum's visitors.

That said, let's see your answers before I post my solutions next week.

Best regards from V.

Re: Short & Sweet Math Challenges #16: Thinking square !

Message #2 Posted by ***Thibaut.be*** *on 11 May 2006, 10:22 a.m.,*
in response to message #1 by Valentin Albillo

I've finally found out the equation for one of the possibilities, but I get overflows while calculating the answer.

I'm currently working on calculating the other possibility.

So looking forward to reading your answer.

Re: Short & Sweet Math Challenges #16: Thinking square !

Message #3 Posted by ***Thomas Okken*** *on 11 May 2006, 10:47 a.m.,*
in response to message #2 by Thibaut.be

> Quote:
>
> I've finally found out the equation for one of the possibilities, but I get overflows while calculating the answer.

SPOILER ALERT

The first problem isn't too hard once you realize that *n^2 ends with the same digits as n* means *n*(n-1) is divisible by 10^10.*
If n is divisble by 2, n-1 is not, and vice versa; also, if n is divisible by 5, n-1 is not, and vice versa. This means that either n or n-1 is divisble by 2^10, and either n or n-1 is divisble by 5^10.
If n is divisble by 2^10, n-1 must be divisble by 5^10, and vice versa. Neither n nor n-1 can be divisble by 2^10 and 5^10, because that would mean n >= 10^10, but the requirement is that n is a 10-digit number, hence n < 10^10.

So, all you have to do is check all multiples of 5^10, from 103*5^10 to 1023*5^10, and check if they are equal to 1 or -1 (mod 1024). If a multiple of 5^10 is equal to 1 (mod 1024), then n is that number; if a multiple of 5^10 is equal to -1 (mod 1024), then n is that number plus 1. Since 5^10 mod 2^10 is prime, no more than one solution of each type can exist.

The following (very quick-and-dirty!) program finds the solutions on the HP-42S:

```
00 { 64-Byte Prgm }
01>LBL "SS16"
02 5
03 10
04 Y^X
05 STO 00
06 1023
07 в
08 STO 01
09>LBL 00
10 RCL 01
11 1024
12 MOD
13 1
14 X!=Y?
15 GTO 01
16 RCL 01
17 PRX
18 Rv
19>LBL 01
20 Rv
21 1023
22 X!=Y?
23 GTO 02
24 RCL 01
25 1
26 +
27 PRX
28>LBL 02
29 RCL 00
30 STO- 01
31 RCL 01
32 X!=0?
33 GTO 00
34 BEEP
35 .END.
```

Result:

```
    8,212,890,625    ***
    1,787,109,376    ***
```

Unfortunately, problem 2 is not so easy. :-) Brute force checking is not practical because the above reasoning does not eliminate enough possibilities. Back to the drawing board to find ways to reduce the amount of work even further...

- Thomas

Re: Short & Sweet Math Challenges #16: Thinking square !

Message #4 Posted by ***Valentin Albillo*** *on 12 May 2006, 7:51 a.m.,*
in response to message #3 by Thomas Okken

Hi, Thomas:

Very clever approach indeed, though I didn't expect anything less from you. Besides, your approach is a *book example* of the hint I usually included in the 'Caveats' section in my past challenges, that is, to achieve a *proper balance* between the two extremes, namely

- (a) the approach of letting the program do all the work by *brute-force* alone, which is likely to result in a *much-too-simple, uninteresting program* which takes ages to deliver, or else,
- (b) having the user think *too much* about the problem, doing all the work, to the point of solving or nearly solving it completely, which results in a *much-too-simple, uninteresting program* which does nothing but regurgitate the user's solution.

The ideal is, of course, what you just did: to have the user think *just a little* about the problem, in order to provide some clever insights, then create an *interesting, smart program* which uses said insights to deliver orders of magnitude faster. Well done.

Now, way to go. And, by the way, though you describe your program as intended for the HP-42S, it actually runs perfectly in the HP-41 family as well, unchanged. I also hope you find the time to solve the remaining Scrolls, and it would be great if you would provide timings for your programs, both running on a physical HP-42S as well as under Free42 in various platforms, if at all possible. It will be interesting and potentially useful for future readers.

Best regards from V.

Re: Short & Sweet Math Challenges #16: Thinking square !

Message #5 Posted by ***Thomas Okken*** *on 13 May 2006, 6:16 p.m.,*
in response to message #4 by Valentin Albillo

> Quote:
>
> I also hope you find the time to solve the remaining Scrolls, and it would be great if you would provide timings for your programs, both running on a physical HP-42S as well as under Free42 in various platforms, if at all possible.

On my HP-42S, my program for problem 1 takes 2 minutes 52 seconds. On Free42 on my PC, it's instantaneous -- probably a matter of milliseconds.

I'm sure it's possible to do much better -- you say you have a program that does the job in just over a minute on an HP-15C... That's almost three times as fast as my program, on a machine that is MUCH slower than an HP-42S!

But, I'll leave my program as it is. Never optimize something that's fast enough already. ;-)

- Thomas

Re: Short & Sweet Math Challenges #16: Thinking square !

*Message #6 Posted by **Bram** on 12 May 2006, 5:15 a.m.,*
in response to message #1 by Valentin Albillo

Hi Valentin,

Thanks for these again interesting challenges.

I happen to notice a strategy that might, I repeat: might lead to an answer.

I start with 10 single digits and throw away those that do not fulfill the criteria. That leaves me the digits 0, 1, 5 and 6.
For each of them I add 10 tens and repeat the actions. One of them that will be left is 76 for 76*76=5776
I add 10 hundreds which will give me (amongst others?) 376 for 376*376=141376
After this: 9376*9376=87909376
and 09376*09376=87909376
and eventually I will get to the already given answer of 1787109376 (I think)

To test this approach I still have to write the requested program for my 32sii, but I fear a lack of time this weekend.
I'll have to find an efficient way to square large numbers. I think it can be done, but it will take a lot of calculating power, I guess. I keep thinking about it. Maybe I'll manage to have a program in time.

I find this number behavior intriguing. I cannot prove if you get all the answers this way, but I'm sure you can comment on it in your solutions.

Have a nice weekend.

Re: Short & Sweet Math Challenges #16: Thinking square !

*Message #7 Posted by **Thomas Okken** on 12 May 2006, 10:20 a.m.,*
in response to message #6 by Bram

Bram's approach will indeed find all solutions. The key observation is that when you add a new digit to the left of an existing n-digit number, this does not affect the final n digits of the square:

```
(x + d * 10^n)^2 - x^2
= x^2 + 2 * x * d * 10^n + 10^(2 * n) - x^2
= (2 * x * d + 10^n) * 10^n
= 0 (mod 10^n)
```

So, you start with the 1-digit numbers 0, 1, 5, and 6, and then, for each of those, try adding all 10 possible digits to the left, and keep the 2-digit numbers that satisfy the condition of the challenge. Repeat for the 3rd digit, and so on. Since for every n-digit solution, all "tails" (numbers obtained by removing leading digits from the solution) also satisfy the condition, you are guaranteed to find **all** solutions this way.

