Mathematics 676-4 **Homework (due Jan. 29)** A. Hulpke

1) Let $F=\mathbb{F}_3$ be the field with 3 elements and let $f=x^2+x-1\in F[x]$. Let $K=F(\alpha)$ where α is a root of f. (As f is irreducible, K is a field with 9 elements. You may assume this.)
a) Show that α is a generator of $K^\times$.
b) Construct a table of Zech-Logarithms for K, based on α.
c) Using this table, calculate $(\alpha^5+\alpha^3)/(\alpha^7+\alpha^2)$.

2) Show that $\sqrt{5+2\sqrt{6}}+\sqrt{5-2\sqrt{6}}=2\sqrt{3}$. (Hint: Consider the square of the expression.)

3) Compute the determinant of

$$\begin{pmatrix} -2 & 1 & -3 \\ -23 & 9 & -24 \\ 16 & 7 & -20 \end{pmatrix}$$

using a modular method with primes < 1000.

4) Using the familiar formula $\det A=\sum_{\sigma\in S_n}\operatorname{sgn}(\sigma)a_{1\sigma(1)}a_{2\sigma(2)}\cdots a_{n\sigma(n)}$ for the determinant of a square matrix $A\in\mathbb{Z}^{n\times n}$, to derive an upper bound on $\det(A)$ in terms of n and $B=\max_{i,j}\left|a_{ij}\right|$. Compare this to the Hadamard bound and tabulate both bounds and their ratio for $1\le n\le 10$.

5) Get accustomed with the system GAP (or another computer algebra system you prefer to use).

An introduction to GAP

You start GAP by clicking the GAP 4.3 icon (on windows) or typing `gap` under Unix. The program will start up and you will get a text prompt, looking like this:

```
gap>
```

You now can type in commands (followed by a semicolon) and GAP will return the result.
To leave GAP you can either call `quit;` or type `<ctl>-d`
The editing commands are similar as in the EMACS editor (depending on the setup, also cursor keys might work):

`<ctl>-p` gets the previous line. (If the cursor is not at the start of the line, it gets the last line that starts like the current one.)

`<ctl>-n` gets the next line.

`<ctl>-f` goes one character to the right.

`<ctl>-b` goes one character to the left.

`<ctl>-a` goes to the start of the line.

`<ctl>-e` goes to the end of the line.

`<ctl>-d` deletes the character under the cursor (insert is automatic).

`<ctl>-k` deletes the rest of the line.

If you want to obtain a copy of your work, you can use `LogTo("filename");` which will create a file with the input and output you got. (Note: The file will appear in the directory in which GAP was started. Under Windows, you might have to give a full path such as `LogTo("c:/mydisk/home");` – however note that paths in GAP are **always** given with forward slashes in Unix style, even under Windows.) `LogTo();` will switch logging off again.
Hints:

- GAP is picky about upper case/lower case. `LogTo` is not the same as `logto`.
- All commands end in a semicolon!
- Type `?` followed by a subject name (and no semicolon...) to get the online help.
- If you get syntax errors, use `<ctl>-p` to get the line back, and correct the errors.
- Incorrect input, interruption of a calculation (by pressing `<ctl>-c` – this can be useful if you tried out a command that takes too long) or (heaven beware – if you encounter any please let me know) bugs can cause GAP to enter a so-called break-loop, indicated by the prompt `brk>`. You can leave this break loop with `quit;` or `<ctl>-d`.
- If you create larger input, it can be helpful to put it in a text file and to read it from GAP. This can be done with the command `Read("filename");`.
- By terminating a command with a double semicolon `;;` you can avoid GAP displaying the result. (Obviously, this is only useful if assigning it to a variable.)
- everything after a hash mark (#) is a comment.

