Mathematics 400c **Homework (due Jan. 28)** A. Hulpke

1) Show, that if there are integes r and s, such that

$$a = r^2 - s^2, \quad b = 2rs, \quad c = r^2 + s^2$$

then (a,b,c) is a pythagorean triple, i.e. $a^2 + b^2 = c^2$.

2) Prove, that the product of two consecutive integers is always divisible by 2, and that the product of three consecutive integers is always divisible by 3.

3) Determine (without using a computer) the number of zeroes at the end of 50!. ($50! = 1 \cdot 2 \cdot \cdots \cdot 50$.)

4) Show that when n is odd, $n^2 - 1$ is a multiple of 8.

5) For an integer n define the "sum of divisors" function as

$$\sigma(n) = \sum_{d|n} d$$

(For example $\sigma(8) = 1+2+4+8 = 15$, $\sigma(6) = 1+2+3+6 = 12$.)
A number n is called perfect if $\sigma(n) = 2n$.)
a) Compute $\sigma(p^n)$ for a prime p.
b) Suppose that $n = 2^{p-1}(2^p - 1)$ with p and $2^p - 1$ prime. Show that n is perfect (i.e. $\sigma(n) = 2n$).

6) Get accustomed with the basic functionality of a computer algebra system – for example GAP as listed below.

---

1 Computer Use

You are permitted to use computers/calculators for calculations, as long as this does not render a problem irrelevant. (As a rule of thumb, that means that calculations that are "new" should be performed by hand, calculations which we did already a while ago may be done mechanically.) For example if the aim of a problem is to calculate an (extended) gcd, you may do division with remainder with a calculator. If the problem is to solve a system of congruences you may do extended gcd calculations, if the problem is the factorization of a large number you may use the computer to solve congruences and so on.

An introduction to GAP

You start GAP by calling `gap` or `gap4` under Unix or by clicking the `gap.bat` icon under Windows.
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.

GAP knows integers of arbitrary length and rational numbers:

```
gap> -3; 17 - 23;
-3
-6
gap> 2^200-1;
160693804425899027554196209234116260252220299378279283530137 5
gap> 123456/7891011+1;
2671489/2630337
```

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:=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 are useful for number theoretic 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> EvalF(76/23); # numeric approximation
"3.3043478260"
gap> IsPrime(6);   #primality test
false
gap> IsPrime(73);
true
gap> NextPrimeInt(73); # next bigger prime
79
gap> 17/2 > 9;
false
gap> 17/2 <= 9;
true
gap> RootInt(1000,2); # rounded root
31
```

By enclosing objects with square brackets, and separating them by commas, you can create a list.
Collections of numbers (or other 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(2^127-1);
[ 170141183460469231731687303715884105727 ]
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
```

You can use the online help to get documentation

```
gap> ?List
Help: Showing ‘Reference: List’
> List( <list> ) ...
```

(Use `?Line Editing` to get a list of edit commands.)
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 ]
```

For a more complex example, we test whether numbers that can be represented as the sum of two squares.

```
gap> max:=1000;
1000
gap> lim:=RootInt(max,2)+1; # rounded up root
32
gap> cands:=[0..lim]; # candidate numbers to be squared
[ 0 .. 32 ]
gap> ForAny(cands,x->ForAny(cands,y->x^2+y^2= 19));
false
gap> ForAny(cands,x->ForAny(cands,y->x^2+y^2= 20));
true
```

Here, we test whether there is any combination of squares that sums up to the number we want to test. (This is far from optimal. Can you think of a better test?)
We now can combine this with a `Filtered` command to test all numbers up to 1000 that have this property. (We look for those n, such that an x exists, and that an y exists such that $x^2+y^2=n$.)

```
gap> Filtered([1..max],
> n -> ForAny(cands,x->ForAny(cands,y->xˆ2+yˆ2=n)) );
[ 1, 2, 4, 5, 8, 9, 10, 13, 16, 17, 18, 20, 25, 26, 29, 32, 34, 36, 37, 40,
```