

# MAPLE Notes for Computer Algebra

Michael Monagan  
Department of Mathematics  
Simon Fraser University  
Updated May 2013.

```
> restart;
```

These notes are for Maple V Release 13. They are platform independent, i.e., they are the same for the Macintosh, PC, and Unix versions of Maple. These notes should be backwards compatible to Maple 9 and forwards compatible to Maple 17.

## ▼ Maple as a Calculator

Input of a numerical calculation uses +, -, \*, /, and ^ for addition, subtraction, multiplication, division, and exponentiation respectively.

```
> 1+2*3-2;
```

5

```
> 2^3;
```

8

```
> 120/105;
```

$\frac{8}{7}$

Because the input involved integers, not decimal numbers, Maple calculates the exact fraction when there is a division, automatically cancelling out the greatest common divisor (GCD). In this case the GCD is 15, which you can calculate specifically as

```
> igcd(120,105);
```

15

Observe that every command ends with a semicolon ; This is a grammatical requirement of Maple. If you forget, Maple will assume that the command is not complete. This allows you to break long commands across a line. For example

```
> 1+2*3/
```

```
> (2+3);
```

$\frac{11}{5}$

We are not going to use decimal numbers very much in this course. However, here is how you would do some decimal calculations. The presence of a decimal point . in a number means that the number is a decimal number and Maple will, by default, do all calculations to 10 decimal places.

```
> 120/105.0;
```

1.142857143

```
> sqrt(2.0);
```

1.414213562

```
> sqrt(2);
```

$$\sqrt{2}$$

Notice the difference caused by the presence of a decimal point in these examples. Now, if you have input an exact quantity, like the  $\sqrt{2}$  above, and you now want to get a numerical value to 3 decimal digits, use the `evalf` command to evaluate to floating point. Use the `%` character to refer to the previous Maple output.

```
> evalf(%,3);
```

$$1.41$$

To input a formula, just use a symbol, e.g.  $x$  and the arithmetic operators and functions known to Maple. For example, here is a quartic polynomial in  $x$ .

```
> x^4-3*x+2;
```

$$x^4 - 3x + 2$$

We are going to use this polynomial for a few calculations. We want to give it the name  $f$  so we can refer to it later. We do this using the assignment operation in Maple as follows

```
> f := x^4-3*x+2;
```

$$f := x^4 - 3x + 2$$

The name  $f$  is now a variable. It refers to the polynomial. Here is its value

```
> f;
```

$$x^4 - 3x + 2$$

To evaluate this as a function at the point  $x = 3$  use the `eval` command as follows

```
> eval(f,x=3);
```

$$74$$

The following commands differentiate  $f$  with respect to  $x$  and factor  $f$  into irreducible factors over the field of rational numbers.

```
> diff(f,x);
```

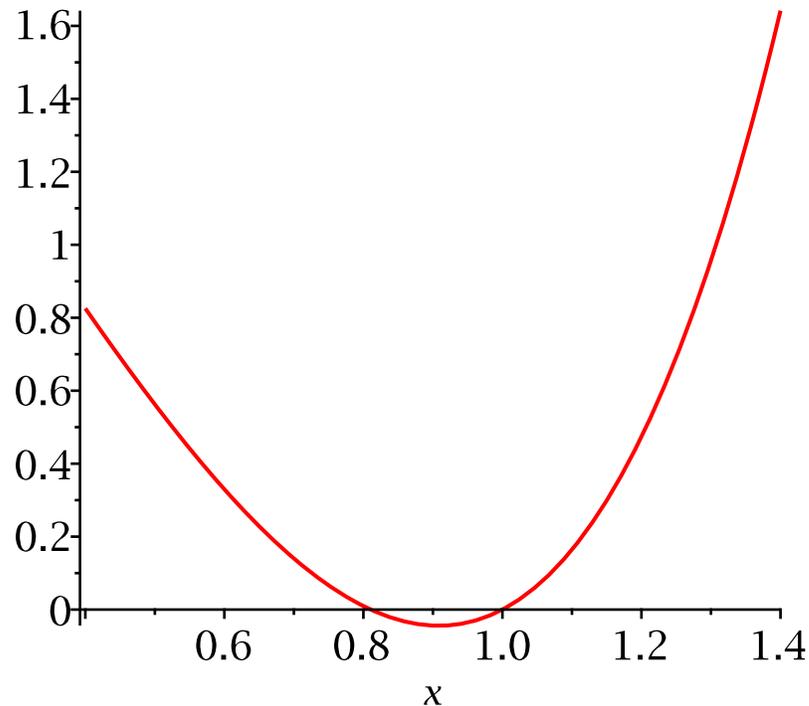
$$4x^3 - 3$$

```
> factor(f);
```

$$(x - 1)(x^3 + x^2 + x - 2)$$

You can graph functions using the plotting commands. The basic syntax for the `plot` command for a function of one variable is illustrated as follows:

```
> plot(f,x=0.4..1.4);
```