It's a bit tricky to implement in RPN, but this approach does have the excellent property that its time usage is proportional to the number of digits squared, which is a lot better than the brute force approach (or the modified brute force approach I used to solve problem 1), which is exponential in the number of digits.

Now, to see if I can get this to work on my HP-42S. If it turns out to be too hard, maybe I'll risk my karma and do it on my HP-48G instead. :-)
Nice challenge -- thanks V!

- Thomas

Re: Short & Sweet Math Challenges #16: Thinking square !

Message #8 Posted by ***Valentin Albillo*** *on 12 May 2006, 10:34 a.m.,*
in response to message #7 by Thomas Okken

Thomas posted:

> *"Nice challenge -- thanks V!"*

> You're welcome. :-)
>
> If you do like this kind of challenge, you'll certainly *love* my next one, come due next June 1st but already in the works. Meanwhile, waiting for your solutions, most specially the 100-digit behemoths.

Best regards from V.

Re: Short & Sweet Math Challenges #16: Thinking square !

Message #9 Posted by ***Thomas Okken*** *on 12 May 2006, 11:10 a.m.,*
in response to message #7 by Thomas Okken

SPOILER ALERT

Bram's approach is even better than I thought, once you add another observation to the mix: given an n-digit number that satisfies the condition x^2=x (mod 10^n), there is *no more than one* "extended" solution x+d*10^n:
Suppose x^2=x (mod 10^n). We're looking for a solution that differs from x only in its first digit, that is, x+p*10^(n-1). So,

```
(x+p*10^(n-1))^2 = (x+p*10^(n-1)) (mod 10^n)
<=> x^2 + 2*x*p*10^(n-1) + p^2*10^(2*n-2) = x + p*10^(n-1) (mod 10^n)
<=> (2*x-1)*p*10^(n-1) + p^2*10^(2*n-2) = 0 (mod 10^n)   (since x^2=x (mod 10^n))
<=> (2*x-1)*p*10^(n-1) = 0 (mod 10^n)
<=> (2*x-1)*p = 0 (mod 10)
```

Since x must end in 0, 1, 5, or 6, (2*x-1) ends in 1 or 9, so (2*x-1) is neither a multiple of 2 nor a multiple of 5. Since p is a 1-digit number, it can only be a multiple of 10 if it is 0 -- which means it is impossible to modify a solution by changing its leading digit. (Unless n = 1, of course; in that case, the penultimate step in the above proof is not allowed.)

So, there can be no more than 2 solutions to the general problem. This simplifies the code to look for those solutions considerably, since there is no need to track an arbitrarily large set of solutions.
I guess that leads to another question: do solutions exist for arbitrarily high values of n? If so, prove it; if not, which n-digit solutions cannot be extended to n+1 digits?

Also note that 8,212,890,625 + 1,787,109,376 = 10,000,000,001 Coincidence...?

\- Thomas

Edited: 12 May 2006, 11:26 a.m. after one or more responses were posted

Re: Short & Sweet Math Challenges #16: Thinking square !

Message #10 Posted by ***Thibaut.be*** *on 12 May 2006, 11:17 a.m.,*
in response to message #9 by Thomas Okken

I found the general equation 5^2^n mod 10^n, but could not simplify it, hence making it unuseable on a calculator. I've also found the formula for the "6" possibility.

Re: Short & Sweet Math Challenges #16: Thinking square !

*Message #11 Posted by **Bram** on 13 May 2006, 6:14 a.m.,*
in response to message #9 by Thomas Okken

If you line up the two answers then you'll see that every pair of digits on the same place add up to 9 (apart from the right most).

Two questions I ask myself (in words, not mathematically ;-):
1. the two numbers can be considered been built up to ten digits. Can they be extended to an inifinitive amount, still matching the criteria?
2. If so, will they to inifinity fulfill the "9 test"?

(looks like science: every answer yields more new questions ;-)

Edited: 13 May 2006, 6:21 a.m.

Re: Short & Sweet Math Challenges #16: Thinking square !

*Message #12 Posted by **Thomas Okken** on 13 May 2006, 1:22 p.m.,*
in response to message #11 by Bram

1. the two numbers can be considered been built up to ten digits. Can they be extended to an inifinitive amount, still matching the criteria?

Yes; Thibaut.be actually gave the formula in his previous posting. It needs to be modified a bit to make it usable on a calculator, but it is definitely correct.

2. If so, will they to inifinity fulfill the "9 test"?

Yes; so that simplifies the programming task even further...

- Thomas

Re: Short & Sweet Math Challenges #16: Thinking square !

*Message #13 Posted by **Thomas Okken** on 13 May 2006, 5:55 p.m.,*
in response to message #1 by Valentin Albillo

SPOILER ALERT

Here's my solution to problem 2:

```
00 { 283-Byte Prgm }
01>LBL "16-2"
02 STO "N"
03 STO "M"
04 3
05 +
06 4
07 B
08 IP
09 STO "S"
10 1
11 NEWMAT
12 STO "REGS"
13 STO "SQ"
14 INDEX "SQ"
15 5
16 STO 00
17>LBL 00
18 1
19 ENTER
20 STOIJ
21 0
22>LBL 01
23 STOEL
24 I+
25 FC? 77
26 GTO 01
27 RCL "S"
28 1
29 -
30 1E3
31 B
32 STO "I"
33>LBL 02
34 0
35 STO "P"
36 RCL "I"
37 IP
38 1E3
39 B
40 STO "J"
41>LBL 03
42 RCL "I"
43 IP
44 RCL "J"
45 IP
46 -
```

```
RCL IND ST X
RCLв IND "J"
STO+ "P"
ISG "J"
GTO 03
RCL "I"
1
+
1
STOIJ
>LBL 04
RCL "P"
1E4
MOD
RCLEL
+
RCL ST X
1E4
MOD
STOEL
CLX
LASTX
B
IP
RCL "P"
1E4
B
IP
+
STO "P"
X=0?
GTO 05
I+
FC? 77
GTO 04
>LBL 05
ISG "I"
GTO 02
RCL "SQ"
STO "REGS"
DSE "M"
GTO 00
RCL "S"
1
-
STO "I"
RCL IND ST X
```

```
94 RCL "N"
95 1
96 -
97 4
98 MOD
99 1
100 +
101 10^X
102 MOD
103 STO IND ST Y
104 CLA
105 SF 21
106 TONE 9
107 GTO 08
108>LBL 06
109 3
110 RCL IND "I"
111 X>0?
112 LOG
113 IP
114 -
115 X=0?
116 GTO 08
117>LBL 07
118 |-"0"
119 DSE ST X
120 GTO 07
121>LBL 08
122 RCL IND "I"
123 AIP
124 ALENG
125 17
126 X<=Y?
127 AVIEW
128 X<=Y?
129 CLA
130 RCL "I"
131 1
132 -
133 STO "I"
134 X>=0?
135 GTO 06
136 ALENG
137 X>0?
138 AVIEW
139 BEEP
140 .END.
```

To use, enter the desired number of digits and say XEQ "16-2".

