

## Maple Lecture 8. Input/Output Formats and Files

In this lecture we see how we save the results computed by Maple to file in different formats and learn how to import data into a Maple session. One important novelty in Maple 10 is the *Document Mode*. This mode is prettier than the other *Worksheet Mode*. For this course we will mostly use the worksheet mode.

The material for this lecture is inspired on [1, Sections 4.1, 4.3, and 4.4].

### 8.1 Maple Input and Output

Unless you suppress the output with a colon, Maple at least confirms your command, by default in a pretty print format. With `lprint` we get a more basic output format (may be suitable for low level communication with other programs). Adjusting the `printlevel`, we make Maple more verbose.

```
[> p := a*x^2 + b*x + c;  
[> sols := solve(p,x);
```

In some situations, a linear print may be more convenient:

```
[> lprint(sols);
```

Linear printing of results may also be a way to compactify huge output produced by Maple. To have Maple print everything in this format, we can do the following:

```
[> interface(prettyprint=0);          # value 0 uses lprint  
[> p; sols;                          # see the linear print  
[> interface(prettyprint=3);        # restore to its default
```

More explicitly, we get the same with converting to strings:

```
[> convert(sols[1],string);          # convert 1st solution to string  
[> s1 := parse(%);                  # reverse the convert
```

The conversion of mathematical data to strings of characters is a useful format for a human readable interface between different computer algebra systems or other programs which permit symbolic input.

If you are curious to know about the inner workings of the solve command, then we can adjust the `printlevel`.

```
[> default_printlevel := printlevel; # to restore later  
[> printlevel := 5:  
[> solve(p,x);
```

Adjusting the `printlevel` is very useful for debugging your own Maple procedures.

```
[> printlevel := default_printlevel: # restore to default
```

### 8.2 Exporting Plots

Once we have a nice picture, we may want to include it in a paper, or send it separately to the printer.

```
[> plotsetup(ps);                   # all plots will become postscript files  
[> plotsetup(default);              # reset to the default
```

With the options of `plotsetup`, we can also set the height and the width of the pictures.

### 8.3 Reading and Writing Files

Although worksheets can store data and instructions permanently, we may want to use Maple procedures or important data in several worksheets. First we illustrate how we can save and retrieve the data from subsection 8.1 above. For this example, we save in the directory MCS320 on the C drive:

```
[> save p, sols, "C:\\MCS320\\myfile";
```

Make sure you know why the double slashes are needed!

```
[> p := 'p': sols := 'sols':          # clear the data to test retrieval
[> p; sols;                          # check if data is gone
[> read "C:\\MCS320\\myfile";        # retrieve data from file
[> p; sols;                          # see if retrieval was fine
```

An alternative to **save** is to redirect the output of Maple to a file:

```
[> writeto("C:\\MCS320\\myfile");
[> p;                                # writes p to the file
[> writeto(terminal);                # output back to screen
```

We can also append to a file with the command **appendto**. Notice that this is the Maple output as it appears on screen. Usually this format is only suitable for printing and not for further Maple processing.

Our second example involves a procedure:

```
[> sum_list := proc(l::list)
>   description 'returns sum of elements in the list l':
>   local i,s:                # local variables
>   s := 0:                  # initialization of result
>   for i from 1 to nops(l) do # run over all operands in l
>     s := s + l[i]:         # add the i-th entry in l
>   end do:
>   RETURN(s):              # explicit return
[> end proc;
```

Observe for a moment the long square bracket along the code of the procedure. This long square bracket indicates that the whole procedure is considered as one execution group. When we type in the procedure, we do not hit enter (or return), but use the **Insert -> Execution Group -> After Cursor** menu. After a few times, we use the short cut, hitting **j** while holding the **Ctrl** key. At the end of the procedure, we use the **Edit -> Split or Join -> Join Execution Groups (or F4)** to create one execution group.

Symbolic computation pioneered “generic programming”: we can use the same piece of code for summing up integers, polynomials, or just symbols. Let us see how this works:

```
[> sum_list([1, 2, 3]);
[> sum_list([a, b, c]);
[> save sum_list, "C:\\MCS320\\sumproc";
[> sum_list := 'sum_list': sum_list; # clear to test retrieval
[> read "C:\\MCS320\\sumproc";
```

### 8.4 Import/export of data

Raw numerical data in the form of matrices of floating-point numbers can be brought in and out of Maple with the commands **readdata** and **writedata**.

Let us first create a matrix:

```
[> indfun := (i,j) -> 1/(i+j-1);    # index function
[> data := matrix(3,5,indfun);      # 3-by-5 Hilbert matrix
[> Digits := 20:                    # write a 20 digit approximation to file
[> writedata(terminal,data,float);  # first write to terminal as test
[> writedata("C:\\MCS320\\mydata",data,float);
[> data := 'data';                  # release variable to check retrieval
```

Matrices in `readdata` are considered to be organized columnwise. Besides the file, we give in the number of columns we wish to read :

```
[> data := readdata("C:\\MCS320\\mydata",5);
```

## 8.5 Low-Level I/O

We can manipulate files very much like if we were programming in C. A sample session is below:

```
[> ourfile := "C:\\MCS320\\testfile":
[> fopen(ourfile,WRITE,TEXT);
[> n := writeline(ourfile,'this is some text');
[> fclose(ourfile);
[> fopen(ourfile,READ,TEXT);
[> readline(ourfile);
[> fclose(ourfile);
[> fopen(ourfile,APPEND,TEXT);
[> fprintf(ourfile,"number of characters on previous line :%d\n",n);
[> fclose(ourfile);
```

## 8.6 Assignments

- Learn about the interface variable **echo** (see the help pages).  
What is it good for? Give a good illustration about its use.
- Explain the difference between `prettyprint` equal to 2 and 3.  
Give an example of what you can do with `prettyprint` at 3, but not at 2.
- Consider  $p = x^3 - 9$ . Set `printlevel` to 20.
  - Execute

```
[> int(p,x);
[> int(p,x);          # yes, we do it again
```

Why did you see less output the second time?
  - Type `diff(p,x)`; and explain what you see. What is the difference with `int(p,x)`? Explain.
- By default, the output of a plot comes in the worksheet. Change the **plotsetup** so that the plot appears in a new window. Give an example to illustrate.
- Type `m := matrix([[1,2],[3,4],[5,6]])`; to create the matrix `m`.  
Give the Maple commands to write `m` to file, and to read the first column of `m` from file, assigning this first column to `c1`.

## References

- [1] A. Heck. *Introduction to Maple*. Springer-Verlag, third edition, 2003.