## Lecture 5 Formats and Disk I/O

### 5. 0 Formats and data conversion

Always remember that all the data in the computer system is stored as binary numbers. These are unreadable for us, since we use either text or decimal numbers. The computer program must always convert decimal numbers into binary and vice-versa. In the Fortran language the FORMAT statement controls how this is done. Don't forget that there are different ways of storing data depending on the number of bytes for each number. REAL or INTEGER are standard data types, but we can also have COMPLEX and CHAR ( or CHARACTER ) data types. The CHAR data type is the only one we can read. All text files are stored as CHAR variables, with one byte for every character, including spaces, punctuation marks etc.

### 5.1 READ FORMAT STATEMENTS

```
      READ (*,100)  x,y,z

100   FORMAT ( 3F12.6)
```

The above program means "listen at the keyboard for three real numbers and store them in variables called x,y and z. The numbers will have a maximum of 12 characters, including sign and decimal point, and have a maximum of 6 decimal places". If the user does not type the decimal point we will get input conversion errors. Note 3 and 3.0 are not the same, why not?

The following are all illegal in terms of Fortran. While they may compile we can expect run time errors or just stupid answers. If you can't spot the mistakes, ask your tutor.

```
      READ (*,101 ) I,J,K
      READ (*,102) X
      READ (*,103) Z
101   FORMAT (3F12.6)
102   FORMT (I6)
103   FORMAT (F2.8)
```

#### 5.1.1 Review questions

What is the Fortran Format specifications for the following:-

-7890  +3.14159  -4.6  -4  -23456.7

#### 5.1.2 Trapping input errors

If the user types in an incorrect format we should get a run time error and the program will crash, i.e. terminate and go back to the operating system. We can use this to our advantage with an ERR statement.

```
      READ (*,1001, ERR=100) X
1001  FORMAT (I4)
```

## 5.2 FORMATTED WRITE STATEMENTS

Format statements allow the programmer to control the appearance and position of output produced by write statements. The basic structure of a format statement is shown in the example below.

```
      pi  = 4*atan(1.0)
      tpi = 2*pi
      npi = 9*pi
      write(*,*)' '
      write(*,100)
      write(*,110)npi,pi,tpi
      write(*,*)' '
      write(*,120)pi,tpi
 100  format(' ','I-Format',7x,'F-Format',7x,'E-Format')
 110  format(' ',2x,i4,10x,f6.4,8x,e8.2)
 120  format(' ','Repeat format example-',2f8.3)
      stop
      end
```

The above would result in the following being printed on the screen

```
 I-Format       F-Format       E-Format
    28           3.1416        0.63E+01

 Repeat format example-   3.142   6.283
```

Notes:

- The number that replaces the second asterisk in the write statement is used to identify which format statement is to be used.
- A formatted write statement can be used to simply print a heading; i.e., `I-Format`, etc.
- The first entry after the open bracket of the format statement is used to control line spacing. The quotes around a blank used in the above forces a new line. Other options exist but they will not be discussed here.
- Any characters between quotes will be written to the screen exactly as they appear in the format statement, including blanks.
- Horizontal spacing is achieved by the "`nx`" construct, where n is the number of spaces required.
- Note that commas are required between all entities.
- Numbers are always printer right justified.
- Format statements can be located anywhere in the program.

**Format types**: In the example above the three most common numeric formats are shown. Symbolically these are defined as:

| | |
|---|---|
| Integer | miw |
| Floating | mfw.d |
| Exponential | mew.d |

m = number of times the field is repeated (m omitted = no repeat)

w = field width – a positive integer defining the maximum spaces the whole number, including sign and decimal point, occupies. In the case of the e-format, it also includes a four character exponent string (e.g., e+01).

d = is a positive integer defining the number of decimal places.

## 5.3 FREE FORMAT

Fortran 95 makes life easier for us by using FREE FORMAT. In this case the computer will decide the best format conversion to use. Examples are:-

```
 READ (*,*) X,I,Y,L
WRITE (*,*) I,L,Y,X
```

The first example means " listen at the keyboard for four numbers, store first as real variable X, the second as an integer called I, the third as areal variable Y and the fourth as another integer called L. The second example means " write out two integer numbers followed by two real numbers , all on the same line.

