Gerald’s Column
by Gerald Fitton

In my last couple of articles I introduced you to what I called Modulo Arithmetic and showed you how it might be used to provide a simple encryption system (one based on multiplication followed by applying the modulus function). As a result I have had a large amount of correspondence about both these subjects. I shall continue this month by describing some Rules for Modulus Arithmetic and Algebra. In next month’s issue I shall use Modulus functions to demonstrate the renowned RSA Public Key Encryption System.

Modulus or Modulo

Colin Singleton, who is a member of the Editorial Board of the *Journal of Recreational Mathematics* has written many letters to me and I have found his correspondence most helpful. However, he has politely taken me to task for using the word Modulo in places where I should have used Modulus. I should have remembered the (little) Latin I did at school. The word Modulo is the Latin ‘Ablative’ case and so it means “with reference to the modulus”. I should have written Modulus Arithmetic and Modulus Algebra! As soon as I read Colin’s letter I knew I had made a mistake! The use of Modulo is correct in phrases such as 15 Modulo 12 is 3 but not in Modulo Algebra.

So let’s get on with a bit of Modulus Arithmetic.

Modulus Arithmetic

To use the modern encryption systems which rely on Modulus Arithmetic we need to be able to evaluate expressions such as (a mod b) when a is a large number. For example we might want to find the remainder when a million or a billion, or even a number with over a hundred digits is divided by another large number. There are three basic ‘Rules’ which can be used to speed up the evaluation of (a mod b) when a is large. The Rules are generally known as the Addition Rule, the Multiplication Rule and the Power Rule (this latter is sometimes known as the Exponentiation Rule).

You can regard multiplication as continued addition. By this I mean that if you want to find 5*3 you can calculate the result of adding together five lots of 3. As an equation we can write 5*3 = 3 + 3 + 3 + 3 + 3. The Rules of multiplication rely upon the validity of the Rules which apply to addition. In its turn the Rules for powers rely upon the Rules for multiplication. So, if we can get to grips with addition then the rest will follow.

The Addition Rule

Equal values can be added or subtracted from an equation such as (b mod m) = (a mod m). For example if (b mod m) = (a mod m) then (b + c) mod m = (a + c) mod m.

As my first worked example I shall consider a 24 hour clock with a = 15 and b = 39. It is easy to show that (39 mod 24) evaluates to 15. You can think of this as: “39 hours after midnight will be 15 hours (3 o’clock in the afternoon)”. The mathematical equation can be written as: (39 mod 24) = 15 or a little more pedantically as (39 mod 24) = (15 mod 24).

Any number can be added to (or subtracted from) both sides of this equation (within the modulus) and the resulting equation will still be true. For example we can add 3000 hours to each side of the equation and we can rest assured that (3039 mod 24) = (3015 mod 24).

Let's look at another example. We shall evaluate (600 mod 24) by adding together six lots of (100 mod 24). The number 100 = 4* 24 + 4 so that (100 mod 24) = 4. or, to be pedantic, (100 mod 24) = (4 mod 24).

The 'clever bit' in the next step is to realise that (200 mod 24) can be split into two lots of (100 mod 24). The arithmetic is 200 = 100 + 100 = (4*24 + 4) + (4*24 + 4). Applying the modulus function to both sides we have (200 mod 24) = ((0 + 4) + (0 + 4)) mod 24 = 8. If you are at all unsure of what has happened here then please read it again until you convince yourself that it is not 'magic' but 'obvious'. Note (n*24) mod 24 = 0 for all n.

[Note to Editor:
Can the following paragraph be included as a table spreading across the whole page? If not then can you arrange the lines with an = sign at the start of each row?]

As an equation (200 mod 24) = (100 mod 24) + (100 mod 24) = (4 + 4) mod 24 = 8.
In the same way we can add another 100 to find that (300 mod 24) = 8 + 4 = 12.
Eventually we arrive at (600 mod 24).

Can you see that this must evaluate to six lots of 4 namely 6*4 = 24?
What is (24 mod 24)? Answer: the 'magic' zero. We'll see why it is 'magic' quite soon!
To summarise, we have shown that (600 mod 24) = 0 using the Addition Rule.

Back to my first example. What is the value of (3039 mod 24)? Well this must be the same as (3000 mod 24) + (39 mod 24). We know that (39 mod 24) = 15 but how can we evaluate (3000 mod 24) more easily than doing a long division?