The program is based on the formula 5^2^n mod 10^n, which Thibaut.be discovered. The difference is that my program performs the repeated squares in n digits, instead of leaving the mod 10^n operation for last; this way, the size of the numbers to be manipulated stays small enough to be manageable. Discarding the leading digits and only keeping the n least significant digits, at each squaring step, is OK, since the higher digits never affect the less significant ones anyway.

To see that 5^2^n mod 10^n satisfies the condition of the challenge:

```
(5^2^n)^2 = 5^2^n (mod 10^n)
<=> 5^2^(n+1) = 5^2^n (mod 10^n)
<=> 5^2^n * (5^2^n - 1) = 0 (mod 10^n)
```

The factor (5^2^n - 1) can be rewritten as

```
(5^2^(n-1) + 1) * (5^2^(n-1) - 1)
= (5^2^(n-1) + 1) * (5^2^(n-2) + 1) * (5^2^(n-2) - 1)
= (5^2^(n-1) + 1) * (5^2^(n-2) + 1) * (5^2^(n-3) + 1) * (5^2^(n-3) - 1)
etc...
```

The full expansion has n+1 factors, all of which are even, so that 5^2^n - 1 is divisible by 2^n. Since 5^2^n is obviously divisble by 5^n, the product 5^2^n * (5^2^n - 1) is divisible by 10^n, so 5^2^n equals (5^2^n)^2 (mod 10^n), Q.E.D.

In another posting, I proved that every n-digit solution has a unique extension to n+1 digits; since the solution must end in 0, 1, 5, or 6, there can be no more than 4 solutions. The solutions ending in 0 and 1 are trivial: they can only be extended by adding leading zeroes. The solutions ending in 5 are generated by Thibaut's formula.
So, all that remains are the solutions ending in 6. I already noticed that the sum of the two 10-digit solutions is 10,000,000,001; could this be true in general? I.e., assuming x is a valid n-digit solution ending in 5,

```
(10^n+1-x)^2 = 10^n+1-x (mod 10^n)
<=> 10^(2*n)+2*10^n+1-2*x*(10^n+1)+x^2 = 10^n+1-x (mod 10^n)
<=> 1-2*x*(10^n+1)+x^2 = 1-x (mod 10^n)
<=> x*(-2*10^n-1)+x^2 = 0 (mod 10^n)
<=> x^2-x = 0 (mod 10^n)
```

So 10^n+1-x is also a solution, ending in 6; this is in fact the only other solution.

I'm still waiting for my HP-42S to finish running the above program; I expect it to take between 2 1/2 and 3 hours. I also ran it on Free42 Decimal, which takes about 2 seconds on my laptop (1.4 GHz Celeron). It returns this result:

```
39530073191081698029
38509890062166509580
```

```
86381100055742342323
08961090041066199773
92256259918212890625
```

I verified that this is correct using the **bc** program under Linux.

- Thomas

UPDATE: The HP-42S took 3 hours 31 minutes. While waiting for it to finish, I realized that there are two pretty simple ways to speed the program up: first, use groups of 5 digits instead of 4, and second, take advantage of the fact that x*x can be computed in slightly more than half the time of the more general x*y. So, basically the same algorithm *could* run in about 75 minutes on a standard HP-42S...
Oh, well, some other time. I haven't even looked at problem 3 yet! :-)

Edited: 13 May 2006, 8:30 p.m.

Re: Short & Sweet Math Challenges #16: Thinking square !

Message #14 Posted by ***Gerson W. Barbosa*** *on 14 May 2006, 5:55 p.m.,*
in response to message #1 by Valentin Albillo

Hi Valentin,

I have found all four pairs of 5-digit numbers that are solution to #3 (11 if the pairs with one or two squares beginning with zero are not discarded). However, since the HP-200LX is technically a PDA and QBASIC is a "computer language" I will not post my solution here. Anyway, the program takes about 45 minutes to run on the 200LX (an estimated 14 hours on a real HP-71B; 9.5 seconds on a 500 MHz Pentium III), which is an eternity compared to your optimized solution.

Best regards,

Gerson

Re: Short & Sweet Math Challenges #16: Thinking square !

Message #15 Posted by ***Valentin Albillo*** *on 14 May 2006, 6:43 p.m.,*
in response to message #14 by Gerson W. Barbosa

Hi, Gerson:

Thanks for your interest, seems you got it right as there are indeed just four solutions which meet the requirements.

As for my "optimized" solutions, matter of fact they are but just the barely reasonably efficient ones I could come up with without spending too much time concocting them, which I don't have. For this particular #3, I've recently found a way to make it even faster than I stated in my initial post, but nevertheless I'll simply post my original solution and will give a hint on how it can be optimized even further.

By the way, I firmly encourage you to try and convert your QBASIC solution to run in any of the acceptable models, say the HP-71B or some RPL machine. Running time is secondary as long as the solutions are correct, as I'm sure yours are, and forum visitors, myself included, will learn and appreciate your very own techniques to deal with these highly unusual problems.

Best regards from V.

Re: Short & Sweet Math Challenges #16: Thinking square !

Message #16 Posted by ***Gerson W. Barbosa*** *on 14 May 2006, 7:56 p.m.,*
in response to message #15 by Valentin Albillo

Hi Valentin,

Thanks for your encouraging words. I began thinking about this new challenge of yours only yesterday night. I had taken a look at it on Friday but this seemed too difficult for a curious one like me to give it a try. Anyway, even knowing I would not come up with a brilliant solution like yours, Bram's, Thomas Okken's et. al. I decided to try number three as an exercise.

My approach is very straightforward: just ten nested loops. The first five loops yield the 252 possible combinations (10C5); the next five loops yeeld the 120 possible five-element permutations (5P5). In all, 30240 permutations are obtained. These five-digit non-repeating digit numbers are squared. Only if the digits of the squares add up to 45 AND the product of the digits is zero, a time consuming digit comparison routine is performed. The forty or so five-digit numbers are stored on a table. At the end, the matching pairs are found and the digit comparison routine is reused. This approach is not obviously the fastest one but at least I am sure I didn't miss any pair. I hope I have made myself clear :-) (I always have trouble trying to explain how I did something, especially in a language not my own).

