



## HP Forum Archive 15

[ [Return to Index](#) | [Top of Index](#) ]

### Short & Sweet Math Challenge #9: Divisibility !

Message #1 Posted by [Valentin Albillo](#) on 16 May 2005, 9:31 a.m.

Hi, all:

("Di-vi-si-bi-li-ty" ... far too many 'i's for just one single word, reminds me of the infamous song "*Miss Lilly Higgins sings shimmy in Mississippi's spring*" ...

Here's the brand new **S&SMC#9** I've just concocted for your HP-programming pleasure, let's hope it gathers as much interest among forum contributors as the previous one did (still getting answers long after it was duly archived). Just like it, this one can also be addressed using just about any HP calculator, combined with a generous helping of ingenuity on the part of the user. Enough intro, let's go right to the

### Challenge:

- (1). Find a 10-digit integer number which consists of the digits 0-9 \*without repetition\* such that taking its L (from 1 to 10) leftmost digits the resulting number is *exactly divisible* by L. For instance, the number 32165 would be a solution, as it has no repeating digits and

3 is exactly divisible by 1 =>  $3/1 = 3$   
 32 is exactly divisible by 2 =>  $32/2 = 16$   
 321 is exactly divisible by 3 =>  $321/3 = 107$   
 3216 is exactly divisible by 4 =>  $3216/4 = 804$   
 32165 is exactly divisible by 5 =>  $32165/5 = 6433$

except for the unfortunate fact that it's got only 5 digits while we're after a 10-digit number. **The solution is unique.** As usual, you're expected to produce a program for your chosen calculator(s) that upon running will find the only solution *and* make sure there are no others.

If you succeed, you may want to extend your quest to also:

- (2). Find out just *how many* solutions there are for N-digit numbers, with N ranging from 1 to 10: for N=10 there's just one solution, but what about N=4 or N=9 ? Numbers beginning with a 0 are to be disregarded, *including "0" itself* (i.e.: "0" is \*not\* to be considered a valid solution for the case N=1). Your program must output just the number of solutions for each N, displaying the solutions themselves is not required.

- (3). Same as (2) above, but *repeated digits are* allowed, the divisibility condition and not beginning with '0' being the only requirements.

As stated, you must produce one or several programs for your HP calculator(s) that will compute what's required, the shorter and faster the better. Giving \*just\* the solutions (but no program) or providing programs for machines other than (preferably HP) \*calculators\* is to be considered as blatant disrespect to the stated rules and/or a clear statement of utter inability to comply within these terms.

Two or three days from now I'll post my solution (plus comments), which is essentially a rather simple, unoptimized 4-line program (plus variations) for a bare-bones [HP-71B](#) that takes less than 2 minutes to find the only solution to (1), less than half an hour to count all [706 solutions](#) to (2), and less than 4 hours to count all [11,453 solutions](#) to (3) (in a physical [HP-71B](#), that is; emulated times for [Emu71](#) @ 2.4 Ghz are 1 second, 8 seconds, and 10 minutes, respectively). These are not the best times possible in a 71B, of course. In order for you to test your programs, results for (2) and (3) when N=4 are 168 and 375 solutions, respectively.

Let's see your efforts. If you can't solve this challenge for N up to 10, by all means do try with a lower limit for N, say 5, but do try all the same. Good luck and happy programming, and above all, enjoy !

Best regards from V.

## Re: Short & Sweet Math Challenge #9: Divisibility !

Message #2 Posted by [GWB](#) on 16 May 2005, 9:46 p.m.,  
in response to message #1 by [Valentin Albillo](#)

Hi Valentin,

So far I have preferred to do it by hand. I just had to make a table of the thirty or so possible solutions according to the first four digits. After eliminating a few that wouldn't fit, I found the solution. (I think it would take longer, at least for me, to get to the solution by writing a program). Very interesting problem, though.

In order to respect your rules, I won't give the solution here. I will just say the first two digits of the solution are the number of an HP calculator, so are the third and fourth digits, the fourth and fifth digits and the eighth and ninth digits, right? I'm looking forward for your and other people's solution.

Regards,

Gerson.

-----  
Just in case, here is my table:

1236 1296 1472 1476 1832 1836 1892

3216 3276 3692 3698 3812 3816 3872  
3876 7236 7296 7412 7416 7832 7836  
7892 9216 9276 9632 9812 9816 9872  
9876

As everybody have seen, odd-position digits have to be odd and even-position digits have to be even. Also, the 5th digit is 5 and the 10th digit is 0. So, beginning with 12365 and trying the remaining even digits 4 and 8, both 123656 and 123658 are not divisible by 6, so it is discarded. After trying 12 more numbers, the solution is eventually found: 3816547290. This scheme is hard, if impossible, to put in an algorithmic form though.

*Edited: 18 May 2005, 6:49 p.m.*

## Re: Short & Sweet Math Challenge #9: Divisibility !

Message #3 Posted by *Eamonn* on 16 May 2005, 10:20 p.m.,  
in response to message #1 by *Valentin Albillo*

Hi Valentin,

Here is a HP-32S solution for the first part of the challenge. It finds the single solution, 3816547290. Running time is 132 seconds.

To use the program, enter GTO P, then R/S. The program stops to display any solutions it finds. It finishes with a branch to a non-existent label. This makes it more obvious to tell when the program has stopped vs. when it is displaying a solution.

I had to resort to some optimizations and trickery to get around the restrictions of the HP-32S. In doing so, I made use of the following observations:

- Since a 2-digit number must be divisible by 2, it must therefore end in an even digit. By the same token, a four, six, eight or ten digit number must also end in an even number.
- For a ten digit number to be divisible by 10, it must end in a zero.
- Since all the even-numbered digits of the solution(s) are even, therefore all the odd-numbered digits must be odd.

Two outer loops in the program generate all the two digit numbers that have the first digit odd and the second digit a non-zero even number. A recursive subroutine generates more digits, making sure that no digits are re-used and testing each time for divisibility by the number of digits in the number. If we ever get a 9-digit number, then it automatically means that we have found a ten digit solution, since the digit zero was not used to form the 9-digit number.

The program should be easily modifiable to solve the second part of the challenge - I'll post it if/when I get some time to complete it. I'll need to think a bit more of how to fit the third part of the challenge into the HP-32S. Restricting the solutions to eight digits may be necessary because the seven levels of stack depth on this device may not be enough.

Thanks for the challenge.

Eamonn.

```

LBL P
0
STO Y ; Flags Variable
STO W ; Count of numbers found
3
STO i ; Divisor
10
STO X ; Handy Constant
1.00902 ; For A = 1 to 9 Step 2
STO A
[23 Bytes Chksum = 43DE]

```

```

LBL A
RCL A
IP
10^X
STO+ Y ; Flags[A] = 1
2.00802 ; For B = 2 to 8 Step 2
STO B
[18.5 Bytes Chksum = A74D]

```

```

LBL B
RCL B
IP
10^X
STO+ Y ; Flags[B] = 1
RCL A
RCL* X
RCL+ B
IP
STO M ; M = A * 10 + B
XEQ D ; call recursive function
RCL B
IP
10^X
STO- Y ; Flags[B] = 0
ISG B
GTO B ; Next B
RCL A
IP
10^X
STO- Y ; Flags[A] = 0
ISG A

```

```
GTO A      ; Next A
RCL Z      ; display the count of numbers found
GTO Z      ; Program finishes on an error - Z is a non-existent label
[37.5 Bytes Chksum = EB63]
```

```
LBL D      ; Recursive function
2.00802
STO (i)    ; For odd digits, test only odd numbers
RCL i
2
/
FP
X=0?
GTO L
1.00902    ; Even digit - so test even numbers
STO (i)
[32.5 Bytes Chksum = 6C06]
```

```
LBL L
RCL Y      ; Y contains the flags
RCL (i)    ; Code to test flag (i)
IP
10^X
/
IP
RCL/ X     ; X = 10
FP
X<>0?
GTO E      ; if Flags[LCV[i]] <> 0 then done
RCL (i)
IP
10^X
STO+ Y     ; Flags[LCV[i]] = 1
RCL M
RCL* X
RCL+ (i)
IP
STO M      ; M = M * 10 + LCV[i]
RCL i
/
FP
X<>0?
GTO F      ; If M is not divisible by i then goto f
9
RCL i
X=Y?
```

```

GTO G      ; if i is already 9 then answer is found
1
STO+ i
XEQ D      ; Else increment i and test for next digit
1
STO- i
GTO F
[52.5 Bytes Chksum = 4385]

LBL G      ; We are here if 9-digit number found
RCL M
RCL *X     ; Display M*10
STOP
STO V      ; Store the result
1
STO+ W     ; Increment count
[10.5 Bytes Chksum = 7F10]

LBL F      ; Some Cleanup
RCL M
RCL/ X
IP
STO M      ; M = M / 10
RCL (i)
IP
10^X
STO- Y     ; Flags[LCV[i]] = 0
[13.5 Bytes Chksum = 9D81]

LBL E
ISG (i)    ; Next LCV[i]
GTO L
RTN
[6 Bytes Chksum = 4EC6]

```

## Re: Short & Sweet Math Challenge #9: Divisibility !

Message #4 Posted by [Arnaud Amiel](#) on 18 May 2005, 4:14 a.m.,  
in response to message #3 by Eamonn

I haven't have time to give it a try yet but an other simplification is that the 5th number has to be 5 or 0 and as the 10th number has to be 0, the 5th is 5.

