Elementary Number Theory with the TI-89/92

Giulio C. Barozzi

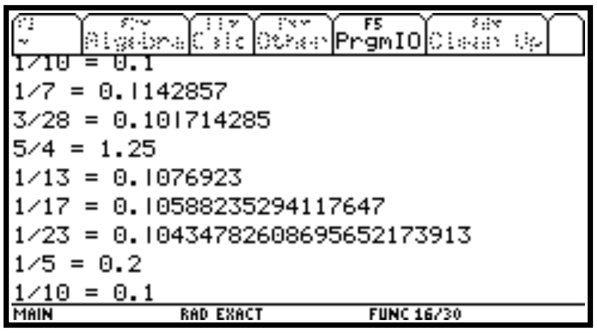
Università di Bologna – Italy
barozzi@ciram.ing.unibo.it
http://eulero.ing.unibo.it/~barozzi

The calculators TI 89 and TI 92 Plus provide a good starting point for explorations in the domain of elementary number theory. Let us recall the basic functions: given the natural numbers n and m, with $m > 0$, `intDiv(n,m)` gives the quotient of n divided by m, `mod(n,m)` gives the remainder of n divided by m, so that the identity

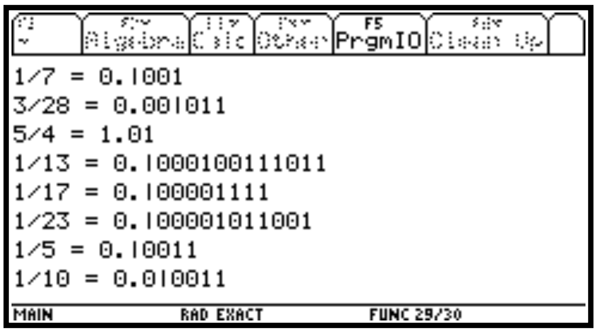
```
n = intDiv(n,m) * m + mod(n,m)
```

holds. Equally important are the functions `factor(n)`, which gives the prime factorization of n, and `isPrime(n)` which gives `True` in n is prime, `False` otherwise.

The first thing we can do, is a program for finding the decimal or the binary representation of a rational number n/m; this has been accomplished by writing the two programs `rd(n,m)` and `rb(n,m)` (see Appendix). A vertical bar (|) separates the non recurring from the recurring digits.



```
1/10 = 0.1
1/7 = 0.|142857
3/28 = 0.10|714285
5/4 = 1.25
1/13 = 0.|076923
1/17 = 0.|0588235294117647
1/23 = 0.|0434782608695652173913
1/5 = 0.2
1/10 = 0.1
MAIN          RAD EXACT          FUNC 16/30
```



```
1/7 = 0.|001
3/28 = 0.00|011
5/4 = 1.01
1/13 = 0.|000100111011
1/17 = 0.|00001111
1/23 = 0.|00001011001
1/5 = 0.|0011
1/10 = 0.0|0011
MAIN          RAD EXACT          FUNC 29/30
```

A bit of theory helps here: a classical result in number theory states that the number of non recurring digits in the decimal representation of n/m equals $\max\{\alpha, \gamma\}$, where

$$m = 2^{\alpha} \cdot 3^{\beta} \cdot 5^{\gamma} \ldots$$

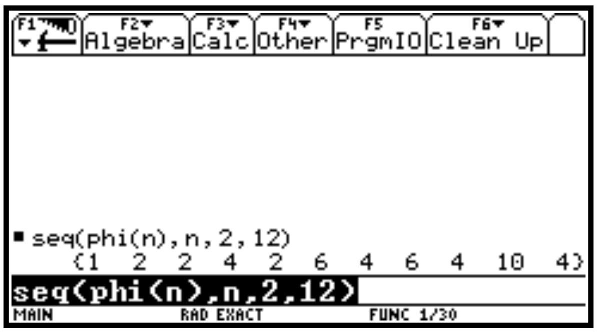
is the factorization of the denominator m. So we know in advance which remainder to check in the process of division.

