Introduction to Modern Fortran

Advanced I/O and Files

Nick Maclaren

nmm1@cam.ac.uk

March 2014

Summary

This will describe some advanced I/O features
Some are useful but only in Fortran 2003
Some are esoteric or tricky to use

- The points here are quite important

Excluded only on the grounds of time

There is a lot more in this area

- Please ask if you need any help

Partial Records in Sequential I/O

Reading only part of a record is supported
Any unread data in the record are skipped
The next READ uses the next record

Fortran 90 allows you to change that

- But ONLY for formatted, external I/O

Specify ADVANCE='no' in the READ or WRITE
This is called non–advancing I/O

Non-Advancing Output

You can build up a record in sections

```
WRITE (*, ’(a)’, ADVANCE=’no’) ’value = ’
IF (value < 0.0) THEN
    WRITE (*, ’("None")’) value
ELSE
    WRITE (*, ’(F5.2)’) value
END IF
```

This is, regrettably, the only portable use

Use for Prompting

```
WRITE (*, ’(a)’, ADVANCE=’no’) ’Type a number: ’
READ (*, *) value
```

That will usually work, but may not

The text may not be written out immediately
Even using FLUSH may not force that

Too many prompts may exceed the record length

Non-Advancing Input

You can decode a record in sections
Just like for output, if you know the format

Reading unknown length records is possible
Here are two recipes that are safe and reliable

Unfortunately, Fortran 90 and Fortran 2003 differ

Recipe (1) - Fortran 2003

```
USE, INTRINSIC :: ISO_FORTRAN_ENV
CHARACTER, DIMENSION(4096) :: buffer
INTEGER :: status, count
READ (1, ’(4096a)’, ADVANCE=’no’, SIZE=count, &
        IOSTAT=status) buffer
```

If IOSTAT is IOSTAT_EOR, the record is short
If IOSTAT is IOSTAT_END, we are at end−of−file

SIZE returns the number of characters read

Recipe (2) - Fortran 90

```
CHARACTER, DIMENSION(4096) :: buffer
INTEGER :: count
READ (1, ’(4096a)’, ADVANCE=’no’, SIZE=count, &
        EOR=10, EOF=20) buffer
```

The EOR branch is taken if the record is short
The following happens whether or not it is

SIZE returns the number of characters read

General Free-Format Input

- Can read in whole lines, as described above
And then decode using CHARACTER operations
You can also use internal files for conversion

- Can use some other language for conversion
I use Python, but Perl is fine, too
Use it to convert to a Fortran–friendly format

- You can call C to do the conversion
That isn’t always as easy as people think it is

List-Directed I/O (1)

This course has massively over–simplified
All you need to know for simple test programs
It is used mainly for diagnostics etc.

Here are a few of its extra features

Separation is by comma, spaces or both
That is why comma needs to be quoted
Theoretically, that can happen on output, too

List-Directed I/O (2)

You may use repeat counts on values
100*1.23 is a hundred repetitions of 1.23

That is why asterisk needs to be quoted
Theoretically, that can happen on output, too

There may be null values in input
“1.23 , , 4.56” is 1.23 , null value, 1.234.56
“100*   ” is a hundred null values

Null values suppress update of the variable

List-Directed I/O (3)

As described, slashes (/) terminates the call
That is why slash needs to be quoted

Before using it in complicated, important code:

- Read the specification, to avoid "gotchas"
- Work out exactly what you want to do with it

Formatted Input for REALs

m in Fn.m etc. is an implied decimal point
It is used only if you don't provide one
The k in En.mEk is completely ignored

And there are more historical oddities
Here is an extended set of rules

- Use a precision of zero (e.g. F8.0)
- Always include a decimal point in the number
- Don't use the P or BZ descriptors for input
- Don't set BLANK='zero' in OPEN or READ

The Sordid Details

If you want to know, read the actual standard
You won't believe me if I tell you!