We now do a few easy calculations. If you have not used GAP before, it might be useful to do these on the computer in parallel to reading.
GAP knows integers of arbitrary length and rational numbers:

```
gap> -3; 17 - 23;
-3
-6
gap> 2^200-1;
1606938044258990275541962092341162602522202993782792835301375
gap> 123456/7891011+1;
2671489/2630337
```

The 'mod' operator allows you to compute one value modulo another. Note the blanks:

```
gap> 17 mod 3;
2
```

GAP knows a precedence between operators that may be overridden by parentheses and can compare objects:

```
gap> (9 - 7) * 5 = 9 - 7  * 5;
false
gap> 5/3<2;
true
```

You can assign numbers (or more general: every GAP object) to variables, by using the assignment operator `:=`. Once a variable is assigned to, you can refer to it as if it was a number. The special variables `last`, `last2`, and `last3` contain the results of the last three commands.

```
gap> a:=2^16-1;
65535
gap> b:=a/(2^4+1);
3855
gap> 5*b-3*a;
-177330
gap> last+5;
-177325
gap> last+2;
-177323
```

The following commands show some useful integer calculations:

```
gap> Int(8/3); # round down
2
gap> QuoInt(76,23); # integral part of quotient
3
gap> 76 mod 23; # remainder (note the blanks)
7
gap> IsPrime(6);   #primality test
false
gap> Gcd(64,30);
2
gap> rep:=GcdRepresentation(64,30);
[ -7, 15 ]
gap> rep[1]*64+rep[2]*30;
2
```

Objects separated by commas and enclosed in square brackets form a list.
Collections of objects are represented by such lists. Lists are also used to represent sets.

```
gap> l:=[5,3,99,17,2]; # create a list
[ 5, 3, 99, 17, 2 ]
gap> l[4]; # access to list entry
17
gap> l[3]:=22; # assignment to list entry
22
gap> l;
[ 5, 3, 22, 17, 2 ]
gap> Length(l);
5
gap> 3 in l; # element test
true
gap> 4 in l;
false
gap> Position(l,2);
5
gap> Add(l,17); # extension of list at end
gap> l;
[ 5, 3, 22, 17, 2, 17 ]
gap> s:=Set(l); # new list, sorted, duplicate free
[ 2, 3, 5, 17, 22 ]
gap> l;
[ 5, 3, 22, 17, 2, 17 ]
gap> AddSet(s,4); # insert in sorted position
gap> AddSet(s,5); # and avoid duplicates
gap> s;
[ 2, 3, 4, 5, 17, 22 ]
```

Results that consist of several numbers are represented as list.

```
gap> DivisorsInt(96);
[ 1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 96 ]
gap> Factors(2^126-1);
[ 3, 3, 3, 7, 7, 19, 43, 73, 127, 337, 5419, 92737, 649657, 77158673929 ]
```

There are powerful list functions that often can save programming loops: `List`, `Filtered`, `ForAll`, `ForAny`, `First`. The notation `i -> xyz` is a shorthand for a one parameter function.

```
gap> l:=[5,3,99,17,2];
[ 5, 3, 99, 17, 2 ]
gap> List(l,IsPrime);
[ true, true, false, true, true ]
gap> List(l,i -> i^2);
[ 25, 9, 9801, 289, 4 ]
gap> Filtered(l,IsPrime);
[ 5, 3, 17, 2 ]
gap> ForAll(l,i -> i>10);
false
gap> ForAny(l,i -> i>10);
true
gap> First(l,i -> i>10);
99
```

A special case of lists are ranges, indicated by double dots. They can also be used to create arithmetic progressions:

```
gap> l:=[10..100];
[ 10 .. 100 ]
gap> Length(l);
91
gap> First(l,IsPrime);
11
gap> l2:=[3,7..99];
[ 3, 7 .. 99 ]
gap> Length(l2);
25
gap> Filtered(l2,IsPrime);
[ 3, 7, 11, 19, 23, 31, 43, 47, 59, 67, 71, 79, 83 ]
gap> Filtered(l2,i-> not IsPrime(i));
[ 15, 27, 35, 39, 51, 55, 63, 75, 87, 91, 95, 99 ]
```

Lists are also used to form vectors and matrices:
A vector is simply a list of numbers. A list of (row) vectors is a matrix. GAP knows matrix arithmetic.

```
gap> vec:=[1,2,3,4];
[ 1, 2, 3, 4 ]
gap> vec[3]+2;
5
gap> 3*vec+1;
[ 4, 7, 10, 13 ]
gap> mat:=[[1,2,3,4],[5,6,7,8],[9,10,11,12]];
[ [ 1, 2, 3, 4 ], [ 5, 6, 7, 8 ], [ 9, 10, 11, 12 ] ]
gap> mat*vec;
[ 30, 70, 110 ]
gap> mat:=[[1,2,3],[5,6,7],[9,10,12]];;
gap> mat^5;
[ [ 289876, 342744, 416603 ], [ 766848, 906704, 1102091 ],
  [ 1309817, 1548698, 1882429 ] ]
gap> DeterminantMat(mat);
-4
```