An experimentation with various prime denominators m leads us to conjecture that the number of recurring digits (sometimes called the "period") is a factor of $n-1$. This is also true for the binary representation (and the same holds with respect to any base).

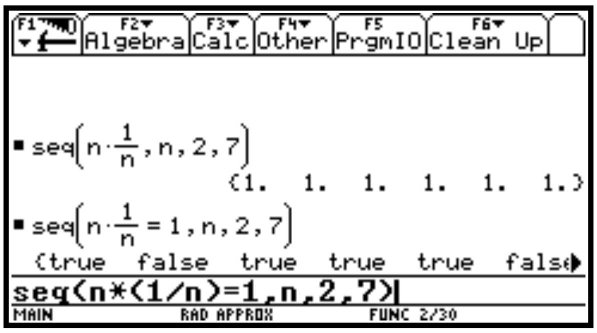
What happens if the denominator m is not prime? It turns out that the length of the "period" is a factor of

$$\phi(m) := \text{ number of numbers less than } m \text{ and coprime with it.}$$

The function phi(n) (see Appendix) computes $\phi(n)$ with a brute force method; the same function can be efficiently computed starting from the factorization of n (see bibliography [6]; the relevant programs are listed in the Appendix).



```
■ seq(phi(n),n,2,12)
   {1   2   2   4   2   6   4   6   4   10   4}
seq(phi(n),n,2,12)
MAIN            RAD EXACT          FUNC 1/30
```



$$\blacksquare\ \text{seq}\left(n\cdot\frac{1}{n},n,2,7\right)$$

```
                    {1.   1.   1.   1.   1.   1.}
```

$$\blacksquare\ \text{seq}\left(n\cdot\frac{1}{n}=1,n,2,7\right)$$

```
 {true  false  true  true  true  false▸
seq(n*(1/n)=1,n,2,7)
MAIN            RAD APPROX         FUNC 2/30
```

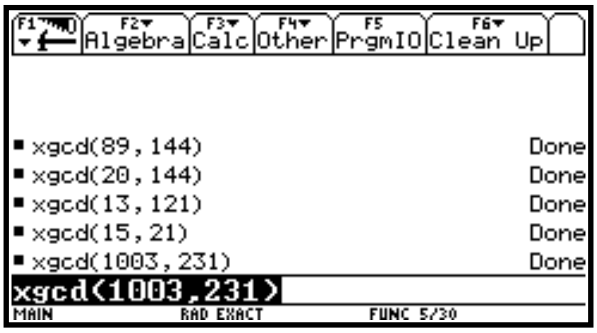
Numbers like 1/5 or 1/10, in general any rational number which cannot be expressed as a binary fraction, has a binary representation which is periodic; such a number cannot be stored exactly as a floating point number.

This explains the rather surprising results shown above (the calculator has been switched to `Approximate` mode).

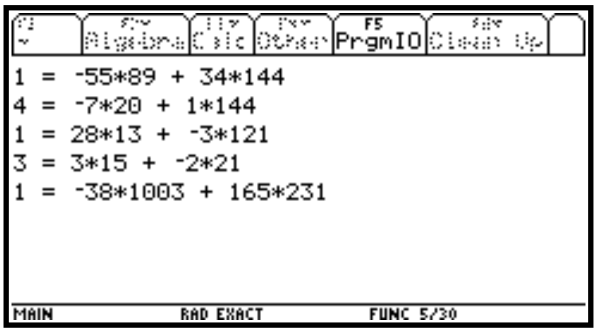
It is not difficult to program the so called “extended version” of Euclid’s algorithm, giving the Bezout’s formula

$$d = \gcd(n, m) = x \cdot n + y \cdot m$$

with x and y convenient integers (of opposite sign). This is done by the program `xgcd`.



```
■ xgcd(89,144)                    Done
■ xgcd(20,144)                    Done
■ xgcd(13,121)                    Done
■ xgcd(15,21)                     Done
■ xgcd(1003,231)                  Done
xgcd(1003,231)
MAIN            RAD EXACT          FUNC 5/30
```