In the graph I can see a local minimum near  $x=0.9$ . We can find this point using calculus. The command `fsolve( f(x)=0, x )`, on input of a polynomial  $f(x)$  computes 10 digit numerical approximations for the real roots of  $f(x)$ .

```
> diff(f,x)=0;
```

$$4x^3 - 3 = 0$$

```
> fsolve(diff(f,x)=0,x);
```

$$0.9085602964$$

Here are some functions which we integrate to try Maple out.

```
> f := t*exp(-t);
```

$$f := te^{-t}$$

```
> integrate(f,t);
```

$$-(1+t)e^{-t}$$

Maple used integration by parts. Here is how you compute the definite integral

$$\int_0^{\infty} f(t) dt.$$

```
> int(f,t=0..infinity);
```

$$1$$

```
> g := sqrt(1-x^2);
```

$$g := \sqrt{1-x^2}$$

```
> int(g,x);
```

$$\frac{x\sqrt{1-x^2}}{2} + \frac{1}{2} \arcsin(x)$$

```
> h := x*exp(-x^2)/ln(1+x);
```

$$h := \frac{x e^{-x^2}}{\ln(1+x)}$$

```
> integrate(h,x);
```

$$\int \frac{x e^{-x^2}}{\ln(1+x)} dx$$

That means Maple could not integrate it. This means there is no elementary closed form for the integral.

```
> f := diff(g,x);
```

$$f := \frac{e^{-x^2}}{\ln(1+x)} - \frac{2x^2 e^{-x^2}}{\ln(1+x)} - \frac{x e^{-x^2}}{\ln(1+x)^2 (1+x)}$$

```
> f := simplify(f);
```

$$f := -\frac{e^{-x^2} (-\ln(1+x) - \ln(1+x)x + 2x^2 \ln(1+x) + 2x^3 \ln(1+x) + x)}{\ln(1+x)^2 (1+x)}$$

```
> integrate(f,x);
```

$$\frac{x e^{-x^2}}{\ln(1+x)}$$

We have used the name  $f$  as variable to refer to formulae and the symbols  $x$  for an unknown in a formula. Often you will have assigned to a name like we have done here to  $f$  but you want now to use the name  $f$  as a symbol again, not as a variable. You can unassign the value of a name as follows

```
> f;
```

$$-\frac{e^{-x^2} (-\ln(1+x) - \ln(1+x)x + 2x^2 \ln(1+x) + 2x^3 \ln(1+x) + x)}{\ln(1+x)^2 (1+x)}$$

```
> f := 'f';
```

$$f := f$$

```
> f;
```

$$f$$

## Calculating Sums and Solving Systems of Equations

The Maple command `sum(f(i), i=m..n)` calculates the sum  $\sum_{i=m}^n f(i)$ . Often you will want to simplify Maple's output. You can use the `expand`, `simplify`, `factor` and `combine` commands to do this.

```
> f := sum(i^2,i=1..n);
```

$$f := \frac{(n+1)^3}{3} - \frac{(n+1)^2}{2} + \frac{n}{6} + \frac{1}{6}$$

> expand(f);

$$\frac{1}{3} n^3 + \frac{1}{2} n^2 + \frac{1}{6} n$$

> factor(f);

$$\frac{n(n+1)(2n+1)}{6}$$

> f := sum( a^i, i=0..n-1 );

$$f := \frac{a^n}{a-1} - \frac{1}{a-1}$$

> simplify(f);

$$\frac{a^n - 1}{a - 1}$$

> f := sum( 2^k, k=0..n-1 );

$$f := 2^n - 1$$

> g := sum( k\*binomial(n,k), k=0..n );

$$g := \frac{2^n n}{2}$$

The Maple command solve( {equations}, {unknowns} ) solves a system of equations for the given unknowns. Here is an example of a linear system in two unknowns.

> sys := {x+y=2,x-y=1};

$$\text{sys} := \{x - y = 1, x + y = 2\}$$

> solve( sys, {x,y} );

$$\left\{ x = \frac{3}{2}, y = \frac{1}{2} \right\}$$

And a non-linear system is input similarly.

> sys := {x^2+y^2=1,x+y=1};

$$\text{sys} := \{x + y = 1, x^2 + y^2 = 1\}$$

> solve(sys,{x,y});

$$\{x = 1, y = 0\}, \{x = 0, y = 1\}$$

> sys := {x^2+y^2=2, x+y=1};

$$\text{sys} := \{x + y = 1, x^2 + y^2 = 2\}$$

> solve(sys,{x,y});

$$\{x = -\text{RootOf}(2\_Z^2 - 2\_Z - 1, \text{label} = \_L1) + 1, y = \text{RootOf}(2\_Z^2 - 2\_Z - 1, \text{label} = \_L1)\}$$

This is Maple's RootOf notation for the solutions. This is avoid generating large complicated radicals in the output. But in this case, the solutions can be expressed simply as radicals. Do this to get radicals.

```
> _EnvExplicit := true;
```

```
    _EnvExplicit := true
```

```
> solve(sys, {x,y});
```

$$\left\{ x = \frac{1}{2} - \frac{\sqrt{3}}{2}, y = \frac{1}{2} + \frac{\sqrt{3}}{2} \right\}, \left\{ x = \frac{1}{2} + \frac{\sqrt{3}}{2}, y = \frac{1}{2} - \frac{\sqrt{3}}{2} \right\}$$

Maple can solve recurrence relations which is useful in the analysis of algorithms. The unoptimized version of the Fast Fourier Transform does two recursive calls on an input of half the size + n multiplications.

```
> FFTcost := { T(n) = 2*T(n/2) + n, T(1) = 0 };
```

$$FFTcost := \left\{ T(1) = 0, T(n) = 2 T\left(\frac{n}{2}\right) + n \right\}$$

```
> rsolve( FFTcost, T(n) );
```

$$\frac{n \ln(n)}{\ln(2)}$$

Here is the recurrence relation for Gaussian elimination of an n by n matrix A over a field F to row Echelon form.

