Version 4, 7 February 2021

0 Contents

1 Introduction

This note covers several topics relevant to modulo arithmetic :

Division giving quotient and remainder

Highest Common Factor, hcf, alias Greatest Common Divisor, gcd

Modular multiplicative inverse (x^{-1} in modulo arithmetic)

Algorithms and proofs of their correctness are given for the latter two, both for Integers and for Natural Numbers.

The note was originally written to support a special-purpose RPN calculator, hence some unusual function names.

2 Quotient and remainder division

2.1 Principles

Suppose we have a division operator, call it divop, defined as follows :
Divide b by a giving quotient q and remainder r such that

$$b = q \times a + r$$

Note :
Usually the numbers a, b, q, r are either

Natural Numbers, (0, 1, 2, …), or
Integers, (…, -2, -1, 0, 1, 2, …).

The definition is the same in both cases.

The definition can also be used for

Real Numbers, (-3.7, 0.0, 0.63, 1.0, 2.154, etc.).

To be useful, an additional constraint is then needed :

that q is an '**integral**' value, (…, -2.0, -1.0, 0.0, 1.0, 2.0, …).

Provided $a \neq 0$ there can be many solutions, for instance

$$q = 0, r = b$$

And

$$q = 1, r = b - a$$

Note : If $a = 0$ then the solution is $r = b$; q can be any number. Not useful.

If

$$b = q \times a + r$$

is a solution then

$$b = q \times a + r \;\; + a - a$$

so

$b = (q + 1) \times a + (r - a)$ When the subtraction is defined
$b = (q - 1) \times a + (r + a)$ When the subtraction is defined

are also solutions.

In general,

$$b = (q + m) \times a + (r - m \times a)$$
$$b = (q - n) \times a + (r + n \times a)$$

for any numbers m and n with some restrictions : if negative numbers are not allowed then the values of m and n are restricted to ones where the subtractions are defined (e.g 2-1, but not 1-2); if the numbers are Real Numbers then the values of m and n are restricted to integral values.

These are all the solutions that meet the requirements when $a \neq 0$. With so many solutions our supposed operator divop is not yet fully specified.

Provided $a \neq 0$ and negative numbers are allowed, we can require r to be in any interval

$$c \leq r < (c + |a|)$$

where c is any number and $|a|$ is abs(a).

If there is a solution then there is only one solution.
When negative numbers are allowed and $a \neq 0$ then there is always a solution.
When negative numbers are not allowed and $a \neq 0$ then there is also always a solution provided $c \leq b$.

This existence and uniqueness property is well known for the case when $c = 0$; it is proved in many textbooks. It can be shown for other values of c by displaying

$$(b - c) = q \times a + (r - c) \text{ with } 0 \leq (r - c) < |a|$$

which has a unique q and $(r - c)$ so that $c \leq r < (c + |a|)$.

Once c is given, our supposed operator divop becomes fully specified.

If negative numbers are not allowed then the only definition that works for all $a, b, q, r \geq 0$ is

$a = 0$: operator undefined
$a > 0$: $b = q \times a + r$ **and** $0 \leq r < a$

I.e The well-known rule where $c = 0$. Note that $a = |a|$ in this case.

If numbers are allowed to be negative then c can be chosen to suit the application, possibly with different values of c for different cases. The choice of value(s) can provoke fierce arguments!

Practical example :
(11 o'clock + 2 hours) modulo 12 is 1 o'clock
(11 o'clock + 1 hour) modulo 12 is 12 o'clock : **not** 0 o'clock!
Here $c = 1$.

2.2 Operator /qr

The binary operator /qr divides b by a returning the quotient q and remainder r, as follows:

$a = 0$: /qr(b, a) is undefined;
$a \neq 0$: /qr(b, a) = (q, r) where $b = q \times a + r$ and $0 \leq r < |a|$

I.e As in divop with $c = 0$.

2.3 Operator /cqr

The ternary operator /cqr divides b by a returning the quotient q and remainder r offset by c, as follows:

if	$a = 0$
then	/cqr(c, b, a) is undefined
else if	negative numbers are not allowed and $b < c$
then	/cqr(c, b, a) is undefined
else	/cqr(c, b, a) = (q, r)
	where $b = q\times a + r$ **and** $c \le r < (c + \|a\|)$

I.e As in divop with c given.

The user can choose whichever value of c is appropriate in the context of the application. E.g A value of 1 for 12 hour o'clock arithmetic. The user is responsible for providing the right value of c in each case.

2.3.1 Implementation of /cqr

Assume we have a, b, c, q, r such that $a \neq 0$ **and**

$$b = q\times a + r$$

but where r is not necessarily in the required interval

$$c \le r < (c + |a|)$$

Note : One possibility is $q = 0$, $r = b$.

How to adjust q and r to meet the desired constraint on r ?

Use one of the two equations as necessary :

$$b = (q - 1)\times a + (r + a)$$
$$b = (q + 1)\times a + (r - a)$$

Apply this repeatedly to increase or decrease r as necessary until r is in the target interval.

If numbers can't be negative then this must be done in a way that assures we never try to go negative or use negative numbers.

Cases :

$r < c$ **and** $a > 0$: q - 1, $r + a$
$r < c$ **and** $a < 0$: $q + 1$, $r - a$, or equally $r + |a|$

$r \geq (c + |a|)$ **and** $a > 0$: $q + 1$, $r - a$
$r \geq (c + |a|)$ **and** $a < 0$: q - 1, $r + a$, or equally r - $|a|$

This could all be done in one while loop after setting suitable increments for q and r, but to avoid negative numbers somewhat more disorderly code must be used.

```
while ( r < c )
  {
      // Not possible if no negative numbers and c=0!
    if (a > 0)
      { q -= 1; r += a; }  // or  r += abs(a);
    else
      { q += 1; r += abs(a); }
  }

while ( r >= (c + abs(a)) )
  {
    if (a > 0)
      { q += 1; r -= a } // or r -= abs(a);
    else
      { q -= 1; r -= abs(a); }
  }
```

This can be tidied up a little :

```
while ( r < c )
  {
      // Not possible if no negative numbers and c=0!
    r += abs(a);
    if (a > 0)
      q -= 1;
    else
      q += 1;
  }


while ( r >= (c + abs(a)) )
  {
    r -= abs(a);
    if (a > 0)
      q += 1;
    else
      q -= 1;
  }
```

If $c < a$ or not much larger then only a few iterations of this step-by-one procedure need to be done. If c is much larger then the time to do the steps could be inconveniently high. To avoid this, the step-by-one procedure can be preceded by a single large step, as follows.