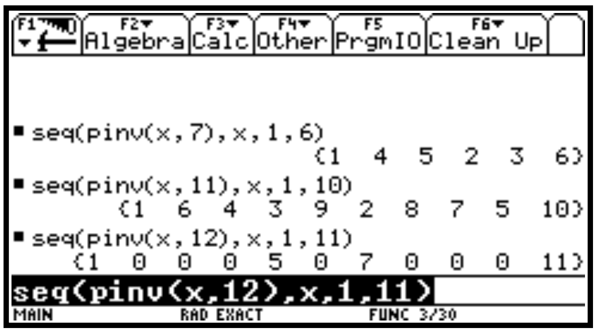
```
1 = -55*89 + 34*144
4 = -7*20 + 1*144
1 = 28*13 + -3*121
3 = 3*15 + -2*21
1 = -38*1003 + 165*231

MAIN            RAD EXACT          FUNC 5/30
```

If n and m are coprime, then $1 = \gcd(n, m) = x \cdot n + y \cdot m$, i.e.

$$x \cdot n = 1 - y \cdot m \iff x \cdot n \equiv 1 \pmod{m}$$

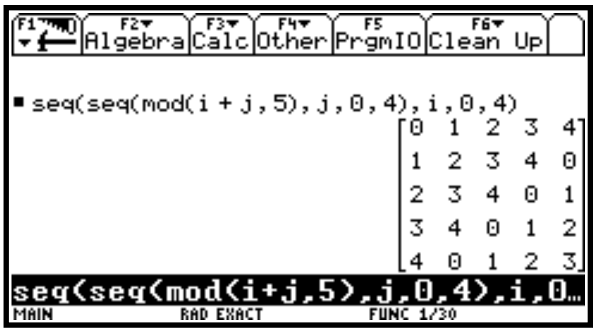
So the euclidean algorithm gives us the reciprocal of n modulo m. If we want to represent each class of congruence modulo m with the corresponding remainder between 0 and $m-1$, than n itself must be reduced modulo m (if necessary).

The function `pinv` computes the reciprocal of n modulo m, when n and m are coprime, and gives (quite arbitrarily) 0 otherwise.

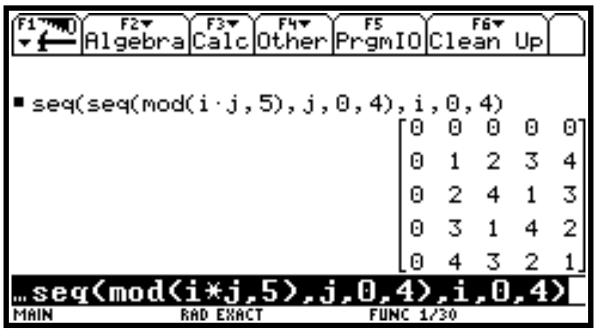


```
■ seq(pinv(x,7),x,1,6)
                      {1  4  5  2  3  6}
■ seq(pinv(x,11),x,1,10)
             {1  6  4  3  9  2  8  7  5  10}
■ seq(pinv(x,12),x,1,11)
          {1  0  0  0  5  0  7  0  0  0  11}
seq(pinv(x,12),x,1,11)
MAIN            RAD EXACT          FUNC 3/30
```

Using the function seq and mod we can construct the addition and multiplication tables modulo m.