Well, that digit-comparison routine is not something I am proud of. I just wrote it a while back for generating the numbers that are displayed in my calculators pictures. If you pay attention to them, they all show 10 non-repeating digits. For instance, my 11C depicted in thread below shows 34679108.25. Dividing this number by pi, we obtaing an integer factor, in this case 11038703, which is related to the calculator by the first two digits (This will work even in a 12-digit calculator, like the 42S).

I am sorry not being able to convert the program to the HP-71B Basic. I'd have to print the HP-71B manual and read it all, as I know practically nothing of 71B.

Best regards,

Gerson.

Edited: 14 May 2006, 7:59 p.m.

Part 3 (just a place holder)

*Message #17 Posted by **Crawl** on 16 May 2006, 3:17 p.m.,*
in response to message #1 by Valentin Albillo

By the time I saw this, challenges 1 and 2 were pretty much solved, which mostly killed my interest in them.

But I do have a calculator program (inelegant though it may be) for challenge 3, and it's running now.

It's already found two solutions:

(35172 , 60984)

and

(57321 , 60984)

I'll post the other answers and the source code to the program later tonight.

Re: Part 3 (full solutions)

*Message #18 Posted by **Crawl** on 16 May 2006, 7:21 p.m.,*
in response to message #17 by Crawl

The solution pairs are

(35172 , 60984)

(57321 , 60984)

(59403 , 76182)

(58413 , 96702)

I used a TI-89 to solve the problem. I could have used an RPL machine, too, I guess, but I just so happened not to. I'm not surprised to hear about a QBasic solution. Comparing digits by converting numbers to strings is part of the idea of the solution, which is easily done in basic languages.

The solution is straightforward, pretty much just brute force. It's rather slow on a real TI-89, but fairly fast on an emulator.

It only finds one 5 digit number whose digits are all different, and whose square has all ten digits. I then manually recorded the result, and manually changed the line

32016->x

to just past the given answer to get the next solution. (eg., to 35178->x).

There are only so many solutions of this kind, so it's trivial for a human being to compare them and see which pairs satisfy the required condition.

Anyway, the code:

Quote:

```
sqrchall()
Prgm


@Program's purpose is to find two 5-digit numbers, together which
 have all digits, 0-9, and whose squares, independently, also each
have all ten digits.


Local x
Local x1
Local s
Local s1
Local n
Local old
Local new


32016->x


Lbl again
string(x)->s


@Now we must test that every digit is unique.
```

```
0->old
1->n


Lbl test1


mid(s,n,1)->s1
2^(expr(s1))->new


If (new and old) =/= 0 Then


x+3->x
If x>98765 Then
Stop
EndIf


Goto again


Else


old+new->old
n+1->n
if n<=5 Then
Goto test1
EndIf
EndIf


x^2->x1
string(x1)->s


0->old
1->n


@Now we test that the square has all the digits


Lbl test2
```

```
mid(s,n,1)->s1
2^(expr(s1))->new

If (new and old) =/= 0 Then

x+3->x
If x>98765 Then
Stop
EndIf

Goto again

Else

old+new->old
n+1->n
if n<=10 Then
Goto test2
EndIf
EndIf

Disp x

EndPrgm
```

The numbers output by the program are

```
35172
37905
39147
43902
46587
53976
54918
57321
58413
59403
60984
63051
```

```
63129
69513
76182
78453
80361
81945
85743
86073
87639
89145
89523
90153
91248
91605
96702
```

The interesting thing is, both the first and last numbers are used in solution pairs.

Edited: 16 May 2006, 7:28 p.m.

Re: Part 3 (full solutions)

*Message #19 Posted by **Crawl** on 17 May 2006, 3:12 p.m.,*
in response to message #18 by Crawl

And for the heck of it, I ported my program to Qbasic:

Quote:

```
DIM x1 AS DOUBLE

DIM m(27)

x = 35172

l = 1

again:

s$ = STR$(x)

old = 0
```

```
n = 1

test1:

s1$ = MID$(s$, n + 1, 1)

new = 2 ^ VAL(s1$)

IF new AND old THEN

x = x + 3 IF x > 98765 THEN GOTO final

GOTO again

ELSE

old = old + new n = n + 1

IF n <= 5 THEN GOTO test1

END IF

x1 = x ^ 2

s$ = STR$(x1)

old = 0

n = 1

test2:

s1$ = MID$(s$, n + 1, 1)

new = 2 ^ VAL(s1$)

IF new AND old THEN

x = x + 3 IF x > 98765 THEN GOTO final
```

```
GOTO again

ELSE

old = old + new n = n + 1

IF n <= 10 THEN GOTO test2

END IF

m(l) = x

x = x + 3

l = l + 1

IF x < 98765 THEN GOTO again

final:

FOR a = 1 TO 26

FOR b = a + 1 TO 27

old1 = 0 old2 = 0

FOR n = 1 TO 5

s$ = STR$(m(a))

s1$ = MID$(s$, n + 1, 1)

old1 = old1 + 2 ^ VAL(s1$)

NEXT

FOR n = 1 TO 5

s$ = STR$(m(b))
```

```
s1$ = MID$(s$, n + 1, 1)

old2 = old2 + 2 ^ VAL(s1$)

NEXT

IF (old1 AND old2) THEN

ELSE

PRINT m(a), m(b)

END IF

NEXT

NEXT
```

It's even more complete than the earlier version, because it actually finds the pairs on its own. It runs (on this computer) in about 9 seconds.

I guess part of the point of these challenges is to impress us with how much can be done with just a calculator. So, I can see why submitting a solution in Qbasic or something could be seen as subverting that purpose.

However, one application of calculators is that they give a portable and easy way of testing "proof of principle" of some programs. So, writing a program on a calculator, verifying that it works, and then running a similar program on a computer to get the answer faster seems to me to be almost a "real world" way of demonstrating a way to use a calculator.

Re: Part 3 (full solutions)

Message #20 Posted by ***Gerson W. Barbosa*** *on 17 May 2006, 6:02 p.m.,*
in response to message #19 by Crawl

Hello Crawl,

> Quote:
>
> ---
>
> It runs (on this computer) in about 9 seconds
>
> ---

I've just tested it on my computer, a Pentium III @ 500MHz: 9.5 seconds, the same time I achieved with my program. I made a slight modification in my program to avoid losing time with unallowed numbers, that is, five-numbers or squares beginning with zero. The running time has dropped to 7.3 seconds. However, if you do a simple modification to your program it will run faster (or less slow) than mine: 5.9 seconds.

Just insert the lines below between **DIM m(27)** and **x = x + 3** and replace the ocurrences of **2 ^ VAL(s1$)** with **P2(VAL(s1$))**.

```
FOR I = 0 TO 9
  P2(I) = 2 ^ I
NEXT I
```

Comparing both programs I realize mine is needlessly more complicated than it should have been.

Regards,

Gerson.

Edited: 17 May 2006, 6:04 p.m.