Gaussian elimination does  $(n-1)^2$  multiplications to zero out the 1st column then repeats on the n-1 by n-1 matrix.

```
> rec := { T(n) = T(n-1) + (n-1)^2, T(1)=0 };
```

$$rec := \{ T(1) = 0, T(n) = T(n-1) + (n-1)^2 \}$$

```
> sol := rsolve( rec, T(n) );
```

$$sol := 2 (n+1) \left(\frac{n}{2} + 1\right) \left(\frac{n}{3} + 1\right) - 5 (n+1) \left(\frac{n}{2} + 1\right) + 4n + 3$$

```
> sort( simplify(sol), n );
```

$$\frac{1}{3} n^3 - \frac{1}{2} n^2 + \frac{1}{6} n$$

Thus Gaussian elimination does  $\frac{n^3}{3} + O(n^2)$  multiplications.

Here is the classical result for Karatsuba's fast integer multiplication algorithm which does 3 multiplications of half the size plus linear work.

```
> Karatsuba := { T(n) = 3*T(n/2) + c*n, T(1) = d };
```

$$Karatsuba := \left\{ T(1) = d, T(n) = 3 T\left(\frac{n}{2}\right) + cn \right\}$$

```
> rsolve( Karatsuba, T(n) );
```

$$dn^{\frac{\ln(3)}{\ln(2)}} - 2cn + 2n^{\frac{\ln(3)}{\ln(2)}}c$$

```
> evalf(%);
```

$$dn^{1.584962501} - 2.cn + 2.n^{1.584962501}c$$

## Integers

Here are some commands which do integer calculations that we will use in the course. Integers in Maple are not limited to the precision of the hardware on your computer. They are limited by an internal limit of Maple to approximately half a million digits. Any calculations that you do with large integers though will take longer for larger integers.

Here is  $2^{100}$ .

```
> 2^100;
1267650600228229401496703205376
```

The command `irem(a,b)` computes the integer remainder of  $a$  divided by  $b$ . The command `iquo(a,b)` computes the integer quotient of  $a$  divided by  $b$ . For example

```
> a := 20;
   b := 7;
a := 20
b := 7
```

```
> r := irem(a,b);
r := 6
```

```
> q := iquo(a,b);
q := 2
```

It should be the case that  $a = b q + r$ . Let's check

```
> a = b*q + r;
20 = 20
```

The commands `igcd(a,b)` and `ilcm(a,b)` compute the greatest common divisor and least common multiple of integers  $a$  and  $b$ , respectively.

```
> igcd(6,4);
2
```

```
> ilcm(6,4);
12
```

The command `igcdex(a,b,'s','t')` outputs  $g = GCD(a, b)$ . It also assigns  $s, t$  integers satisfying the equation  $s a + t b = g$  and satisfying  $|s| < |b|$  and  $|t| < |a|$ . So this command implements the extended Euclidean algorithm. For example

```
> g := igcdex(3,5,'s','t');
g := 1
```

```
> s;
2
```

```
> t;
-1
```

```
> s*3+t*5;
1
```

The command `isprime(n)` outputs false if  $n$  is composite and true if  $n$  is prime. The

command **ifactor**(n) computes the integer factorization of an integer. For example

```
> isprime(997);
                                     true
> isprime(1001);
                                     false
> ifactor(1001);
                                     (7) (11) (13)
```

Now, if you are not sure what a command does, or how to use it, you can use Maple's on-line help system. You input ?command and then a return. Try the following

```
> ?isprime
> ?ifactor
```

You should get a window with some information about the command and how to use it. Almost all of the on-line help files have examples. If you don't know the name of the command, you can use the help browser to search for the command - see the Help menu at the top right of the window.

For the algorithms based on the Chinese remainder theorem we will need a source of primes. Say 32 bit primes. The command **nextprime**(n) outputs the first prime greater than or equal to n and the command **prevprime**(n) finds the first prime less than or equal to n.

```
> p := prevprime(2^32);
                                     p := 4294967291
> p := prevprime(p);
                                     p := 4294967279
```

Often we will need random numbers, random integers in the range 0..p-1. You can create a random number generator using the rand command as follows. Then call it to create random numbers.

```
> R := rand(10):
> R();
                                     3
> R();
                                     5
> seq( R(), i=1..10 );
                                     4, 0, 7, 4, 9, 1, 1, 3, 7, 2
```

You can also create long random integers.

```
> U := rand(10^100):
> U();
7515910497651723810085474342109878238417236700313088149511552477688536\
216189999714759845649963501017
```

## Lists

The simplest data structure in Maple is a list. The elements of a list may be of any type. To create a list of values enclose them in square brackets [, ]. Lists may be nested of course and the entries may be of any type.

```
> restart;
```

```
> E := []; # the empty list
```

```
E := []
```

```
> L := [1,2,-3,4,1];
```

```
L := [1, 2, -3, 4, 1]
```

```
> M := [[1,2,3],[x,y,z]];
```

```
M := [[1, 2, 3], [x, y, z]]
```

To count the number of entries in a list use nops(L) command.

```
> nops(L);
```

```
5
```

```
> nops(M);
```

```
2
```

To access the i'th element of a list (counting from 1) use a subscript. A negative subscript counts from the end.

```
> L[3];
```

```
-3
```

```
> L[-1];
```

```
1
```

```
> M[2];
```

```
[x, y, z]
```

```
> M[2][2];
```

```
y
```

Use the following to extract a sublist

```
> L[2..3];
```

```
[2, -3]
```

```
> L[2..-1];
```

```
[2, -3, 4, 1]
```