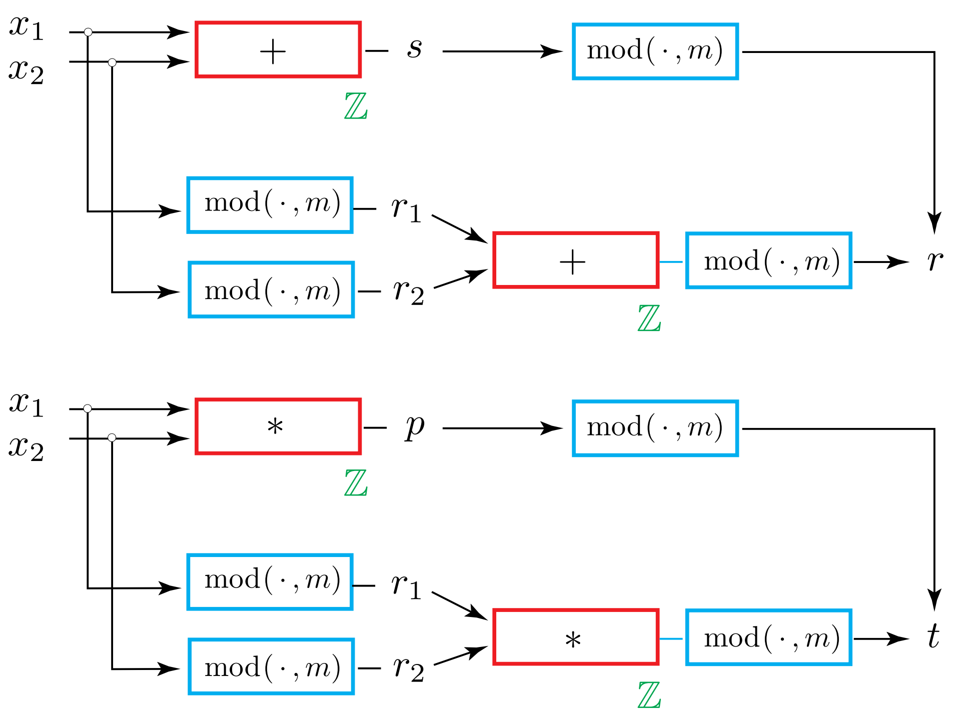
```
■ seq(seq(mod(i + j,5),j,0,4),i,0,4)
                              [0  1  2  3  4]
                              [1  2  3  4  0]
                              [2  3  4  0  1]
                              [3  4  0  1  2]
                              [4  0  1  2  3]
seq(seq(mod(i+j,5),j,0,4),i,0…
MAIN            RAD EXACT          FUNC 1/30
```



```
■ seq(seq(mod(i·j,5),j,0,4),i,0,4)
                              [0  0  0  0  0]
                              [0  1  2  3  4]
                              [0  2  4  1  3]
                              [0  3  1  4  2]
                              [0  4  3  2  1]
…seq(mod(i*j,5),j,0,4),i,0,4)
MAIN            RAD EXACT          FUNC 1/30
```

If we call $\mathbb{Z}_m$ the set of classes of congruence modulo m, then a classical theorem states that $\mathbb{Z}_m$ is a field iff m is prime, otherwise it is a commutative ring with unity. The invertible elements are those which are coprime with m, so their number is exacty $\phi(m)$.

FIGURE 1



The schemes in Figure 1 show the compatibility of the arithmetic operations in $\mathbb{Z}$ with the congruence modulo m.

Particularly important, for the sequel, is the computation of powers modulo m. As soon as a partial product exceedes m, it can be substituted by its remainder, so that no number greater than $(m-1)^2$ is involved in the computation.

For the moment we limit ourselves to the following consideration: suppose we want to compute

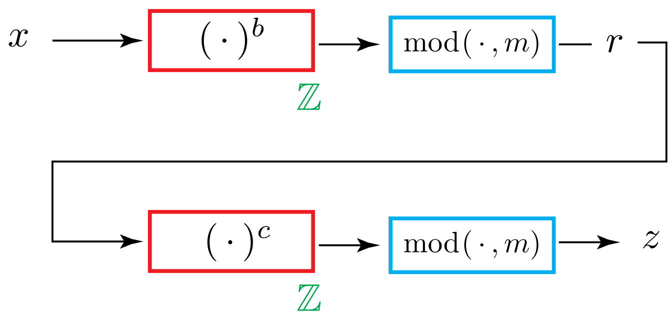
$$z = x^{b \cdot c} \pmod{m},$$

where x is some integer between 0 and $m-1$ and b and c are natural numbers.

Since $x^{b \cdot c} = \left(x^b\right)^c$, we can perform the following steps (see Figure 2)

$$r := x^b \pmod{m}, \quad z = r^c \pmod{m}.$$

FIGURE 2



In fact the function pwm(a,b,n) we have written (where a, b, n are natural numbers with $n > 0$), computes a^b mod n by cleverly combining the reduction modulo n and the binary representation of the exponent b (bibliography [4]). It achieves both the results of reducing the number of multiplications and of keeping each partial product less than or equal to $(n-1)^2$.