We know that (600 mod 24) = 0. The 'magic' zero allows us to replace multiples of (600 mod 24) with 0 so that any number of 600s can be added together and the modulus will always be zero. The number 3000 is five lots of 600 so (3000 mod 24) = 0.

Hence (3039 mod 24) = (3000 mod 24) + (39 mod 24) = 0 +(15 mod 24) = 15.

The Addition Rule can be useful if you want to find the the answer to arithmetic questions such as: "What is the value of (a million mod 24)?". However, it is not as useful as the Multiplication Rule! Using the Multiplication Rule (next section) we shall discover that (10^n) mod 24 = 16 for all values of n greater than . . . well, you'll have to wait and see!

The Multiplication Rule

Each side of an equation such as (b mod m) = (a mod m) can be multiplied by a constant within the modulus operation. For example it follows that (b*c) mod m = (a*c) mod m.

Let us continue with the example of the 24 hour clock. Suppose we want to evaluate 1039 mod 24. First we can use the Addition Rule to split the 1039 into 1000 + 39 and then evaluate (1000 mod 24) and (39 mod 24) separately.

All we need to work out is the value of (1000 mod 24). But why stop there. Let’s get really ambitious and see if we can evaluate 1000000000000000039 mod 24. Certainly you can’t divide that number by 24 with a pocket calculator!

Our starting point for evaluating this number is that (10 mod 24) = (10 mod 24).

[Note to Editor: Perhaps this can be laid out as a table spreading across the whole page?]

Multiply both sides by 10 to get	100 mod 24 = 100 mod 24 = 4.
Multiply by 10 again and we have	1000 mod 24 = 40 mod 24 = 16.
Multiply by 10 again and we have	10000 mod 24 = 160 mod 24 = 16.
Multiply by 10 again and we have	100000 mod 24 = 160 mod 24 = 16.
Multiply by 10 again and we have	1000000 mod 24 = 160 mod 24 = 16.
Multiply by 10 again and we have	10000000 mod 24 = 160 mod 24 = 16.

I’m sure that you get the picture. The modulus of the rather large number which I can describe as (10^n+39) mod 24 is (16+39) mod 24 = (55 mod 24) = 7. The time a million and 39, or even a billion and 39 hours from midnight will be seven o’clock in the morning!

As I have said, modern encryption systems require the evaluation of the modulus of very large numbers. The Multiplication Rule is useful for finding the modulus of such numbers but it is not as useful as the Power Rule—patience and all will be revealed!

There is no ‘Division Rule’. You can not apply division to both sides of a modulus equation. Division doesn’t exist. However, as we saw in previous issues of Archive, often we can find inverse multipliers which ‘undo’ (many) multiplication operations.

The Multiplicative Inverse

Before we move on to the Power Rule let’s do a bit of Modulus Algebra. The simple encoding system of the previous two issues of Archive can be described mathematically as follows. We choose an ASCII code such as 71 (for “G”), let’s call it x. We multiply x by our ‘multiplier’, m, to find (m*x) and then find the remainder using our modulus 256. We can write this as y = (m*x) mod 256 where y is the encoded form of x.

Suppose the decoding multiplier is n. The decoding process is represented by the equation x = (n*y) mod 256. Substitute (m*x) mod 256 for y in this equation and we get
x = (n*m*x) mod 256. Now to the ‘difficult bit’. Can we prove this inverse ‘works’?

One consequence of the Multiplication Rule is that it doesn’t matter in what order you do the multiplications nor does it matter when you apply the modulus function. In ordinary arithmetic we can either multiply the m and x first or we can multiply the n and m first! Using algebraic notation this can be written as: (n*m*x) = n*(m*x) = (n*m)*x.

The Multiplication Rule for Modulus Arithmetic leads to the following piece of Algebra
(n*m*x mod 256) = (n mod 256)*(m*x mod 256) = (n*m mod 256)*(x mod 256).

The inverse multiplier, n, was chosen such that (n*m) mod 256 = 1 so that the right hand side of this equation reduces to 1 * (x mod 256) = x. We have proved that if we choose the inverse multiplier so that (n*m) mod 256 = 1 then n will ‘undo’ the encoding of m.

Last month I included a spreadsheet, [Inv_P], which finds the multiplicative inverse of any (valid) number relative to any modulus. On this month's disc you will find the same algorithm as a PipeDream (and Fireworkz) custom function, [c_Inv], which you can use in your own spreadsheets. There are [TestInv] files which call the [c_Inv] functions so you can enter your own numbers and see it working.

