

Computational Astrophysics

Writing files & Output formatting

1 Writing Files

To print out data to the screen, you will already be familiar with the `print` statement shown in the following examples:

```
print *,'Hello World!'
print *,variable
print *,'X=',x,' Y=',y
```

However in some cases you might want to write to a file rather than print to the screen. To do this, you will need three commands: `open`, `write` and `close`. These commands are used as follows:

- `open(unit=xx,file='filename')` is used to open a file. If the file does not exist, it is created. Replace `xx` by any number greater than 6 - this is called the *unit number*, and will be used to refer to the file whenever you want to write to or close it. Also, replace `filename` with whatever you wish to call the file.
- `write(xx,*)` is used to write to a file. Replace `xx` by the unit number. After this write anything you would have used in a `print` statement, for example:

```
write(xx,*) 'Hello World!'
write(xx,*) variable
write(xx,*) 'X=',x,' Y=',y
```

- `close(unit=xx)` is used to close the file. Once again, replace `xx` by the unit number. Do not forget to close any file which you have opened, as you may encounter problems otherwise.

As noted above, if the file opened does not exist, it will be created. Also, if you open an existing file, and start writing to it, it will be deleted and the program will write it from scratch.

2 Output formatting

In the `print` and `write` commands, you will have used the `*` symbol, which means that you leave it to the computer to decide how to display the numbers, for example how many significant digits to show, or whether to use scientific notation (e.g. `1.343E+03`) or normal floating point notation (e.g. `1312.312`).

However, it is possible to customize this rather than leaving it up to the computer. To do this, you can use an output format, which consists of a series of formatting instructions for each variable, separated by commas, and enclosed in brackets and inverted commas: `'(xx,xx,xx,xx,xx)'` For example `print '(E9.3)',variable` will print out a number of the form `x.xxxE+xx`.

To format numbers, you can use the following formatting instructions, where `x` is the total number of symbols used to display a variable and `y` is the number of numbers after the decimal place. An underscore means a space:

<code>Ix</code>	Integer	e.g. <code>I5 => __353</code>
<code>Fx.y</code>	Standard floating point	e.g. <code>F10.4 => ____3.4410</code>
<code>Ex.y</code>	Scientific form (with leading 0)	e.g. <code>E10.4 => 0.3441E+01</code>
<code>ESx.y</code>	Scientific form	e.g. <code>ES10.4 => 3.4410E+00</code>
<code>Ax</code>	Character	e.g. <code>A10 => STRING____</code>
<code>xX</code>	Spaces	e.g. <code>4X => ____</code>

You can then combine the individual formats inside the brackets. You can also include strings *inside* the formatting statements. The following example demonstrates this:

```
program formatting
  implicit none
  integer :: i
  i=124214
  print '(="i=",I8)',i
end program formatting
```

This produces the output:

```
i=__124214
(_=space)
```

Note that each format must be associated with a variable (except the `X` format for spaces, and if you use a string inside a formatting statement). For example `print '(ES10.4)',x,y` is not valid because there is only one formatting element and two variables.

To use output formatting when writing to a file, simply replace the `*` by the format, e.g.

```
write(13,'("i=",I8)') i instead of write(13,*) i
```

Output formatting can be very useful for example if you want to print a table on the screen, so that you can be sure that each column will always be the same width.