I didn't go any further than that and may not before the weekend.

Arnaud

**Re: Short & Sweet Math Challenge #9: Divisibility !**

Message #5 Posted by **J-F Garnier** on 18 May 2005, 7:54 a.m.,  
in response to message #4 by Arnaud Amiel

Going one step further in this approach to solve Valentin's first problem:

The 4-digit number being divisible by 4, and the 3rd digit being {1,3,7,9}, the 4th digit is either 2 or 6.

The 8-digit number being divisible by 8, the 7th digit being {1,3,7,9}, and the 6th being even, the only combinations of digits 7th and 8th are: 16, 32, 72, 96, so the 8th digit is either 2 or 6.

As digits 4th and 8th are either 2-6 or 6-2, the two other even digits 2nd and 6th are either 4-8 or 8-4.

Finally, there are 48 remaining candidates: 24 combinations for the 4 odd digits 1st, 3rd, 7th and 9th, digit 7th fixing digit 8th and in turn digit 4th, and 2 combinations for digits 2nd and 6th.

Below is the (probably) fastest HP-71B program to find the solution:

```

10 DATA 143658729,143258967,147658329,147258963,149658327,149658723
20 DATA 341658729,341258967,347258169,347258961,349258167,349658721
30 DATA 741658329,741258963,743258169,743258961,749258163,749658321
40 DATA 941658327,941658723,943258167,943658721,947258163,947658321
50 DATA 183654729,183254967,187654329,187254963,189654327,189654723
60 DATA 381654729,381254967,387254169,387254961,389254167,389654721
70 DATA 781654329,781254963,783254169,783254961,789254163,789654321
80 DATA 981654327,981654723,983254167,983654721,987254163,987654321

```

```

90 FOR I=1 TO 48
100 READ A
110 IF MOD(A,9)=0 THEN
120 X=A
130 X=X DIV 100
140 IF MOD(X,7)=0 THEN
150 X=X DIV 10
160 IF MOD(X,6)=0 THEN
170 X=X DIV 1000
180 IF MOD(X,3)=0 THEN
190 DISP A*10
200 END IF

```

```
210     END IF
220     END IF
230     END IF
240 NEXT I
```

(Using the IF-ENDIF structure, and listed with PBLIST command from JPCROM)

J-F

*Edited: 18 May 2005, 8:20 a.m.*

### **Re: Short & Sweet Math Challenge #9: Divisibility !**

*Message #6 Posted by [Marcus von Cube](#) on 18 May 2005, 8:36 a.m.,  
in response to message #5 by J-F Garnier*

The probably fastest program, following this approach, is to just enter the solution and print it out ;-)

The problem is that all these simplifications will only work for 10 digit numbers. If you test for less digits, you'll have to test many more candidates.

My idea is, not build the numbers with rules like the ones mentioned until now and check for divisibility, but to go the other way round:

- 1) Start with the left most digit, setting it to 1.
- 2) Find a starting point for the nth digit by adding a 0 to the right of the number found so far, dividing the result by n (integer divide), adding 1 and multiplying again by n. This gives the first number to test and should preserve the digits to the left.
- 3) If the number satisfies the condition(s) (i. e. has no duplicate digits for problems (1) or (2)), increase the divisor n and goto step 2. The algorithm has found a solution, if n is big enough. It stops, if we get an overflow with n=1 (the left most digit was already 9).
- 4) Try another number by adding the divisor n. If there is no overflow to the digit to the left, repeat step 3.
- 5) If adding n to the last digit would cause an overflow, we must decrease n, drop the last digit and go back to step 4.

I haven't implemented the algorithm so far but I feel that can be done on a 41 or 42S or even a smaller device (a 16C in integer mode?)

### **Re: Short & Sweet Math Challenge #9: Divisibility !**

*Message #7 Posted by [Eamonn](#) on 19 May 2005, 7:06 p.m.,*



*in response to message #3 by Eamonn*

I finally had time to look into a solution for parts 2 and 3 of S&SMC9. Initially I was hoping to write a program for the HP-32S. I could easily modify my previous program to count the number of solutions for part 2 and part 3 that there are for a single n-digit number. This program could then be run for each value of n.

To count the number of solutions for a n-digit number, the program generates all the solutions for n-1, n-2, ... digits etc. To calculate the number of solutions for 3 digits, the program calculates all the 2-digit and 1-digit solutions. To calculate the number of solutions for 4 digits, all the work that was previously done to count the number of 3-digit solutions must be re-done. Not very efficient.

Ideally, the program would just count the number of solutions for n=10, and store the number of solutions found for n=9, 8, etc along the way. However, the HP-32S doesn't have enough memory to store both the program and the counts of the solutions.

I decided to go with a calculator with more memory. Unfortunately, I don't have in my possession a HP calculator with more memory than the HP-32S. So, I went ahead and wrote a program in Basic for the Sharp PC-1262. It's not a HP, and some would say that it's more a pocket computer than a calculator, but the latter could also be said of the HP-71B.

Anyway, here is my solution for parts (2) and (3) of the challenge. For part (2), run the program as-is. For part (3), change the first statement on line 10 to V=0. The main optimization is to only consider even-valued numbers for the even digit positions, the values 0 and 5 for the fifth digit position and the value 0 for the tenth digit position. When the program finishes, the number of solutions for a n-digit number can be found by looking at C(n).

```

10 V = 1: I = 2: DIM C(10): DIM L(10): DIM Z(10): DIM F(10): FOR A = 1 TO 10: READ Z(A):NEXT A
20 FOR A = 1 TO 9: N=A: F(A) = V: C(1) = C(1) + 1: GOSUB 30: F(A) = 0: NEXT A: END
30 L(I) = 0: N = N * 10
35 IF F(L(I)) = 1 THEN 70
40 Y = N / I: IF Y <> INT(Y) THEN 70
50 C(I) = C(I) + 1: IF I = 10 THEN 70
60 F(L(I)) = V: I = I + 1: GOSUB 30: I = I - 1: F(L(I)) = 0
70 L(I) = L(I) + Z(I): IF L(I) < 10 LET N = N + Z(I): GOTO 35
80 N = INT(N/10): RETURN
90 DATA 1, 2, 1, 2, 5, 2, 1, 2, 1, 10

```

It takes about 15 minutes to find the following solutions for part (2)

#digits	#solutions
1 ----	9
2 ----	41
3 ----	115
4 ----	168
5 ----	220
6 ----	88
7 ----	51

```

8 ---- 9
9 ---- 4
10 ---- 1

```

It takes about 4 hours to find the following solutions for part (3)

```

#digits #solutions
1 ---- 9
2 ---- 45
3 ---- 150
4 ---- 375
5 ---- 750
6 ---- 1200
7 ---- 1713
8 ---- 2227
9 ---- 2492
10 ---- 2492


```

## Re: Short & Sweet Math Challenge #9: Divisibility !

Message #8 Posted by [Valentin Albillo](#) on 20 May 2005, 4:56 a.m.,  
in response to message #7 by Eamonn

Hi, Eamonn !

My most sincere congratulations, your answers to S&SMC#9, all three variants, are absolutely \*correct\*, and with very good timings as well, for the models used. Which is more, you solved (1) in such a RAM-challenged calculator as the HP32S and (2)-(3) in a BASIC dialect that doesn't allow for recursion, which is also quite bold on your part.

The [SHARP PC-1262](#) you used is a really, really, really wonderful machine, absolutely comparable if not superior to anything HP had to offer at the time, and even today. Physically, it was very similar (an upgrade, actually) to the SHARP PC-1260, this is one of mine (actual size is more or less like an HP-15C): 

Just a small comment on your BASIC version. Lines such as

```
DIM C(10): DIM L(10): DIM Z(10): DIM F(10)
```

can be entered instead as

```
DIM C(10),L(10),Z(10),F(10)
```

which is more readable, as well as shorter and faster. Also, lines such as

N=A: F(A) = V: C(1) = C(1) + 1

can be entered instead as

N=A, F(A) = V, C(1) = C(1) + 1

which is faster, and probably more readable as well.

Since it seems no more people are taking the challenge (too difficult, perhaps ?) I'll give my solutions soon.

Thank you very much for your continued interest \*AND\* superb contributions and

Best regards from V.

*Edited: 20 May 2005, 5:11 a.m.*

### **Re: Short & Sweet Math Challenge #9: Divisibility !**

*Message #9 Posted by [Eamonn](#) on 21 May 2005, 1:39 p.m.,  
in response to message #8 by Valentin Albillo*

Hi Valentin,

Thanks for the feedback and the Basic programming tips - I only recently purchased the PC-1262, without manual, and it is indeed a very nice machine with really great capabilities.