First do

/cqr$(0, b, a) = (q, r)$ and
/cqr$(0, c, a) = (q_c, r_c)$ where
$b = q \times a + r$
$c = q_c \times a + r_c$
and r and r_c are close to zero

Notice that

$b = q \times a - (q_c \times a) + (q_c \times a) + r$

so

$b = (q - q_c) \times a + (q_c \times a) + r$

but

$q_c \times a = c - r_c$

so

$b = (q - q_c) \times a + c - r_c + r$

r and r_c are close to zero so the new remainder is close to c, as desired. The starting point for the step-by-one procedure is now (q_1, r_1) where

$q_1 = (q - q_c)$
$r_1 = b - q_1 \times a$

2.4 Operators mod and %

mod is the modulo operator. Given (b, a), the operator mod divides b by a and returns the remainder r as follows:

$a \leq 0$: mod(b, a) is undefined;
$a > 0$: mod$(b, a) = r$ where $b = q \times a + r$ and $0 \leq r < a$

I.e It returns the r value of /qr(b, a).

Note that a is not allowed to be zero or negative. b can be positive or zero; it can also be negative if the number system in use has negative numbers. The result is zero or positive even when b is negative.

The operator **%** is an infix alias for mod, defined by :

$b \% a =$ mod(b, a)

It can be used as an alternative notation when desired.

Warning : This is the definition of mod (and **%**) used in this document. Programming languages typically define it to return the r value of /cqr with c

depending on the sign of b, the sign of a, whether a and b have the same sign, etc. Different languages can have different rules.

mod returns r values with $0 \le r < a$. What if a different range is needed ?
We could follow /cqr and use c as an index, writing mod_c , defined by :

$a \le 0$: $\text{mod}_c\,(b, a)$ is undefined;
$a > 0$: $\text{mod}_c\,(b, a) = r$ where $b = q \times a + r$ and $c \le r < (c + |a|)$

Then h o'clock is written $h\ \text{mod}_1\ 12$.

Using mod to mean mod_0 is a small abuse of notation. It would be unlikely to cause confusion.

2.4.1 Proof that intermediate values can be reduced modulo *B*

This is a result that is used later on in proofs. It can also be used in some of the algorithms.

To prove :
Given numbers x, y, B with $0 < B$, and an operator op that is + or ×,
then
$(x \text{ op } y) \% B$
$= [\ (x \% B) \text{ op } y\ ] \% B$
$= [\ x \text{ op } (y \% B)\ ] \% B$
$= [\ (x \% B) \text{ op } (y \% B)\ ] \% B$

Remark :
If numbers can be negative then this result also applies when the operator op is **-** (minus).

Proof :

if $B \le 0$
then there is nothing to prove,
otherwise :

Define some quotients and remainders
$x = q_x \times B + r_x$ for some q_x, r_x with $0 \le r_x < B$
$y = q_y \times B + r_y$ for some q_y, r_y with $0 \le r_y < B$
and
$(r_x \text{ op } r_y) = q \times B + r$ for some q, r with $0 \le r < B$

Now substitute the expressions :

A :

$(x \text{ op } y) \% B$
$= ((q_x \times B + r_x) \text{ op } (q_y \times B + r_y)) \% B$
$= ((\ldots) \times B + (r_x \text{ op } r_y)) \% B$
$= ((\ldots) \times B + (q \times B + r)) \% B$ — By the definition of q, r
$= ((\ldots + q) \times B + r) \% B$
$= r$ — By the definition of % in 2.4

B :

$[(x \% B) \text{ op } y] \% B$
$= [(q_x \times B + r_x) \% B \text{ op } (q_y \times B + r_y)] \% B$
$= [r_x \text{ op } (q_y \times B + r_y)] \% B$ — By the definition of % in 2.4
$= ((\ldots) \times B + (r_x \text{ op } r_y)) \% B$
$= ((\ldots) \times B + (q \times B + r)) \% B$ — By the definition of q, r
$= ((\ldots + q) \times B + r) \% B$
$= r$ — By the definition of % in 2.4

C :

$[x \text{ op } (y \% B)] \% B$
$= [(q_x \times B + r_x) \text{ op } (q_y \times B + r_y) \% B] \% B$
$= [(q_x \times B + r_x) \text{ op } r_y] \% B$ — By the definition of % in 2.4
$= ((\ldots) \times B + (r_x \text{ op } r_y)) \% B$
$= ((\ldots) \times B + (q \times B + r)) \% B$ — By the definition of q, r
$= ((\ldots + q) \times B + r) \% B$
$= r$ — By the definition of % in 2.4

D :

$[(x \% B) \text{ op } (y \% B)] \% B$
$= [(q_x \times B + r_x) \% B \text{ op } (q_y \times B + r_y) \% B] \% B$
$= [r_x \text{ op } r_y] \% B$ — By the definition of % in 2.4
$= [q \times B + r] \% B$ — By the definition of q, r
$= r$ — By the definition of % in 2.4

Therefore, by **A**, **B**, **C**, **D** :

r
$= (x \text{ op } y) \% B$
$= [(x \% B) \text{ op } y] \% B$
$= [x \text{ op } (y \% B)] \% B$
$= [(x \% B) \text{ op } (y \% B)] \% B$

QED

This theorem can be applied repeatedly to sub-expressions if desired, as well as to whole expressions.

But note the warning at the end of section 4.5.

2.5 Proofs of some properties of the % operator

These are some results that are used later on in proofs.

Reminder : $x \% B$ is undefined if $B \leq 0$, see section 2.4.

.1 **To prove** :
if $0 \leq x < B$ **then** $x \% B = x$

if not $(0 \leq x < B)$
then there is nothing to prove,
otherwise :

$x = 0 \times B + x$ with $0 \leq x < B$, and $B > 0$
so
$(x \% B) = x$ Definition of % in 2.4

QED

Some consequences :
if $B > 0$ **then** $0 \% B = 0$
if $B > 1$ **then** $1 \% B = 1$

.2 **To prove** :
$(x \% B) \% B = x \% B$

if $B \leq 0$
then both sides are undefined,
otherwise :

$x = q \times B + r$ for some q, r, with $0 \leq r < B$,
so
$x \% B = r$ Definition of % in 2.4

and

$(x \% B) \% B$
$= r \% B$
$= r$ By .1
$= x \% B$

QED

.3 **To prove** :
$(c \times B + x) \% B = x \% B$ for any integral c

if $B \leq 0$
then both sides are undefined,
otherwise :

$x = q \times B + r$ for some q, r with $0 \leq r < B$

so

$(c \times B + x) \% B$
$= ((c + q) \times B + r) \% B$
$= r$ Definition of % in 2.4
$= x \% B$ Definition of % in 2.4

QED

.4 **To prove :**
if $B > 0$
then $[x + (B - x \% B) \% B] \% B = 0$

That is, $(B - x \% B) \% B$ is the modular additive inverse of x with respect to B, if it exists. In effect, it is '$-x$'.

