GAP for MATHS 328

Robin Christian, Arkadii Slinko

March 1, 2007

1. How to access GAP

You have access to GAP from the computers in the Basement Computing Lab (BCL). The room number is B91. This room is in the basement of the Computer Science building, the door to BCL is next to the door to the stairwell. One way of getting there is to go into the basement of the Maths & Physics building, go all the way down the long corridor with several classrooms in it, go through the door at the end of that corridor and turn left, then go straight ahead until you find yourself at the door to the lab.

In order to use GAP from a computer in BCL, you will need to re-boot, and then select 'Linux' when the boot menu appears. To re-boot a computer that is running Windows, click the 'Shut Down' button, and then select 'Restart' from the drop down list that appears. Once the computer boots into Linux you can log in, using your usual EC username and password.

Once you have logged in, there will be some menus in the top left hand corner of your screen. Left click the 'Applications' menu, and you will be able to find a web browser ('Internet → Firefox'), the net login application for external internet access ('Internet → NetLogin'), a unix terminal ('System Tools' → 'Terminal'), and a GAP launcher ('Debian Menu → Apps → Math → GAP'). Alternatively you may select ('System' → 'Terminal application (Console)') and then type 'gap' after the prompt.

You will not have access to your usual EC home directory. When you want to log out, you have to left click the 'Actions' menu and select 'Log Out'.

If you have any problems with any of this, please let your lecturer know as soon as possible. If you do not try to use GAP until one day before the assignment is due in and then you discover you have problems, you will not be given an extension!

2. Handing in assignment answers using GAP

When you have completed some work in GAP, you can copy your work by clicking and dragging to select all of the work, then left clicking the 'Edit' menu at the top of the terminal window and selecting 'Copy'. Open any text file and select 'Edit' → 'Paste', then save the file.

You will be able to hand in your answers either by printing them out and attaching them to the rest of your assignment, or by e-mailing them to Dominic Searles (dnsearles@gmail.com), who is the marker for this course in 2007, before 4pm on the assignment due date. Tutorial answers can be emailed to Arkadii (a.slinko@auckland.ac.nz). Your input commands will be required as well as the output.

There may be a few more specific instructions later about assignment answer submission. If so, these will be mentioned in class and an announcement will be made on Cecil.

3. Information about GAP

There is plenty of information about GAP available on-line. In particular, a reference manual and tutorial can be found at

http://www-gap.mcs.st-and.ac.uk/Doc/manuals.html

They are also available on Cecil.

4. The GAP interface

Once you have started GAP, you can start working straight away. If you type a simple command (for example, 'quit') followed by a semi-colon, GAP will evaluate your command immediately. If you press enter without entering a semi-colon, GAP will simply give you a new line to continue entering more input. This is useful if you want to write a more complicated command, perhaps a simple program. If you wanted your simple command to be evaluated, then simply enter a semi-colon on the new line and press enter again. Since GAP ignores whitespace, this will work just the same as if you had entered the semi-colon in the first place. A semi-colon will not always cause GAP to evaluate straight away, GAP is able to work out whether you have finished a complete set of instructions or are part of the way through entering a program.

Another way to interact with GAP, which is particularly useful for things you want to do more than once, is to prepare a collection of commands and programs in a text file. Then you can type the command 'Read("MyGAPprog.txt");' and GAP will evaluate all of the instructions in your text file. If your file is not in the same place that GAP was launched from, you will have to provide its relative path (for example, "../../GAPprogs/Example1.txt").

5. Programming in GAP: Variables, lists, sets and loops

You can declare a variable in GAP using the ':=' operator. For example, if you want a variable 'n' to equal 2000, you would enter 'n := 2000;', or if you want 'n' to be the product of 'p' and 'q' you would enter 'n := p*q;'. You can also declare lists using the ':=' operator, for example, 'zeros := [0,0,0];'. The command 'list:=[m..n];' defines the list of integers $m, m+1, m+2, \ldots, n$. A list may have several identical numbers in it. Lists have a length ('Length(listName)'), and their entries can be referenced individually

by typing 'listName[index]' (indices start from 1!). In GAP a list of primes ≤ 1000 is stored. It is called 'Primes'. This is very useful.

```
gap> Primes;
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79,
83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167,
173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263,
269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367,
373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463,
467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587,
593, 599, 601, 607, 613, 617, 619, 631, 641,643, 647, 653, 659, 661, 673, 677, 683,
691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811,
821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929,
937, 941, 947, 953, 967, 971, 977, 983, 991, 997 ]
```