Re: Part 3 (full solutions)

Message #21 Posted by ***Crawl*** *on 17 May 2006, 10:01 p.m.,*
in response to message #20 by Gerson W. Barbosa

Thanks for the comments. I also realized belatedly that I inadvertantly cheated on my second program. I started searching at 35172, which is actually the first solution, instead of 32016. (The smallest number could be sqr(1023456789) = 31992, but that has two repeated digits. 32016 is the smallest number greater than 31992 that has no repeated digits and is divisible by three)

(And I kind of cheated anyway, by DIM'ing m with 27 entries, because I knew ahead of time that there were 27 5-digit candidates, but that's a minor cheat, imho)

I tried my program, with your suggestions, on this computer (a 3800+ AMD Athlon; I'm not sure of the clockspeed, but it must be Ghz), and it finished in 1.5 seconds! And that's Qbasic. A compiled Quickbasic .exe file should finish almost instantly.

Edited: 17 May 2006, 10:10 p.m.

Re: Part 3 (full solutions)

Message #22 Posted by ***Gerson W. Barbosa*** *on 18 May 2006, 8:52 p.m.,*
in response to message #21 by Crawl

Hi again,

I had figured the smallest number should be greater than 35136 because I had wrongly taken the square root of 1234567890 instead of 1023456789. Anyway, I forgot to take this into account in my previous program. Now, by discarding numbers beginning with 0, 1 and 2 the running time has dropped a little: 5.7 seconds (2 seconds on my son's AMD Athlon 1700+ next room :-) (1700+ and 3800+ in Athlon processors allegedly means they are equivalent to Pentium processors running at 1.7 GHz and 3.8 GHz respectively although their actual clock speeds are lower). On the 200LX it took 27 minutes and 46 second to run (measured with the built-in timer in QBasic), faster but 20 times slower than Valentin's program.

Your QBX-compiled program runs in less than 1 second (~ 0.8 s) on this computer (500 MHz). I estimate it runs in about 120 milliseconds on a 3800+ Athlon.

Gerson.

Alternate Solution to Problems 1 and 2

Message #23 Posted by ***Crawl*** *on 28 May 2006, 11:06 a.m.,*
in response to message #22 by Gerson W. Barbosa

It occured to me that you don't need to "solve" for the next digit to get the correct answer. Well, it depends on how you do it. For the "6" solution, you do. But for the "5" solution, you don't! The correct next digit is given by squaring the current number.

Example, the one digit solution is "5". 5^2 = 25. So, the correct two digit solution is "25".

25^2 = 625. The correct 3 digit solution is 625.

625^2 = 390625. So, the correct 4 digit solution would be 625 again. We can save some time here then and say the 5 digit solution is 90625.

90625^2 = 8212890625. The 6 digit solution is 890625.

890625^2 = 793213890625. The 7 digit solution is 3890625.

I didn't notice anyone else make that observation.

Anyway, this is the program that will solve for any number of digits. You put the number of digits you want on the stack, then call the program:

```
<< 5 1 -> N X M
  <<
     DO X SQ ->STR DUP SIZE M - DUP SUB OBJ-> 10 M ^ * X + 'X' STO
     M 1 + 'M' STO
     UNTIL M N ==
     END X
  >>
>>
```

It gives the exact same answers as quoted before for the 10 and 100 digit solutions, so no need to repost those.

It takes just over 14 seconds to give the 100 digit solution on a real 49G+.

It could be sped up a little bit by grabbing two digits when the next digit is zero (like I did above "manually"), but oh, well.

I might try some of the other posted ideas for speeding up my program for challenge 3, if I have the time and inclination, for the heck of it (since of course the challenge is technically over).

Edited: 28 May 2006, 2:22 p.m.

S&SMC#16: My Original Solutions & Comments

Message #24 Posted by ***Valentin Albillo*** *on 19 May 2006, 6:43 a.m.,*
in response to message #1 by Valentin Albillo

Hi all,

As always, thanks to all of you who were interested in my S&SMC#16, most specially to the ones who took their time to analyze and eventually solve it. As promised, I'm posting now my original solutions plus relevant comments:

Scroll 1: Being yourself ...

> *"Write a program that finds out all 10-digit integers (base 10, of course) that upon being squared, the result's rightmost digits exactly reproduce the original number"*

These are well-known numbers, usually called *automorphs*. In a sense, they can be considered additional roots to the equation:

$$x^2 = x$$

if we're willing to accept 'infinite' numbers as solutions. For all numeric bases, there are always two solutions, namely **0** and **1**. For N-base numbers where N is either *prime* or the power of a prime, these are the *only* solutions. On the other hand, when N=10, we also have two other solutions, namely:

... 8212890625 and **... 1787109376**

which can be extended infinitely. In all numeric bases, their sum is always of the form 11, 101, 1001, 10001, 1000000000...0000000001, etc, so it suffices to compute just *one* of them and the other can be immediately written down with a trivial computation. As Thomas Okken and Bram demonstrated, a little theoretical work can do wonders to help implement a simple, concise algorithm, and in particular, to stablish that **5** and **6** are the only possible 1-digit solutions in base 10 (disregarding the trivial solutions 0 and 1, which can't be extended with non-zero digits), and they can be extended indefinitely a digit at a time. Which is more, only the "5" solution needs to be extended, the "6" solution is computed immediately from it.

This is my original 4-line (101-byte) version for the HP-71B:

```
1 N=5 @ FOR K=2 TO 10 @ P=10^K @ R=P/10 @ FOR D=0 TO 9
2 M=R*D+N @ IF NOT MOD(M*M-M,P) THEN 4
3 NEXT D
4 N=M @ NEXT K @ DISP M;10000000001-M
```

Running it produces:

```
>RUN
   8212890625    1787109376
```

in negligible time. This works Ok thanks to the *extreme high-quality nature of HP's arithmetic routines*, as implemented in the HP-71B and most other HP calc models, which allows for proper computation of MOD even when you would fear that lost digits beyond 12-digit precision in the computation of M^2 would lead to inaccurate results. That's not so and the above program works fine in the HP-71B. It might **not** if converted to other less accurate, non-HP models.