To append (prepend) elements to a list use the following.

```
> op(L);
```

```
1, 2, -3, 4, 1
```

```
> [op(L),5];
```

```
[1, 2, -3, 4, 1, 5]
```

To test if an element is in a list use

```
> member(2,L);
```

```
true
```

Although you can assign to an entry of a list (as if it were an array) if the list has less than 100 elements, do not do this. It creates a copy of the entire list. So it's not efficient. Use Arrays .

```
> L[2] := 10;
```

```
L2 := 10
```

```
> L;
```

```
[1, 10, -3, 4, 1]
```

## Loops and If statements.

```
> restart;
```

To do a sequence of calculations it will be handy to know how to use some of Maple's looping commands and also the if command. To execute a command in Maple conditionally use the if command which has either of the following forms

```
if <condition> then <statements> else <statements> fi
```

or just

```
if <condition> then <statements> fi
```

```
> if 2>1 then print(good) else print(bad) fi;  
good
```

To execute one or more statements zero or more times in a loop use the for command. It has the following form

```
for <variable> from <start> to <end> do <statements> od
```

```
> for i from 1 to 5 do i^2; od;
```

```
1
```

```
4
```

```
9
```

```
16
```

```
25
```

To execute some statements while a condition is true use the while loop. It has the syntax

```
while <condition> do <statements> od
```

In the following example we repeatedly divide an integer  $n$  by 2 until it is odd.

```
> n := 12; while irem(n,2) = 0 do n := iquo(n,2); od;
      n:= 12
      n:= 6
      n:= 3
```

Here is a loop to calculate the GCD of two integers  $a$  and  $b$  using Euclid's algorithm. Notice that this loop has three statements in the body of the loop - between the **do** ... **od**, each of which is terminated by a semicolon. You don't have to put them on the same line as I have done here.

```
> a := 64; b := 20;
      a:= 64
      b:= 20
> while b <> 0 do
>   r := irem(a,b); a := b; b := r;
> od;
      r:= 4
      a:= 20
      b:= 4
      r:= 0
      a:= 4
      b:= 0
```

Thus 4 should be the GCD(64,20). A check with Maple

```
> igcd(64,20);
      4
```

This is a good point to mention what the relational operators are and the logical operators

The relational operators are  $<$ ,  $>$ ,  $=$ ,  $<=$ ,  $>=$ ,  $<>$  (not equals).

The logical operators are **and**, **or** and **not**.

We will use Maple to obtain the first prime bigger than the integer  $128^6$ . Note, the **nextprime** command does this automatically.

```
> p := 128^6+1;
      while not isprime(p) do p := p + 2 od;
      p:= 4398046511105
      p:= 4398046511107
      p:= 4398046511109
      p:= 4398046511111
      p:= 4398046511113
      p:= 4398046511115
```

```
p := 4398046511117
```

```
p := 4398046511119
```

```
> nextprime(128^6);
```

```
4398046511119
```

Two other useful looping constructs are the **map** command and the **seq** command and the **add** command. The examples show what the commands do.

```
> L := [1,2,3,4,5];
```

```
L := [1, 2, 3, 4, 5]
```

```
> map( f, L );
```

```
[f(1), f(2), f(3), f(4), f(5)]
```

```
> map( isprime, L );
```

```
[false, true, true, false, true]
```

```
> seq( i^2, i=1..5 );
```

```
1, 4, 9, 16, 25
```

```
> seq( L[i], i=1..nops(L) );
```

```
1, 2, 3, 4, 5
```

```
> seq( isprime(L[i]), i=1..nops(L) );
```

```
false, true, true, false, true
```

```
> seq( L[i]*x^(i-1), i=1..nops(L) );
```

```
1, 2 x, 3 x2, 4 x3, 5 x4
```

```
> L := [seq( n^2, n=L )];
```

```
L := [1, 4, 9, 16, 25]
```

```
> add( f(i), i=1..5 );
```

```
f(1) + f(2) + f(3) + f(4) + f(5)
```

```
> add( i^2, i=1..5 );
```

```
55
```

```
> add( x[i], i=0..5 );
```

```
x0 + x1 + x2 + x3 + x4 + x5
```

```
> add( x^i, i=0..5 );
```

```
1 + x + x2 + x3 + x4 + x5
```

Read the help files for these commands, they are very handy.

```
> ?map
```

```
> ?seq
```

```
> ?add
```

## ▼ Maple Functions and Procedures

A simple function, like the function  $ek(x) = a*x+b \text{ mod } n$  may be input using the arrow notation in Maple, as follows

```
> ek := x -> 3*x+5 mod 26;
                                     ek := x → 3 x + 5 mod 26
> ek(1);  ek(7);
                                     8
                                     0
```

A procedure in Maple takes the form

```
proc( p1, p2, ... )
local l1, l2, ... ;
global g, g2, ... ;
    statement1;
    statement2;
    ....
    statementn;
end proc
```

There may be zero or more parameters, one or more locals, one or more globals and one or more statements in the procedure body.