A first interesting result in modular arithmetic (the Latin name *modulus* was in fact chosen by C.F. Gauss in his *Disquisitione Arithmeticæ*, 1799) is due to P. de Fermat (1602-1665): it is the so called *Fermat's little theorem*:

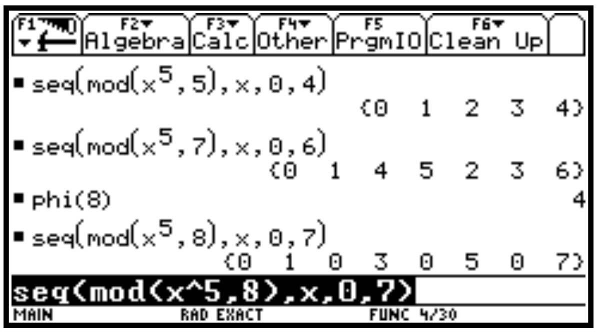
If n is prime, than for any x between 0 and $n-1$ one has $x^n \equiv x \bmod n$.

In other words: we can recover x by computing a power of it modulo n. Notice that if n is prime, the numbers between 1 and $n-1$ are obviously coprime with it.

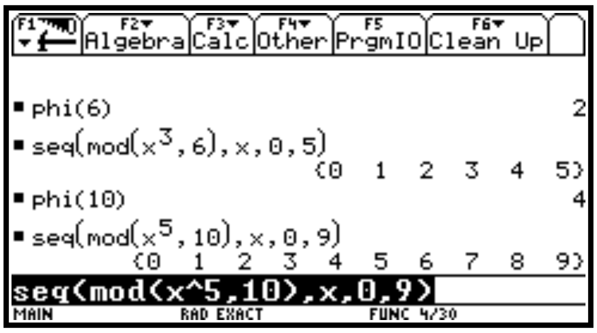
What happens if n is not prime? The answer came from Euler: for any x less than n coprime with it one has

$$x^{\phi(n)+1} \equiv x \bmod n.$$

A bit of experimentation shows that for $n = 4 = 2^2$ or $n = 8 = 2^3$, the equality we have written is false for x non coprime with n.



```
■ seq(mod(x^5,5),x,0,4)
                              {0  1  2  3  4}
■ seq(mod(x^5,7),x,0,6)
                        {0  1  4  5  2  3  6}
■ phi(8)                                    4
■ seq(mod(x^5,8),x,0,7)
                     {0  1  0  3  0  5  0  7}
seq(mod(x^5,8),x,0,7)
MAIN          RAD EXACT         FUNC 4/30
```



```
■ phi(6)                                    2
■ seq(mod(x^3,6),x,0,5)
                           {0  1  2  3  4  5}
■ phi(10)                                   4
■ seq(mod(x^5,10),x,0,9)
                  {0  1  2  3  4  5  6  7  8  9}
seq(mod(x^5,10),x,0,9)
MAIN          RAD EXACT         FUNC 4/30
```

But, surprisingly, for $n = 6 = 2 \cdot 3$, $n = 10 = 2 \cdot 5$ or $n = 15 = 3 \cdot 5$ the equality holds for every x between 0 and $n - 1$, the case $x = 0$ being obvious.

The following theorem holds:

If $n = p \cdot q$, with p and q distinct primes, then $x^{\phi(n)+1} \equiv x \bmod n$, for every natural number $x < n$.

How to compute $\phi(n) = \phi(p \cdot q)$? It is not difficult to show the ϕ is *multiplicative*, i.e. if a and b are coprime

$$\phi(a \cdot b) = \phi(a) \cdot \phi(b).$$

So, in the case of p and q distinct primes,

$$\phi(p \cdot q) = \phi(p) \cdot \phi(q) = (p-1)(q-1) = pq - p - q + 1,$$

and finally

$$\phi(n) + 1 = \phi(p \cdot q) + 1 = n - p - q + 2.$$

Suppose we can write $\phi(n) + 1$ as a product:

$$n - p - q + 2 = b \cdot c,$$

then $x \equiv x^{\phi(n)+1} = x^{b \cdot c}$ can be trasmitted from a sender to a receiver by using the following scheme (see Figure 3):

$$x \rightarrow \mathtt{pwm}(x, b, n) = r \rightarrow \mathtt{pwm}(r, c, n) = x$$

The result holds for any natural number x less than the modulus n. To code the message we need n and b (the so called *public* keys), to decode one needs n and c (the *private* key).

