

FORTRAN Binary I/O

There are two tricky steps in reading binary data in FORTRAN. First you must open the file with the proper mode, then you must correctly read and interpret the data values. There is no one correct way to do either of these steps. It often takes a fair bit of trial and error to get it right. It is therefore essential that you have test data to read with documented examples of known values.

Opening files

There are several keywords to get right when opening a file. Most important are FORM, ACCESS, and RECL. You almost always want FORM='UNFORMATTED', ACCESS='DIRECT'. Some FORTRANs have a FORM='SYSTEM' or FORM='BINARY' option. This is often the best way to go if it is available. With this mode the ACCESS and RECL keywords are omitted and the file is read as a continuous stream of bytes.

The RECL keyword specifies the record length. Standard FORTRAN record lengths are specified in 4 byte increments. If your record is not an integer multiple of 4 you will have to read 2 or 4 records at a time. Some FORTRAN compilers have a switch to allow record lengths to be specified in bytes. Some FORTRANs assume this by default. If you can't find out by reading the documentation you will have to experiment.

For files with mixed length records you will have to figure some common denominator between all the different record lengths. For example, if your file has a 300 byte header followed by an array with 304 values per row, each a single byte then your largest common factor is 4 and the record size is 1 (unless your record size is specified in bytes, then it's 4).

Reading and interpreting data

The most important things to get right in reading the data are - byte order, datum size and format, scaling factors, and matrix order.

Byte order is a confusing concept referring to the manner in which integer values are interpreted by the computer hardware. There are many different terms to describe particular byte orders (big-endian vs. little-endian, network vs. host, etc.) All I'll tell you here is that there are two camps, Dec and Intel are in one camp (little-endian) and pretty much everyone else is in the other camp.

One byte data never needs to be swapped. Two byte data can often be swapped by a system utility. On Unix use the dd command -

```
dd conv=swab if=input_file of=output_file
```

The datum size is simply how many bytes were used to store each value. Usually 1, 2, 4, or (rarely) 8. The format is one of: integer, unsigned integer, or floating point. The data are most often stored as fixed point integers, that is, scaled by some factor of ten. These are simply read as integers and then multiplied by the appropriate scale factor.

FORTRAN doesn't have a one byte integer storage class, so byte data must always be converted with the ICHAR function. ICHAR interprets the datum as a signed value.

The standard trick in FORTRAN for reading unsigned data is to read each datum value into a signed integer of the same size and then copy numbers into the next bigger size INTEGER or REAL. Most negative numbers are stored as twos-complement these days so the conversion consists of adding the appropriate offset. For byte data that would be 2^8 i.e. 256, for 2 byte data it's 2^{16} i.e. 65536, and for 4 bytes data it's 2^{32} (whatever that is). If you know that none of the data values are greater than half the largest possible value then no conversion is necessary. For example the largest possible unsigned byte value is 255. If you know that all of the data is less than or equal to 127 then no conversion (other than ICHAR) is needed.

Matrix order refers to the manner in which sequential information (the data in the file) is stored in a multidimensional array. In FORTRAN the leftmost subscript varies most rapidly. It's simplest to talk about two dimensional data. In this case the row subscript varies most rapidly and therefore the data is said to be "stored by column." Binary data is often stored by row, that is the column varies most often. To remedy this mismatch you can either transpose the data and store each matrix element in its proper place, or you could simply transpose your meaning of row and column and refer to elements by column and row.

In direct access I/O each time a READ statement is executed one full record is read. This means you cannot read a single value at a time. You must declare a buffer large enough to hold a whole record (or 2 or 4 if necessary). Repeatedly read data into that buffer and then extract individual values from the buffer.

Examples

Unsigned byte data in a two dimensional array. [readplds.f](#)

Unsigned byte data in a two dimensional array with a 300 byte header. [readgts.f](#)

Sample output:

```
00255   304   4481.79939.4345.00558.4154.0234.0 SSMI13  cn   001-9999
-9999   031-9999-9999 1997   001   00000250199701.N13                      ARCTIC
```

```
SSMI MONTHLY MEAN TOTAL ICE CONCENTRATION(DMSP F13 JAN)JANUARY
1997ARCT
IC      SSM/IONSSMIGRID CON Coast253Pole251Land254      05/19/1999
136192 values read
```

[199701.txt](#)

Scaled 2 byte data in two dimensional array. [readarr1.f](#)

Sample output:

```
column, row lat, lon
      721      721  29.89000      45.00000
        1        1  29.89000     -135.0000
        1      721  29.89000     -45.00000
      721        1  29.89000      135.0000
     361      361  90.00000      0.0000000E+00
```

Same data using FORM= ' SYSTEM' [readarr2.f](#)

Sample output:

```
row, column, lat, lon
      721      721  29.89000      45.00000
        1        1  29.89000     -135.0000
        1      721  29.89000     -45.00000
      721        1  29.89000      135.0000
     361      361  90.00000      0.0000000E+00
```

Notice that row and column are transposed in the two versions of readaari!

Additional Resources

[Fortran FAQ](#)
[comp.lang.fortran](#)

[Ken Knowles <knowlesk@nsidc.org>](#)

Last modified: Fri Dec 20 13:07:07 2002

\$Revision: 1.4 \$