This was a good challenge on which to try out Sharp Basic programming. It certainly was a lot faster to develop on the Sharp than on the HP, although that may be partly because I had already done most of the work already on the HP.

One thing I like about your challenges is that they are very well designed. It's always interesting to see the approaches everyone takes to solving them and to see just what can be done on a vintage machine with limited resources. Looking forward to the next challenge.

Best Regards,

Eamonn.

### **Re: Short & Sweet Math Challenge #9: Divisibility !**

*Message #10 Posted by [Valentin Albillo](#) on 23 May 2005, 9:18 a.m.,  
in response to message #9 by Eamonn*

Hi, Eamonn:

Eamonn posted:

*"I only recently purchased the PC-1262, without manual, and it is indeed a very nice machine with really great capabilities."*

Yes, indeed, one of the very best in its class. As for the manual, its BASIC syntax and capabilities are very much like other SHARP models, so you can use their manuals profitably with it.

*"This was a good challenge on which to try out Sharp Basic programming. It certainly was a lot faster to develop on the Sharp than on the HP, although that may be partly because I had already done most of the work already on the HP."*

Among the many HP and SHARP models I've tried, the fastest one for program development was the [SHARP PC-1350 or -1360](#), thanks to its large display (4 lines x 24 characters). You can develop complicated programs starting from absolute scratch without ever committing a single line of code to paper.

*"One thing I like about your challenges is that they are very well designed. It's always interesting to see the approaches everyone takes to solving them and to see just what can be done on a vintage machine with limited resources. Looking forward to the next challenge."*

Thank you very much for your interest, I'm glad you enjoyed the challenge, I was also quite happy with your successful efforts to solve it. Next challenge in a couple of weeks or so.

Best regards from V.

### **Negligible suggestion**

Message #11 Posted by [Andrés C. Rodríguez \(Argentina\)](#) on 17 May 2005, 10:46 p.m.,  
in response to message #1 by Valentin Albillo

Valentin: while my current work schedule prevents me from accepting your (once again) excellent challenges (at least when there is no weekend included in the allowed time interval), I would just like to point that (despite the manner the song title appears in many Internet sites) I recall it as "Miss Lilly Higgins sings shimmy in Mississippi Springs"; a place in the real USA, indeed.

You may also like the sequel "Doctor Bob Gordon shops hot dogs from Boston"; also from Les Luthiers, of course.

Best regards, and thank you for the challenges!

**Re: Negligible suggestion**

Message #12 Posted by **Valentin Albillo** on 18 May 2005, 4:47 a.m.,  
in response to message #11 by **Andrés C. Rodríguez** (Argentina)

Hi, Andrés ! :-)

Andrés posted:

*"... there is no weekend included in the allowed time interval"*

You have a point there. Certainly, posting the S&SMCs on Fridays, say, would allow interested people to tackle them at leisure during the weekend, when there's more free time and the mind is (relatively) free from the week's work chores. So I'll proceed that way from now on, thanks a lot for pointing this out.

*"I would just like to point that (despite the manner the song title appears in many Internet sites) I recall it as "Miss Lilly Higgins sings shimmy in Mississippi Springs"; a place in the real USA, indeed."*

Regrettably, I don't have the CD at hand to look the exact name of the song appearing in there, but I'm almost sure it's like I posted. I think your version makes more sense, at least to me, but if they called their song that way, so be it.

*"You may also like the sequel "Doctor Bob Gordon shops hot dogs from Boston"; also from Les Luthiers, of course."*

And, which of course, is a "foxtrot" ! (what else ?) :-) Not to mention their *"Papa Garland had a hat and a jazz band and a mat and a black fat cat"*, which is also priceless.

I do have their full discography and 'videography' in CD and DVD and if you're ready for an extra thrill, do watch any of their shows in DVD *with the English subtitles on!!*. Seeing the tremendous, heroic efforts to try and translate their extremely humoristic word plays and innuendos into English is fun beyond belief, specially as the translator mostly succeeds, against all odds !. Certainly, "Les Luthiers" are a real, real asset to your marvelous people and country.

Again, thanks for your interest and

Best regards from V.

**Teen's solution (Re: Short & Sweet Math Challenge #9: Divisibility !)**

Message #13 Posted by **GWB** on 18 May 2005, 8:53 p.m.,  
in response to message #1 by **Valentin Albillo**

Gentlemen,

I've just found this (hand) solution by a 13-year old schoolboy from New Zealand:

[http://www.nrich.maths.org.uk/public/viewer.php?obj\\_id=796&part=solution&refpage=viewer.php](http://www.nrich.maths.org.uk/public/viewer.php?obj_id=796&part=solution&refpage=viewer.php)

Regards,

Gerson (43)

*Edited: 18 May 2005, 8:59 p.m.*

### The Basket Case

*Message #14 Posted by **whuy** on 19 May 2005, 7:52 a.m.,  
in response to message #13 by GWB*

That same site has a nice problem too, that can serve as another SSMC: The Basket Case [http://www.nrich.maths.org.uk/public/viewer.php?obj\\_id=1137&part=index&refpage=monthindex.php](http://www.nrich.maths.org.uk/public/viewer.php?obj_id=1137&part=index&refpage=monthindex.php)

A girl buys 4 items, but mistakenly multiplies their prices instead of adding them up. She arrives at a total of \$7.11, which turns out to be the correct total. What were the respective prices of the 4 items?

I've been able to solve it using a 48/49 program.. no doubt Valentin can write it in 2 or 3 lines on his 71B ;-)

Cheers, Werner

### Re: The Basket Case

*Message #15 Posted by **Valentin Albillo** on 19 May 2005, 11:34 a.m.,  
in response to message #14 by whuy*

Hi, Werner:

Werner posted:

*"A girl buys 4 items, but mistakenly multiplies their prices instead of adding them up. She arrives at a total of \$7.11, which turns out to be the correct total. What were the respective prices of the 4 items? [...] I've been able to solve it using a 48/49 program.. no doubt Valentin can write it in 2 or 3 lines on his 71B ;-)"*

With this kind of 'calculator challenges' one has to strive for a proper compromise between these two extremes:

- on the one hand, one does think very little about the problem and opts instead for a pure brute force approach. The resulting program is very short but extremely 'brute' and runs for ages before finding any solutions. No good.
- on the other hand, one does think a whole lot about the problem, and after a long mental effort one manages to reduce the search to much fewer cases, so the resulting program incorporates all this hardly-won knowledge, and does very little search. Why, the \*user\* did the search for the program, so it might run much faster but at the expense of lots of user's time and effort. No good, either.

I advocate that the proper compromise between these two extremes is to give the problem a little think, but not too much, just the obvious elimination of provably impossible cases and the like, then write a program that will actually do the search, as it was supposed to do to begin with.

With respect to your problem, and accordingly to this challenge-solving philosophy, I considered the problem for a few minutes, and extracted the following fast guidelines:

- The problem deals with prices in cents, so all numbers appearing here are in multiples of 0.01. It's best to get rid of those fixed-point decimals and use integers instead. This means the program will search for numbers adding up to 711 and whose product is  $711 * 100 * 100 * 100$ . Once a solution is found, we'll simply scale down its numbers by 100.
- If integer numbers must have a product equal to 711000000, at least one of them must be equal to this product's largest prime divisor or a multiple thereof. For this number (711), a very quick test reveals that its largest prime factor is 79. So one of the (scaled-to-integer) prices, say A, must be either 79 or a multiple of 79.
- The remaining prices, B, C, and D, are bound by simple limits regarding to their sum and/or their divisibility of the product, which cost nothing to take into account.

I then purposely refrained from spending any more \*manual\* effort on my part, and wrote instead a short, simple program incorporating those quick discoveries, namely this 5-liner for the bare-bones HP-71B, which took me just 5 minutes to write and test:

```

1  K=711 @ D=79 @ M=100 @ R=K*M^3 @ S=R/D @ FOR A=D TO K-D STEP D
2  FOR B=1 TO (K-A)/3 @ IF MOD(S,B) THEN 5 ELSE T=A+B @ F=A*B
3  FOR C=B TO (K-T)/2 @ IF F*C*(K-T-C)=R THEN DISP A/M;B/M;C/M;(K-T-C)/M @ END
4  NEXT C
5  NEXT B @ NEXT A @ DISP "Not found"

```

Upon running, it displays:

```
>RUN
```

```
3.16 1.2 1.25 1.5
```

which is a correct solution as both the sum and the product of these quantities do equal 7.11.

The running time is some 30 seconds under [Emu71](#) @ 1.4 Ghz, and thus should be some 10-15 minutes in a physical 71B.

Best regards from V.

P.S.: Should you *insist* in your "no doubt Valentin can write it in 2 or 3 lines on his 71B ;-)" remark, this version

```
1 K=711 @ D=79 @ M=100 @ R=K*M^3 @ S=R/D @ FOR A=D TO K-D STEP D @ FOR B=1 TO (K-A)/3
2 T=A+B @ F=A*B @ FOR C=B TO (K-T)/2 @ IF F*C*(K-T-C)=R THEN DISP A/M;B/M;C/M;(K-T-C)/M @ END
3 NEXT C @ NEXT B @ NEXT A @ DISP "Not found"
```

does it in just 3 lines by omitting a single divisibility criterium, but it takes 3-4 times as long to find the solution. Compromise, compromise !