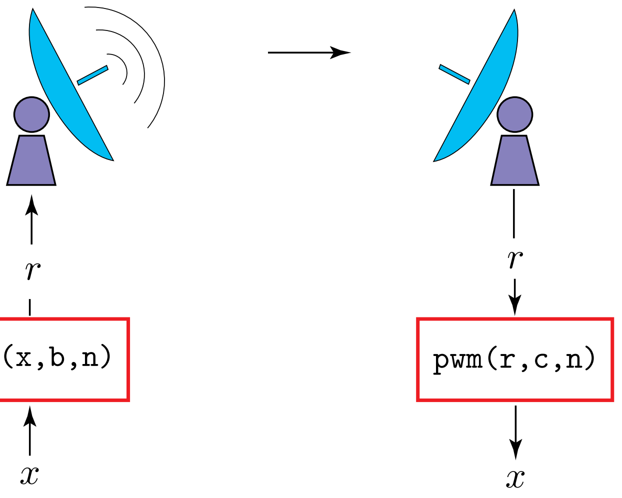
Now

$$b \cdot c = n - p - q + 2 \;\Rightarrow\; c = \frac{n - p - q + 2}{b}.$$

If p and q are big primes (say, 100 digits long) they can't be recovered from the knowledge of n in a reasonable time.

How to produce "big" primes? A theorem by Dirichlet (P.G. Lejeune-Dirichlet, 1805-1859) can help:

FIGURE 3



> If a and b are coprime, then the arithmetic progression $a + k \cdot b$, $k \in \mathbb{N}$, contains an infinite number of primes.

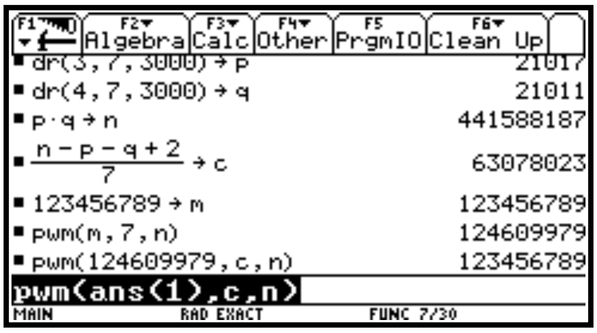
The program `dr(a,b,k)` explores the arithmetic progression $a + k \cdot b$ starting from a given k until a prime is found. The availability of the primality testing function `isPrime` is essential at this point.

Let us give a practical example. Suppose we want to use the private key $c = 7$. Then $n - p - q + 2 = (p - 1)(q - 1) + 1$ must be a multiple of 7, hence congruent to 0 modulo 7. This means that $(p - 1)(q - 1)$ must be congruent to 6 modulo 7, and since $6 = 2 \cdot 3$ we can find p congruent to 3 and q congruent to 4 modulo 7.

We can find as many primes as we like in the arithmetic progressions $3 + k \cdot 7$ and $4 + k \cdot 7$, and use them as values for p and q. For instance `dr(3,7,3000)` gives $p = 21\,017$ and `dr(4,7,3000)` gives $q = 21\,011$, so

$$n = p \cdot q = 441\,588\,187;$$

any "message" x less than n can be transmitted using the scheme we already know.



```
■ dr(3,7,3000) → p                21017
■ dr(4,7,3000) → q                21011
■ p·q → n                     441588187
■ (n - p - q + 2)/7 → c        63078023
■ 123456789 → m               123456789
■ pwm(m,7,n)                  124609979
■ pwm(124609979,c,n)          123456789
pwm(ans(1),c,n)
```

The procedure is known as the RSA method, from the initials of its discoverers R. Rivest, A. Shamir, L. Adleman in 1978.