## 5.4 READING FROM AND WRITING TO FILES

We can use Fortran to save our data on the disk, but we must decide what format to use to save the data in. Binary data is quicker for the computer, but we can't read it! As far as we are concerned it is better to save the data as a text file in decimal, which means all the numbers have to be converted from binary to decimal text under format control. Try the folowing programs and see what they do:-

```
      PROGRAM NEWDATAFILE
      OPEN (UNIT=14,FILE='fred.DAT',STATUS='NEW')
      WRITE (*,*)  ' TYPE IN A LIST OF REAL NUMBERS, ENDING WITH A LETTER'
100   READ (*,1000,ERR=101) X
      WRITE (14,1000) X
      GOTO 100
101   CLOSE(14)
      END
```

The program NEWDATAFILE should have created a new file called 'fred.dat' and since we used formatted I/O you should be able to read the file using a simple word processor such as NOTEBOOK. Here is another example. Try to get this program working once and see what happens when you try to run the program a second time.

```
      PROGRAM FLOPPY
      OPEN (UNIT=14,FILE='A:fred.DAT',STATUS='NEW')
      WRITE (*,*)  ' TYPE IN A LIST OF REAL NUMBERS, ENDING WITH A LETTER'
102   READ (*,1000,ERR=101) X
      WRITE (14,1000) X
      GOTO 100
103   CLOSE(14)
1000  FORMAT (F12.3)
      END
```

Before data can be read from or written to a file, the file must first be “connected” to the program. This is achieved using an `open` statement. An example of the simplest form of an open statement is given below. Other forms are available, but they will not be discussed here.

```
      open(14,file='test.dat')
      open(16,file='test.dmp')
      read(14,*)a,b,c
      read(14,*)i,j,k
      write(*,*)' '
      write(16,100)i,a
      write(*,*)' '
      write(16,100)j,b
      write(*,*)' '
      write(16,100)k,c
100   format(' ',i4,e14.3)
      close(14)
      close(16)
      stop
      end
```

Notes

- Open statements assign unit numbers (14 and 16, respectively, in the above) to files.
- The format of the open statement is the same for input and output files (at least for the simple cases shown above).
- The first number in the read or write statement corresponds to the number assigned to a particular file in the open statement.
- File names have to appear in single quotes.
- The extensions used are purely for convenience. Any permissible Fortran name, with or without an extension, is acceptable.
- Files are closed using the `close` statement. The close statement can appear anywhere in the program after the last use of the file.

## 5.5 FILES, RECORDS AND FIELDS

Files are divided into RECORDS and records are subdivided into FIELDS. There will be one record in the file for every time we have executed a WRITE statement to that file. Every time we perform a READ statement from the file we get one record back, and the file position counter moves on to the next record.

The following writes three records with one field in each record:-

```
          WRITE (14,1000) X
          WRITE (14,1000) Y
          WRITE (14,1000) Z
1000    FORMAT (F12.2)
```

The following writes one record with three fields:-

```
          WRITE (14,1001) X,Y,Z
1001    FORMAT (F12.2)
```

**5.6 MISTAKES IN FORTRAN FILE I/O**

Try to find the mistakes in the following program, and explain why it doesn't work. Remember when you open a data file in FORTRAN you must know the exact format used to write the file in the first place! This can cause problems – YOU HAVE BEEN WARNED.

```
              PROGRAM TESTIO
             DIMENSION Y(100)
!--------
!INPUT SOME DATA AND SAVE ON DISK
!--------
              OPEN (UNIT=14,FILE='fred.DAT',STATUS='NEW')
              WRITE (*,*)  ' TYPE IN A LIST OF REAL NUMBERS, ENDING WITH A LETTER'
    104       READ (*,1000,ERR=105) X
              WRITE (14,1000) X
              GOTO 104
    105       CLOSE(14)

!------------
! reopen file and process
!-----------
             OPEN (UNIT=1,FLE='FRED.DAT',STATUS='OLD')
             OPEN (UNIT=2,FLE='BILL.DAT',STATUS='NEW')
!-----------
!  open old file and get data into memory
!------------
            N=0
             DO 106  KK=1,100
            READ (1,1001,ERR=106) Y(KK)
             N=N+1
    106       CONTINUE
    !-----------
    !save data in a new file
    !---------
    107       DO 108  KK=1,N
              WRITE (2,1002) Y(KK)
    108     CONTINUE


             CLOSE (1)
             CLOSE(2)
            STOP
 1000        FORMAT (F12.3)
 1001        FORMAT (F12.2)
 1002        FORMAT (F12.1)
              END
```

**Keep a record of your input data, use the NOTEBOOK ( or WORD ) to open the data files and explain what has happened to the data.**