Mathematics 566 **Homework (due Aug. 29th)** A. Hulpke

D&F;1.1: 1,6,18,27

D&F;1.2: 1a),12

1) Read the "Preliminaries" chapter. (If any of this material is completelty new to you and you have problems with it, come and see me.)

2) Work though the introduction to GAP at the end of these problems.

3) (GAP[1]) Find how many elements in $\mathbb{Z}/n\mathbb{Z}$ are invertible for $n = 4, 8, 16, 128, 7, 35, 49$.

4) (GAP) Let $a = \begin{pmatrix} \bar{1} & \bar{1} \\ \bar{1} & \bar{2} \end{pmatrix}$ and $b = \begin{pmatrix} \bar{2} & \bar{1} \\ \bar{1} & \bar{1} \end{pmatrix}$, where the bar denotes residue classes in $\mathbb{Z}/3\mathbb{Z}$. Let $G = \langle a, b \rangle$ be the group generated by a and b. (This group is usually called Q_8.)
Find the elements of G and their orders (either by checking whether `IsOne(mat^n);` or by computing `Order(mat)` for a matrix `mat`). Is G abelian?

An introduction to GAP

To start GAP, either type `gap` at the system prompt or click the GAP icon[2]. 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.

[1] This indicates that you can use GAP to solve the problem. If you insist doing it by hand, I won't stop you, but some of the calculations might become extremely tedious otherwise.

[2] The version on stokes is a bit old. I will see to get this updated soon.

`<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 want to give a full path such as `LogTo("c:/mydisk/home");`.) `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, I'd recommend you 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> 3^132;
955004950796825236893190701774414011919935138974343129836853841
gap> 27/18;
3/2
```

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
gap> 5/3>=2;
false
```

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:=6*(3+17);
120
gap> a+a^2-a^3;
-1713480
gap> last+2;
-1713478
gap> last+2;
```

By enclosing objects with square brackets, and separating them by commas, you can create a list. (A special case are *ranges*, indicated by double dots.) There are convenient functions `List`, `Filtered`, `ForAll`, `ForAny`,... for working on lists.

```
gap> l:=[1,2,3,4,5,6,7,10];
[ 1, 2, 3, 4, 5, 6, 7, 10 ]
gap> l2:=[3..12];
[ 3 .. 12 ]
gap> Difference(l,l2);
[ 1, 2 ]
gap> List(l,IsPrimeInt);
[ false, true, true, false, true, false, true, false ]
gap> Filtered(l,IsPrimeInt);
[ 2, 3, 5, 7 ]
gap> ForAll(l,IsPrimeInt);
false
gap> List(l,i->i^3);
[ 1, 8, 27, 64, 125, 216, 343, 1000 ]
```

A *vector* is simply a list of numbers. One can access elements of a list using square brackets as well. 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]];
[ [ 1, 2, 3 ], [ 5, 6, 7 ], [ 9, 10, 12 ] ]
gap> matˆ5;
[ [ 289876, 342744, 416603 ], [ 766848, 906704, 1102091 ],
  [ 1309817, 1548698, 1882429 ] ]
gap> RankMat(mat);
3
gap> DeterminantMat(mat);
-4
```

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

```
gap> 3*matˆ2-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’’ calculation
(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[3]

```
gap> im:=Integers mod 7;
GF(7)
gap> One(im);
Z(7)^0
gap> a:=5*One(im);
Z(7)^5
gap> b:=3*One(im);
Z(7)
gap> a+b;
Z(7)^0
gap> Int(a+b);
1
```

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 ] ]
```

[3] **Why?** Every nonzero element is invertible, and there is a better way to represent elements internally – in effect, GAP will avoid having to do division-with-remainder when doing arithmetic. We will learn more about this later in the course.