*Edited to remove a couple of typos*

*Edited: 19 May 2005, 12:19 p.m.*

## Re: The Basket Case

Message #16 Posted by [why](#) on 19 May 2005, 3:15 p.m.,  
in response to message #15 by Valentin Albillo

Hi, Valentin! It is the only solution, as well. I took a slightly different (worse..) approach ,but managed to find it in 'reasonable time' as well. Hope you didn't spoil the fun for everyone else, posting an excellent solution so quickly. The one thing that aggravates me about the programs you post for the 71B is the fact that you quit a loop early in just about each one of them - and how easy it is in the 71B. And what a mess it is on the 48/49..

Cheers, Werner

## Re: The Basket Case

Message #17 Posted by [Valentin Albillo](#) on 20 May 2005, 4:43 a.m.,  
in response to message #16 by why

Hi, Werner ! :

Werner posted:

*"I took a slightly different (worse..) approach ,but managed to find it in 'reasonable time' as well."*



Why don't you post it ? I always have a lot of fun looking at those unfathomable RPL listings ... ;-)

*"Hope you didn't spoil the fun for everyone else, posting an excellent solution so quickly."*

Well, thanks for your kind comment but you were posting a *\*different\** challenge of yours in *\*my\** original challenge's thread, instead of posting anything related to it, and that's pretty unpolite to begin with. If you don't want your new challenge "spoiled", go and create your own thread ! :-)

*"The one thing that aggravates me about the programs you post for the 71B is the fact that you quit a loop early in just about each one of them - and how easy it is in the 71B. And what a mess it is on the 48/49.."*

*\*Everything\** is a mess in RPL. Where you had the simplicity and sheer joy of keystroke programming, you've got a "structured", high-level language that will force you to do things *\*its\** way instead of allowing you to do things *\*your\** way, that you'll never entirely master and that will have you spend inordinate amounts of time to try and remember how to do the simplest things. Where you had the simplicity and extreme readability of BASIC, you've got an unfathomable bunch on nearly incomprehensible, ghastly-looking commands and statements that no one, not even the original programmer, can understand later except by careful and painful analysis, if at all ...

And yes, you'll be amazed to see what can be done on a 71B in just 3-4 lines of code, and how easy and natural it all is. If in doubt, have a look at my "[Baker's Dozen](#)" articles in Datafile, specially the second one (Vol. 2, alas not yet online, you'll need to get the corresponding [Datafile issue](#)).

Thanks for your interesting challenge and comments, and

Best regards from V.

### **Bad Languages (OT)**

Message #18 Posted by [Thomas Okken](#) on 20 May 2005, 7:06 a.m.,  
in response to message #17 by Valentin Albillo

*\*Everything\* is a mess in RPL.*

Hear, hear!

Once we're done hunting down and roasting over a slow fire the persons responsible for RPL, let's go after the idiot who decided that Java would not have **goto**. ;-)

Long ago, I did a bit of PostScript programming. The result was almost as psychedelic as Lisp. If programming is all about the satisfaction of getting complex and difficult work done, these weird languages are a godsend, because they make *\*everything\** difficult...

- Thomas

### Re: Bad Languages (OT)

Message #19 Posted by **Valentin Albillo** on 20 May 2005, 7:42 a.m.,  
in response to message #18 by Thomas Okken

Hi, Thomas:

Thomas posted:

*"Once we're done hunting down and roasting over a slow fire the persons responsible for RPL, let's go after the idiot who decided that Java would not have goto. ;-)"*

Agreed. Look for some Wickes guy for the first abomination, don't know about the Java culprits.

*"If programming is all about the satisfaction of getting complex and difficult work done, these weird languages are a godsend, because they make *\*everything\** difficult..."*

Agreed. And the saddest thing of all is that, in the end, all those fancy, goto-forbidding languages are ultimately translated into machine-language programs, *where statistics say that 30-35% of all instructions are branching instructions, aka **goto's** !*

I'd like to see all those nerdy goto-haters try and write a compiler which does its job without ever trashing its style by using and generating goto's for the compiled code. :-)

Best regards from V.

P.S.: Also, I happen to remember that in the realm of HP-41C synthetic programming, two of the most powerful synthetic instructions were the pair **STO b/RCL b**. You could do just about *everything* by cleverly using them, from creating any synthetic instruction or text as a program line, to byte-count a program, to create assignments, to defeat PRIVATE, you name it !

And guess what ? STO b **IS** a GOTO !

### \*Good\* programming languages (S-OT)

Message #20 Posted by **John L. Shelton** on 20 May 2005, 2:21 p.m.,  
in response to message #19 by Valentin Albillo

S-OT = "Still off-topic"

It's a mistake to think that because a compiler generates code of quality "X", that humans should write using the same quality "X". Otherwise, why use compilers at all? The point to a compiler is to allow us to write programs in something "expressive", better fitting our minds or the problem domain. Computers execute machine language designed to be efficient and general purpose.

There are many studies, some classic, demonstrating why the use of "goto" leads to more errors in human-written programs. In 25 years of programming in Lisp, Smalltalk, and (yuk) Java, I haven't missed "goto".

What's a real shame is that "modern" languages like Java have been only small improvements over earlier languages, in terms of productivity. Java, according to several studies, is about 8 times as productive as writing in macro-assembly language. We ask our programmers today to do about 20 times as much functionality as we did a generation ago, and wonder why projects still take years. We need languages that are 100 times as productive (or customers with simpler project requirements.)

You are welcome to write machine-efficient code, for fun or profit. The skills required should not be forgotten - indeed, we need these skills inside new compilers. It's also fun, for nostalgia reasons, to program in primitive environments. I enjoy memories from the 1960s and 1970s, dealing with very small memory, very slow CPUs, and having to fine-tune things on paper before having a few seconds of time on the shared computer.

But for most modern programming today, I want very powerful languages that minimize human error, and good compilers to give me efficiency. But I value correctness and productivity much more than efficiency; I can buy more computer horsepower much more easily than buying developer time.

Some day, our computers will interview our customers, guided by good consultants, and the problem will be modeled correctly in the computer, with automatically generated programs appearing daily. As the customer plays with the incremental solutions, commenting on their merits and drawbacks, the consultant and computer will collaborate to generate the next versions.

### **Re: \*Good\* programming languages (S-OT)**

*Message #21 Posted by [Thomas Okken](#) on 24 May 2005, 7:48 a.m.,  
in response to message #20 by John L. Shelton*

*There are many studies, some classic, demonstrating why the use of "goto" leads to more errors in human-written programs. In 25 years of programming in Lisp, Smalltalk, and (yuk) Java, I haven't missed "goto".*

I suspect that those studies find what they want to find... Personally, I can get things done in Java, but I enjoy being able to use goto in C/C++, because it allows me to write code like [this](#).

I'm sure some would consider this example to be a good case *\*against\** the use of goto, but I find this kind of code easier to write *\*and\** easier to read -- if you have to do this type of function without goto, you end up having to break the function into smaller ones, and/or introduce state variables and use lots of if statements, all of which makes the flow of control harder to follow.

Just my \$0.02!

- Thomas

## Re: *\*Good\** programming languages (S-OT)

Message #22 Posted by *Valentin Albillo* on 24 May 2005, 9:13 a.m.,  
in response to message #21 by Thomas Okken

Hi, Thomas:

Thomas posted:

*"I suspect that those studies find what they want to find..."*

Let's see. What follows is a small sample of random Intel assembler source code obtained from a random search using Google:

```

start:  mov ax, @data
        mov ds, ax
        mov ah, 9h           ;print help message
        mov dx, offset help
        int 21
hlp1:   mov ah, 06h          ; read character from keyboard
        mov dl, 0ffh
        int 21h
        jz lp1              ; repeat if character not ready
        cmp al, 00h         ; if function key then exit
        je exit
        cmp al, 32          ; else if control code
        jae disp1
        call ctrl_code      ; then process control code
        jmp lp1
disp1:  push bx
        xor bx, bx          ; page zero on video memory
        mov bl, [attrib]    ; get character attribute
        mov cx, 1           ; one character to write
        mov ah, 9           ; write char + attribute
        int 10h            ; use BIOS call
        call bumpcur        ; next cursor position

```

```

        jmp lp1                ; repeat
exit:   mov ax, 4c00h
        int 21h
END     start

```

I count 25 instructions (lines), and unless I'm mistaken, there are no less than 11 branching instructions that divert execution flow elsewhere. That's 44% for this particular, absolutely random sample. Even if you only want to count those branching instructions that do not return, there are still 5 of them (20%) in this very small piece of code.

"I suspect that you find unworthy those studies that do not find what you want to find..." :-)

Best regards from V.

### Re: **\*Good\*** programming languages (S-OT)

*Message #23 Posted by . on 24 May 2005, 6:45 p.m.,  
in response to message #22 by Valentin Albillo*

Yes, but 99% of people don't write in assembler. Those comments apply to structured programming languages.