Bibliography

[1] G.C. Barozzi: *Teoria elementare dei numeri con la TI 89/92*, Ipotesi vol. 2 (1999), n.2, 15-21, Pitagora Editrice (Bologna);

[2] H. Davenport: *Higher Arithmetic*, Cambridge University Press, 1992;

[3] G.H. Hardy, E.M. Wright: *An Introduction to the Theory of Numbers*, Clarendon Press, Oxford, 1979;

[4] D.E. Knuth: *The Art of Computer Programming*, Vol. 1: *Fundamental Algorithms* (1973), Vol. 2: *Seminumerical Algorithms* (1980), Addison Wesley, Reading MA;

[5] R. Rivest, A. Shamir, L. Adleman: *A method for obtaining digital signatures and public-key cryptosystems*, Communication ACM 21 (1978)n 120-128;

[6] L. Verardi: *Scomposizioni e funzioni aritmetiche con TI 89/92*, Ipotesi, Vol. 2 (1999), n.3, 11-15, Pitagora Editrice (Bologna) .

Appendix – Program listings

The following program computes the decimal representation of the rational number n/m ($n, m > 0$; a vertical bar (|) separates the non recurring from the recurring digits.

```
rd(n,m)
Prgm
ClrIO
Local g,c,h,k,r,j,rr,ad
gcd(n,m) → g
If g > 1 Then
 n/g → n:  m/g → m
EndIf
string(n)&"/"&string(m)&" = "&string(intDiv(n,m))&"."→ ad
m → r:  0 → h:  0 → k
While mod(r,2) = 0
 h+1 → h:  r/2 → r
EndWhile
While mod(r,5) = 0
 k+1 → k:  r/5 → r
EndWhile
max(h,k) → k:  0 → j:  0 → rr:  mod(n,m) → r
While r ≠ rr
 If j = k Then
  r → rr:  ad&"|" → ad
 EndIf
 j+1 → j:  intDiv(10*r,m) → c:  mod(10*r,m) → r
 ad&string(c) → ad
EndWhile
Disp ad
```

```
EndPrgm
```

The following program computes the binary representation of the rational number n/m $(0 < n < m)$.

```
rb(n,m)
Prgm
ClrIO
Local g,c,h,r,j,rr,ab
gcd(n,m) → g
If g > 1 Then
 n/g→ n:  m/g→ m
EndIf
string(n)&"/"&string(m)&" = "&string(intDiv(n,m))&"." → ab
m → r:  0 → h
While mod(r,2) = 0
 h+1 → h:  r/2 → r
EndWhile
0 → j:  0→ rr:  mod(n,m) → r
While r ≠ rr
 If j = h Then
  r → rr:  ab&"|" → ab
 EndIf
 j+1 → j
 intDiv(2*r,m)→ c:  mod(2*r,m)→ r:  ab&string(c) → ab
EndWhile
Disp ab
EndPrgm
```

The following function equals 1 if x and y are coprime, otherwise equals 0.

```
co(x,y)
Func
If gcd(x,y) > 1 Then
 0
Else
 1
EndIf
EndFunc
```

The following function (Euler's phi function) computes the number of numbers less than n and coprime with it.

```
phi(n)
Func
If isPrime(n) Then
 n-1
```

```
Else
 sum(seq(co(x,n), x, 1, n-1))
EndIf
EndFunc
```

The following function computes the reciprocal of n modulo m (if n and m are coprime) and gives 0 otherwise. It is based on Fermat's little theorem.

```
inv(n,m)
Func
If gcd(n,m) = 1 Then
 mod(n^(phi(m)-1),m)
Else
 0
EndIf
EndFunc
```

Comment: The computation of $\mathrm{mod}(n^{(\phi(m)-1)}, m)$ is quite naïve, and can result in an overflow. The following version takes advantage of the function `pwm` (see below) and is definitely to be used instead.

```
minv(n,m)
Func
Local k
If gcd(n,m)=1 Then
 phi(m)-1 → k:  pwm(n, k, m)
Else
 0
EndIf
EndFunc
```

