Semester 1 **Introduction to Magma for Cryptography** 2004

**Magma** is a powerful computer algebra system, developed in Sydney, which combines a library of high-performance algorithms with its own object-oriented computer language. The library of functions and functionality are accessed from an interpretted user shell. This makes it easy to develop sophisticated computer projects quickly, but also to use `Magma` interactively as an interface to the large body of built-in mathematical type: integers and rational numbers (of arbitrary size), finite fields, residue class rings ($\mathbb{Z}/m\mathbb{Z}$), polynomials, vector spaces and matrices, and much more.

## The Magma Shell.

To start the `Magma` shell, just type:

```
> magma
```

(where the symbol `>` is the prompt and not typed). You should see something like:

```
Magma V2.10-14    Wed Mar 17 2004 12:18:31 on milan
Type ? for help.  Type <Ctrl>-D to quit.
>
```

**Entering commands.**
Basic types like integers and rational numbers are builtin and can be typed directly at the command line without declaring their type. Every command in `Magma` is terminated by a semicolon.

```
> 1 + 1;
2
> Factorization(2^64+1);
[ <274177, 1>, <67280421310721, 1> ]
> 231/23 * 2/55;
42/115
```

Basic operations such as `+, -, *, /` are part of the `Magma` library of functions (with automatically recognition of the types of the arguments and application the correct algorithm which applies). More advanced algorithms like `Factorization` are also part of the `Magma` library of functions.

## Types and Parents

Every object in `Magma` has a *type* and a *parent*. The type is used for function *overloading* – there can exist multiple functions in `Magma` with the same name, each operating on objects of different types.

```
> Type(2);
RngIntElt
> Type(2/3);
FldRatElt
> Type(Type(2));
Cat
```

The parent of an object is itself a object in `Magma`.

```
> Parent(2);
Integer Ring
> Type(Parent(2/3);
FldRat
```

## Assignment and Output

The assignment operator is `:=`. Unlike other strongly typed languages, the type of the object on the left (here `x`) does not have to be declared in advance.

```
> x := 3;
> x := x^6;
```

To print an object in `Magma`, just type it at the command line (followed by a semicolon):

```
> x;
729
```

The output of the previous three commands are stored in a buffer and accessed by `$1`, `$2`, and `$3`.

## Booleans and Boolean Operators

The boolean truth values `true` and `false` have their own types in `Magma`. The comparison operators `eq, lt, le, gt` and `ge` return boolean values, and the boolean operators `and` and `or` take them as arguments.

```
> 2 lt 1 or 3 ge 4;
false
```

## Sequences and Sets

A sequence of elements, indexed over the positive integers $1, 2, \ldots$, is created by a comma-separated list inside of square brackets [ ], and a set is defined by a similar list inside of curly brackets { }.

```
> [ 1, 2, 2, 1 ];
[ 1, 2, 2, 1 ]
> { 1, 2, 2, 1 };
{ 1, 2 };
```

There exists a special constructor for intervals $[n \ldots m]$. A useful construction for forming sequences and sets is the following, which we describe first by example.

```
> [ n^2 : n in [-5..5] ];
[ 25, 16, 9, 4, 1, 0, 1, 4, 9, 16, 25 ]
> { n^2 : n in [-5..5] };
{ 0, 1, 4, 9, 16, 25 }
```

The left-hand side of the colon `:` can be a function or expression in a variable whose values are given by the right-hand side. Note that in this context the `in` operator enumerates all elements of the sequence or set it operates on. A powerful extension to this syntax is the use of a boolean test in conjunction with iteration over a sequence of set.

```
> { n : n in [-5..5] | (n^2 - n + 1) lt 20 };
{ -3, -2, -1, 0, 1, 2, 3, 4 }
```

## Iteration Operators

For most of the standard operators `op` (e.g. `+, *, and, or`), which are commutative (`x op y` is the same as `y op x`) and return values in the same parent, has an associated iteration operator `&op` (e.g. `&+, &*, &and, &or`) which applies to sequences. If the operation is also associative (i.e. `(x op y) op z` is the same as `x op (y op z)`), then it also applies to sets.

```
> &+[ 1, 2, 2, 1 ];
6
> &+{ 1, 2, 2, 1 };
3
> &or[ (x^2 + x + 1) eq 19 : x in [-10..10] ];
false
> { {x,y} : x, y in [-10..10] | (x^2 + x*y + y^2) eq 19 };
{
{ -5, 3 },
{ -3, 5 },
{ -2, 5 },
{ -3, -2 },
{ 2, 3 },
{ -5, 2 }
}
```

## Strings

The most common built-in `Magma` types we will use in this course are strings. A string is created by enclosing input in double quotes.

```
> S := "This
>   is
>      a
```

```
>         string.";
> S;
This
    is
       a
          string.
```

As above, a string can contain newline characters and spaces. The two characters \ and " have special functionality, and must be typed as \\ and \", respectively. The newline string can be created directly by "\n" or "\r". Since the structure of strings is analogous to a sequence of characters, they support a variety of sequence operators, such as cat for concatenation and <string>[i] for access to the i-th character of <string>.

```
> s := "But Angie, Angie, ain't it time we said good-bye?\n";
> t := "With no loving in our souls ";
> u := "and no money in our coats\n";
> v := "You can't say we're satisfied\n\n";
> w := "...they can't say we never tried";
> angie := &cat[ s,t,u,v,w ];
> angie;
But Angie, Angie, ain't it time we said good-bye?
With no loving in our souls and no money in our coats
You can't say we're satisfied

...they can't say we never tried
> I := [56..58] cat [125..130] cat [5,144,139,165,57,56];
> &cat[ angie[i] : i in I ];
no satisfAction
```