### Re: **\*Good\*** programming languages (S-OT)

*Message #24 Posted by **Gunnar Degnbol** on 25 May 2005, 4:51 p.m.,  
in response to message #21 by Thomas Okken*

Quote:

I enjoy being able to use goto in C/C++, because it allows me to write code like [this](#). I'm sure some would consider this example to be a good case *\*against\** the use of goto

Yes. The first goto in your example can be rewritten as:

```

        *dst_exp = src_exp;
} else {
    if (src_exp > *dst_exp) {
        // etc.
        *dst_exp = src_exp;
        /* Now that dst is aligned with src, proceed using the code

```

```

    * for the dst_exp == src_exp case.
    */
    // goto add_matching;
}
if (src_exp == *dst_exp) {
    int carry;
    // add_matching:
    carry = 0;
    // etc.
} else /* src_exp < *dst_exp */ {
    int carry;

```

The second goto becomes:

```

if (d <= 34359738367.0 && d >= -34359738368.0) {
    n = (int8) d;
    // etc.
    return chars_so_far;
}
if (base_mode != 2) {
    string2buf(buf, buflen, &chars_so_far, "<Too Big>", 9);
    return chars_so_far;
}

```

Quote:

---

but I find this kind of code easier to write \*and\* easier to read

---

No. In both cases the logic in the goto-less version is much clearer. If you did not have goto, you would have been forced to think clearly.

Quote:

---

if you have to do this type of function without goto, you end up having to break the function into smaller ones, and/or introduce state variables and use lots of if statements, all of which makes the flow of control harder to follow.

---

Such cases exist, but they are rare. It is possible some of the other gotos in the example are in this group, but I doubt it. I have similar logic in [Stak](#), my RPN calculator for phones (and PCs), written in Java, and I never missed goto.

The most common situation where the alternative to goto is state variables is for breaking out of multiple loops. Java has [labelled break](#) to deal with that.

I find it odd that Valentin tries to [defend](#) goto using the prevalence of jumps in assembler code, while [simultaneously](#) extolling the readability of high-level HP 71B Basic over RPN code. goto-less Java code is of course compiled to bytecode full of jumps, but so what?

### Re: **\*Good\*** programming languages (S-OT)

Message #25 Posted by [Valentin Albillo](#) on 26 May 2005, 5:02 a.m.,  
in response to message #24 by Gunnar Degnbol

Hi, Gunnar:

Gunnar posted:

*"I find it odd that Valentin tries to defend goto using the prevalence of jumps in assembler code, while simultaneously extolling the readability of high-level HP 71B Basic over RPN code. goto-less Java code is of course compiled to bytecode full of jumps, but so what?"*

It's only that I find it *highly hypocritical* to generate "compiled bytecode full of jumps" because that's obviously more efficient, while making it **\*impossible\*** for the professional programmer to use them because of some alleged higher 'purity' or 'clarity' of the source code (the resulting object code will be a mess of jumps anyway).

I'm a professional programmer and I absolutely resist being **\*forced\*** to use or not use some natural feature, based on some people's haughty ideas. I don't want anyone *deciding for me*, from their purist committee, based on theoretical guidelines, whether this or that feature would make the code clearer, faster, or more efficient.

I *am* the one writing that *particular* code for that very *specific* situation and so I'm the best qualified person to decide whether it would be convenient to do so or not, that's a decision for *me* to make and I think it's utterly preposterous and unacceptable to be **\*forced\*** to follow arguable theoretical considerations regardless of the situation at hand.

Make the feature available, even if don't recommending its use, and I'll probably *won't* use it 99.999% of the time. But for the 0.001% of situations where it would be convenient or even critical to use it, I don't want to go through all kinds of hoops and loops because someone decreed so.

Best regards from V.

**Re: \*Good\* programming languages (S-OT)**

*Message #26 Posted by . on 26 May 2005, 6:23 a.m.,  
in response to message #25 by Valentin Albillo*

"It's only that I find it highly hypocritical to generate "compiled bytecode full of jumps" because that's obviously more efficient, while making it *\*impossible\** for the professional programmer to use them because of some alleged higher 'purity' or 'clarity' of the source code (the resulting object code will be a mess of jumps anyway)."

I think you are still missing the point. Yes, native code or bytecode contains tons of gotos (or branches, or jumps, or whatever you want to call them). That's probably because no-one has made a CPU with WHILE, FOR, etc instructions in hardware. Of course GOTO's are fast on that level.

But how on earth can you possibly compare the output of a language (which 99.99% of people don't look at anyway) to the language itself? Almost no-one codes in assembly these days because high-level languages are far more productive. Programmers look at the high level source constantly.

If I write

```
while (a!=b)a++;
```

Does it matter that the resulting assembly code contains GOTO's? No. I, the programmer, do not have to look at them. Just like I don't need to know how the CPU ALU functions, or how the transistors work, that level of detail is abstracted away.

Have you ever tried working on a serious software project full of GOTO's? I have. The previous programmer was an idiot and abused the GOTO operation. The resulting mess was very difficult to maintain. If the GOTO operation had been restricted he would have been forced to use some structure instead. The resulting code would still have been ugly, but at least have had some order.

Minimizing the number of GOTOs goes a long way to improve maintainability. Eliminating them completely is rather extreme but I'd rather that than deal with such horrible spaghetti code again.

However comparing the compiled binary to high level language constructs is completely missing the point.



**Re: \*Good\* programming languages (S-OT)**

Message #27 Posted by **Wayne Brown** on 26 May 2005, 10:46 a.m.,  
in response to message #26 by .

Quote:

---

Have you ever tried working on a serious software project full of GOTO's? I have. The previous programmer was an idiot and abused the GOTO operation. The resulting mess was very difficult to maintain. If the GOTO operation had been restricted he would have been forced to use some structure instead. The resulting code would still have been ugly, but at least have had some order.

---

But that was the fault of the programmer, not the language. I agree with Valentin: The language should make *anything* possible, and rely on the programmer to exercise self-restraint and use good programming practices, rather than trying to *force* the programmer into making the "right" choices. That's what I love about the philosophy behind UNIX and (the early versions of) the C programming language: The availability of power tools shouldn't be restricted just because some idiots will use them to cut off their fingers.

**Re: \*Good\* programming languages (S-OT)**

Message #28 Posted by **John L. Shelton** on 26 May 2005, 1:38 p.m.,  
in response to message #27 by Wayne Brown

I beg to differ with several points:

- (1) A language should *\*not\** cater to all styles of programming. Else we wind up with bloated languages, like Ada (anyone besides me remember that fiasco?) A programming language should have one or two simple paradigms; if a user doesn't like it, move on to another language. We have thousands of computer programming languages, so we don't need to build a new one that supports *\*all\** styles. After all, C doesn't support the object programming paradigm (natively.)
- (2) In most commercial environments, the programmer that develops something is not the same as the maintainer of the code. Countless projects have been ruined by genius programmers who left to go on to another exciting project, leaving unmaintainable code for the next person to understand. Unless you can commit to maintaining code forever, you *\*don't\** win points saying as developer you should set the standards.
- (3) Some languages consider long-term maintainability, and through fiat or example, show the developer how to write code that is easily maintained. Of course, "clever" developers can defeat this. Java was designed to

support simple object programming, with a typical application having dozens of classes and hundreds of methods. By design, maintenance of a Java program should be easy: each method is a page or less of code, including documentation, that a future developer can read and understand. But I've seen plenty of C programmers twist Java, claiming efficiency, by coding ONE class and ONE method ("main()") and utterly defeating the purpose of maintainability.

Gee, this is fun to discuss. Applicability to the HP Calculator Museum? Well, I guess we need to understand all the issues when writing calculator programs: efficiency as well as maintainability. The earlier programmable calcs encouraged efficiency since the resources (time, space) were very constrained. Later calcs gave us much more expressive power, more memory, and faster processors, allowing us to write programs more easily, and ones that could be more easily understood by others. But not everyone has taken advantage of these advances.\

### **Re: \*Good\* programming languages (S-OT)**

*Message #29 Posted by [Eric Smith](#) on 26 May 2005, 3:59 p.m.,  
in response to message #28 by John L. Shelton*

Quote:

\_\_\_\_\_

A language should *\*not\** cater to all styles of programming. Else we wind up with bloated languages,

\_\_\_\_\_

I agree with you on that.

Quote:

\_\_\_\_\_

Else we wind up with bloated languages, like Ada (anyone besides me remember that fiasco?)

\_\_\_\_\_

I strongly disagree with that. Ada is not bloated, nor does it cater to all styles of programming. People confuse the fact that the Ada Language Reference Manual was deliberately written to try to cover all the details and corner cases with the idea that it's a large language. But C++ is a much more bloated and inconsistent language than Ada. And the current ISO C standard is *\*larger\** than the Ada standard.

A better example of a bloated language that tried to be all things to all people was PL/I. Ada was designed by a small team, but PL/I was designed by a committee that tried to mash together Fortran, COBOL, RPG, and Algol into one language with additional stuff thrown in to boot. Very few compilers for the full PL/I language were ever written; most implement only a subset.