Computation of the extended GCD (Bezout's formula).

```
xgcd(n,m)
Prgm
ClrIO
Local xv,xn,sv,sn,tv,tn,q,r
n → xv:  m → xn
1 → sv:  0 → sn:  0 → tv:  1 → tn
While xn > 0
 intDiv(xv,xn) → q:  mod(xv,xn) → r
 xn → xv:  r → xn
 sv-q*sn → r:  sn → sv:  r → sn
 tv-q*tn → r:  tn → tv:  r → tn
EndWhile
Disp string(xv)&" = "&string(sv)&"*"&string(n)&" +
"&string(tv)&"*"&string(m)
EndPrgm
```

Computation of the extended GCD (this version stores intermediate results as matrices).

```
mgcd(n,m)
Prgm
Local w,q,s
[[n,1,0][m,0,1]] → w
While w[2,1] > 0
 intDiv(w[1,1],w[2,1]) → q
 w[1] → s:  w[2] → w[1]:  s-q*w[2] → w[2]
EndWhile
Disp string(w[1,1])&" = "&string(w[1,2])&"*"&string(n)&" +
"&string(w[1,3])&"*"&string(m)
EndPrgm
```

The following function, exactly as `inv` and `minv`, computes the reciprocal of n modulo m (if n and m are coprime) and gives 0 otherwise. It is based on the algorithm for the computation of the `xgcd`.

```
pinv(n,m)
Func
ClrIO
Local xv,xn,sv,sn,q,r
n → xv:  m → xn:  1 → sv:  0 → sn
While xn > 0
 intDiv(xv,xn) → q:  mod(xv,xn) → r
 xn → xv:  r → xn
 sv-q*sn → r:  sn → sv:  r → sn
EndWhile
If xv = 1 Then
  mod(sv,m)
Else
  0
EndIf
EndFunc
```

The following function computes x^n modulo m in an efficient way. No number exceeding $(m-1)^2$ is involved in the computation.

```
pwm(x,n,m)
Func
Local z
1 → z:  mod(x,m) → x
While n > 0
```

```
  If mod(n,2) > 0 Then
   n-1 → n:  z*x → z:  mod(z,m) → z
  EndIf
   n/2 → n:  x*x → x:  mod(x,m) → x
EndWhile
```

The following function finds a prime in the arithmetic sequence $a + k \cdot b$, $k \in \mathbb{N}$, starting from a given k. The numbers a and b are coprime (Dirichlet's theorem).

```
dr(a,b,k)
Func
Local p
a+k*b → p
While isPrime(p) = false
 p+b → p
EndWhile
p
EndFunc
```

The following program (due to L. Verardi [6]) displays the prime factorization of an integer as a two-rows matrix; first row: prime factors, second row: their esponents.

```
factint(nn)
Prgm
ClrIO
Local fa,lf,po,df,ed,ld,p1,p2
[[1][1]] → fatt
string(factor(nn)) → fa
Disp nn:  Disp fa
dim(fa) → lf:  0 → kk
While lf ≠ 0
 kk+1→kk:  instring(fa,"*") → po
 If po ≠ 0 Then
  left(fa,po-1) → df
 Else
  fa → df
 EndIf
 instring(df,"^") → ed
 If ed ≠0 Then
  dim(df) → ld:  expr(left(df,ed-1)) → p1:  expr(right(df,ld-ed)) → p2
 Else
  expr(df) → p1:  1 → p2
 EndIf
 augment(fatt,[[p1][p2]]) → fatt
 If po ≠ 0 Then
```

```
    right(fa,lf-po) → fa:  dim(fa) → lf
   Else
    0 → lf
   EndIf
  EndWhile
  subMat(fatt,1,2,2,kk+1) → fatt
  Disp fatt
  EndPrgm
```

The following function computes $\phi(n)$ using the results of `factint`.

```
  euler(nn)
  Prgm
  Local i
  factint(nn):  nn → ee
  For i,1,kk
   ee*(1-1/(fatt[1,i])) → ee
  EndFor
  Disp ee
  EndPrgm
```