If you choose a number for your Public Key multiplier which is unsuitable (see later in this article) then the error message "Bad Choice" will be returned. The function [c_Inv] is used as part of the process for setting up the famous Public Key Encryption System.

Look back three paragraphs and you'll see the equation (n*m) mod 256 = 1 used to find a multiplicative inverse. This is not its only significance as we shall see in the next section.

The Power Rule

Each side of an equation such as (b mod m) = (a mod m) can be raised to the same power within the modulus operation so that (b^c) mod m = (a^c) mod m.

Let's consider evaluating (5^100) mod 24. The 'hard way' would be to keep raising 5 one power at a time until we reached the massive number corresponding to 5^100. Then we could start dividing by 24 until ultimately we arrive at a remainder.

The easier way is to use the Power Rule.

Start with 5 mod 24 = 5 and square both sides to get (5^2) mod 24 = (25 mod 24) = 1.

You will see that we have the 'magic' 1 on the right hand side of the equation. Raise both sides to the power 50 to get ((5^2)^50)) mod 24 = 1^50. The right hand side is easy to evaluate because 1 raised to any power is still 1. The left hand side can be rewritten as (5^(2*50)) mod 24 = (5^100) mod 24. We have found that (5^100) mod 24 = 1 with hardly any pain at all!.

Modern encryption systems use power functions rather than plain multiplication. Consequently they require a fast method of finding the value of expressions such as (x^e) mod m where e and m are large numbers, typically a hundred digits long! x is the value which encodes to y. The method of evaluation makes use of the Power Rule.

Here is a simple example of how it works. Suppose we want to evaluate 7^64 mod 13.

We start with	(7^2) mod 13 =	49 mod 13 =	10.
Then we square both sides to get	(7^4) mod 13 =	(10^2) mod 13 =	9.
Then we square both sides to get	(7^8) mod 13 =	(9^2) mod 13 =	3.
Then we square both sides to get	(7^16) mod 13 =	(3^2) mod 13 =	9.
Then we square both sides to get	(7^32) mod 13 =	(9^2) mod 13 =	3.
Then we square both sides to get	(7^64) mod 13 =	(3^2) mod 13 =	9.

[Note to Editor: Once again a table perhaps?]

I am indebted to Colin Singleton for a simple method of evaluation of (x^11 mod m). The method he suggested is to note that 11 = 8 + 2 + 1 and then 'pick off' the powers.

Using Colin's method the Multiplication Rule can be used to find, say, (7^11) mod 13 as: (7^(8+2+1)) mod 13 = (7^8) mod 13 * (7^2) mod 13 * (7^1) mod 13 = (3*10*7) mod 13.

(3*10) mod 13 = 4 hence (7^11) mod 13 = (4*7) mod 13 = (28 mod 13) = 2.

The combination of the Power and Multiplication Rules which I have just demonstrated is used in modern encryption systems to evaluate (x^e) mod m for large values of e and m. I have converted the algorithm to a PipeDream custom function, [c_Exp], which is on the monthly disc. I shall use it next month in my demonstration of the RSA System.

Highest Common Factor

I remember doing these at school when I was about nine years old.

I don't think they form part of the school curriculum any more. Highest Common Factors (HCF) are useful in encryption systems (and in modulus algebra) because they help us to discover whether we have chosen Public and Private Keys which lead to the essential one to one mapping between the numbers representing the plain text (the x values) and the numbers to which they encode (the y values).

The HCF of two numbers is the largest number (lower than both) which will divide into both numbers leaving no remainder. Using the modulus notation, a and b have the HCF h if both a mod h = 0 and b mod h = 0 and there is no larger number than h for which this is true. As an example the HCF of 20 and 50 is 10 because 10 is a factor of both 20 and 50 and there is no number larger than 10 which is a factor of both 20 and 50.

On the monthly disc you will find the PipeDream (and Fireworkz) custom function [c_HCF] which finds the HCF of two numbers. The algorithm which I use is one devised by the ex-patriot Ancient Greek mathematician, Euclid, who lived in Alexandria around 300 BC. It relies on the fact that the HCF of a and b (with a > b) is the same as the HCF of a and (a − b). (a − b) is smaller than a and smaller than b so the problem of finding the HCF is easier after the subtraction. Euclid's method is to repeat this subtraction process until an appropriate 'exit condition' is reached. The exit condition is that both numbers are the same. This final number is the HCF.