The solutions are indeed correct, as we have:

8212890625^2 = 6745157241<u>8212890625</u>

1787109376^2 = 3193759921<u>787109376</u>

We also have the following interesting results:

8212890625 + 1787109376 = <u>10000000001</u>

8212890625 * 1787109376 = 1467733384<u>0000000000</u>

8212890625 * (8212890625-1) = 6745157241<u>0000000000</u>

```
1787109376 * (1787109376-1) =  3193759920000000000
```

$(1787109376-1)^2 = 319375991\underline{8212890625}$

$(8212890625-1)^2 = 674515724\underline{01787109376}$

The equivalent RPN version for the HP-15C is this short, fast 44-step routine:

```
01 *LBL A   14  10      27 ENTER    37 PSE
02  6       16  /       28  X^2     38 STO 0
03 STO 0    17 STO 3    29 LASTX    39 ISG 1
04 2.01     18  .09     30  -       40 GTO 1
08 STO 1    21 STO I    31 RCL/ 2   41  1
09*LBL 1    22*LBL 0    32 FRAC     42 RCL 0
10 RCL 1    23 RCL I    33 ISG I    43 RCL- 2
11 INT      24 INT      34 TEST 0   44  -
12 10^X     25 RCL* 3   35 GTO 0
13 STO 2    26 RCL+ 0   36 X<>Y
```

which is essentially a direct translation of the above, which the added nice touch that it will pause to let you see each correct digit as it's being added. Let's run it:

```
GSB A -> (76) -> (376) -> ...
      -> 8212890625
 X<>Y -> 1787109376
```

which only takes *85 seconds to run*, including 9 one-second pauses.

Amazingly this program *also works*, despite the fact the HP-15C is limited to 10-digit results, not 12-digit as in the HP-71B !. This would seem to be a fatal pitfall when having to deal with the square of 10-digit integers, which comes out to 19/20-digit results, but thanks again to the *utterly incredible* quality of HP's arithmetic routines as implemented in the HP-15C, the above program *does run fine*, against all odds, and directly computes the correct "6" solution, which is immediately used to get the "5" solution without further computation. Again, it would very likely fail if converted to non-HP 10-digit models.

Scroll 2: ... Big Time !

"Write a program to find and orderly output all solutions with the same requirements as above, but this time for 100-digit integers"

My original solution is this short, 635-byte program for the HP-71B:

```
100 DESTROY ALL @ DIM M$[128],N$[128],P$[256] @ M$="5"
110 FOR I=2 TO 100 @ CALL MS(M$,P$) @ M$=REV$(P$)[I,I]&M$ @ NEXT I
120 GOSUB 140 @ DISP @ M$[100]="6"
130 FOR I=1 TO 99 @ M$[I,I]=STR$(9-VAL(M$[I,I])) @ NEXT I @ GOSUB 140 @ END
140 FOR I=1 TO 5 @ DISP M$[20*I-19,20*I] @ NEXT I @ RETURN
150 !
160 SUB MS(P$,R$) @ OPTION BASE 1 @ DIM A$[512],E$[512] @ A$=FNL$(P$)
170 X=LEN(A$) DIV 6 @ Z=2*X @ U=10^6 @ V=U-1 @ DIM A(X),C(Z),C$[Z*6]
180 FOR I=1 TO X @ A(X+1-I)=VAL(A$[I*6-5,I*6]) @ NEXT I
190 FOR I=1 TO X @ M=A(I) @ IF RES THEN L=I ELSE 230
200 FOR J=1 TO X @ N=A(J) @ IF RES THEN P=M*N @ C(L)=C(L)+RES ELSE 220
210 P=RES @ IF RES>V THEN C(L)=RMD(P,U) @ C(L+1)=C(L+1)+P DIV U
220 L=L+1 @ NEXT J
230 NEXT I @ FOR I=Z TO 1 STEP -1 @ IF C(I) THEN 250
240 NEXT I
250 C$=STR$(C(I))
260 FOR I=I-1 TO 1 STEP -1 @ C$=C$&FNL$(STR$(C(I))) @ NEXT I @ R$=C$
270 DEF FNL$[512](A$) @ E$=A$
280 IF RMD(LEN(E$),6) THEN E$="0"&E$ @ GOTO 280 ELSE FNL$=E$
```

where the main program is a mere 5 lines long, and simply calls subprogram MS to compute the multiprecision square of a multiprecision integer. MS is a very simple subprogram which takes two string arguments, namely:

```
P$   contains the multiprecision number to square, as a string
R$   contains the multiprecision result
```

Having the number and result be strings is much less efficient and more memory-wasting than using arrays, but it makes for easy use *right from the keyboard* and I happened to have at hand this little subprogram I wrote long ago. Tha main program simply uses it to compute the "6" solution, and then outputs both the 100-digit "6" solution and "5" solution in 20-digit groups:

```
>RUN
   39530073191081698029
   38509890062166509580
   86381100055742342323
   08961090041066199773
   92256259918212890625


   60469926808918301970
   61490109937833490419
   13618899944257657676
   91038909958933800226
   07743740081787109376
```

These results are indeed correct, as we do have:

```
    3953007319108169802938509890062166509580863811000557423423230896109004106619977392256259918212890625^2
    =
    1562626686492275980643313531100701521758660998626415856673412452830198131380176443023792967609238890
    3953007319108169802938509890062166509580863811000557423423230896109004106619977392256259918212890625
and
    6046992680891830197061490109937833490419136188999442576576769103890995893380022607743740081787109376^2
    =
    3656612048275936374766293750976368502596933376625301009826950660612189918140221658511273131183457641
    6046992680891830197061490109937833490419136188999442576576769103890995893380022607743740081787109376
```

By the way, this program is general in nature so you can use it to compute more or less than 100 digits by simply changing some constants here and there. Also, as stated, you can use subprogram MS from the keyboard to square large integer numbers, like this:

```
>M$="31479267847267265637800300423742371877 51" @ CALL MS(M$,P$) @ P$


   990944304199994668606301320793256246285161361497649412232480230377977722443800 1
```

Scroll 3: All included

"Write a program to find and output all pairs of 5-digit integers such that together they do include all digits from 0 to 9 and each of their respective 10-digit squares also include all digits from 0 to 9 as well"

This is my original 13-line, 393-byte program for the HP-71B:

```
 10 DESTROY ALL @ DIM S(50) @ A$="1234567890" @ L=0 @ FOR A=3 TO 9
 20 T=10000*A @ FOR B=0 TO 9 @ IF B=A THEN 100
 30 U=T+1000*B @ FOR C=0 TO 9 @ IF C=A OR C=B THEN 90
 40 V=U+100*C @ FOR D=0 TO 9 @ IF D=A OR D=B OR D=C THEN 80
 50 W=V+10*D @ FOR E=0 TO 9 @ IF E=A OR E=B OR E=C OR E=D THEN 70
 60 N=W+E @ IF NOT SPAN(A$,STR$(N*N)) THEN L=L+1 @ S(L)=N @ DISP N;N*N
 70 NEXT E
 80 NEXT D
 90 NEXT C
100 NEXT B @ NEXT A @ DISP "Checking matching pairs ..."
110 FOR I=1 TO L @ A=S(I) @ FOR J=I+1 TO L @ B=S(J)
120 IF NOT SPAN(A$,STR$(A)&STR$(B)) THEN DISP A;B,A*A;B*B
130 NEXT J @ NEXT I @ DISP "OK"
```