### Re: **\*Good\* programming languages (S-OT)**

Message #30 Posted by **Wayne Brown** on 27 May 2005, 1:05 p.m.,  
in response to message #28 by John L. Shelton

Quote:

---

(1) A language should *\*not\** cater to all styles of programming. Else we wind up with bloated languages, like Ada (anyone besides me remember that fiasco?) A programming language should have one or two simple paradigms; if a user doesn't like it, move on to another language. We have thousands of computer programming languages, so we don't need to build a new one that supports *\*all\** styles. After all, C doesn't support the object programming paradigm (natively.)

---

I don't think languages should *cater* to all styles, but neither should they try to render certain styles impossible. Plain C doesn't encourage object-oriented programming, but it doesn't throw up deliberate roadblocks to stop it either. C is flexible enough to be twisted into some pretty odd shapes by a determined-enough programmer (witness the "Obfuscated C" contests!). For example, I once worked with a programmer who was enamored of Pascal. His C programs were full of weird things like **#define** statements to let him use **begin** and **end** rather than { and }. I didn't think his programs were very impressive as either C *or* Pascal examples, but I thought it was neat that he could even attempt such a thing.

The place where I work now had a good deal of code written in COBOL before switching to a language called Natural. Before long we had a fair number of programs written in what some of us call NATBOL, which is a Natural program written by a former COBOL programmer to look as much like COBOL as possible. Now we're in the middle of switching to Oracle and PL/SQL, so I'm sure there are some really weird hybrids growing out there somewhere. Fortunately I'm a sysadmin now so I don't have to deal with application code much these days.

Quote:

---

(2) In most commercial environments, the programmer that develops something is not the same as the maintainer of the code. Countless projects have been ruined by genius programmers who left to go on to another exciting project, leaving unmaintainable code for the next person to understand. Unless you can commit to maintaining code forever, you *\*don't\** win points saying as developer you should set the standards.

---

I've had to maintain plenty of horrible code written by other people over the years. Usually, if I could figure out what was going on, I tried to imitate the other person's style so that it was hard to tell (without reading the comments) which code was mine. If it was too convoluted, then I might end up rewriting much (or maybe all) of the program to make it more maintainable (especially if I expected to have to deal with it again myself). It's inconvenient when that happens, but I'd rather deal with that than have to face more restrictions on my own programming choices.

### **Exactly correct**

*Message #31 Posted by **Gene** on 27 May 2005, 2:15 p.m.,  
in response to message #28 by John L. Shelton*

The basic reason for structured programming is maintenance of the code.

When programming for Accenture, structure programming was required by every client because, once we finished the application development, we were gone and they were left with the code.

If one is developing something for oneself, big deal. Developing for others, you do what the others want.

### **Re: Exactly correct**

*Message #32 Posted by **Eric Smith** on 28 May 2005, 1:31 a.m.,  
in response to message #31 by Gene*

Quote:

---

If one is developing something for oneself, big deal.

---

Actually one of the lessons that many experienced programmers have learned is to write and document all code well, even if you think it's just for yourself or "throwaway code". Otherwise, you'll have to fix a problem with it five years later, and wind up cursing the idiot that wrote it.

Another key insight separating experienced programmers from the rest is that you do NOT want to write your code in an exceptionally clever manner, except where absolutely necessary (and then it should be well-documented). Debugging is harder than coding, so if you write code as cleverly as you can, it is guaranteed that you will have trouble debugging and maintaining it. Cleverness often comes back to bite you in various ways. Often clever tricks make code harder to enhance than simple code would. It is very easy to out-clever" yourself.

### **That's one very interesting point...(was: Exactly correct)**

*Message #33 Posted by **Vieira, Luiz C. (Brazil)** on 28 May 2005, 2:15 a.m.,  
in response to message #32 by Eric Smith*

Hi Eric, Gene, guys;

I read Gene's post and now yours, Eric, and I feel that many programmers are not aware of what you wrote here. I myself try the best I can to enhance performance, reduce memory use (both and environment) and keep an understanding code. You wrote something that actually meets my own thoughts:

Quote:

---

(...)separating experienced programmers from the rest is that you do NOT want to write your code in an exceptionally clever manner, except where absolutely necessary (and then it should be well-documented). Debugging is harder than coding, so if you write code as cleverly as you can, it is guaranteed that you will have trouble debugging and maintaining it. Cleverness often comes back to bite you in various ways.

---

I sometimes find some earlier programs of mine, mainly HP41 related RPN codes, that I simply have no clue about why did I use this or that sequence. As mentioned by Sheldon:

Quote:

---

The earlier programmable calcs encouraged efficiency since the resources (time, space) were very constrained. Later calcs gave us much more expressive power, more memory, and faster processors, allowing us to write programs more easily, and

ones that could be more easily understood by others. But not everyone has taken advantage of these advances.

---

I'm not a programmer as for a professional activity, I'm an Electrical Engineer, and I took programming as a professional tool when it was necessary and/or advantageous. And I learned a lot, but I am completely aware of the fact that in many circumstances, a true programmer would go far beyond my imagination and skills.

My own 'efficiency measurement' when programming is the time and effort taken to finish and/or refine a program. If time spent goes beyond my expectations and availability, I either cut refinement off or either stop and try another approach. But this is my own way of doing things. I think that software engineers, along with experienced programmers, are the ones with the correct tools to deal with these variables when writing a code. I think that for those with programming skills and no specific development technique, programming relates to knowledge as well as to inspiration and art. And if these parameters are not well balanced, final code may contain some awkward sequences and 'manuvres' that would be hard to understand and mainly to update or change. I think that's what happens to my early programs...

I'd like to enhance my ability to program. Better yet, I'd like to know if I actually know how to program or if I am just able to connect commands, functions, operators and variables to predefined structure blocks in a reasonable sequence so I execute it, check for errors, correct it and close this loop till it works. In my point of view, this is not exactly what a real programmer should do. Imagine an engineer trying to design a building and, after some try-outs, the last 'version' does not fall apart...

O.K., O.K., too exaggerated, I agree with you... but just give the reasoning a try.

Luiz (Brazil)

### **Re: That's one very interesting point...(was: Exactly correct)**

*Message #34 Posted by [Eric Smith](#) on 28 May 2005, 1:47 p.m.,  
in response to message #33 by [Vieira, Luiz C. \(Brazil\)](#)*

Quote:

---

I'd like to know if I actually know how to program or if I am just able to connect commands, functions, operators and variables to predefined structure blocks in a reasonable sequence so I execute it, check for errors, correct it and close this loop till it works.

---

That's what almost all programmers do.

Quote:

---

In my point of view, this is not exactly what a real programmer should do. Imagine an engineer trying to design a building and, after some try-outs, the last 'version' does not fall apart...

---

What you're looking for is "Software Engineering". However, the vast majority of people who call themselves Software Engineers are not, by any reasonable definition of engineering.

The idea of engineering is to have documented and proven methods and practices, and to employ them consistently to produce products (i.e., software), rather than using ad-hoc solutions. Most programming consists of throwing code at the wall and seeing what sticks.

There are various attempts to bring proper engineering methodology to software. For instance, UML and "design patterns". But such practices are not widely used. Where they are used, they're often not used consistently. and even when they're used consistently, they're not adequate to bring software engineering to the same level of "engineeringness" as other engineering disciplines.

I am not personally a fan of either UML or design patterns, though I at least appreciate the effort to bring some attempt at rigor to the field.

### **Re: Exactly correct**

*Message #35 Posted by **Eric Smith** on 28 May 2005, 1:50 p.m.,  
in response to message #32 by Eric Smith*

I wrote:

Quote:

---

you do NOT want to write your code in an exceptionally clever manner, except where absolutely necessary (and then it should be well-documented)

---

I should clarify that cramming a program into a tiny amount of memory (such as on an older RPN calculator) is exactly the sort of situation where I'd acknowledge that extreme cleverness is sometimes absolutely necessary. But that doesn't waive the documentation requirement.

I *\*don't\** think this generally applies nearly as much to the HP-48/49, as they have plenty of memory.

### **Re: Exactly correct**

*Message #36 Posted by **Les Bell [Sydney]** on 29 May 2005, 7:48 p.m.,  
in response to message #31 by Gene*

Quote:

---

If one is developing something for oneself, big deal. Developing for others, you do what the others want.

---

Tain't necessarily so. As a consultant whose development projects are split up by other assignments, travel, etc. I find it essential to comment stuff liberally so that when I come back to either continue development or perform maintenance, I can quickly figure out what on earth I was thinking. I also make heavy use of CVS, since it lets me work at my desktop machine as well as my laptop while travelling.

In my youth, my first language was Algol, and I hardly even knew of the goto for the first few years. Then, while at university, I was introduced to BASIC for quick'n'dirty engineering work, and it was only after writing and trying to maintain a few unstructured programs that I became a convert to structured programming.

Much later, I had to develop a partial implementation of the dBASE database language interpreter. I did it in PL/I Subset G and forced myself to not use any goto's, use the scoping rules of the language by write short (one-page) procedures in their own files, etc. It took me a day for the database access and b+tree indexing routines, a day for the recursive descent parser and the various language operators and functions, and a final day to glue it all together with miscellaneous other modules. In all, it was



around 80 separate source files, each around one, maybe two pages in length. I turned it over to a junior programmer to test; she found two or three bugs and only one other bug was found after it went into production.