We can find the prime in 100-th position and the position of 953 as follows:

```
gap> Length(Primes);
168
gap> Primes[100];
541
gap> Position(Primes,953);
162
```

Sets cannot contain multiple occurrences of elements and the order of elements does not matter. Basically GAP views sets as ordered lists without repetitions. The command Set converts a list into a set.

```
gap> list:=[2,5,8,3,5];
[ 2, 5, 8, 3, 5 ]
gap> Add(list,2);
gap> list;
[ 2, 5, 8, 3, 5, 2 ]
gap> set:=Set(list);
[ 2, 3, 5, 8 ]
gap> RemoveSet(set,2);
gap> set;
[ 3, 5, 8 ]
```

For loops and while loops exist in GAP. Both have the same format:

for (while) [condition] do [statements] od;

For example, the following for loop squares all of the entries in the list 'boringList', and places them in the same position in the list 'squaredList':

```
gap> boringList:=[2..13];
```

```
[ 2 .. 13 ]
gap> squaredList:=[1..Length(boringList)];
[ 1 .. 12 ]
gap> for i in [1..Length(boringList)] do
> squaredList[i]:=boringList[i]^2;
> od;
gap> squaredList;
[ 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169 ]
```

Here is the example of using a while loop. We want to square the first five numbers of the boringList.

```
gap> boringList:=[2..13];
[ 2 .. 13 ]
gap> i:=1;
1
gap> while i<6 do
> boringList[i]:=boringList[i]^2;
> i:=i+1;
> od;
gap> boringList;
[ 4, 9, 16, 25, 36, 7, 8, 9, 10, 11, 12, 13 ]
```

List may contain other lists. Analyse the following program that lists all pairs of twin primes not exceeding 1000. It also illustrates the use of 'if-then' command.

if [condition] then [statements] fi;

```
gap> twinpairs:=[];
[  ]
gap> numbers:=[1..167];
[ 1 .. 167 ]
gap> for i in numbers do
> if Primes[i]=Primes[i+1]-2 then
> Add(twinpairs,[Primes[i],Primes[i+1]]);
> fi;
> od;
gap> twinpairs;
[ [ 3, 5 ], [ 5, 7 ], [ 11, 13 ], [ 17, 19 ], [ 29, 31 ], [ 41, 43 ],
  [ 59, 61 ], [ 71, 73 ], [ 101, 103 ], [ 107, 109 ], [ 137, 139 ],
  [ 149, 151 ], [ 179, 181 ], [ 191, 193 ], [ 197, 199 ], [ 227, 229 ],
  [ 239, 241 ], [ 269, 271 ], [ 281, 283 ], [ 311, 313 ], [ 347, 349 ],
  [ 419, 421 ], [ 431, 433 ], [ 461, 463 ], [ 521, 523 ], [ 569, 571 ],
  [ 599, 601 ], [ 617, 619 ], [ 641, 643 ], [ 659, 661 ], [ 809, 811 ],
  [ 821, 823 ], [ 827, 829 ], [ 857, 859 ], [ 881, 883 ] ]
```

6. Useful GAP commands: Number Theory

Try the following:

DivisorsInt(123456789);
FactorsInt(123456789);
PrintFactorsInt(123456789);
PrimePowersInt(123456789);
IsPrime(123456789);
NextPrimeInt(123456789);
PrevPrimeInt(123456789);
123456789 mod 34567;
Gcdex(123456789,987654321);
ChineseRem([43,71,93],[7,8,9]);
GcdInt(12345, 6789).

7. Annotated exercises: Number Theory

The command

```
DivisorsInt(n);
```

can be used to find all of the divisors of an integer. For example,

Exercise 1 Find all divisors of 142857142.

The remainder and quotient of n divided by m are given by the commands

```
RemInt(n, m); QuoInt(n, m);
```

respectively. For example,

Exercise 2 Compute the remainder and quotient upon dividing $m = 383$ by $n = 9786354$.