Upon running, it first does find out all 5-digit (nonrepeated) numbers whose 10-digit squares feature all digits from 0 to 9, then automatically matches them in pairs to quickly produce all four solutions, namely:

```
>RUN
   35172  1237069584
   37905  1436789025
```

```
 39147  1532487609
 43902  1927385604
 46587  2170348569
 53976  2913408576
 54918  3015986724
 57321  3285697041
 58413  3412078569
 59403  3528716409
 60984  3719048256
 63051  3975428601
 63129  3985270641
 69513  4832057169
 76182  5803697124
 78453  6154873209
 80361  6457890321
 81945  6714983025
 85743  7351862049
 86073  7408561329
 87639  7680594321
 89145  7946831025
 89523  8014367529
 90153  8127563409
 91248  8326197504
 91605  8391476025
 96702  9351276804


Checking matching pairs ...


 35172  60984        1237069584  3719048256
 57321  60984        3285697041  3719048256
 58413  96702        3412078569  9351276804
 59403  76182        3528716409  5803697124


 OK
```

This runs in less than 9 seconds in Emu71 @ 2.4 Ghz, and less than 25 minutes in a real HP-71B.

The algorithm used is pretty straightforward: 5 nested loops generate all 5-digit numbers from 30000 to 99999, in an optimized way to minimize computation, and *sieving out* those having repeated digits. The ones which remain after the sieving process are then squared and their squares are checked to see if they feature all digits from 0 to 9. The ones which pass muster are then collected in an array, later to be matched to select compatible pairs that together include all digits as well, which are suitably output with their respective squares for added visual confirmation.

Note:

You might be interested to see how a 10-digit number is checked to see if it includes all digits from 0 to 9. This is done in two separate occasions at lines 60 and 120. Instinctively, one would resort to using a loop to do the check, but it's far better to use the **SPAN** function as shown, thus completely avoiding both loops within the maximum nesting level, and so speeding up the program a real lot.

The SPAN function is a little-known, little-used string function which can be found in a number of different LEX files for the HP-71B, though it was originally featured in STRNGLEX, which is a very common LEX, included in a number of ROMs and thus easily available. In particular. it comes already installed and ready to use in Emu71.

If you don't have Emu71 or STRNGLEX, you can substitute it for a simple loop, using POS instead, but it's good to know about non-obvious, time-saving uses for such functions as SPAN. By the way, the program can still be optimized even further by noticing that once you have generated a 5-digit number N which doesn't have repeated digits (say 34567), then another suitable candidate is 99999-N (99999-34567 = 65432) which is automatically guaranteed to also have no repeated digits as well. This can reduce the outer loop (and running time) by almost 50%. I'll leave that as an "exercise for the reader" :-)

Well, that's all for now. Hope you enjoyed it and thanks again to the kind contributors, your postings were as keen as usual and certainly enlightening to all interested readers. See you in S&SSMC#17, coming next month.

Best regards from V.

Edited: 19 May 2006, 7:11 a.m.

Re: S&SMC#16: My Original Solutions & Comments

Message #25 Posted by ***Gerson W. Barbosa*** *on 21 May 2006, 3:37 p.m.,*
in response to message #24 by Valentin Albillo

Hello Valentin,

Thanks for this one more S&SMC. I am looking forward to the next one.

Though this only shows my "lack of programming skills", I am submitting my solution to the third problem. It is essentially the same one I wrote last Sunday with some minor improvements (added line 46; replaced the testing routine, not so much faster but more compact than the previous one). It is still in QBASIC, I hope you don't mind. Anyway, this should be easy to port to SHARP or CASIO pocket computers (I am not sure their BASIC dialects can handle line 520 though).

It runs in slightly less than 30 minutes on the HP-200LX (precisely 29m49s). In short, I'd need an HP-200LX to do what you can do on the HP-71B, which is about 19 times slower, according to a certain benchmark. And I needed ten nested loops and countless more lines while you needed only five loops and a few lines! I am just glad I don't have to program for a living, otherwise I'd starve :-)

Best regards,

Gerson.

```
--------------------------------------------------------------------------------


 96702  58413  9351276804  3412078569
 59403  76182  3528716409  5803697124
 60984  35172  3719048256  1237069584
 60984  57321  3719048256  3285697041


 15:56:07
 16:25:56


--------------------------------------------------------------------------------


  5 CLS
  7 T$ = TIME$
 10 DEFDBL X-Z
 15 DEFLNG A-B
 17 DEFINT I-V
 18 DIM X(50)
 20 FOR I = 0 TO 5
 22   LOCATE 2, 2: PRINT I;
 25   FOR J = I + 1 TO 6
 30     FOR K = J + 1 TO 7
 35       FOR L = K + 1 TO 8
 40         FOR M = L + 1 TO 9
 43           A(1) = I: A(2) = J: A(3) = K: A(4) = L: A(5) = M
 45           FOR N = 1 TO 5
 46             IF A(N) < 3 THEN 180
 50             FOR P = 1 TO 5
 60               IF N = P THEN 170
 70               FOR Q = 1 TO 5
 75                 IF (N = Q) OR (P = Q) THEN 160
 90                 FOR R = 1 TO 5
 95                   IF (N = R) OR (P = R) OR (Q = R) THEN 150
110                   FOR S = 1 TO 5
```

```
115                     IF (N = S) OR (P = S) OR (Q = S) OR (R = S) THEN 140
130                     X = 10000 * A(N) + 1000 * A(P) + 100 * A(Q) + 10 * A(R) + A(S)
135                     Y = X * X
138                     GOSUB 500
140                   NEXT S
150                 NEXT R
160               NEXT Q
170             NEXT P
180           NEXT N
190         NEXT M
195       NEXT L
200     NEXT K
205   NEXT J
210 NEXT I
212 CLS : KT = 100
214 ' Searches for matching pairs
215 FOR I1 = 0 TO 48
217    IF X(I1) = 0 THEN 250
220    FOR J1 = I1 + 1 TO 49
225      Y = 100000 * X(I1) + X(J1)
230      IF X(J1) <> 0 THEN GOSUB 500 ELSE 240
235    NEXT J1
240 NEXT I1
250 PRINT : PRINT " "; T$: PRINT " "; TIME$
260 END
499 ' Tests whether squares include digits from 0 to 9
500 Y$ = STR$(Y): TS$ = "1111111111"
510 FOR T = 2 TO 11
515    V = VAL(MID$(Y$, T, 1))
520    MID$(TS$, V + 1, 1) = "0"
530 NEXT T
540 IF TS$ = "0000000000" THEN IF KT <> 100 THEN X(KT) = X: KT = KT + 1 ELSE PRINT X(I1); X(J1); X(I1) ^ 2; X(J1) ^ 2
600 RETURN
```

Update:

Checking my original program again, I discovered lines 47, 65, 80, 100 and 120 were not necessary. So these lines have been erased in the listing above and line 130 has been changed. Now the program runs in 29m36s and is five lines shorter (still too many lines though). I hope there are no other primary mistakes left.

```
 47             B(1) = A(N)
 65               B(2) = A(P)
 80                 B(3) = A(Q)
```

```
100                    B(4) = A(R)
120                      B(5) = A(S)
130                      X = 10000 * B(1) + 1000 * B(2) + 100 * B(3) + 10 * B(4) + B(5)
```

Edited: 21 May 2006, 11:17 p.m.