The command `Display` can be used to get a nicer output:

```
gap> 3*mat^2-mat;
[ [ 113, 130, 156 ], [ 289, 342, 416 ], [ 492, 584, 711 ] ]
gap> Display(last);
[ [  113,  130,  156 ],
  [  289,  342,  416 ],
  [  492,  584,  711 ] ]
```

To compute in the integers modulo a number, we have to create special objects to represent the residue classes.

```
gap> im:=Integers mod 6; # represent numbers for ‘‘modulo 6’’ calculations
(Integers mod 6)
```

To convert “ordinary” integers to residue classes, we have to multiply them with the“One” of these residue classes, the command `Int` converts back to ordinary integers:

```
gap> a:=5*One(im);
ZmodnZObj( 5, 6 )
gap> b:=3*One(im);
ZmodnZObj( 3, 6 )
gap> a+b;
ZmodnZObj( 2, 6 )
gap> Int(last);
2
```

(If one wants one can get all residue classes or – for example test which are invertible).

```
gap> Elements(im);
[ ZmodnZObj(0,6),ZmodnZObj(1,6),ZmodnZObj(2,6),ZmodnZObj(3,6),
  ZmodnZObj(4,6),ZmodnZObj(5,6) ]
gap> Filtered(last,x->IsUnit(x));
[ ZmodnZObj( 1, 6 ), ZmodnZObj( 5, 6 ) ]
gap> Length(last);
2
```

If we calculate modulo a prime the output looks a bit different, since GAP uses Zech Logarithms.

```
gap> im:=Integers mod 7;
GF(7)
gap> One(im);
Z(7)^0
gap> a:=6*One(im);
Z(7)^3
gap> b:=3*One(im);
Z(7)
gap> a+b;
Z(7)^2
gap> Int(a+b);
2
```

We can also calculate in matrices modulo a number:

```
gap> mat:=[[1,2,3],[5,6,7],[9,10,12]]*One(im);
[ [ Z(7)^0, Z(7)^2, Z(7) ], [ Z(7)^5, Z(7)^3, 0*Z(7) ],
  [ Z(7)^2, Z(7), Z(7)^5 ] ]
gap> Display(mat);
 1 2 3
```

```
 5 6 .
 2 3 5
gap> mat^-1+mat;
[ [ Z(7)^4, Z(7)^4, Z(7)^4 ], [ Z(7)^3, Z(7)^0, Z(7)^5 ],
  [ Z(7), Z(7)^0, Z(7)^3 ] ]
```

As a general rule of thumb, if you want to create objects modulo a prime, the numbers have to be transformed in the appropriate finite field.
To create polynomials, one has to define indeterminates first. Indeterminates are displayed either by their internal numbers or you can prescribe names.

```
gap> x:=X(Rationals,1); # number
x_1
gap> x:=X(Rationals,"x");
x
```

One can now generate a polynomial by arithmetic operations. It is possible to get the coefficients list of a polynomial back.

```
gap> x^5+3*x-15/2;
x^5+3*x-15/2
gap> CoefficientsOfUnivariatePolynomial(x^5+3*x^3-15/2*x);
[ 0, -15/2, 0, 3, 0, 1 ]
```

It is also possible to generate polynomials from coefficient lists. However the syntax might initially be confusing. The first argument indicates the family of objects, i.e. the domain over which the polynomial is to be considered:

```
gap> UnivariatePolynomialByCoefficients(FamilyObj(1),[0,1,2,3],1);
3*x^3+2*x^2+x
gap> UnivariatePolynomialByCoefficients(FamilyObj(Z(5)),Z(5)^0*[0,1,2,3],1);
Z(5)^3*x_1^3+Z(5)*x_1^2+x_1
```

Polynomials support the arithmetic operations as well as `mod`, `QuotientRemainder` and `Degree`.