Proof :

if $B \leq 0$
then there is nothing to prove,
otherwise :

As $0 \leq (x \% B) < B$ the subtraction is always defined even if negative numbers are not allowed. Definition of % in 2.4

$[x + (B - x \% B) \% B] \% B$
$= [x + (B - x \% B)] \% B$ By 2.4.1
$= [x \% B + (B - x \% B)] \% B$ By 2.4.1
$= [x \% B + B - x \% B] \% B$
$= [1 \times B + 0] \% B$
$= 0$ Definition of % in 2.4

QED

.5 **To prove :**
if $[B > 0$ and $C > 0]$
then $(x \% (B \times C)) \% B = x \% B$

if $[B \leq 0$ or $C \leq 0]$
then there is nothing to prove
otherwise :

LHS :
$x = q_1 \times (B \times C) + r_1$ for some q_1, r_1 with $0 \leq r_1 < B \times C$
$= q_1 \times (B \times C) + q_2 \times B + r_2$ for some q_2, r_2 with $0 \leq r_2 < B$
$= (q_1 \times C + q_2) \times B + r_2$

RHS :
$x = q_0 \times B + r_0$ for some q_0, r_0 with $0 \leq r_0 < B$

But q, r values are unique,
so

$r_2 = r_0$

and

$(x \% B \times C) \% B = r_1 \% B = r_2 = r_0$
$= x \% B$ Definition of % in 2.4

QED

.6 **To prove :**
if $[x > 1$ **and** $y > 1$ **and** hcf$(x, y) = 1]$
then $x \% y \neq 0$

where hcf(x, y) is the Highest Common Factor of x and y

if [$x \le 1$ **or** $y \le 1$ **or** $\text{hcf}(x, y) \ne 1$]
then there is nothing to prove
otherwise :

Case 1 : $x < y$

$1 < x < y$,

so

$x \% y = x$ By **.1**

but

$x \ne 0$ As x > 1

so

$x \% y \ne 0$ in Case 1

Case 2 : $x \ge y$

$x = q\times y + r$ for some q, r with $0 \le r < y$ Definition of % in 2.4

$q > 0$ Because $x > 0$ and $y > 0$ and $x \ge y$

assume $x \% y = 0$

then

$x = q\times y$ Definition of % in 2.4

$\text{hcf}(x, y) = y$ Definition of Highest Common Factor

but

$y > 1$ Premise

so

$\text{hcf}(x, y) \ne 1$

and

$\text{hcf}(x, y) = 1$ Premise

contradiction!

Therefore

$x \% y \ne 0$ in Case 2 Reductio ad absurdam

By Case 1 and Case 2

$x \% y \ne 0$

QED

.7 To prove :

if $B > 0$ **and** $x \% B = y \% B$
then $(x\times a) \% B = (y\times a) \% B$

if $B \le 0$ **or** $x \% B \ne y \% B$
then there is nothing to prove
otherwise :

Define some quotients and remainders

$x = q_x \times B + r_x$ for some q_x, r_y with $0 \le r_x < B$
$y = q_y \times B + r_y$ for some q_y, r_y with $0 \le r_y < B$

Now

$x \% B = r_x$ and $y \% B = r_y$ Definition of % in 2.4

so

$r_x = r_y$ Premise
$= r$, say

therefore

$(x \times a) \% B$
$= ((x \% B) \times a) \% B$ By 2.4.1
$= (r \times a) \% B$

and

$(y \times a) \% B$
$= ((y \% B) \times a) \% B$ By 2.4.1
$= (r \times a) \% B$

so

$(x \times a) \% B = (r \times a) \% B = (y \times a) \% B$

QED

3 The Highest Common Factor (hcf) algorithm

3.1 The plan

.1 State the hcf algorithm and its pre- and post-conditions.

.2 Prove that it always terminates.

.3 Prove that it always produces a common factor.

.4 Prove that it always produces the highest common factor.

3.2 The hcf algorithm and its pre- and post-conditions

The problem

Given numbers a and b, find their highest common factor h. That is, find a number, $h = \text{hcf}(b, a)$, that divides both a and b exactly (a common factor), and is the highest such number.

Here, "*numbers*" are either all Natural Numbers, or all Integers.

Pre-conditions

Both a and b are positive : $0 < a$ **and** $0 < b$

Post-conditions

$h = \text{hcf}(b, a)$ is a common factor of a and b (h divides both a and b exactly)
and
h is the highest such common factor.

The algorithm :

Calculate a succession of couples starting with $(a_0, b_0) = (a, b)$, and ending with the couple $(a_m, 0)$.

For $0 \leq n < m$ the successor of the couple (a_n, b_n) is (a_{n+1}, b_{n+1}) where
$a_{n+1} = b_n$
$b_{n+1} = a_n \% b_n$

The last couple in the sequence is $(a_m, 0) = (h, 0)$.
h is the answer : $\text{hcf}(b, a) = h$.

3.3 Proof that the algorithm always terminates

The definition of $r = x \% y$ is that $x = q \times y + r$ for some quotient q with $0 \leq r < y$.

There is always a solution, and only one, provided $y > 0$.
Hence, in going from (a_n, b_n)
to
$(a_{n+1}, b_{n+1}) = (b_n, \quad a_n \ \% \ b_n)$
we must have $0 \leq b_{n+1} < b_n$.

The sequence starts at (~ , b) , proceeds through a succession of couples with a strictly decreasing right-hand part that is never less than 0, and ends at (~′ , 0) , so there can at most be $b + 1$ elements in the sequence, usually fewer.

Therefore the sequence of couples is always finite; the algorithm always terminates.

QED

3.4 Proof that the algorithm always produces a common factor

Consider the final couple in the sequence. It is $(a_m, 0) = (h, 0)$
for some $m > 0$.

Note : $b > 0$ by a pre-condition so the last element is not the first element.

.1 **To prove :**
for all $0 \leq n < m$
if h is a common factor of a_{n+1} and b_{n+1}
then h is a common factor of a_n and b_n, .

By the definition of the sequence,
$a_{n+1} = b_n$
$b_{n+1} = a_n \% b_n$

A :
$b_n = a_{n+1}$
so immediately :
if h is a factor of a_{n+1}
then h is a factor of b_n.