And don't trust any books on this matter
They all over–simplify it like crazy

In any case, I doubt that any of you care
Follow the above rules and you don't need to

Choice of Unit Number

Preconnected units are open at program start
Includes at least ones referred to by UNIT=*

- OPEN on them will close the old connection

Can check for an open unit using INQUIRE

Fortran 2003 has a way of getting their numbers
Has names in the ISO_FORTRAN_ENV module

Critical only for significant, portable programs

INQUIRE By File (1)

You can check if a file exists or is open

```
LOGICAL :: here
INQUIRE (FILE='name', EXIST=here)
INQUIRE (FILE='name', OPENED=here)
```

- These answers may not mean what you expect

E.g. a new, output file may be open but not exist

- Name matching may be textual or by identity

Watch out when using ln or ln –s

INQUIRE By File (2)

Can query SIZE, READ, READWRITE, WRITE
Don’t bet on it – not all compilers support them sanely
Some others, too, but not under Unix–like systems

Most other queries are handled like inquire by unit
Subject to matching the file name correctly
If not connected always return UNKNOWN
Not exactly the most useful behaviour!

However, at least they DO say UNKNOWN
And don’t simply return plausible nonsense

INQUIRE By Unit (1)

Inquire by unit most usefully does two things:
Checks if the unit is currently connected
Returns the record length of an open file

```
LOGICAL :: connected
INQUIRE (UNIT=number, OPENED=connected)

INTEGER :: length
INQUIRE (UNIT=number, RECL=length)
```

You can ask about both together, of course

INQUIRE By Unit (2)

There are other potentially useful specifiers
Not all of them make much sense under POSIX

You can get all of the specifiers used for OPEN
Could be useful when writing generic libraries

SIZE gives the size of the file, probably in bytes
This is only in Fortran 2003, and unreliable
Again, nothing to do with Fortran, as such

See the references for details on them

Unformatted I/O

Using pipes or sockets is unreliable
The reasons are complicated and historical

So is unformatted I/O of derived types
The same applies in C++, for very similar reasons

- Ask for advice if you need to do these

Namelist

Namelist is a historical oddity, new in Fortran 90
This sounds impossible, but I assure you is true

- Not recommended, but not deprecated, either

STREAM Files

Fortran 2003 has introduced STREAM files
These are for interchange with C–like files
They provide all portable features of C

- They allow positioning, like C text files

I advise avoiding the POS= specifier
It's full of gotchas in both C and Fortran

I/O of Derived Types

The DT descriptor has been mentioned

- Unfortunately, it's often not implemented

You can do almost anything you need to
But this course cannot cover everything

Asynchronous I/O

Mainframes proved that it is the right approach
Fortran 2003 introduced it

- For complicated reasons, you should avoid it

- This has nothing to do with Fortran

Don't use POSIX asynchronous I/O, either
And probably not Microsoft's . . .

BACKSPACE

Don't go there

It was provided for magnetic tape file support
In those days, could often read backwards, too

It's almost always a performance disaster, at best
And it very often doesn't actually work reliably

- Again, that is NOT specific to Fortran

It applies to using seek in C/C++, too
Never reposition on sequential files
Rewinding to the beginning is usually OK

Oddities of Connection

- Try to avoid these, as they are confusing

You will see them in some of the references

Files can be connected but not exist
Ones newly created by OPEN may be like that

Units can be connected when the program starts
Ask me if you want to know why and how

OPEN can be used on an existing connection
It modifies the connection properties

Other Topics

There are a lot more optional features
You must read Fortran's specifications for them

Fortran 2003 adds many slightly useful features
Most compilers don't support many of them yet
The above has described the most useful ones

And a few features should be avoided entirely

For more on this, look at the OldFortran course

Last Reminder

Be careful when using Fortran I/O features
They don’t always do what you expect

It is much cleaner than C/POSIX, but . . .

Fortran’s model is very unlike C/POSIX’s
Fortran’s terminology can be very odd

The underlying C/POSIX can show through
In addition to Fortran’s own oddities