The local and global statements are optional. Variables in the procedure body that are not explicitly declared as parameters, locals, or globals are declared to be local automatically if assigned to, otherwise they are global. The value returned by the procedure is the value of *statementn*, the last statement in the body of the procedure or the value of an explicit return statement. Type declarations for parameters and local variables need not be explicitly given. Some examples will help.

```
> f := proc(x) y := x^2; y-1; end proc;
Warning, `y` is implicitly declared local to procedure `f`
                                     f := proc({x}) local y; y := x^2; y - 1 end proc
> f(2);
                                     3
> f(z);
                                     z2 - 1
```

This next example searches a list L for the value x. It outputs the position of the first occurrence of x in L and 0 otherwise. I am also telling Maple that the input should be of type list. Below is an example with inputs of type integer. See **?type** for how to specify types and for what types are available if you need them.

```
> position := proc(x::anything, L::list) local i;
    for i from 1 to nops(L) do if L[i]=x then return i fi; od;
    0; # meaning x is not in the list
end proc;
position := proc({x::anything, L::list})
```

```

local i;
for i to nops(L) do if L[i] = x then return i end if end do; 0
end proc

```

```

> position(x,[u,v,w,x,y,z]);
4
> position(y,[u,v,w]);
0

```

This next example is an implementation of the Euclidean algorithm. It uses the multiple assignment.

```

> a,b := 2,3;
a, b := 2, 3
> EuclideanAlgorithm := proc(a::integer,b::integer) local c,d,r;
(c,d) := (abs(a),abs(b));
while d <> 0 do r := irem(c,d); (c,d) := (d,r); od;
c;
end proc;
> EuclideanAlgorithm(24,210);
6

```

Procedures may be nested, nested lexical scoping is used (a la Pascal).

Procedures may be returned and passed freely as parameters.

The simplest debugging tool is to insert print statements in the procedure. For example

```

> EuclideanAlgorithm := proc(a::integer,b::integer) local c,d,r;
(c,d) := (abs(a),abs(b));
while d <> 0 do r := irem(c,d); print(r); (c,d) := (d,r); od;
c;
end proc;
> EuclideanAlgorithm(24,210);
24
18
6
0
6

```

The next simplest debugging tool is the trace command. All assignment statements are displayed.

```

> trace(EuclideanAlgorithm);
EuclideanAlgorithm
> EuclideanAlgorithm(24,210);
{--> enter EuclideanAlgorithm, args = 24, 210
c, d := 24, 210

```

```

                r := 24
                24
            c, d := 210, 24
                r := 18
                18
            c, d := 24, 18
                r := 6
                6
            c, d := 18, 6
                r := 0
                0
            c, d := 6, 0
                6
<-- exit EuclideanAlgorithm (now at top level) = 6}
                6

```

The printf command can be used to print more detailed information in a controlled format. It works just like the printf command in the C language. The main difference is the %a option for printing algebraic objects like polynomials. E.g.

```

> printf( "A polynomial %a\n", x^2-2*y*x );
A polynomial x^2-2*y*x

```

Here we print the quotients in the Euclidean algorithm. Notice the three argument version of the iquo command. It computes and returns the quotient but assigns the third input (a variable) the value of the remainder. Notice that the quotients, with exception of the first one, are typically small.

```

> q := iquo( 6, 4, 'r' );
                q := 1

```

```

> r;
                2

```

```

> EuclideanAlgorithm := proc(a::integer,b::integer) local c,d,r,q;
    (c,d) := (abs(a),abs(b));
    while d <> 0 do
        r := irem(c,d,'q');
        printf("Quotient = %d\n",q);
        (c,d) := (d,r); od;
    c;
end proc:

```

```

> EuclideanAlgorithm(123456789,54321);
Quotient = 2272
Quotient = 1
Quotient = 2
Quotient = 1

```

```
Quotient = 1
Quotient = 1
Quotient = 14
Quotient = 1
Quotient = 2
Quotient = 1
Quotient = 26
```

3

Here is a recursive implementation of Euclid's algorithm.

```
> EuclideanAlgorithm := proc(a::integer,b::integer)
    if a<0 then EuclideanAlgorithm(-a,b)
    elif b<0 then EuclideanAlgorithm(a,-b)
    elif a<b then EuclideanAlgorithm(b,a)
    elif b=0 then a
    else EuclideanAlgorithm(b,irem(a,b))
    fi;
end:
> EuclideanAlgorithm(-30,16);
```

2

There is more. See **?proc** if you need more information or more tools.

I'd like to give another example of a recursive program. Suppose we have an array A of n elements and we want to sort the array. One way to do this is as follows

Step 1: sort the first n-1 elements recursively

Step 2: insert  $A_n$  the last element in the right place.

That's it!

```
> InsertionSort := proc(A,n) local i, x;
    if n=1 then return; end if; # there's nothing to do!
    InsertionSort(A,n-1); # recursively sort the first n-1 elements
    print(n-1,A); # so we can see what happened
    x := A[n];
    for i from n-1 by -1 to 1 while x < A[i] do A[i+1] := A[i]; od;
    A[i+1] := x;
    return;
end:
> A := Array([5,1,9,2,6]):
InsertionSort(A,5);
A;
```

1, [ 5 1 9 2 6 ]

2, [ 1 5 9 2 6 ]

3, [ 1 5 9 2 6 ]

4, [ 1 2 5 9 6 ]

[ 1 2 5 6 9 ]

```

> A := Array([5,9,2,6,1]):
InsertionSort(A,5);
A;
1, [ 5 9 2 6 1 ]
2, [ 5 9 2 6 1 ]
3, [ 2 5 9 6 1 ]
4, [ 2 5 6 9 1 ]
   [ 1 2 5 6 9 ]

```

## Modular Arithmetic

Modular arithmetic is done using the `mod` operator in Maple. By default, Maple uses the positive range for the integers modulo  $m$ , that is, the result is calculated in the range  $0..m-1$ .

```

> restart;
> 12 mod 7;
5
> 2+3*3 mod 7;
4

```

To compute  $a^{-1} \bmod m$ , you can do either of the following

```

> 2^(-1) mod 7;
4
> 1/2 mod 7;
4

```

To compute  $a^n \bmod m$  you can do either

```

> 2 ^ 200 mod 7;
4
> 2 &^ 200 mod 7;
4

```

Use the latter. The difference is that in the first case, the integer  $2^{200}$  was computed then reduced modulo  $m$ . In the second case, all products were reduced modulo  $m$  so no large integers occurred.