GAP does not provide automatic conversion between bases. One way of doing base conversion is to use the p-adic numbers package, feel free to investigate this on your own. Another way is to write simple programs. For example, 120789 can be converted to binary as follows:

```
gap> n := 120789;
120789
gap> base := 2;
2
gap> rems := [];
[ ]
gap> pos := 1;
```

```
1
gap> while n > 0 do;
> rems[pos] := RemInt(n,base);
> n := QuoInt(n,base);
> pos := pos + 1;
> od;
gap> n;
0
gap> rems;
[ 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1 ]
```

That is, 120789 is 11101011111010101 in binary. If you are not sure why I read the list rems in the reverse order, you need to study the base conversion algorithm in your lecture notes. As for converting from another base into decimal, you should now be able to do:

Exercise 3 Write a simple program to convert 100011100001111100000 from binary to decimal.

The greatest common divisor of m and n is given by

```
GcdInt(m, n);
```

For example,

Exercise 4 Compute the gcd of $m = 17682346$ and $n = 90001234$.

To find a, b such that $am + bn = gcd(m, n)$, use the GAP command

```
Gcdex(m,n);
```

For example,

```
Gcdex(108,801);
```

returns

```
rec( gcd := 9, coeff1 := -37, coeff2 := 5,  coeff3 := 89, coeff4 := -12 )
```

where a =coeff1, b =coeff2 (what are coeff3 and coeff4?).

Exercise 5 Find a, b such that $am + bn = gcd(m, n)$ for the m and n given in Exercise 4.

To find the least common multiple of m and n, use the GAP command

```
LcmInt(m, n);
```

For example,

Exercise 6 Compute the gcd and lcm of 7093542 and 2468013.

The Euler totient function $\phi(n)$ is given by the command

```
Phi(n)
```

For example,

Exercise 7 Compute $\phi(2^{15}-1)$ and $\phi(2^{17}-1)$, then deduce whether or not $2^{15}-1$ and $2^{17}-1$ are prime (hint: what is $\phi(p)$ if p is prime?).

The next prime after n is given by the command

```
NextPrimeInt(n);
```

The nth prime (for $n < 168$) is given by the command

```
Primes[n];
```

If we want to know the nth prime for larger values of n, we will have to create our own list of primes instead of using the built-in one. For example, this creates a list of the first 5000 primes:

```
gap> biggerPrimes := [];
[  ]
gap> counter := 1;
1
gap> currentPrime := 2;
2
gap> while counter < 5000 do;
> biggerPrimes[counter] := currentPrime;
> counter := counter + 1;
> currentPrime := NextPrimeInt(currentPrime);
> od;
```

It should be easy to modify this by changing the loop condition to complete:

Exercise 8 Create a list of all primes less than 1000000.

The prime factorization of n is given by the command

```
FactorsInt(n);
```

For example,

Exercise 9 What are the prime factors of $2^{63}-1$?

The Chinese remainder theorem commands give us the minimal solution $N \geq 0$ of $N \equiv a_1 \mod n_1$, $N \equiv a_2 \mod n_2$, ..., $N \equiv a_k \mod n_k$.

The command for the Chinese remainder theorem is

```
ChineseRem([n1,n2,...,nk],[a1,a2,...,ak]);
```

For example,

```
ChineseRem([5,7],[1,2]);
```

returns 16.

Exercise 10 Solve $x \equiv 2 \mod 3$, $x \equiv 1 \mod 4$, $x \equiv 3 \mod 5$.

The multiplicative order of $a \mod m$ is given by

```
OrderMod(a, m);
```

For example,

Exercise 11 Find the order of 10 mod 77.

The command

```
PowerMod(r, e, m);
```

returns the eth power of r modulo m. For example,

Exercise 12 Verify this example of the RSA crypto-system: Let $p = 17$, $q = 19$, $n = pq = 323$ so $\phi(n) = \phi(323) = 288$. Let $e = 97$ (the public exponent), so that $d = 193$ (the private exponent). Let $m = 100$ be the message. The "cipher text" is $c = 168$ since $m^e = 100^{97} \equiv 168 \mod 323$.

The primitive root mod m is given by

```
PrimitiveRootMod(m);
```

and the discrete log of a to the base b modulo m is given by

```
LogMod(a, b, m);
```

For example,

Exercise 13 Find the discrete log of 11 to base 5 mod 97, if it exists.

Exercise 14 Is 197 a prime? Is 2 a primitive root mod 197? Find the discrete log of 91 to base 2 mod 197, if it exists (ans: 44).

The commands

```
RootInt(n, k); LogInt(n, b);
```

can be used to determine respectively the integer part of the kth (positive real) root of n and the logarithm of n to the base b, that is, $\lfloor \sqrt[k]{n} \rfloor$ and $\lfloor \log_b(n) \rfloor$.