Worksheet 6. Input and Output

Most programs (except those that run other programs) contain input or output. Both fortran and matlab can read and write binary files, but we will stick to ascii.

It is worth noting that input and output are very time consuming. While outputting lots of information about the state of your variables while debugging or refining a program can be very useful, you can speed it up considerably by switching off all but essential output.

Fortran Input and Output

Fortran accepts input from a file or from the keyboard, and writes output either to the screen or to a file.

Fortran formatting codes Fortran can produce formatted output in which the way numbers and strings are displayed on the screen can be precisely controlled.

Number conversion To produce output or read input the program must be told how to convert between its internal representation of numbers (and characters) and the human readable forms shown on screen. This is specified through an *edit descriptor* in a *format specification*. E.g. to print an integer in a field 10 characters wide we would use the edit descriptor `I10` (I for integer, 10 for 10 characters wide). The format specification takes the form `(i10, f10.3,a10)` and can be placed within the read/write/print statement in quotes or in a separate numbered format statement. I.e. either

```
   print '(i10,f10.3,a10)', a,b,c
```

or

```
   print 10,  a,b,c
10 format(i10,f10.3,a10)
```

The second form can be useful where you have many print or write statements requiring the same format code.

Format Definition A format specification may be given in one of three ways. (i) By reference to a numbered format statement. E.g.

```
    print 100, q
100 format(f10.3)
```

(ii) As a character expression with the format specification in brackets. Note that the character expression can be a contained in a fortran variable and even constructed within the program. E.g.

```
character(len=*), parameter :: form='(f10.3)'
print form, q
```

or

```
character(4) :: c1
character(3) :: c2
c1='(f10'; c2='.3)'
print c1//c2, q     ! // concatentates (joins) strings
```

(iii) as an asterisk: `*` this allows unformatted input or output.

Units Input and output is directed to or from a unit specified either implicitly or explicitly. Units can either be an integer or an expression that evaluates to an integer

```
    nunit=4; i=2; j=-4
    read(4,'(f10.3)') q
    read(nunit,'(f10.3)') q
    read(4*i+j,100) q
100 format(f10.4)
```

Formatted Input The `read` statement is designed for reading input from files or the keyboard. Without a unit it reads from standard input either the keyboard or a stream piped in (don't worry if you don't know what this means) and takes the form

```
read fmt [,list]
```

where `fmt` is a formatting code e.g. `'(F18.6)'` and `list` is a list of variables. `list` is optional to allow the read statement to skip to the next line of input.

The read statement with a unit takes the form

```
read([unit=]u, [fmt=]fmt) [list]
```

where `u` is an integer referring to a file which has been opened and assigned a unit by the `open` command.

- Write a fortran program using all the lines of code above. Don't forget to use the `IMPLICIT NONE` statement.

Matlab Input and Output

User Input The `input` function reads user input

```
octave:4> x=input('Starting point: ');
Starting point: 23
octave:5> disp(x)
 23

octave:6> x=input('Starting point: ');
Starting point: [2 3 4]
octave:7> x
x =
   2   3   4
```

If an additional `'s'` is appended to the arguments to `input` the input is interpreted as a string. A (very) primitive function plotting program can be written in matlab with

```
while true
  f=input('function to plot: ','s')
  fplot(f,[-10 10])
endwhile
```

(Terminate the infinite loop with `CTRL-C`.)

Output to the screen The simplest way to display a matlab variable is with the disp command.

```
octave:11> disp('Here is a 6x6 magic square'), magic(6)
Here is a 6x6 magic square
ans =
   35    1    6   26   19   24
    3   32    7   21   23   25
   31    9    2   22   27   20
    8   28   33   17   10   15
   30    5   34   12   14   16
    4   36   29   13   18   11
```

Better control of format can be achieved using `fprintf` (matlab, along with many other programs written in C, use C syntax for input and output)

```
octave:17> fprintf('%6.3f\n',pi)
 3.142
```

`f` stands for fixed point notation, as with fortran `e` denotes exponential format. Unlike fortran strings are formatted with `%s`.

The function `sprintf` has similar syntax to `fprintf` but sends its output to a string.

```
octave:26> a=sprintf("pi to 5 decimal places is %7.5f",pi);
octave:27> disp(a)
pi to 5 decimal places is 3.14159
```

File Input and Output Firstly a file must be opened with the `fopen` command. It takes two arguments, the first is a string containing the filename, the second a string code explaining how the file is to be used: `'w'` means the file is to be written to, `'r'` means the file is to be read from. The output from `fopen` is a file identifier which the `fprintf` function uses.

```
A=[30 40 60 70];
fid=fopen('conversion.dat','w');
fprintf(fid,'%g miles per hour = %g kilometres per hour\n',[A; 8*A/5]);
fclose(fid);
```

When there are more variables to be typeset than there are format codes `fprintf` recycles the old format codes. Thus the file `conversion.dat` contains

```
30 miles per hour = 48 kilometres per hour
40 miles per hour = 64 kilometres per hour
60 miles per hour = 96 kilometres per hour
70 miles per hour = 112 kilometres per hour
```

The file can be read with the `fscanf` function

```
octave:31> fid=fopen('conversion.dat','r');
octave:32> X=fscanf(fid,'%g miles per hour = %g kilometres per hour');
octave:33> disp(X)
    30
    48
    40
    64
    60
    96
    70
   112
octave:35> X=reshape(X,2,4)'
X =
```

```
    30    48
    40    64
    60    96
    70   112
```

Alternatively you can specify the shape of the matrix

```
X=fscanf(fid,'%g miles per hour = %g kilometres per hour',[2,inf])';
X =
    30    48
    40    64
    60    96
    70   112
```

Also worth looking into is the `textscan` function.

- Write a matlab m-file that reads in 5 (x,y) pairs and plots a graph
- Modify the m-file so that it reads the data from a file
- Write a function that takes the name of the data file and the plotting style as arguments e.g. `mygraph('points.dat','r+')`
- Modify the function so that it plots a function of the y data rather than the data itself (e.g. $\sin y$)