AB39.4.3 A Note on Integral Division

by L.G.L.T. Meertens, Mathematisch Centrum, Amsterdam.

Editor's note This paper is taken from a letter by Lambert Meertens in reply to someone who had pointed out that the mod operator in ALGOL 68 does not provide the same result as the remainder implied by the ÷ operator. This is a sad tale, involving a lot of history going back to the design of ALGOL 60 and the fact that most American computers worked in "sign and modulus" notation at that time. The matter has been raised before (see AB28.3.2, L04 and L05, half of which was accepted) but it has always been difficult to excite any concern over it. The conclusion of the present paper is that it is essentially the ÷ operator which is wrong. Future language designers please note.

My distrust for arguments based on "natural choice for widely spread computers" almost parallels my dislike for inconsistency.

i) The origin of integral division lies in the following question:
How many times b may be taken from a?
or, in a more mathematical expression
max {q | q ≥ 0 ∧ q × b may be taken from a}
or max {q | q ≥ 0 ∧ q × b ≤ a}.

If we abbreviate this to a quot b (from Latin quotiens = how many times), we might define

```
op quot = (int a, b) int:
if a < 0 ∧ b > 0 then undefined
elif b ≤ 0 then undefined
else int q := 0 ;
     while (q + 1) × b ≤ a do q +:= 1 od ;
     q
fi .
```

Note that the reasons behind the undefinedness for a < 0 ∧ b > 0 and for b ≤ 0 are of a different nature: in the first case no natural q satisfies q × b ≤ a, in the second case no such maximal q exists.

Several ways exist to relax the undefinedness. A "natural" way would be to express the original question algorithmically thus:

```
int q := 0 ;
while b may be taken from a
do (take b from a, q +:= 1) od
```

which would have a quot b equal to zero for a ≤ b.

Another direction is indicated by algebraic considerations, viz, by the wish to extend the validity of a = q × b → a quot b = q from natural a and positive b to arbitrary integral a and non-zero b.
One way to obtain the desired result is to define

a ÷ b = sign a × sign b × (abs a quot abs b) ,

chosen in ALGOL 60/68 and the hardware of many a computer, but this is certainly not the only way. Arguments for this choice in ALGOL 68 were the compatibility with ALGOL 60 and the Bauer-Samelson criterion (since the "normal" question is that for which a is natural and b positive). It would, however, have been possible, and, I think now, have been desirable, to define the operation in such a way that

(a + n × b) ÷ b = a ÷ b + n

would have been valid for arbitrary a and n and non-zero b. For example, in the binary search algorithm, an assignation like mid := (left + right) ÷ 2 will occur, and it is clearly desirable that this is not sensitive to a simultaneous shift in the bounds. At present, if we define

```
op mid = (ref [ ] real x1) ref real:
          x1 [(lwb x1 + upb x1) ÷ 2 ],
```

then

```
[-1 : 0] real x1 ; mid x1 :=: mid x1 [@1 ]
```

yields false!

ii) The origin of the modulo-operation lies in algebra:
Given a positive b, the integral numbers ℤ may be split into b residue classes, denoted 0, 1, . . ., b - 1, where m = {n ∈ ℤ | n ≡ m (mod b)} .
We now want an operator mod such that a ∈ m ↔ a mod b = m.

Again, this may in some way be extended to arbitrary non-zero b, e.g., by using $\mathbb{Z}_b = \mathbb{Z}_{-b}$, since $\mathbb{Z}_b$ is the quotient group ℤ /{b}, where the ideal {b} = bℤ = (-b)ℤ = {-b}. Note that a is already arbitrary integral.

iii) The inconsistency.

Define a rem b = a - a quot b × b. ALGOL 60 programmers will have felt a need for such an operation. As soon as one is doing multi-length integral arithmetic, base conversion, etc., the remainder is as important as the quotient. In view of the "size" of arithmetic values and the conversion routines in ALGOL 68, the need will have disappeared largely. Moreover, unless a and b have opposite and different signs, the programmer may use

a rem b = a mod b.

The "inconsistency" is now that this does not hold for all a and b.

Possible remedies:

R_i: Redefine ÷. This is a clean solution, which has been adopted in ALEPH.

R_{ii}: Strike mod. But whom do we serve by this? Expressing mod by means of ÷ is cumbersome and bug-prone (since the programmer is likely to overlook the possibility of negative a), and is probably dealt with more efficiently in code.

R_{iii}: Add rem. This is also a clean solution. However, there seems to be little need for it, and whatever need is left will quite likely be concerned with natural a and positive b (which is catered for by mod), and, if not, the user will as likely want a quot b to yield zero for a ≤ b, and therefore, a rem b to yield a, as any other result.

It seems too late for any of these, but it will be clear that I should favour R_i. For future ALGOL 68-ish languages, I should like to see something like op (÷, ÷×) = (int a, b) struct (int, int) : c (quotient, remainder) c.