We will use a loop to verify that Fermat's (little) theorem holds for  $p = 13$  but not for  $n = 14$ .

```

> p := 13;
for i from 0 to p-1 do (i^p mod p) = i od;

```

```
p := 13
0 = 0
1 = 1
2 = 2
3 = 3
4 = 4
5 = 5
6 = 6
7 = 7
8 = 8
9 = 9
10 = 10
11 = 11
12 = 12
```

```
> n := 14;
  for i from 0 to n-1 do (i^n mod p) = i od;
```

```
n := 14
0 = 0
1 = 1
4 = 2
9 = 3
3 = 4
12 = 5
10 = 6
10 = 7
12 = 8
3 = 9
9 = 10
4 = 11
1 = 12
0 = 13
```

You can use the `ichrem` command to solve the Chinese remainder problem.  
Suppose we want to solve  $u \equiv 3 \pmod{5}$  and  $u \equiv 4 \pmod{7}$  and  $u \equiv 1 \pmod{3}$ .



To compute the degree of a polynomial and read off a coefficient in  $x^i$  do

```
> degree(f,x); coeff(f,x,2);
```

$$4$$
$$-3$$

```
> degree(a,x); degree(a,y);
```

$$6$$
$$6$$

```
> coeff(a,x,2); coeff(a,y,2);
```

$$y^4$$
$$-x^4$$

We will only need polynomials in one variable and mostly work in the rings  $\mathbf{Z}[x]$  and  $\mathbf{Z}_p[x]$  where  $p$  will be a prime integer. In what follows we show operations for  $\mathbf{Z}_p[x]$  and also  $\mathbf{Q}[x]$ . For help for operations for polynomials see ?polynomial. For help for operations in  $\mathbf{Z}_p[x]$  see ?mod.

Here are two polynomials

```
> a := 2*x^6-3*x^5+3*x+3;
```

$$a := 2x^6 - 3x^5 + 3x + 3$$

```
> b := 3*x^4-4*x^3+1;
```

$$b := 3x^4 - 4x^3 + 1$$

The command **eval**( a(x), x=k ) evaluates the polynomial a(x) at x = k. The command **Eval**( a, x=k ) **mod** p does this modulo p. For example

```
> eval( a, x=2 );
```

$$41$$

```
> Eval( a, x=2 ) mod 7;
```

$$6$$

Here is how we can tabulate the values of this polynomial for all values in  $Z_7$ . We conclude that a(x) has no roots.

```
> seq( Eval(a,x=i) mod 7, i=0..6 );
```

$$3, 5, 6, 6, 4, 4, 5$$

We can interpolate a polynomial from it's values as follows

```
> a;
```

$$2x^6 - 3x^5 + 3x + 3$$

```
> X := [seq(i,i=0..8)];
```

$$X := [0, 1, 2, 3, 4, 5, 6, 7, 8]$$

```
> Y := [seq( eval(a,x=i), i=0..8 )];
```

$$Y := [3, 5, 41, 741, 5135, 21893, 70005, 184901, 426011]$$

```
> interp(X,Y,p);
```

$$2p^6 - 3p^5 + 3p + 3$$

The command **expand(a\*b)** multiplies out the product  $a b$ . The command **Expand(a\*b) mod p** does the product modulo  $p$ , that is, all coefficients in the resulting polynomial are reduced modulo  $p$ . For example

```
> p := 5;
```

$$p := 5$$

```
> expand(a*b);
```

$$6x^{10} - 17x^9 + 2x^6 + 12x^8 + 6x^5 - 3x^4 + 3x - 12x^3 + 3$$

```
> Expand(a*b) mod p;
```

$$3 + 3x + 3x^3 + 2x^4 + x^5 + 2x^6 + 2x^8 + 3x^9 + x^{10}$$

The operations **rem(a,b,x)** and **quo(a,b,x)** compute, respectively, the remainder  $r$  and quotient  $q$  of  $a$  divided by  $b$  satisfying  $a = bq + r$  with  $r = 0$  or  $\deg(r) < \deg(b)$ . The corresponding operations for  $Z_p$  are **Rem(a,b,x) mod p** and **Quo(a,b,x) mod p**. For example

```
> r := rem(a,b,x);
```

$$r := \frac{85}{27} + \frac{28}{9}x - \frac{2}{3}x^2 - \frac{16}{27}x^3$$

```
> q := quo(a,b,x);
```

$$q := \frac{2}{3}x^2 - \frac{1}{9}x - \frac{4}{27}$$

```
> expand( a = b*q+r );
```

$$2x^6 - 3x^5 + 3x + 3 = 2x^6 - 3x^5 + 3x + 3$$

```
> r := Rem(a,b,x) mod p;
```

