Input/output to files

Arjen Markus

Deltares

June 6, 2018

I/O statements

Input and output to files is done by the following statements:

- `open( ... )` – connect the program to a file
- `close( ... )` – disconnect the file
- `read( ... )` – read data from a file (or a string)
- `write( ... )` – write data to a file (or a string)
- `inquire( ... )` – get information about a file
- `flush( ... )` – flush all data in the internal buffers to the file

Types of files

The Fortran standard distinguishes a number of file types.
The contents can be:

- Readable by humans – formatted files
- Binary – raw bytes, difficult to read, but quite efficient in size and speed

Note: "unformatted" files are binary files with a record structure.

Types of files (2)

Another difference is how files are accessed:

- Sequentially – data can only be read from start to finish
- "Directly" – you can read data in any order
- As a "stream" – no inherent structure

```
open( 10, file = 'myfile.out', access = 'stream' )
write( 10 ) array
```

You can use "pos =" to control the reading/writing:

```
read( 11, pos = 33 ) x          ! Read from an arbitrary position

inquire( 11, pos = position )   ! Get the current position
```

Information about files

The `inquire` statement returns information about files:

```
logical :: exist
integer :: size

inquire( file = 'myfile.out', exist = exist, size = size )
if ( exist ) then
    write(*,*) 'The file "myfile.out" exists'
    write(*,*) 'Size in bytes: ', size
else
    write(*,*) 'The file "myfile.out" does not exist'
endif
```

Error handling

Use keywords to check if an open/read/write succeeds:

```
character(len=100) :: string

open( 10, file = 'myfile.inp', status = 'old', &
    iostat = ierr)

if ( ierr /= 0 ) then
    write(*,*) 'Error opening the file "myfile.out"'
endif

read( 10, *, iostat = ierr, iomsg = string ) array

if ( ierr /= 0 ) then
    write(*,*) 'Error reading the file "myfile.out"'
    write(*,*) trim(string)
endif
```

Advancing and non-advancing reading/writing

Control the positioning after a read/write statement:

```
use iso_fortran_env, only: output_unit

write(*, '(a)', advance = 'no') 'Enter a number: '
flush( output_unit )

read(*,*) x

write(*,*) 'Square: ', x**2
```

Note the use of `flush` and `output_unit`.

Formats

List-directed ("*") reads and writes are quite flexible, but offer little control.
You can also explicit formats – just a few examples:

- `I10`: read/print an integer with ten positions
- `I10.10`: integer with ten positions and leading zeroes – 0000000123
- `I0`: integer, but use as many positions as needed

```
n = 1
write( string, '(a,i0,a)' ) 'file', n, '.out'

==> file1.out
```

- `G0`: integer and reals, use effective format, width and precision

```
write(*,'(3(g0,1x))') 1.23, 1.23e4, 1.23e-4

==> 1.230000 12300.00 .1230000E-03
```

Other features

There are a lot of details concerning input/output.
Here are a few other details:

- Automatic assignment of a file unit number:

```
open( newunit = lun, file = 'myfile.out' )
```

- Flushing output files:

```
flush( lun )
```

- Asynchronous read/write:

```
open( 20, file = "myfile.inp",asynchronous="yes" )

read( 20, '(I4)', asynchronous="yes" ) (array(j), j=1,10000)

... continue processing - without the array

!
! We need the array now, so make sure it has been read
!
wait( 20, iostat = ios )
```