As an example let's find the HCF of a = 50 and b = 20. The value of (a − b) = 30. Next step; the difference between 30 and 20 is 10. Continue with this algorithm (replace the larger number with the difference) and you will eventually arrive at a situation when both numbers are 10. The HCF of 50 and 20 is 10. My custom function [c_HCF], which you'll find on the monthly disc, uses this continuous subtraction algorithm discovered by Euclid.

Relatively Prime

One (smaller) number is relatively prime to another if the small number does not divide exactly into the larger number. In nearly all encryption systems which use modulus arithmetic the Key must be chosen to be a number which is relatively prime to the modulus. A pair of numbers are relatively prime if their HCF is 1. If the Key is not relatively prime to the modulus then one to one correspondence of x and y does not exist and the encryption system fails.

The HCF of two numbers is found as an incidental part of my custom function [c_Inv] (on the monthly disc). If the HCF of the chosen multiplier and the modulus is not 1 then the custom function returns the error message "Bad Choice"!

Lowest Common Multiple

I remember doing these at school when I was about nine years old. I was a wizard at fractions so I mourn their passing! They have been replaced by 'Rotational Symmetry' which is a useful mathematical concept but not much use for encryption algorithms.

Finding the LCM of denominators of fractions is essential if you want to add or subtract them. I don't think much time is spent on fractions in today's curriculum. Everything is digital and decimal. We did sums much harder than "What is a half minus a third?"

One way of finding the LCM of a set of numbers is to completely factorize all of them and then build up the LCM from the prime factors. For our purposes we shall need the LCM of only two numbers and an easier method is multiply the two numbers together and divide the product by their HCF. Better still, divide the larger number by the HCF (it will always divide exactly) and then multiply by the smaller number. This method avoids finding the prime factors, always a slow and sometimes an impossibly time consuming job.

If the two numbers are, say 15 and 20 then the HCF is 5 and the LCM is 15*(20/5) = 60.

I have not included a separate custom function for finding the LCM because I shall be using the 'divide by the HCF' method in my exposition of the Public Key System.

Public Key Encryption System

You will have to wait until next month for a description of how all these modulus functions are used in the famous Public Key Encryption System now known as the RSA System devised in 1977 by the trio Ronald Rivest, Adi Shamir and Leonard Adleman of MIT . It first saw the light of day in the August 1977 issue of Scientific American. This month I shall limit myself to a brief description.

This is the method: Choose two prime numbers; let us call them p and q. Find m = p*q. This number, m, is one part of the Public Key. Also find L which is the LCM of (p − 1) and (q − 1). Find a number, e, such that the HCF of L and e is 1 (e and L are relatively prime). The number e is the second part of the Public Key. Find d, the multiplicative inverse of e relative to L. The number d satisfies the equation (d*e) mod L = 1. The Private Key consists of m and d.

The encoding function is y = (x^e) mod m and the decoding function is x = (y^d) mod m. The numbers e and m are published as the Public Key and the numbers d and m make up the corresponding Private Key.

On the monthly disc you will find the file [TestKey] which will allow you to set up your own pair of Keys (using the RSA System). It contains all sorts of error checks such as returning an error message if you choose a value for p or q which is not a prime number. This 'prime' check uses the custom function [c_Prime].

The size of the numbers which can be entered into my [TestKey] spreadsheet is limited by the precision available through the Floating Point Emulator. It is possible to develop programs and spreadsheets which use multiple precision arithmetic—but that's another story for another day! My [TestKey] spreadsheet does demonstrate the principles but is not as secure as an implementation which uses much larger numbers.

Other Files

Also on the monthly disc are files called [TestExp], [TestHcf], [TestInv] and [TestPrime]. These spreadsheets do little more than call the corresponding custom function and allow you to enter your own values so that you can test the custom functions. Although I developed all these files using PipeDream I have ported them to Fireworkz. The Fireworkz files will Load into Resultz (there's a copy of a demo version of Resultz on a recent Archive disc) and into Fireworkz Pro.

Finally

You can write to me at the address given in Paul's Fact File.

If you do need some help then please let me have an example file which you have created; it saves me so much work and reduces the chances of misunderstanding your problem. Return postage and a self addressed label will be appreciated.

I do give help with spreadsheets including Eureka, Schema, Fireworkz and PipeDream. I can even load files into Excel and Save them in a format which can be loaded into a RISC OS spreadsheet. A recent addition to my portfolio is Microsoft Works so I can help with these conversions too.