B :
$a_n \ \% \ b_n = b_{n+1}$

By the definition of %,

$a_n = q \times b_n + (a_n \% b_n)$ for some q

That is,

$a_n = q \times a_{n+1} + b_{n+1}$ for some q .

if h is a factor of a_{n+1} **and** b_{n+1}

then $a_n = q \times a' \times h + b' \times h$ for some a' and b'

$a_n = (q \times a' + b') \times h$ for some a' and b' , so h is a factor of a_n.

By A and B,

if h is a common factor of a_{n+1} and b_{n+1}

then h is a common factor of a_n and b_n.

QED

.2 To prove :

h is a common factor of a and b.

h is a common factor of $a_m = h$ and of $b_m = 0$ (vacuously).

Therefore, by .1 and induction, h is a common factor of a_n and b_n for all $0 \le n \le m$, and hence of $a_0 = a$ and $b_0 = b$.

QED

3.5 Proof that the algorithm always produces the highest common factor

.1 To prove :

for all $0 \le n < m$

if c is any common factor of a_n and b_n

then c is a common factor of a_{n+1} and b_{n+1}.

Reminder :

Any $c \neq 0$ is a factor of 0, vacuously, and $b_m = 0$, so therefore c is a factor of b_m.

By the definition of the sequence,

$a_{n+1} = b_n$

$b_{n+1} = a_n \% b_n$ and

$b_n > 0$

A :

$a_{n+1} = b_n$

so immediately

if c is a factor of b_n
then c is a factor of a_{n+1}.

B :

$b_{n+1} = a_n \% b_n$

By the definition of %

$a_n = q \times b_n + (a_n \% b_n)$ for some q with $0 \le (a_n \% b_n) < b_n$

so

$a_n \ge q \times b_n$

and the subtraction

$a_n - q \times b_n$

is defined even if negative numbers are not allowed.

Therefore

$(a_n \% b_n) = a_n - q \times b_n$

and it is always defined

so

$b_{n+1} = a_n - q \times b_n$

if c is a factor of both a_n and b_n
then $a_n = a' \times c$ for some a' and
$b_n = b' \times c$, for some b'

so

$b_{n+1} = a' \times c - q \times b' \times c = (a' - q \times b') \times c$

therefore

if c is a factor of both a_n and b_n
then c is a factor of b_{n+1}

By A and B,

if c is a common factor of a_n and b_n
then c is a common factor of a_{n+1} and b_{n+1}

QED

.2 To prove :

if c is any common factor of $a = a_0$ and of $b = b_0$
then **for all** $0 \le n \le m$
c is a common factor of a_n and b_n

By .1 and induction

if c is a common factor of a_0 and b_0
then **for all** $0 \le n \le m$
c is a common factor of a_n and b_n

QED

.3 **To prove** :

h is the highest common factor of a and b.

For any common factor, c, of a and b we have that
$c \le a$ and $c \le b$

As

there exists a common factor of a and b, namely 1, and c cannot be arbitrarily large

Then

there is a highest common factor, $cmax$, of a and b

By .2

$cmax$ is a factor of a_m
$a_m = h$

so $cmax \le h$

By 3.4, item .2

h is a common factor of a and b

so $h \le cmax$

Therefore

$cmax \le h \le cmax$

so $h = cmax$

I.e h is the highest common factor of a and b

QED

3.6 Some observations

.1 Negative numbers

Suppose negative numbers are allowed. If *c* is a common factor of *a* and *b* then -*c* is also a common factor.

Also 'highest' can be regarded as the most positive rather than the one with the greatest magnitude.

Thus the definition of the hcf function can be extended to a function hcf* on all non-zero integers by defining :

hcf*(b, a) = hcf(abs(b), abs(a)), where $a \neq 0$ and $b \neq 0$

This definition can also be used when negative numbers are not allowed.

.2 Zero numbers

The algorithm defined in 3.2 never produces a zero result; it is always 1 or more. The case of attempting to apply hcf* with *a* or *b* zero can be reported unambiguously by returning a zero result.

Thus the hcf* function can be extended to the function hcf** on all numbers by defining :

if ($a \neq 0$ and $b \neq 0$)
then hcf**(b, a) = hcf(abs(b), abs(a))
otherwise hcf**(b, a) = 0

.3 '*The*' hcf **function**

At this point the hcf** function can be renamed as hcf, so extending the function defined in section 3.2 to all Natural Numbers and all Integers.

4 The inverse modulo (inv_mod) algorithm

4.1 The inv_mod algorithm : Outline and pre- and post-conditions

The problem

Given numbers *Num* and *Base*, find the modular multiplicative inverse, *Inv*, of *Num* with respect to *Base*, if it exists. That is, find a number

$$Inv = \text{inv\_mod}(Num, Base)$$

such that

$$\text{mod}(\, Num \times Inv, Base\,) = 1$$

In other words, *Inv* is Num^{-1} in modulo *Base* arithmetic.

Here, "*numbers*" are either all Natural Numbers, or all Integers.

Symbol definitions

Some symbols used in the description of the algorithm and in the proofs need to be defined more explicitly than usual.

.1 The × and · operators

Given any two numbers x and y, then $x \times y$ is defined here to be x multiplied by y using the rules of multiplication for Natural Numbers when x and y are Natural Numbers and the rules for Integers when x and y are Integers.

$x \cdot y$ is defined here to mean the same as $x \times y$. (Dot can be easier to read in a long expression). And $x \cdot y \,\%\, B$ is defined to mean $(x \cdot y) \,\%\, B$ and $x \,\%\, B \cdot C$ to mean $x \,\%\, (B \cdot C)$.

.2 The (-*x*) symbol

Given any two numbers x and B with $0 < B$, then there are two cases relevant here :

if the number system in use allows negative numbers
then $(-x)$ means $-x$
else $(-x)$ means $[B - (x \,\%\, B)] \,\%\, B$

The value of B is assumed from the context. When "*numbers*" are Natural Numbers B will always be *Base* in what follows.

Pre-conditions

.1 $Base > 1$

Otherwise either $Base = 1$ and $Num \% Base = 0$, which has no inverse,
or
$Base \leq 0$ so $Num \% Base$ is not defined, see 2.4

.2 $Num \% Base \neq 0$

0 has no inverse, as usual.

.3 $\text{hcf}(Num, Base) = 1$

Otherwise an inverse does not exist, see 4.3.

Post-conditions

.1 The result, $Inv = \text{inv\_mod}(Num, Base)$,
is the desired inverse, obeying
$\text{mod}(Num \times Inv, Base) = 1$

.2 $0 \leq Inv < Base$

The result is always tidied up.

The algorithm : Outline

Note : There are other algorithms, see Wikipedia (search term Modular Multiplicative Inverse).