Re: S&SMC#16: My Original Solutions & Comments

*Message #26 Posted by **Valentin Albillo** on 22 May 2006, 5:22 a.m.,*
in response to message #25 by Gerson W. Barbosa

Hi, Gerson:

Gerson posted:

> *"Thanks for this one more S&SMC. I am looking forward to the next one."*
>
> > You're welcome, thanks to you for your continued interest and excellent solutions to my challenges.
>
> *"It is still in QBASIC, I hope you don't mind. Anyway, this should be easy to port to SHARP or CASIO pocket computers (I am not sure their BASIC dialects can handle line 520 though)."*
>
> > As for minding, I'll pass this one and yes, it's pretty easy to port it to SHARP models, line 520 can be entered as is in models such as the SHARP PC-1350, PC-1475, PC-1403H, etc, but not so in earlier models such as the original SHARP PC-1211 (aka TRS-80 PC-1 in the US).
>
> *"I am just glad I don't have to program for a living, otherwise I'd starve :-)"*
>
> > Thanks for the compliment, Gerson, but don't be so unfair to yourself, your solutions are indeed correct and perfectly adequate as they *do* solve the challenge. To be able to analyze an unusual problem, decide on a workable algorithm, then correctly implement it and get valid solutions is no mean feat in itself at all, quite the opposite.
>
> I hope you'll like my very next S&SMC#17, which will be out next month and, in a sense, it's the 'complementary' of this one. Also, it includes a 41-related subproblem ;-)

Best regards from V.

Program to solve challenges #1 & #2

Message #27 Posted by ***Eduardo*** *on 22 May 2006, 5:56 p.m.,*
in response to message #1 by Valentin Albillo

Though the deadline to post solutions has passed, I'd like to send an entry for problems #1 and #2. Number theory being very close to my heart and having recently discovered the HP-49G/G+ function "IABCUV", it suffices to use successive approximations Ю la Newton's method to "roots" of the polynomial x^2-x (not in the sense that x^2-x is near zero, but in the sense that it ends in many zeros). I realized any attempt would involve arithmetic with large integers, so nothing better suited to this purpose than the exact mode of the HP-49G/G+ that allows for many-digit integral arithmetic.

Note that successive approximations to a root of x^2-x are done via the usual Newton method by iterating y=x-(x^2-x)/(2x-1). Using modular arithmetic instead, we obtain the solution below. What IABCD does in the case at hand is compute -(x^2-x)/(2x-1) modulo a suitable power of 10.

Program 'HUNDRED' follows:

```
<< 10 -> X P << 1 SWAP START P SQ 'P' STO X 2 * 1 - P X X SQ - IABCUV DROP X + P MOD 'X' STO NEXT X >> >>
```

Size: 133.5, Crc # 64962d

Usage:

EXACT mode needed (flag 105 cleared in HP 49G).

Level 2: integer n (greater than or equal to 1) Level 1: integer x0 (equal to 5 or 6)

Output: Integer x with last digit x0 and such that x^2 has the same last 2^n digits as x.

Each iteration doubles the number of digits of the (infinitely long formal) solution, hence the interpretation of the parameter n as giving the "approximate" solution to 2^n digits (or less).

Note that x may have fewer than 2^n digits. x0 is the "zeroth-level" solution: namely x0=6 (since 6^2 ends in 6) or x0=5 (since 5^2 ends in 5)

For instance: 4 5 HUNDRED gives 6259918212890625, a 16-digit number whose square ends in the same 16 digits (16=2^4, and the number ends in 5). Taking only the last 10 digits solves half of challenge #1. The other half of challenge #1 is solved by taking the last 10 digits of 4 6 HUNDRED (calculation takes less than a second in my Tungsten E2 emulating the 49G).

Similarly, the last 100 digits of 7 5 HUNDRED and 7 6 HUNDRED solve challenge #2 (less than 2 seconds in my Tungsten E2 emulating the 49G).

Eduardo

Hats off... (N.T.)

Message #28 Posted by ***Vieira, Luiz C. (Brazil)*** *on 23 May 2006, 12:39 a.m.,*
in response to message #27 by Eduardo

Thanks, Luiz! (N.T.)

Message #29 Posted by ***Eduardo*** *on 23 May 2006, 11:24 a.m.,*
in response to message #28 by Vieira, Luiz C. (Brazil)

Re: Program to solve challenges #1 & #2

Message #30 Posted by ***Valentin Albillo*** *on 23 May 2006, 8:55 a.m.,*
in response to message #27 by Eduardo

Hi, Eduardo:

Really, really excellent RPL solution for the 49 series !!

I was wondering why RPL solutions to my challenges are usually few and far between, when actually RPL machines are indeed the more powerful models available for the tasks featured, and thus, in theory, ideally suited to this kind of computation-intensive challenges of mine.

Fortunately, yours is a perfect example of the innovative solutions one can arrive at using the powerful RPL capabilities present in HP's flagship models.

Never mind the (nonexistent) deadline, all ideas and implementations are always welcome, not the mention such original ones as yours. I sincerely hope you'll be interested in future S&SMCs and meanwhile, thanks for contributing and

Best regards from V.

Re: Program to solve challenges #1 & #2

Message #31 Posted by ***Eduardo*** *on 23 May 2006, 11:08 a.m.,*
in response to message #30 by Valentin Albillo

Valentin,

Thanks for your praise. My first HP calculator was a 28S that I bought when I was in high school, so RPL is what I'm most familiar with. Later I sold it to upgrade to a 48SX which was later unfortunately stolen. Now that I could more easily afford calculators, the golden era is over. I have all models in the 48-49 series, a 42S that I happened by chance to buy from Raymond del Tondo almost three years ago, a 33s, and a couple others (20s and 32Sii with barely working ENTER key). But I still miss my 28S on occasion.

I envy you, and would love to have a 71B. My first programmable calculators were actually a Tandy/Radio Shack PC7 and later a Casio FX-850P, both with BASIC language. I also would like a 41CX very much. Unfortunately, it's unlikely I'll ever buy them if I have to pay eBay prices for them. Not to mention I dislike and distrust eBay in general.

I read your piece "Long Live the HP 42S" a few months back. Thanks for your contributions to this community.

Eduardo

Edit: I noticed you are actually selling some of these jewels! I'll drop you a note.

Edited: 23 May 2006, 11:20 a.m.

[Return to Index | Top of Index]

GTO Go back to the main exhibit hall