So for me, the payoff from structured programming is more reliable code generated faster.

Of course, the real problem is not the goto statement - it's the lack of the comefrom statement in most languages. Inclusion of the comefrom makes debugging much easier.

;) )

Best,

--- Les

[\[http://www.lesbell.com.au\]](http://www.lesbell.com.au)

### **GOTO's: There is a "third way"**

*Message #37 Posted by **Karl Schneider** on 27 May 2005, 1:07 a.m.,  
in response to message #20 by John L. Shelton*

I have followed this thread about the GOTO statement, with some contributors wanting to retain the function and others lenthing the undisciplined "spaghetti code" that it can lead to.

One way for a software-development organization to utilize programming features in a disciplined manner is to enforce software standards with independent "peer review" of code.

In an organization during the latter 1980's, I was a Fortran programmer on mainframe systems, then later a "software standards enforcer" on Fortran code. There was a strong emphasis at the time for development of structured code and the implementation of software standards to enhance maintainability of the program code.

In our Fortran coding standards, the "GOTO" statement was allowed only in certain instances where Fortran 77 did not provide a "structured" method of achieving the task -- e.g., jumping out of a "DO-loop" before the maximum number of iterations had been reached. (Fortran 77 did not have "DO-until" or "DO-while" constructs, although later versions do, starting with Fortran 90.)

I seem to remember Kernighan and Ritchie recommending against the use of "goto" in C programs except when absolutely necessary, but it was still made available in the 1989 ANSI standard for the C language.

There is merit in the discontinuance of poor constructs when a vastly-better way is provided, even at the expense of easy portability to newer versions. For example, Fortran 77 introduced the CHARACTER type variable, and did not support Fortran 66's archaic "Hollerith literal" method of loading strings into variables of other types.

BTW, I do remember Ada. The US Department of Defense was pushing adoption of Ada in the 1980's as a standard programming language, but I don't think the effort was very successful.

-- KS

*Edited: 27 May 2005, 1:33 a.m.*

### **Re: The Basket Case**

*Message #38 Posted by **Raymond Del Tondo** on 20 May 2005, 6:11 p.m.,  
in response to message #17 by Valentin Albillo*

Hi,

> \*Everything\* is a mess in RPL

>

I totally disagree here!

But I think you meant your statement as a joke:-)

> Where you had the simplicity and extreme readability of BASIC

>

This is considered to be the 2nd part of the kidding statement...

Even simple BASIC programs can be relatively unreadable.

I recently saw such an example in datafile, right in the middle;-)

Someone tried to squish as many as possible statements into one line,  
thus reducing the program size by a few bytes,  
but abandoning readability totally IMHO.

Back then when I programmed the 71B in BASIC,  
I even put local labels in the middle of lines,  
just to save the byte or two required for the line number.

I loved these byte saving techniques a lot,  
and still love to optimize programs in size and speed,  
but I also learned to love structured source text.

You could write a complex RPL program in one single line, too,  
but it could get as unreadable as said BASIC program.

With RPL, you have the choice:  
Writing BASIC-like spaghetti code,  
which will be unreadable to yourself after a while,  
unless you provide a good documentation,  
or writing structured code,  
where you can at least recognize the bigger  
functional blocks even after months.

BTW: In RPL, you don't need a an explicit GOTO statement,  
because you have other nice ways to transform  
program execution to somewhere else.

And RCL b/STO b on the HP-41 are nice and fast GTO alternatives,  
but they're far away from being even considered as clean programming style.  
They can even be dangerous,  
and thus I don't recommend the use of RCL b/STO b,  
especially if you give programs to someone else,  
who doesn't know how to handle RCL b/STO b and alike.

Regards

Raymond

### **Re: The Basket Case**

*Message #39 Posted by [Valentin Albillo](#) on 23 May 2005, 5:31 a.m.,  
in response to message #38 by Raymond Del Tondo*

Hi, Raymond:

Raymond posted:

*"I think you meant your statement as a joke:-)"*

Not exactly. I don't recall adding a smiley after my statement.

*"Even simple BASIC programs can be relatively unreadable. I recently saw such an example in datafile, right in the middle;-). Someone tried to squish as many as possible statements into one line, thus reducing the program size by a few bytes, but abandoning readability totally IMHO."*

I *\*do\** notice your 'sarcastic' emoticon, which I take it to mean you're referring to some program of mine published in Datafile. If that's so, you're absolutely wrong. I do **not** *"squish as many as possible statements into one line"* to reduce *\*program\** size but to reduce *\*article\** size.

**Datafile** is a small-format publication, just 32-48 A5 pages, published every other month, so saving space is *critical*, in my opinion, to try and fit as many interesting articles as possible from as many contributors as possible. Were I to publish my program listings in 'pretty printing' format, with every statement on its own line, indentations and such, it would take up to 5 times more valuable space in Datafile for the same net article content. I would never hog Datafile's space that way, publishing 'pretty looking' listings at the expense of not having space for other people's routines.

As a simple example, my solution to S&SMC#9 could be published there in 4 lines, thus:

```
100 SUB SR(N,L) @ FOR I=MOD(L,2)+4*(L=5) TO 9 STEP 2+3*(L=5)
110 IF FLAG(I) THEN 130 ELSE M=N+I @ IF MOD(M,L) THEN 130
120 SFLAG I @ IF L<9 THEN CALL SR(10*M,L+1) @ CFLAG I ELSE DISP M*10 @ CFLAG I
130 NEXT I
```

or else, 'pretty formatted' this way:

```
100 SUB SR(N,L)
105     FOR I=MOD(L,2)+4*(L=5) TO 9 STEP 2+3*(L=5)
110         IF FLAG(I) THEN
112             GOTO 130
114         ELSE
116             M=N+I
118             IF MOD(M,L) THEN GOTO 130
119         END IF
120         SFLAG I
122         IF L<9 THEN
123             CALL SR(10*M,L+1)
124             CFLAG I
125         ELSE
```

```

126          DISP M*10
127          CFLAG I
128          END IF
130          NEXT I
135  END SUB

```

taking 5 times as much space in the page. When you've got only 32-48 small (A5) pages for *everyone, every two months*, hogging 400% more space is unacceptable, specially when your intended audience (Datafile readers) are *\*very\** skilled people, used to program HP calculators, and who won't have any difficulty whatsoever understanding your listing or even formatting it to their own taste if desired.

You're a Datafile subscriber, I take, so tell me if you'd rather have Datafile space used up to include some interesting RPL routines or else to have instead Mr. Albillo's program for the HP-71B 'pretty-printed', with no statements "squished into one line".

*"Back then when I programmed the 71B in BASIC, I even put local labels in the middle of lines, just to save the byte or two required for the line number."*

Correct me if I'm wrong, but I think that any "local label" (say "PEPE") and its correspondig GOTO (GOTO "PEPE") will waste many more bytes than the two required for the line number, right ?

*"You could write a complex RPL program in one single line, too, but it could get as unreadable as said BASIC program."*

I don't think you're being serious here, or else you're confusing semantic unreadability with syntactic unreadability. Any BASIC code is highly human-readable for English readers because it uses mostly English keywords with an English-like syntax. You can put every word in a single line, or a lot of words on a line, or even suppress spaces between words, yet it will still be mostly readable and understandable, say for instance:

```

FOR Quantity = 10 TO 20
  LET Cost = Price * Quantity
  PRINT Price, Quantity, Cost
NEXT Quantity

```

or even not formatted at all

```

FOR Quantity = 10 TO 20 : LET Cost = Price * Quantity : PRINT Price, Quantity, Cost : NEXT Quantity

```

pose no understanding problems for any English reader, be he/she a BASIC programmer or not. On the other hand, RPL code such as:

```

<< Cucu a 1
rdPOS SUB SWAP 1 \->LIST + a 'rdPOS'
INCR rdSIZ SUB +
'rdNAME' RCL STO
rdINIT \>>

```

is inherently very difficult to fathom, however formatted, for most everyone, including RPL programmers themselves. So I don't think your next paragraph is to be taken seriously at all, but that's your opinion and this is *\*my\** opinion. Let's stop here and avoid flame wars and other childish behaviour, such quasi-religious topics can't be discussed profitably and I know from experience that trying to change other people's deeply-rooted beliefs, however irrational they might be, is doomed to bitter failure.

Best regards from V.