Perform the hcf algorithm, but recording information in a sequence along the way. At each stage of the calculation record six values :

.1 The reducing pair of numbers of the hcf algorithm;
.2 The quotient and remainder of the division performed at that stage of calculating the hcf;
.3 A pair of numbers, to be calculated later.

Now run backwards along the sequence, putting values into the third pair of numbers. An initial value is put into an element near the end. A calculated value is put into the preceding elements in turn until reaching the beginning of the sequence.

Finally, derive the required inverse from the values now in the first element of the sequence.

4.2 The inv_mod algorithm : Details

Form a sequence of records, each holding six values :

a : as in the hcf algorithm
b : as in the hcf algorithm
q : quotient where $a = q{\cdot}b + r$ with $0 \le r < b$
r : remainder where $a = q{\cdot}b + r$ with $0 \le r < b$
c : value that is propagated backwards; resembles a
d : value that is propagated backwards; resembles b

.1 Forward calculation

Construct the first element of the sequence :

$a_0 = Num$
$b_0 = Base$

Add successor elements :

$a_{n+1} = b_n$
$b_{n+1} = a_n \% b_n$

until the last element has been reached, the element where to continue would mean dividing by zero.

At the last element :

$a_m = h$ for some h
$b_m = 0$ for some m (specifically, the least m such that $b_m = 0$)

Ensure that at each element the values q_n, r_n obey :

for $0 \le n < m$
$a_n = q_n{\cdot}b_n + r_n$ with $0 \le r_n < b_n$
(i.e the quotient q and remainder r on dividing a_n by b_n)

and

for $n = m$
q_m = "don't care"
r_m = "don't care"

There can be a check on the three pre-conditions during this process :

At the beginning check that

$Base > 1$, i.e that $b_0 > 1$

and that

$Num \% Base \neq 0$, i.e that $r_0 \neq 0$.

At the end check that

$\text{hcf}(Num, Base) = 1$, i.e that $a_m = h = 1$.

.2 Backward calculation

Ensure that at each element, c and d obey :

at the last two elements :

$c_{m-1} = c_m =$ "don't care"
$d_{m-1} = d_m =$ "don't care"

at the last but two element :

$c_{m-2} = q_{m-2}$
$d_{m-2} = (-1)$

and for all other elements :

for $0 \leq n < m-2$
$c_n = q_n \cdot (-c_{n+1}) + d_{n+1}$
$d_n = c_{n+1}$

Note : Remember the definition of $(-x)$ in 4.1, Symbols .2.

.3 The result is :

$\text{Inv\_mod}(Num, Base)$
$= Inv$
$= (Base - (d_0 \% Base)) \% Base$

I.e The result is $(-d_0)$ which is then adjusted if necessary so $0 \leq Inv < Base$.
(However, *Inv* won't be 0 as this is precluded by the pre-conditions).

Remark 1 : Inconvenient sub-expressions

To avoid negative numbers or too-large numbers it is valid to replace any sub-expression, *exp*, in the calculation with *exp* % *Base* , see 2.4.1.

Remark 2 : Compare a, b with c, d

Consider the elements of the sequence :

$a_n = q_n \cdot b_n + (a_n \% b_n)$ from the definition of q_n

so

$(a_n \% b_n) = a_n - q_n \cdot b_n$ This subtraction is always defined, see 3.5 item .1 B

Now

$a_{n+1} = b_n$ hcf algorithm

$b_{n+1} = a_n \% b_n$ hcf algorithm

$= a_n - q_n \cdot b_n$

$= a_n - q_n \cdot a_{n+1}$

so

$a_n = q_n \cdot a_{n+1} + b_{n+1}$

$b_n = a_{n+1}$

Notice the similarity of c, d to a, b.

4.3 Proof that hcf(*Num*, *Base*) = 1 is a necessary requirement

Consider two numbers, a and B, both positive, with the common factor c. 1 is a factor of both a and B so c does exist.

Suppose that there is a number, i, that is the multiplicative inverse of a with respect to B, namely

$\text{mod}(a \cdot i, B) = 1$

That is

for some q

$a \cdot i = q \cdot B + 1$ Definition of mod

so

$a \cdot i - q \cdot B = 1$

This subtraction is defined even if negative numbers are not allowed.

Both a and B have the common factor c, so

for some a', b'

$a = a' \cdot c$

$B = b' \cdot c$

Thus

$a' \cdot c \cdot i - q \cdot b' \cdot c = 1$

so

$$c \cdot (a' \cdot i - q \cdot b') = 1$$

The numbers have integral values of one kind or another, so there cannot be a solution for *i* unless the left hand side of the equation evaluates to

$$1 \times 1$$

That is, for any *a* and *B*,

$$c = 1$$

is a necessary requirement if *a* is to have an inverse with respect to *B*.

c is a factor of both *a* and *B*, $c = 1$, and cannot be greater than 1, so 1 is the highest common factor. That is

$$\text{hcf}(a, B) = 1.$$

QED

The correctness of the algorithm will then prove that the pre-conditions are sufficient requirements.

4.4 Proof that the sequence always has an element m - 2

Consider two numbers, *Num* and *Base*, that obey the pre-conditions given in section 4.1. The algorithm constructs a sequence of records starting at element 0 and finishing at element *m*.

Element 0 contains the value

$(a_0, b_0) = (Num, Base)$ By construction, see 4.2 item .1

Element *m* contains the values

$(a_m, b_m) = (h, 0)$ where $h = \text{hcf}(Num, Base)$

By construction, see 4.2 item .1

Thus

$$b_0 = Base$$

$$b_m = 0$$

but by a pre-condition on *Base*

$Base > 0$ By 4.1 Precondition .1

so

$$b_0 \neq b_m$$

and

$$m \neq 0$$

Therefore $m > 0$ so element m-1 exists and contains the values

$(a_{m-1}, b_{m-1}) = (q_{m-1} \cdot h, h)$ By 4.2 Remark 2

but

$h = 1$ By 4.1 Precondition .3

so

$(a_{m-1}, b_{m-1}) = (q_{m-1}, 1)$

Thus

$b_0 = Base$

$b_{m-1} = 1$

but by a pre-condition on *Base*

$Base > 1$ By 4.1 Precondition .1

so

$b_0 \neq b_{m-1}$

and

$m \neq 1$

Therefore $m > 1$ so element m-2 exists.

QED

4.5 Proof that the result is always correct : for Integers

The desired inverse is $(-d_0)$ % *Base*. Some details of the proof of correctness depend on the precise definition of "$(-d_0)$". It is convenient to give separate proofs for Integers and Natural Numbers.

This section contains the proof for the case when negative numbers are allowed – the Integers.

