Input / Output

So far we have omitted issue of reading in data and producing output from the program. We have been using function `print` without defining it as it is straightforward and easy to use. However, `print` function can only write to the standard output (screen by default).

Fortran includes a pair of more advanced functions `read` and `write` for reading and writing data to either standard output of a file. Basic usage of both of the function (to use standard streams) is simple and in comparison to `print` the usage of `write` looks like this

```
print *, 'Few variables : ', var1, var2, var3
write(*,*) 'Few variables : ', var1, var2, var3
```

both lines all equivalent. The `*` in `print` function defines the default format of the output. The same role has the second `*` in the `write` function. The first `*` in the `write(*,*)` defines the output e.g. screen or file, `*` is default output to the screen.

Similarly to `write` , reading data from the keyboard (default and standard input) may look like this

```
read(*,*) var1, var2, var3
```

which will wait until 3 variables are given from the keyboard followed by `Enter` key. In analogy to `write` the first `*` defines the source and the second the format.

The general form of the I/O statements is

```
read( unit#, format, options) list-of-variables
write( unit#, format, options) list-of-variables
```

list of variables must be separated by commas. `options` are optional, however the statement must include unit# and format at least as default with `(*,*)` . `unit#` is an integer which is unique for every output and input stream (file, screen or keyboard). Standard input (keyboard) and standard output (screen) have default unit numbers `5` and `6` .

Unit	Definition
5	Standard input (keyboard)
6	Standard output (screen)

The unit number is predefined for standard output and input. If file is required as an input the unit number is assigned to a given file in the file opening function `open`. The `open` function has general form

```
open( UNIT=number, FILE='FileName', options ... )
```

After opening a file `FileName` and assigning it the `UNIT` number, `read` and `write` functions may be used with that number to reference the file.

Every `read` command reads a one line and proceeds to the new line, every `write` command writes one new line and puts end-of-line character.

Example

```
program file_io

  implicit none

  integer :: i, max_i, io_unit
  character(len=20) :: file_name

  io_unit = 100
  file_name = "TestFile.txt"
  max_i = 10

  open(unit=io_unit, file=file_name, status="new")

  do i=1, max_i
    write(io_unit,*) i, 2*i
  end do

  close(io_unit)

end program file_io
```

which creates file `TestFile.txt` with the following content

```
           1           2
           2           4
           3           6
           4           8
           5          10
           6          12
           7          14
           8          16
           9          18
          10          20
```

We have used function `close(unit#)` to close the file.

Format

There are two ways to define the format of the output

1. Using `format` line, for example

   ```
   write(*,10) 'Result is : ', value
   10 format(a,f6.2)
   ```

2. With an inline formatting string

   ```
   write(*,'(a,f6.2)') 'Result is : ', value
   ```

The format descriptor (`a` and `f6.2` above) depends on the data type.

Type	Descriptior	Info
`integer`	`nIw`	`n` integer variables of width `w`
`real`	`nFw.d`	`n` real variables in decimal notation of `w` width with `d` decimal places
`real`	`nEw.d`	`n` real variables in scientific notation of `w` width with `d` decimal places
`character`	`nAw`	`n` character variables, each of `w` width
	`x...x`	String of characters - string to print defined in format line (`x` is any character here)
	`nX`	Horizontal spacing (`n` spaces)
	`/`	Vertical space
	`Tc`	Tab, `c` is the column number

Example

```
program format_example

  use iso_fortran_env

  integer none

  integer :: int1, int2, int3
  real :: var1, var2, var3
  character(len=10) :: string1, string2

  string1 = 'String1'
  string2 = 'String2'

  int1 = 123
  int2 = 1234
  int3 = 12345

  var1 = 3.1415
  var2 = 123.456789
  var3 = 0.0345678

  write(OUTPUT_UNIT, '(2a10)') string1, string2
  write(OUTPUT_UNIT, '(2a8,f16.4,2i10)') string1, string2, var1, int1, int2

  write(OUTPUT_UNIT, '(a8,f6.4,i5)') string1, var1, int1
  write(OUTPUT_UNIT, '(2e12.4)') var1, var2

  write(OUTPUT_UNIT, '(4(a7,3x))') string1, string2, string1, string2

end program format_example
```

which prints

```
String1   String2
String1 String2           3.1415       123      1234
String1 3.1415  123
  0.3141E+01  0.1235E+03
String1   String2   String1   String2
```