*Edited: 23 May 2005, 5:41 a.m.*

## S&SMC#9: My Solutions and Comments

Message #40 Posted by [Valentin Albillo](#) on 20 May 2005, 6:36 a.m.,  
in response to message #1 by Valentin Albillo

Hi, all:

Thanks to all people interested in my S&SMC#9, there weren't a lot of entries but the ones posted were extremely interesting and accurate, my sincerest congratulations to the contributors. These are my original solutions:

- **Sub-Challenge (1):**

This kind of problem is ideally suited for a recursive solution and thankfully HP-71B's powerful BASIC dialect allows for it. My solution is a 4-line recursive subprogram just 136 bytes long. All its 'smarts' are encapsulated in the careful setup of the FOR statement and the recursivity itself:

```
100 SUB SR(N,L) @ FOR I=MOD(L,2)+4*(L=5) TO 9 STEP 2+3*(L=5)
110 IF FLAG(I) THEN 130 ELSE M=N+I @ IF MOD(M,L) THEN 130
120 SFLAG I @ IF L<9 THEN CALL SR(10*M,L+1) @ CFLAG I ELSE DISP M*10 @ CFLAG I
130 NEXT I
```

Executing it from the keyboard, produces the only solution:

```
>CFLAG ALL @ CALL SR(0,1)
```

```
3816547290
```

in 28 seconds (0.32 sec in [Emu71](#) @ 2.4 Ghz)

- **Sub-Challenge (2):**

Likewise, this is best solved by a slight variation of the above subprogram, the following 4-line recursive subprogram (149 bytes, notice the FOR loop is set up \*differently\* than in the previous subprogram):

```

100 SUB SR(N,L,H,C) @ FOR I=L=1 TO 9 STEP 2-MOD(L,2)+4*(L=5)
110 IF FLAG(I) THEN 130 ELSE M=N+I @ IF MOD(M,L) THEN 130
120 SFLAG I @ IF L<H THEN CALL SR(10*M,L+1,H,C) @ CFLAG I ELSE C=C+1 @ CFLAG I
130 NEXT I

```

Calling it from the keyboard produces a list of the number of solutions for each N (706 in all):

```

>FOR I=1 TO 10 @ CFLAG ALL @ C=0 @ CALL SR(0,1,I,C) @ DISP I;C @ NEXT I

1 9
2 41
3 115
4 168
5 220
6 88
7 51
8 9
9 4
10 1

```

in just 25 minutes (7 seconds in Emu71 @ 2.4 Ghz)

- **Sub-Challenge (3):**

Again, some variation does the job, this time an even simpler and \*shorter\* (3-line, 115 byte) recursive subprogram:

```

100 SUB SR(N,L,H,C) @ FOR I=L=1 TO 9 STEP 2-MOD(L,2)+4*(L=5) @ M=N+I
110 IF MOD(M,L) THEN 120 ELSE IF L<H THEN CALL SR(10*M,L+1,H,C) ELSE C=C+1
120 NEXT I

```

Executing it from the keyboard produces the list of the number of solutions (11,453 in all):

```

>FOR I=1 TO 10 @ CFLAG ALL @ C=0 @ CALL SR(0,1,I,C) @ DISP I;C @ NEXT I

1 9
2 45
3 150
4 375
5 750

```

6 1200  
 7 1713  
 8 2227  
 9 2492  
 10 2492

in some 3 hours (only 10 min in Emu71 @ 2.4 Ghz).

### Comments:

However it might seem, these subprograms are anything but optimized, according to [my philosophy on how these challenges should be met](#). There are many things that can be done to speed them up, so way to go.

For case (3), though the number of solutions seems to grow forever that's not the case. Actually, 2492 is the maximum number of solutions for any N (in this case, for N=9 and N=10).

After that (N=11, 12, etc), the number of solutions decreases steadily till N=25, which again has a unique, 25-digit solution. Since none of the digits 0-9 results in a number divisible by 26 when added, there are no solutions for N=26 and beyond.

Thanks for your interest, see you all in a future S&SMC#10.

Best regards from V.

*Edited to remove a couple typos*

*Edited: 20 May 2005, 6:59 a.m.*

## Re: Short & Sweet Math Challenge #9: Divisibility !

Message #41 Posted by [Chris Dean](#) on 20 May 2005, 11:45 a.m.,  
 in response to message #1 by [Valentin Albillo](#)

Before you close I have a 49g+ solution which uses only the simplest rules for the odd and even 5 and 0 placings of the digits. The program was very laborious to write but gets the solution as required. It depends on the view point of the solver whether they are interested in the solution or elegance of the software solution. The choice is for the programmer only. The code would have been optimised but firstly I was interested in getting the solution. here it is

```
<< 1 9 FOR A << 2 8 FOR B << 1 9 FOR C IF 100 A * 10 B * + C + 3 MOD 0 == THEN << 2 8 FOR D IF 1000 A * 100 B * + 10 C * + D + 4 MOD 0
== THEN << 2 8 FOR E IF 100000 A * 10000 B * + 1000 C * + 100 D * + 50 + E + 6 MOD 0 == THEN << 1 9 FOR F IF 1000000 A 8 100000 B * +
10000 C * + 1000 D * + 500 + 10 E * + F + 7 MOD 0 == THEN << 2 8 FOR G IF 10000000 A * 10000000 B * + 1000000 C * + 100000 D * + 5000 +
100 E * + 10 F * + G 8 MOD 0 == THEN << 1 9 FOR H IF 100000000 A * 100000000 B * + 10000000 C * + 1000000 D * + 50000 + 100 E * + 100
```



```
F * + 10 G * + H + 9 MOD 0 == THEN IF 'A<>5 AND C<>5 AND F<>5 AND H<>5 AND A<>C AND A<>F AND A<>H AND C<>F AND C<>H
AND F<>H AND B<>D AND B<>E AND B<>G AND D<>E AND D<>G AND E<>G' THEN 1000000000 A * 100000000 B * + 10000000 C * +
1000000 D * + 500000 + 10000 E * + 1000 F * + 100 G * + 10 H * END END 2 STEP >> ->NUM END 2 STEP >> ->NUM END 2 STEP >> ->NUM
END 2 STEP >> ->NUM END 2 STEP >> ->NUM END 2 STEP >> ->NUM END 2 STEP >> ->NUM END 2 STEP >> ->NUM
```

The <> is for 'not equals'. As a programmer I can rip this software solution to shreds for inefficiency but as stated earlier I was more interested in the result and it does that job yielding 3816547290.

## Re: Short & Sweet Math Challenge #9: Divisibility ! (with formatted RPL listing)

Message #42 Posted by [Jeff O.](#) on 20 May 2005, 11:58 a.m.,  
in response to message #41 by Chris Dean

begging your permission, or at least your pardon, here's the listing with the formatting that I believe you intended:

Before you close I have a 49g+ solution which uses only the simplest rules for the odd and even 5 and 0 placings of the digits. The program was very laborious to write but gets the solution as required. It depends on the view point of the solver whether they are interested in the solution or elegance of the software solution. The choice is for the programmer only. The code would have been optimised but firstly I was interested in getting the solution. here it is

```
<< 1 9 FOR A
  << 2 8 FOR B
    << 1 9 FOR C
      IF 100 A * 10 B * + C + 3 MOD 0 == THEN
        << 2 8 FOR D
          IF 1000 A * 100 B * + 10 C * + D + 4 MOD 0 == THEN
            << 2 8 FOR E
              IF 10000 A * 1000 B * + 100 C * +
                100 D * + 50 + E + 6 MOD 0 == THEN
                << 1 9 FOR F
                  IF 100000 A * 8 10000 B * + 1000 C * + 1000 D * +
                    500 + 10 E * + F + 7 MOD 0 == THEN
                    << 2 8 FOR G
                      IF 1000000 A * 1000000 B * + 100000 C * +
                        10000 D * + 5000 + 100 E * + 10 F * + G 8 MOD 0 == THEN
                        << 1 9 FOR H
                          IF 10000000 A * 10000000 B * + 1000000 C * +
                            100000 D * + 50000 + 1000 E * + 100 F * +
                              10 G * + H + 9 MOD 0 == THEN
                              IF 'A<>5 AND C<>5 AND F<>5 AND H<>5 AND
                                A<>C AND A<>F AND A<>H AND C<>F AND
                                  C<>H AND F<>H AND B<>D AND B<>E AND
                                    B<>G AND D<>E AND D<>G AND E<>G' THEN
```

```
1000000000 A * 100000000 B * + 10000000 C * +  
1000000 D * + 500000 + 10000 E * + 1000 F * +  
100 G * + 10 H *  
END  
END 2 STEP >> ->NUM  
END 2 STEP >> ->NUM  
END 2 STEP >> ->NUM  
END 2 STEP >> ->NUM  
END 2 STEP >> ->NUM  
END 2 STEP >> ->NUM  
END 2 STEP >> ->NUM  
END 2 STEP >> ->NUM
```

The  $\neq$  is for 'not equals'. As a programmer I can rip this software solution to shreds for inefficiency but as stated earlier I was more interested in the result and it does that job yielding 3816547290.

*Edited: 20 May 2005, 11:59 a.m.*

### **Re: Short & Sweet Math Challenge #9: Divisibility ! (with formatted RPL listing)**

Message #43 Posted by [Chris Dean](#) on 20 May 2005, 2:34 p.m.,  
in response to message #42 by [Jeff O.](#)

Jeff

Thanks for the formatting. That is what I did in the word document I produced!!

Thanks again

Chris Dean

---

[ [Return to Index](#) | [Top of Index](#) ]



[Go back to the main exhibit hall](#)