Reminder : When negative numbers are allowed the $(-x)$ symbol is defined in section 4.1 Symbol Definitions .2 to mean $-x$.

Remember that

$Base > 1$ By 4.1 Precondition .1

so 1 % *Base* = 1 throughout these proofs

Start by proving a general intermediate result**.**

.1 To prove :

for all n **such that** $0 \leq n \leq m-2$

$a_n \cdot (-d_n) + b_n \cdot (-c_n) = 1$ when negative numbers are allowed

.1.1 To prove :

$a_{m-2}\cdot(-d_{m-2}) + b_{m-2}\cdot(-c_{m-2}) = 1$

a_m
$= h$ By construction, see 4.2 item .1
$= 1$ By 4.1 Precondition .3

b_m
$= 0$ By construction, see 4.2 item .1

a_{m-1}
$= q_{m-1}\cdot a_m + b_m$ From 4.2 item .3 Remark 2
$= q_{m-1}$

b_{m-1}
$= a_m$ From 4.2 item .3 Remark 2
$= 1$

a_{m-2}
$= q_{m-2}\cdot a_{m-1} + b_{m-1}$ From 4.2 item .3 Remark 2
$= q_{m-2}\cdot q_{m-1} + 1$

b_{m-2}
$= a_{m-1}$ From 4.2 item .3 Remark 2
$= q_{m-1}$

c_{m-2}
$= q_{m-2}$ By construction, see 4.2 item .2

d_{m-2}
$= (-1)$ By construction, see 4.2 item .2

Therefore

$a_{m-2}\cdot(-d_{m-2}) + b_{m-2}\cdot(-c_{m-2})$
$= (q_{m-2}\cdot q_{m-1} + 1)\cdot(-(-1)) + q_{m-1}\cdot(-q_{m-2})$
$= q_{m-2}\cdot q_{m-1} + 1 - q_{m-1}\cdot q_{m-2}$
$= 1$

QED

.1.2 To prove :

for all n **such that** $0 \le n < m-2$

$a_n\cdot(-d_n) + b_n\cdot(-c_n) = a_{n+1}\cdot(-d_{n+1}) + b_{n+1}\cdot(-c_{n+1})$

Expand

$a_n \cdot (-d_n) + b_n \cdot (-c_n)$

$= (q_n \cdot a_{n+1} + b_{n+1}) \cdot (-d_n) + a_{n+1} \cdot (-c_n)$ By 4.2 Remark 2

$= (q_n \cdot a_{n+1} + b_{n+1}) \cdot (-c_{n+1}) + a_{n+1} \cdot (-(q_n \cdot (-c_{n+1}) + d_{n+1}))$

By construction see 4.2 item .2

$= a_{n+1} \cdot q_n \cdot (-c_{n+1}) + b_{n+1} \cdot (-c_{n+1})$

$- a_{n+1} \cdot q_n \cdot (-c_{n+1}) + a_{n+1} \cdot (- d_{n+1})$

$= a_{n+1} \cdot (-d_{n+1}) + b_{n+1} \cdot (-c_{n+1})$

QED

.1.3 Therefore by .1.1, .1.2 and induction

for all n **such that** $0 \le n \le m-2$

$a_n \cdot (-d_n) + b_n \cdot (-c_n) = 1$

QED

Now prove correctness.

.2 To prove :

$(-d_0)$ % *Base* is the (tidied up) multiplicative inverse of *Num* w.r.t. *Base*

That is, that

$\text{mod}(Num \times (-d_0), Base) = 1$

or, equally, that

$Num \cdot (-d_0)$ % $Base = 1$

Proof :

$a_0 \cdot (-d_0) + b_0 \cdot (-c_0) = 1$ By .1 when $n = 0$

so

$[a_0 \cdot (-d_0) + b_0 \cdot (-c_0)]$ % *Base*

$= 1$ % *Base*

$= 1$ By 2.5 item .1

as $Base > 1$ by 4.1 Pre-condition .1

But

$a_0 = Num$ By construction, see 4.2 item .1

$b_0 = Base$ By construction, see 4.2 item.1

So

$[Num \cdot (-d_0) + Base \cdot (-c_0)]$ % $Base = 1$

hence

$Num \cdot (-d_0)$ % $Base = 1$ By 2.5 item .3

Conclusion :

The (tidied up) multiplicative inverse of *Num* with respect to *Base*
= $(-d_0)$ % *Base*
= (*Base* - (d_0 % *Base*)) % *Base*   By definition of $(-x)$, 4.1 Symbols .2

QED

Note : By 2.4.1 this result still holds if some or all of the intermediate results when calculating c_n and d_n are reduced modulo *Base*.

.3 Incidental proof

if *Num* > 1
then $(-c_0)$ is the multiplicative inverse of *Base* w.r.t. *Num*

Remark 1 :

$Num \neq 0$ by 4.1 Pre-condition .2;

if *Num* < 0 **then** *Base* % *Num* is not defined;

if *Num* = 1 **then** *Base* % *Num* = 0 so *Base* has no inverse.

Remark 2 : To confirm :

Num > 1 here;
hcf(*Base*, *Num*) = hcf(*Num*, *Base*) = 1;
Base % $Num \neq 0$ By 2.5 item .6 (remembering *Base* > 1)

The inv_mod pre-conditions are met

so

inv_mod(*Base*, *Num*) is defined

Reminder :

"Numbers" here are Integers so $(-c_0)$ means $-c_0$ and $(-d_0)$ means $-d_0$

Proof

if $Num \leq 1$
then there is nothing to prove
otherwise :

$a_0 \cdot (-d_0) + b_0 \cdot (-c_0) = 1$ By .1

so, as in the proof of .2 but with % *Num* instead of % *Base*,

$[Num\cdot(-d_0) + Base\cdot(-c_0)] \% Num = 1$

$[Num\cdot(-d_0) \% Num + Base\cdot(-c_0)] \% Num = 1$

$[0 + Base\cdot(-c_0)] \% Num = 1$

hence

$Base\cdot(-c_0) \% Num = 1$

Conclusion :

Provided *Num* > 1,
the (tidied up) multiplicative inverse of *Base* with respect to *Num*
$= (-c_0) \% Num$
$= (Num - (c_0 \% Num)) \% Num$

QED

Warning : Unlike **.2**, this result **.3** does **not** hold if intermediate results are reduced modulo *Base*.

On the other hand, provided *Num* > 1, then both **.2** and **.3** do still hold if intermediate results are reduced modulo *Num·Base* (by 2.5 item .5).

