

Re: Make 100 by using + - x / and 1~9

Source: <http://sci.tech-archive.net/Archive/sci.math/2007-10/msg05850.html>

- *From:* Bill Dubuque <wgd@xxxxxxxxxxxxxxxxxxxxxxxx>
 - *Date:* 30 Oct 2007 16:39:55 -0500
-

Alfred Heiligenbrunner <xyz@xxxxxx> wrote:

On 2007-10-25 17:22:50 -0400, 131208@xxxxxxxxxx said:

100 = 1 () 2 () 3 () 4 () 5 () 6 () 7 () 8 () 9
() can only be among +, -, *, /

Here a solution in Mathematica:

```
Select[Flatten[
Table[
Inner[StringJoin, {"1", "2", "3", "4", "5", "6", "7", "8"},
{"+", "-", "*", "/"}][{i1, i2, i3, i4, i5, i6, i7, i8}]],
StringJoin]
<> "9",
{i1, 4}, {i2, 4}, {i3, 4}, {i4, 4}, {i5, 4}, {i6, 4}, {i7, 4},
{i8, 4}]],
(ToExpression[#] == 100) &]
```

time is rather short (6 seconds on a Pentium 4, 3 GHz, 1.5 GB RAM)
I wonder, if there is not a more efficient or more elegant code.

Presented below is an elementary recursive solution, coded in Common Lisp. It's about 7000 times faster (0.0009 vs. 6 secs). For simplicity we consider first an unoptimized version that uses rational arithmetic and is 200 times faster (0.03 secs) It proceeds by collecting a sum (= list L) of product terms P (interpreting the expression as a sum of products / inverses) The first two recursive calls extend the current product P by a factor of J or 1/J. The final two recursive calls terminate the product P, with both possible signs, and subtract it from the target sum N (and exclude any negative product at front).

(F 100 2 1 ()) where

(defun F (N J P L) ; N + sum L is invariant

Re: Make 100 by using + - x / and 1~9

```
(if (> J 10)
  (if (= N 0) (print (cons '+ (reverse L))))
  (progn
    (when (< J 10)
      (F N (+ J 1) (* P J) L)
      (F N (+ J 1) (/ P J) L))
    (F (- N P) (+ J 1) J (cons P L))
    (if L (F (+ N P) (+ J 1) J (cons (- P) L))))))
```

To gain a further factor of 33 speedup, to 0.0009 secs, we scale the entire computation by 72 so it employs only integer (vs. rational) arithmetic (cf. explanation below).

```
(defun F (N J P L) ; N + sum L is invariant
  (if (> J 10)
    (if (= N 0) (print (cons '+ (mapcar #'(lambda (M) (/ M 72))
      (reverse L))))))
    (progn
      (when (< J 10)
        (F N (+ J 1) (* P J) L)
        (when (= 0 (mod P J))
          (F N (+ J 1) (/ P J) L)))
        (F (- N P) (+ J 1) (* 72 J) (cons P L))
        (if L (F (+ N P) (+ J 1) (* 72 J) (cons (- P) L))))))
```

The scaling trick works as follows. Observe, for p prime, if $p^n \parallel 9!$ (i.e. $9! = p^n m$, not $p|m$), then $n/2$ is the highest power of p that can occur in the denominator of any fractional terms if their sum is to be integral. (Proof hint: if some denominator takes more than half of the n p 's, then not enough p 's will remain to cancel it, whether the remaining p 's are placed in the numerator of the same fraction, or the denominator of other fractions). Hence, since $9! = 2^7 3^4 5 7$, we infer any denominator is bounded by $72 = 2^3 3^2$. Now, after scaling the entire computation by 72, we can exclude all inexact divisions, which results in the above integer–arithmetic version.

To keep the code simple (for the sake of those new to Lisp) product factors are not output (but they're easily inferred). The code would run just as fast even if it output factors. Below is the invocation and the output (manually indented).

```
(F 7200 2 72 ()) ->
(+ 24 5 6 56 9) ;i.e. 1*2*3*4 + 5 + 6 + 7*8 + 9 = 100
(+ 24 5 6 -7 72)
(+ 6 -20 42 72)
(+ 6 4 5 6 7 72)
(+ 1 20 7 72) ; i.e. 1 + 2*3*4*5/6 + 7 + 8*9 = 100
(+ 1 6 20 -6 7 72)
```

Re: Make 100 by using + - x / and 1~9

(+ 1 -6 20 6 7 72)
(+ 1 -6 -4 30 7 72)
(+ 1 -6 -4 -5 42 72)
(+ 1 2 -12 30 7 72)
(+ 1 2 -12 -5 42 72)
(+ 1 2 3 -20 42 72)
(+ 1 2 3 4 5 6 7 72)
(+ 1 -2 60 42 8 -9)
(+ 1 -2 60 -6 56 -9)

One could even derive further optimizations with more effort,
e.g. eliminate all divisions — an exercise for the reader.

One can easily translate the Lisp code to other languages.
In C, the speedup would probably increase to over 10000.
Anyone up for a try in C or another programming language?

Timings on a Dell D820 laptop: Core 2 Duo T7200 2GHz, 2GB,
running Allegro Common Lisp (ACL 6.0) on Windows XP SP2.

—Bill Dubuque

.