`iso_fortran_env` module

The `iso_fortran_module` defines the following variable for defining the input/output units

Variable	Description
`ERROR_UNIT`	Unit for error reporting
`INPUT_UNIT`	Unit for input
`IOSTAT_END`	Value assigned to `IOSTAT=` if end-of-file occured
`IOSTAT_EOR`	Value assigned to `IOSTAT=` if end-of-record occured
`OUTPUT_UNIT`	Unit for output

File openning

As we have mentioned above the general structure of the `open` function is

```
open( UNIT=number, FILE='FileName', options ... )
```

with `options` defining e.g. type of the file access, type of file. The most common parameters of the `open` functions are

Option	Values	Description
`unit`	`integer` variable	Unique number to identify the file
`file`	`string` variable	Name of the file to open
`status`	`old`, `new`, `replace`, `scratch`, unknown	Character of the file
`err`	`integer` variable	label to `goto` if an error occurs
`iostat`	returns	0 if no error, different in case of an error (compiler specific)
`form`	`formatted or unformatted`	with formating or binary
`access`	`sequential or direct`	free access or record based
`action`	`read`, `write`, `readwrite`	mode of operation

For example

To create new formatted file `File.txt`,

```
open( unit=100, file='File.txt', status='new', form='formatted',
access='sequential')
```

To open existing file for reading only

```
open( unit=101, file='TextFile.dat', status='old', action=`read`)
```

To create a temporary file (created at `open` and removed at `close`) in binary format

```
open( unit=102, file='matrix.dat', status='scratch', form='unformatted',
 access='sequential')
```

Example

```
program file_format

  implicit none

  integer :: io_unit = 100
  integer :: i, max_i = 10

  character(len=20) :: file_name

  file_name = 'File1.txt'

  ! Text file
  open( unit=io_unit, file=file_name, status='new', form='formatted', &
    access='sequential')

  do i=1, max_i
    write(io_unit,'(i4)') i
  end do

  close( io_unit )

  ! Binary file
  file_name = 'File2.dat'

  open( unit=io_unit, file=file_name, status='new', form='unformatted', &
    access='sequential')

  do i=1, max_i
    write(io_unit) i
  end do

  close( io_unit )

end program file_format
```

which creates two files with the same information. `File1.txt` is formatted, `File2.dat` is a binary file.

Writing to a string

One can create strings dynamically in the program with the Fortran option to write into a variable. Unit number is replaced by a character variable that holds the string we would like to create, for example

```
program string_io

  use iso_fortran_env

  implicit none

  character(len=50) :: string
  integer :: num

  num = 1

  write(string,'("matrix_",i3.3,".data")') num

  write(OUTPUT_UNIT, '(a)') string

end program string_io
```

gives

```
matrix_001.data
```

This feature of Fortran is very useful for 1) conversion of numbers to strings, 2) Dynamically creating strings e.g. commands or file names.

I/O miscellaneous

1. When a file is created in Fortran using `write` commands, the `read` commands must have the same format and appear in the same order
2. `read` and `write` operate in line at the time mode i.e. one `write` will create new line and `read` will read to the end of the line (even if data is not used)
3. Binary i.e. `form=unformatted` files are not directly transferable to other programming language as they include formating bits not only data. It is possible to read Fortran binary files in C but some data needs to be discarded.
4. Binary format `form=unformatted` should be used if full precision of the number is required in the file. Binary files are not 64bit - 32bit transferable.