4.6 Proof that the result is always correct : for Natural Numbers

The inverse is $(-d_0) \% Base$. Some details of the proof of correctness depend on the precise definition of "$(-d_0)$". It is convenient to give separate proofs for Integers and for Natural Numbers.

This section contains the proof for the case when negative numbers are not allowed – the Natural Numbers.

Reminder : When negative numbers are not allowed the $(-x)$ symbol is defined in 4.1 Symbol .2 to mean $[B - (x \% B)] \% B$, with $B = Base$ in this context.

By the definition of % in 2.4 we have $0 \le (x \% B) < B$ so the subtraction is always defined.

Note : Care must be taken that no step in the proofs appears to subtract a larger number from a smaller one.

Remember that

Base > 1 By 4.1 Precondition .1

so 1 % *Base* = 1 throughout these proofs

Start by proving some general results concerning (-*x*).

.1 To prove :

$(-x) \% Base = (-x)$

$(-x) = [Base - (x \% Base)] \% Base$ Definition of (-*x*) in 4.1 Symbols .2

so

$(-x) \% Base$

$= [Base - (x \% Base)] \% Base \% Base$

$= [Base - (x \% Base)] \% Base$ By 2.5 item .2

$= (-x)$ Definition of (-*x*) in 4.1 Symbols .2

QED

.2 To prove :

$(x + (-x)) \% Base = 0$

$(x + (-x)) \% Base$

$= [x + (Base - x \% Base) \% Base] \% Base$

Definition of (-*x*) in 4.1 Symbols .2

$= 0$ By 2.5 item .4

QED

.3 To prove :

$(-(-x)) = x \% Base$

Case 1 : $x \% Base = 0$

$(-(-x))$

$= [Base - (-x) \% Base] \% Base$ Definition of (-*x*) in 4.1 Symbols .2

$= [Base - (Base - x \% Base) \% Base \% Base] \% Base$

Definition of (-*x*) in 4.1 Symbols .2

$= [Base - (Base - 0) \% Base] \% Base$

By this case and 2.5 item .2

$= [Base - 0] \% Base$ Definition of % in 2.4

$= 0$ Definition of % in 2.4

$= x \% Base$ This case

Case 2 : $0 < x \% Base < Base$

$(-(-x))$
$= [Base - (-x) \% Base] \% Base$ Definition of $(-x)$ in 4.1 Symbols .2
$= [Base - (Base - x \% Base) \% Base \% Base] \% Base$
Definition of $(-x)$ in 4.1 Symbols .2
$= [Base - (Base - x \% Base)] \% Base$ By 2.5 item .2, twice
$= [Base + (x \% Base - x \% Base) - (Base - x \% Base)] \% Base$
$= [x \% Base + (Base - x \% Base) - (Base - x \% Base)] \% Base$
$= [x \% Base] \% Base$
$= x \% Base$ By 2.5 item .2

Therefore, by Case 1 and Case 2
$(-(-x)) = x \% Base$

QED

.4 To prove :
$(-(x + y)) = [(-x) + (-y)] \% Base$

Let
$x = q_x \cdot Base + r_x$ for some q_x, r_x with $0 \le r_x < Base$
$y = q_y \cdot Base + r_y$ for some q_y, r_y with $0 \le r_y < Base$
and
$(r_x + r_y) = q \cdot Base + r$ for some q, r with $0 \le r < Base$

We have
$q \cdot Base \le (r_x + r_y) < (Base + Base)$
so
$q < (1 + 1)$
$q \le 1$

LHS
$(-(x + y))$
$= [Base - (x + y) \% Base] \% Base$ Definition of $(-x)$ in 4.1 Symbols .2
$= [Base - (q_x \cdot Base + r_x + q_y \cdot Base + r_y) \% Base] \% Base$
$= [Base - (r_x + r_y) \% Base] \% Base$ By 2.5 item .3
$= [Base - r] \% Base$ Definition of r above and of % in 2.4

RHS
$[(-x) + (-y)] \% Base$
$= [(Base - x \% Base) \% Base + (Base - y \% Base) \% Base] \% Base$
Definition of $(-x)$ in 4.1 Symbols .2
$= [(Base - x \% Base) + (Base - y \% Base)] \% Base$ By 2.4.1

$= [\ (Base - (q_x \cdot Base + r_x)\ \%\ Base) +$
$(Base - (q_y \cdot Base + r_y)\ \%\ Base)\ ]\ \%\ Base$ — As above
$= [\ (Base - r_x) + (Base - r_y)\ ]\ \%\ Base$ — Definition of % in 2.4
$= [\ (Base + Base) - (r_x + r_y)\ ]\ \%\ Base$
$= [\ (Base + Base) - (q \cdot Base + r)\ ]\ \%\ Base$ — Definition of q, r
$= [\ (1 - q) \cdot Base + (Base - r)\ ]\ \%\ Base$ — $q \leq 1$, $r < Base$
$= [Base - r]\ \%\ Base$ — By 2.5 item .3

so LHS = RHS

QED

Now prove that the result of the algorithm is always correct.
Start by proving a general intermediate result.

.5 To prove :
for all n **such that** $0 \leq n \leq m\text{-}2$
$[a_n \cdot (-d_n) + b_n \cdot (-c_n)]\ \%\ Base = 1$ when negative numbers are not allowed

.5.1 To prove :
$[a_{m-2} \cdot (-d_{m-2}) + b_{m-2} \cdot (-c_{m-2})]\ \%\ Base = 1$

Remember that $1 < Base$ by 4.1 Precondition .1.

a_m
$= h$ — By construction, see 4.2 item .1
$= 1$ — By 4.1 Precondition .3

b_m
$= 0$ — By construction, see 4.2 item .1

a_{m-1}
$= q_{m-1} \cdot a_m + b_m$ — From 4.2 item .3 Remark 2
$= q_{m-1}$

b_{m-1}
$= a_m$ — From 4.2 item .3 Remark 2
$= 1$

a_{m-2}
$= q_{m-2} \cdot a_{m-1} + b_{m-1}$ — From 4.2 item .3 Remark 2
$= q_{m-2} \cdot q_{m-1} + 1$