$$r := 2x^3 + x^2 + 2x$$

```
> q := Quo(a,b,x) mod p;
```

$$q := 4x^2 + x + 3$$

```
> Expand( a = b*q+r ) mod p;
```

$$2x^6 + 2x^5 + 3x + 3 = 2x^6 + 2x^5 + 3x + 3$$

The commands **gcd(a,b)** and **lcm(a,b)** compute, respectively the greatest common divisor and least common multiple of two polynomials. The corresponding operations for  $Z_p$  are **Gcd(a,b) mod p** and **Lcm(a,b) mod p**. For example

```
> gcd(x^4-2*x^2+2,x^4+1);
```

$$1$$

```
> Gcd(x^4-2*x^2+2,x^4+1) mod p;
```

$$x^2 + 2$$

The command **gcdex(a,b,x,'s','t')** outputs  $g = GCD(a, b)$ . It also outputs through the input parameters  $s, t$  integers satisfying the equation  $sa + tb = g$  and satisfying  $\deg(s) < \deg(b)$  and  $\deg(t) < \deg(a)$ . The corresponding command for  $Z_p$  is **Gcdex(a, b, x, 's', 't') mod p**. For example

```
> gcdex(a,b,x,'s','t');
```

```

1
> s;

$$\frac{6847}{25565} - \frac{1356}{5113}x + \frac{8358}{25565}x^2 - \frac{3312}{25565}x^3$$


$$\frac{5024}{25565} - \frac{201}{25565}x + \frac{4958}{25565}x^3 - \frac{4734}{25565}x^2 + \frac{2208}{25565}x^5 - \frac{1188}{5113}x^4$$


```

```
> expand( s*a+t*b );
```

```

1
> Gcdex(a,b,x,'s','t') mod p;

$$x^2 + 3x + 1$$


```

```
> s;
```

$x + 4$

```
> t;
```

$x^3 + 3x^2 + 3x + 4$

```
> Expand(a*s+t*b) mod p;
```

$x^2 + 3x + 1$

The command **irreduc**(a) outputs true if the polynomial  $a(x)$  is irreducible and the command **factor**(a) outputs the factorization of  $a(x)$  into irreducible factors over the integers. The corresponding commands for  $Z_p$  are **Irreduc(a) mod p** and **Factor(a) mod p**. For example

```
> factor(a);
```

$2x^6 - 3x^5 + 3x + 3$

```
> factor(b);
```

$(3x^2 + 2x + 1)(x - 1)^2$

```
> Factor(a) mod 5;
```

$2(x + 1)(x + 4)^5$

```
> Factor(b) mod 5;
```

$3(x^2 + 4x + 2)(x + 4)^2$

The polynomial  $x^2 + x + 1$  is irreducible modulo 2

```
> Factor(x^2+x+1) mod 2;
```

$x^2 + x + 1$

and hence the finite field of 4 elements can be represented by polynomials of degree  $< 2$  over the integers modulo 2, i.e. the polynomials  $R = \{0, 1, x + 1, x\}$ . We construct the multiplication table M for this finite field as follows.

```
> R := [0,1,x,x+1];
```

```
M := matrix(4,4);
```

$R := [0, 1, x, x + 1]$

```
M := array(1..4, 1..4, [ ])
```

```
> for i to 4 do  
  for j to 4 do M[i,j] := Rem(R[i]*R[j],x^2+x+1,x) mod 2 od;  
od;  
> print(M);
```

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & x & x+1 \\ 0 & x & x+1 & 1 \\ 0 & x+1 & 1 & x \end{bmatrix}$$

See `?mod` for other operations on polynomials over the integers modulo  $p$ .

## Subscripted Names and Arrays

Variables may be subscripted. For example, here is a polynomial in  $x_1, x_2, x_3$ . You can assign to the subscripts.

```
> restart;
```

```
> f := 1-x[1]*x[2]*x[3];
```

```
> x[1] := 3;
```

$$f := 1 - x_1 x_2 x_3$$
$$x_1 := 3$$

```
> f;
```

$$1 - 3 x_2 x_3$$

There may be more than one subscript and the subscripts may be any value.

Arrays are like arrays from computing science. Here is how to create a one-dimensional array  $A$  with values indexed from 1 to 5.

```
> A := Array(1..5);
```

$$A := [ 0 \ 0 \ 0 \ 0 \ 0 ]$$

By default, the entries in the array  $A$  are initialized to 0.

```
> A[1] := 3;
```

$$A_1 := 3$$

```
> A[1];
```

$$3$$

```
> for i from 2 to 5 do A[i] := 3*A[i-1] od;
```

$$A_2 := 9$$
$$A_3 := 27$$

$$A_4 := 81$$

$$A_5 := 243$$

Here is a Maple procedure for multiplying two positive integers of length  $m$  and  $n$  stored in the arrays  $A$  and  $B$  base 10 where the arrays are indexed from 0, so  $A$  is indexed from 0 to  $m-1$ .  $N$

```
> IntMul := proc(m::posint,n::posint,A::Array,B::Array)
  local C, i,j,carry,t;
  C := Array(0..m+n-1);
  for i from 0 to m-1 do
    carry := 0;
    for j from 0 to n-1 do
      t := A[i]*B[j]+carry+C[i+j];
      C[i+j] := irem(t,10,'carry');
    od;
    C[i+j] := carry;
  od;
  C;
end;
```

```
> a,b := 9876,1234;
```

$a, b := 9876, 1234$

```
> A := convert(a,base,10);
```

$A := [6, 7, 8, 9]$

The above is a list. Here is a short way to make it into an Array.

```
> A := Array(0..3,A);
```