$b_{m\text{-}2}$
$= a_{m\text{-}1}$ From 4.2 item .3 Remark 2
$= q_{m\text{-}1}$

$c_{m\text{-}2}$
$= q_{m\text{-}2}$ By construction, see 4.2 item .2

$d_{m\text{-}2}$
$= (-1)$ By construction, see 4.2 item .2

Therefore

$[a_{m\text{-}2}\cdot(-d_{m\text{-}2}) + b_{m\text{-}2}\cdot(-c_{m\text{-}2})]\ \%\ Base$
$= [(q_{m\text{-}2}\cdot q_{m\text{-}1} + 1)\cdot(-(-1)) + q_{m\text{-}1}\cdot(-q_{m\text{-}2})]\ \%\ Base$
$= [(q_{m\text{-}2}\cdot q_{m\text{-}1} + 1)\cdot(1\ \%\ Base) + q_{m\text{-}1}\cdot(-q_{m\text{-}2})]\ \%\ Base$ By .3
$= [(q_{m\text{-}2}\cdot q_{m\text{-}1} + 1)\cdot 1 + q_{m\text{-}1}\cdot(-q_{m\text{-}2})\ ]\ \%\ Base$ By 2.5 item .1
$= [1 + q_{m\text{-}1}\cdot(\ q_{m\text{-}2} + (-q_{m\text{-}2}))\ ]\ \%\ Base$
$= [1 + q_{m\text{-}1}\cdot[(\ q_{m\text{-}2} + (-q_{m\text{-}2}))\ \%\ Base]\ ]\ \%\ Base$ By 2.4.1, thrice
$= [1 + 0]\ \%\ Base$ By .2
$= 1$ By 2.5 item .1

QED

.5.2 To prove :

for all n **such that** $0 \le n < m\text{-}2$

$[a_n\cdot(-d_n) + b_n\cdot(-c_n)]\ \%\ Base = [a_{n+1}\cdot(-d_{n+1}) + b_{n+1}\cdot(-c_{n+1})]\ \%\ Base$

Expand

$[a_n\cdot(-d_n) + b_n\cdot(-c_n)]\ \%\ Base$
$= [(q_n\cdot a_{n+1} + b_{n+1})\cdot(-d_n) + a_{n+1}\cdot(-c_n)\ ]\ \%\ Base$ By 4.2 item .3 Remark 2
$= [a_{n+1}\cdot[q_n\cdot(-d_n) + (-c_n)] + b_{n+1}\cdot(-d_n)]\ \%\ Base$
$= [a_{n+1}\cdot[q_n\cdot(-\ c_{n+1}) + [\ (-(q_n\cdot(-c_{n+1}) + d_{n+1}))\ ]\ ] + b_{n+1}\cdot(-c_{n+1})]\ \%\ Base$
By 4.2 item .2
$= [a_{n+1}\cdot[q_n\cdot(-\ c_{n+1}) + [\ (-(q_n\cdot(-c_{n+1}))) + (-d_{n+1})\ ]\ \%\ Base]$
$+ b_{n+1}\cdot(-c_{n+1})]\ \%\ Base$ By .4
$= [a_{n+1}\cdot[\ [q_n\cdot(-\ c_{n+1}) + (-(q_n\cdot(-c_{n+1})))]\ \%\ Base + (-d_{n+1})]$
$+ b_{n+1}\cdot(-c_{n+1})]\ \%\ Base$ By 2.4.1, twice
$= [a_{n+1}\cdot[0 + (-d_{n+1})] + b_{n+1}\cdot(-c_{n+1})]\ \%\ Base$ By .2
$= [a_{n+1}\cdot(-d_{n+1}) + b_{n+1}\cdot(-c_{n+1})]\ \%\ Base$

QED

.5.3 **Therefore** by .5.1, .5.2, and induction,
for all n **such that** $0 \le n \le m\text{-}2$
$[a_n \cdot (-d_n) + b_n \cdot (-c_n)] \ \% \ Base = 1$

QED

Finally prove correctness.

.6 **To prove** : $(-d_0)$ is the multiplicative inverse of *Num* w.r.t. *Base*

That is, that
mod($Num \times (-d_0)$, *Base*) = 1 or, equally, that
$Num \cdot (-d_0) \ \% \ Base = 1$

$[\ a_0 \cdot (-d_0) + b_0 \cdot (-c_0)\ ] \ \% \ Base = 1$ By .5 when $n = 0$

but
$a_0 = Num$ and
$b_0 = Base$

so
$[\ Num \cdot (-d_0) + Base \cdot (-c_0)\ ] \ \% \ Base = 1$

hence
$Num \cdot (-d_0) \ \% \ Base = 1$ By 2.5 item .3

Conclusion :
The (tidied up) multiplicative inverse of *Num* with respect to *Base*
$= (-d_0) \ \% \ Base$
$= (Base - (d_0 \ \% \ Base)) \ \% \ Base$

QED

Note : By 2.4.1 this result still holds if some or all of the intermediate results when calculating c_n and d_n are reduced modulo *Base*.

4.7 Proof that the result is unique

Remember that
$Base > 1$ By 4.1 Precondition .1
so $1 \ \% \ Base = 1$ throughout this proof

Assume that t and u are both inverses, so that
$Num \cdot t \ \% \ Base = 1$ **with** $0 < t < Base$
$Num \cdot u \ \% \ Base = 1$ **and** $0 < u < Base$

$Num \cdot t \,\%\, Base = 1 = Num \cdot u \,\%\, Base$
$Num \cdot t \,\%\, Base = Num \cdot u \,\%\, Base$

so

$Num \cdot t \cdot t \,\%\, Base = Num \cdot t \cdot u \,\%\, Base$	By 2.5 item .7
$(Num \cdot t \,\%\, Base) \cdot t \,\%\, Base = (Num \cdot t \,\%\, Base) \cdot u \,\%\, Base$	By 2.4.1
$1 \cdot t \,\%\, Base = 1 \cdot u$ % Base	Definition of t
$t = u$	By 2.5 item .1 and definitions of t, u

The (tidied up) multiplicative inverse is unique (and it exists when the pre-conditions are true).

QED

4.8 Some observations

.1 *Base* = 1

Base = 1 is a peculiar case but it is well defined and so need not be rejected. When *Base* = 1, $n \,\%\, Base = 0$ so n has no inverse for any n.

.2 *Num* = 1

One case can be implemented immediately without executing the algorithm :

if *Base* > 1 **and** *Num* = 1 **then** the inverse is 1.

.3 **Minimum sequence size**

The minimum length of the element sequence is 3.
E.g when *Num* = 1 and *Base* = 2 the sequence goes
$(1, 2) \to (2, 1) \to (1, 0)$.

.4 **Division in modular arithmetic**

If *Base* is known and fixed in a particular context then the inverse of *Num* can be written as $1/Num$ or Num^{-1}. In general, the division a/b can then be defined to be $a \times (1/b)$ or $a \times b^{-1}$ (not defined when $b = 0$).

.5 All *Num* and *Base* values

As the inverse can never be zero then, as in the hcf function, zero can be used to indicate that *Num* or *Base* does not meet the pre-conditions. With this rule the inv_mod function can be extended to all numbers.

The End