$A := \text{Array}(0..3, \{(0) = 6, (1) = 7, (2) = 8, (3) = 9\}, \text{datatype} = \text{anything}, \text{storage} = \text{rectangular}, \text{order} = \text{Fortran\_order})$

```
> B := Array(0..3,convert(b,base,10));
```

$B := \text{Array}(0..3, \{(0) = 4, (1) = 3, (2) = 2, (3) = 1\}, \text{datatype} = \text{anything}, \text{storage} = \text{rectangular}, \text{order} = \text{Fortran\_order})$

```
> C := IntMul(4,4,A,B);
```

$C := \text{Array}(0..7, \{(0) = 4, (1) = 8, (2) = 9, (3) = 6, (4) = 8, (5) = 1, (6) = 2, (7) = 1\}, \text{datatype} = \text{anything}, \text{storage} = \text{rectangular}, \text{order} = \text{Fortran\_order})$

```
> c := add( C[i]*10^i, i=0..7 );
```

$c := 12186984$

```
> a*b;
```

$12186984$

The same basic procedure can be used to multiply polynomials. Suppose we represent a polynomial as a list of coefficients. To allow us to convert from Maple's polynomial representation we'll write a Maple procedure `MapleToList` and `ListToMaple` to convert from and to Maple's polynomial representation.

```

> MapleToList := proc(f::polynom(rational),x) local i;
    if f=0 then return [] fi;
    [seq( coeff(f,x,i), i=0..degree(f,x) )];
end:
ListToMaple := proc(L,x) local i;
    add( L[i]*x^(i-1), i=1..nops(L) );
end:
> f := 3*x^3-12*x+5;
                                f:= 3 x3 - 12 x + 5
> F := MapleToList(f,x);
                                F:= [5, -12, 0, 3]
> ListToMaple(F,x);
                                3 x3 - 12 x + 5

```

Now we can write the polynomial multiplication procedure - again, we'll use Arrays of coefficients. We'll show how to construct the Array in two ways, the second being a shortcut for the first.

```

> PolMul := proc(f::list(rational),g::list(rational))
    local C,m,n,i,j,A,B;
    if f=[] or g=[] then return [] fi;
    m := nops(f)-1; A := Array(0..m);
    for i from 0 to m do A[i] := f[i+1] od;
    n := nops(g)-1; B := Array(0..n,g);
    C := Array(0..m+n+1);
    for i from 0 to m do
        for j from 0 to n do
            C[i+j] := A[i]*B[j]+C[i+j];
        od;
    od;
    [ seq( C[i], i=0..m+n ) ];
end:
> a := -62-50*x^5-12*x^4-18*x^3+31*x^2-26*x;
                                a:= -62 - 50 x5 - 12 x4 - 18 x3 + 31 x2 - 26 x
> b := x^4+5*x^2+6*x+7;
                                b:= x4 + 5 x2 + 6 x + 7
> ListToMaple( PolMul( MapleToList(a,x), MapleToList(b,x) ), x );
                                -434 - 554 x - 249 x2 - 70 x3 - 99 x4 - 538 x5 - 329 x6 - 268 x7 - 12 x8 - 50 x9
> expand(a*b);
                                -434 - 554 x - 249 x2 - 70 x3 - 99 x4 - 538 x5 - 329 x6 - 268 x7 - 12 x8 - 50 x9

```

## ▼ Lists or Arrays and Benchmarking

PLEASE note that appending to a list of length  $n$  items makes a copy so costs  $O(n)$ . So

don't do the following for example.

```
> n := 5;
L := [];
p := 2;
for i to n do L := [op(L), p]; p := nextprime(p); od;
L := [ ]
n := 5
p := 2
L := [2]
p := 3
L := [2, 3]
p := 5
L := [2, 3, 5]
p := 7
L := [2, 3, 5, 7]
p := 11
L := [2, 3, 5, 7, 11]
p := 13
[2, 3, 5, 7, 11]
```

Why not? Because this uses  $O(n^2)$  space and time. It will be very slow for large  $n$ . You are much better off using an Array or table. For example use

```
> n := 5;
A := Array(1..n);
p := 2;
for i to n do A[i] := p; p := nextprime(p); od;
A;
n := 5
A := [ 0 0 0 0 0 ]
p := 2
A1 := 2
p := 3
A2 := 3
p := 5
A3 := 5
p := 7
```

$A_4 := 7$

$p := 11$

$A_5 := 11$

$p := 13$

[ 2 3 5 7 11 ]

This is so important that I'm going to time it so you appreciate the difference.

```
> n := 5000:
  A := Array(1..n):
  p := 2:
  st := time():
  for i to n do A[i] := p: p := nextprime(p): od:
  time()-st;
                                0.097
```

```
> n := 5000:
  L := []:
  st := time():
  for i to n do L := [op(L),p]: p := nextprime(p): od:
  time()-st;
                                0.291
```

The same quadratic behaviour occurs with sets and polynomials. So these are quadratic too.

```
> n := 5000:
  S := {}:
  p := 2:
  st := time():
  for i to n do S := S union {p}: p := nextprime(p): od:
  time()-st;
                                0.359
```

```
> n := 5000:
  f := 0:
  st := time():
  for i to n do f := f + x^p: p := nextprime(p): od:
  time()-st;
                                1.521
```

If you need to do something like this and n is large then first create the values in an Array or table and then construct the set, list or polynomial like this.

```
> st := time(): S := {seq( u, u=S )}: time()-st;
                                0.001
```

```
> st := time(): f := add( x^p, p=L ): time()-st;
                                0.025
```

